Université de Liège
Faculté des Sciences Appliquées
Département de Chimie Appliquée
Laboratoire d'Analyse et de
Synthèse des Systèmes Chimiques

Optimal synthesis of sensor networks

Carine Gerkens

Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences de l'Ingénieur
Mai 2009

To Marie, Emmanuel and Pierre

Cherchons comme cherchent
ceux qui doivent trouver
et trouvons comme trouvent
ceux qui doivent chercher encore.
Car il est écrit :
celui qui est arrivé au terme
ne fait que commencer.

Saint Augustin

Summary

To allow monitoring and control of chemical processes, a sensor network has to be installed. It must allow the estimation of all important variables of the process. However, all measurements are erroneous, it is not possible to measure every variable and some types of sensors are expensive.

Data reconciliation allows to correct the measurements, to estimate the values of unmeasured variables and to compute a posteriori uncertainties of all variables. However, a posteriori standard deviations are function of the number, the location and the precision of the measurement tools that are installed.

A general method to design the cheapest sensor network able to estimate all process key variables within a prescribed accuracy in the case of steady-state processes has been developed. That method uses a posteriori variances estimation method based on the analysis of the sensitivity matrix. The goal function of the optimization problem depends on the annualized cost of the sensor network and on the accuracies that can be reached for the key variables. The problem is solved by means of a genetic algorithm.

To reduce the computing time, two parallelization techniques using the message passing interface have been examined: the global parallelization and the distributed genetic algorithms. Both methods have been tested on several examples.

To extend the method to dynamic processes, a dynamic data reconciliation method allowing to estimate a posteriori variances was necessary. Kalman filtering approach and orthogonal collocation-based moving horizon method have been compared. A posteriori variances computing has been developed using a similar method than the one used for the steady-state case. The method has been reconciled on several small examples.

On the basis of the variances estimation an observability criterion has been defined for dynamic systems so that the sensor network design algorithm could be modified for the dynamic case.

Another problem that sensor networks have to allow to solve is process faults detection and localisation. The method has been adapted to generate sensor networks that allow to detect and locate process faults among a list of faults in the case of steady-state processes.

Thesis organisation
The thesis begins with an introductive chapter followed by a chapter devoted to the state of the art.

The first part of the thesis is devoted to the design of sensor networks for steady-state processes. It is organized this way:

- The third chapter is devoted to non linear steady-state data reconciliation. After a presentation of the necessity of data reconciliation, the problem is formulated. The constrained optimization problem is transformed into an unconstrained one using Lagrange method. One obtains the sensitivity matrix of the problem. By writing the necessary conditions for optimality, it allows the computation of the reconciled process variables and their a posteriori variances.

- In the fourth chapter, the optimization strategy used to solve the sensor network design problem is described. That problem being combinatorial and generally multimodal, it is solved by means of genetic algorithm.

- The five steps of the sensor network design algorithm are described in the fifth chapter.

- The sixth chapter is devoted to the algorithm parallelization. It begins with a description of some parallelization notions and routines. Then two parallelization techniques are described: global parallelization and distributed genetic algorithms.

- In the seventh chapter, four examples are presented: an ammonia synthesis loop, a combined cycle power plant, a ketene cracker and a naphta reformer. For all those processes, the sensor network design is carried out and the two parallelization techniques are compared. The parameters of the distributed genetic algorithms are studied for the ammonia synthesis loop.

- The design of sensor networks allowing the detection and the localisation of process faults occurring in steady-state processes is studied in chapter 8. The first part of the chapter consists of the description of the fault detection method that is used. Then the sensor network design algorithm is explained. Finally, the program is applied on two water networks.

- Chapter 9 is devoted to the conclusions of this first part.

In the second part of the thesis, dynamic data reconciliation and estimation of a posteriori variances are approached. It is divided into six chapters:

- In chapter 10, the dynamic data reconciliation problem is formulated.

- Chapter 11 is devoted to filtering methods. After a general introduction to those techniques, the extended Kalman filter is described.

- Moving-horizon estimation is approached in chapter 12. The chapter begins by the description of a moving window. Two methods are studied in this chapter. In the first method, the dynamic model is integrated by means of the fourth order Runge-Kutta method while in the second one differential equations are discretized by means of orthogonal collocations. In both approaches, a successive quadratic algorithm is used to perform optimization. In the second method, the optimization of collocations variables and process variables can be carried out sequentially or simultaneously.

- Chapter 13 is devoted to the theoretical development of a posteriori variances in the case of dynamic processes. This development is similar to the one used for the steady-state case.

- Three examples are studied in chapter 14: one tank, a network of five tanks and a stirred tank reactor with heat exchange. For each cases, the profiles on the entire horizon are given, a priori and a posteriori standard deviations are compared and error distributions are drawn.

- Chapter 15 is devoted to the conclusions of this second part.

The last part of the thesis has for subject the sensor network design for dynamic processes.

- In chapter 16, the sensor network design algorithm developed for steady-state processes is adapted to dynamic processes. The frequency of measurement is taken into account. The program is based on an observability criterion obtained thanks to a posteriori variances development.

- In chapter 17, sensor networks are designed for the three examples studied in chapter 14.

- Chapter 18 is devoted to the conclusions of this last part.

General conclusions and future works are presented in chapter 19.

Résumé

Afin de permettre le suivi et le contrôle des procédés chimiques, un réseau de capteurs doit être installé. Il doit permettre l'estimation de toutes les variables importantes du procédé. Cependant, toutes les mesures sont entachées d'erreurs, toutes les variables ne peuvent pas être mesurées et certains types de capteurs sont onéreux.
La réconciliation de données permet de corriger les mesures, d'estimer les valeurs des variables non mesurées et de calculer les incertitudes a posteriori de toutes les variables. Cependant, les écarts-types a posteriori sont fonction du nombre, de la position et de la précision des instruments de mesure qui sont installés.
Une méthode générale pour réaliser le design du réseau de capteur le moins onéreux capable d'estimer toutes les variables clés avec une précision déterminée dans le cas des procédés stationnaires a été développée. Cette méthode utilise une technique d'estimation des variances a posteriori basée sur l'analyse de la matrice de sensibilité. La fonction objectif du problème d'optimisation dépend du coût annualisé du réseau de capteurs et des précisions qui peuvent être obtenues pour les variables clés. Le problème est résolu au moyen d'un algorithme génétique.
Afin de réduire le temps de calcul, deux techniques de parallélisation utilisant une interface de passage de messages (MPI) ont été examinées: la parallélisation globale et les algorithmes génétiques distribués. Les deux méthodes ont été testées sur plusieurs exemples.
Afin d'étendre la méthode aux procédés fonctionnant de manière dynamique, une méthode de réconciliation dynamique des données permettant le calcul des variances a posteriori est nécessaire. La méthode des filtres de Kalman et une technique de fenêtre mobile basée sur les collocations orthogonales ont été comparées. Le calcul des variances a posteriori a été développé grâce à une méthode similaire à celle utilisée dans le cas stationnaire. La méthode a été validée sur plusieurs petits exemples.
Grâce à la méthode d'estimation des variances a posteriori, un critère d'observabilité a été défini pour les systèmes dynamiques de sorte que l'algorithme de design de réseaux de capteurs a pu être adapté aux systèmes dynamiques.
Un autre problème que les réseaux de capteurs doivent permettre de résoudre est la détection et la localisation des erreurs de procédé. La méthode a été adaptée afin de générer des réseaux de capteurs permettant de détecter et de localiser les erreurs de procédé parmi une liste d'erreurs dans le cas des procédés fonctionnant de manière stationnaire.

Organisation de la thèse

La thèse commence par un chapitre introductif suivi d'un chapitre consacré à l'état de l'art.

La première partie de la thèse est consacrée au design de réseaux de capteurs pour les procédés stationnaires. Elle est organisée de cette façon :

- Le troisième chapitre est consacré à la réconciliation de données non linéaire dans le cas des procédés stationnaires. Après une présentation de la nécessité de la réconciliation des données, le problème est formulé. Le problème d'optimisation contraint est transformé en un problème non contraint grâce à la méthode de Lagrange. On obtient la matrice de sensibilité du problème. En écrivant les conditions nécessaires d'optimalité, elle permet le calcul des variables réconciliées et de leur variances a posteriori.

- Dans le quatrième chapitre, la stratégie d'optimisation utilisée pour résoudre le problème de design de réseaux de capteurs est décrite. Ce problème étant combinatoire et généralement multimodal, il est résolu au moyen d'algorithmes génétiques.

- Les cinq étapes de l'algorithme de design des réseaux de capteurs sont décrites au chapitre 5.

- Le sixième chapitre est consacré à la parallélisation de l'algorithme. Il débute par la description de certaines notions et routines de parallélisation. Ensuite, deux techniques de parallélisation sont décrites : la parallélisation globale et les algorithmes génétiques distribués.

- Quatre exemples sont présentés au chapitre 7 : une boucle de synthèse d'ammoniac, une centrale turbine-gaz-vapeur, une unité de crackage de cétènes et une unité de reforming de naphta. Pour tous ces procédés, le design du meilleure réseaux de capteurs a été réalisé et les deux techniques de parallélisation sont comparées. Les paramètres des algorithmes génétiques distribués sont étudiés dans le cas de la boucle de synthèse d'ammoniac.

- Le design de réseaux de capteurs permettant la détection et la localisation de fautes de procédés dans les procédés fonctionnant de manière stationnaire est étudié au chapitre 8. La première partie de ce chapitre consiste en la description de la méthode de détection de pannes qui est utilisée. Ensuite, la méthode de design de réseaux de capteurs est expliquée. Finalement , l'algorithme est appliqués pour deux réseaux de distribution d'eau.

- Le chapitre 9 est consacré aux conclusions de cette première partie.

Dans la seconde partie de la thèse, la réconciliation de données dynamique et l'estimation des variances a posteriori sont abordées. Elle est divisée en six chapitres :

- Au chapitre 10, le problème de réconciliation de données dynamique est formulé.
- Le chapitre 11 est consacré aux méthodes de filtrage. Après une introduction générale aux techniques de filtrage, le filtre de Kalman étendu est décrit.
- Les techniques de fenêtre de temps mobile sont abordées au chapitres 12. Le chapitre débute par un description d'une fenêtre de temps mobile. Deux méthodes sont étudiées dans ce chapitre. Dans la première technique, le modèle dynamique est intégré au moyen de la méthode de Rune-Kutta du quatrième ordre tandis que dans la seconde méthode les équations différentielles sont discrétisées au moyen de collocations orthogonales. Dans les deux approches, un algorithme de programmation séquentielle quadratique est utilisé pour réaliser l'optimisation. Dans la seconde méthode, l'optimisation des variables de collocation et des variables du procédé peut être réalisée de manière séquentielle ou simultanée.
- Le chapitre 13 est consacré au développement théorique des variances a posteriori dans le cas des procédés dynamiques. Ce développement est similaire a celui utilisé dans le cas stationnaire.
- Trois exemples sont étudiés au chapitre 14 : une cuve, un réseau de cinq cuves et un réacteur à cuve parfaitement mélangée avec échange de chaleur. Pour chaque cas, les profiles des variables sont donnés pour l'ensemble de l'horizon de temps, les variances a priori et a posteriori sont comparées et les distributions des erreurs sont dessinées.
- Le chapitre 15 est consacré aux conclusions de la seconde partie.

La dernière partie de la thèse a pour sujet le design de réseaux de capteurs pour les procédés fonctionnant de manière dynamique.

- Au chapitre 16, l'algorithme de design de réseaux de capteurs développé pour le cas stationnaire est adapté aux procédés dynamiques. La fréquence des mesures est prise en compte. Le programme se base sur un critère d'observabilité obtenu grâce au développement de la méthode d'estimation des variances a posteriori.
- Au chapitre 17, le design de réseaux de capteurs est réalisé pour les trois exemples étudiés au chapitre 14.
- Le chapitre 18 est dédié aux conclusions de cette dernière partie.

Les conclusions générales et des pistes de travaux futures sont présentés au chapitre 19.

Acknowledgements

I would like to take the opportunity that is given to me to sincerely express my gratitude to everyone who close or by far has collaborated to the realization of this thesis.

I want to sincerely thank Professor Georges Heyen for the confidence he accorded me, by giving me the opportunity to carry out a Ph.D. within his staff, as well as for the precious advices he gave me.

I also would like to thank all the members of my thesis committee, particularly the exterior members, who accepted to participate to this Ph.D.

I am grateful to the Walloon Region and the European Social Funds who funded this research by way of the convention First-Europe OPTIMES. I also want to thank you Professor Marquardt for allowing me to realize an internship as part of the OPTIMES research project.

I want to thank you Professor Boris Kalitventzeff and Belsim s.a. who accepted to be the industrial partner of this project. I am grateful to Christophe Pirnay who helped me creating the interface of the program.

I thank you the members of the LASSC for the working atmosphere. I am particularly grateful to Christophe Ullrich for the memorable discussions we had about the dynamic data reconciliation. I also thank him for his writing advices.

Finally I would like to warmly thank my family, especially, my husband Mickaël and my children Marie, Emmanuel and Pierre for their support and their patience during all those years of research.

Contents

List of Figures

List of Tables

Nomenclature

Fault detection

$\boldsymbol{\mathcal{M}}$	the jacobian matrix of the model equations
$\boldsymbol{\Sigma}$	the signature matrix
ϵ	the noise on the variable
τ_{ij}	the lowest magnitude of the i^{th} residual that allows to distinguish between the noise and the fault f_j
e_i	the precision of the sensor on the i^{th} variable
f	the fault on the variable
$f_i\left(x_j\left(t\right)\right) \quad j=1,...,n$	the i^{th} fault at time t
m_{ij}	the element (ij) of the matrix of the derivatives of the residuals with respect to the variable
$r_i\left(t\right)$	the i^{th} residual at time t
$r_{\epsilon,i}$	the contribution of the noise to the i_{th} residual
$r_{f,i}$	the contribution of a unique fault f_j affecting the i^{th} residual
x	the the true value of the variable
$x_i\left(t\right)$	the i^{th} variable at time t
y	the measurement of the variable
n	the number of functions

Steady-state data reconciliation

A	the Jacobian matrix of the measured variables
B	the Jacobian matrix of the unmeasured variables

$\mathbf{C}$	the vector of the independent terms of the constraints
$\mathbf{f}(\mathbf{x}, \mathbf{z})$	the link equations
$\mathbf{W}$	the weight matrix (size= m x m)
$\mathbf{x}$	the vector of reconciled variables (size= m)
$\mathbf{y}$	the vector of measured variables (size= m)
$\mathbf{z}$	the vector of unmeasured variables (size= n)
m	the number of measured variables
n	the number of unmeasured variables
p	the number of equations of the model
$var(a)$	the variance of variable a

Dynamic data reconciliation

$\mathbf{\Lambda}$	the lagrange multipliers
$\boldsymbol{\mathcal{E}}$	the jacobian matrix of process and collocation constraints
$\boldsymbol{\mathcal{P}}$	the global weight matrix of the process variables
Φ	the goal function to minimize
σ	the precisions of the measurements
$\sigma_{u_{i,j}}$	the standard deviation on the input variable i at measurement time j
$\sigma_{x_{i,j}}$	the standard deviation on the differential state variable i at measurement time j
$\sigma_{z_{i,j}}$	the standard deviation on the algebraic state variable i at measurement time j
$\mathbf{A}$	the link equations
$\mathbf{B}$	the relations between the differential state variables and the Lagrange interpolation polynomials
$\mathbf{C}$	the linear interpolations of the values of input variables
$\mathbf{D}$	the residuals of the differential state equations at all collocation nodes
$\mathbf{E}$	the continuity constraints of the differential state variables between two discretization intervals

$\mathbf{F}$	the vector of independant terms of the linearisation of the link equations at the measurement times
$\mathbf{f}$	the differential constraints
$\mathbf{F}^c$	the vector of independant terms of the linearisation of the link equations at the collocation times
$\mathbf{G}$	the vector of independant terms of the linearisation of the $\mathbf{D}$ constraints at the collocation times
$\mathbf{g}$	the unequality constraints
$\mathbf{h}$	the equality constraints
$\mathbf{M}$	the sensitivity matrix
$\mathbf{P}_u$	the global weight matrix for the input variables
$\mathbf{P}_x$	the global weight matrix for the differential state variables
$\mathbf{P}_z$	the global weight matrix for the algebraic variables
$\mathbf{R}_u$	the relaxation factor of the input variables at the initial time of the moving horizon
$\mathbf{R}_x$	the relaxation factor of the differential state variables at the initial time of the moving horizon
$\mathbf{W}_u$	the weight matrix of the input state variables
$\mathbf{W}_x$	is the weight matrix of the differential state variables
$\mathbf{W}_z$	the weight matrix of the algebraic state variables
θ_k	the collocation times
F_goal	the the goal function
h_1	the measurement frequency
h_2	the size of the interpolation interval of the input variables
h_3	the size of the discretization interval of the differential state variables
h_4	the size of the window
h_5	the move of the window
N	the size of the sensitivity matrix $\mathbf{M}$

n_θ	the degree of Lagrange polynomials
$n_\theta^\star$	the number of collocation nodes on each discretization interval
n_{A^c}	the number of $\mathbf{A}^c$ constraints
n_A	the number of $\mathbf{A}$ constraints
n_B	the number of $\mathbf{B}$ constraints
n_{C^c}	the number of $\mathbf{C}^c$ constraints
n_C	the number of $\mathbf{C}$ constraints
$n_{discr\ int}$	the number of discretization intervals on the moving horizon
n_D	the number of $\mathbf{D}$ constraints
n_E	the number of $\mathbf{E}$ constraints
$n_{interp\ int}$	the number of interpolation intervals on the moving horizon
$n_{t_{mes}}$	the number of measurement times on the moving horizon different from the initial time of the horizon
$n_{t_{mes}}^\star$	the number of measurement times on the moving horizon including the initial time of the horizon
$n_{u_{mes}}$	the number of measured input variables
n_u	the number of input variables
$n_{x_{mes}}$	the number of measured differential state variables
n_x	the number of differential state variables
$n_{z_{mes}}$	the number of measured algebraic state variables
n_z	the number of algebraic variables
t_j	the measurement times
$time\ 0$	the initial time of the reconciliation window
$time\ N$	the last time of the reconciliation window
$u_{i,0}^{CI}$	the initial condition of the input variable i. It corresponds to the estimation of that variable at the same time of the previous reconciliation horizon.
$u_{i,j}$	the estimation of the input variable i at measurement time j

$u_{i,j}^m$	the measurement of the input variable i at measurement time j
$var(a)$	the variance of variable a
$x_{i,0}^{CI}$	the initial condition of the differential state variable i. It corresponds to the estimation of that variable at the same time of the previous reconciliation horizon
$x_{i,j}$	the estimation of the differential state variable i at measurement time j
$x_{i,j}^m$	the measurement of the differential state variable i at measurement time j
$z_{i,j}$	the estimation of the algebraic state variable i at measurement time j
$z_{i,j}^m$	the measurement of the algebraic state variable i at measurement time j

Filtering methods

$\overline{\mathbf{x}}_j$	the mean value of variables $\mathbf{x}$ at time step j
$\overline{\mathbf{y}}_j$	the mean value of variables $\mathbf{y}$ at time step j
$\mathbf{f}\left(\mathbf{x}_{j-1}, \mathbf{u}_{j-1}, \mathbf{v}_{j-1}\right)$	a set of non linear functions of $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{v}$ at time step j-1
$\mathbf{F}_j$	the Jacobian matrix of partial derivatives of f with respect to $\mathbf{x}$ at time step j (extended Kalman filter)
$\mathbf{h}\left(\mathbf{x}_j, \mathbf{w}_j\right)$	a set of non linear functions of $\mathbf{x}$ and $\mathbf{w}$ at time step j
$\mathbf{H}_j$	the Jacobian matrix of partial derivatives of h with respect to $\mathbf{x}$ at time step j (extended Kalman filter)
$\mathbf{P}_{j\|j-1}$	the covariance matrix of estimation errors at time step j
$\mathbf{P}_{j\|j}$	the covariance matrix of prediction errors at time step j
$\mathbf{Q}_j$	the process noises covariance matrix
$\mathbf{R}_j$	the measurement noises covariance matrix
$\mathbf{u}_{j-1}$	the vector of input variables at time step j-1
$\mathbf{v}_{j-1}$	the vector of process noises at time step j-1
$\mathbf{V}_j$	the Jacobian matrix of partial derivatives of f with respect to $\mathbf{v}$ at time step j (extended Kalman filter)
$\mathbf{w}_j$	the vector of measurement noises at time step j
$\mathbf{W}_j$	the Jacobian matrix of partial derivatives of h with respect to $\mathbf{w}$ at time step j (extended Kalman filter)

$\mathbf{x}_j$ the vector of state variables at time step j

$\mathbf{y}_j$ the vector of measurements at time step j

$\widehat{\mathbf{x}}_{j|j-1}$ the vector of predicted states at time step j

$\widehat{\mathbf{x}}_{j|j}$ the vector of estimated states at time step j

K_j the gain matrix at time step j

Sensor network design

$\text{penalty}_{\text{singular matrix}}$ the penalty factor for a singular sensitivity matrix

$\text{penalty}_{\text{target}}$ the penalty factor for the non-respected targets on key parameters

σ_i the accuracy obtained by the sensor network for the key variable i

σ_i^{target} the accuracy required for the key parameter i

C_{max} the cost of the most expensive sensor network

$fitness$ the goal function

$N_{\text{key variables}}$ the number of process key variables

$N_{\text{operating points}}$ the number of operating points

N_{windows} the number of reconciliation windows chosen for the sensor network design

$N^{\star}_{t_{mes}}$ the number of measurement times on the moving horizon including the initial time of the horizon

Parallelization

k_{chrom} the number of individuals estimated by one processor

n_{sensors} the number of sensors in the network

n_{eval} the number of goal function evaluations carried out by one processor for one individual

N_{pop} the size of the population

r_{sensors} the remainder of the division of the number of sensors in the chromosomes plus one by the number of processors

r_{chrom} the remainder of the division of the number of chromosomes by the number of processors

T_1 the time required to solve a problem B on a sequential computer

T_p the time required to solve a problem B on a cluster of p processors

Chapter 1

Introduction

1.1 Problem position

Nowadays, despite the progress achieved since the invention of the concept of control charts by Shewhart in the twenties, process control and monitoring remain challenging problems. Indeed, food and pharmaceutical industries, industries who produce high added value products, need very pure components. Security and environmental rules become more and more strict. Moreover, one always wants to produce more and less expensive. Thus the knowledge of the processes has to be more and more precise and one can not anymore achieve satisfactory process control by measuring only a few process variables. Process faults (pipes or tanks failure, catalyst deactivation,...) must be detected and localised faster, and any deviation from the nominal operating points must be corrected immediately.
One could imagine to quickly solve the problem by measuring all process variables but this would not be enough. Indeed, how could the efficiency of a compressor, the productivity of a plant, a reaction conversion ... be measured by a sensor? Moreover all measurements are erroneous to some extend and the precision reached on the estimates of unmeasurable variables is a function of the measurements precision. Furthermore, the measurement of some variables requires expensive sensors (concentration measurements). So one will prefer, if the precision constraints on key variables remained satisfied, estimating those variables from the measurements of other process variables instead of buying expensive measurement tools.
The technique of data reconciliation allows to estimate unmeasured variables and their accuracies. If redundant measurements are available, measurements are corrected and the uncertainty due to measurement errors can be reduced, so that measured variables are better known.
However, the number, the location and the precision of the sensors influence the accuracy of the estimates. The choice of the required sensors appears thus to be a very important task for industrial control and monitoring.
As it would be shown in the state of the art (chapter 2), the problem of sensor networks

design has been approached by several researchers during the last two decades.

1.2 Objectives

The thesis adresses two main objectives:

- The first objective is the development of a systematic method to design the cheapest sensor network that allows to satisfy all the following constraints for dynamic as well as steady-state processes:
  - all the process variables should be computable, even if they can not be directly measured (efficiency of a particular unit, reaction conversion...);
  - all key variables should be estimated within a prescribed accuracy;
  - if the process has several operating modes, the sensor network should be able to satisfy the two first conditions for all of them;
  - sometimes, one may want to be sure that the sensor network will be able to satisfy the two first conditions even in the case of one sensor failure or if one sensor is switched off for maintenance. So, one has to test those conditions for all configurations of the sensor network obtained by switching off one sensor.

  A variant of the method should allow the search of the cheapest sensor network that is able to detect and locate specific process faults in steady-sate processes.
- Secondly the development of a method to estimate a posteriori variances in the case of dynamic processes. That objective requires a dynamic data reconciliation algorithm that allows to estimate input variables as well as differential state and algebraic variables, and can generate a linearized sensitivity matrix.

1.3 Summary

In this study a general method to design the cheapest sensor network able to estimate all process key variables within a prescribed accuracy has been developed for stationary processes. It is based on a posteriori variances estimation method developed for steady-state data reconciliation. Using linearization of the process model at the nominal operating point, it allows to deal with non linear equations and to treat energy balances as well as mass balances.
In the suggested method, the problem is formulated as an optimization problem whose objective function depends on the annualized costs of the chosen sensors and on the accuracies achieved for the process key variables. A binary decision is assigned to each possible sensor (the presence or the absence of the sensor in the final network). Thus the optimization problem is not derivable but is often multimodal. We proposed to solve it using a genetic algorithm. When the process needs to be monitored for a wide range of operating

conditions, a single linearization might not be adequate for all of them. Thus one should consider optimizing simultaneously several linearized systems to identify a suitable compromise.
The computer time required to reach the solution is rather long with the proposed method and, because of sensitivity matrix inversions necessary to variances estimation, it increases more quickly than the size of the problem. Fortunately genetic algorithms can easily be parallelized. A global method and distributed genetic algorithms are approached in this study.
A variant of our sensor network design method allowing to detect and locate process fault is presented. The objective function does not depends anymore on the accuracies reached for the key variables, but depends on the dectectability and the isolability of the fault. The algorithm uses a method of process fault detection and localisation similar to the one described by Ragot (Ragot and Maquin, 2006).

Several techniques to correct dynamic data exist. Most of them seek the estimation of states or parameters. Other include the possibility to correct input variables. The main dynamic data reconciliation techniques are filtering methods and moving-horizon approaches. Nowadays, no dynamic data reconciliation method has proved is superiority on the other ones for all cases.
To transpose the sensor design method to dynamic data reconciliation, a posteriori variances must be evaluated analytically. We derived the equations allowing to estimate them in a general way.
The method proposed in this thesis is a moving-horizon approach. The discretization of differential equations is carried out by orthogonal collocations. The optimization of process and discretization variables is carried out simultaneously using the successive quadratic programming algorithm developed by Kyriakopoulou (Kyriakopoulou, 1997). The method provides a linearized system of equations like in the steady-state case. A sensitivity matrix can thus be evaluated and its non singularity is an a posteriori observability criterion of the system.

The thesis begins with an introductive chapter followed by a chapter devoted to the state of the art. The study is divided in three parts. The first part of the thesis (chapters 3 to 9) is devoted to the problem of sensor networks design for steady-state processes. Before developing the method based on steady-state data reconciliation and genetic algorithms, the problem of steady-state data reconciliation is formulated, a posteriori variances estimation is clarified and evolutionary algorithms are described.
Message passing interface algorithms are then used to develop parallelization techniques allowing to reduce the computing time. Global parallelization and distributed genetic algorithms are described and tested on several examples.
The sensor network design algorithm is finally modified to detect and locate process faults from a list of simulated faults.

The second part of the study (chapters 10 to 15) approaches the problems of dynamic data reconciliation and a posteriori variances estimation. First of all, the dynamic data

reconciliation problem is formulated. Then several filtering and moving horizon techniques are described: the extended Kalman filter and two moving horizon methods coupled with a successive quadratic programming optimization. In the first moving window method the integration of differential equations is carried out by means of the fourth order Runge-Kutta method. In the case of the second moving horizon technique, differential equations are discretized by means of orthogonal collocations. The process and discretization variables can be optimized sequentially or simultaneously. Those methods are compared on two examples.
The simultaneous orthogonal collocations based method being the most advantageous for the remaining of the project, it is chosen.
A method to estimate a posteriori variances from the discretized differential equations is developed: the constrained discretized optimization problem is transformed into an unconstrained one using Lagrange multipliers. If the optimality conditions are satisfied, one obtains the sensitivity matrix of the problem from which a posteriori variances can be deduced. The method is validated on several small examples.

The last part of the thesis (chapters 16 to 18) addresses the design of sensor networks for dynamic processes. A method based on the analysis of the sensitivity matrix is proposed like for the steady-state case. An a posteriori observability criterion is defined on the bases of that analysis. The algorithm is tested on the same examples that the dynamic data reconciliation method developed in the second part.

The general conclusions and future perspectives are presented in chapter 19.

Chapter 2

State of the art

This chapter is devoted to a bibliography study concerning the four main research areas useful to reach the objectives aimed by the thesis.

Sensor network design

During the last two decades, several researchers have pprposed methods to solve the problem of sensor network design. In 1987, Kretsovalis and Mah (Kretsovalis and Mah, 1987) developed methods based on linear algebra to design sensor networks that maximise the estimation accuracy. As they assumed that all variables are observable, their objective was to optimize the placement of redundant sensors.
Some years after, Madron (Madron, 1992) solved the problem of the design of the cheapest sensor network by using a graph oriented-method.
Ali and Narasimhan (Ali and Narasimhan, 1993) introduced the concept of reliability of estimation of a variable which is the probability with which a variable can be estimated when sensors are likely to fail. They have applied it to the design of observable sensor networks for stationary linear processes. They extended their work to redundant sensor networks (Ali and Narasimhan, 1995). They used a branch and bound optimization method to minimize the cost.
Sen et al (Sen et al., 1998) combined the concepts of graph theory and genetic algorithm in the case of linear processes. Their algorithm allowed to optimize a single criterion, either minimal cost, or maximum estimation accuracy. They limited their research to non-redundant sensor networks.
Bagajewicz (Bagajewicz, 1997) proposed a MINLP method to solve the problem of the cheapest sensor network for linear processes that can be submitted to constraints like a prescribed precision. The method is based on graph theory and linear algebra. He established with Sanchez (Bagajewicz and Sanchez, 1999b) the duality between the model of the maximum precision and the model of the minimum cost. They also presented a method for upgrading a sensor network with the goal of achieving a certain degree of observability or redundancy for a specified set of variables (Bagajewicz and Sanchez, 1999a).

Heyen et al proposed (Heyen and Gerkens, 2002), (Heyen et al., 2002) a general formulation for the sensor placement problem. Their goal is to reduce the noise while computing the estimates of all key variables within a prescribed accuracy. Studied problems are no longer limited to flow measurements and linear constraints. Optimization is carried out by means of genetic algorithms.
Carnero et al (Carnero et al., 2005) used an evolutionary technique based on genetic algorithms. It combines the use of structured populations in the form of neighborhood and a local search strategy. This method was applied to mass balances only.
Bhushan et al (Bhushan et al., 2008) used the concept of lexicographic optimization to solve the problem of the design of robust sensor network for fault diagnosis. Wailly at al (Wailly and Héraud, 2005), (Wally et al., 2008) also used the lexicographic programming and the Groëber bases to solve the sensor placement problem. The advantage of those methods is to allow to extend the mathematical process to the n-linear case.
Muradore et al presented (Muradore et al., 2006) a method for determining the optimal location of sensor in distributed sensor systems. This method has the advantage not do require an explicit model of the process. It is based on a sequential algorithm that selects the most informative measurement input at each iteration and updates the input and output spaces by subtracting information coming from the regressor.
Singh and Hahn (Singh and Hahn, 2005) proposed a method to determine the position of the sensor inside the process unit. This technique can be applied to linear and non-linear systems, to determine the optimal sensor network for states or for parameters estimation. It combines the computation of the observability covariance matrix with established measures for locating sensors in the case of linear processes. Van de Wouwer et al (de Wouwer et al., 2000) developed a criterion based on the test of independence between the response of the sensors in the case of state estimation or between the parameters sensitivities in the case of parameter estimation. That independence is measured by means of the Gram determinant. The advantage of this technique is avoiding the manipulation of covariance matrices. Waldraff et al (Walfraff et al., 1998) presented several observability measures based on observability matrix, observability gramian and the Popov-Belevitch-Hautus rank test. Those measures are only possible if input variables are considered as perfectly known. The authors used them to choose the optimal location of sensors in a tubular reactor.
Wong et al (Wang et al., 2002) optimized sensor location in a way to ensure fault observability and the fault resolution. Their technique is based on graph and on principal component analysis.
In the case of the dynamic processes, Benqlilou et al (Benqlilou et al., 2003), (Benqlilou, 2004) and (Benqlilou et al., 2005) solved the problem of sensor placement in the case where the dynamic reconciliation is made by means of Kalman filter. This involves that input variables are considered as perfectly known. Only mass balances were solved in his study. Genetic algorithms are used to perform optimization.

Heuristic algorithms

In the sixties, Fogel (Fogel et al., 1966) developed evolutionary programming to design state machines for predicting sequences and symbols. This method evolved in the eighties and became similar to evolution strategy. That method developed by Rechenberg (Rechenberg, 1971) and Schwefel (Schwefel, 1974) is based on the ideas of adaptation and evolution.
In the beginning of the seventies, John Holland and his colleagues described the learning classifiers systems and developed genetic algorithms (Holland, 1975). Those algorithms are described and illustrated in the book of Goldberg (Goldberg, 1989). In their paper, Herrera et al (Herrera et al., 1999) described several methods to distribute genetic algorithms, especially the method that is used in section B.2.2.
Other heuristic methods similar to genetic algorithms have been developed at the end of the eighties and at the beginning of the nineties. One can cite

- Greedy randomized adaptive search procedures developed by Feo and Resende in 1989 (Feo and Resende, 1995). Those algorithms consist of successive constructions of a greedy randomized solution improved by means of a local search. A greedy randomized solution is generated by choosing elements from a list in which they are ranked by a greedy function according to the quality they can achieve. Solution variability is achieved by placing good elements in a restrictive list from which they are chosen at random.

- Tabu search developed by Fred Glover (Glover and Laguna, 1997). This optimization method is based on the premise that an intelligent technique to solve problem must incorporate adaptive memory and responsive exploration. It is inspired by the traditional transmission of tabus by means of social memory which is subject to modifications over time. The status of the forbidden elements of tabu search being related to evolving memory can be changed according to time and circumstances.

- Ant algorithms developed by Dorigo (Colorni et al., 1996), (Dorigo et al., 2000). Those metaheuristics are multi-agents systems inspired by the observation of the behavior of ant colonies searching for food. Ants are able to solve shortest path problem in their natural environment: they can find shortest path to reach a food source from their nest without a good vision. Indeed, they use an aromatic essence called *pheromone* to give information to their colony about the food source. While an ant moves to the food source, it lays pheromone on the ground. The quantity of the deposit of pheromone depends on the quality of the food source and of the length of the path. As ants have a tendency to follow pheromone trails instead of choosing a new path, they will choose the path containing the largest amount of pheromone. After a certain period of time, shorter paths will be traversed more often than longer ones and will thus obtain larger amount of pheromone so that other ants will be attracted by those trails who will then be intensified. New ants will than be stimulated and the trails will be reinforced once more. Roughly speaking pheromone trails leading to rich, nearby food sources will be more frequented and will grow faster than trails leading to poor, far away food sources. Travelling salesman, quadratic assignment, vehicle routing, job shop scheduling, graph coloring, time tabling... are some examples of ant algorithms applications. Gutjahr (Gutjahr, 2000) described a general framework based on construction graph for solving combinatorial optimization problem by way of ant strategies. Ant algorithms can be parallelized like genetic algorithm as shown in the papers from Bullnheimer et al (Bullnheimer et al., 1997), Talbi et al (Talbi et al., 2001) and Shekolar et al (Shelokar et al., 2004).

- Simulated annealing invented by Kirkpatrick et al, (Kirkpatrick et al., 1983) and V. Cerny (Verny, 1985). This method is a generic probalistic metaheuristic algorithm for the global optimization problem. It is based on the analogy between the simulation of annealing of solids and the problem of solving large combinatorial optimization problems. In metallurgy, annealing is a physical process in which a solid is heated up in a heat bath by increasing the temperature of the bath to a maximum temperature at which all the particles of the solids arrange themselves randomly in the liquid phase. This heating is followed by a controlled cooling during which the temperature of the bath is slowly decreased. If the maximum temperature is high enough and the cooling is slow enough, all particules arrange themselves in the lowest energy state of a corresponding lattice. At each temperature of the cooling phase, the solids reach a thermal equilibrium characterized by a probability of being in a state with energy **E** given by the Boltzmann distribution:

$$P\{E = \mathbf{E}\} = \frac{1}{Z\left(T\right)}\ exp\left(-\frac{E}{k_B T}\right) \tag{2.1}$$

where

  - $Z\left(T\right)$ is a normalization factor called the partition function;
  - $exp\left(-\frac{E}{k_B T}\right)$ is the Boltzmann factor;
  - k_B is the Boltzmann constant: $k_B = 1.38\ 10^{-23} J/K$

When the temperature decreases, the Boltzmann distribution gives higher probabilities to the lowest energy states. When the temperature is close to zero, only the minimum energy state has a probability of occurrence higher than zero. However, if the cooling is too quick, the solid can not reach thermal equilibrium for all temperatures and a metastable amorphous structure is obtained instead of the lowest energy cristalline lattice structure. Metastable amorphous structures also appear when a quench is applied to the material namely when the heat bath temperature is lowered instantaneously.
A similar optimization method to simulated annealing is quantum annealing. In this method, thermal fluctuations are replaced by quantum fluctuations. It can outperforms simulated annealing if potential goal function landscape consists of very high but thin barriers surrounding shallow local minima.

Data reconciliation

Kuehn and Davidson (1961) first used steady-state data reconciliation in industry. Their objective was to correct process data in a way to satisfy mass balances. Some years later, Vaclavek (Vaclavek, 1968), (Vaclavek, 1969) also worked on the variable classification problem and on the formulation of the reconciliation model. Mah et al. suggested a procedure of variable classification based on graph theory (Mah et al., 1976). Crowe (Crowe, 1989) proposed an analysis based on a projection matrix method allowing to obtain a reduced system. A classification algorithm for general non linear equation systems was proposed by Joris and Kalitventzeff (Joris and Kalitventzeff, 1987). Heyen et al (Heyen et al., 1996) proposed a method based on the Jacobian matrix analysis to perform sensibility analysis: influence of the measurements and their accuracies on the accuracies of state variables, detection of state variables influenced by the accuracy of a particular measurement. Narasimhan and Jordache (Narasimhan and Jordache, 2000) provide a detailed perspective on the history of data reconciliation.

In the case of dynamic data reconciliation, several techniques exists. We were first interested in filtering techniques , in particular the Kalman filters, then our attention was turned on the moving horizon methods.
Kalman filter is said to be proposed by Kalman in 1960 (Kalman, 1960), (Kalman and Bucy, 1961) though Thiele and Swerling developed a similar algorithm earlier. That algorithm was first implemented by Schmidt (Schmidt, 1980) to solve the trajectory estimation problem for the Apollo program. This linear Kalman filter is used to carry out the reconciliation of linear dynamic processes.
It has been extended to deal with non linear systems (Karjala and Himmelblau, 1996), (Narasimhan and Jordache, 2000): the non linear part of the model is linearized thanks to a first order Taylor serie around the current estimated. This second filter is called *extended Kalman filter*. Kalman filter are well described in (Rousseaux-Peigneux, 1988) and (Welch and Bishop, 2001). Some authors, compared the extended Kalman filter with the moving horizon estimation (Jang et al., 1986), (Albuquerque and Biegler, 1995), (Haseltine and Rawlings, 2005). They showed that the results obtained by the moving horizon estimation are better than the one obtained with the extended Kalman filter but at the cost of more computing ressources.
A third method based on second order divided differences is also used in the case of non linear systems (Norgaard et al., 2000).
Nowadays, a wide variety of filtering methods has been developed (see for example (Bai et al., 2006), (Moraal and Grizzle, 1995), (Chen et al., 2008))and is widely used in engineering applications such as radars, computer vision, autonomous or assisted navigation.

Jang et al (Jang et al., 1986) introduced in 1986 the notion the moving horizon technique. It has been used afterwards by several authors like (Kim et al., 1991), (Liebman et al., 1992), (McBrayer et al., 1998), (Kong et al., 2000), (Barbosa et al., 2000), (Vachhani et al., 2001), (Abu-el zeet et al., 2002).
Differential equations can be discretized by means of orthogonal collocations. Frazer, Jones

and Skan and Lanczos applied for the first time collocation methods to differential equations in the thirties. Villadsen and his coauthor (Villadsen and Michelsen, 1978) have shown that the collocation nodes chosen at the zeros of orthogonal polynomials give the best results for the solving of differential equations. Other authors like Mingfang et al. (Kong et al., 2000) proposed to choose the measurement times as collocation nodes in the case of dynamic data reconciliation.
Biegler (Biegler, 1984) used orthogonal collocations to reduce the dynamic optimization problem to a constrained non linear optimization problem. He solved the resulting optimization problem thanks to a successive quadratic programming method. He also proposed to solve simultaneously collocation and process equations. Later, he presented with Albuquerque (Albuquerque and Biegler, 1995), (Albuquerque and Biegler, 1996) a method for decomposing the optimization algorithm in a way to reduce the computing effort. Their approach is based on line search, hessian update (Gauss-Newton or BFGS), and quadratic programming subproblem solving in the parameters space. Some years after, Biegler and his colleagues (Cervantes et al., 2000), (Biegler et al., 2002) applied an interior point method to non linear programming problem: they developed a reduced space decomposition for the NLP problem and applied quasi-Newton methods to the reduced Hessian. That approach allowed them applying their decomposition method to orthogonal collocations but it requires analytical derivatives. Kameswaran and Biegler (Kameswaran and Biegler, 2006), presented the advantages of simultaneous dynamic optimization strategies. Biegler and Lang (Biegler, 2007), (Lang and Biegler, 2007) described their simultaneous dynamic optimization method. In this method, they discretized state and input variables using collocations on finite elements. The differential equations are discretized by means of Radau collocations because they allow constraints to be set at the end of each element so that the system is better stabilized. Those authors also presented a description of their moving window.
Kabouris and Georgakakos (Kabouris and Georgakavos, 1996) extended the simplified maximum likelihood estimators for linear moving-window proposed by Maybeck in 1982. This technique is theoretically advantageous compared to the bayesian estimation approach that leads to the extended Kalman filter.
Barbosa et al (Barbosa et al., 2000) also transformed the differential equations using orthogonal collocations. They solved the constrained non linear problem using a successive quadratic programming method. BFGS (Broyden-Fletcher-Goldfrab-Shanno) technique was used to update the hessian matrix.
Vacchani et al (Vachhani et al., 2001) proposed a two-level approach to perform non linear dynamic data reconciliation and parameter bias detection. The first level consists of identifying biased parameter by means of fault diagnosis methods. In the second level, the non linear programming problem is solved so that measurement are reconciled and the biased parameter is estimated. In 2006, they combined the unscented Kalman filter used for non linear systems with recursive non linear dynamic data reconciliation (Vachhani et al., 2006). Their approach called the unscented recursive non linear dynamic data reconciliation is able to achieve state and parameter estimation. Accuracies of estimates is improved because the non linearity is not approximated during state and error covariance

matrix estimation.
Kim et al (Kim et al., 1990) compared three non linear error-in-variables parameter estimation: simultaneous parameter estimation and data reconciliation, two-stage error-in-variables using non linear programming and nested error-in-variables using non linear programming. They concluded that two-stage error-in-variables using non linear programming is superior with respect to robustness and computing efficiency, and is recommended in very non linear regions. They proposed a sequential approach based on numerical integration for parameter estimation and compared it with least-square based methods and non linear dynamic estimation using orthogonal collocations (Kim et al., 1991). Their non linear dynamic error-in-variable model gave better estimates than the least-square based methods, especially in presence of measurement bias. It is faster than the approach using orthogonal collocations and the optimization problem to be solved is smaller. Liebman et al (Liebman et al., 1992) presented a similar technique which is no longer limited to problem without constraints on the states. The method is able to performs as well dynamic data reconciliation as steady-state data reconciliation, it can estimate unknown parameters and unmeasured input variables. Liebman et al described the discretization of the differential equations by means of orthogonal collocations, thus generating a set of algebraic constrained equations. The time horizon is briefly described as well as the jacobian matrix of the constraints. The method is illustrated on a reactor test case that has become a well known reference. The approach was extended by McBrayer et al (McBrayer et al., 1998) so that it allowed measurement bias detection.
Renfro et al suggested (Renfro et al., 1987) to use the spline collocations instead of orthogonal collocations. The advantage of those collocations is that they allow to optimize larger and more difficult systems. Like Kameswaran and Biegler (Kameswaran and Biegler, 2006), they recommended a simultaneous approach for optimization and discretization.
Chen and Romagnoli (Chen and Romagnoli, 1998) suggested a formulation of dynamic data reconciliation that includes the outlier information so that dynamic data reconciliation and outliers detection can be carried out simultaneously.
Raff et al (Raff et al., 2005) suggested a moving window state estimation approach with global guaranteed convergence. In that method, they integrated optimization-based estimators and observability maps in the moving window approach.
Bagajewicz and Jiang (Bagajewicz and Jiang, 1997) presented a technique for dynamic data reconciliation based on an integral form of the equation system. This approach can be used in the presence of gross errors.
Binder et al (Binder et al., 1998) presented a deterministic formulation of the dynamic data reconciliation problem. That formulation is based on the theory of inverse problems. They also proposed a mathematical framework based on multiscale techniques for the discretization of the problem and for the solution of the discretized non linear equation system. Then they proposed a method to generate automatically a hierarchy of discretization grids to reach the best compromise between data and regulation error (Binder et al., 2002).
Wongrat et al (Wongrat et al., 2005) solved the problem of non-linear steady-state data reconciliation using genetic algorithm. In this method, the parameters of the genetic algorithm are first of all searched before the data reconciliation problem is solved. The method

is more time consuming than the other ones.

Fault detection

Gross errors and systematic biases must generally be eliminated before data reconciliation is carried out.
In the twenties, Shewhart invented the concept of control charts.
In the sixties and the seventies three types of statistical tests have been proposed for gross error detection (Reilly and Carpani, 1963), (Almasy, 1975) and (Mah et al., 1976). They are based on the residuals, individually or collectively (chi-square tests). Mah and Tamhane (Mah and Tamhane, 1982) proposed an identification test to identify the sources and location of gross errors.
Romagnoli and Stephanopoulos (Romagnoli and Stephanopoulos, 1981) presented a technique to analyse a set of measurements in the presence of gross errors. Based on the criterion of satisfaction of mass and energy balances, the method allows to quickly identify the source of errors. It work by serial deletion of one or more measurements from the set without requiring each time a new computation of the problem because available information coming from the original problem is used.
Narasimhan and Mah (Narasimhan and Mah, 1987) proposed a general method for identifying gross errors for steady-state processes. Their generalized likelihood ratio technique is based on the likelihood ratio statistical test. It allows the identification of all types of gross errors. They also developed a strategy that allows to identify multiple gross errors by using serial compensation of gross errors. Those authors extended their method to dynamic processes (Narasimhan and Mah, 1988).
Amand et al (Amand, 1999) and (Amand et al., 2001) developed a method to detect fault in chemical processes based on principal component analysis combined with data reconciliation. This method allows to reduce the number of variables that have to be monitored. It is decomposed in two steps: the fault detection (if some components are out of their confidence region) and the cause localisation (by estimation of the effect of those components on the original variables). In 2002, Wang et al (Wang et al., 2002) improved the principal component analysis technique. They replaced the Q statistic used in conventional principal component analysis by two new statistics: the principal-component-related variable residuals and the common variables residuals.
Jia et al (Jia et al., 1998) proposed a non linear principal component analysis technique based on the input-training neural network. They defined multivariate statistical process control charts with non-parametric control limits instead of the traditional limits based on the normality assumption.
Ragot et al proposed (Ragot et al., 2003) a blind approach to detect and isolate sensor faults. That method does not need any knowledge about the process model. It is only based on the analysis of the data so that it requires strong hypothesis about the input signals of the studied process. Later (Ragot and Maquin, 2006), they developed a technique for fault measurement detection in an urban water network. We used this method based

on the analysis of the signature matrix and fuzzy rules for the design of sensor networks allowing to detect and locate process faults.
Mc Brayer and Edgar (McBrayer and Edgar, 1995) proposed a method for bias detection and estimation for dynamic processes. Their method is based on the examination of the residuals.
Kong et al (Kong et al., 2004) developed a gross error identification strategy based on parameter estimation. This method suits for dynamic processes and allow the detection of simultaneous measurement gross errors. They described the condition for which a system is identifiable for gross error detection: let the system of equations:

$$\begin{cases} \frac{dx}{dt} = f\left(x, u\right) \\ \frac{dx}{dt} = f\left(x + \delta_x, u + \delta_u\right) \end{cases}$$

If the column vectors of $\bigtriangledown f$ are linearly independent, the system of equations has a unique solution of δ_x and δ_u and then the system is identifiable for gross error detection. The authors obtained the same conclusion for differential algebraic equation systems.
Bagajewicz and Jiang (Bagajewicz and Jiang, 1998) developed a technique for multiple gross error detection based on the integral approach for linear dynamic data reconciliation they presented earlier (Bagajewicz and Jiang, 1997).
Wang et al (Wang et al., 2004) presented an improvement in the measurement test - nodal test proposed by Yang. Their approach is used to solve the decrease of matrix rank in gross error detection and data reconciliation. Gross errors are detected before being corrected by successive iterations.
Bhagwat et al (Bhagwat et al., 2003a), (Bhagwat et al., 2003b) presented a method to detect process fault during the transition operations. In this model-based fault detection scheme, non linear transient systems are decomposed into multiple linear modeling regimes. State estimation and residuals computation are carried out by means of Kalman filters and open-loop observers. On-line detection and localisation of faults are enabled thanks to residuals analysis by means of thresholds, fault tags and logic charts.

Part I

Network design for steady-state processes

Chapter 3

Steady-state data reconciliation

Chemical plants must be efficiently monitored to allow production accounting as well as for the enforcement of safety and environmental rules. Process control can only take place if enough measurements are carried out on the plant. Those measurements and laboratory analysis are generally erroneous. So information about the process is inaccurate, the state of the process at a certain time is misrepresented and the control of the plant is not as efficient as it could be so that the process conditions may be not met, the efficiency indicators may be badly estimated and the safety rules may be unsatisfied. Some errors come from the measurement tool itself: indeed, each sensor has a limited precision. Moreover, most of the sensors must be calibrated regularly to avoid systematic errors due to a sensor giving always an under or an over estimated value. Finally, during the transmission and the conversion of the signal, noise is added to the initial measurement value. Another source of error comes from the fact that the measurement frequency depends on the kind of sensor: the fastest measurements are carried out in some seconds while compositions analysis can take up to several hours. It is thus impossible to have the information about the whole plant at a precise time. This problem can be partly solved by taking average values of measurements that can be done in the case of steady-state processes. The location of the sensors in the unit is also important for the validity of measurements: indeed, the value will be different following, for example, if the sensor is placed in the middle of a reactor or near its walls, or if the region where the sensor is placed is turbulent or not. Finally gross errors can appear if a sensor fails.
The relationship between a measurement of a variable and its true value can be represented as follow:

$$y_{measured} = y_{true} + e_y \tag{3.1}$$

where

- $y_{measured}$ is the measured value of the variable y;
- y_{true} is the true value of the variable y;
- e_y is the error on the variable y.

Errors on the measurements can be of three kinds:

- random errors: those errors are typically assumed to be zero-mean and normally distributed (Gaussian errors). This kind of error is generally attributed to the non reproducibility of the measurement tool. Moreover, neither its magnitude nor its sign can be predicted. Those errors can not be completely eliminated and are always present in the measurements.

- systematic errors: this second type of error appears when the sensor gives systematically incorrect values that are always higher or lower than the true value. Their mean error is never zero. A bad calibration of the measurement tool can lead to those errors.

- gross errors: those errors are generally caused by non random events. In this case, the value of the measurement is very different from the expected one. They can be produced by sensor or process failures.

Statistical methods based on the process model, such as data reconciliation, have been developed to analyse and reconcile plant measurements. The objective of such method is to reduce errors on measurements as much as possible and to estimate all variables of the process whatever they are measured or not. Doing this way, key parameters of the plant can be estimated from the corrected measurements and the model equations. Such performance indicators can be efficiencies of turbines or compressors, productivities, reaction conversions, catalyst deactivations or other performances or safety indicators. To perform data reconciliation, the number of available measurements should be larger than the number of degrees of freedom of the model. However, it may happen that the number of measurements is higher than the number of degrees of freedom of the model so that the key variables can be obtained in different ways using different measurement combinations. Those measurement redundancies are not a problem but is a source of information: indeed, using the concept of data reconciliation, they allow to detect and quantify the errors and to reduce the uncertainty on the measurements and the estimated variables. The data reconciliation algorithm corrects only the redundant variables. The redundancies can be of three types:

- temporal: the same measurement is carried out at different times. This type of redundancy is used in dynamic data reconciliation which is described in chapter 8;

- spacial: this type of redundancy is obtained when several identical sensors are installed to mesure the same variable;

- structural: this type of redundancy consists of estimating the same variable with different sensors of different types. For example (figure 3.1), a flow rate can be estimated by the measurement of a flow meter, as a function of the pressure drop between two pressure measurements, or by the following ratio $\frac{Q}{C_p(T_2-T_1)}$ where T_1 and T_2 are two temperature measurements, C_p is the specific heat of the stream and Q is the heat load of the exchanger.

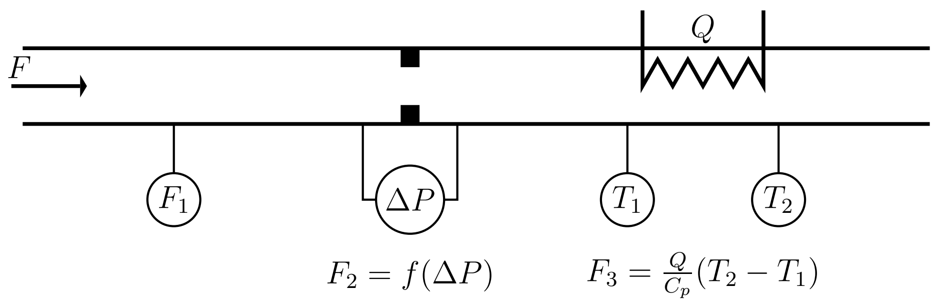


Figure 3.1: Flow rate measurements

Kuehn and Davidson (Kuehn and Davidson, 1961) were the first to use steady-state data reconciliation in industry. Their objective was to correct process data so that the mass balances were satisfied. Nowadays data reconciliation methods correct energy balances as well as mass balances.

3.1 Formulation for non linear steady-state systems

This section on data reconciliation treats non linear steady-state problems where all variables are not necessary measured. The equations of the model are mass, component and energy balances, equilibrium equations and link equations that relates measurements to state or key variables. This reconciliation problem can be formulated this way:

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{z}} (\mathbf{x}-\mathbf{y})^T \mathbf{W} (\mathbf{x}-\mathbf{y}) \\ &s.t. \qquad \mathbf{f}(\mathbf{x},\mathbf{z}) = \mathbf{0} \end{aligned} \tag{3.2}$$

where

- $\mathbf{x}$ is the vector of estimated variables (size= m);
- $\mathbf{y}$ is the vector of measured variables (size= m);
- $\mathbf{z}$ is the vector of unmeasured variables (size= n);
- $\mathbf{W}$ is the weight matrix (usually the inverse of the measurements errors covariance) (size= $m\ x\ m$);
- $\mathbf{f}(\mathbf{x},\mathbf{z})$ are the link equations.

This constrained minimization problem can be transformed into an unconstrained minimization problem using the vector of Lagrange multipliers Λ:

$$\min_{\mathbf{x},\mathbf{z},\Lambda} L(\mathbf{x},\mathbf{z},\Lambda) = (\mathbf{x}-\mathbf{y})^T \mathbf{W} (\mathbf{x}-\mathbf{y}) + 2\ \Lambda^T\ \mathbf{f}(\mathbf{x},\mathbf{z}) \tag{3.3}$$

The solution must verify the optimality conditions:

$$\begin{aligned}\frac{\delta\ L}{\delta\ \mathbf{x}} &= \mathbf{W}\,(\mathbf{x}-\mathbf{y})+\mathbf{A}^T\ \mathbf{\Lambda}=\mathbf{0}\\ \frac{\delta\ L}{\delta\ \mathbf{z}} &= \mathbf{B}^T\ \mathbf{\Lambda}=\mathbf{0}\\ \frac{\delta\ L}{\delta\ \mathbf{\Lambda}} &= \mathbf{f}\,(\mathbf{x},\mathbf{z})=\mathbf{0}\end{aligned} \tag{3.4}$$

where

- $\mathbf{A}=\frac{\delta\ \mathbf{f}(\mathbf{x},\mathbf{z})}{\delta\ \mathbf{x}}$ is the Jacobian matrix of the measured variables
- $\mathbf{B}=\frac{\delta\ \mathbf{f}(\mathbf{x},\mathbf{z})}{\delta\ \mathbf{z}}$ is the Jacobian matrix of the unmeasured variables
- $\mathbf{C}$ is the vector of the independent terms of the constraints

The last optimality condition can be linearized this way:

$$\frac{\delta\ L}{\delta\ \mathbf{\Lambda}}=\mathbf{A}\ \mathbf{x}+\mathbf{B}\ \mathbf{z}+\mathbf{C} \tag{3.5}$$

The equation system 3.4 is non linear and has to be solved iteratively. For small size problems, the Newton-Raphson method can be used. It requires a solution for successive linearizations of the initial equation system 3.4:

$$\begin{pmatrix}\mathbf{x}\\ \mathbf{z}\\ \mathbf{\Lambda}\end{pmatrix}=\mathbf{M}^{-1}\begin{pmatrix}\mathbf{W}\ \mathbf{y}\\ \mathbf{0}\\ \textbf{-C}\end{pmatrix} \tag{3.6}$$

where $\mathbf{M}$ is the Jacobian matrix of the equation system 3.4 called the *sensitivity matrix*:

$$\mathbf{M}=\begin{pmatrix}\mathbf{W} & \mathbf{0} & \mathbf{A}^T\\ \mathbf{0} & \mathbf{0} & \mathbf{B}^T\\ \mathbf{A} & \mathbf{B} & \mathbf{0}\end{pmatrix} \tag{3.7}$$

For larger size problems numerical algorithms like Powell's dogleg method give generally good results (Chen and Stadherr, 1981). If the problems are very large, the sparsity of the sensitivity matrix has to be taken into account and the algorithm has to be modified as described, for example, by Chen and Stadherr (Chen and Stadherr, 1984). In that case, the measurements are generally considered as independent from one another so that the weight matrix $\mathbf{W}$ is reduced to a diagonal matrix whose elements are the inverse of the variances of the measured variables.
If the problem contains inequality constraints, the NLP problem is solved directly using a sequential quadratic programming algorithm (Kyriakopoulou, 1997). In SQP algorithms, an approximation of the original problem is solved at each iteration. The quadratic objective function is retained and the contraints of the model are linearized around the current

estimate of the solution.
Once the problem has converged, a sensitivity analysis can be carried out using the linearized equation system 3.6. This equation system shows that the reconciled values of variables **x**, **z** and **Λ** are linear combinations of the measurements. So, the sensitivity matrix **M** allows to evaluate how the reconciled values of the model variables depend on the measurements and their standard deviations. **x** and **z** variables are thus estimated this way:

$$\begin{aligned}
x_i &= \sum_{j=1}^{m+n+p} (M)^{-1}_{i,j} \begin{pmatrix} \mathbf{W\ y} \\ \mathbf{0} \\ \textbf{-C} \end{pmatrix}_j \\
&= \sum_{j=1}^{m} (M)^{-1}_{i,j}\ W_{i,j}\ y_j - \sum_{j=1}^{p} (M)^{-1}_{i,m+n+j}\ C_j \qquad (3.8)\\
z_i &= \sum_{j=1}^{m+n+p} (M)^{-1}_{m+i,j} \begin{pmatrix} \mathbf{W\ y} \\ \mathbf{0} \\ \textbf{-C} \end{pmatrix}_j \\
&= \sum_{j=1}^{m} (M)^{-1}_{m+i,j}\ W_{i,j}\ y_j - \sum_{j=1}^{p} (M)^{-1}_{m+i,m+n+j}\ C_j \qquad (3.9)
\end{aligned}$$

where

- m is the number of measured variables;
- n is the number of unmeasured variables;
- p is the number of equations of the model.

Knowing that the variance of a linear combination LC of several variables x_i is given by:

$$LC = \sum_{i=1}^{m} a_i\ x_i \qquad (3.10)$$

$$var\,(LC) = \sum_{i=1}^{m} {a_i}^2\ var\,(x_i) \qquad (3.11)$$

the variances of the variables of the model are estimated this way:

$$\begin{aligned} var\left(x_i\right) &= \sum_{j=1}^{m} \left\{ (M)^{-1}_{i,j} \;\; W_{j,j} \right\}^2 \; var\left(y_j\right) \\ &= \sum_{j=1}^{m} \frac{\left[(M)^{-1}_{i,j} \right]^2}{var\left(y_j\right)} \end{aligned} \tag{3.12}$$

$$\begin{aligned} var\left(z_i\right) &= \sum_{j=1}^{m} \left\{ (M)^{-1}_{m+i,j} \;\; W_{j,j} \right\}^2 \; var\left(y_j\right) \\ &= \sum_{j=1}^{m} \frac{\left[(M)^{-1}_{m+i,j} \right]^2}{var\left(y_j\right)} \end{aligned} \tag{3.13}$$

The analysis of the variances of the reconciled variables gives the effect of each measurement in the estimation of the variables of the problem. It is then possible to find out the measurements that have a great importance and must be carried out precisely and those that do not contribute a lot in the variables calculation.
The ratio between the variance of the reconciled value and the corresponding measurement gives the confidence improvement given by the reconciliation. Only the measured variables can be improved by reconciliation. Nevertheless, the reliability of the estimates of the unmeasured variables is also quantified.
The sensitivity analysis also allows to locate sensors whose accuracy should be improved to reduce the uncertainty on the process key parameters. Indeed, for each variable, a list of all measurements used to estimate the variables is drawn up. The variance of the reconciled variable is then estimated as well has its sensitivity with respect to the variances of the measurements.
In the case of the sensor placement algorithm, all measurements will be considered as measured but a very high variances will be attributed to the unmeasured variables (10^{+30}). Thus the equation system 3.6 becomes:

$$\begin{pmatrix} \mathbf{x} \\ \Lambda \end{pmatrix} = \mathbf{M}^{-1} \begin{pmatrix} \mathbf{W\ y} \\ \textbf{-C} \end{pmatrix} \tag{3.14}$$

with the sensitivity matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{W} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \tag{3.15}$$

This uncertainty of unmeasured variables is so high that the algorithm will consider their inverse equal to zero in the sensitivity matrix and the contribution of those variables with respect to the reconciled variables and their variances estimations will be null. All variables will then be estimated thanks to equation 3.8 and the corresponding variances thanks to equation 3.12.

Chapter 4

Description of the optimal sensor network design algorithm

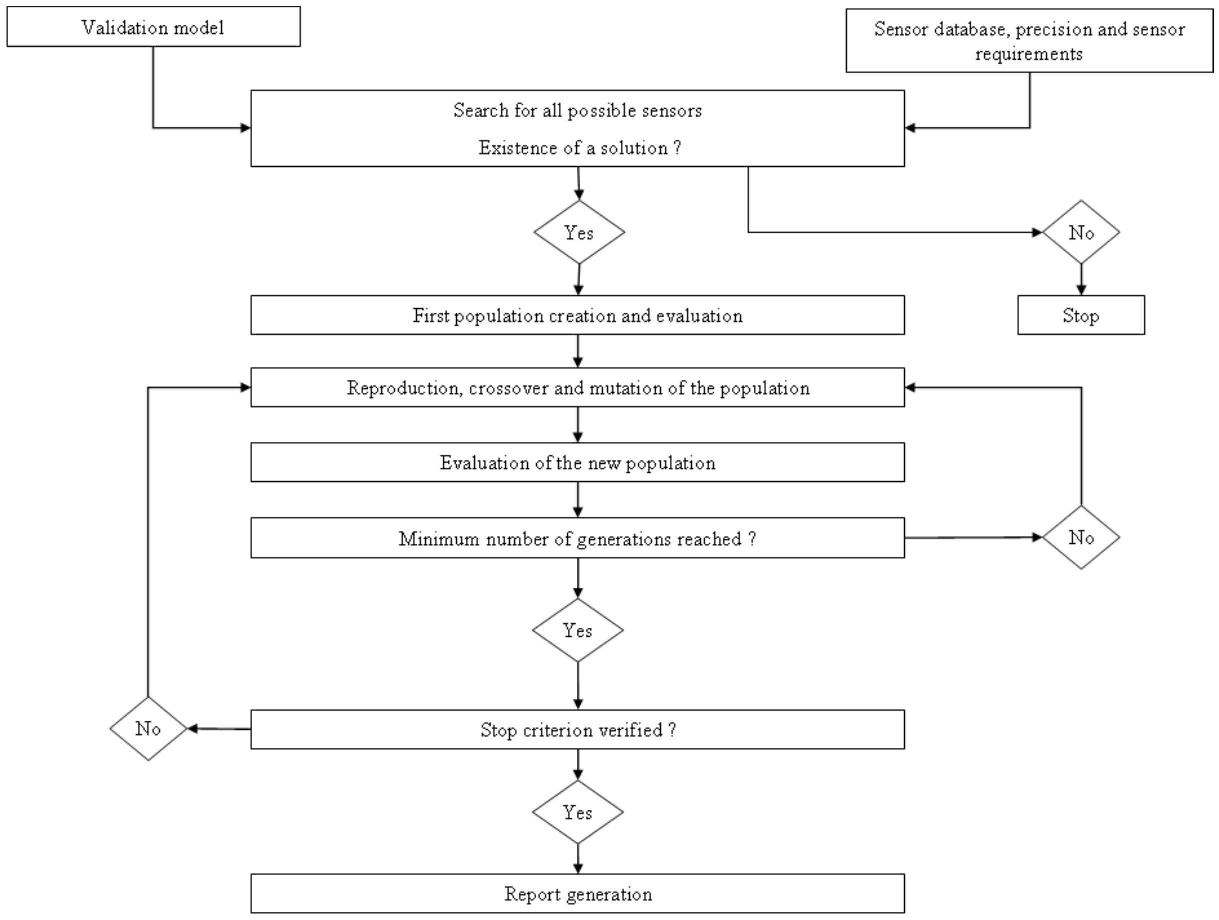
In this chapter, a method to design the best sensor networks for the steady-state case will be proposed. The networks suggested by the method should be the less expensive ones that allow to satisfy all the following constraints:

- all the variables of the process should be computable, even unmeasurable variables (efficiency of a particular unit, reaction conversion...);
- all key variables should be estimated within a prescribed accuracy;
- if the process has several operating modes, the sensor network should be able to satisfy the two first conditions for all of them;
- sometimes, one may want to be sure that the sensor network will be able to satisfy the two first conditions even in the case of one sensor failure or if one sensor is switched off for maintenance. So, one has to test those conditions for all configurations of the sensor network obtained by switching off one sensor.

In most of the cases, the objective function of this problem is multimodal. Moreover, the problem is not derivable and contained a large number of binary parameters (the sensors are chosen or not) so that a genetic algorithm is used to perform the optimization. The implementation that is adopted is based on the freeware code developed by Carroll (Carroll, 2001). The selection of individuals involves tournament selection with a shuffling technique for choosing pairs for mating. The evolution algorithm includes single-point cross-over and jump mutation. Evolution algorithms are described in annex A.

The optimal sensor network design algorithm is carried out in five steps, which are described in details in this chapter:

1. Formulation of the reconciliation model and model linearisation;
2. Specification of sensor database, precision requirements and sensor requirements;



Validation model
Sensor database, precision and sensor requirements
Search for all possible sensors
Existence of a solution ?
Yes
No
Stop
First population creation and evaluation
Reproduction, crossover and mutation of the population
Evaluation of the new population
Minimum number of generations reached ?
No
Yes
Stop criterion verified ?
No
Yes
Report generation

Figure 4.1: Flow diagram of the genetic algorithm

3. Verification of the problem feasibility;
4. Optimization of the sensor network;
5. Report generation.

A flow diagram of the algorithm is drawn on figure 4.1.

4.1 Formulation of the reconciliation model and model linearisation

The reconciliation model of the process is first built with the Vali4 software data reconciliation (Belsim, 2004). In this software, the icons that represent the common unit operations are drawn in a process flow diagram and linked together by means of material and energy streams. For material streams, a model for physical and thermodynamic properties is chosen from the proposed list: ideal, Soave, Redlich-Kwong, NBS, Peng-Robinson...

The values of some measurements are specified in the Vali4 software. The number of the measurements must at least be equal to the number of degrees of freedom of the process to allow the equation system to be solved. Redundancy gives supplementary information and the reconciliation algorithm gives a *least square solution*. As all variables are not state variables (enthalpy, pressure, partial molar flow rate...), the Vali4 software writes automatically the link equations for standard measurements types that are not state variables (mass flow rates, molar fractions, temperatures...). For non-standard unmeasured variables (heat transfert coefficient, compressor efficiency, pressure drop, reaction conversion...), which need often to be known within a prescribed accuracy, links equations have to be created. Once the process is modeled, the reconciliation problem is solved using either the Lagrange multiplier method or the SQP solver. When the solution is obtained, the sensitivity analysis is carried out (Heyen et al., 1996) and a report file is generated. It contains the value of all variables and their reconciled accuracy, the linearised equations and the non-zero coefficients of the jacobian matrix. As the equations of the model are linearised at the operating point, a report is needed for each operating point for which the measurement system has to be designed. Moreover, a report is necessary for the first operating point calculated with ideal thermodynamics. Indeed, in certain thermodynamic models, the enthalpy is lightly influenced by the pressure. So, in the jacobian matrix of the constraints, zeros are replaced by the contributions of the pressures on the enthalpies. The sensitivity matrix is thus a little different from the one obtained with the ideal thermodynamic. To ensure that the observability criterion, which is the non singularity of the sensitivity matrix, is satisfied, the chosen sensor network has to be tested on the sensitivity matrix corresponding to the ideal thermodynamic. The thermodynamic specific to the model are used to estimate properly the values of a posteriori variances.

4.2 Specification of sensor database, precision requirements and sensor requirements

Besides the Vali4 reports, the program needs three data files. Those files contain the sensor database, the precision requirements and the sensor requirements.

4.2.1 Sensor database

The sensor database is a list of sensors from which the algorithm must choose all the possible sensor placements for the studied plant. It must include, for each sensor, the following information:

- the name of the sensor;
- the annualized cost of the sensor. This cost must take into account:

  - the annualized purchase cost;
  - the annualized installation cost;
  - the annualized operating cost.

  All those costs depend on the studied process: indeed, for example, the cost of the sensor will be different if it is placed in a flow of water or if it must resists to acid or base.

- the parameters σ_{A_i} and σ_{B_i} that allow the estimation of the sensor standard deviation σ_j for the measured variable $X_j^{'}$:

$$\sigma_j = \sigma_{A_i} + \sigma_{B_i} * X_j^{'} \tag{4.1}$$

- the type of measurement place:
  - S for a stream variable (temperature, pressure);
  - M for a mixing variable (flowrate, concentration);
  - U for a unit variable (efficiency of the unit);
- the minimal and maximal values of the sensor measurement range;
- the type of variable the sensor is able to measure. Types of variable are defined as in Vali4 :
  - T for a temperature;
  - P for a pressure;
  - DP* for a pressure drop;
  - MF* for a molar fraction;
  - MOLF for a molar flow rate;
  - MASSF for a mass flow rate.

4.2.2 Precision requirements

In this file, key variables that have to be known precisely are listed with their required target accuracy (maximum standard deviation). Those variables are, for instance, turbine efficiency, catalyst deactivation, productivity, reactive conversion...
At each generation of the genetic algorithm, it is checked whether the key variables accuracies are achieved or not. In the case they are not acceptable, a penalty is added to the goal function of the optimization problem; otherwise, a bonus is added.

4.2.3 Sensor requirements

A file is proposed to list

- existing sensors that should not be removed;
- plant locations where sensor placements have to be avoided;
- plant locations where sensor locations are mandatory.

4.3 Verification of the problem feasibility

This step of the program begins by computing the list of all the sensors that can be placed in the plant. Each chromosome created by the algorithm will contain a number of binary decisions equal to the maximum number of sensors. The binary genes take the value 1 or 0 depending if the corresponding sensors are chosen or not. To check the problem feasibility, the algorithm first generates an individual whose genes are all set to "1", thus implementing all possible sensors. This is the most expensive solution, but it should also result in the most precise estimates for the key process variables. Thus, if the required precision is not archived for this solution, one may conclude that the problem has no solution. If several sensors measure the same variables, only the variance of the most accurate one is taken into account.
To ensure there exists a solution to the studied problem, the two following conditions have to be met:

- The sensitivity matrix of the problem is non-singular for the ideal thermodynamic case as well as for all operating points. If this condition is not met, the program stops.
- The accuracies on all key parameters have to be acceptable. If this condition is not met, the program may continue, but a penalty is added to the goal function.

There are different ways to cure those problems:

- adding more accurate sensors to the data base;
- adding sensors able to measure other types of variable;
- adding more extra measurable variables so that more variables can be measured.

Individuals of the first generation are chosen randomly by biasing the initial chromosome: to be sure that the number of chosen sensors is at least equal to the number of degrees of freedom of the problem, a high probability of selection if fixed for each sensor. A value of 80 % is typically chosen but this parameter appears not to be critical for the problem of optimal sensor design. For the other parameters of the genetic algorithm, the following values are generally chosen:

- the population size, which does not either appear to be critical , was most of the time chosen to 20 individuals per population;
- the probability of reproduction was fixed to 50 %;
- the probability of single-point cross-over was chosen to 50 %
- the probability of jump mutation after reproduction and cross-over is 1 %.

4.4 Optimization of the sensor network

Knowing that a feasible solution exists, the search for the optimal sensor configuration can begin. The fitness function is evaluated as follows:

- If the sensitivity matrix is singular:

$$\text{Fitness} = -C_{max}\ \text{penalty}_{\text{singular matrix}} \tag{4.2}$$

  where

  - C_{max} is the cost relative to the most expensive sensor network which corresponds to the first chromosome;
  - $\text{penalty}_{\text{singular matrix}}$ is the penalty factor for a singular sensitivity matrix and is generally chosen equal to 2.

- otherwise

$$\text{Fitness} = -\text{cost} - \sum_{i=1}^{N_{\text{key variables}}} \begin{cases} -\frac{\sigma_i}{\sigma_i^{\text{target}}} & if \quad \frac{\sigma_i}{\sigma_i^{\text{target}}} \leq 1 \\ \frac{\sigma_i^2}{\left(\sigma_i^{\text{target}}\right)^2} \frac{C_{max}\ \text{penalty}_{\text{target}}}{N_{\text{key variables}}} & if \quad 1 < \frac{\sigma_i}{\sigma_i^{\text{target}}} < 10 \\ 10^2 \frac{C_{max}\ \text{penalty}_{\text{target}}}{N_{\text{key variables}}} & if \quad \frac{\sigma_i}{\sigma_i^{\text{target}}} \geq 10 \end{cases} \tag{4.3}$$

  where

  - $N_{\text{key variables}}$ is the number of process key variables;
  - $\text{penalty}_{\text{target}}$ is the penalty factor for the targets on key parameters that are not respected and is fixed equal to 2;
  - σ_i is the accuracy obtained by the sensor network for the key variable i;
  - σ_i^{target} is the accuracy required for the key parameter i.

  In this second case, a penalty is added to the objective function if the required accuracy on a key parameter is not satisfied. Otherwise a gain is added.

If the sensor network has to be available for several operating points, the fitness becomes:

$$\text{Fitness} = \sum_{j=1}^{N_{\text{operating points}}} \text{fitness}_j - (N_{\text{operating points}} - 1)\ \text{cost} \tag{4.4}$$

where $N_{\text{operating points}}$ is the number of operating points.
If a measurement system observable in the case of one sensor failure has to be carried out, the fitness is evaluated for all configurations obtained by eliminating successively one of the sensors, and the worst one is kept.
Once the population has been generated the goal function has to be evaluated. In order to estimate reconciled variances of all variables, the sensitivity matrix corresponding to the specific thermodynamic has to be inverted before a posteriori variances are estimated by means of equations 3.6. The sensitivity matrix being symmetric, the subroutine "MA29AD" from Harwell library (Harwell, 1990), which allows to factorize a symmetric matrix, was first used. However, the problem that is attempted to be solved contains a large number of variables and equations and its sensitivity matrix is very sparse. Thus a sparse matrix factorisation code, available in Belsim, was selected. This code is based on the algorithm proposed by Chen and Stadherr (Chen and Stadherr, 1984). It allows to reduce the computing time by 25 for matrices of size 300 so that it was finally chosen.
At each generation, the best individual is kept and duplicated in the case it would mutate during the next generation.
If after a specified number of generations (n) the best chromosome remains unchanged, it is considered has the solution of the problem. There is no certainty that this solution is the best one, but it is available and much better than the first individual.
The computing times to reach the solution with the algorithm can be very important for large scale problems has it is illustrated in annex B. That is why parallelization has been used as described in chapter 5. Parallelization results are presented on chapter 6.

4.5 Report generation

At the last step, the programme generates a report containing the list of all possible sensors and the list of chosen sensors with their location. It also gives, for the key parameters, a comparison between the accuracies obtained by the best network and the target ones. Finally, files containing the name of the variables with their values, a priori accuracies and physical units are generated. Those files can be used as measurements files directly in the Vali4.

Chapter 5

Case studies

Several processes have been studied to test the performance of the algorithm and compare the ways it has been parallelized. Four of them are described in this chapter:

- an ammonia synthesis loop: this process is a middle size problem (224 variables) whose data have been obtained by simulation;
- a combined cycle electricity generation plant: the size of this problem is similar to the ammonia synthesis loop (260 variables) but the values of the variables have been collected by measurement;
- a ketene cracker: this problem is a quite larger size problem (631 variables) whose variables have also been measured;
- a naphta reformer: this process is a larger size problem (1263 variables) whose data have been obtained by simulation;

For all the examples, genetic algorithm as been stopped after 200 generations without any progress.

5.1 Ammonia synthesis loop

5.1.1 Process description

The gas interring into the plant (see figure 5.2), composed of nitrogen, hydrogen, argon and methane, is compressed to 280 bar by a two stages compressor with two intercoolers. A third compressor stage allows to compress the recycled gas. The gas is then heated before entering the reactor where the ammonia synthesis takes place. The reactor product must be cooled and partially condensed to recover the ammonia. This cooling takes place in four steps: the gaseous products first go through a waste heat boiler, then they heat the reactor feed, they are chilled in a water cooled exchanger, and finally in the evaporator of an ammonia refrigeration loop. The condensation product is later flashed to a lower pressure so that ammonia with a purity of 99.9 % can be obtained at the exit of the flash drum. The gaseous stream that exits the liquid-vapor separator is recycled to recover unreacted hydrogen. A part of this stream must however be purged to avoid accumulation of inerts. This process is composed of:

- 14 units: 1 two-stage compressor with two intercoolers, 1 ammonia converter, 1 recycle mixer, 1 recycle compressor, 1 reactor preheater, 1 waste heat boiler, 1 waste heat condenser, 1 purge divider, 1 liquid-vapor separator and 1 flash drum;
- 19 material streams composed of ammonia, argon, methane, hydrogen and nitrogen;
- 10 utility streams composed of steam and cooling water;
- 4 mechanical streams;
- 224 variables;
- 177 constraints equations.

Key parameters

The process comprises 52 key parameters to be monitored. They are listed with their prescribed standard deviations in the table below (Table 5.1).

Table 5.1: Ammonia synthesis loop: prescribed standard deviations on key parameters

Variable name	Standard deviation
Ammonia partial molar flow rate of stream AM1	5 %
Mass flow rate of stream BFW	2 %
Temperature of stream BFW	1 K
Pressure of stream BFW	0.5 bar
Mass flow rate of stream CW1	2 %
Temperature of stream CW1	1 K
Pressure of stream CW1	0.1 bar
Temperature of stream CWR1	1 K
Pressure of stream CWR1	1 %
Mass flow rate of stream CW2	2 %
Pressure of stream CWR2	1 %
Mass flow rate of stream CW3	2 %
Pressure of stream CWR3	1 %
Temperature of stream STM	1 K
Pressure of stream STM	0.5 bar

continued on next page

Variable name	Standard deviation
Mass flow rate of stream 1	5 %
Argon molar fraction of stream 1	0.001
Hydrogen molar fraction of stream 1	0.001
Methane molar fraction of stream 1	0.001
Nitrogen molar fraction of stream 1	0.001
Temperature of stream 1	1 K
Argon molar fraction of stream 6	0.002
Hydrogen molar fraction of stream 6	0.005
Ammonia molar fraction of stream 14	0.005
Argon molar fraction of stream 14	0.005
Hydrogen molar fraction of stream 14	0.005
Nitrogen molar fraction of stream 14	0.005
Mass flow rate of stream 15	1 %
Ammonia molar fraction of stream 18	0.005
Argon molar fraction of stream 14	0.005
Hydrogen molar fraction of stream 14	0.005
Nitrogen molar fraction of stream 14	0.005
Thermal power of heat intercooler E-101	2 %
Heat transfert coefficient of intercooler E-101	10 %
Thermal power of heat intercooler E-102	5 %
Heat transfert coefficient of intercooler E-102	15 %
Thermal power of heat intercooler E-103	2 %
Heat transfert coefficient of intercooler E-103	5 %
Thermal power of heat intercooler E-104	2 %
Heat transfert coefficient of intercooler E-104	5 %
Thermal power of heat intercooler E-105	5 %
Heat transfert coefficient of intercooler E-105	10 %
Thermal power of heat intercooler E-106	3 %
Heat transfert coefficient of intercooler E-106	10 %
Extent of the reaction at reactor R-101	1 %
Temperature deviation from equilibrium at reactor R-101	5 K
Mechanical power of compressor C-101	5 %
Efficiency of compressor C-101	0.02
Mechanical power of compressor C-102	5 %
Efficiency of compressor C-102	0.02
Mechanical power of compressor C-103	5 %
Global mechanical power WCOMP	5 %

Sensor database

The sensor database describes 15 sensor types:

Table 5.2: Ammonia synthesis loop: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values
Thermocouple A	10	0.5+0.1%	200 K	400 K
Thermocouple B	30	0.2+0.05%	375 K	800 K
Thermocouple C	50	0.2+0.1%	750 K	2000 K
Pressure Gauge A	20	1%	0 bar	20 bar
Pressure Gauge B	30	0.01+1%	15 bar	300 bar
Delta_p	0	0.001+1%	-0.5 bar	300 bar
Poly-Analyser	140	0.001+1%	0	1
Chromatograph A	400	0.0001+0.5%	0	1
Chromatograph B	200	0.001+1%	0	1
Molar Flowmeter	60	2%	0 mol/s	200 mol/s
Mass Flowmeter A	40	5%	0 kg/s	25 kg/s
Mass Flowmeter B	60	2%	20 kg/s	75 kg/s
Mass Flowmeter C	100	1%	50 kg/s	200 kg/s
Mass Flowmeter D	150	2%	175 kg/s	1000 kg/s

The initial solution that is obtained with this list of sensors costs 8600 cost units and counts 117 sensors:

- 14 chromatographs;
- 7 poly-analysers (One means by poly-analyser a measurement tool that is able to measure the concentration of several specified components in one stream. In this case, the poly-analyser will give the concentrations in ammonia, methane and argon);
- 30 mass flowmeters;
- 31 thermocouples;
- 29 pressure-gauges;
- 6 virtual pressure drop meters (Those sensors do not exists in reality. They are virtually created because Vali4 requires the specification of the units pressure drops).

This sensor network allows to identify all key parameters with acceptable accuracy.

5.1.2 Solution

Case of one operating point

When a sensor schedule is designed for a single operating point, a feasible sensor network is obtained after 76 seconds on a processor M dothan (1.6 GHz). This requires 361 generations and 7241 goal function evaluations for a population of 20 individuals. That corresponds to 100 goal function evaluations per second. This measurement system costs 1591 costs units and counts 45 sensors:

- 1 chromatograph;
- 7 mass flowmeters;
- 20 thermocouples;
- 11 pressure-gauges;
- 6 virtual pressure drop meters.

This sensor network is represented on figure 5.2. As it can be seen on figure 5.1, the solution is quickly reached: the optimal sensor network is already obtained after 160 generations. This solution is much better than the initial one but there is no guarantee that it is the absolute optimum. To be sure to find the best solution, all configurations should be tested. That corresponds to $2^{117} = 1.66\ 10^{35}$ configurations.

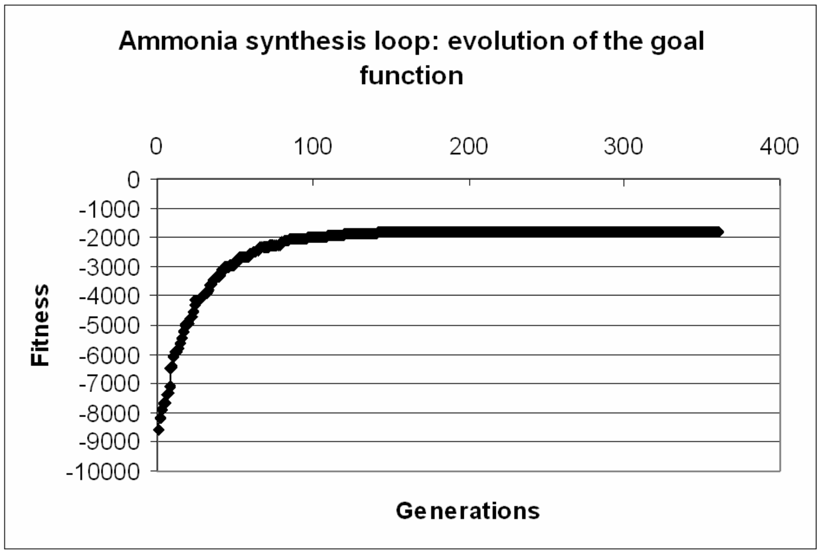


Figure 5.1: Ammonia synthesis loop: evolution of the goal function with the number of generations

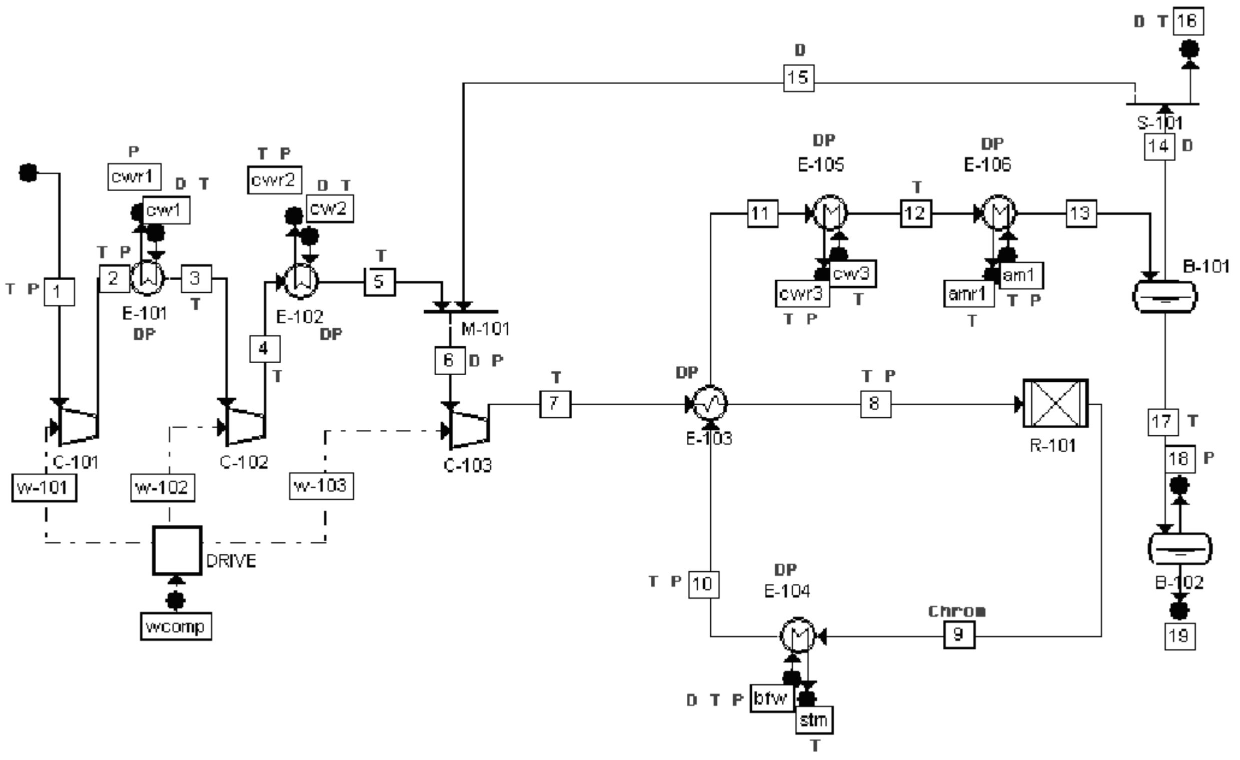


Figure 5.2: Ammonia synthesis loop: solution in the case of one operating point

Case of one sensor failure

When the goal is to design, a sensor network that remains feasible even in the case of one sensor failure, the solution is obtained after 1 hour and 45 minutes on a processor M dothan (1.6 GHz). That requires 407 generations and 487899 goal function evaluations for a population of 20 individuals. This measurement system costs 3900 cost units and counts 81 sensors:

- 2 chromatographs;
- 1 poly-analyser;
- 16 mass flowmeters;
- 1 molar flowmeter;
- 32 thermocouples;
- 17 pressure-gauges;
- 12 virtual pressure drop meters.

This sensor network is represented on figure 5.3.

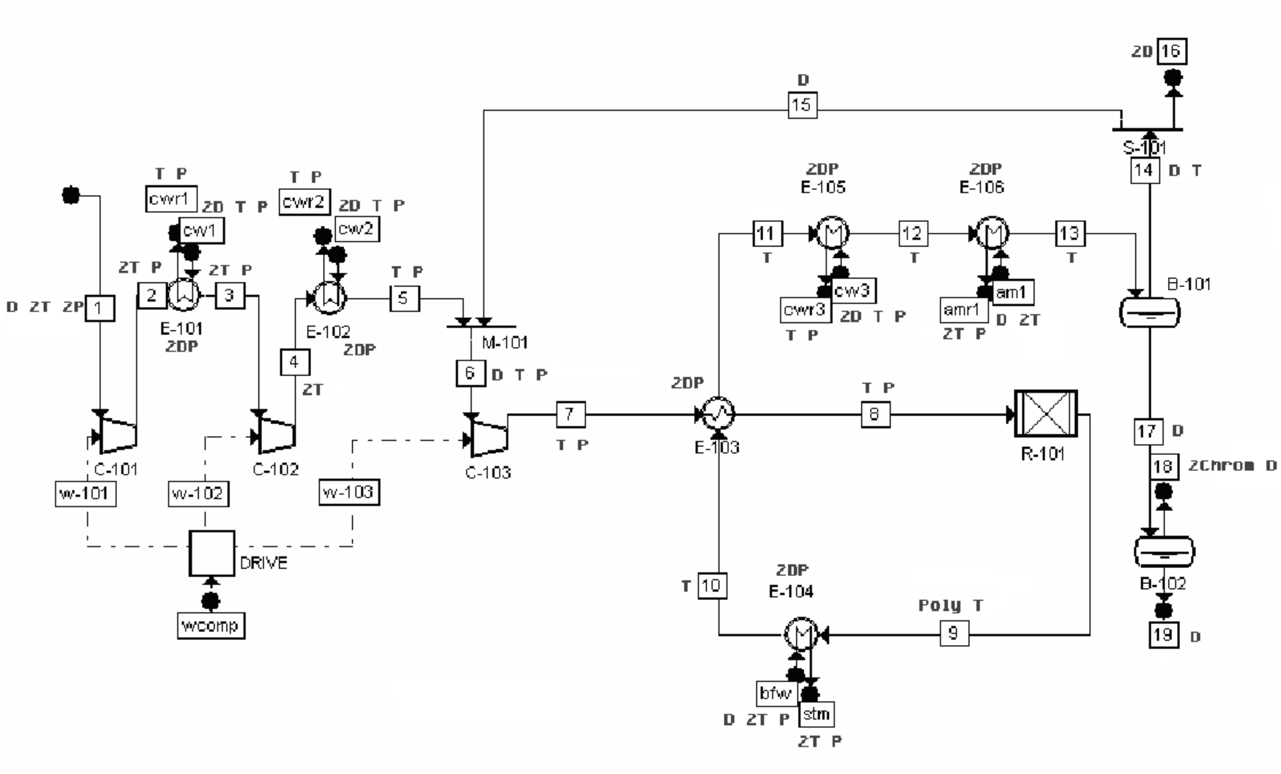


Figure 5.3: Ammonia synthesis loop: solution in the case of one sensor failure

5.1.3 Global parallelization

For the ammonia synthesis loop, parallelization results have been obtained thanks to the computer cluster of the Hemeris Society. This cluster is composed of 64 nodes (Apple Power Mac G4 bi-processors) interconnected by fast and giga Ethernet.

Case of one operating point

In the case of the obtention of a redundant sensor network, global parallelization has been applied to populations of respectively 20, 40 and 100 individuals. The search was stopped after 200 generations without progress in all cases. In the case of a population of 20 individuals, the results are presented on table 5.3 and on figure 5.4.

Table 5.3: Ammonia synthesis loop: global parallelization: 20 chromosomes: case of one operating point

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Efficiency %
2	278	300	100
4	140	164	91.4
5	111	133	90.2
10	57	71	84.5
20	30	45	66.7

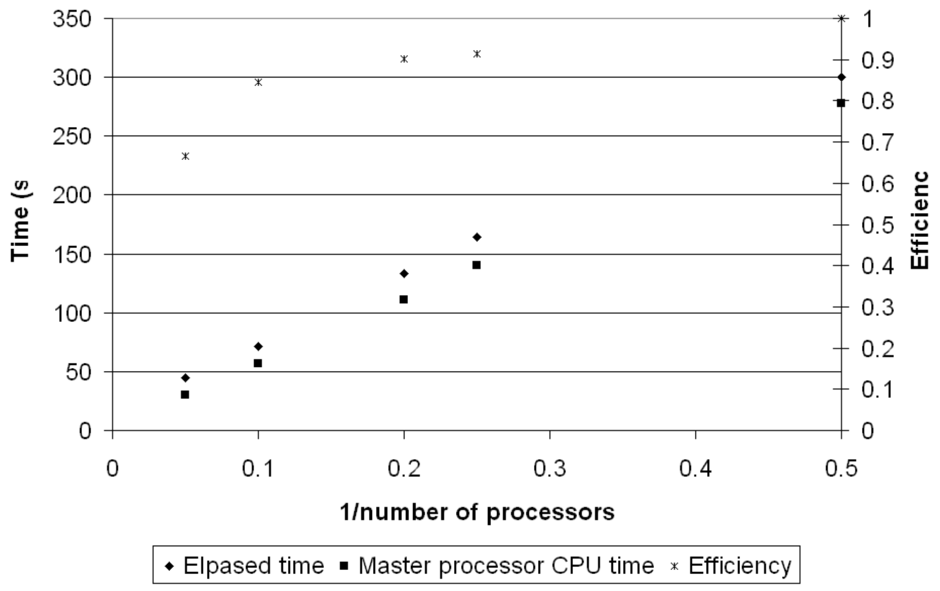


Figure 5.4: Ammonia synthesis loop: global parallelization: 20 chromosomes: case of one operating point

It can be seen that the elapsed time and the master processor CPU are linearly dependent on the inverse of the number of processors.
The efficiency has been calculated this way: it was not possible to obtain the elapsed time for a single processor on the Hemeris cluster, the reference time was taken equal to twice the reference time for two processors, so that the efficiency is equal to 100% for two processors. If the time for one processor had been available, all efficiencies would have been lower. It appears that the efficiency decreases with the number of processors. It decreases sharply when the number of processors is near or equal to the number of individuals. Indeed, in that case, as each processor evaluates one individual, processors that are in charge with an individual corresponding to a singular sensitivity matrix remain idle while the others are calculating a posteriori variances. As the number of individuals per processor increases, the individuals corresponding to singular matrices are better distributed between the different

processors. This loss of efficiency can not be predicted precisely a priori. Other losses of efficiency come from the connections between processors and from the sequential part of the program. In this case efficiency losses are less important than the one coming from the singular matrices, but their contribution to the total loss of efficiency increases when the number of processors decreases.
Results are presented on table 5.4 and on figure 5.5 in the case of a population of 40 individuals, and on table 5.5 and on figure 5.6 in the case of a population of 100 individuals. The same conclusions as in the case of a population of 20 chromosomes can be made. When the number of processors is equal to the number of individuals, it appears that the efficiency decreases more in the case of 40 chromosomes than in the case of 20, and in the case of 100 than in the case of 40.

Table 5.4: Ammonia synthesis loop: global parallelization: 40 chromosomes: case of one operating point

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Efficiency %
2	382	408	100
4	191	216	94.4
5	153	178	91.7
8	96	117	87.2
10	77	96	85
20	40	52	78.5
40	22	41	49.8

Table 5.5: Ammonia synthesis loop: global parallelization: 100 chromosomes: case of one operating point

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Efficiency %
2	1454	1506	100
4	723	790	95.3
5	580	643	93.7
10	291	345	87.7
20	147	183	82.3
25	118	155	77.7
50	63	108	55.8
100	41	86	35

Case of one sensor failure

When designing a measurement system that remains redundant in the case of one sensor failure, the global parallelization results are given on table 5.6 and on figure 5.7 for a

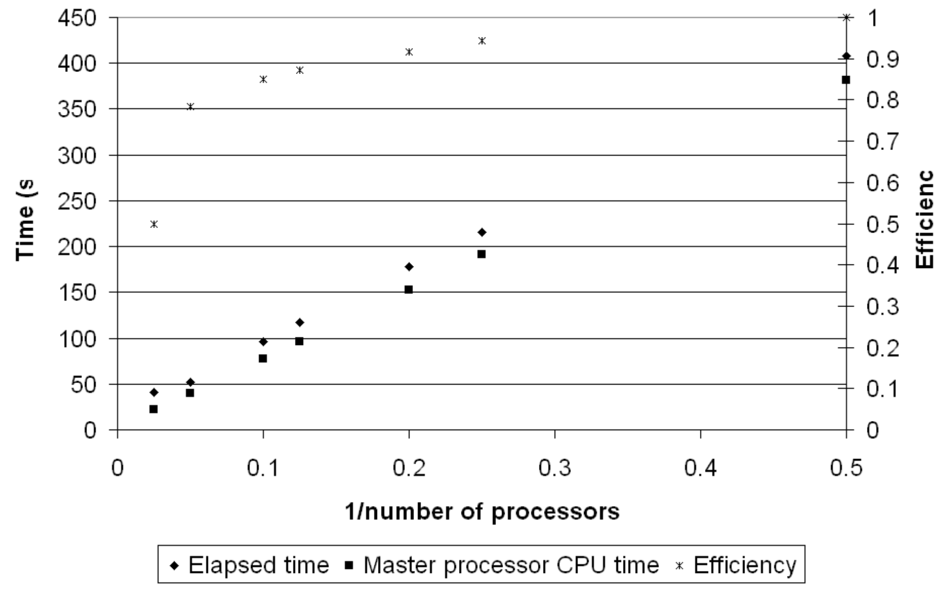


Figure 5.5: Ammonia synthesis loop: global parallelization: 40 chromosomes: case of one operating point

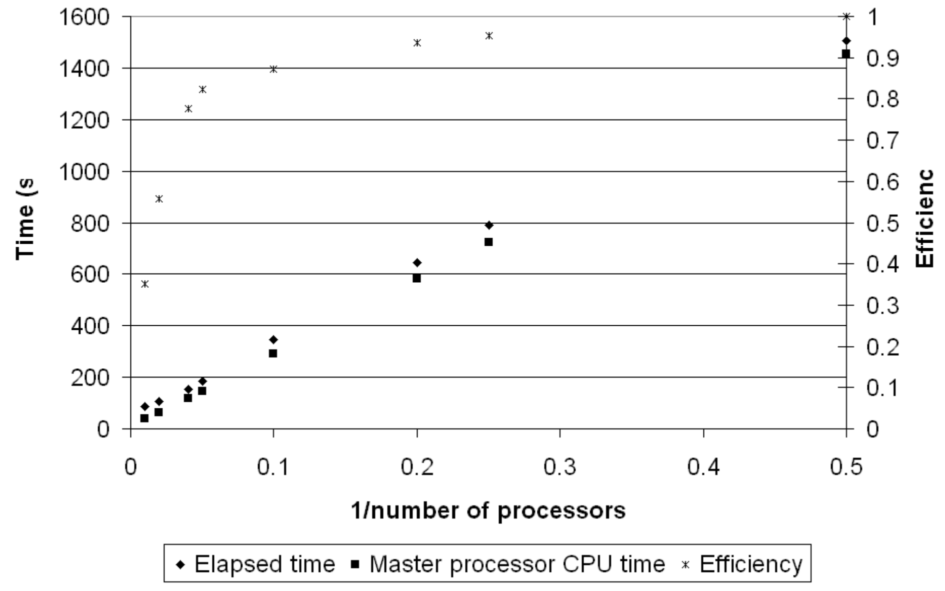


Figure 5.6: Ammonia synthesis loop: global parallelization: 100 chromosomes: case of one operating point

population of 20 chromosomes and a stop criterion after 200 generations without progress. It appears that the elapsed time and the master processor CPU are linearly dependent on the inverse of the number of processors.
The efficiency decreases with the number of processors, but more slowly than in the preceding cases. Indeed, if the number of processors is equal to the size of the population, each processor must carry out a number of goal function evaluations equal to the number of chosen sensors plus one, or one if the sensor network corresponds to a singular sensitivity matrix. So, the waiting times between the different processors are better distributed. The loss of efficiency increases more rapidly when the number of processors is higher than the number of individuals, namely, when the number of goal function evaluations per processor approaches one.

Table 5.6: Ammonia synthesis loop: global parallelization: 20 chromosomes: case of one sensor failure

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Efficiency %
2	14444	14642	100
4	7238	7541	97.1
5	5797	6108	95.9
10	2906	3208	91.3
20	1464	1730	84.6
40	814	1044	84.4
60	567	808	72.7
80	428	711	62

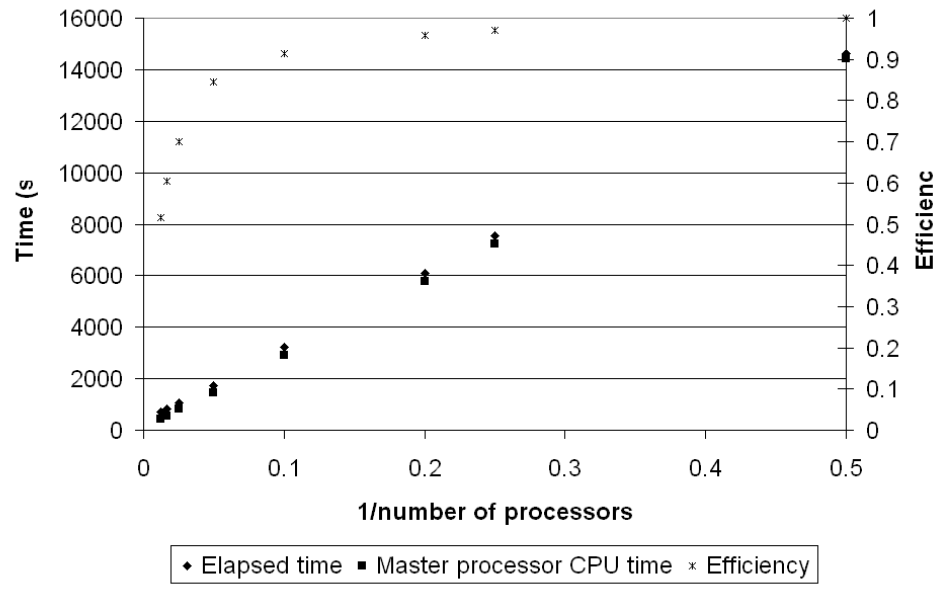


Figure 5.7: Ammonia synthesis loop: global parallelization: 20 chromosomes: case of one sensor failure

5.1.4 Distributed genetic algorithms

In this section, three parameters of the distributed genetic algorithms are discussed:

- the size of the sub-populations;
- the number of sub-populations;
- the number of generations between two migrations.

Two size of sub-populations have been studied (10 and 20 individuals) for different numbers of processors. The results are presented on table 5.7 and on figures 5.8 and 5.9 for the sub-populations of 10 chromosomes, and on table 5.8 and on figure 5.10 and 5.11 for the sub-populations of 20 chromosomes. Those results have been obtained for a number of generations between to migrations equal to 5, a number of migrating agents equal to 2 and a stop criterion equal to 40*5 generations.
The elapsed time and the master process CPU time evolve in the same way with respect to the number of sub-populations for the two sizes of sub-populations, but the goal function is not always the best one for the number of sub-populations corresponding to the higher number of generations. Indeed, as the sub-populations evolve independently from one another and start with different individuals, the way followed to find the solution may be very different from one case to another, so that the algorithm stops at different local minima. The solutions would approach each other if the stop criterion is increased, but there is no way to know a priori how to fix it. Indeed 100 may be enough in one case while 500 would not be enough in another one.

It does not seem to be interesting to take too many sub-populations. Indeed, if the number of sub-populations increases, the elapsed time does not always decreases. Moreover, the total work (elapsed time $*$ number of processors) generally increases with the number of processors. An increase of the number of sub-populations has always for impact a use of more computing ressources to obtain, after a more important work, a solution that is not always better than in the case of a smaller number of sub-populations. A number of sub-populations of five seems to be a good compromise.
The goal function values variate in the same intervalle following that sub-populations contains 10 or 20 chromosomes. The computing time being smaller for sub-populations of 10 chromosomes, it is favorable to take sub-populations of 10 chromosomes. If a too small size of sub-populations is taken, information contained in the initial individuals can be lost quite fast and the algorithm may converge quickly to a local minimum far away from the global minimum.

Table 5.7: Ammonia synthesis loop: distributed genetic algorithms: sub-populations of 10 chromosomes

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations
2	387	421	1521	765
3	211	226	1491	395
4	303	328	1611	550
5	238	258	1591	435
6	434	459	1541	825
7	504	551	1490	980
8	374	421	1551	720
9	345	385	1551	650
10	383	433	1470	745
12	257	305	1511	555
14	193	231	1470	360
16	371	450	1511	730
18	435	516	1470	850
20	265	332	1470	515

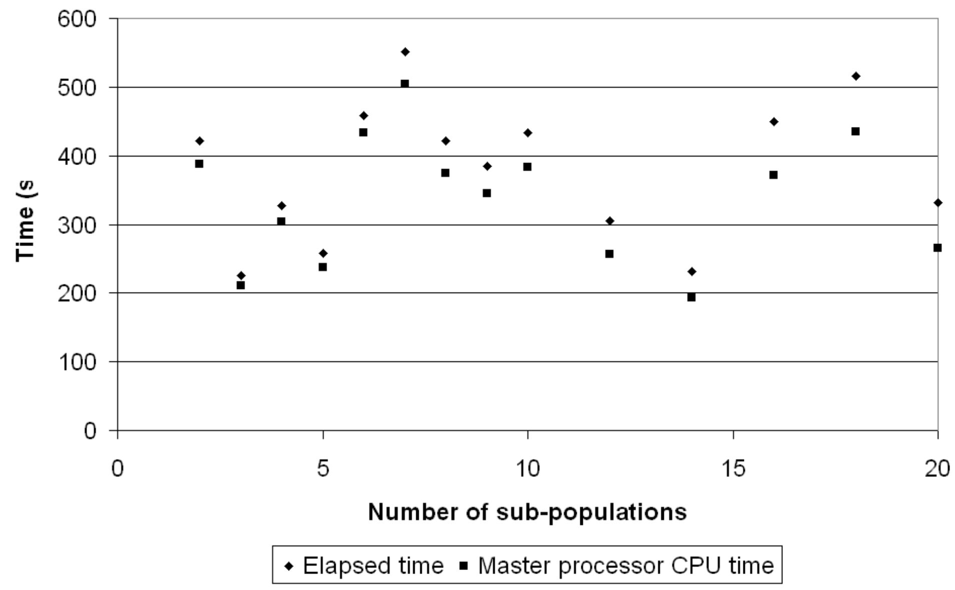


Figure 5.8: Ammonia synthesis loop: distributed genetic algorithms: influence of the number of 10 chromosomes sub-populations on the computing time: case of one operating point

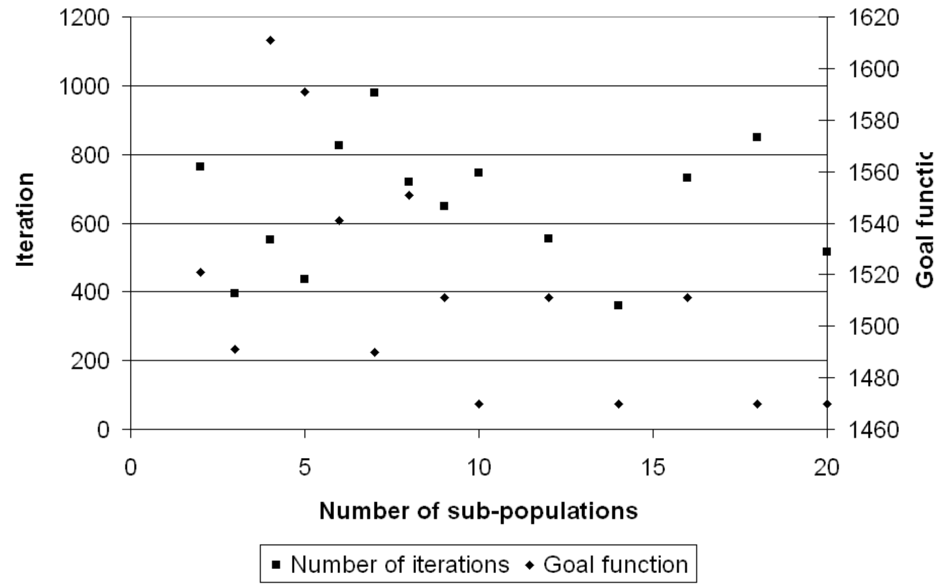


Figure 5.9: Ammonia synthesis loop: distributed genetic algorithms: influence of the number of 10 chromosomes sub-populations on the number of iterations and on the goal function: case of one operating point

Table 5.8: Ammonia synthesis loop: distributed genetic algorithms: sub-populations of 20 chromosomes

Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations
2	583	607	1501	545
3	496	521	1470	475
4	480	513	1551	440
5	680	716	1590	610
6	416	451	1471	390
7	656	710	1470	625
8	578	638	1511	555
9	454	528	1471	445
10	393	434	1540	345
12	898	1001	1511	850
14	574	650	1470	555
16	783	881	1470	770
18	490	564	1470	460
20	417	499	1470	395

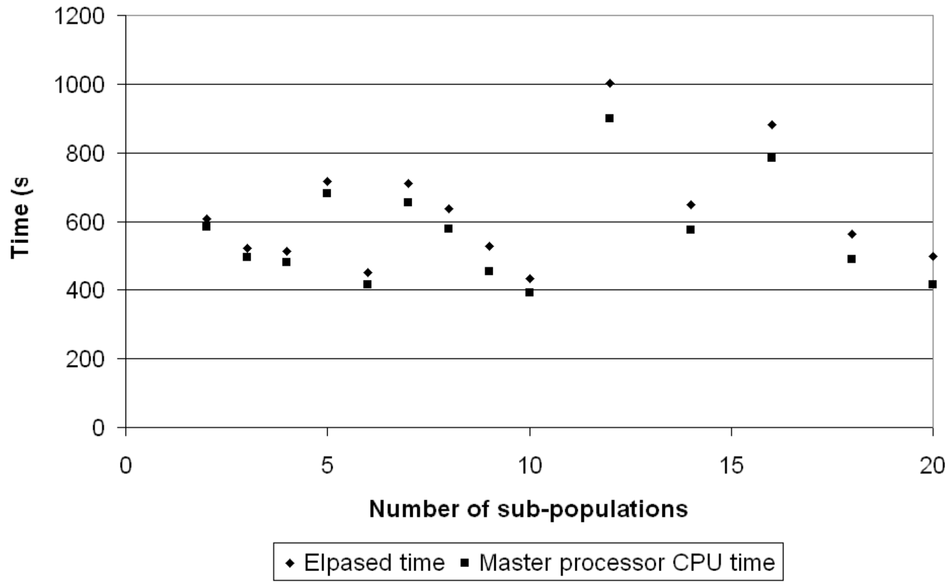


Figure 5.10: Ammonia synthesis loop: distributed genetic algorithms: influence of the number of 20 chromosomes sub-populations on the time: case of one operating point

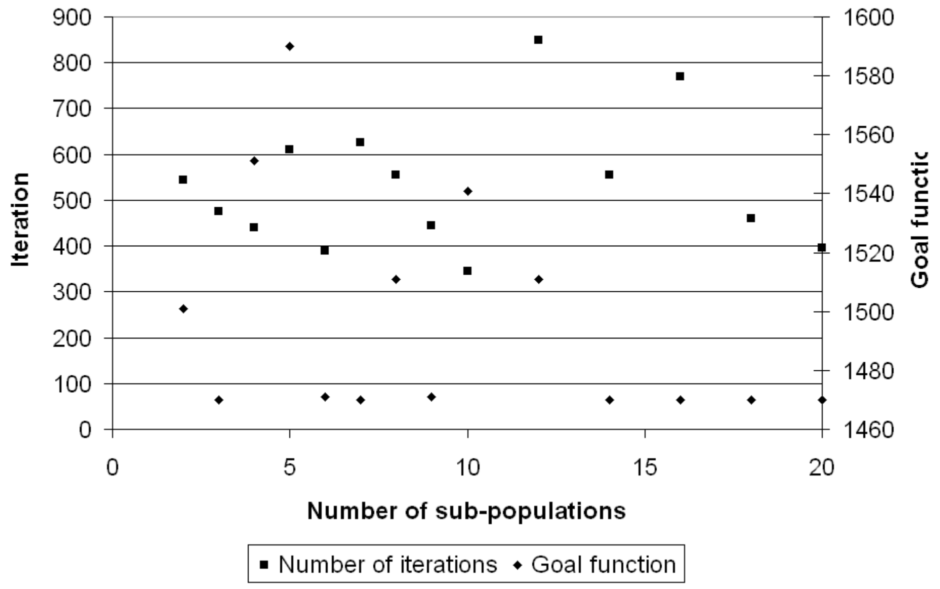


Figure 5.11: Ammonia synthesis loop: distributed genetic algorithms: influence of the number of 20 chromosomes sub-populations on the number of iterations and on the goal function: case of one operating point

To study the influence of the number of generations between two consecutive migrations, the size of the global population was fixed to 50, the size of the sub-populations to 10 and the stop criterion to 200. The results can be seen on table 5.9 and on figures 5.12 and 5.13.

Table 5.9: Ammonia synthesis loop: distributed genetic algorithms: influence of the number of generations between two migrations

Number of generations between 2 migrations	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations
2	221	226	1491	455
5	238	258	1591	435
10	289	322	1591	540
20	309	320	1541	560

The value of the goal function, the computing time and the total number of generations vary independently of the number of generations between two successive migrations. The value of five generations between two successive migrations found in the literature seems to be a good compromise.

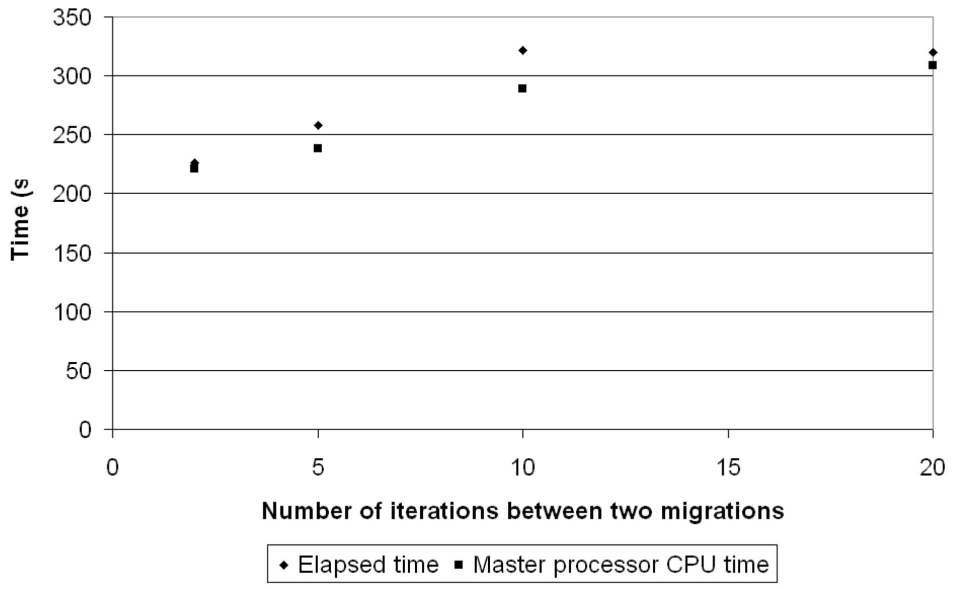


Figure 5.12: Ammonia synthesis loop: distributed genetic algorithms: influence on the computing time of the number of generations between 2 migrations: case of one operating point

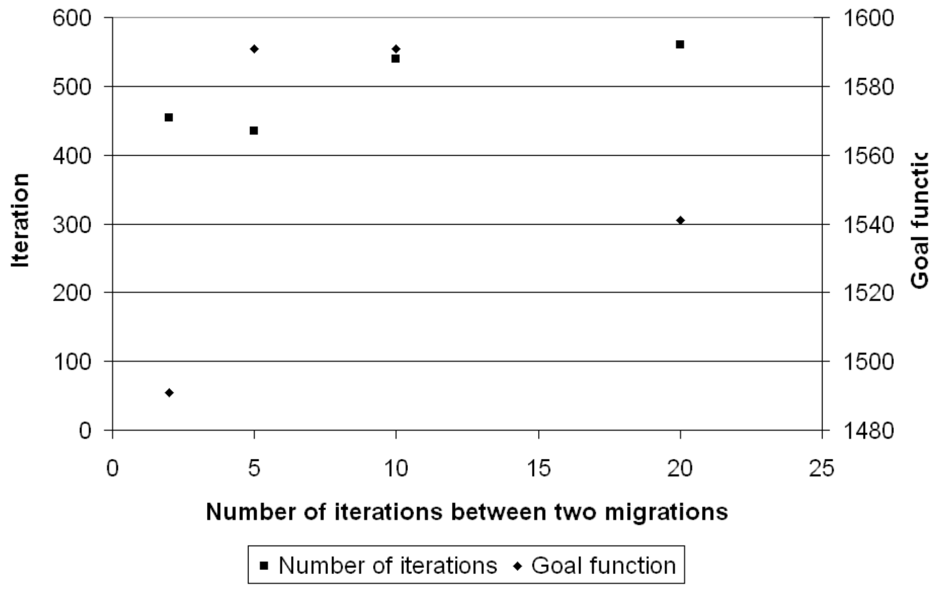


Figure 5.13: Ammonia synthesis loop: distributed genetic algorithms: influence on the number of iterations and on the goal function of the number of generations between 2 migrations: case of one operating point

5.1.5 Methods comparison

To decide which parallelization method is the most efficient, the times, the number of iterations, the final values of the goal function and the efficiencies have been compared. In all the cases, the global size of the population was fixed to 50 individuals and the stop criterion was taken equal to 200 for global parallelization and 40∗5 for distributed genetic algorithms. For distributed genetic algorithms, the number of generations between two successive migrations was fixed to 5 and the number of migrating individuals was taken equal to 2. The results are given in the tables 5.10 for global parallelization and 5.11 for distributed genetic algorithms.

Table 5.10: Ammonia synthesis loop: global parallelization

Global parallelization					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
2	766	798	1591	543	100
5	305	349	1591	543	91.5
10	154	199	1591	543	80.2

Table 5.11: Ammonia synthesis loop: distributed genetic algorithms

Distributed genetic algorithms					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
2	652	676	1471	466	118
5	238	258	1591	431	123.7
10	77	133	1470	381	120

It can be seen on figure 5.14 that the elapsed time and the master processor CPU time are better for the distributed genetic algorithms. The same tendency is observed in the case of the total number of generations (figure 5.16).
The goal function values are also better in the case of distributed genetic algorithms (figure 5.16). Indeed, they do not change with the number of processors in the case of global parallelization because the evolution of the population does not depend on that number in that case. In contrary, for distributed genetic algorithms, the global population evolution depends on the number of sub-populations: indeed,

- the size of the sub-populations depends on the number of sub-populations;
- individuals migrate from one sub-population to another one so that mating chromosomes is different in function of the way they move.

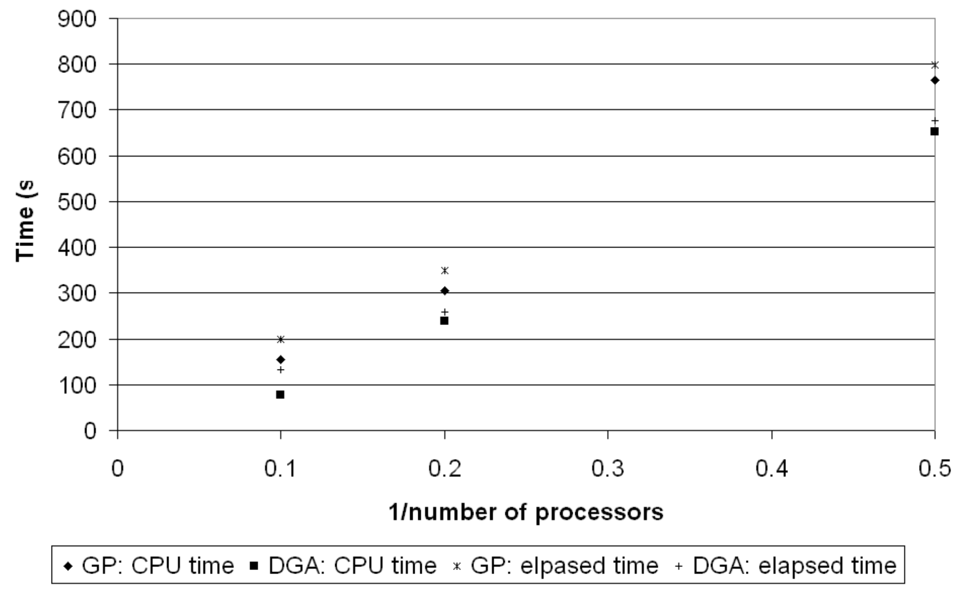


Figure 5.14: Ammonia synthesis loop: times comparison

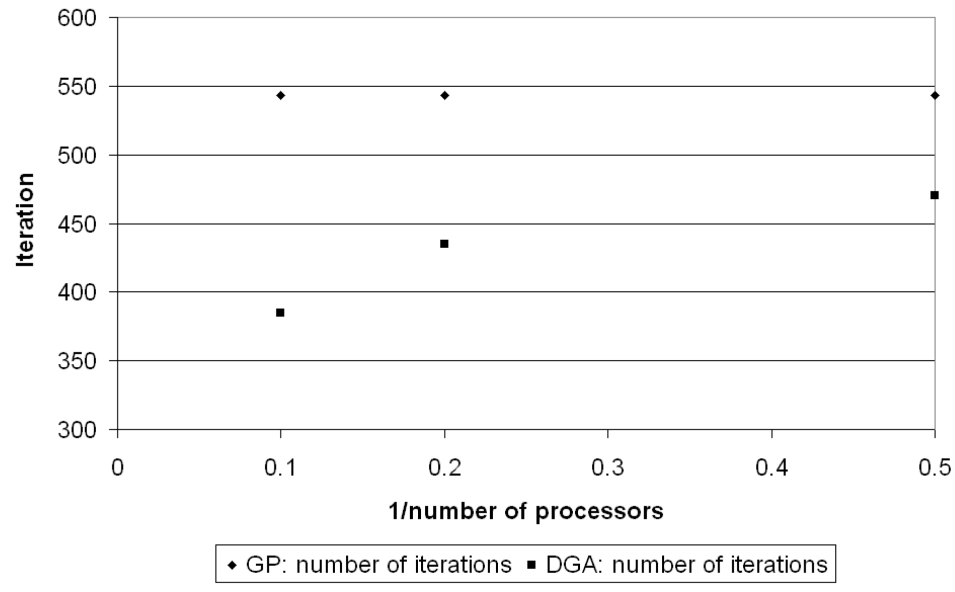


Figure 5.15: Ammonia synthesis loop: number of generations comparison

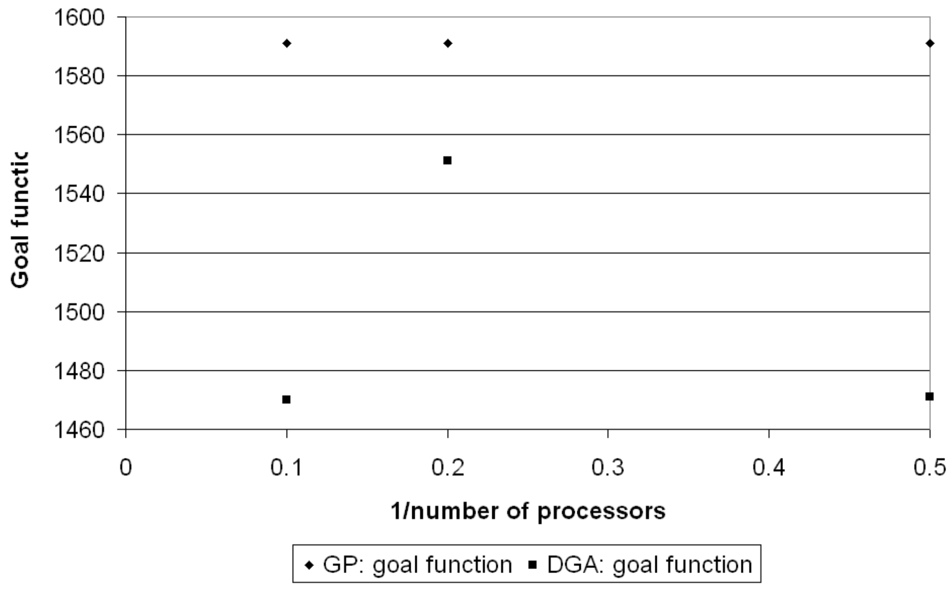


Figure 5.16: Ammonia synthesis loop: goal function comparison

Thus, the way followed to obtain the solution is different for each sub-population number, so that, as the optimal solution found by the algorithm may not be the best one, values of the goal function may be a little different.
It appears on figure 5.17 that the efficiency is much better in the case of distributed genetic algorithms. The efficiency is even better than 100% in some cases. Indeed, as sub-populations are subject to migrations and contain different initial individuals, the way followed by the algorithm to reach the solution is different. Moreover, the gain of time given by the reduction of the number of iterations is higher than the loss of time caused by the migration operations between sub-populations.
In the case of the ammonia synthesis loop, both parallelization methods allowed to reduce the computing time, but distributed genetic algorithms are better in all points of view.

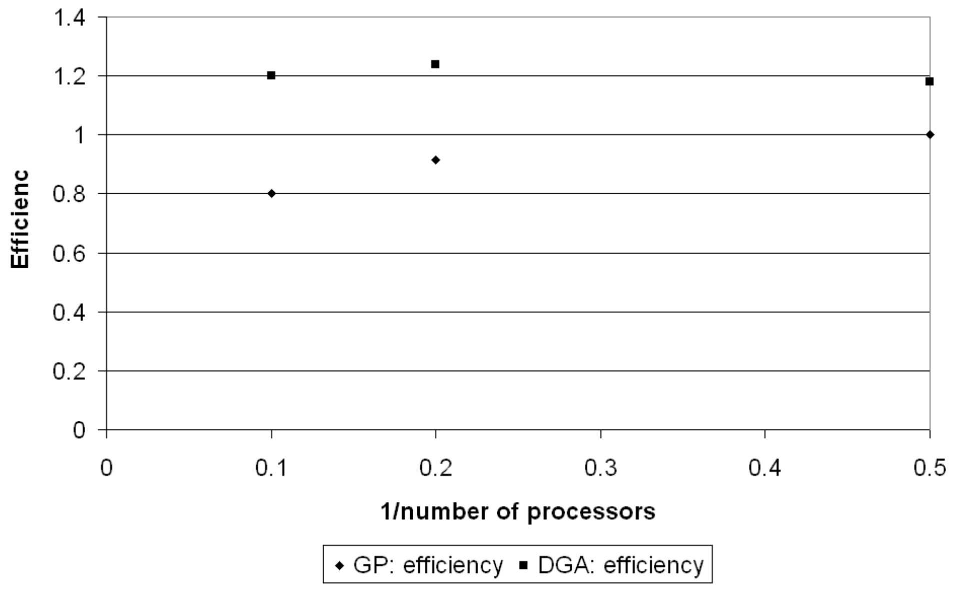


Figure 5.17: Ammonia synthesis loop: efficiency comparison

5.2 Combined cycle power plant

5.2.1 Process description

The second process is a combined cycle power plant. In this process, air is compressed before entering the combustion chamber where it reacts with the fuel (natural gas, modeled as a methane-ethane mixture). The combustion gas is later expanded in two successive turbines. Then it is cooled in a steam generator before being rejected to the environment. The generator allows to raise super heated steam at two pressure levels. Superheated steam feeds two turbines and is condensed before being recirculated to a boiler.
This process is composed of:

- 28 units: 1 air compressor, 1 combustion chamber, 2 turbines at the exit of the combustion chamber, 1 high pressure superheater, 1 high pressure vaporiser, 2 high pressure economizers, 1 low pressure superheater, 1 low pressure vaporiser, 1 low pressure economizer, 1 heater, 5 pumps, 1 high pressure turbine, 1 low pressure turbine, 1 condenser, 2 steam drums, 2 liquid-vapor separators, 1 stream divider, 2 stream mixers and 1 purge divider;
- 17 material streams composed of oxygen, water, nitrogen, methane, ethane and carbon dioxide;
- 28 utility streams composed of vapor and cooling water;
- 10 mechanical streams;
- 1 thermal stream;
- 260 variables;
- 216 constraints equations.

This example will be first studied in the case of the nominal operating point. Secondly, it will be studied considering two operating points simultaneously: the nominal conditions and the summer operating conditions. Performance of the parallelized algorithm will be analyzed for this later case.

Key parameters

The process comprises 10 key parameters which are listed with their prescribed standard deviations in the table below (Table 5.12).

Table 5.12: CC power plant: prescribed standard deviations on key parameters

Variable name	Standard deviation
Mechanical power of the air compressor	5 %
Efficiency of the air compressor	5 %
Mechanical power of the first gas turbine	5 %
Efficiency of the first gas turbine	5 %
Mechanical power of the second gas turbine	5 %
Efficiency of the second gas turbine	5 %
Mechanical power of the high pressure steam turbine	5 %
Efficiency of the high pressure steam turbine	2 %
Vapor fraction at the exit of the low pressure vaporizer	2 %
Vapor fraction at the exit of the high pressure vaporizer	2 %

Sensor database

The sensor database describes the following sensor types:

Table 5.13: CC power plant: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values
Thermocouple A	10	0.5+0.1%	0 K	250 K
Thermocouple B	10	0.5+0.1%	200 K	400 K
Thermocouple C	30	0.2+0.05%	375 K	800 K
Thermocouple D	50	0.2+0.1%	750 K	2000 K
Pressure Gauge A	20	1%	0 bar	20 bar
Pressure Gauge B	30	0.01+1%	15 bar	300 bar
Delta_p	0	0.001+1%	-0.5 bar	300 bar
O_2 Gauge	50	0.0005+0.5%	0	1
CO_2 Gauge	50	0.0005+0.5%	0	1
Poly-Analyser	140	0.001+1%	0	1
Chromatograph A	400	0.0001+0.5%	0	1
Chromatograph B	200	0.001+1%	0	1
Molar Flowmeter	60	2%	0 mol/s	200 mol/s
Mass Flowmeter A	40	5%	0 kg/s	25 kg/s
Mass Flowmeter B	60	2%	20 kg/s	75 kg/s
Mass Flowmeter C	100	1%	50 kg/s	200 kg/s
Mass Flowmeter D	150	2%	175 kg/s	1000 kg/s
Mass Flowmeter E	200	2%	1000 kg/s	5000 kg/s

The initial solution that is obtained with this list of sensors costs 14790 cost units and counts 206 sensors:

- 8 chromatographs;
- 3 oxygen gauges;
- 3 carbon dioxide gauges;
- 98 mass flowmeters;
- 48 thermocouples;
- 45 pressure-gauges;
- 1 virtual pressure drop meters.

This sensor network allows to satisfy all accuracies on key parameters. In this example, the solution space counts $2^{206} = 1.03 * 10^{62}$ different solutions.

5.2.2 Solution

Case of one operating point

In the case of one operating point, a feasible sensor network is obtained after 587 seconds on a processor Celeron (2.2 GHz) for a population of 48 individuals. This requires 933 generations and 44785 goal function evaluations. That corresponds to 76 goal function evaluations per second. This measurement system costs 1820 cost units and counts 49 sensors:

- 2 oxygen gauges (one for air and one for combustion gas);
- 2 carbon dioxide gauges (one for fuel and one for combustion gas);
- 9 mass flowmeters;
- 24 thermocouples;
- 11 pressure-gauges;
- 1 virtual pressure drop meter.

This sensor network is represented on figure 5.18.

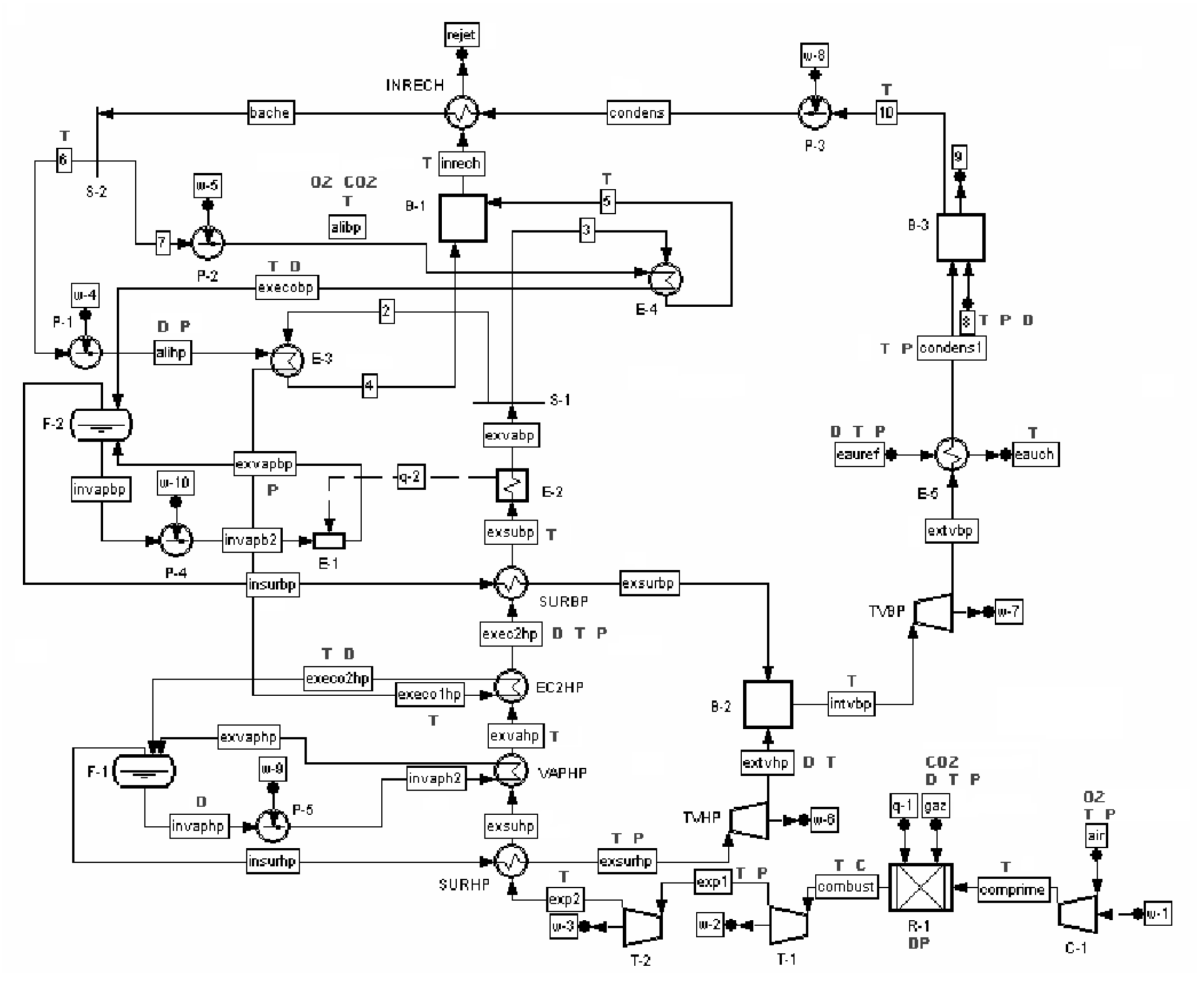


Figure 5.18: CC power plant: solution in the case of one operating point

Case of two operating points

In the case of two operating points, a feasible sensor network is obtained after 1595 seconds on a processor Celeron (2.2 GHz) for a population of 48 individuals. This requires 1343 generations and 128930 goal function evaluations. This measurement system costs 1820 cost units and counts 44 sensors:

- 1 chromatograph for the combustion gas (replacing one oxygen and one carbon dioxide gauges);
- 1 oxygen sonde for the air;
- 1 carbon dioxide sonde for the fuel;
- 7 mass flowmeters;
- 23 thermocouples;
- 10 pressure-gauges;
- 1 virtual pressure drop meter.

This sensor network is represented on figure 5.19.

Case of one sensor failure

In the case of one operating point, an observable sensor network in the case of one sensor failure is obtained after 486 minutes on a processor Celeron (2.2 GHz) for a population of 48 individuals. That requires 823 generations and 3139426 goal function evaluations. This measurement system costs 3840 cost units and counts 78 sensors:

- 2 chromatographs;
- 2 oxygen gauges (one for the air and one for the combustion gas);
- 16 mass flowmeters;
- 37 thermocouples;
- 19 pressure-gauges;
- 2 virtual pressure drop meters.

This sensor network is represented on figure 5.20.

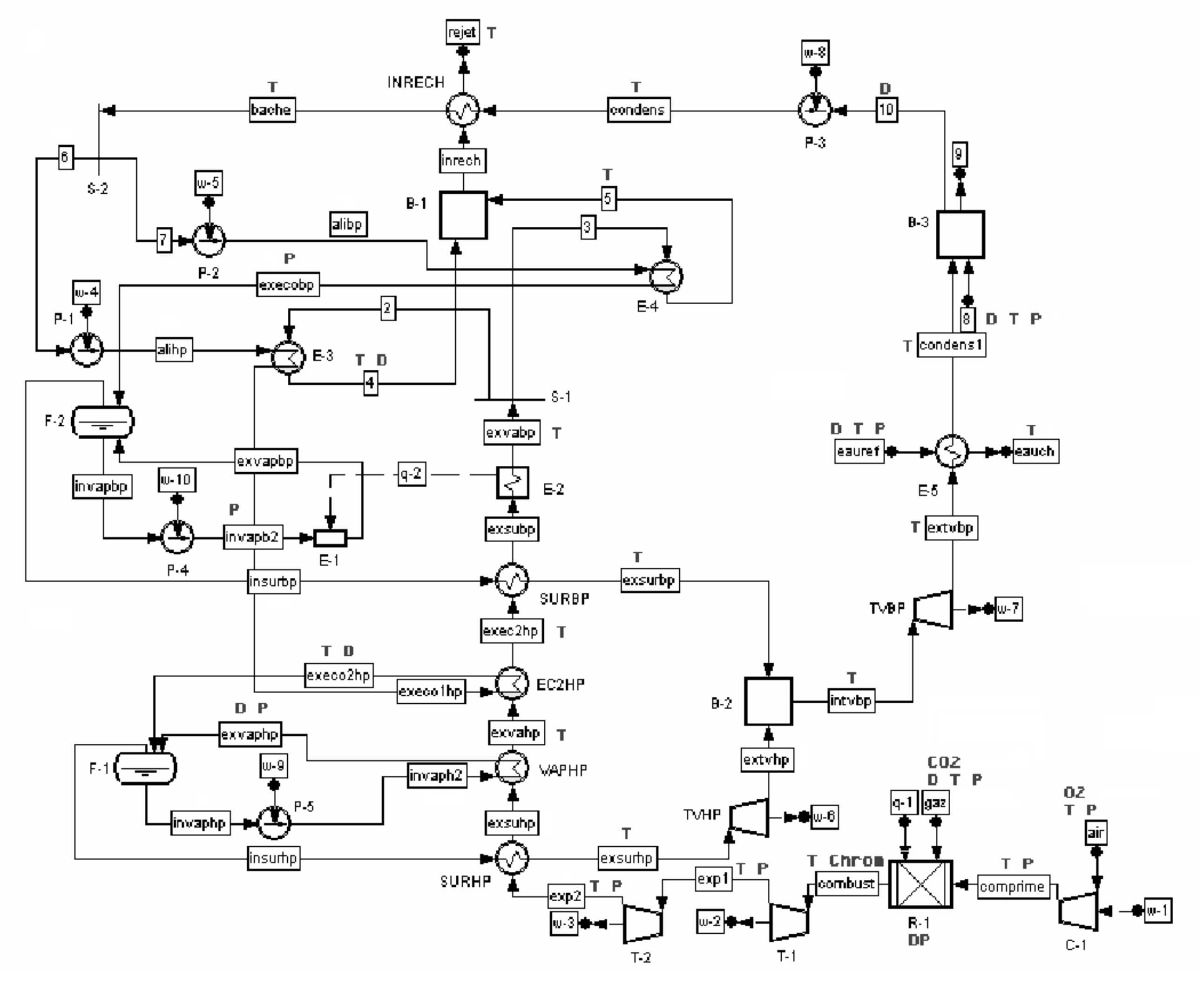


Figure 5.19: CC power plant: solution in the case of two operating points

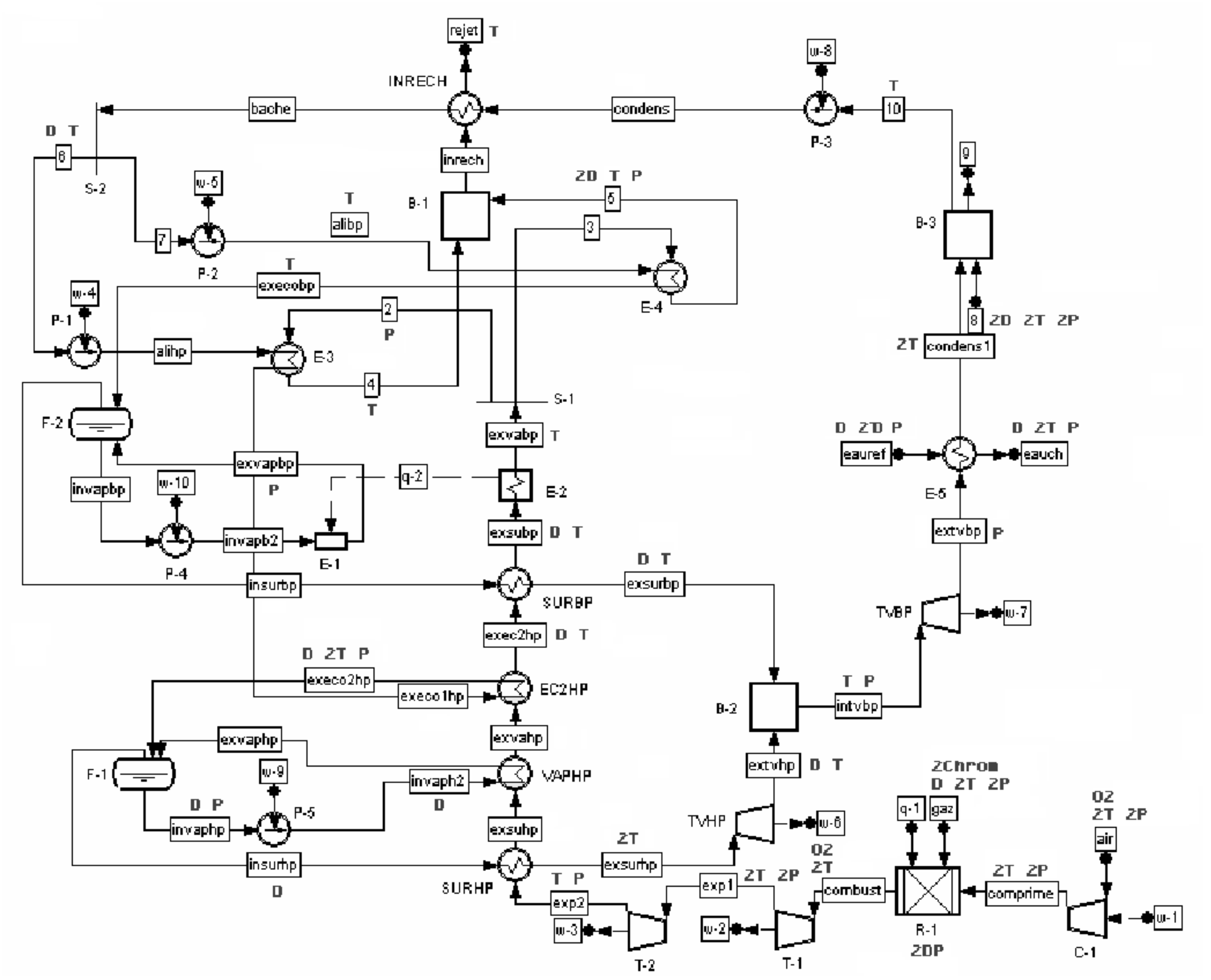


Figure 5.20: CC power plant: solution in the case of one sensor failure

5.2.3 Parallelization: methods comparison

For this example and the two following ones, the parallelization has been carried out by running MPI services on the computer network available for students in the chemical engineering department. This is composed of 12 computers: 8 have Celeron 2.2 GHz processors and 4 have Pentium 4.3 GHz processors. They are linked by a 100 Mps ethernet network to a switch. For a number of processors until 8, the work was shared between the 8 Celeron processors, the other 4 being used only for a number of processors equal to twelve. As those 4 computers work faster, their use has not introduced loss of elapsed time. The times obtained can not be reproduced exactly: indeed, as the computers are connected to the University network and not to a dedicated hub, the connection times can increase when the network is used by other persons. Moreover, some Windows services or updates runs at some times during the day time and make the processor work more slowly than at another time. For those reasons, only the best times have been kept in the analysis. To obtained less biased times, the computers should be connected to each other only and contain only the logiciels that are necessary for the application.
In all the cases (except 8∗ and 12∗), the global size of the population was fixed to 48 individuals and the stop criterion was taken equal to 200 for global parallelization and 40∗5 for distributed genetic algorithms. For distributed genetic algorithms, the number of generations between two successive migrations was fixed to 5 and the number of migrating individuals was taken equal to 2. The results are given in the tables 5.14 for global parallelization and 5.15 for distributed genetic algorithms.

Table 5.14: CC power plant: global parallelization

Global parallelization					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
1	1589	1595	1807	1343	100
2	760	819	1807	1343	97.4
3	512	569	1807	1343	93.4
4	387	439	1807	1343	90.8
6	257	331	1807	1343	80.3
8	197	252	1807	1343	79.1
12	134	173	1807	1343	76.8

Table 5.15: CC power plant: distributed genetic algorithms

Distributed genetic algorithms					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
1	1589	1595	1807	1343	100
2	446	468	1997	881	170.4
3	177	236	1848	481	225.3
4	175	202	1898	731	197.4
6	109	127	1907	601	209.3
8	144	173	2218	1161	115.2
8∗	157	201	1908	421	99.2
12	67	85	2087	761	156.4
12∗	112	153	1847	561	86.9

As it can be seen on figure 5.21 both parallelization techniques allows for this example to reduce the computing time which remains inversely proportional to the number of processors in the case of global parallelization. The elapsed times and the master processor CPU times are better for distributed genetic algorithms. The solution is reached after less generations for distributed genetic algorithms (figure 5.22). We note nevertheless that the total number of generations is different for each number of sub-populations, that is due to the fact that, as already mentioned for the first case study, sub-populations evolve independently from one another so that the way followed by the global population is different in each case. This phenomemon explains the difference in the total number of iterations. Those different ways followed by the populations can also explain the different local optima reached for a same stop criterion (figure 5.23).

The goal function values are for this example better in the case of global parallelization (figure 5.23). For distributed genetic algorithms, when the number of processors increases, the size of the sub-populations decreases and is equal to 4 individuals in the case of 12 processors, and 6 for 8 processors. In those cases, the size of sub-populations became too small and the algorithm converges to quickly to a local minimum probably by lack of diversity in the genotype of the individuals. That explains that high value of goal function and the so small elapsed times and numbers of generations. To avoid this local convergence, for cases 8∗ and 12∗ the global population was multiplied by 2 being equal to 96 and the stop criterion was divided by 2 becoming equal to 20∗5 generations so that the number of fitness evaluations remains the same but the sub-populations contains more individuals: 8 instead of 4 for 12 processors and 12 instead of 6 for 8 processors. In can be observed that the goal function values are better for the two ∗ cases but the computing times are bigger. The numbers of generations are smaller for the ∗ cases but each of them corresponds to a number of goal function evaluations twice bigger.

In the case of global parallelization, the efficiency decreases with the number of processors (5.24). It is much better in the case of distributed genetic algorithms, but this better efficiency does not always correspond to optimal values of the goal function. The efficiency

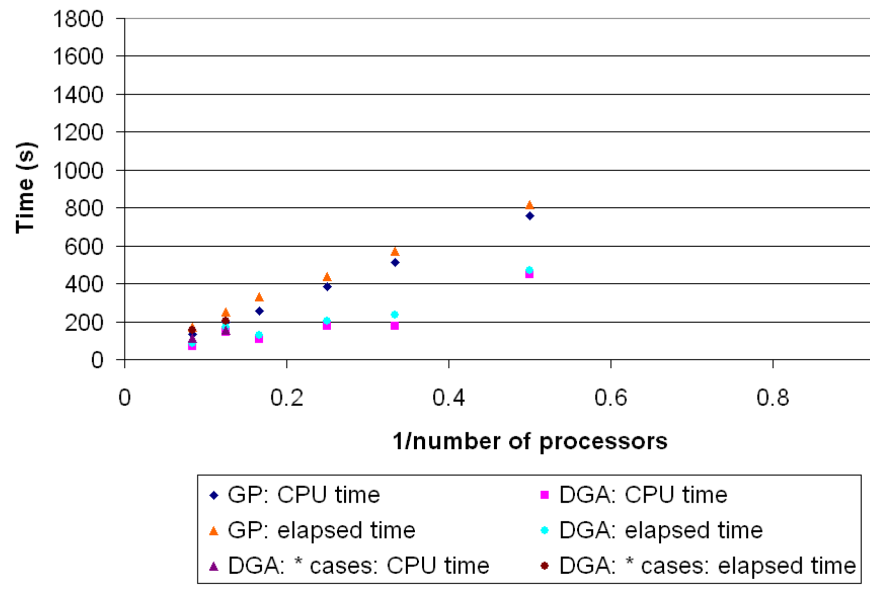


Figure 5.21: CC power plant: times comparison

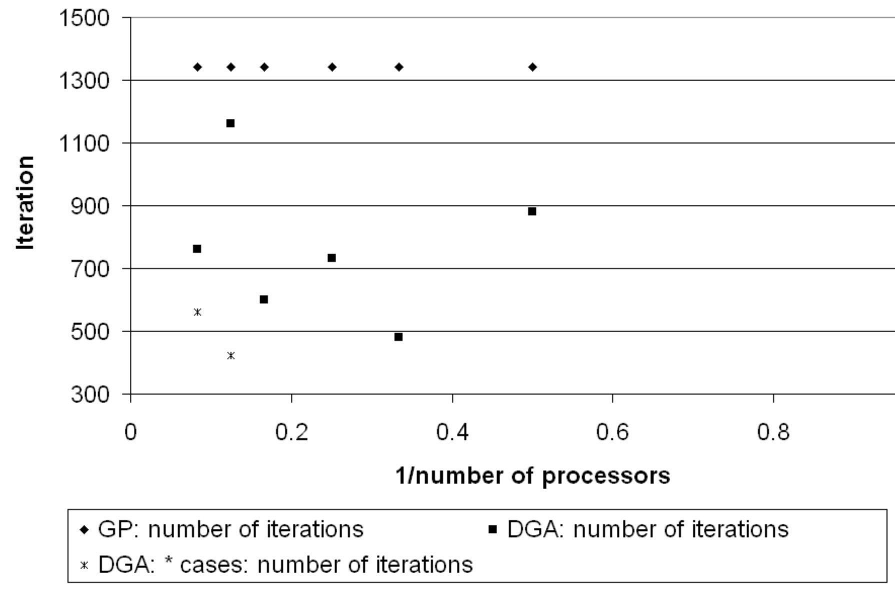


Figure 5.22: CC power plant: number of generations comparison

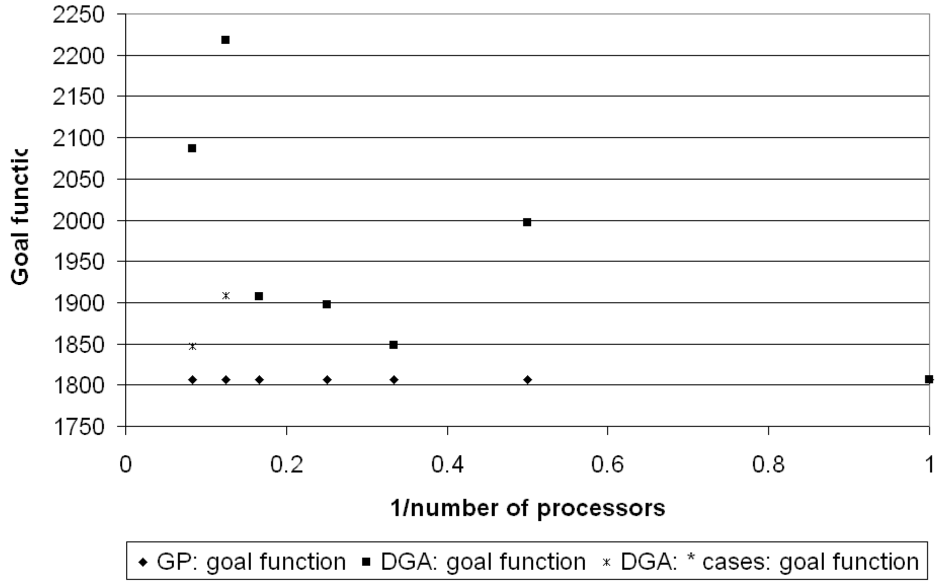


Figure 5.23: CC power plant: goal function comparison

is smaller for the two ∗ cases but the solution is much better. So, it seems preferable to increase the size of the sub-populations and decrease the stop criterion when the number of individuals per sub-populations becomes to small.
For this example, distributed genetic algorithms are better than global parallelization concerning the computing times, but give worse goal function values. The ∗ cases are situated in between, their computing times is higher than for the standard cases but smaller than for global parallelization, and their goal function is better than for the standard cases but worse than for global parallelization. They seem to be a good compromise between the value of the goal function and the computing time.

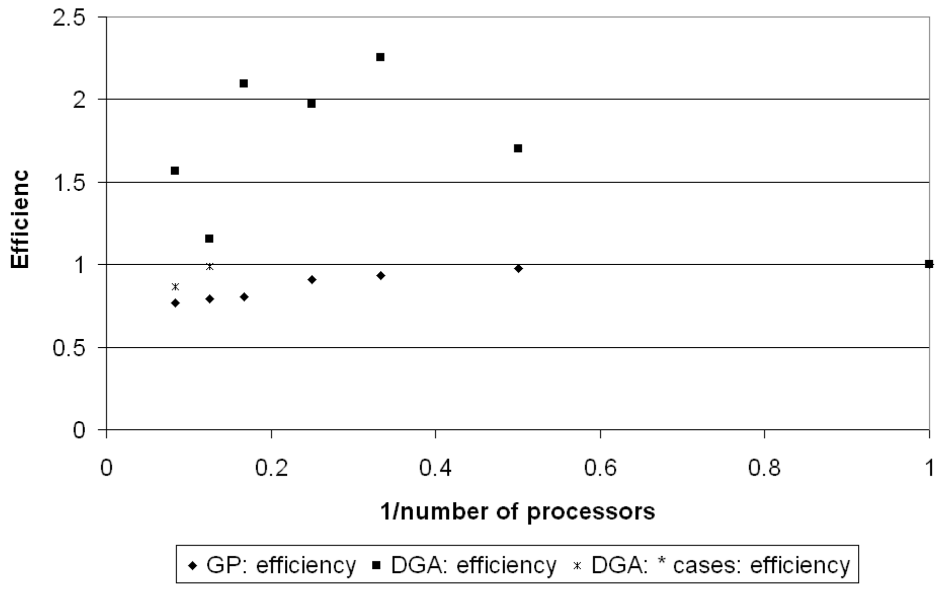


Figure 5.24: CC power plant: efficiency comparison

5.3 Ketene cracker

5.3.1 Process description

The third example is a ketene cracker. The plant model has been provided by Belsim company. It has been developed for Wacker Chemie and represent the ketene plant on their Burghausen site.
In this process, acetic acid is evaporated in a preheater before it enters into a furnace where the catalytic pyrolysis occurs. The catalyst, triethyl phosphate (TEP), allows the cracking of acetic acid into ketene and water. TEP is first transformed into phosphoric acid, which loses a water molecule. Acetic acid is then added to meta phosphoric acid and, finally, acetic anhydride is rearranged into ketene and phosphoric acid. TEP is used as catalyst instead of phosphoric acid because if phosphoric acid is directly injected in the furnace, it reacts with itself to form a dimer. If TEP is injected at the right place, this dimerisation does not occur.
The furnace is heated so that the outlet gas reaches a temperature of approximately $700°C$. The exhaust gas is cooled rapidly in two chillers (a water one and a brine one). The liquid phase (containing mainly water and unconverted acetic acid) is separated from the vapor phase (containing mainly ketene) to avoid backward reaction. This vapor phase is further purified in distillation columns. Although the pyrolysis reaction is catalyzed, parallel reactions occur and ketene contains impurities such as ethene, methane, carbon monoxide, carbon dioxide... The ketene contained in the liquid phase is then transformed into acetic anhydride, itself transformed into acetic acid that will be reused in the plant.
This process is composed of:

- 23 units: 1 compressor, 1 ketene cracker, 4 reactors, 7 heat exchangers, 5 liquid-vapor separators, 4 stream mixers and 1 pump;
- 34 material streams composed of keten, triethylphosphate, phosphoric acid, acetic acid, acetic anhydride, water, carbon dioxide, carbon monoxide, oxygen, nitrogen, methane, ethane, ethene, propadiene, propylène, normal-butane, ammonium phosphate and ammonia;
- 12 utility streams composed of vapor and water;
- 2 mechanical streams;
- 4 thermal streams;
- 631 variables;
- 557 constraints equations.

Key parameters

The process comprises 5 key parameters which are listed with their prescribed standard deviations in the table below (Table 5.16).

Table 5.16: Ketene cracker: prescribed standard deviations on key parameters

Variable name	Standard deviation
Acetic acid conversion at the exit of the cracker	1 %
Selectivity at the exit of the cracker	1 %
Acetic acid conversion after quench	1 %
Selectivity after quench	1.5 %
Global yield	1.5 %

Sensor database

The sensor database describes the following sensor types:

Table 5.17: Ketene cracker: sensor database

Sensor types	Annualized costs (€)	Accuracies	Minimum values	Maximum values
Thermocouple A	2000	3 K	200 K	400 K
Thermocouple B	2000	3 K	375 K	800 K
Thermocouple C	2000	3 K	750 K	1500 K
Pressure Gauge A	2000	2%	0 bar	2.5 bar
Pressure Gauge B	3000	2%	0 bar	20 bar
Pressure Gauge C	3000	2%	15 bar	300 bar
Delta_p	0	0.001+1%	0 bar	200 bar
O_2 Gauge	3000	0.0005+0.5%	0	1
CO_2 Gauge	3000	0.0005+0.5%	0	1
CO Gauge	3000	0.0005+0.5%	0	1
Poly-Analyser UV	15000	0.001+1%	0	1
Chromatograph	50000	0.001+0.5%	0	1
Molar Flowmeter	2000	3%	0 mol/s	2000 mol/s
Mass Flowmeter A	2000	3%	0 kg/s	25 kg/s
Mass Flowmeter B	2000	3%	20 kg/s	75 kg/s
Mass Flowmeter C	2000	3%	50 kg/s	200 kg/s
Density meter	2000	0.05+1%	0	2000
Cold density meter	2000	0.005+1%	0	2
Pitot tube	6000	3%	0.1 m^3/s	10 m^3/s

The initial solution that is obtained with this list of sensors costs 201800 € and counts 276 sensors:

- 27 chromatographs;
- 11 poly-analysers (measurement tool able to give the concentration in acetic acid, keten and acetic anhydride);
- 1 oxygen gauge;
- 1 carbon monoxide gauge;
- 88 mass flowmeters;
- 8 molar flowmeters;
- 47 thermocouples;
- 78 pressure-gauges;
- 1 Pitot tube;
- 1 density meters;
- 2 cold density meters;
- 8 virtual pressure drop meters.

This sensor network allows to satisfy all accuracies on key parameters. In this example, the solution space counts $2^{276} = 1.21 * 10^{83}$ different solutions.

In the case of the ketene cracker, four sensors have to be placed in the plant at the operator's request:

- 1 Pitot tube on stream keten2;
- 1 density meter on stream dunnsr6;
- 1 cold density meter on stream dunnsrd;
- 1 cold density meter on stream 3essig1.

5.3.2 Solution

Case of one operating point

In the case of one operating point, a feasible sensor network is obtained after 9 hours 40 minutes on a processor Celeron (2.2 GHz) for a population of 48 individuals. This requires 1550 generations and 74401 goal function evaluations. That corresponds to 124 goal function evaluations per minute. This measurement system costs 168000 € and counts 65 sensors:

- 1 chromatograph;
- 1 carbon dioxide gauge;
- 13 mass flowmeters;
- 1 molar flowmeter;
- 25 thermocouples;
- 11 pressure-gauges;
- 1 Pitot tube;
- 1 density meter;
- 2 cold density meters;
- 8 virtual pressure drop meters.

This sensor network is represented on figures 5.25 and 5.26.

Case of one sensor failure

In the case of one operating point, a feasible sensor network in the case of one sensor failure is obtained after 61 hours on computer network composed of four processors Pentium4 (3 GHz) for a population of 48 individuals. That requires 1281 generations and 4 242 413 goal function evaluations. This measurement system costs 260000 € and counts 81 sensors:

- 2 chromatographs;
- 1 carbon dioxide gauge;
- 12 mass flowmeters;
- 20 thermocouples;
- 35 pressure-gauges;
- 1 Pitot tube;
- 1 density meter;
- 2 cold density meters;
- 14 virtual pressure drop meters.

This sensor network is represented on figures 5.27 and 5.28.

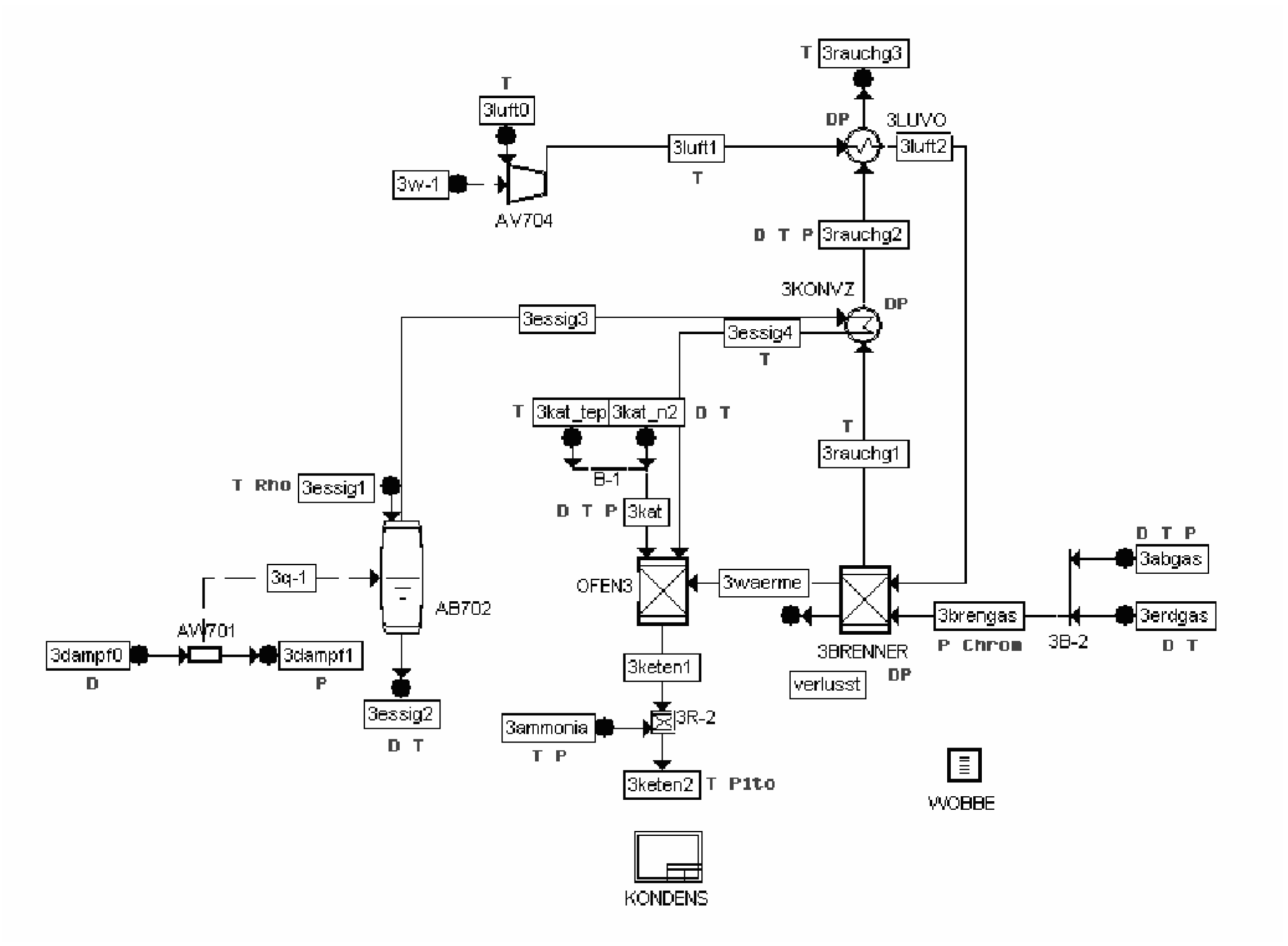


Figure 5.25: Ketene cracker: solution in the case of one operating point (part 1)

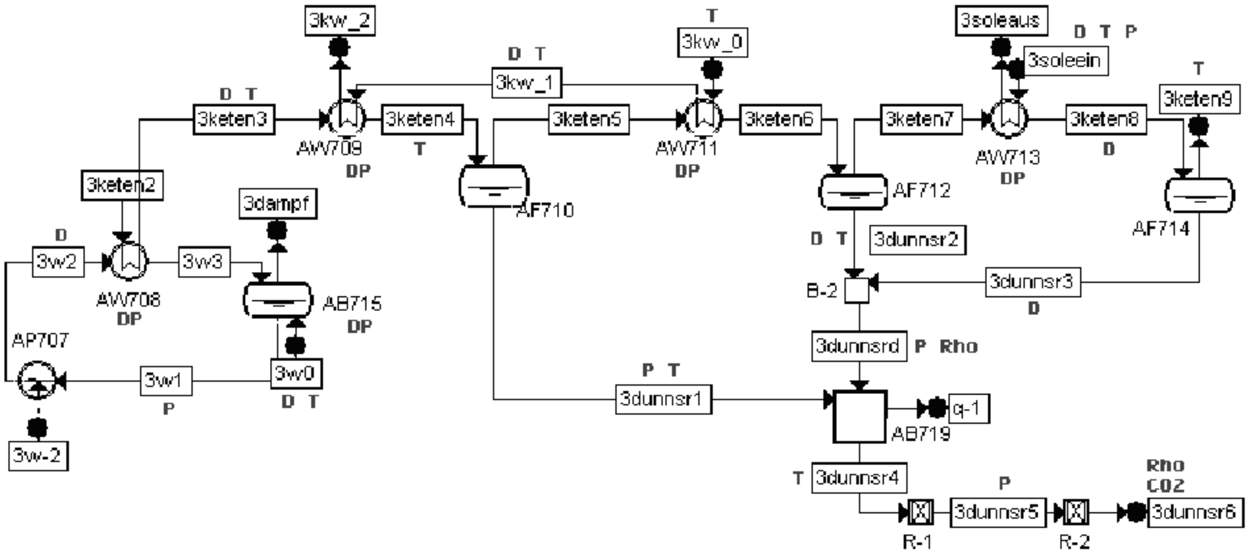


Figure 5.26: Ketene cracker: solution in the case of one operating point (part2)

5.3.3 Parallelization: methods comparison

In all the cases, the global size of the population was fixed to 48 individuals and the stop criterion was taken equal to 200 for global parallelization and 40∗5 for distributed genetic algorithms. Calculation were run on the 12 computers of the student network. As for the combined cycle power plant, distributed genetic algorithms have also been tested for a total population of 96 individuals and a stop criterion equal to 20∗5 generations (for 8 and 12 processors). For distributed genetic algorithms, the number of generations between two successive migrations was fixed to 5 and the number of migrating individuals was taken equal to 2. The results are given in the tables 5.18 for global parallelization and 5.19 for distributed genetic algorithms.

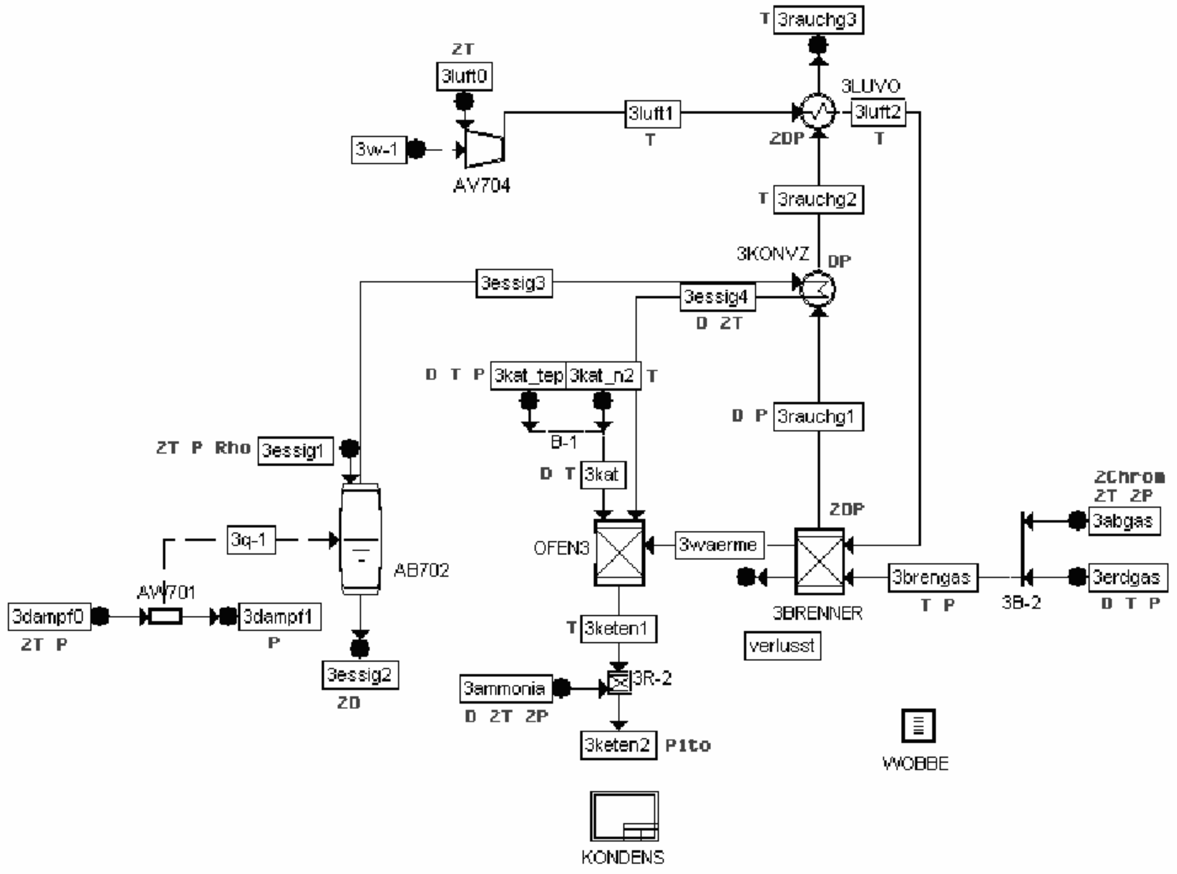


Figure 5.27: Ketene cracker: solution in the case of one sensor failure (part 1)

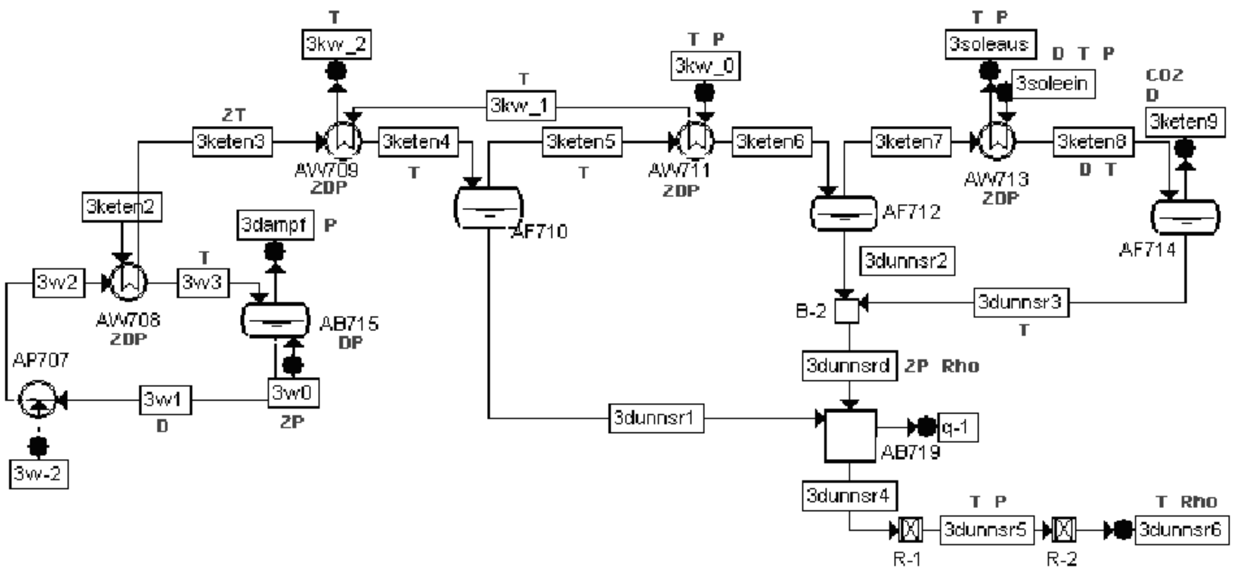


Figure 5.28: Ketene cracker: solution in the case of one sensor failure (part 2)

Table 5.18: Ketene cracker: global parallelization

Global parallelization					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
1	35780	35910	168000	1550	100
2	17227	18076	168000	1550	99.3
3	11585	12480	168000	1550	95.9
4	8568	9540	168000	1550	94.1
6	5728	6655	168000	1550	89.9
8	4332	5140	168000	1550	87.3
12	2854	3551	168000	1550	84.3

Table 5.19: Ketene cracker: distributed genetic algorithms

Distributed genetic algorithms					
Number of processors	Master processor CPU time (s)	Elapsed time (s)	Goal function	Number of generations	Efficiency (%)
1	35780	35910	168000	1546	100
2	8905	9438	160000	721	190.2
3	7611	8272	160000	1001	144.7
4	5519	6109	171000	1041	147.0
6	4245	5010	159000	1121	119.5
8	1589	2119	160000	641	211.8
8∗	2982	3997	160000	641	112.3
12	2134	2674	167000	1321	111.9
12∗	2258	2589	160000	521	115.6

It can be seen on figure 5.29 that the elapsed time and the master processor CPU time are inversely proportional to the number of processors in the case of global parallelization. The computing times are better in the case of distributed genetic algorithms. For the ∗ cases, the computing times are similar or higher than for the standard cases. Concerning the global number of generations (figure 5.30), it is smaller for distributed genetic algorithms.

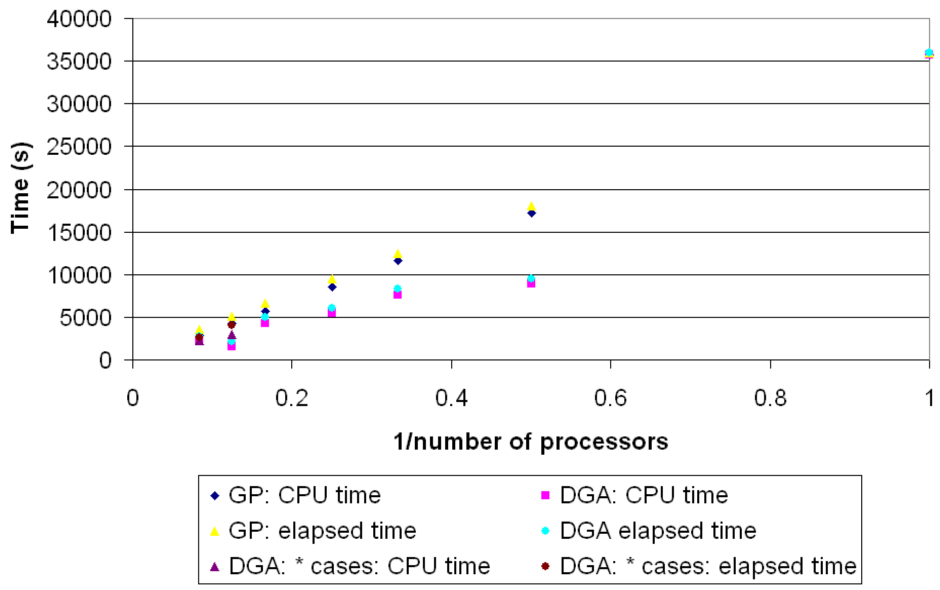


Figure 5.29: Ketene cracker: times comparison

The goal function is better in most of the cases for distributed genetic algorithms (figure 5.31). We can note that as for the other examples, the optimum reached with distributed genetic algorithm is different for each number of sub-populations as well as the total number

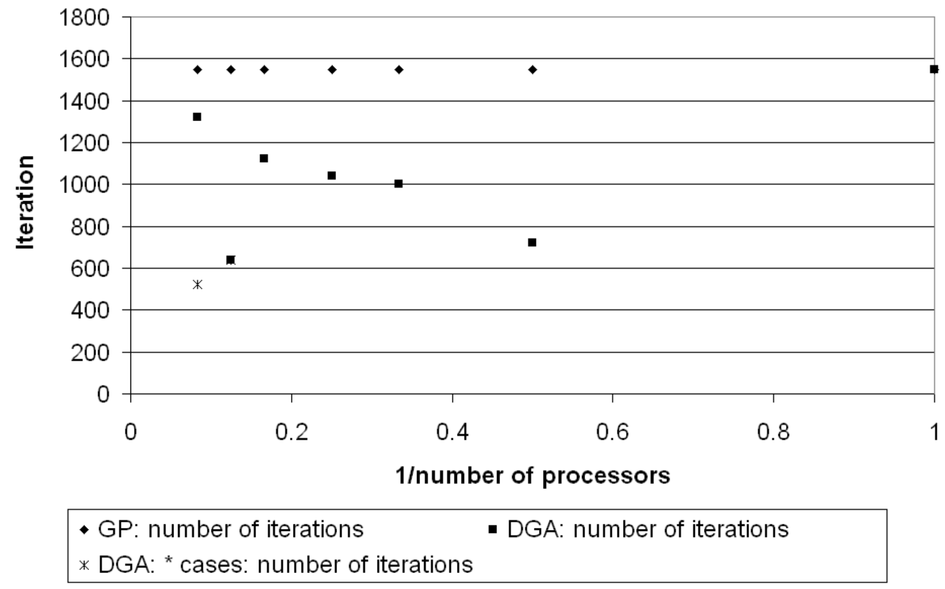


Figure 5.30: Ketene cracker: number of generations comparison

of generations. The explanation remains the independent evolution of the sub-populations. For the $*$ cases, it is better or equal than for the standard cases. It appears that an increase in computing time for the $*$ cases does not necessary implies a better goal function for this example.
In this example, the efficiency also decreases with the number of processors (figure 5.32) in the case of global parallelization. It is always better for distributed genetic algorithms. For this example, distributed algorithms are better than global parallelization in all points of view. The $*$ cases are sometimes better and sometimes worse than the standard cases. This depends mainly on the way followed to reach the solution which is different for every case while distributed genetic algorithms are used.

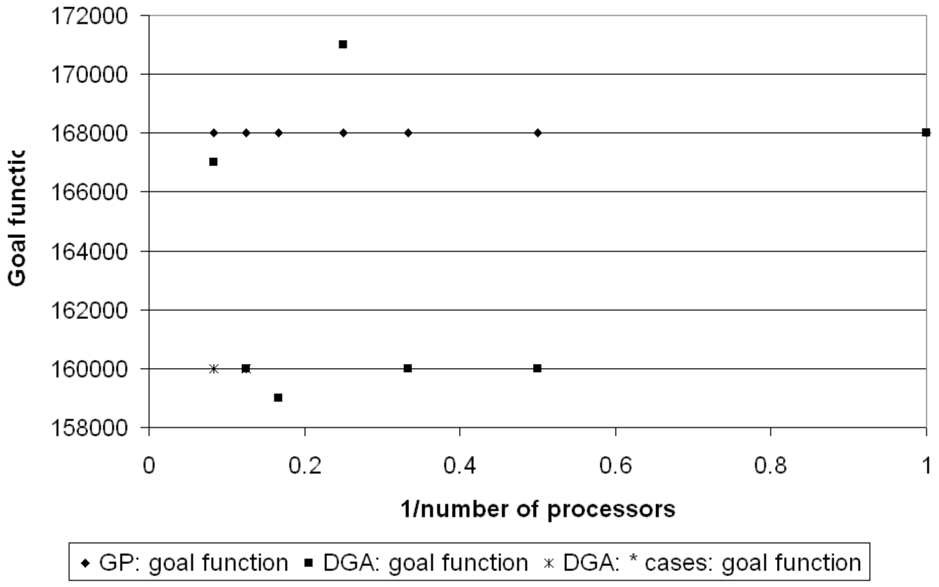


Figure 5.31: Ketene cracker: goal function comparison

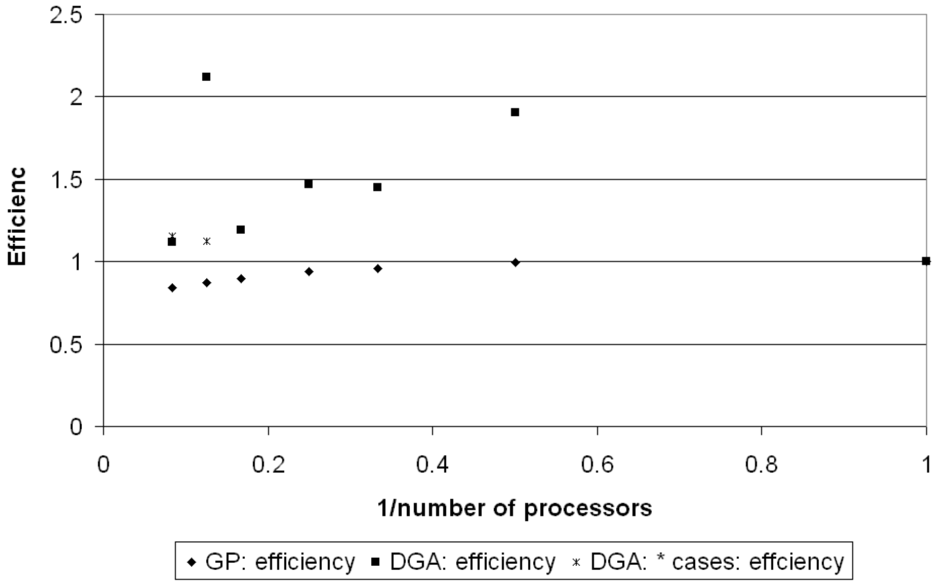


Figure 5.32: Ketene cracker: efficiency comparison

5.4 Naphta reformer

5.4.1 Process description

The last example is a naphta reformer. This catalytic reforming process is a typical refining process. Its original function is to upgrade low octane straight-run naphtha to higher octane number, by converting n-paraffins to iso-paraffins and naphtenes to aromatics. The hydrogen generated by the aromatic producing reactions also plays a major role in various hydrotreating units. The key performance indicators of the catalytic reforming process are, among others, the C5+ yields as well as the yields of light compounds, and especially H2. This process is made of 4 main sections:

- the preheat of the feed;
- the reaction section, with intermediate reheaters between the reactors;
- the gas-liquid separation (and compression) section;
- the stabilizer section were the liquid from the gas-liquid separation section is fractionated into the product streams.

This process is composed of:

- 80 units;
- 104 material streams composed of hydrogen, methane, ethane, propane, iso-butane, normal-butane, 2-methylbutane, normal-pentane, 2-2-dimethylbutane, normal-hexane, cyhexane, benzene, methylcyhexane, 1-1-dimethylcyhexane, 2-2-dimethylpentane, normal-heptane, toluene, ethylbenzene, normal-octane, 2-2-dimethylhexane, 2-2-3-trimethylhexane, 3-3-5-trimethylheptane, normal-nonane, normal-decane, normal-propylcyclohexane, iso-butylcyclohexane, 1-ethyl-2-methylbenzene and 1-3-diethylbenzene;
- 10 utility streams composed of steam and water;
- 12 mechanical streams;
- 21 thermal streams;
- 1263 variables;
- 1116 constraints equations.

Key parameters

The process comprises 9 key parameters which are listed with their prescribed standard deviations in the table below (Table 5.20).

Table 5.20: Naphta reformer: prescribed standard deviations on key parameters

Variable name	Standard deviation
Hydrogen yields (weight %)	10 %
Methane yields (weight %)	10 %
Ethane yields (weight %)	10 %
Propane yields (weight %)	10 %
Normal-butane yields (weight %)	10 %
Iso-butane yields (weight %)	10 %
Normal-pentane yields (weight %)	10 %
2-methylbutane yields (weight %)	10 %
C_5^+ (weight %)	10 %

Sensor database

The sensor database counts the following types of sensors:

Table 5.21: Naphta reformer: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values
Thermocouple A	20	0.2+0.1%	0 K	200 K
Thermocouple B	10	0.5+0.1%	200 K	400 K
Thermocouple C	30	0.2+0.05%	375 K	800 K
Thermocouple D	50	0.2+0.1%	750 K	2000 K
Pressure Gauge A	20	1%	0 bar	20 bar
Pressure Gauge B	30	0.01+1%	15 bar	300 bar
Pressure Gauge C	30	0.01+1%	290 bar	3000 bar
Delta_p	0	0.001+1%	-0.5 bar	300 bar
Chromatograph A	400	0.0001+0.5%	0	1
Chromatograph B	200	0.001+1%	0	1
Molar Flowmeter	60	2%	0 mol/s	20000 mol/s
Mass Flowmeter A	40	5%	0 kg/s	25 kg/s
Mass Flowmeter B	60	2%	20 kg/s	75 kg/s
Mass Flowmeter C	100	1%	50 kg/s	200 kg/s
Mass Flowmeter D	150	2%	175 kg/s	10000 kg/s
Volume Flowmeter	60	2%	0 m^3/h	20000 m^3/h
Cold Volume Flowmeter	60	2%	0 m^3/h	2000 m^3/h
Density Meter C	60	2%	0 kg/s	2000
CFDP Meter D	10	1%	-0.5	300

The initial solution that is obtained with this list of sensors costs 20930 cost units and counts 276 sensors:

- 24 chromatographs;
- 176 mass flow meters;

- 20 molar flow meters;
- 106 thermocouples;
- 102 pressure-gauges;
- 7 density meters;
- 1 volume flow meter;
- 12 cold volume flow meters;
- 25 virtual pressure drop meters.

This sensor network allows to satisfy all accuracies on key parameters. In this example, the solution space counts $2^{473} = 1.21 * 10^{83}$ different solutions.
This example is more complex than the previous ones: more sensors can be selected and the process involves more variables and equations.

5.4.2 Solution

Case of one operating point

In the case of one operating point, a feasible sensor network is obtained after 6 days and 4 hours on a processor Celeron (2.2 GHz) for a population of 48 individuals. This requires 1618 generations and 77665 goal function evaluations. That corresponds to 9 goal function evaluations per minute. This measurement system costs 1960 cost units and counts 97 sensors:

- 3 chromatographs;
- 10 mass flow meters;
- 45 thermocouples;
- 13 pressure-gauges;
- 1 density meter;
- 25 virtual pressure drop meters.

Case of one sensor failure

The case of the obtention of a redundant sensor network in the case of one sensor failure has not been treated. Indeed, it was estimated that the computing time would have been in the order of three months if the computing work was shared on the twelve available computers.

5.4.3 Parallelization: methods comparison

In all the cases, the global size of the population was fixed to 48 individuals and the stop criterion was taken equal to 200 for global parallelization and 40∗5 for distributed genetic algorithms. As for the two other examples, distributed genetic algorithms have also been tested for a global population of 96 individuals and a stop criterion equal to 20∗5 generations (for 8 and 12 processors). For distributed genetic algorithms, the number of generations between two successive migrations was fixed to 5 and the number of migrating individuals was taken equal to 2. The results are given in the tables 5.22 for global parallelization and 5.23 for distributed genetic algorithms.

Table 5.22: Naphta reformer: global parallelization

Global parallelization					
Number of processors	Master processor CPU time (min)	Elapsed time (min)	Goal function	Number of generations	Efficiency (%)
1	8888	8905	1955	1618	100
2	4368	4527	1955	1618	98.3
3	2910	3044	1955	1618	97.5
4	2107	2341	1955	1618	95.1
6	1548	1619	1955	1618	91.7
8	1042	1240	1955	1618	89.8
12	711	845	1955	1618	87.8

Table 5.23: Naphta reformer: distributed genetic algorithms

Distributed genetic algorithms					
Number of processors	Master processor CPU time (min)	Elapsed time (min)	Goal function	Number of generations	Efficiency (%)
1	8888	8905	1955	1618	100
2	2853	3007	2066	1041	148.1
3	1562	1904	2067	1161	155.9
4	1694	1832	2335	1281	121.5
6	891	1019	1978	1081	145.6
8	957	1227	1708	1961	90.7
8∗	1647	1731	1976	1281	64.3
12	545	603	2237	1361	123.1
12∗	1250	1354	1846	1461	54.8

It can be seen on figure 5.33 that the elapsed time and the master processor CPU time are inversely proportional to the number of processors in the case of global parallelization. The computing times are better in the case of distributed genetic algorithms. For the ∗ cases, the computing times are the worst. Concerning the global number of generations (figure

5.34), it is generally smaller for distributed genetic algorithms. It is sometimes better for the ∗ cases and sometime worse than for the standard cases.

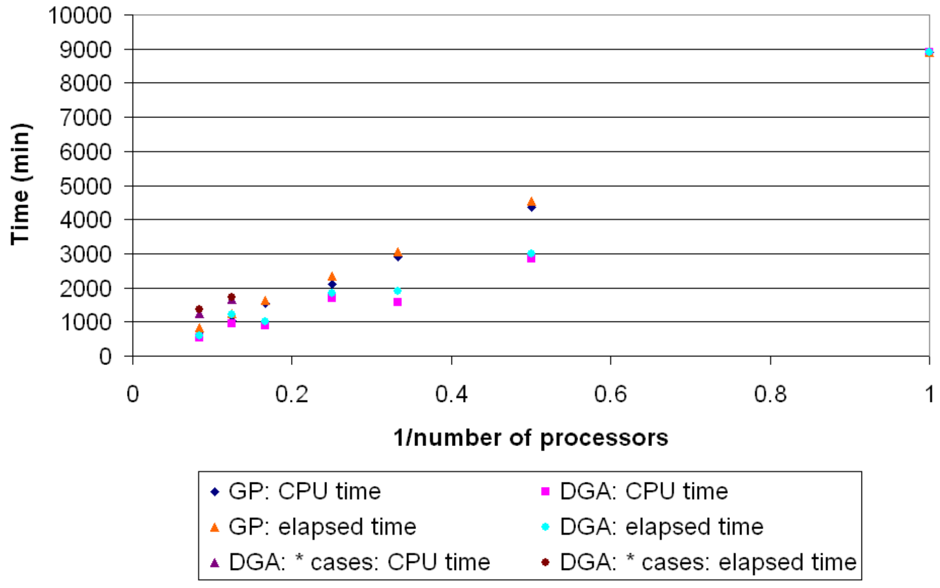


Figure 5.33: Naphta reformer: times comparison

The final value of the goal function is generally better for global parallelization (figure5.35). For the ∗ cases, it is sometimes better and sometimes worse than for the standard cases. The independent evolution of the sub-populations is the reason of the different local optima and the different total numbers of iterations reached when the number of sub-populations changes.
In this last example, the efficiency also decreases with the number of processors in the case of global parallelization (figure 5.36). It is the best for the standard cases of distributed genetic algorithms and the worst for the ∗ cases.
In the case of this example, global parallelization and the ∗ cases seem to be the most adapted if good goal function values are desired. If the computing time is privileged, the standard cases of the distributed genetic algorithms must be chosen. For this example, distributed algorithms are better than global parallelization in all points of view. The ∗ cases are sometimes better and sometimes worse than the standard cases. This mainly depends on the way followed to reach the solution which is different for every case while distributed genetic algorithms are used.
We can conclude that industrial size problems can be handled by the proposed algorithm. Even if the global optimal solution is not always located, a significant cost reduction with respect to the worst case solution is always achieved. Computer time grows significantly for larger size problems but parallelization provides an alternative that can be implemented on a small network of standard computers.

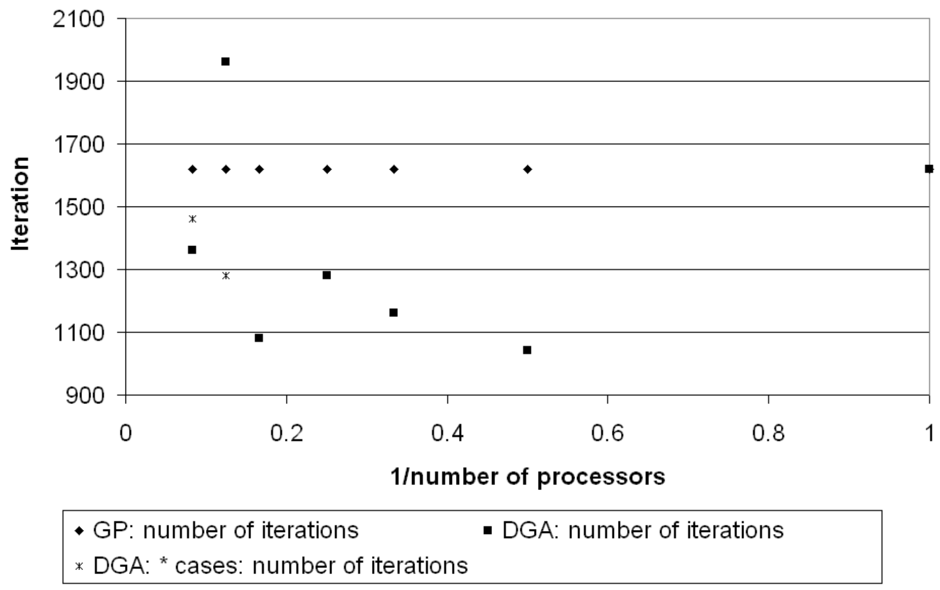


Figure 5.34: Naphta reformer: number of generations comparison

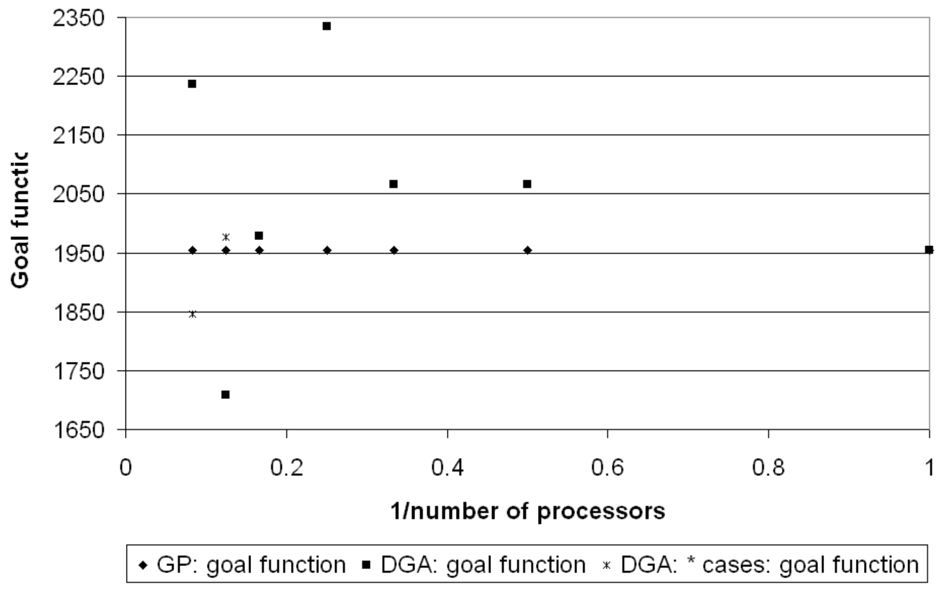


Figure 5.35: Naphta reformer: goal function comparison

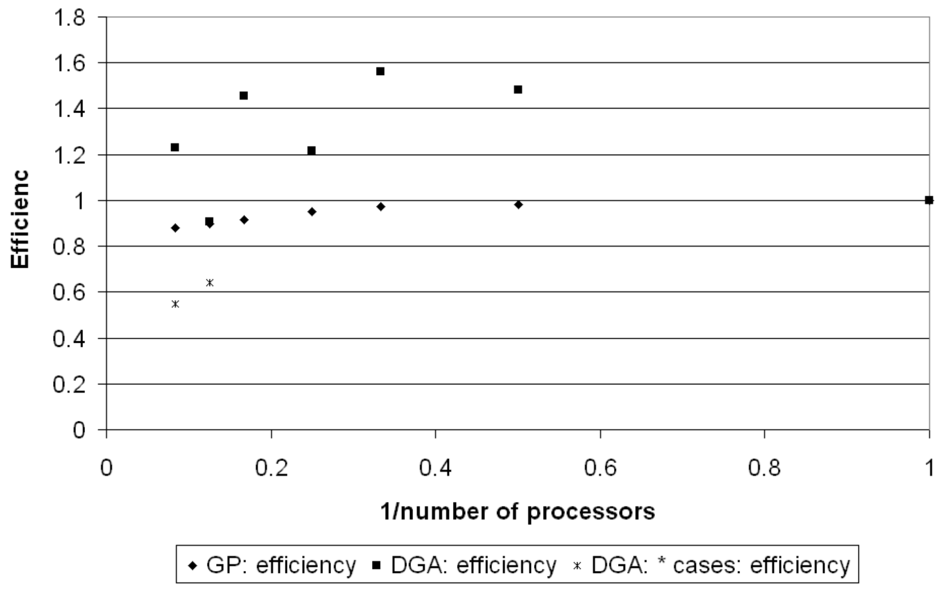


Figure 5.36: Naphta reformer: efficiency comparison

Chapter 6

Fault detection and localisation

Fault detection and isolation is another important subject in process monitoring. We will handle here an example where the objective is to detect leaks in a pipe network, based on flow measurements. Indeed, fluid leaks are expensive and must be detected as quickly as possible. However, all measurements are erroneous and the sensor precision has a great influence on the detectability and isolability of process faults, so that the sensor precision must be taken into account when a network is chosen. In this chapter, a general method to design the cheapest sensor network able to detect and locate a list of faults in a given process is proposed. The method is based on the fault detection method proposed by J. Ragot and D. Maquin (Ragot and Maquin, 2006). Those authors use the notion of sensibility to fault to decide weather a residual is influenced or not by a specified fault on the process.
The problem is similar to the problem of design of the cheapest sensor network allowing to reconcile the process and estimate all process key parameters within a prescribed accuracy: it is also multimodal, not derivable and involves many binary variables. That is why the genetic algorithm has been used with a different goal function.
The method is illustrated for two water networks of different sizes. The detected faults are leaks from pipes and storage tanks leaks but other types of fault could also be simulated and detected.

6.1 Fault detection and isolation

The objective of fault detection is to determine whether the measurements remain in the range of the values given by the process model for a chosen operating mode of the plant. If the distance between measurements and estimations is too important, a fault is detected. The fault detection and localisation techniques are carried out in two steps: the estimation of the residuals and the decision. In a way to be sure that all the faults that can occur in a process are detectable, the signature matrix Σ must be analysed. This matrix is the occurrence matrix of the variables in the residual equations. As an example, let us consider the following process containing four residuals and six variables at time t:

$$\begin{aligned}
r_1(t) &= f_1(x_1(t), x_2(t), x_5(t), x_6(t)) \\
r_2(t) &= f_2(x_1(t), x_2(t), x_3(t), x_5(t), x_6(t)) \\
r_3(t) &= f_3(x_3(t), x_5(t), x_6(t)) \\
r_4(t) &= f_4(x_2(t), x_4(t), x_5(t))
\end{aligned} \tag{6.1}$$

The corresponding signature matrix has the form:

$$\mathbf{\Sigma} = \begin{vmatrix} X & X & 0 & 0 & X & X \\ X & X & X & 0 & X & X \\ 0 & 0 & X & 0 & X & X \\ 0 & X & 0 & X & X & 0 \end{vmatrix} \tag{6.2}$$

A fault is detectable if the corresponding column in the signature matrix contains at least one non-zero element. A fault can be located if the corresponding column in the signature matrix is different from all other columns of the signature matrix. The fault localisation consists of deducing what is the fault from the values of the residuals. For that purpose, fuzzy rules are elaborated from the signature matrix. They are linguistic "if-then" constructions of the general form "if A then B" where A are the premises and B the consequences of the rule.
As noise influences the value of the residuals, some random perturbations which are not caused by any fault in the process can reach magnitude such that the corresponding residuals are then associated to faults. Thus faults may be detected even when none occurs and false alarms are then triggered. Taking into account temporal persistence allows to improve the detection procedure. For that purpose, one takes the average of the values of the variables for the k last measurement times, instead of instantaneous measurements.
The sensitivities of residuals to a given fault situation are different so that the magnitude of the residual deviations allows to characterize a fault situation. The detectability and isolability of faults can then be improved by using this difference of sensitivity. Let y(t) be the measurement of a variable of the process at measurement time t. It is always the sum of the true value x(t), the noise $\epsilon(t)$ and the fault f(t):

$$y(t) = x(t) + \epsilon(t) + f(t) \tag{6.3}$$

The true value satisfying completely the process model, the residual is composed of two terms: the contribution of the noise r_ϵ and the contribution of the fault r_f so that the effect of the fault can be masked by the effect of the noise according to their relative magnitudes. The noise contribution to the i^{th} residual is defined as follows:

$$r_{\epsilon,i} = \sum_{j=1}^{n} m_{ij}\epsilon_j \tag{6.4}$$

where m_{ij} are the elements of the matrix $\mathcal{M}$ of the derivatives of the residuals with respect to the variables. If the errors are replaced by the precision of the sensors e_j, one obtains the upper bound of the contribution of the noise on the i^{th} residual:

$$r_{\epsilon,i} = \sum_{j=1}^{n} |m_{ij}| e_j \tag{6.5}$$

In the same way, the contribution of a unique fault f_j affecting the i^{th} residual is defined as follows:

$$r_{f,i} = m_{ij} f_j \tag{6.6}$$

The lowest magnitude of the i^{th} residual that allows to distinguish between the noise and the fault f_j is defined by the bound:

$$\tau_{ij} = \frac{\sum_{j=i}^{n} |m_{ij}| e_j}{|m_{ij}|} \tag{6.7}$$

So, the i^{th} residual is sensitive to fault f_j if the magnitude of that fault is higher than τ_{ij}. A fault f_j will be located if for all non-zero element of the signature matrix, the absolute value of the corresponding residual has a value higher than the corresponding bound τ_{ij} and for each zero element of the signature matrix, the absolute value of the corresponding residual has a value smaller than a fixed high value. Let, for example, the jacobian matrix of the model equations 6.1:

$$\mathcal{M} = \left| \begin{array}{cccccc} 1 & -0.5 & 0 & 0 & 1 & -2.5 \\ 2 & -4 & 2 & 0 & 3 & 1 \\ 0 & 0 & 3 & 0 & -2 & -1 \\ 0 & 6 & 0 & -5 & -4 & 0 \end{array} \right| \tag{6.8}$$

For the following error vector $e = (0.5, 1, 0.8, 0.4, 1, 0.4)$, the corresponding bounds matrix is given by:

$$\tau = \left| \begin{array}{cccccc} 3 & 6 & \infty & \infty & 3 & 1.2 \\ 5 & 2.5 & 5 & \infty & 3.3 & 10 \\ \infty & \infty & 1.6 & \infty & 2.4 & 4.8 \\ \infty & 2 & \infty & 2.4 & 3 & \infty \end{array} \right| \tag{6.9}$$

So, the third fault will be detected and located if the second residual has an absolute value higher than 5 and the third one an absolute value higher than 1.6.

6.2 Method description

The optimal sensor network that allows to detect and locate all the specified faults is carried out in five steps:

- Process and faults simulation;
- Specification of the sensor database and the sensor requirements;
- Verification of the problem feasibility;
- Optimisation of the sensor network;
- Report generation.

6.2.1 Process and faults simulation

We consider here processes consisting in tanks, pipes, mixers and dividers, such as the one shown in figure 6.3. The process is first simulated for a normal, fault free operating. The differential balance equations are integrated by means of the fourth order Runge-Kutta method. Then, for each possible fault one decides the minimal magnitude of the fault that should be detected by the sensor network, for example a leak of 1% of a stream flow rate. The faults are simulated one by one by increasing progressively their magnitude until the minimal fault that should be detected is reached. The values of the flowrates at both ends of each pipe are recorded for each fault. No noise is added to the variables at this step because the noise will depend on the precision of the measurement tools. The number of time measurements that are used for computing the variables mean values depends on the frequency of the measurements and the speed at which the fault should be detected. If the number of measurement times is higher, the fault detection and localisation is slower and better. If this number is too small, the noise influences more the magnitude of the residuals and the fault detection is more difficult. The value of that parameter has been chosen to five. This parameter, as well as the time at which the fault should be detected, is studied for the first example presented here after.

6.2.2 Specification of the sensor database and the sensor requirements

The sensor database and the sensor requirements files contain the same information as the ones defined in the first part of the thesis.

6.2.3 Verification of the problem feasibility

The step of feasibility of the problem verification begins by listing all the sensors that can be placed in the plant. For each sensor of this list, a binary gene is created. If a variable is measured by more than one sensor, the precision of the most accurate one is taken into account for the bounds calculation. The residual bounds and the residuals are estimated for the initial sensor network: indeed, a noise bounded by the accuracy of the sensor is added to each variable for each measurement time before the mean of the variables and the residuals are calculated. The noise on the variables and then their values depend thus on the sensor network as well as on the residual bounds. To ensure that a solution exists to the studied problem, the next condition has to be met: the initial sensor network has to be able to detect all the simulated faults. If it is not the case, new sensor types that are more precise can be added to the data base or the minimal magnitudes of the faults that should be detected should be increased.

6.2.4 Optimisation of the sensor network

Once one is sure that a solution exists, it can be optimized. The objective function to be minimized is evaluated this way:

- if all the faults can be detected and located, the goal function is the sum of the costs of all the sensors in the chosen network;
- if at least one fault can not be detected or located, the goal function is a multiple (2 has been chosen) of the maximum cost .

The goal function being generally multimodal, the problem being not derivable and containing only binary parameters, the genetic algorithm is still used for this version of the program. The probabilities of the evolution mechanisms remain unchanged as well as the size of the population:

- probability for each gene to be chosen at first generation: 80%;
- size of the population: 20 individuals;
- probability of reproduction: 50%
- probability of single-point cross-over: 50%;
- probability of jump mutation: 1%.

The goal function of each individual of the new population is evaluated. The best one is then kept and duplicated. The solution is still reached if after a specified number of generations the goal function of the best one remains unchanged.

6.2.5 Report generation

The report generated here contains the same information as the one generated for the first version of the program. The only difference is that there is no prescribed accuracy on key variables.

6.3 Cases study

In this paragraph, two water networks are studied.

6.3.1 First example

This first one is composed of five storage tanks and ten water pipes (6.1).

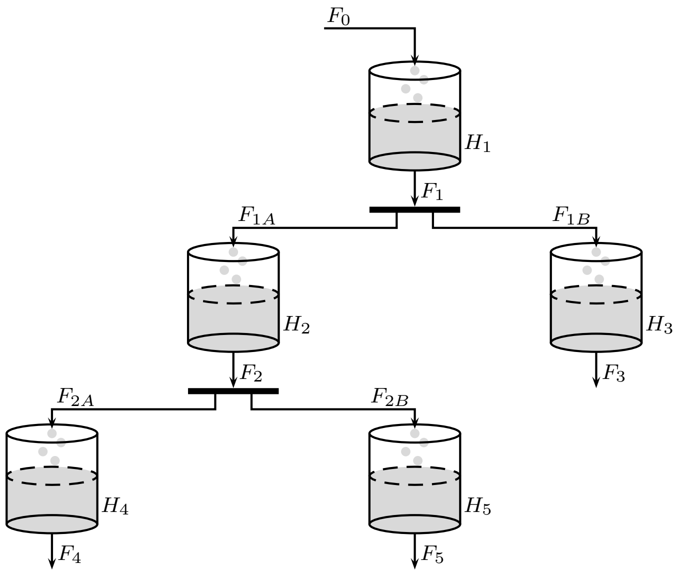


Figure 6.1: Flow sheet of the first example of fault detection

The fifteen faults that should be detected and located are water leaks in the storage tanks or in the pipes. In the storage tanks, the level meters can be situated at one place and the flow rate in the pipes can be measured at the beginning and at the end of each pipe, which means 25 possible sensor locations. The sensor database contains three level meters with different accuracies and prices, and 10 flowmeters with different accuracies, prices and measurement domains:

Table 6.1: First water network: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values
Height A	40	0.05	0 m	70 dm
Height B	50	0.03	0 m	70 dm

continued on next page

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values
Height C	60	0.01	0 m	70 dm
Mass Flowmeter B	50	5%	20 kg/s	75 kg/s
Mass Flowmeter BB	70	2%	20 kg/s	75 kg/s
Mass Flowmeter BBB	90	1%	20 kg/s	75 kg/s
Mass Flowmeter BBBB	110	0.5%	20 kg/s	75 kg/s
Mass Flowmeter BBBBB	130	0.2%	20 kg/s	75 kg/s
Mass Flowmeter C	60	5%	50 kg/s	200 kg/s
Mass Flowmeter CC	80	2%	50 kg/s	200 kg/s
Mass Flowmeter CCC	100	1%	50 kg/s	200 kg/s
Mass Flowmeter CCCC	120	0.5%	50 kg/s	200 kg/s
Mass Flowmeter CCCCC	140	0.2%	50 kg/s	200 kg/s

With this database, it is possible to place 135 sensors. That corresponds to a solution space of $2^{135} = 4.4 * 10^{40}$ solutions. This most expensive measurement system has a total cost of 11950 cost units.
The number of measurement times and the time until the leak is detected are examined on figure 6.2 and table 6.2 here after:

Table 6.2: Cost evolution with the number of measurement times and the detection time

Time needed to detect leak	4 measurements times	5 measurements times	6 measurements times	7 measurements times
6 s	2510	2600	-	-
7 s	2370	2420	2510	2620
8 s	2250	2330	2370	2460
9 s	2110	2230	2250	2280
10 s	2090	2100	2170	2220
11 s	1960	2080	2000	2070
12 s	1950	1950	1930	2030
13 s	1900	1920	1880	1980
14 s	1900	1860	1840	1930
15 s	1900	1860	1840	1860
16 s	1900	1860	1840	1880
17 s	1900	1860	1840	1830
18 s	1900	1860	1840	1830
19 s	1900	1860	1840	1830
20 s	1900	1860	1840	1830

It appears that for a same number of measurement times taken for the means calculation, the cost of the optimum sensor network decreases when the fault increases and becomes easier to detect. Once the fault is maximum, the cost of the network remains unchanged with the elapsed time. If the elapsed time is small, the leaks are not maximum and the sensor network found is more expensive when the number of measurement times increases: indeed for the highest number of measurement times, the mean is not very different from the normal case. Thus the fault can not be detected easily. On the other hand, the sensor network is cheaper for a higher number of measurement times when the fault is maximum and is easier to detect. For a passed time of 16 s and 7 measurement times, the cost does not decrease as expected: this can be due to the fact that the noise which is a little different for each case: indeed the noises on several measurement times may cancel each other, making the fault more difficult to detect.

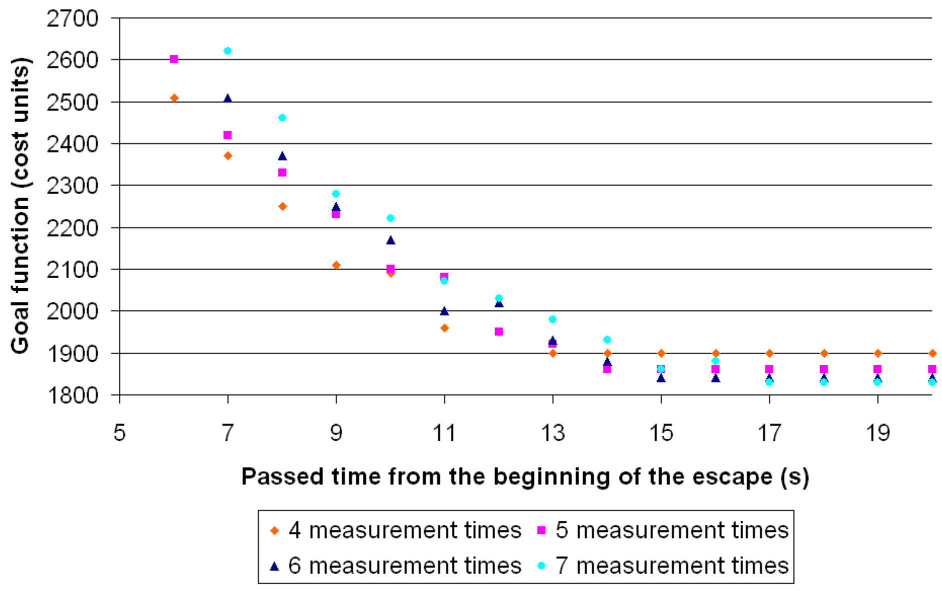


Figure 6.2: Influence on the goal function of the number of measurement times and the elapsed time from the beginning of the leak until detection

Obtaining the solution requires 14770 generations (295421 goal function evaluations) for a stop criterion of 6000 generations, a number of measurement times equal to five and passed time from the beginning of the escape of 9 s. The optimal sensor network is obtained after 301 seconds on a 1.6 GHz computer. This solution costs 1860 costs unit and counts 25 sensors: one for each possible sensor location. It allows to detect and locate all the 15 faults. In this case of the design of the optimal sensor network that is able to detect and locate all the faults, the optimization consists of choosing one sensor for each sensor location in a way that the faults can be detected and located, while minimizing the annualized cost. The most expensive of those network costs 3100 cost units (1240 cost units more than the optimal one).

6.3.2 Second example

The second water network (6.3) is composed of 14 storage tanks and 31 pipes so that there are 76 possible sensor locations and 45 fault to be detected and located. The sensor

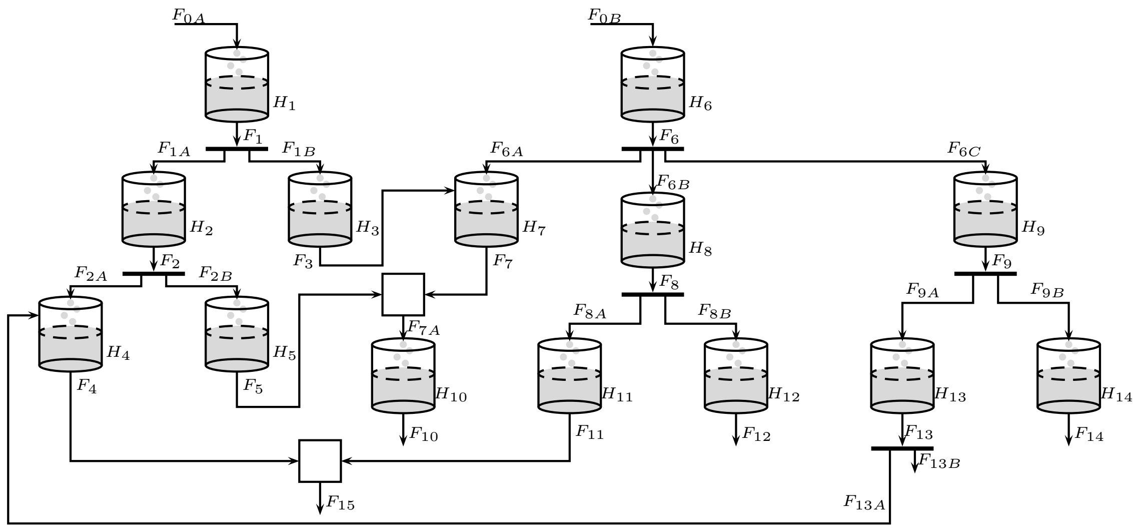


Figure 6.3: Flow sheet of the second example of fault detection

database contains this time three level meters with different accuracies and prices, and 15 flowmeters with different accuracies, prices and measurement domains.

Table 6.3: Second water network: sensor database

Sensor types	Costs	Accuracies	Minimum values	Maximum values
Height A	40	0.05	0 m	70 dm
Height B	50	0.03	0 m	70 dm
Height C	60	0.01	0 m	70 dm
Mass Flowmeter A	40	5%	0 kg/s	25 kg/s
Mass Flowmeter AA	60	2%	0 kg/s	25 kg/s
Mass Flowmeter AAA	80	1%	0 kg/s	25 kg/s
Mass Flowmeter AAAA	100	0.5%	0 kg/s	25 kg/s
Mass Flowmeter AAAAA	120	0.2%	0 kg/s	25 kg/s
Mass Flowmeter B	50	5%	20 kg/s	75 kg/s
Mass Flowmeter BB	70	2%	20 kg/s	75 kg/s
Mass Flowmeter BBB	90	1%	20 kg/s	75 kg/s
Mass Flowmeter BBBB	110	0.5%	20 kg/s	75 kg/s
Mass Flowmeter BBBBB	130	0.2%	20 kg/s	75 kg/s
Mass Flowmeter C	60	5%	50 kg/s	200 kg/s
Mass Flowmeter CC	80	2%	50 kg/s	200 kg/s

continued on next page

Sensor types	Costs	Accuracies	Minimum values	Maximum values
Mass Flowmeter CCC	100	1%	50 kg/s	200 kg/s
Mass Flowmeter CCCC	120	0.5%	50 kg/s	200 kg/s
Mass Flowmeter CCCCC	140	0.2%	50 kg/s	200 kg/s

The initial network counts 392 possible sensors. This corresponds to a solution space of $2^{392} = 10^{118}$ solutions. This sensor network has a total cost of 34100 costs units. Obtaining the solution requires 26104 generations (522101 goal function evaluations) for a stop criterion of 6000 generations, a number of measurement times equal to five and a detection time from the beginning of the leak of 15 s. The optimal sensor network is obtained after 5667 seconds on a 1.6 GHz computer. This solution costs 6200 cost units and counts 76 sensors: one by possible sensor location and allows to detect and locate all the 45 faults. In this case also the design of the optimal sensor network able to detect and locate all the faults consists of optimizing the selection of one sensor for each location in a way that the faults can be detected and located, while minimizing the annualized cost. The most expensive of those networks costs this time 9000 cost units (2800 cost units more than the optimal one).

6.3.3 Conclusions

The proposed method allows to determine a sensor network that is able to detect and locate a specified list of tank and pipe leaks. This network is much cheaper than the initial one but due to the optimization method, there is no guarantee that it is always the best one.
The number of measurement times required to declare a fault must be chosen as a function of the problem. It should not be to high because it increases the time elapsed between the occurrence of the fault and its detection, or the sensor network required is more expensive. On the other hand, a too small number of measurement times may cause the detection of faults that does not exists.
This method could be transposed for other types of faults such as the catalyst deactivation or the loss of efficiency in a compressor.
As computing times becomes important when the size of the problem increases, a parallelisation of the method could be envisaged for larger size problems.

Chapter 7

Conclusions part I

The proposed method allows to achieve an important progress in the design of sensor networks in the case of steady-state processes. Indeed, the method using the linearised equations of the model at the operating point can be treated easily for models including energy balances and non linear equations in addition to the mass balances. Moreover, the calculation of a posteriori standard deviations based on steady-state data reconciliation allows to introduce a term in the goal function that takes into account the accuracy of some key variables of the process that have to be known precisely for the efficient monitoring of the plant.
The design of sensor networks allowing to detect a specified list of process faults has also been envisaged and gives good results. A compromise has to be done between the speed of fault detection and the necessity to make the difference between punctual measurement error and process faults.
The optimization method based on genetic algorithms allows to find a solution which is much better than the initial one for all the studied examples. This solution found light differs from the global optimum since all possible solutions are not tested but it is reached within a computing time that is reasonable in comparison to the one required for the evaluation of all possible solutions.
The systematic application of such a method while creating a new plant should allow to achieve important savings. Indeed, sensor networks installed on the industries are generally reduced to the minimum because of the cost of the sensors. Some variables that seem to be without interest at a moment can however appear to be very important for the control of the process or for the evaluation of a key performance parameter. It is then very expensive to install new sensors after the plant has been put in operation. Moreover the global cost of a sensor network upgrade can then reveal to be more important than the one that would have been determined by design method.
The method also allows to treat in a unique calculation several operating points and thus find sensor networks that limit the uncertainty on key performance indicators when the plant changes its operating point to obtain, for example, a product with other specifications.
Being able to design a sensor network that remains efficient in the case of one sensor failure is also interesting: indeed, it improved the system availability and allows to repair without

interrupting the plant operation.
The parallelization of the program allowed to reduce the computing time whatever the method that is used and the studied example. In the case of global parallelization, the computing time is always inversely proportional to the number of processors and the efficiency decreases when the number of processors increases. The efficiency loss increases more quickly when the number of processors becomes nearer to the number of goal functions that has to be evaluated at each generation. So that, a compromise has to be found between the number of processors and the computing time.
In the case of distributed genetic algorithms, those tendencies can not be reproduced. Indeed, since the path followed to reach the solution is different, the network that is found differs from the one found with global parallelization if local minima are reached instead of the global solution. The computing time can also be different according to the number of generations needed to produce by random evolution an efficient solution. Following the studied example, distributed genetic algorithms are sometimes better and sometimes worse than global parallelization. For a same example, the choice between global parallelization and distributed genetic algorithms is not always clear-cut. One method may need less time to detect convergence, but remain stuck in a local optimum, but the ranking may change when a different number of processors is used. Moreover, it also appears from the examples, that it is usually preferable if the size of the sub-populations is small in the case of distributed genetic algorithms to decrease the stop criterion and increase the size of the sub-populations by the same factor. So, when the user decides to use a parallelised version of the software, he never knows in advance if he has chosen the best method but he is sure to reach the solution within a shorter computing time.

Part II

Dynamic data reconciliation

Chapter 8

Formulation of the dynamic data reconciliation problem

The dynamic data reconciliation allows to use temporal redundancies by taking into account several discrete measurements for some state variables. The general dynamic data reconciliation problem can be formulated as in (Liebman et al., 1992), (Rao, 2000):

$$\min_{x_{i,j},z_{i,j},u_{i,j}} \Phi\left[x_{i,j}, x_{i,j}^{m}, z_{i,j}, z_{i,j}^{m}, u_{i,j}, u_{i,j}^{m}, \sigma\right] \tag{8.1}$$

subject to

- differential constraints:
$$f\left(\frac{dx_{i,j}}{dt}, x_{i,j}, z_{i,j}, u_{i,j}\right) = 0 \tag{8.2}$$

- equality constraints:
$$\mathbf{h}\left[x_{i,j}, z_{i,j}, u_{i,j}\right] = \mathbf{0} \tag{8.3}$$

- inequality constraints:
$$\mathbf{g}\left[x_{i,j}, z_{i,j}, u_{i,j}\right] \geq \mathbf{0} \tag{8.4}$$

with the initial conditions:

$$x_{i,0} \quad = \quad x_{i,0}^{CI} \tag{8.5}$$

where

- $x_{i,j}$ is the estimation of the differential state variable i at measurement time j;
- $x_{i,j}^{m}$ is the measurement of the differential state variable i at measurement time j;
- $x_{i,0}^{CI}$ is the initial condition of the differential state variable i. It corresponds to the estimation of that variable at the same time of the previous reconciliation horizon.

- $z_{i,j}$ is the estimation of the algebraic state variable i at measurement time j;
- $z^m_{i,j}$ is the measurement of the algebraic state variable i at measurement time j;
- $u_{i,j}$ is the estimation of the input variable i at measurement time j;
- $u^m_{i,j}$ is the measurement of the input variable i at measurement time j;
- σ are the precisions of the measurements.

For most applications, the objective function is the sum for all the times and all the measurements of weighted square residues:

$$\begin{aligned}\Phi\left[x_{i,j}, x^m_{i,j}, z_{i,j}, z^m_{i,j}, u_{i,j}, u^m_{i,j}, \sigma\right] &= \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{x_{mes}}} \left(\frac{x_{i,j} - x^m_{i,j}}{\sigma_{x_{i,j}}}\right)^2 \\ &+ \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{z_{mes}}} \left(\frac{z_{i,j} - z^m_{i,j}}{\sigma_{z_{i,j}}}\right)^2 \\ &+ \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{u_{mes}}} \left(\frac{u_{i,j} - u^m_{i,j}}{\sigma_{u_{i,j}}}\right)^2 \qquad (8.6)\end{aligned}$$

where

- $n_{x_{mes}}$ is the number of measured differential state variables;
- $n_{z_{mes}}$ is the number of measured algebraic state variables;
- $n_{u_{mes}}$ is the number of measured input variables;
- $\sigma_{x_{i,j}}$ is the standard deviation on the differential state variable i at measurement time j;
- $\sigma_{z_{i,j}}$ is the standard deviation on the algebraic state variable i at measurement time j;
- $\sigma_{u_{i,j}}$ is the standard deviation on the input variable i at measurement time j.

When the objective function is formulated this way, all the measurements that are carried out between time 0 and time N are reconciled simultaneously. When the time horizon increases (when new measurements are carried out), the calculation efforts increase. At time N, the measurements carried out in time zero are reconciled $N+1$ times with different neighboring measurements. The choice of the best reconciled values remains an open question: for on-line control, one should probably choose the value corresponding to the last measurement time; for archiving or for off-line calculations, the values of the middle of the time horizon are preferred.

In the filtering method which is described in chapter 9, the information contained in the $N-1$ first times is summarized in the prediction vector. So that, only the measurements carried out at the last time step of time horizon is used at each reconciliation time.
In chapter 10, the moving window methods are explained. They also allow to use only a part of the measurements carried out during the time horizon.

Chapter 9

Filtering methods

9.1 Introduction

In this chapter, some generalities are given concerning filtering methods. Then the extended Kalman filter is described. This is a sequential estimation algorithm that produces minimum variance linear estimates of state variables.
Kalman filter is said to be invented by Rudolf E. Kalman in 1960 (Kalman and Bucy, 1961), (Kalman, 1960) though Thorvald N. Thiele and Peter Swerling developed a similar algorithm earlier. The first implementation of Kalman filter was carried out by Stanley Schmidt (Schmidt, 1980) to solve the trajectory estimation problem for the Apollo program. This first filter called *linear Kalman filter* is used to the reconciliation of linear dynamic processes. The Kalman filter has then been extended to deal with non linear systems (Karjala and Himmelblau, 1996), (Narasimhan and Jordache, 2000): the non linear part of the model is linearized thanks to a first order Taylor serie around the current estimated. This second filter is called *extended Kalman filter*. An other method based on second order divided differences is also used in the case of non linear systems (Norgaard et al., 2000). Nowadays, a wide variety of Kalman filters has been developed and is widely used in engineering applications such as radars, computer vision, autonomous or assisted navigation.

Dynamic systems can be described by means of two models:

- the dynamic model;
- the observation model.

Filtering methods proceed in two steps for each estimation:

- the time update: the state variables and the covariance matrix of the prediction errors are predicted from the dynamic model;
- the observation update: the gain matrix is first calculated by weighting measurements with predicted state. The states and the covariance matrix of the estimation errors are then carried out.

9.2 Extended Kalman Filter

In the case of the extended Kalman filter, the dynamic model is a set of non linear dynamic difference equations:

$$\mathbf{x}_j = \mathbf{f}\left(\mathbf{x}_{j-1}, \mathbf{u}_{j-1}, \mathbf{v}_{j-1}\right) \tag{9.1}$$

and the observation model:

$$\mathbf{y}_j = \mathbf{h}\left(\mathbf{x}_j, \mathbf{w}_j\right) \tag{9.2}$$

In those equations,

- $\mathbf{x}_j$ is the vector of state variables at time step j;
- $\mathbf{u}_{j-1}$ is the vector of input variables at time step j-1;
- $\mathbf{v}_{j-1}$ is the vector of process noises at time step j-1;
- $\mathbf{y}_j$ is the vector of measurements at time step j;
- $\mathbf{w}_j$ is the vector of measurement noises at time step j;
- $f\left(\mathbf{x}_{j-1}, \mathbf{u}_{j-1}, \mathbf{v}_{j-1}\right)$ is a set of non linear functions of $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{v}$ at time step j-1;
- $h\left(\mathbf{x}_j, \mathbf{w}_j\right)$ is a set of non linear functions of $\mathbf{x}$ and $\mathbf{w}$ at time step j;
- $\mathbf{v}$ and $\mathbf{w}$ are assumed to be white noises, independent from one another and with zero mean:

$$E\left[\mathbf{v}_j\ \mathbf{w}_l^T\right] = \mathbf{0} \qquad \forall j, l \tag{9.3}$$
$$E\left[\mathbf{v}_j\right] = \mathbf{0} \qquad \forall j \tag{9.4}$$
$$E\left[\mathbf{v}_j\ \mathbf{v}_l^T\right] = \mathbf{Q}_j \delta_{j,l} \tag{9.5}$$
$$E\left[\mathbf{w}_j\right] = \mathbf{0} \qquad \forall j \tag{9.6}$$
$$E\left[\mathbf{w}_j\ \mathbf{w}_l^T\right] = \mathbf{R}_j\ \delta_{j,l} \tag{9.7}$$

  where

  - $\mathbf{Q}_j$ is the process noises covariance matrix. This matrix may change at each time step;
  - $\mathbf{R}_j$ is the measurement noises covariance matrix. This matrix may change at each time step.

The filter counts four initial conditions:

-
$$\widehat{\mathbf{x}}_{0|0} = E\left[\mathbf{x}_0\right] = \overline{\mathbf{x}}_0 \tag{9.8}$$

- $$\mathbf{P}_{0|0} = cov\left(\mathbf{x}_0\right) = E\left[\left(\mathbf{x}_0 - \overline{\mathbf{x}}_0\right)\left(\mathbf{x}_0 - \overline{\mathbf{x}}_0\right)^T\right] = \mathbf{P}_0 \tag{9.9}$$

- $\mathbf{x}_0$ and $\mathbf{w}_j$ are not correlated:
$$E\left[\mathbf{x}_0\ \mathbf{w}_j^T\right] = \mathbf{0} \qquad \forall j \tag{9.10}$$

- $\mathbf{x}_0$ and $\mathbf{v}_j$ are not correlated:
$$E\left[\mathbf{x}_0\ \mathbf{v}_j^T\right] = \mathbf{0} \qquad \forall j \tag{9.11}$$

As the noises vectors $\mathbf{v}_j$ and $\mathbf{w}_j$ are zero mean and can not be known at each time step in practice, the model equations 9.1 and 9.2 can be approximated by considering they are null:

$$\begin{aligned} \overline{\mathbf{x}}_j &= \mathbf{f}\left(\widehat{\mathbf{x}}_{j-1|j-1}, \mathbf{u}_{j-1}, \mathbf{0}\right) && (9.12)\\ \overline{\mathbf{y}}_j &= \mathbf{h}\left(\overline{\mathbf{x}}_j, \mathbf{0}\right) && (9.13)\end{aligned}$$

If one linearizes those estimates:

$$\begin{aligned} \mathbf{x}_j &\approx \overline{\mathbf{x}}_j + \mathbf{F}_{j-1}\ \left(\mathbf{x}_{j-1} - \widehat{\mathbf{x}}_{j-1|j-1}\right) + \mathbf{V}_{j-1}\ \mathbf{v}_{j-1} && (9.14)\\ \mathbf{y}_j &\approx \overline{\mathbf{y}}_j + \mathbf{H}_j\ \left(\mathbf{x}_j - \overline{\mathbf{x}}_j\right) + \mathbf{W}_j\ \mathbf{w}_j && (9.15)\end{aligned}$$

where

- $\mathbf{F}_{j-1}$ is the Jacobian matrix of partial derivatives of f with respect to $\mathbf{x}$:
$$F_{j-1}\left(k,l\right) = \left.\frac{\delta f\left(k\right)}{\delta \mathbf{x}\left(l\right)}\right|_{\mathbf{x}=\widehat{\mathbf{x}}_{j|j-1}, \mathbf{u}=\widehat{\mathbf{u}}_{j-1}, \mathbf{v}=\mathbf{0}} \tag{9.16}$$

- $\mathbf{V}_{j-1}$ is the Jacobian matrix of partial derivatives of f with respect to $\mathbf{v}$:
$$V_{j-1}\left(k,l\right) = \left.\frac{\delta f\left(k\right)}{\delta \mathbf{v}\left(l\right)}\right|_{\mathbf{x}=\widehat{\mathbf{x}}_{j|j-1}, \mathbf{u}=\widehat{\mathbf{u}}_{j-1}, \mathbf{v}=\mathbf{0}} \tag{9.17}$$

- $\mathbf{H}_{j-1}$ is the Jacobian matrix of partial derivatives of h with respect to $\mathbf{x}$:
$$H_{j-1}\left(k,l\right) = \left.\frac{\delta h\left(k\right)}{\delta \mathbf{x}\left(l\right)}\right|_{\mathbf{x}=\widetilde{\mathbf{x}}_j, \mathbf{w}=\mathbf{0}} \tag{9.18}$$

- $\mathbf{W}_{j-1}$ is the Jacobian matrix of partial derivatives of h with respect to $\mathbf{w}$:
$$W_{j-1}\left(k,l\right) = \left.\frac{\delta h\left(k\right)}{\delta \mathbf{w}\left(l\right)}\right|_{\mathbf{x}=\widetilde{\mathbf{x}}_j, \mathbf{w}=\mathbf{0}} \tag{9.19}$$

9.2.1 Filtering equations or observation update

In the case of the extended Kalman filter, the estimated states are written:

$$\widehat{\mathbf{x}}_{j|j} = \widehat{\mathbf{x}}_{j|j-1} + \mathbf{K}_j \ \left[\mathbf{y}_j - \mathbf{h}\left(\widehat{\mathbf{x}}_{j|j-1}, \mathbf{0}\right)\right] \tag{9.20}$$

Where $\mathbf{K}_j$ is *the gain matrix*:

$$\mathbf{K}_j = \left(\mathbf{P}_{j|j-1}\ \mathbf{H}_j^T\right)\left[\mathbf{H}_j\ \mathbf{P}_{j|j-1}\ \mathbf{H}_j^T + \mathbf{W}_j\ \mathbf{R}_j\ \mathbf{W}_j^T\right]^{-1} \tag{9.21}$$

If one is more confident in the measurement that in the model, the measurement covariance matrix $\mathbf{R}_j$ approches zero and the gain matrix weights the residual more heavily:

$$\lim_{\mathbf{R}_j \to 0} \mathbf{K}_j = \mathbf{H}_j^{-1} \tag{9.22}$$

On the other hand, if one is more confident in the model that in the measurements, the measurement covariance matrix weights the residual less heavily:

$$\lim_{\mathbf{P}_{j|j-1} \to 0} \mathbf{K}_j = 0 \tag{9.23}$$

The covariance matrix of the estimation errors takes the form:

$$\mathbf{P}_{j|j} = \left[\mathbf{I} - \mathbf{K}_j\ \mathbf{H}_j\right]\ \mathbf{P}_{j|j-1} \tag{9.24}$$

9.2.2 Prediction equations or time update

The predicted states take the form:

$$\widehat{\mathbf{x}}_{j|j-1} = \mathbf{f}\left(\widehat{\mathbf{x}}_{j-1|j-1}, \mathbf{u}_{j-1}, \mathbf{0}\right) \tag{9.25}$$

and the covariance matrix of the prediction errors:

$$\mathbf{P}_{j|j-1} = \mathbf{A}_j\ \mathbf{P}_{j-1|j-1}\ \mathbf{A}_j^T + \mathbf{V}_j\ \mathbf{Q}_{j-1}\ \mathbf{V}_j^T \tag{9.26}$$

9.2.3 Diagram of the extended Kalman filter

The extended Kalman filter is made of the equations 9.20, 9.21, 9.24, 9.25 and 9.26. The process is shown in figure 9.1. The first estimation is based on the initial conditions $\widehat{\mathbf{x}}_{0|0}$ and $\mathbf{P}_{0|0}$.

9.3 Disadvantages of the extended Kalman filter and filtering methods

One of the disadvantages of filtering methods is the fact that they consider the input variables as perfectly known; so one is not able to correct them or to take their uncertainties

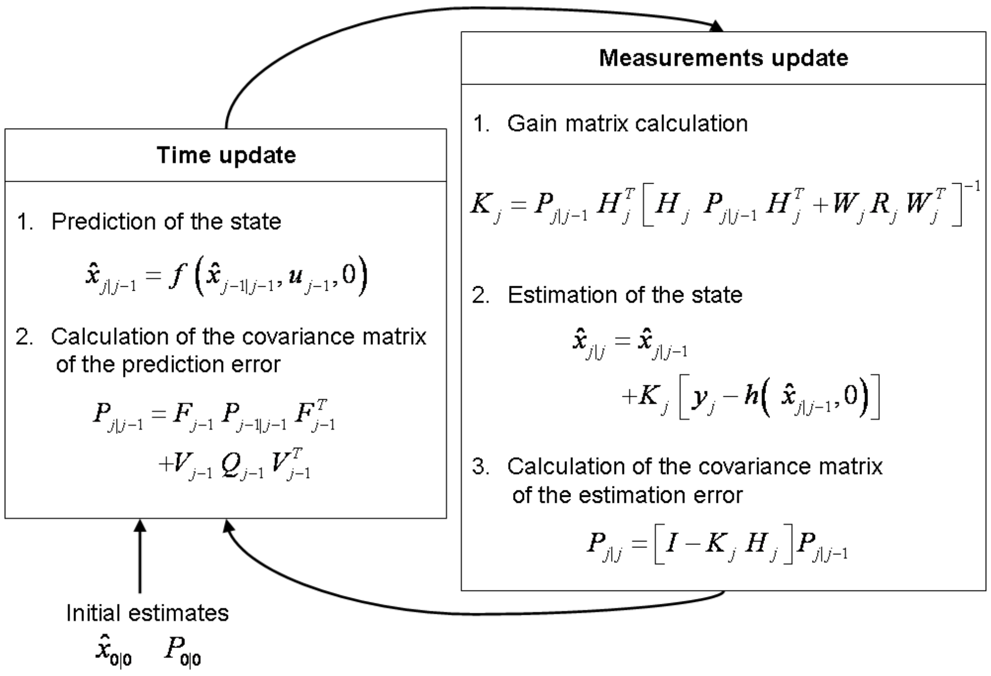

Measurements update
1. Gain matrix calculation
$K_j = P_{j|j-1} H_j^T \left[H_j P_{j|j-1} H_j^T + W_j R_j W_j^T \right]^{-1}$
2. Estimation of the state
$\hat{x}_{j|j} = \hat{x}_{j|j-1} + K_j \left[y_j - h(\hat{x}_{j|j-1}, 0) \right]$
3. Calculation of the covariance matrix of the estimation error
$P_{j|j} = \left[I - K_j H_j \right] P_{j|j-1}$
Time update
1. Prediction of the state
$\hat{x}_{j|j-1} = f(\hat{x}_{j-1|j-1}, u_{j-1}, 0)$
2. Calculation of the covariance matrix of the prediction error
$P_{j|j-1} = F_{j-1} P_{j-1|j-1} F_{j-1}^T + V_{j-1} Q_{j-1} V_{j-1}^T$
Initial estimates
$\hat{x}_{0|0}$ $P_{0|0}$

Figure 9.1: Diagram of the extended Kalman filter

into account. However all measurements are erroneous and it would be interesting to consider the uncertainties on those measurements to correct the other variables and estimate their a posteriori uncertainties.
Another disadvantage of filtering methods is that they don't take into account inequality constraints as well as the bounds on the filtered variables so they may appear out of bounds after filtering. The advantage of filtering is that it is much faster than the moving horizon methods.
In the case of very non linear differential equations, the convergence is not easy to achieve with the extended Kalman filter. Moreover, one needs the partial derivatives of the dynamic and of the observation model which can not always be achieved.
Considering all those disadvantages, we decided to turn towards another dynamic data reconciliation method based on the moving horizon idea.

Chapter 10

Moving-Horizon estimation

As indicated in chapter 8, solving the dynamic reconciliation problem would involve an increasing complex problem if all measurements collected since start-up are considered. The moving window techniques described in this chapter allow to reduce the size of the optimization problem because they only take into account the last measurements collected during a limited time frame. Those methods carry out an optimization for a part or all the measurements for each window. The data collected before the current window are not considered in the estimation for current time. For example, on figure 10.1, only the measurements carried out between the time t_0 and t_6 are reconciled.

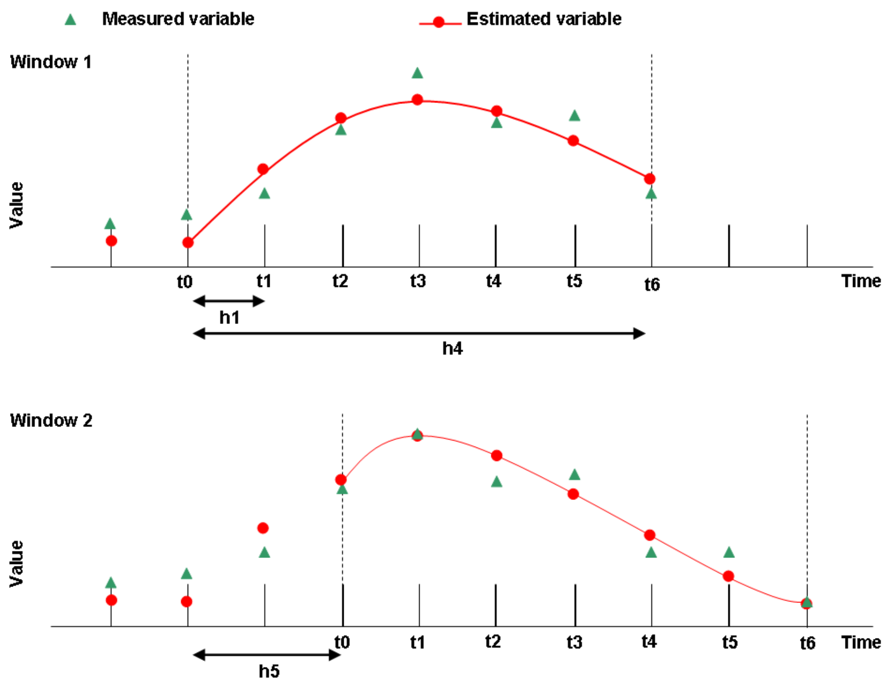


Figure 10.1: Reconciliation window

The most complex algorithms allow to treat variables that are not measured at the same

time or are sampled at different frequencies.
The size of the reconciliation horizon and the moving of the window are chosen in a way to ensure a sufficient temporal redundancy while keeping an optimization problem of reasonable size. Moreover, they depend on the studied problem and on the measurements frequency relative to the system dynamics. If measurements are very frequent, averaging the measurements may be used to maintain a non-linear problem of manageable dimension.
A window movement is described by three parameters as shown in figure 10.1:

- h_1 is the measurement frequency;
- h_4 is the window length;
- h_5 is the window movement between two successive optimizations: $h_5 \leq h_4$.

Those parameters are represented on figure 10.1. In this chapter, two moving window methods will be described:

- the explicit integration of the differential equations by means of the fourth order Runge-Kutta method;
- the discretization of the system equations thanks to orthogonal collocations.

10.1 Explicit integration method

In this moving horizon method, the dynamic reconciliation problem is formulated under the form of a general non-linear programming problem. The dynamic model is integrated explicitly on the reconciliation window thanks to the fourth order Runge-Kutta integration method, assuming that the initial conditions and the evolution of input variables (input variables are represented by linear interpolations) are known. This method is represented on figure 10.2 and can be decomposed in nine steps, which are described here after.

1. Variables classification: the variables are classified in three types, each of them being decomposed in two groups: the measured and the unmeasured variables:
    - the differential state variables, which are the integration variables that constitute the differential equations corresponding to mass and energy balances;
    - the input variables of the system: they can be the feed flow rate, the reflux flow rate, the temperature of an incoming stream,...
    - the algebraic state variables, which are calculated from the differential state variables and the input variables.
2. Initialisation of the window parameters: the three parameters of the window (described in the preceding paragraph) have to be specified in this step. In the case of explicit integration method, a fourth parameter has to be defined: the size of the

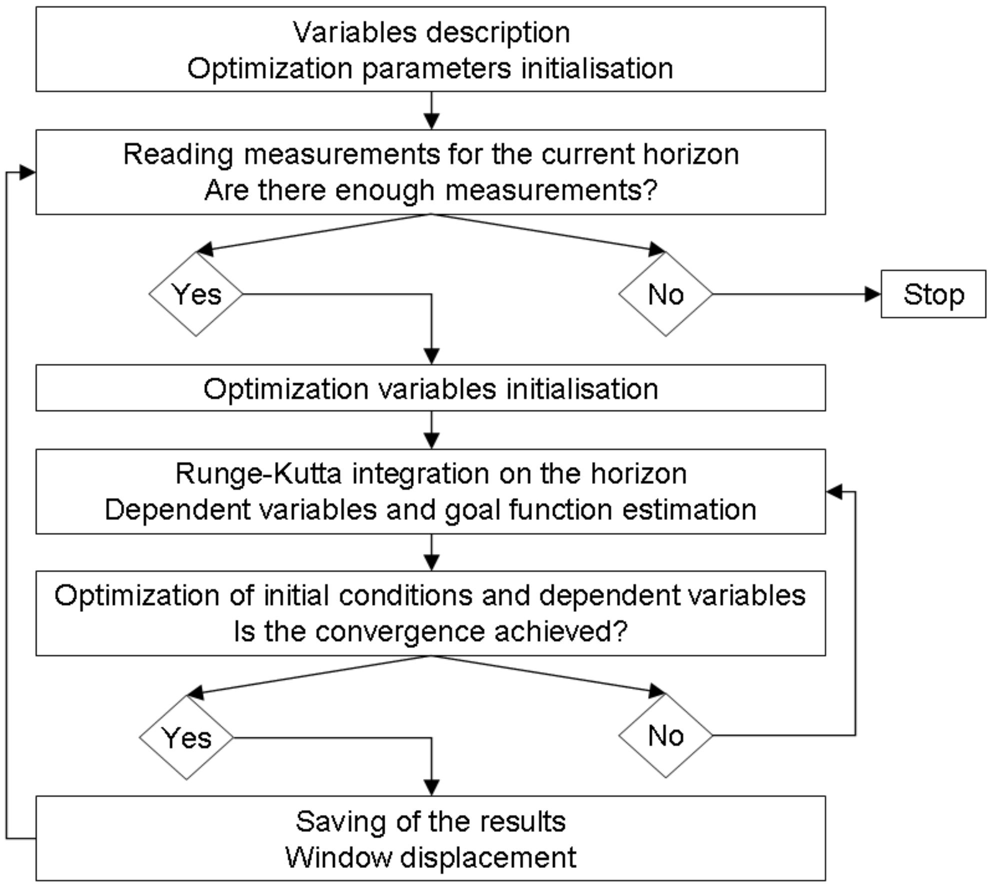

Variables description
Optimization parameters initialisation
Reading measurements for the current horizon
Are there enough measurements?
Yes
No
Stop
Optimization variables initialisation
Runge-Kutta integration on the horizon
Dependent variables and goal function estimation
Optimization of initial conditions and dependent variables
Is the convergence achieved?
Yes
No
Saving of the results
Window displacement

Figure 10.2: Flowchart of the explicit integration method

interpolation interval of the input variables h_2. If all the measurements are carried out at the same time, one can write: $h_2 = k_{1a}\ h_1$. Input variables are supposed to vary linearly in each interpolation interval. The size of the window h_4 has to be a multiple of the size of the interpolation interval: $h_4 = k_2\ h_2$. The window parameters are represented on figure 10.3.

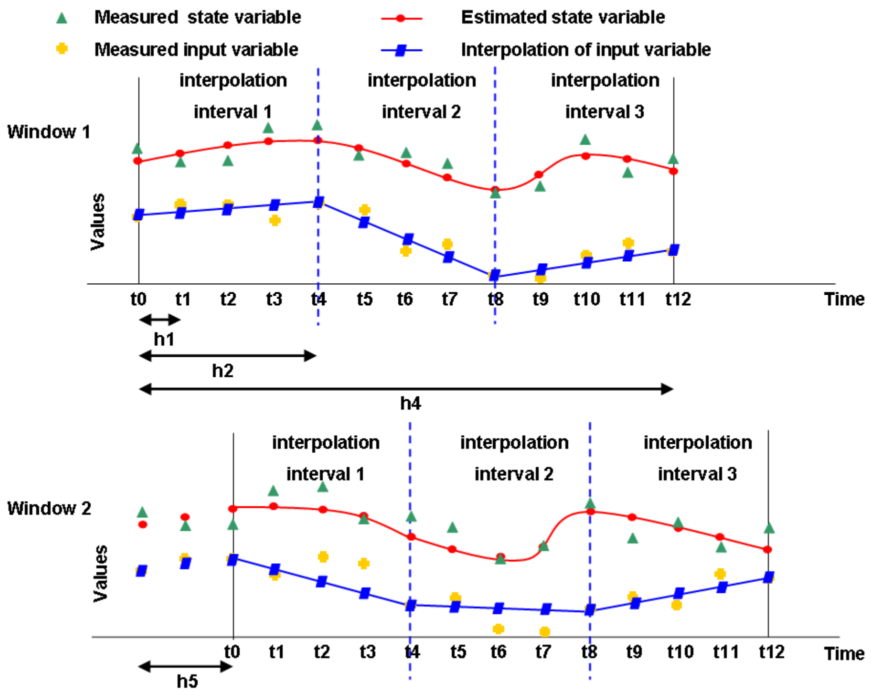


Figure 10.3: Parameters of the reconciliation window in the case of the explicit integration method

3. Reading the measurements: the measurements of the horizon are read in this third step. Their values can come directly from a plant or are simulated thanks to a dynamic simulation software. If there are not enough measurements, the program stops in the case of the simulation or waits for new measurements in the case of data coming from a plant.

4. Initialisation of the optimization variables: the variables that will be optimized are the values of the differential states variables at the initial time of the reconciliation horizon and the values of the input variables at the initial time of the reconciliation window and at the final times of all interpolation intervals of that window. Measurements are taken as initial values for the first window whereas estimates of the

previous window are taken in the other cases. Optimized variables are scaled to improve the efficiency of the optimization algorithm.

5. Integration of the differential equations of the process by the fourth order Runge-Kutta method: the principle of this method is to obtain an approached numerical solution of differential equations of the system.

6. Estimation of the algebraic variables: the algebraic variables are estimated at each integration step by Runge-Kutta.

7. Calculation of the goal function F_{goal} on the reconciliation horizon:

$$\begin{aligned} F_{goal} &= \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{x_{mes}}} \left(\frac{x_{i,j} - x_{i,j}^m}{\sigma_{x_{i,j}}} \right)^2 \\ &+ \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{z_{mes}}} \left(\frac{z_{i,j} - z_{i,j}^m}{\sigma_{z_{i,j}}} \right)^2 \\ &+ \sum_{j=time\ 0}^{time\ N} \sum_{i=1}^{n_{u_{mes}}} \left(\frac{u_{i,j} - u_{i,j}^m}{\sigma_{u_{i,j}}} \right)^2 \\ &+ \sum_{i=1}^{n_{x_{mes}}} \frac{\left(x_{i,0} - x_{i,0}^{CI}\right)^2}{R_{x_i}^2} + \sum_{i=1}^{n_{u_{mes}}} \frac{\left(u_{i,0} - u_{i,0}^{CI}\right)^2}{R_{u_i}^2} \end{aligned} \tag{10.1}$$

where

- $x_{i,j}$ is the estimation of the differential state variable i at measurement time j;
- $x_{i,j}^m$ is the measurement of the differential state variable i at measurement time j;
- $x_{i,0}^{CI}$ is the initial condition of the differential state variable i. It corresponds to the estimation of that variables in the previous reconciliation horizon.
- $z_{i,j}$ is the estimation of the algebraic state variable i at measurement time j;
- $z_{i,j}^m$ is the measurement of the algebraic state variable i at measurement time j;
- $u_{i,j}$ is the estimation of the input variable i at measurement time j;
- $u_{i,j}^m$ is the measurement of the input variable i at measurement time j;
- $u_{i,0}^{CI}$ is the initial condition of the input variable i. It corresponds to the estimation of that variable at the same time of the previous reconciliation horizon.
- $n_{x_{mes}}$ is the number of measured differential state variables;
- $n_{z_{mes}}$ is the number of measured algebraic state variables;
- $n_{u_{mes}}$ is the number of measured input variables;

- $\sigma_{x_{i,j}}$ is the standard deviation on the differential state variable i at measurement time j;
- $\sigma_{z_{i,j}}$ is the standard deviation on the algebraic state variable i at measurement time j;
- $\sigma_{u_{i,j}}$ is the standard deviation on the input variable i at measurement time j;
- R_{x_i} is the relaxation factor on differential state variable i;
- R_{u_i} is the relaxation factor on input variable i;

The relaxation terms of the goal function have for objective to ensure that state and input variables at the beginning of the reconciliation horizon are not too different from the ones obtained during the preceding reconciliation.

8. Optimization of the initial conditions and the coefficients of the polynomes describing the independent variables: this optimization has first been carried out by Davidon's algorithm (Davidon, 1975) for small size problems. For larger size problems a successive quadratic programming (SQP) method has been used (Kyriakopoulou, 1997). Davidon's algorithm is an optimization algorithm without line search, but with evaluation of the goal function gradient by divided differences. This method uses the information contained in the second order derivatives (an approximation of the Hessian matrix is carried out) to improve the search by calculating a step length.

   The SQP algorithm that has been used is an interior point method developed by Kyriakopoulou (Kyriakopoulou, 1997). SQP algorithms are based on a quadratic approximation of the non linear goal function and a linearisation of constraints equations around the current point:

$$\left[\triangle f\left(\mathbf{x}^0\right)\right]^T\ \Delta\mathbf{x} + (\Delta\mathbf{x})^T\ \mathbf{Q}\ \Delta\mathbf{x} \tag{10.2}$$

$$\mathbf{h}_j(x) \approx \mathbf{h}_j\left(\mathbf{x}^0\right) + \left[\triangle\mathbf{h}_j\left(\mathbf{x}^0\right)\right]^T\ \Delta\mathbf{x} \tag{10.3}$$

   where $\mathbf{Q}$ is the approximation of the Hessian matrix of the following Lagrange function:

$$L = f - \boldsymbol{\lambda}^T\ \mathbf{g} - \boldsymbol{\mu}^T\ \mathbf{h} \tag{10.4}$$

   with $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ the Lagrange multipliers.

9. Saving of the reconciliation results and the moving of the window. Which results keeping is a question whose answer depends on the use that must be done of the data. In the case of process control, one will need the estimated values corresponding to the last measurement time as soon as possible to be able to react fast. However, if one desires the better estimates of the variables, one should keep the estimates obtained after several reconciliations of the measurement time, so that the reconciled result obtained when the data lies close to the middle of the window will probably be kept.

The main disadvantage of this method is that, unlike the filtering algorithms, there is no explicit method to calculate a posteriori variances from the results. If this method is used in the future, an algorithm allowing to calculate them must be developed.

10.2 Method based on orthogonal collocations

In this method, the differential equations of the system are solved by means of orthogonal collocations. State variables and their derivatives are represented by polynomials. Differential equations are transformed into algebraic equations whose unknowns are the polynomial coefficients. Those algebraic equations must be satisfied at several discrete times chosen to minimize the numerical approximation errors.
The main advantage of this method is that it will be possible to calculate a posteriori variances by modifying the theory that is used in steady-state data reconciliation (Heyen et al., 1996).
In this section, the collocations constraints and the different steps of the method are described. Examples of polynomial approximations and the way to choose the collocation nodes are explained in annex C.

10.2.1 Description of the moving window algorithm

The algorithm allowing to apply the orthogonal collocations method to a moving window is defined by five parameters in the case of orthogonal collocations. They are represented on figure 10.4.
The parameters h_1, h_2, h_4 and h_5 of the window are the same as the ones defined previously in this chapter. The size h_3 of discretization interval of the differential state variables is an additional parameter. This interval has been chosen as a multiple of the measurement frequency: $h_3 = k_{1b}\ h_1$. The size of the window h_4 has to be a multiple of the size of the discretization interval: $h_4 = k_3\ h_3$. The last parameter that has to be specified is the order of Lagrange polynomials.
The times θ_k represent the collocation times of the different discretization intervals of the window. The sequential algorithm is written so that the discretization and the interpolation intervals have the same size. The distinction between them must only be done in the case of the simultaneous algorithm. The times t_j are the measurement times. The choice of the window parameters depends on the studied problem. One has to take into account the following remarks:

- The more the window increases, the more the number of opimization variables increase, and so do the computing time. So, the size of the window has to be chosen in a way to keep reasonable computing times. Moreover, if the size of the window is too important, convergence problems may occur.

- Contrary, one has to keep redundancy in the problem, so that the size of the reconciliation window has to contain enough measurement times.

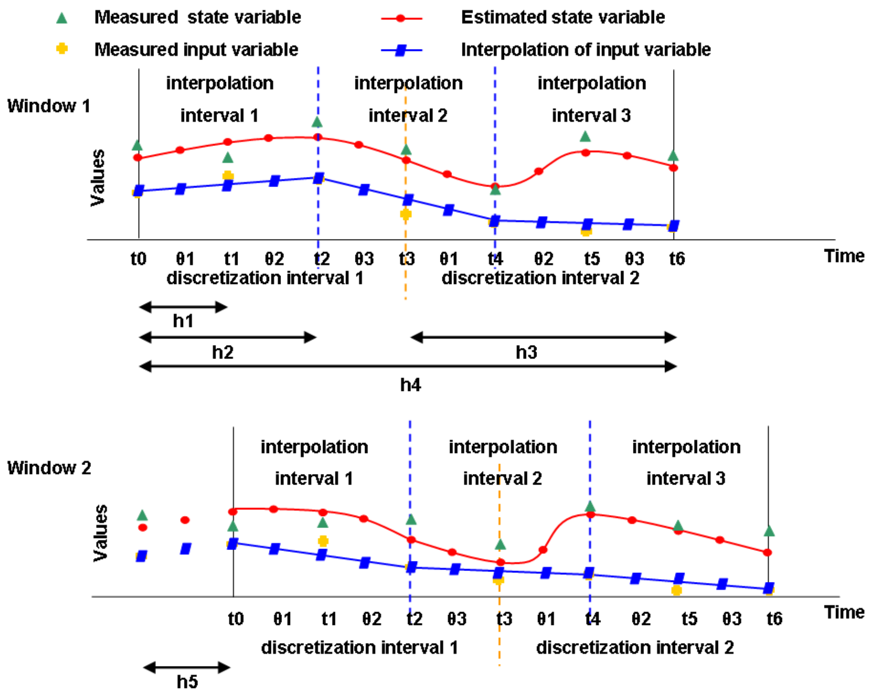


Figure 10.4: Parameters of the reconciliation window in the case of orthogonal collocations

- The size of the discretization interval and the order of the Lagrange polynomials must be chosen to allow to follow the process while keeping redundancy.
- For the frequency of measurements, one can take all measurement times, or if the process variables don't vary very much, calculate mean values on several measurement times. This second proposition allows to make a first filter on the variables by reducing the effect of the noise.

10.2.2 Constraints of the optimization problem

In the case of orthogonal collocations, the goal function is submitted to five sets of constraints:

- the link equations: they are relations between all process variables. Those constraints have to be satisfied as well at the measurements times as at the collocations nodes:

$$\mathbf{A} = f\left(t_j, \mathbf{x}, \mathbf{z}, \mathbf{u}\right) = \mathbf{0} \qquad \forall t_j \tag{10.5}$$

$$\mathbf{A}^c = f^c\left(\theta_k, \mathbf{x}^c, \mathbf{z}^c, \mathbf{u}^c\right) = \mathbf{0} \qquad \forall \theta_k \tag{10.6}$$

- the relations between the differential state variables and the Lagrange interpolation polynomials at all measurement times of the mowing horizon except at the initial times of the discretization intervals t_{CI}:

$$B_{i,j} = x_{i,j} - \sum_{k=0}^{n_\theta} l_k(t_j)\; x^c_{i,k} = 0 \qquad \forall i,\ \forall t_j \neq t_{CI} \tag{10.7}$$

- the linear interpolations of the values of input variables between times t_{ini} and t_f of the interpolation horizon at the other times of that horizon:

$$C_{i,j} = u_{i,j} - u_{i,t_{ini}} - \frac{t_j - t_{ini}}{t_f - t_{ini}}\left(u_{i,t_f} - u_{i,t_{ini}}\right) = 0 \qquad \forall i,\ \forall t_j \neq t_{ini},\ t_f \tag{10.8}$$

$$C^c_{i,j} = u^c_{i,k} - u_{i,t_{ini}} - \frac{\theta_k - t_{ini}}{t_f - t_{ini}}\left(u_{i,t_f} - u_{i,t_{ini}}\right) = 0 \qquad \forall i,\ \forall \theta_k \tag{10.9}$$

- the residuals of the differential state equations at all collocation nodes:

$$D_{i,j} = \sum_{s=0}^{n_\theta} \dot{l}_s(\theta_k)\; x^c_{i,\theta_s} - g\left(\theta_k, \mathbf{x}^c_k, \mathbf{z}^c_k, \mathbf{u}^c_k\right) = 0 \qquad \forall i,\ \forall \theta_k \neq t_{ini} \tag{10.10}$$

- the continuity constraints of the differential state variables between two discretization intervals:

$$E_{i,j} = \left[\sum_{k=0}^{n_\theta} {}_k(t_f)\; x^c_{i,k}\right]_{t_f, discr\ int\ q-1} - \left[\sum_{k=0}^{n_\theta} l_k(t_{ini})\; x^c_{i,k}\right]_{t_{ini}, discr\ int\ q} = 0 \qquad \forall i \tag{10.11}$$

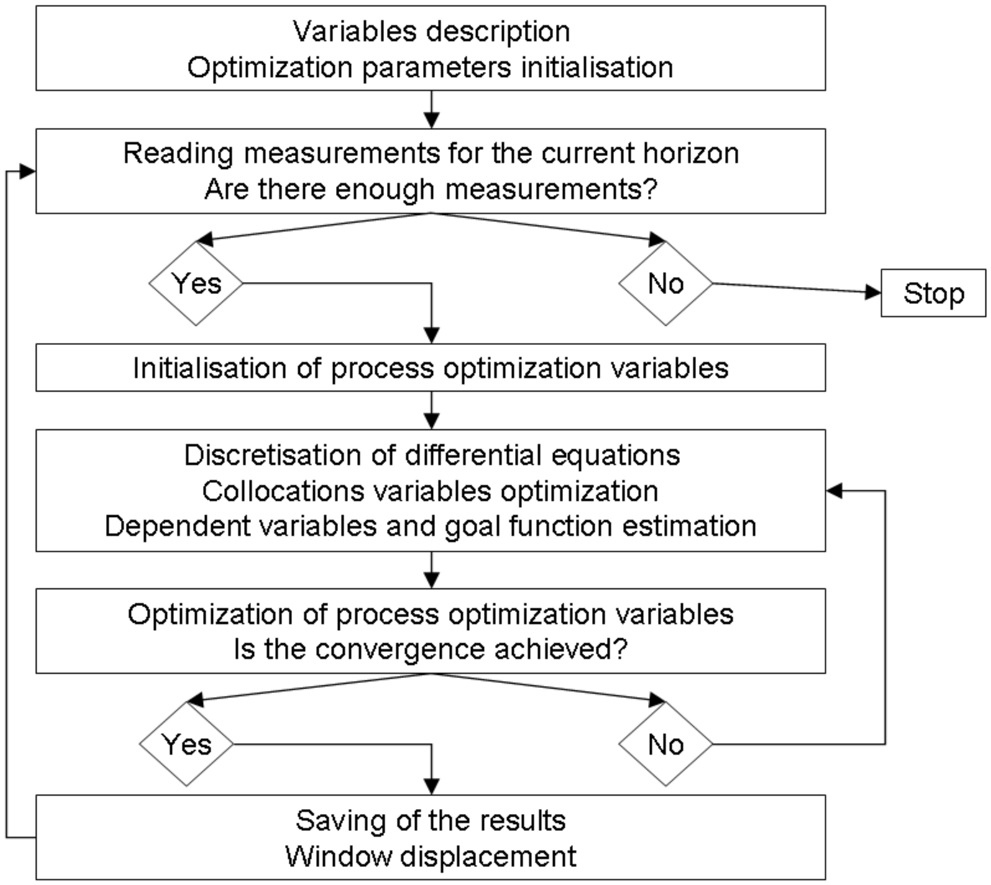

Variables description
Optimization parameters initialisation
Reading measurements for the current horizon
Are there enough measurements?
Yes
No
Stop
Initialisation of process optimization variables
Discretisation of differential equations
Collocations variables optimization
Dependent variables and goal function estimation
Optimization of process optimization variables
Is the convergence achieved?
Yes
No
Saving of the results
Window displacement

Figure 10.5: Flowchart of the orthogonal collocations method: sequential algorithm

10.2.3 Description of the sequential algorithm

In the sequential algorithm, process variables and collocation variables are optimized separately: two calculation loops are embedded. Collocation variables are calculated in the inside loop by means of Broyden's algorithm (Broyden, 1965) and process variables are estimated in the outside loop thanks to Davidon's algorithm (Davidon, 1975). Those two numerical methods can be used for small size problems. For larger size problems, a dog-leg method (Chen and Stadherr, 1981) and a SQP algorithm (Kyriakopoulou, 1997) must respectively replace Broyden and Davidon's algorithms.
The sequential algorithm is composed of the nine following steps and is represented on figure 10.5.

1. Variables classification: the variables are classified in three types as in the explicit integration method.

2. Initialisation of the window parameters: the five parameters of the window h_1, h_2, h_3, h_4 and h_5 have to be specified in this step.

3. Reading of the measurements: this step is the same as in the explicit integration method.

4. Initialisation of the optimization variables of the process: the variables that will be optimized are the values of the differential state variables at the collocation nodes of the reconciliation horizon and the values of the input variables at the initial time of the reconciliation window and at the final times of all interpolation intervals of that window. Measurements are taken as initial values for the first window whereas estimates of the previous window are taken in the other cases.
   To ensure that each optimization variable will have a similar weight, they are multiplied by a scaling factor before being optimized.

5. Discretization of the differential equations of the process: the collocation nodes are chosen as the zeros of Jacobi orthogonal polynomial and differential state variables and their derivatives are interpolated by Lagrange polynomials. Constraints **D** and **E** are written and solved using to Broyden's algorithm (Broyden, 1965) if the number of variables is not too important. For larger size problem, a dogleg method (Chen and Stadherr, 1981) is used.
   Once the collocation variables are obtained, one can estimate the values of differential state variables at all measurement times.

6. Estimation of the algebraic variables: the algebraic variables are deduced from the estimates of differential state variables and the interpolation of input variables.

7. Calculation of the goal function F_{goal} on the reconciliation horizon: the goal function is the same as for the explicit integration method 10.1.

8. Optimization of process optimization variables: this optimization has first been carried out by the Davidon algorithm (Davidon, 1975) for small size problems. For larger size problems a successive quadratic programming (SQP) method has been used (Kyriakopoulou, 1997).
   As long as the convergence is not achieved, one must go back to step five and use the new values of the process optimization variables. Otherwise, one can continue to the next step.

9. Saving of the reconciliation results and moving of the window.

10.2.4 Description of the simultaneous algorithm

Contrary to the sequential algorithm, the simultaneous algorithm optimizes in one step all variables: process variables (the values of the differential state variables at the initial time of the optimization window, the values of the input variables at the initial time of the reconciliation window and at the final times of all interpolation intervals of that window) and the collocations variables (the values of the differential state variables at the collocation nodes).

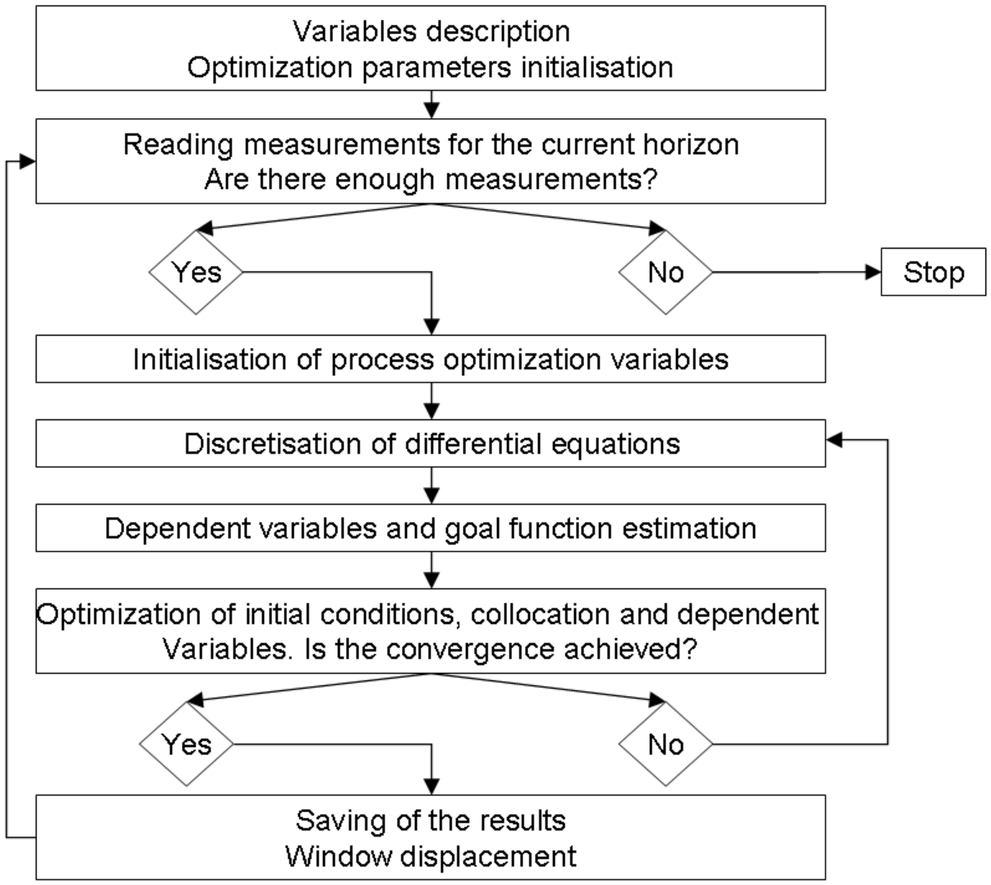

Variables description
Optimization parameters initialisation
Reading measurements for the current horizon
Are there enough measurements?
Yes
No
Stop
Initialisation of process optimization variables
Discretisation of differential equations
Dependent variables and goal function estimation
Optimization of initial conditions, collocation and dependent Variables. Is the convergence achieved?
Yes
No
Saving of the results
Window displacement

Figure 10.6: Flowchart of the orthogonal collocations method: simultaneous algorithm

The simultaneous algorithm is composed of nine steps and is represented on figure 10.6.

1-4 The four first steps are the same as for the sequential algorithm.

5 Discretization of the differential equations of the process: the nodes of collocations are chosen as the zeros of Jacobi orthogonal polynomial and differential state variables and their derivatives are approximated by Lagrange polynomials.
One can interpolate the values of differential state variables at all measurement times from their values at the collocation nodes.
For the first optimization, the initial values of collocation variables are the values of the measurements or their linear interpolation at the collocation times. For later optimizations overlapping, the estimated values of the previous window are taken for the common part of the window. For the remaining of the window containing new measurements, one proceeds as for the first window.

6 Estimation of the algebraic variables: the algebraic variables are deduced from the estimates of differential state variables and the interpolation of input variables.

7 Calculation of the goal function F_{goal} on the reconciliation horizon: the goal function is the same as for the sequential method 10.1.

8 Optimization of all optimization variables: this optimization is carried out by a successive quadratic programming (SQP) method (Kyriakopoulou, 1997).
If the convergence is not achieved, one must go back to step five. Otherwise, one can continue to the next step.

9 Saving of the reconciliation results and moving of the window.

As it will be illustrated in the next chapter for the first example, the simultaneous algorithm gives better results than the sequential one. Those results agree with the study of Biegler (Biegler, 1984).

Chapter 11

Calculation of a posteriori variances

The method for estimating a posteriori variances in the case of dynamic data reconciliation described in this chapter has been inspired from the one described in (Heyen et al., 1996) for the stationary case.
If orthogonal collocations are used, the dynamic reconciliation problem can be formulated this way:

$$\min_{\substack{x^c_{i,k} \\ \left[u_{i,ini},u_{i,f}\right]_{interp\ int\ 1} \\ \left[u_{i,f}\right]_{interp\ int\ s} \\ \forall k,\ \forall s\neq 1,\ \forall i}} \left\{ \begin{array}{l} \displaystyle\sum_{j=0}^{n_{t_{mes}}} \sum_{i=1}^{n_{x_{mes}}} \left(x_{i,j} - x^m_{i,j}\right)^2 W_{x_{i,i,j}} \\ \displaystyle+\sum_{j=0}^{n_{t_{mes}}} \sum_{i=1}^{n_{z_{mes}}} \left(z_{i,j} - z^m_{i,j}\right)^2 W_{z_{i,i,j}} \\ \displaystyle+\sum_{j=0}^{n_{t_{mes}}} \sum_{i=1}^{n_{u_{mes}}} \left(u_{i,j} - u^m_{i,j}\right)^2 W_{u_{i,i,j}} \\ \displaystyle+\sum_{i=1}^{n_x} \left(x_{i,0} - x^{CI}_{i,0}\right)^2 R_{x_{i,i}} \\ \displaystyle+\sum_{i=1}^{n_u} \left(u_{i,0} - u^{CI}_{i,0}\right)^2 R_{u_{i,i}} \end{array} \right. \tag{11.1}$$

where

- $x_{i,j}$ is the estimation of the differential state variable i at measurement time j;
- $x^m_{i,j}$ is the measurement of the differential state variable i at measurement time j;
- $x^{CI}_{i,0}$ is the initial condition of the differential state variable i. It corresponds to the estimation of that variables at the final time of the previous reconciliation horizon.
- $z_{i,j}$ is the estimation of the algebraic state variable i at measurement time j;
- $z^m_{i,j}$ is the measurement of the algebraic state variable i at measurement time j;

- $u_{i,j}$ is the estimation of the input variable i at measurement time j;
- $u_{i,j}^m$ is the measurement of the input variable i at measurement time j;
- $u_{i,0}^{CI}$ is the initial condition of the input variable i. It corresponds to the estimation of that variable at the same time of the previous reconciliation horizon.
- $x_{i,k}^c$ is the collocation variable i at collocation time k;
- $z_{i,k}^c$ is the value of algebraic state variable i at collocation time k;
- $u_{i,k}^c$ is the value of input variable i at collocation time k;
- $\mathbf{W}_x$ is the weight matrix of the differential state variables;
- $\mathbf{W}_z$ is the weight matrix of the algebraic state variables;
- $\mathbf{W}_u$ is the weight matrix of the input state variables;
- $\mathbf{R}_x$ is the relaxation factor of the differential state variables at the initial time of the moving horizon;
- $\mathbf{R}_u$ is the relaxation factor of the input variables at the initial time of the moving horizon.

This objective function is submitted to the five kinds of constraints described in section 10.2:

$$\mathbf{A} = f\left(t_j, \mathbf{x}, \mathbf{z}, \mathbf{u}\right) = \mathbf{0} \qquad \forall t_j \tag{11.2}$$

$$\mathbf{A}^c = f^c\left(\theta_k, \mathbf{x}^c, \mathbf{z}^c, \mathbf{u}^c\right) = \mathbf{0} \qquad \forall \theta_k \tag{11.3}$$

$$B_{i,j} = x_{i,j} - \sum_{k=0}^{n_\theta} l_k(t_j)\ x_{i,k}^c = 0 \qquad \forall i,\ \forall t_j \neq t_{CI} \tag{11.4}$$

$$C_{i,j} = u_{i,j} - u_{i,t_{ini}} - \frac{t_j - t_{ini}}{t_f - t_{ini}}\left(u_{i,t_f} - u_{i,t_{ini}}\right) = 0 \qquad \forall i,\ \forall t_j \neq t_{ini},\ t_f \tag{11.5}$$

$$C_{i,j}^c = u_{i,k}^c - u_{i,t_{ini}} - \frac{\theta_k - t_{ini}}{t_f - t_{ini}}\left(u_{i,t_f} - u_{i,t_{ini}}\right) = 0 \qquad \forall i,\ \forall \theta_k \tag{11.6}$$

$$D_{i,j} = \sum_{s=0}^{n_\theta} \dot{l}_s(\theta_k)\ x_{i,\theta_s}^c - g\left(\theta_k, \mathbf{x}_k^c, \mathbf{z}_k^c, \mathbf{u}_k^c\right) = 0 \qquad \forall i,\ \forall \theta_k \neq t_{ini} \tag{11.7}$$

$$E_{i,j} = \left[\sum_{k=0}^{n_\theta} l_k(t_f)\ x_{i,k}^c\right]_{t_f, discr\ int\ q-1} - \left[\sum_{k=0}^{n_\theta} l_k(t_{ini})\ x_{i,k}^c\right]_{t_{ini}, discr\ int\ q} = 0 \qquad \forall i \tag{11.8}$$

This constrained minimization problem can be transformed into an unconstrained minimization problem using the vector of Lagrange multipliers $\mathbf{\Lambda}$:

$$\min_{\substack{\mathbf{x}_{i,k}^{c} \\ [\mathbf{u}_{i,ini},\mathbf{u}_{i,f}]_{interp\ int\ 1} \\ [\mathbf{u}_{i,f}]_{interp\ int\ s} \\ \mathbf{\Lambda}_A,\mathbf{\Lambda}_{A^c},\mathbf{\Lambda}_B,\mathbf{\Lambda}_C,\mathbf{\Lambda}_{C^c},\mathbf{\Lambda}_D,\mathbf{\Lambda}_E \\ \forall k,\ \forall s\neq 1,\ \forall i}} \mathbf{L}(\mathbf{x},\mathbf{x}^m,\mathbf{z},\mathbf{z}^m,\mathbf{u},\mathbf{u}^m,\mathbf{\Lambda}_A,\mathbf{\Lambda}_{A^c},\mathbf{\Lambda}_B,\mathbf{\Lambda}_C,\mathbf{\Lambda}_{C^c},\mathbf{\Lambda}_D,\mathbf{\Lambda}_E) \tag{11.9}$$

with

$$\begin{aligned}
&\mathbf{L}(\mathbf{x},\mathbf{x}^m,\mathbf{z},\mathbf{z}^m,\mathbf{u},\mathbf{u}^m,\mathbf{\Lambda}_A,\mathbf{\Lambda}_{A^c},\mathbf{\Lambda}_B,\mathbf{\Lambda}_C,\mathbf{\Lambda}_{C^c},\mathbf{\Lambda}_D,\mathbf{\Lambda}_E) \\
&= \sum_{j=0}^{n_{tmes}} \left[\sum_{i=1}^{n_{xmes}} \left(x_{i,j}-x_{i,j}^{m}\right)^2 W_{x_{i,i,j}} + \sum_{i=1}^{n_{zmes}} \left(z_{i,j}-z_{i,j}^{m}\right)^2 W_{z_{i,i,j}} + \sum_{i=1}^{n_{umes}} \left(u_{i,j}-u_{i,j}^{m}\right)^2 W_{u_{i,i,j}} \right] \\
&+\sum_{i=1}^{n_x} \left(x_{i,0}-x_{i,0}^{CI}\right)^2 R_{x_{i,i}} + \sum_{i=1}^{n_u} \left(u_{i,0}-u_{i,0}^{CI}\right)^2 R_{u_{i,i}} \\
&+\sum_{j=0}^{n_{tmes}} \sum_{i=1}^{n_A} \Lambda_{A_{i,j}}^{T}\ f\left(t_j,\mathbf{x},\mathbf{z},\mathbf{u}\right) \\
&+\sum_{q=1}^{n_{discr\ int}} \left[\sum_{k=0}^{n_\theta} \sum_{i=1}^{n_{A^c}} \Lambda_{A_{i,k}^{c}}^{T}\ f^c\left(\theta_k,\mathbf{x}^c,\mathbf{z}^c,\mathbf{u}^c\right) \right]_q \\
&+\sum_{q=1}^{n_{discr\ int}} \left[\sum_{j\in q} \sum_{i=1}^{n_x} \Lambda_{B_{i,j}}^{T} \left(x_{i,j} - \sum_{k=0}^{n_\theta} l_k(t_j)\ x_{i,k}^{c} \right) \right]_q \\
&+\sum_{s=1}^{n_{interp\ int}} \left[\sum_{j\ \mid\ t_j\in s;\ t_j\neq t_{ini},\ t_f}\ \sum_{i=1}^{n_u} \Lambda_{C_{i,j}}^{T} \left(u_{i,j} - u_{i,t_{ini}} - \frac{t_j - t_{ini}}{t_f - t_{ini}} \left(u_{i,t_f} - u_{i,t_{ini}}\right) \right) \right]_s \\
&+\sum_{s=1}^{n_{interp\ int}} \left[\sum_{k\ \mid\ \theta_k\in s}\ \sum_{i=1}^{n_{u^c}} \Lambda_{C_{i,k}^{c}}^{T} \left(u_{i,k}^{c} - u_{i,t_{ini}} - \frac{\theta_k - t_{ini}}{t_f - t_{ini}} \left(u_{i,t_f} - u_{i,t_{ini}}\right) \right) \right]_s \\
&+\sum_{s=1}^{n_{discr\ int}} \left[\sum_{k=1}^{n_\theta} \sum_{i=1}^{n_{x^c}} \Lambda_{D_{i,k}}^{T} \left(\sum_{j=0}^{n_\theta} \dot{l}_j(\theta_k)\ x_{i,\theta_j}^{c} - g\left(\theta_k,\mathbf{x}_k^c,\mathbf{z}_k^c,\mathbf{u}_k^c\right) \right) \right]_s \\
&+\sum_{q=1}^{n_{discr\ int}-1} \sum_{i=1}^{n_x} \Lambda_{E_{i,q}}^{T} \left(\left[\sum_{k=0}^{n_\theta} l_k(t_f)\ x_{i,k}^{c} \right]_{t_f,discr\ int\ q} - \left[\sum_{k=0}^{n_\theta} l_k(t_{ini})\ x_{i,k}^{c} \right]_{t_{ini},discr\ int\ q+1} \right)
\end{aligned} \tag{11.10}$$

The solution must verify the optimality conditions:

$$\frac{\partial\ L}{\partial\ x_{i,j}} = P_{x_{i,j}} \left(x_{i,j} - x^m_{i,j}\right) + \sum_{s=1}^{n_A} \left(\frac{\partial A_{s,j}}{\partial x_{i,j}}\right)^T \ \Lambda_{A_{s,j}} + \left(\frac{\partial B_{i,j}}{\partial x_{i,j}}\right)^T \ \Lambda_{B_{i,j}} = 0 \qquad \forall i,\ \forall j$$

$$\frac{\partial\ L}{\partial\ z_{i,j}} = P_{z_{i,j}} \left(z_{i,j} - z^m_{i,j}\right) + \sum_{s=1}^{n_A} \left(\frac{\partial A_{s,j}}{\partial z_{i,j}}\right)^T \ \Lambda_{A_{s,j}} = 0 \qquad \forall i,\ \forall j$$

$$\begin{aligned}\frac{\partial\ L}{\partial\ u_{i,j}} &= P_{u_{i,j}} \left(u_{i,j} - u^m_{i,j}\right) + \sum_{s=1}^{n_A} \left(\frac{\partial A_{s,j}}{\partial u_{i,j}}\right)^T \ \Lambda_{A_{s,j}} \\ &\quad + \sum_{s=1}^{n_{interp\ int}} \left[\sum_{\substack{l\ \mid\ t_l \in s \\ t_j \neq t_{ini},\ t_f}} \left(\frac{\partial C_{i,l}}{\partial u_{i,j}}\right)^T \ \Lambda_{C_{i,l}} + \sum_{r\ \mid\ \theta_r \in s} \left(\frac{\partial C^c_{i,r}}{\partial u_{i,j}}\right)^T \ \Lambda_{C^c_{i,r}} \right]_s \\ &\quad = 0 \qquad \forall i,\ \forall j\end{aligned}$$

$$\begin{aligned}\left[\frac{\partial\ L}{\partial\ x^c_{i,k}}\right]_{discr\ int\ q} &= \left[\sum_{r=1}^{n_A} \left(\frac{\partial A^c_{r,k}}{\partial x^c_{i,k}}\right)^T \ \Lambda_{A^c_{r,k}} + \sum_{r\ \mid\ r \in q} \left(\frac{\partial B_{i,r}}{\partial x^c_{i,k}}\right)^T \ \Lambda_{B_{i,r}} \right. \\ &\quad \left. + \sum_{r=0}^{n_\theta} \left(\frac{\partial D_{i,r}}{\partial x^c_{i,k}}\right)^T \ \Lambda_{D_{i,r}} + \left(\frac{\partial E_{i,s}}{\partial x^c_{i,k}}\right)^T \ \Lambda_{E_{i,s}} \right]_{discr\ int\ q} = 0 \qquad \forall i,\ \forall k\end{aligned}$$

$$\frac{\partial\ L}{\partial\ z^c_{i,k}} = \sum_{s=1}^{n_{A^c}} \left(\frac{\partial A^c_{s,k}}{\partial z^c_{i,k}}\right)^T \ \Lambda_{A^c_{s,k}} + \sum_{q=1}^{n_x} \left(\frac{\partial D_{q,k}}{\partial z^c_{i,k}}\right)^T \ \Lambda_{D_{q,k}} = 0 \qquad \forall i,\ \forall k$$

$$\begin{aligned}\frac{\partial\ L}{\partial\ u^c_{i,k}} &= \sum_{s=1}^{n_{A^c}} \left(\frac{\partial A^c_{s,k}}{\partial u^c_{i,k}}\right)^T \ \Lambda_{A^c_{s,k}} + \left(\frac{\partial C^c_{i,k}}{\partial u^c_{i,k}}\right)^T \ \Lambda_{C^c_{i,k}} \\ &\quad + \sum_{q=1}^{n_x} \left(\frac{\partial D_{q,k}}{\partial u^c_{i,k}}\right)^T \ \Lambda_{D_{q,k}} = 0 \qquad \forall i,\ \forall k\end{aligned}$$

$$\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_A} = A = 0$$

$$\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_{A^c}} = A^c = 0$$

$$\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_B} = B = 0$$

$$\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_C} = C = 0$$

$$\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_{C^c}} = C^c = 0$$

$$\begin{aligned}\frac{\partial\ L}{\partial\ \mathbf{\Lambda}_D} &= D = 0\\ \frac{\partial\ L}{\partial\ \mathbf{\Lambda}_E} &= E = 0\end{aligned} \tag{11.11}$$

with

$$\begin{array}{lll} P_{x_{i,i,0}} = W_{x_{i,i,0}} + R_{x_{0,0}} & \forall i \\ P_{x_{i,i,j}} = W_{x_{i,i,j}} & \forall i,\ \forall j \neq 0 \\ P_{z_{i,i,j}} = W_{z_{i,i,j}} & \forall i,\ \forall j \\ P_{u_{i,i,0}} = W_{u_{i,i,0}} + R_{u_{0,0}} & \forall i \\ P_{u_{i,i,j}} = W_{u_{i,i,j}} & \forall i,\ \forall j \neq 0 \end{array} \tag{11.12}$$

The seven last optimality conditions can be linearized this way:

$$\begin{aligned}
\frac{\partial\ L}{\partial\ \Lambda_{A_{r,j}}} &= \sum_{i=1}^{n_x} \frac{\partial A_{r,j}}{\partial x_{i,j}}\ x_{i,j} + \sum_{i=1}^{n_z} \frac{\partial A_{r,j}}{\partial z_{i,j}}\ z_{i,j} + \sum_{i=1}^{n_u} \frac{\partial A_{r,j}}{\partial u_{i,j}}\ u_{i,j} + F_{r,j} \qquad \forall j,\ \forall r\\
\left[\frac{\partial\ L}{\partial\ \Lambda_{A^c_{r,j}}}\right]_{discr\ int\ q} &= \left[\sum_{i=1}^{n_{xc}} \frac{\partial A^c_{r,j}}{\partial x^c_{i,j}}\ x^c_{i,j} + \sum_{i=1}^{n_{zc}} \frac{\partial A^c_{r,j}}{\partial z^c_{i,j}}\ z^c_{i,j} \right.\\
&\qquad \left. + \sum_{i=1}^{n_{uc}} \frac{\partial A^c_{r,j}}{\partial u^c_{i,j}}\ u^c_{i,j} + F^c_{r,j}\right]_{discr\ int\ q} \qquad \forall j,\ \forall r,\ \forall q\\
\left[\frac{\partial\ L}{\partial\ \Lambda_{B_{i,j}}}\right]_{discr\ int\ q} &= \left[\frac{\partial B_{i,j}}{\partial x_{i,j}}\ x_{i,j} + \sum_{k=0}^{n_\theta} \frac{\partial B_{i,j}}{\partial x^c_{i,k}}\ x^c_{i,k}\right]_{discr\ int\ q} \qquad \forall i,\ \forall j \in q\\
\frac{\partial\ L}{\partial\ \Lambda_{C_{i,j}}} &= \sum_{r=0}^{n_{tmes}} \frac{\partial C_{i,j}}{\partial u_{i,r}}\ u_{i,r} \qquad \forall i,\ \forall j\\
\left[\frac{\partial\ L}{\partial\ \Lambda_{C^c_{i,k}}}\right]_{discr\ int\ q} &= \left[\sum_{r\ |\ r\in q} \frac{\partial C^c_{i,k}}{\partial u_{i,r}}\ u_{i,r} + \frac{\partial C^c_{i,k}}{\partial u^c_{i,k}}\ u^c_{i,k}\right]_{discr\ int\ q} \qquad \forall i,\ \forall j,\ \forall k,\ \forall q\\
\left[\frac{\partial\ L}{\partial\ \Lambda_{D_{i,k}}}\right]_{discr\ int\ q} &= \left[\sum_{r=0}^{n_\theta} \frac{\partial D_{i,k}}{\partial x^c_{i,r}}\ x^c_{i,r} + \frac{\partial D_{i,k}}{\partial z^c_{i,k}}\ z^c_{i,k}\right.\\
&\qquad \left. + \frac{\partial D_{i,k}}{\partial u^c_{i,k}}\ u^c_{i,k} + G_{i,k}\right]_{discr\ int\ q} \qquad \forall i,\ \forall k,\ \forall q\\
\frac{\partial\ L}{\partial\ \Lambda_{E_{i,q}}} &= \left[\sum_{r=0}^{n_\theta} \frac{\partial E_{i,q}}{\partial x^c_{i,r}}\ x^c_{i,r}\right]_{discr\ int\ q} - \left[\sum_{r=0}^{n_\theta} \frac{\partial E_{i,q}}{\partial x^c_{i,r}}\ x^c_{i,r}\right]_{discr\ int\ q+1}\\
&\qquad q = 1, ..., n_{discr\ int} - 1
\end{aligned} \tag{11.13}$$

The equation system 11.11 is non linear and has to be solved iteratively. However, after linearisation at the solution according to 11.13, it can be written into the following matricial form:

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{z} \\ \mathbf{u} \\ \mathbf{x}^c \\ \mathbf{z}^c \\ \mathbf{u}^c \\ \mathbf{\Lambda}_A \\ \mathbf{\Lambda}_{A^c} \\ \mathbf{\Lambda}_B \\ \mathbf{\Lambda}_C \\ \mathbf{\Lambda}_{C^c} \\ \mathbf{\Lambda}_D \\ \mathbf{\Lambda}_E \end{pmatrix} = \mathbf{M}^{-1} \begin{pmatrix} \mathbf{P_x}\ \mathbf{x}_m \\ \mathbf{P_z}\ \mathbf{z}_m \\ \mathbf{P_y}\ \mathbf{u}_m \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-F} \\ \textbf{-F}^c \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-G} \\ \mathbf{0} \end{pmatrix} \tag{11.14}$$

where $\mathbf{M}$ is the Jacobian matrix of the equation system 11.11. It is called the *sensitivity matrix*:

$$\mathbf{M} = \left(\begin{array}{c|c} \boldsymbol{\mathcal{P}} & \boldsymbol{\mathcal{E}}^T \\ \hline \boldsymbol{\mathcal{E}} & \mathbf{0} \end{array}\right) \tag{11.15}$$

$$\boldsymbol{\mathcal{P}} = \left(\begin{array}{ccc|ccc} \mathbf{P_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P_z} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P_u} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right) \tag{11.16}$$

$$\boldsymbol{\mathcal{E}} = \left(\begin{array}{ccc|ccc}
\frac{\partial \mathbf{A}}{\partial \mathbf{x}} & \frac{\partial \mathbf{A}}{\partial \mathbf{z}} & \frac{\partial \mathbf{A}}{\partial \mathbf{u}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{A}^c}{\partial \mathbf{x}^c} & \frac{\partial \mathbf{A}^c}{\partial \mathbf{z}^c} & \frac{\partial \mathbf{A}^c}{\partial \mathbf{u}^c} \\
\frac{\partial \mathbf{B}}{\partial \mathbf{x}} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{B}}{\partial \mathbf{x}^c} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{C}}{\partial \mathbf{u}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{C}^c}{\partial \mathbf{u}} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{C}^c}{\partial \mathbf{u}^c} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{D}}{\partial \mathbf{x}^c} & \frac{\partial \mathbf{D}}{\partial \mathbf{z}^c} & \frac{\partial \mathbf{D}}{\partial \mathbf{u}^c} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{E}}{\partial \mathbf{x}^c} & \mathbf{0} & \mathbf{0}
\end{array}\right) \tag{11.17}$$

As for the stationary case, the sparsity of the sensitivity matrix has to be taken into account and the algorithm has to be modified as described, for example, by Chen and Stadherr (Chen and Stadherr, 1984) if the problems are very large. In that case, the measurements remain generally considered as independent from one another so that the weight matrix $\mathbf{W}$ is reduced to a diagonal matrix whose elements are the inverse of the variances of the measured variables.

If the problem contains inequality constraints, the NLP problem is solved directly using a sequential quadratic programming algorithm (Kyriakopoulou, 1997).

Once the problem has been solved, a sensitivity analysis can be carried out using the linearized equation system 11.14. This equation system shows that the reconciled values of variables $\mathbf{x}$, $\mathbf{z}$, $\mathbf{u}$, $\mathbf{x}^c$, $\mathbf{z}^c$, $\mathbf{u}^c$, $\mathbf{\Lambda}_A$, $\mathbf{\Lambda}_{A^c}$, $\mathbf{\Lambda}_B$, $\mathbf{\Lambda}_C$, $\mathbf{\Lambda}_{C^c}$, $\mathbf{\Lambda}_D$ and $\mathbf{\Lambda}_E$ are linear combinations of the measurements. So, the sensitivity matrix $\mathbf{M}$ allows to evaluate how the reconciled values of the model variables depends on the measurements and their standard deviations. The variables $\mathbf{x}$, $\mathbf{z}$ and $\mathbf{u}$ are thus estimated this way:

$$
\begin{aligned}
x_{i,j} &= \sum_{s=1}^{N} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}},s}
\begin{pmatrix}
\mathbf{P}_x\ \mathbf{x} \\
\mathbf{P}_z\ \mathbf{z} \\
\mathbf{P}_u\ \mathbf{u} \\
\mathbf{0} \\
\mathbf{0} \\
\mathbf{0} \\
\textbf{-F} \\
\textbf{-F}^c \\
\mathbf{0} \\
\mathbf{0} \\
\mathbf{0} \\
\textbf{-G} \\
\mathbf{0}
\end{pmatrix}_s \\
&= \sum_{s=1}^{n_x} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}},(s-1)n^\star_{t_{mes}}+r+1}\ P_{x_{s,s,r}}\ x_{s,r} \\
&\quad + \sum_{s=1}^{n_z} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}},n_x n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{z_{s,s,r}}\ z_{s,r} \\
&\quad + \sum_{s=1}^{n_u} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}},(n_x+n_z)n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{u_{s,s,r}}\ u_{s,r} \\
&\quad - \sum_{s=1}^{n_A} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}},(n_x+n_z+n_u)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n^\star_{t_{mes}}+r+1}}\ F_{s,r} \\
&\quad - \sum_{s=1}^{n_{A^c}} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}},(n_x+n_z+n_u+n_A)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n_{t_{mes}{}^\star}+r+1}}\ F^c_{s,r} \\
&\quad - \sum_{s=1}^{n_x} \sum_{r=1}^{n^\star_\theta\ n_{discr\ \ int}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}},(n_x+n_z+n_u+n_A+n_{A^c})n^\star_{t_{mes}}+n_B \\ +n_C+n_{C^c}+(n_{x^c}+n_{z^c}+n_{u^c}+s-1)\left(n^\star_\theta\ n_{discr\ \ int}\right)+r}}\ G_{s,r}
\end{aligned}
\tag{11.18}
$$

$$\begin{aligned}
z_{i,j} &= \sum_{s=1}^{N} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},s} \begin{pmatrix} \mathbf{P}_x\ \mathbf{x} \\ \mathbf{P}_z\ \mathbf{z} \\ \mathbf{P}_u\ \mathbf{u} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-F} \\ \textbf{-F}^c \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-G} \\ \mathbf{0} \end{pmatrix}_s \\
&= \sum_{s=1}^{n_x} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},(s-1)n^\star_{t_{mes}}+r+1}\ P_{x_{s,s,r}}\ x_{s,r} \\
&\quad + \sum_{s=1}^{n_z} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},n_x n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{z_{s,s,r}}\ z_{s,r} \\
&\quad + \sum_{s=1}^{n_u} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},(n_x+n_z)n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{u_{s,s,r}}\ u_{s,r} \\
&\quad - \sum_{s=1}^{n_A} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},(n_x+n_z+n_u)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n^\star_{t_{mes}}+r+1}}\ F_{s,r} \\
&\quad - \sum_{s=1}^{n_{A^c}} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},(n_x+n_z+n_u+n_A)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n^\star_{t_{mes}}+r+1}}\ F^c_{s,r} \\
&\quad - \sum_{s=1}^{n_x} \sum_{r=1}^{n^\star_\theta\ n_{discr\ \ int}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+n_x n^\star_{t_{mes}},(n_x+n_z+n_u+n_A+n_{A^c})n^\star_{t_{mes}} \\ +n_B+n_C+n_{C^c}+(n_{x^c}+n_{z^c}+n_{u^c}+s-1)\left(n^\star_\theta\ n_{discr\ \ int}\right)+r}}\ G_{s,r}
\end{aligned} \tag{11.19}$$

$$
\begin{aligned}
u_{i,j} &= \sum_{s=1}^{N} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},s}
\begin{pmatrix}
\mathbf{P}_x\ \mathbf{x} \\ \mathbf{P}_z\ \mathbf{z} \\ \mathbf{P}_u\ \mathbf{u} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-F} \\ \textbf{-F}^c \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \textbf{-G} \\ \mathbf{0}
\end{pmatrix}_s \\
&= \sum_{s=1}^{n_x} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},(s-1)n^\star_{t_{mes}}+r+1}\ P_{x_{s,s,r}}\ x_{s,r} \\
&\quad + \sum_{s=1}^{n_z} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},n_x n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{z_{s,s,r}}\ z_{s,r} \\
&\quad + \sum_{s=1}^{n_u} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},(n_x+n_z)n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{u_{s,s,r}}\ u_{s,r} \\
&\quad - \sum_{s=1}^{n_A} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},(n_x+n_z+n_u)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n^\star_{t_{mes}}+r+1}}\ F_{s,r} \\
&\quad - \sum_{s=1}^{n_{A^c}} \sum_{r=0}^{n_{t_{mes}}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},(n_x+n_z+n_u+n_A)n^\star_{t_{mes}} \\ +(n_{x^c}+n_{z^c}+n_{u^c})\left(n^\star_\theta\ n_{discr\ \ int}\right)+(s-1)n^\star_{t_{mes}}+r+1}}\ F^c_{s,r} \\
&\quad - \sum_{s=1}^{n_x} \sum_{r=1}^{n^\star_\theta\ n_{discr\ \ int}} (M)^{-1}_{\substack{j+(i-1)n^\star_{t_{mes}}+(n_x+n_z)n^\star_{t_{mes}},(n_x+n_z+n_u+n_A+n_{A^c})n^\star_{t_{mes}} \\ +n_B+n_C+n_{C^c}+(n_{x^c}+n_{z^c}+n_{u^c}+s-1)\left(n^\star_\theta\ n_{discr\ \ int}\right)+r}}\ G_{s,r}
\end{aligned}
\tag{11.20}
$$

where

- N is the size of the sensitivity matrix $\mathbf{M}$
- $n_{t_{mes}}$ is the number of measurement times on the moving horizon different from the initial time of the horizon;
- $n^\star_{t_{mes}}$ is the number of measurement times on the moving horizon including the initial time of the horizon;

- n_θ is the degree of Lagrange polynomials;
- $n_\theta^\star$ is the number of collocation nodes on each discretization interval;
- $n_{discr\ int}$ is the number of discretization intervals on the moving horizon;
- $n_{interp\ int}$ is the number of interpolation intervals on the moving horizon;
- $n_x = n_{x^c}$ is the number of differential state variables;
- $n_z = n_{z^c}$ is the number of algebraic variables;
- $n_u = n_{u^c}$ is the number of input variables;
- n_A is the number of link equations at measurement times;
- n_{A^c} is the number of link equations at collocation nodes;
- n_B is the number of **B** constraints: $n_B = n_x\ n_\theta\ n_{discr\ int}$;
- n_C is the number of **C** constraints: $n_C = n_u\left(n_{t_{mes}} - n_{interp\ int} - 1\right)$;
- n_{C^c} is the number of $\mathbf{C}^c$ constraints: $n_{C^c} = n_u\ n_\theta\ n_{discr\ int}$;
- n_D is the number of **D** constraints: $n_D = n_x\ n_\theta\ n_{discr\ int}$;
- n_E is the number of **E** constraints: $n_E = n_x\left(n_{discr\ int} - 1\right)$.

Knowing that the variance of a linear combinaison LC of several variables x_i is given by:

$$LC = \sum_{i=1}^{m} a_i\ x_i \tag{11.21}$$

$$var\left(LC\right) = \sum_{i=1}^{m} {a_i}^2\ var\left(x_i\right) \tag{11.22}$$

The variances of the variables of the model are estimated this way:

$$\begin{aligned} var\left(x_{i,j}\right) \;=\; & \sum_{s=1}^{n_x}\sum_{r=0}^{n_{t_{mes}}} \left[(M)^{-1}_{j+(i-1)n^\star_{t_{mes}},(s-1)n^\star_{t_{mes}}+r+1}\ P_{x_{s,s,r}}\right]^2\ var\left(x^m_{s,r}\right) \\ & +\sum_{s=1}^{n_z}\sum_{r=0}^{n_{t_{mes}}} \left[(M)^{-1}_{j+(i-1)n^\star_{t_{mes}},n_x n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{z_{s,s,r}}\right]^2\ var\left(z^m_{s,r}\right) \\ & +\sum_{s=1}^{n_u}\sum_{r=0}^{n_{t_{mes}}} \left[(M)^{-1}_{j+(i-1)n^\star_{t_{mes}},(n_x+n_z)n^\star_{t_{mes}}+(s-1)n^\star_{t_{mes}}+r+1}\ P_{u_{s,s,r}}\right]^2\ var\left(u^m_{s,r}\right) \end{aligned} \tag{11.23}$$

$$\begin{aligned}
var\left(z_{i,j}\right) = & \sum_{s=1}^{n_x}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{j+(i-1)n^\star_{tmes}+n_x n^\star_{tmes},(s-1)n^\star_{tmes}+r+1}\ P_{x_{s,s,r}}\right]^2\ var\left(x^m_{s,r}\right) \\
& +\sum_{s=1}^{n_{\mathbf{z}}}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{\substack{j+(i-1)n^\star_{tmes}+n_x n^\star_{tmes},n_x * n_{tmes} \\ +(s-1)n^\star_{tmes}+r+1}}\ P_{z_{s,s,r}}\right]^2\ var\left(z^m_{s,r}\right) \\
& +\sum_{s=1}^{n_u}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{\substack{j+(i-1)n^\star_{tmes}+n_x n^\star_{tmes},(n_x+n_z)n^\star_{tmes} \\ +(s-1)n^\star_{tmes}+r+1}}\ P_{u_{s,s,r}}\right]^2\ var\left(u^m_{s,r}\right)
\end{aligned} \tag{11.24}$$

$$\begin{aligned}
var\left(u_{i,j}\right) = & \sum_{s=1}^{n_x}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{j+(i-1)n^\star_{tmes}+(n_x+n_z)n^\star_{tmes},(s-1)n^\star_{tmes}+r+1}\ P_{x_{s,s,r}}\right]^2\ var\left(x^m_{s,r}\right) \\
& +\sum_{s=1}^{n_z}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{\substack{j+(i-1)n^\star_{tmes}+(n_x+n_z)n^\star_{tmes},n_x n^\star_{tmes} \\ +(s-1)n^\star_{tmes}+r+1}}\ P_{z_{s,s,r}}\right]^2\ var\left(z^m_{s,r}\right) \\
& +\sum_{s=1}^{n_u}\sum_{r=0}^{n_{tmes}} \left[(M)^{-1}_{\substack{j+(i-1)n^\star_{tmes}+(n_x+n_z)n^\star_{tmes},(n_x+n_z)n^\star_{tmes} \\ +(s-1)n^\star_{tmes}+r+1}}\ P_{u_{s,s,r}}\right]^2\ var\left(u^m_{s,r}\right)
\end{aligned} \tag{11.25}$$

Thus the calculation of sensitivity matrix $\mathbf{M}$ allows to assess the variance of reconciled variables, knowing the variances of all measured variables.

Chapter 12

Cases study

In this chapter, three examples from the ones that have been studied will be treated:

- one tank;
- a stirred tank reactor with heat exchange;
- a network of five tanks.

For each of them the results of the reconciliation and the estimation of a posteriori variances will be discussed.

12.1 One tank

This first example is very simple, but it allows to illustrate all concepts developed in the previous chapters.
It consists of a dynamic balance for a tank, whose feed flowrate F_0 is the independent variable. The outlet flowrate F_1 is proportional to the level of the tank. The section of the tank is considered constant. The tank is represented on figure 12.1.

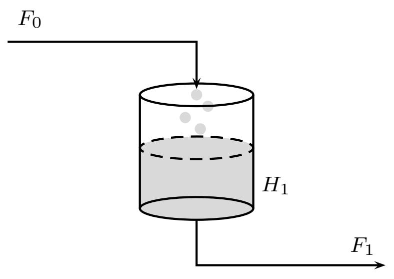


Figure 12.1: Flowsheet of the tank

The model is defined by two equations: one differential equation and one link equation:

$$\frac{dh_1}{dt} = \frac{F_0}{A_1} - \frac{F_1}{A_1} \tag{12.1}$$

$$F_1 = \beta_1 \; H_1 \tag{12.2}$$

$$\Updownarrow$$

$$\frac{d\mathbf{x}_1}{dt} = \frac{\mathbf{u}_1}{A_1} - \frac{\mathbf{z}_1}{A_1} \tag{12.3}$$

$$\mathbf{z}_1 = \beta_1 \; \mathbf{x}_1 \tag{12.4}$$

Three variables are related by the model equations:

- one input variable: the feed flowrate F_0;
- one differential state variable: the height in the tank H_1:
- one algebraic variable: the outlet flowrate F_1.

Two parameters are specified as follow:

Table 12.1: One tank: model parameters

Parameters	Values	Units
A_1	10	dm^2
β_1	1.5	-

A set of pseudo measurements has been obtained by solving the model and adding random noise to the variables. In this simulation the differential equations are integrated by means of the fourth order Runge-Kutta method with a step of 1 s. The process has been simulated during 481 seconds and the simulated measurements are given every second. The input variable remains constant during some time intervals but changes abruptly every 40 seconds as indicated in table 12.2

Table 12.2: One tank: feed flowrate evolution

Times	F_0	Units
0 s	50	dm^3/s
41 s	100	dm^3/s
81 s	60	dm^3/s
121 s	30	dm^3/s
161 s	40	dm^3/s
201 s	50	dm^3/s

continued on next page

Times	F_0	Units
240 s	80	dm^3/s
281 s	120	dm^3/s
321 s	70	dm^3/s
361 s	40	dm^3/s
401 s	80	dm^3/s
441 s	120	dm^3/s
481 s	90	dm^3/s

A Gaussian noise with a standard deviation of 1 % of the variable value is added to the simulation results of the flowrates before carrying out the data reconciliation. Errors on height measurements are Gaussian with zero mean and a standard deviation of 2 cm. The initial conditions of the reconciliation problem are listed in the next table 12.3:

Table 12.3: One tank: initial values

Variables	Initial values	Units
H_1	60.0	dm
F_{in}	60.0	dm^3/s
F_{out}	50.0	dm^3/s

The parameters of the reconciliation window have been chosen as indicated in table 12.4:

Table 12.4: One tank: window parameters

Parameters	Values
h_1	1
h_2	4
h_3	4
h_4	49
h_5	2
Order of Lagrange polynomials	2

12.1.1 Results for the steps

In the first three figures (figures 12.2, 12.3 and 12.4), the profiles of the three variables of the tank are represented for the whole duration of the simulation. All charts represented in this chapter represent the reconciled values that will leave the window for the next reconciliation. One can see that all variables follow the simulated true value. Nevertheless, the input variables follow more the noise than the other ones. More over, in the case of input variable, the reconciled values are not able to represent closely the steps. That is due to the fact that one tries to represent a discontinuous function by means of second order polynomials.

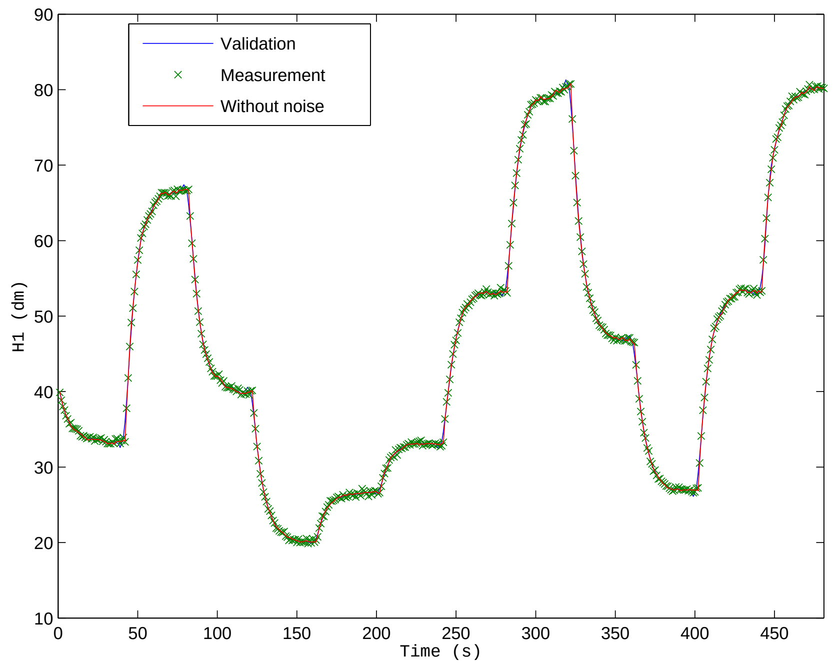


Figure 12.2: One tank: steps: height: profile

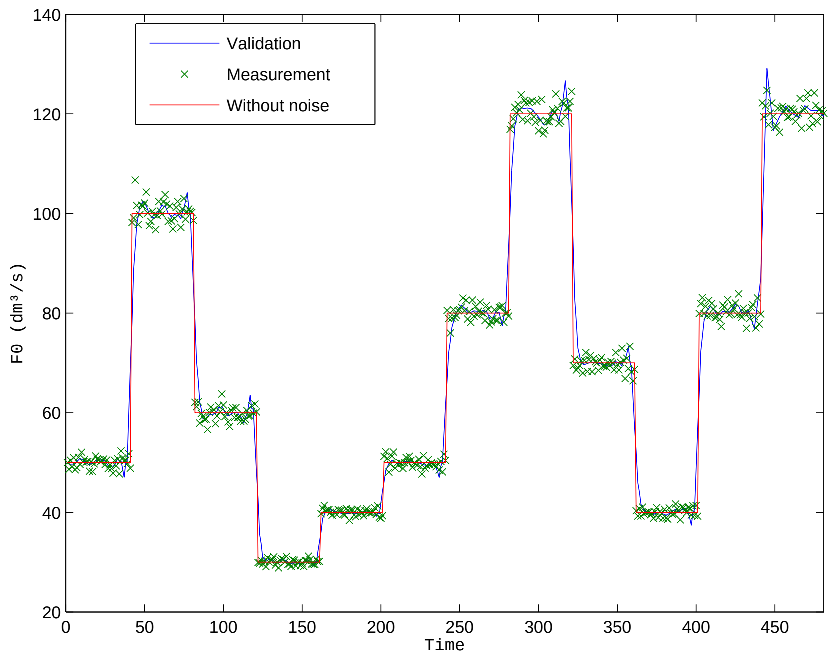


Figure 12.3: One tank: steps: feed flowrate: profile

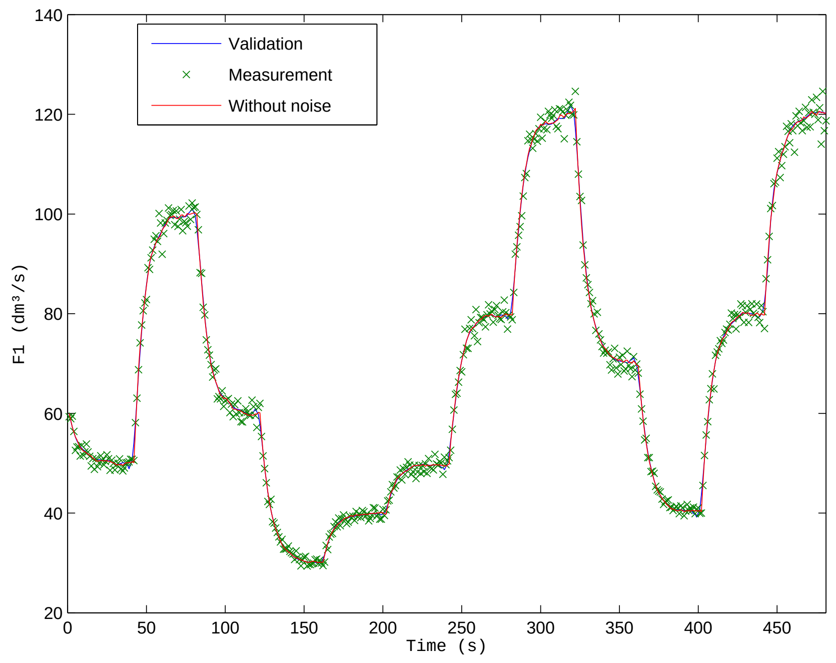


Figure 12.4: One tank: steps: outlet flowrate: profile

reconciliation and measurement errors of the three variables are represented on figures 12.5, 12.6 and 12.7 for a reconciliation window. Reconciliation errors are generally less important than simulation noise. Nevertheless, reconciliation errors are much larger at the discontinuities. Moreover, reconciliation errors are more important and measurement noise is less corrected for the feed flowrate.

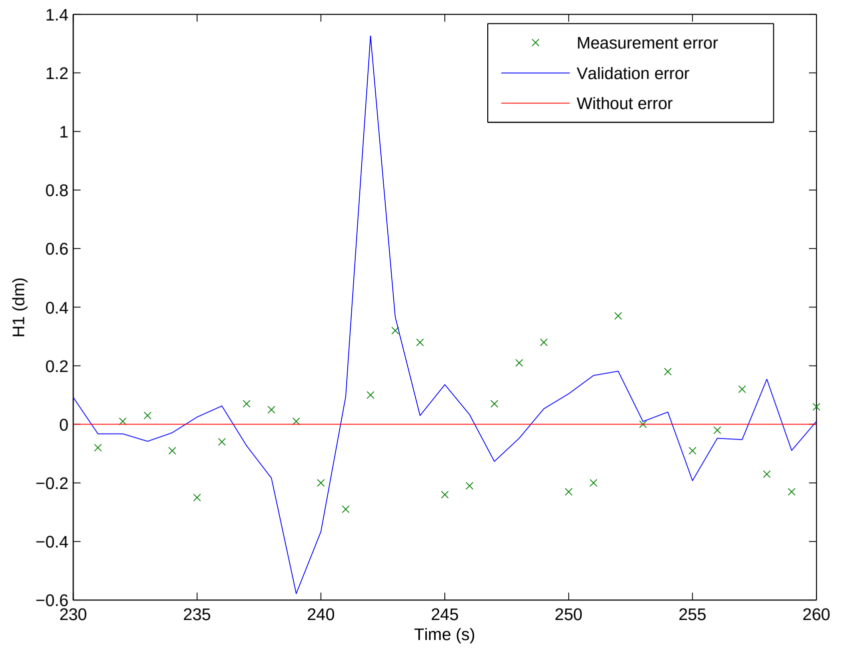


Figure 12.5: One tank: steps: height: errors

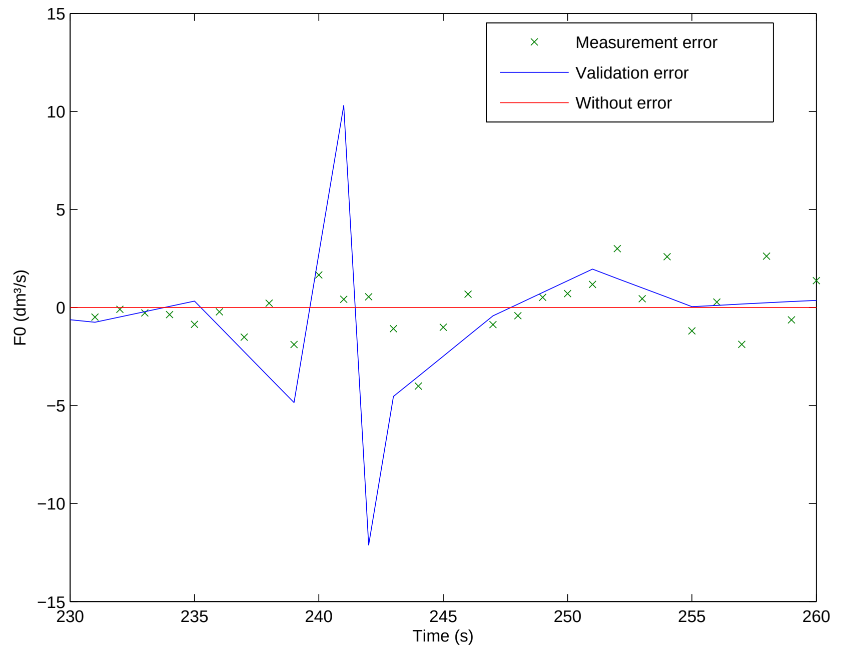


Figure 12.6: One tank: steps: feed flowrate: errors

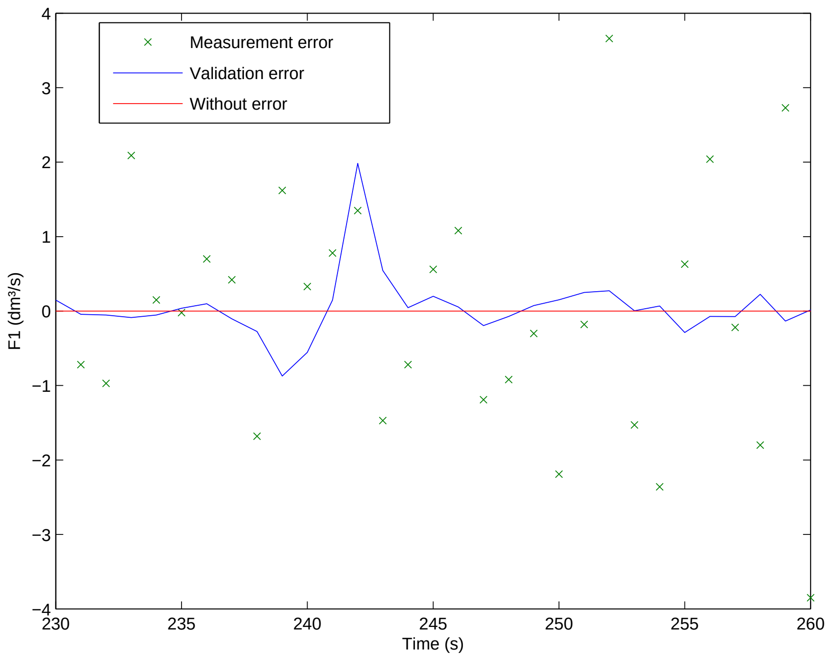


Figure 12.7: One tank: steps: outlet flowrate: errors

Next we analyse results for a single optimization, namely the reconciliation window extending from $t = 230\ s$ to $t = 260\ s$. In the next three figures (figures 12.8, 12.9 and 12.10) error bars have been added to the profiles ($error\ bars = value \pm once\ the\ standard\ deviation$). This allows to compare a priori (for measurement) and a posteriori (for reconciled variable) standard deviations for the same reconciliation window. For all variables a posteriori standard deviations are distinctly reduced compared to a priori ones. Nevertheless, one can note that true values are not always included in a posteriori standard deviations of reconciled values.

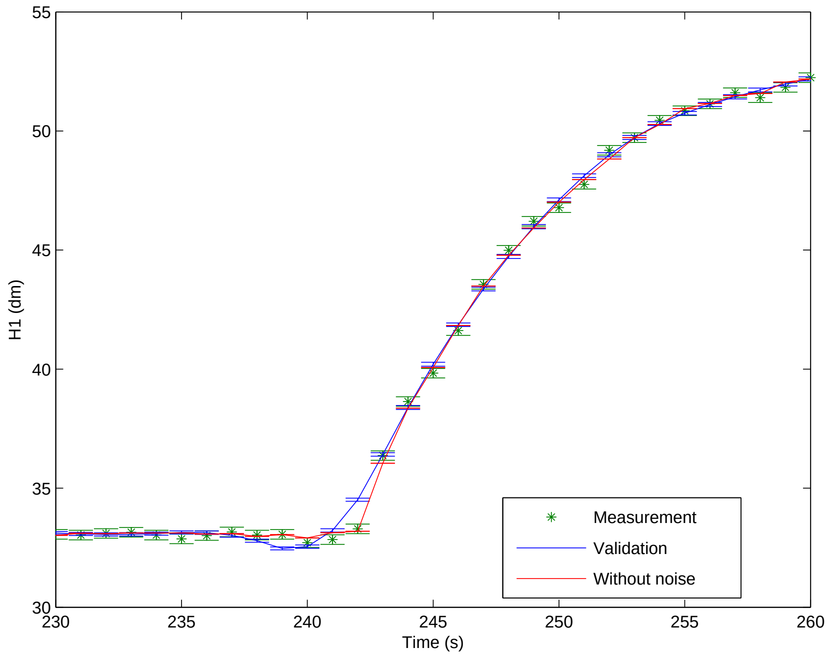


Figure 12.8: One tank: steps: height: standard deviation comparison

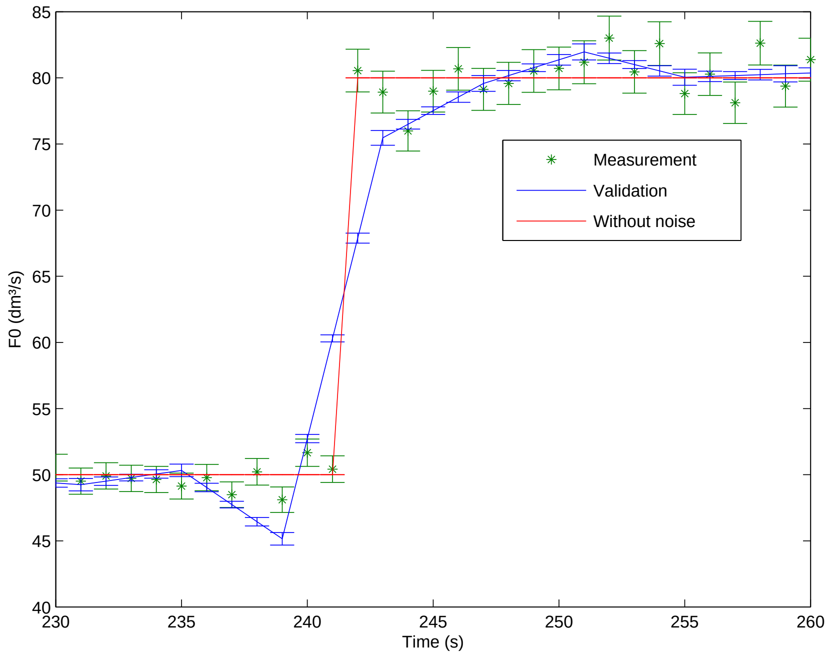


Figure 12.9: One tank: steps: feed flowrate: standard deviation comparison

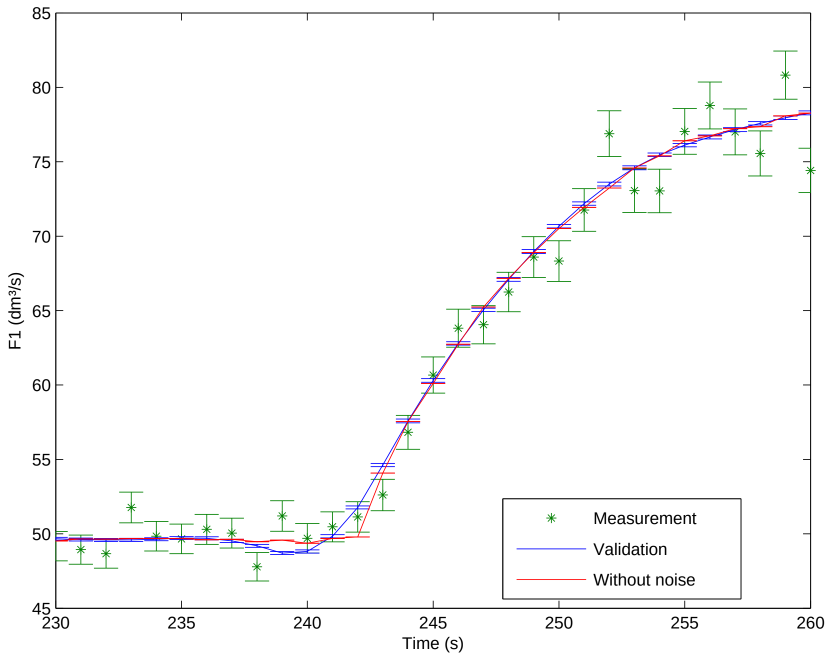


Figure 12.10: One tank: steps: outlet flowrate: standard deviation comparison

Because of the discontinuity that appears at the steps, we have also simulated a case where input variations are smoother: linear variations of feed and smooth variations according to a first order response $\frac{dx}{dt} = 0.1\,(x - x_{target})$ where x_{target} is the target value of the variable x ate the end of the perturbation..

12.1.2 Results for the linear variations of feed

The linear variations have been defined with a constant slope (± 2 unit flow/unit time) until the next target value is reached.

If one compares the profiles of the three variables of the complete simulation time (figures 12.11, 12.12 and 12.13), one can see that the reconciled values follow the true values for all variables. The discontinuities at the time corresponding to the input variable variations are much smaller than for the step transitions.

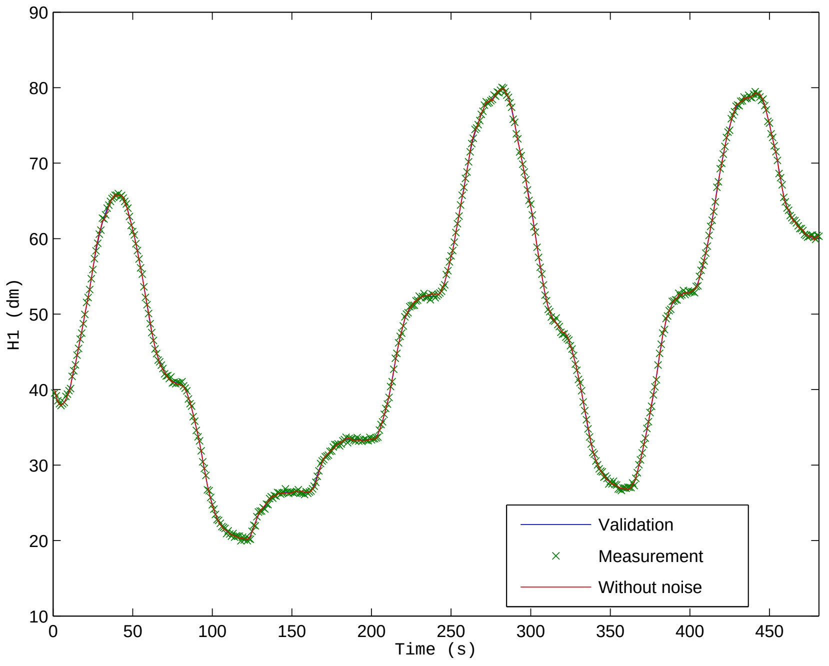


Figure 12.11: One tank: linear variations of feed: height: profile

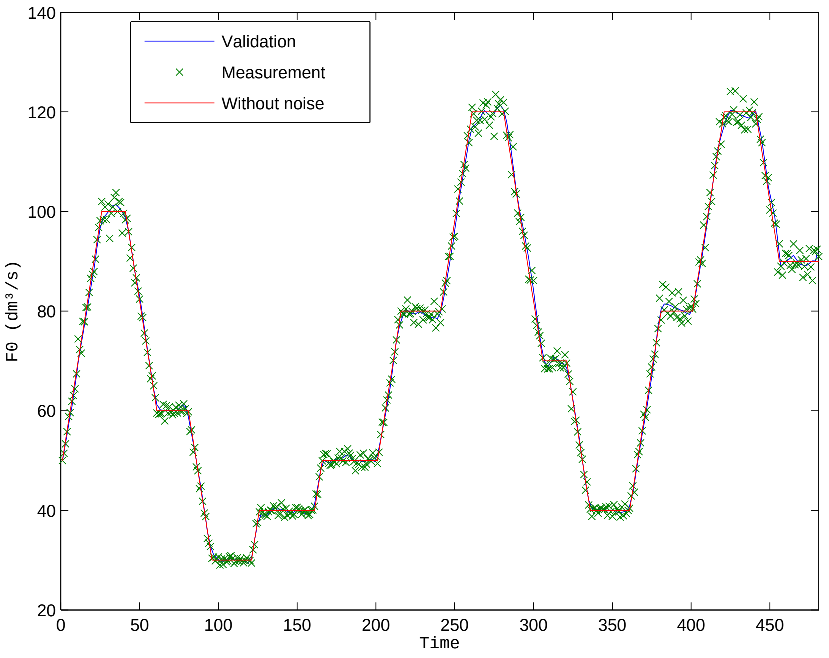


Figure 12.12: One tank: linear variations of feed: feed flowrate: profile

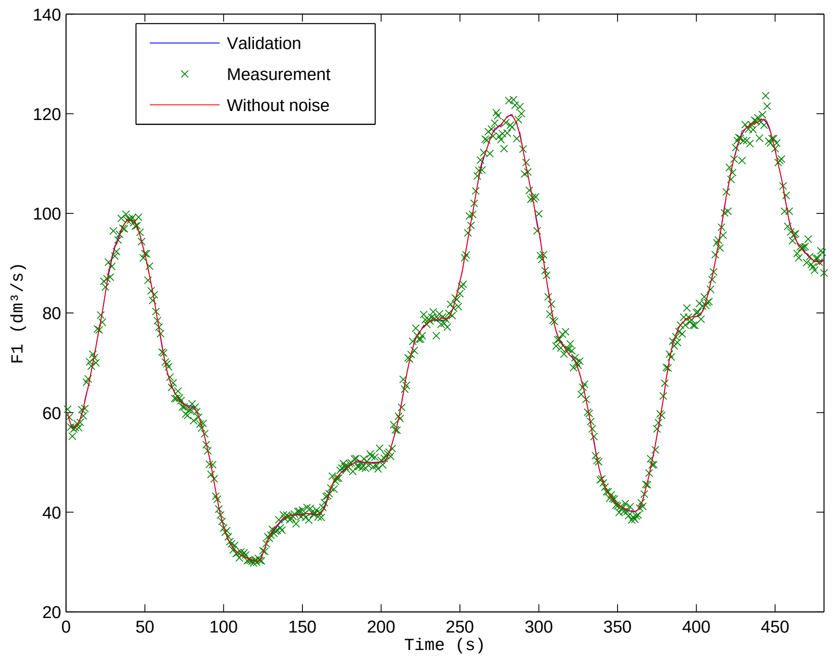


Figure 12.13: One tank: linear variations of feed: outlet flowrate: profile

In figures 12.14, 12.15 and 12.16, reconciliation and measurement errors are represented for a single reconciliation window. The important mismatch at the time corresponding to input variable variations is no more noticeable.

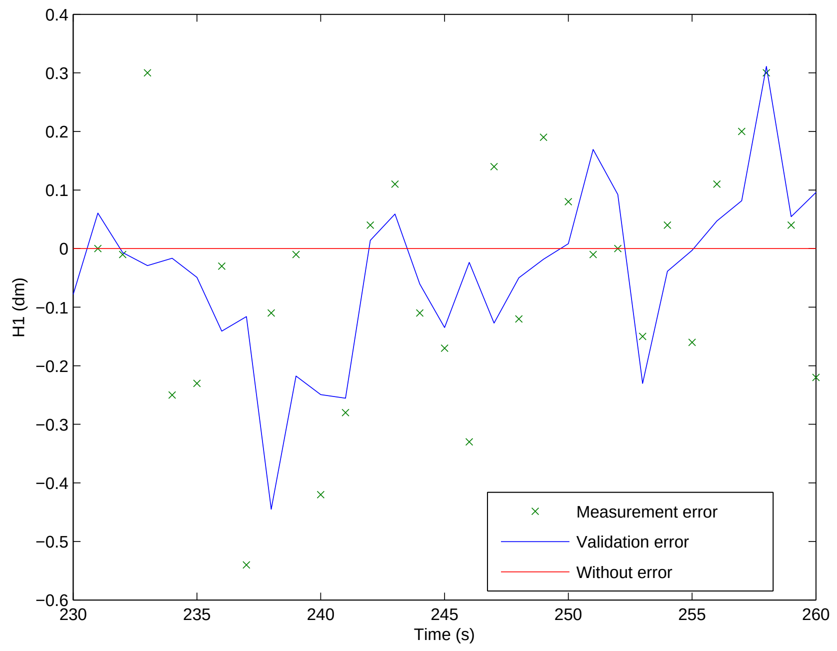


Figure 12.14: One tank: linear variations of feed: height: errors

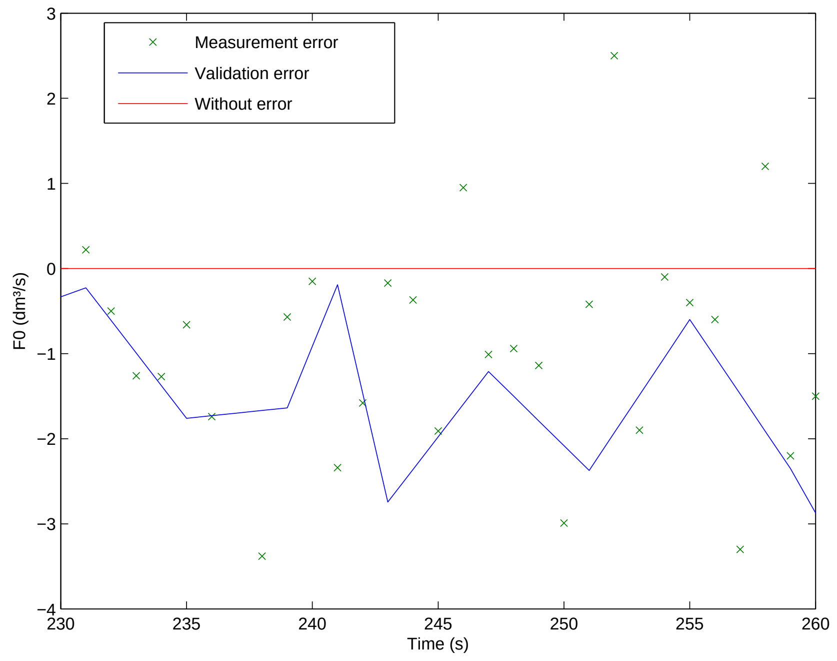


Figure 12.15: One tank: linear variations of feed: feed flowrate: errors

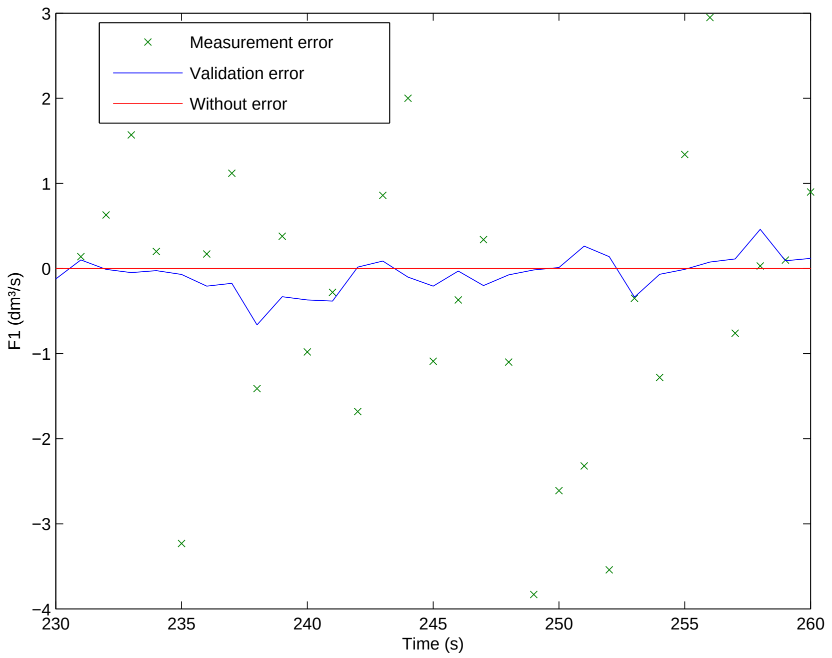


Figure 12.16: One tank: linear variations of feed: outgoing flowrate: errors

In the next three figures (figures 12.17, 12.18 and 12.19), the a priori and a posteriori standard deviations are compared for the same reconciliation window. For all variables a posteriori standard deviations are distinctly reduced compared to a priori ones. Nevertheless, because of the Gaussian distribution of noise, as for the step perturbations, one can note that the true values are not always included within the confidence limits the reconciled values (errors bars represent the reconciled value *pm* once the a posteriori standard deviation).

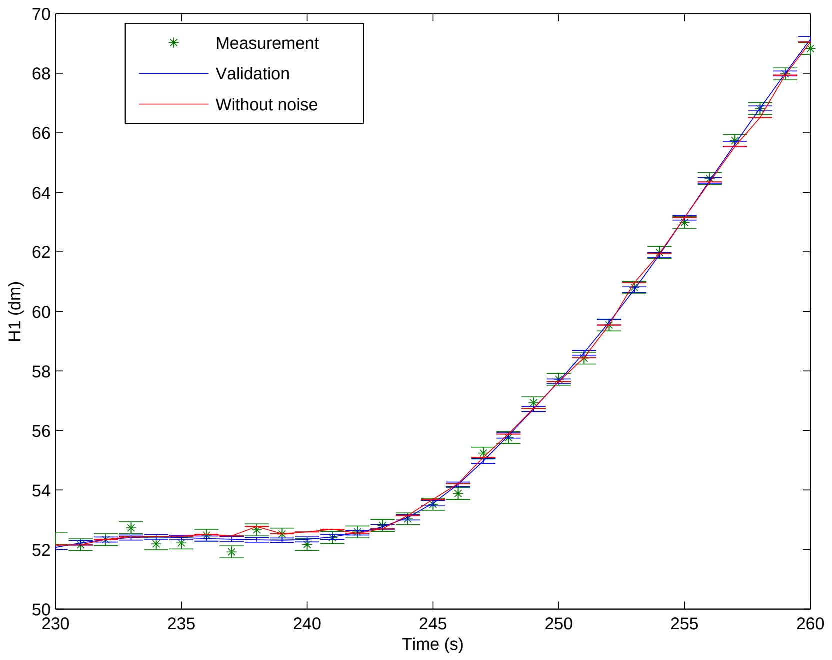


Figure 12.17: One tank: linear variations of feed: height: standard deviation comparison

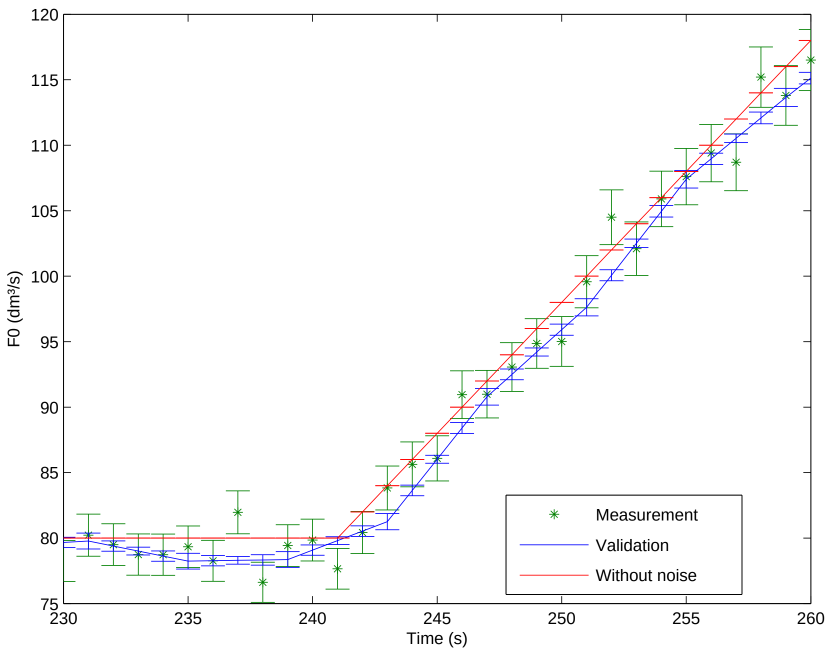


Figure 12.18: One tank: linear variations of feed: feed flowrate: standard deviation comparison

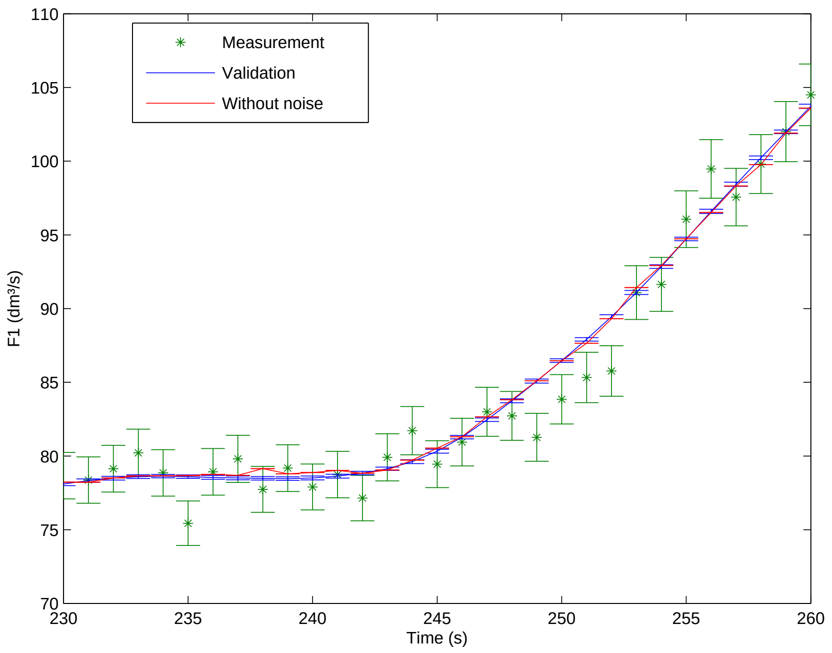


Figure 12.19: One tank: linear variations of feed: outgoing flowrate: standard deviation comparison

12.1.3 Results for the smooth perturbations $\frac{dx}{dt} = 0.1\,(x - x_{target})$

The profiles of the three variables of the whole time horizon are represented on figures 12.20, 12.21 and 12.22. As for the linear perturbations, one can see that the reconciled values follow the true values for all variables. The mismatch at the time corresponding to the input variable variations is also much smaller than for the step perturbations.

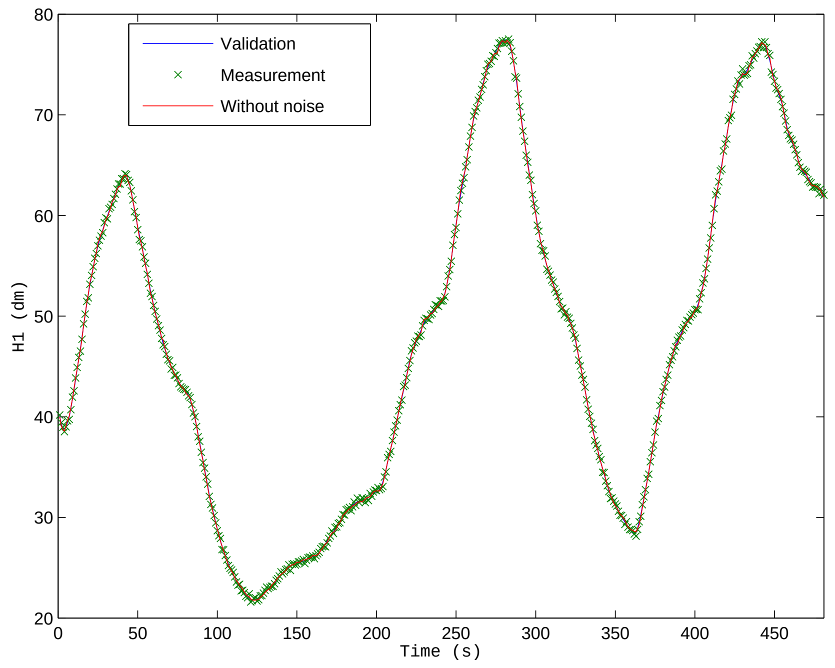


Figure 12.20: One tank: smooth perturbations: height: profile

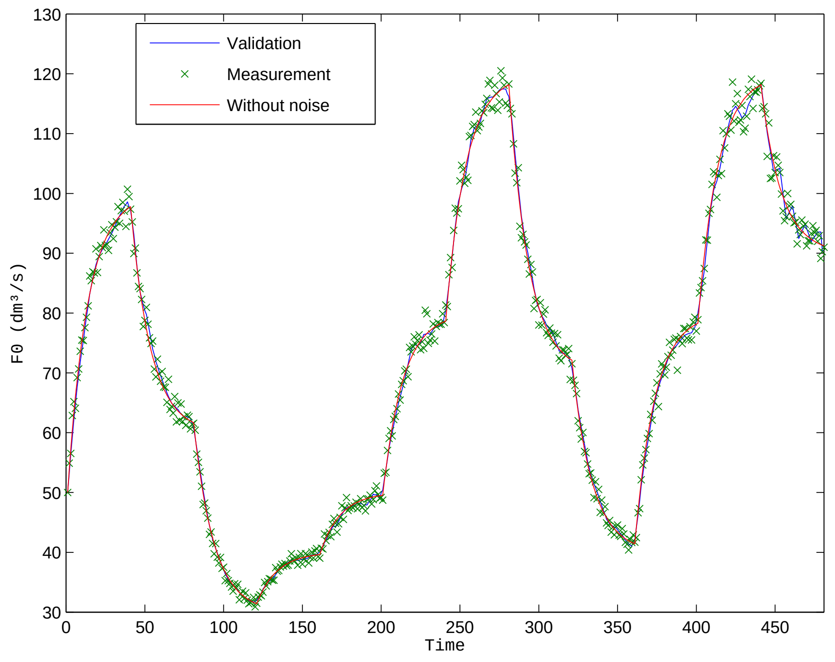


Figure 12.21: One tank: smooth perturbations: feed flowrate: profile

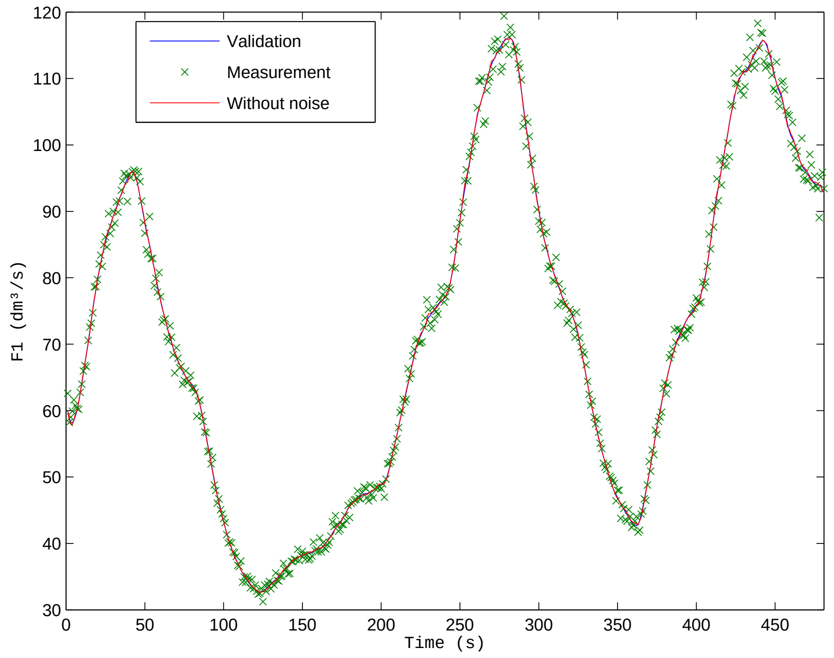


Figure 12.22: One tank: smooth perturbations: outgoing flowrate: profile

Concerning the errors (figures 12.23, 12.24 and 12.25), one can see that they are of the same order of magnitude as for the linear variations and no strong mismatch is noticed at the time corresponding to input variable transitions.

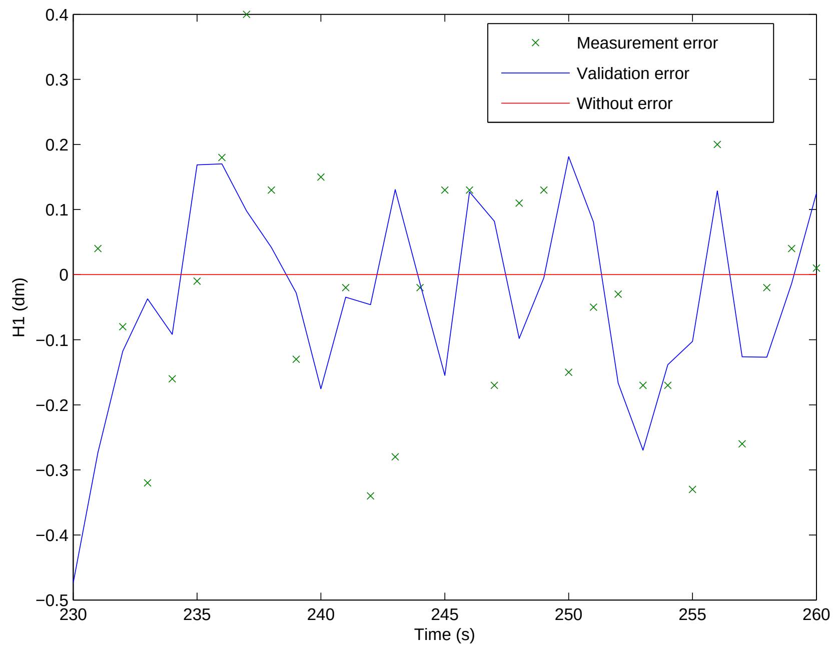


Figure 12.23: One tank: smooth perturbations: height: errors

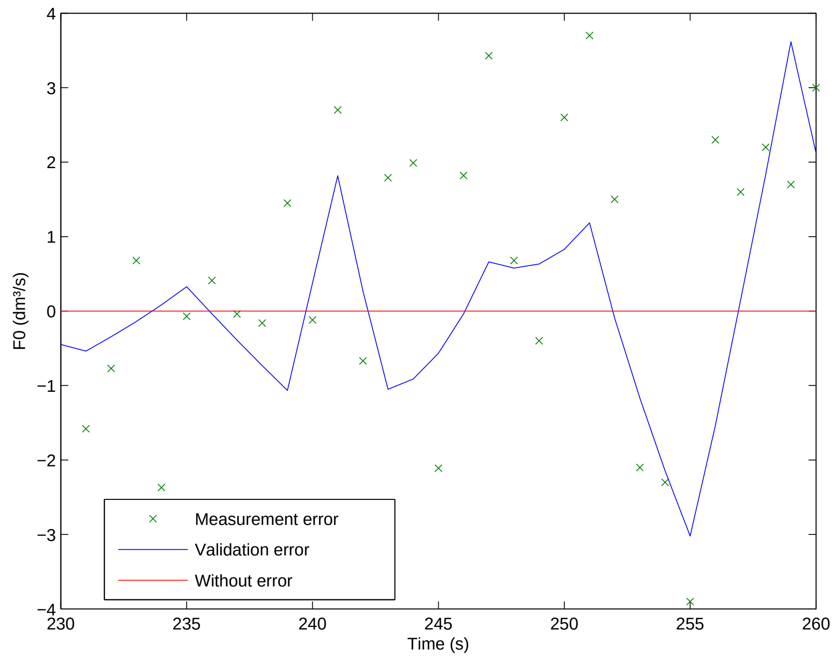


Figure 12.24: One tank: smooth perturbations: feed flowrate: errors

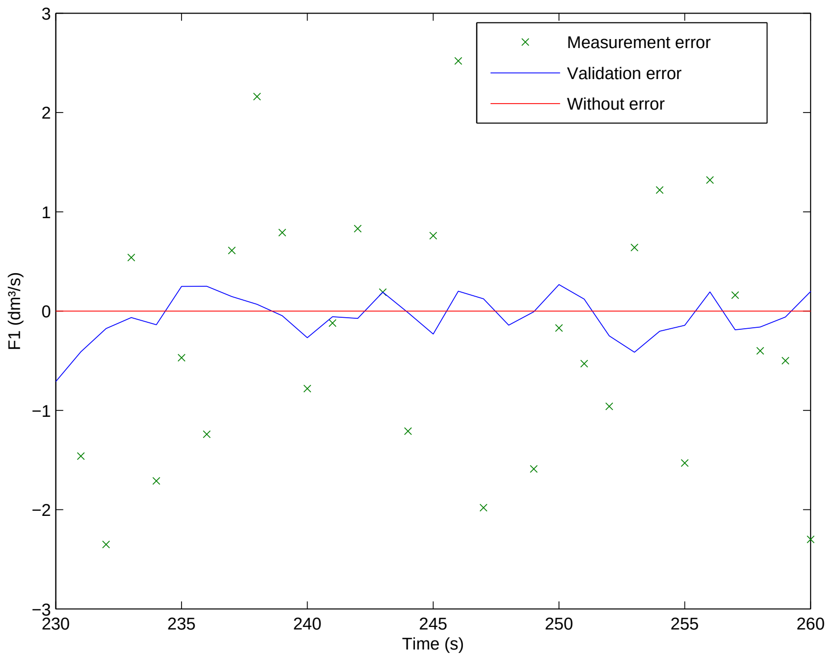


Figure 12.25: One tank: smooth perturbations: outgoing flowrate: errors

In the next three figures (figures 12.26, 12.27 and 12.28), the a priori and a posteriori standard deviations are compared for a single reconciliation window. For some measurements times of the reconciliation window, a priori and a posteriori standard deviations are listed in tables 12.6, 12.7 and 12.8.
It appears that a posteriori standard deviations are distinctly reduced compared to a priori ones for the three variables. As for the linear variations, one can note that the true values are not always included in the a posteriori standard deviations of the reconciled values, as expected for a Gaussian distribution .
Standard deviations are less reduced for the first and the last measurement times of the reconciliation window, whatever the variable.
In the next table 12.5, a posteriori standard deviation reduction mean factors are listed with their standard deviations. It appears that the mean reduction factor is larger for the algebraic state variable.

Table 12.5: One tank: smooth perturbations: a posteriori standard deviation reduction mean factors

Variables	A posteriori standard deviation reduction factors: means $= \frac{\sum \frac{A posteriori standard deviation}{A priori standard deviation}}{n}$	A posteriori standard deviation reduction factors: standard deviations $= \frac{\sum\left(\frac{A posteriori standard deviation}{A priori standard deviation} - mean\right)^2}{n-1}$
H_1	2.3	0.33
F_{in}	4.0	1.3
F_{out}	13	2.4

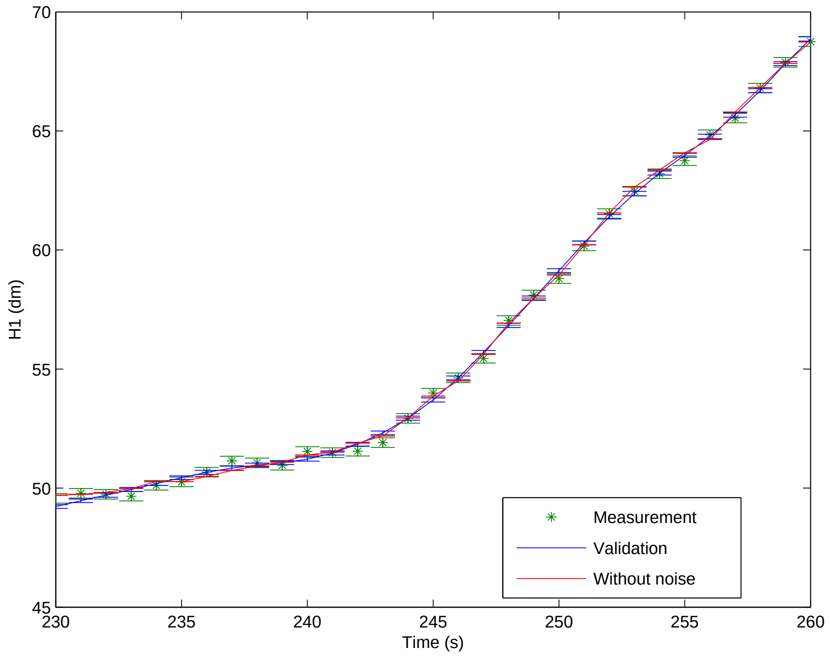


Figure 12.26: One tank: function: height: standard deviation comparison

Table 12.6: One tank: smooth perturbations: H_1: a posteriori standard deviation reduction factors in a single window

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
			$= \frac{A priori standard deviation}{A posteriori standard deviation}$
220	0.20	0.16	1.2
225	0.20	0.087	2.3
230	0.20	0.077	2.6

continued on next page

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
			$= \frac{A priori standard deviation}{A posteriori standard deviation}$
235	0.20	0.076	2.6
240	0.20	0.086	2.3
245	0.20	0.089	2.2
250	0.20	0.082	2.4
255	0.20	0.081	2.4
260	0.20	0.094	2.1
265	0.20	0.096	2.1

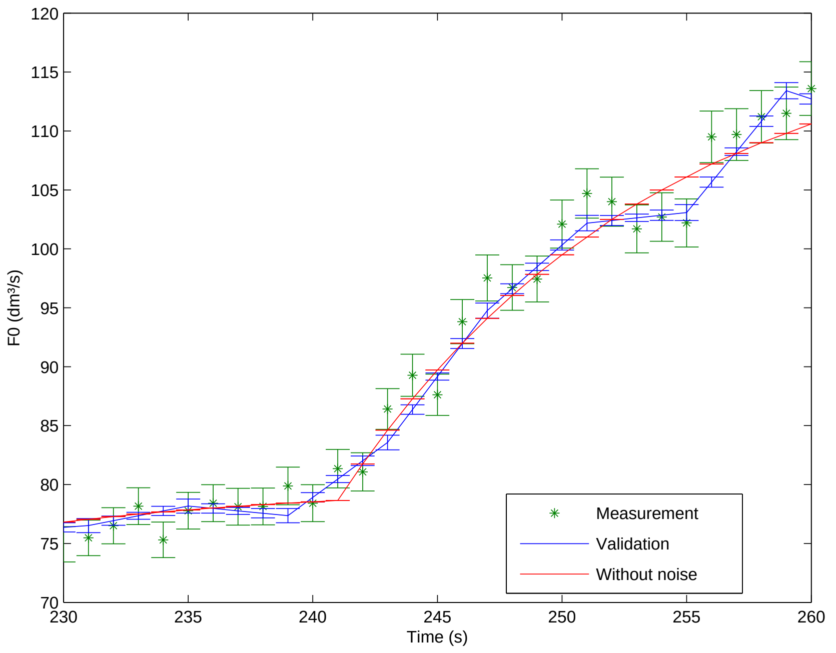


Figure 12.27: One tank: smooth perturbations: feed flowrate: standard deviation comparison

Table 12.7: One tank: smooth perturbations: F_0: a posteriori standard deviation reduction factors in a single window

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
220	1.49	0.99	1.5
225	1.48	0.31	4.8
230	1.50	0.39	3.8
235	1.56	0.60	2.6
240	1.57	0.39	4.0

continued on next page

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
245	1.75	0.31	5.6
250	2.04	0.43	4.7
255	2.04	0.67	3.0
260	2.27	0.44	5.2
265	2.27	0.49	4.6

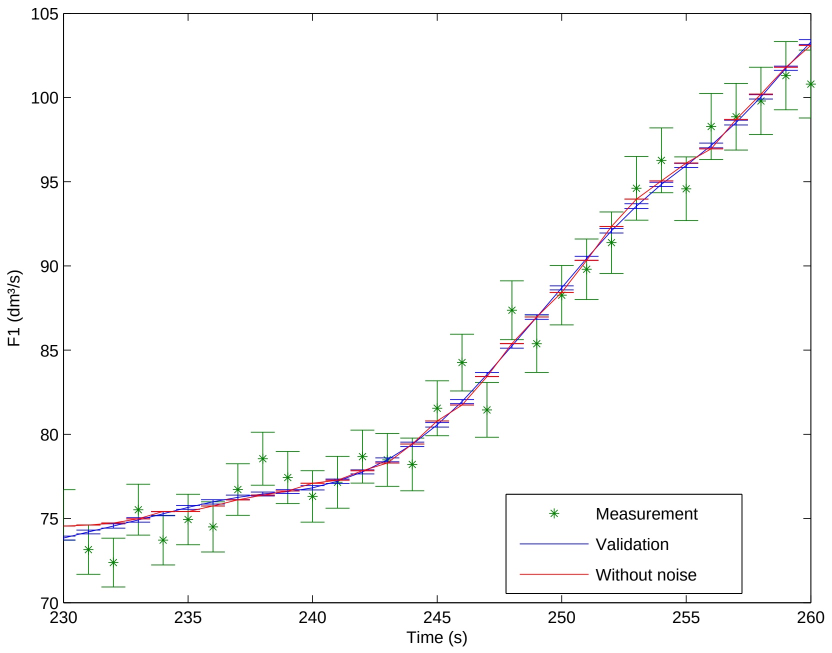


Figure 12.28: One tank: smooth perturbations: outgoing flowrate: standard deviation comparison

Table 12.8: One tank: smooth perturbations: F_1 : a posteriori standard deviation reduction factors in a single window

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
220	1.36	0.23	5.9
225	1.20	0.13	9.2
230	1.50	0.12	12.5
235	1.50	0.11	13.6
240	1.53	0.13	11.8
245	1.63	0.13	12.5
250	1.76	0.12	14.7
255	1.89	0.12	15.8

continued on next page

Times	A priori standard deviations	A posteriori standard deviations	Reduction factors
260	2.02	0.14	14.4
265	2.06	0.14	14.7

Four distributions of errors will be compared (see figure 12.29):

- the distribution of measurement errors;
- the distribution of the values that have been reconciled one time (corresponding to the h_5 more recent measurement times of the reconciliation window). This distribution curve is called *beginning*.
- the distribution of the reconciled values that leave that reconciliation window (corresponding to the h_5 oldest measurement times of the reconciliation window). This distribution curve is called *end*.
- the distribution of the reconciled values that are situated at the middle of the reconciliation window (h_5 measurement times for each reconciliation). This distribution curve is called *middle*.

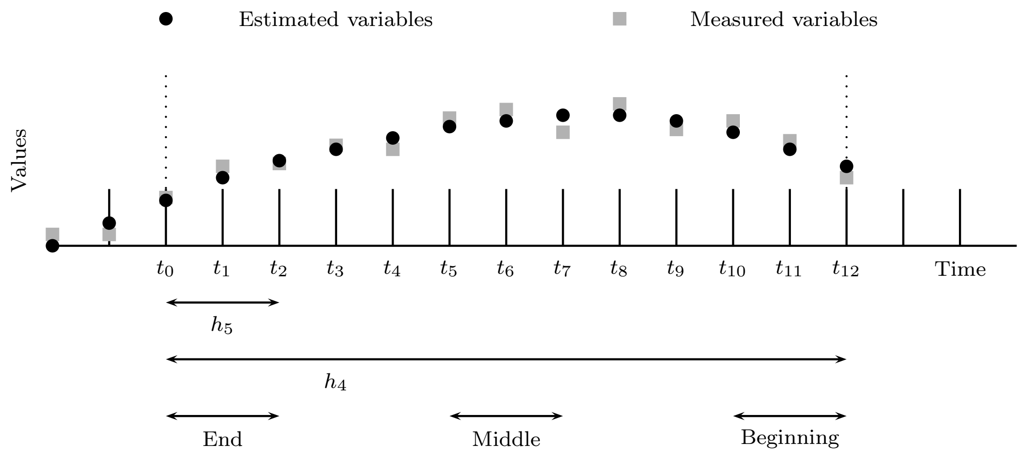


Figure 12.29: One tank: smooth perturbations: schema of the error distributions

The last three figures (figures 12.30, 12.31 and 12.32) represent the distributions of the errors for all variables. One can see the means of errors are nearly zero, so that the normal noise with zero mean that was imposed for the simulation data is well reproduced for the reconciliation distributions.
It also appears that the error distribution is reduced while reconciling the data: indeed the distributions are larger for the measurements errors than the three other ones.

Concerning the way of saving the data, the best curve is the same for the algebraic state variable and for the differential state variable since they are related by a constraint equation. For differential state variable and algebraic state variable, there is not a big difference between the three cuves while the difference is more important for the input variable.

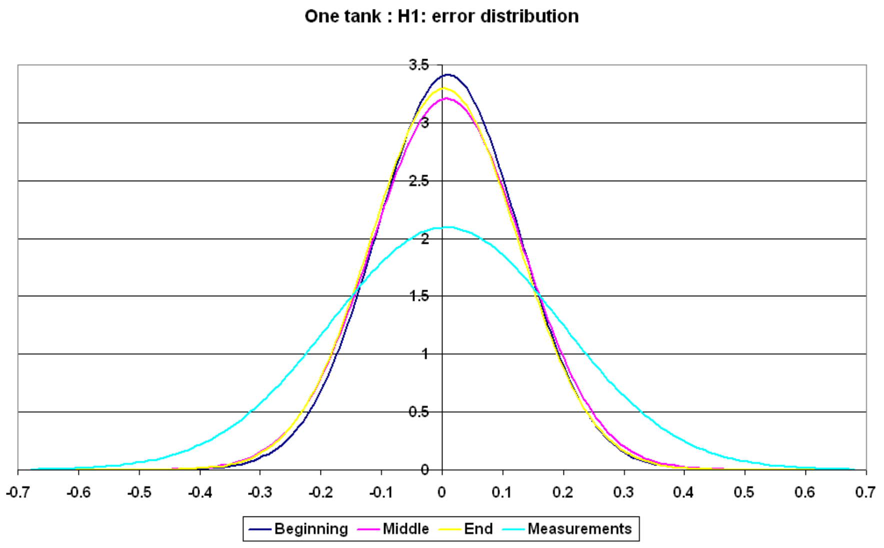


Figure 12.30: One tank: smooth perturbations: height: error distribution

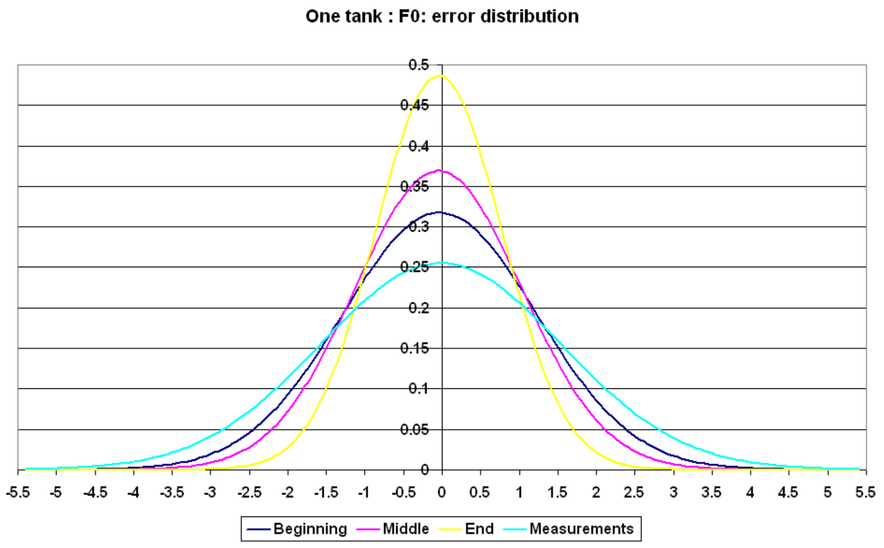


Figure 12.31: One tank: smooth perturbations: feed flowrate: error distribution

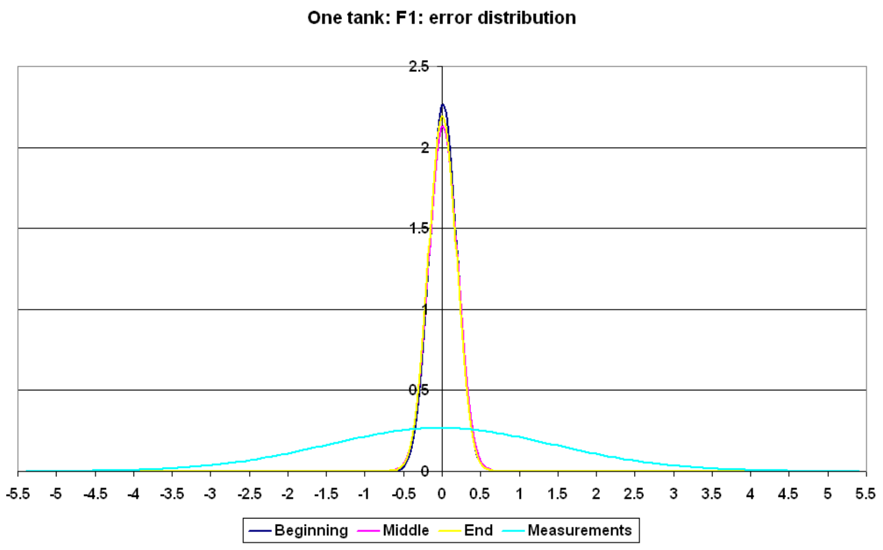


Figure 12.32: One tank: smooth perturbations: outgoing flowrate: error distribution

For the other examples, only smooth perturbations of the inputs will be carried out.

12.2 Stirred tank reactor with heat exchange

This second example comes from a publication of Liebman (Liebman et al., 1992). It consists of a perfectly mixed reactor with a heat exchange system, where a first-order exothermic reaction takes place. The system is represented on figure 12.33.

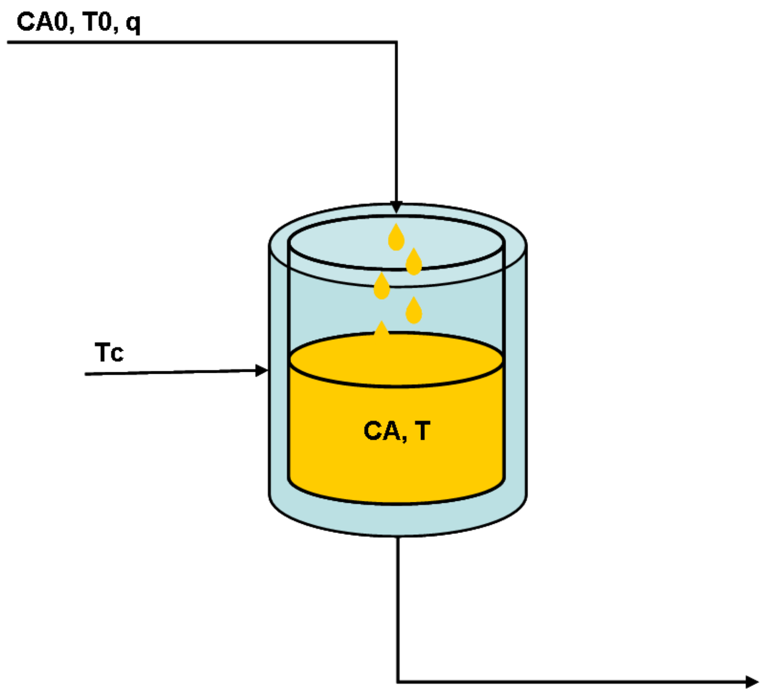


Figure 12.33: Flowsheet of the stirred tank reactor with heat exchange

This system is described by two differential equations:

$$\frac{dC_A}{dt} = \frac{q}{V}\ (C_{A0} - C_A) - \alpha_d\ k\ C_A \tag{12.5}$$

$$\frac{dT}{dt} = \frac{q}{V}\ (T_0 - T) - \alpha_d\ \frac{\Delta H_r}{\rho\ C_p}\ k\ C_A\ - \frac{U A_R}{\rho\ C_p\ V}\ (T - T_c) \tag{12.6}$$

with

$$k = k_0\ e^{\frac{-E_A}{T}} \tag{12.7}$$

This example is described by six variables:

- four input variables: the feed flowrate q, the feed concentration C_{A0},the feed temperature T_0 and the cooling temperature T_C;
- two differential state variables: the concentration C_A and the temperature T.

The model counts eleven parameters fixed as follow:

Table 12.9: Stirred tank reactor with heat exchange: model parameters

Parameters	Values	Units
V	1000	cm^3
ΔH_r	-27000	cal/mol
ρ	0.001	g/cm^3
C_p	1	$cal/mol\ K$
U	$5e^{-4}$	$cal/cm^2\ s\ K$
A_R	10	cm^2
α_d	1	-
k_0	$7.86e^{12}$	s^{-1}
E_A	$14090\ K$	
C_{Ar}	$1e^{-6}$	mol/cm^3
T_r	100	K

The model can be transformed into dimensionless equations by normalizing the concentration and the temperature by a reference concentration A_r and a reference temperature T_r so that the equations of the system are transformed into:

$$\begin{aligned}
\frac{dC'_A}{dt} &= \frac{q}{V}\ (C'_{A0} - C'_A) - \alpha_d\ k\ C'_A && (12.8)\\
\frac{dT'}{dt} &= \frac{q}{V}\ (T'_0 - T') - \alpha_d\ \frac{\Delta H_r\ C_{A_r}}{\rho\ C_p}\ k\ C'_A\ - \frac{U A_R}{\rho\ C_p\ V}\ (T' - T'_C) && (12.9)\\
k &= k_0\ e^{\frac{-E_A}{T'\ T_r}} && (12.10)\\
&\Updownarrow \\
\frac{d\mathbf{x}_1}{dt} &= \frac{q}{V}\ (\mathbf{u}_2 - \mathbf{x}_1) - \alpha_d\ k\ \mathbf{x}_1 && (12.11)\\
\frac{d\mathbf{x}_2}{dt} &= \frac{q}{V}\ (\mathbf{u}_3 - \mathbf{x}_2) - \alpha_d\ \frac{\Delta H_r\ C_{A_r}}{\rho\ C_p}\ k\ \mathbf{x}_1\ - \frac{U A_R}{\rho\ C_p\ V}\ (\mathbf{x}_2 - \mathbf{u}_4) && (12.12)\\
k &= k_0\ e^{\frac{-E_A}{\mathbf{x}_2\ T_r}} && (12.13)
\end{aligned}$$

In those equations, the quote represent the scaled variables: $C'_A = \frac{C_A}{C_{A_r}}$, $C'_{A0} = \frac{C_{A0}}{C_{A_r}}$ and $T' = \frac{T}{T_r}$.

A set of pseudo-measurements has been obtained by a dynamic simulation of the process. In this simulation the differential equations are integrated using the fourth order Runge-Kutta method with a time step of 0.5 s. The process has been simulated during 1251 seconds and the simulated measurements are given every 10 seconds. Input variables evolve as follow during the simulation: transitions between feed temperature set points have been modeled as first order responses with a 200 seconds time constant and feed concentration transitions with a 50 seconds time constant.

Table 12.10: Stirred tank reactor with heat exchange: input variables evolution (reduced variables)

Times (s)	q cm^3/s	C'_{A0}	T'_0	T'_C
0 s	10.0	6.5	3.5	3.4
101 s	10.0	7.5	3.4	3.4
301 s	10.0	7.0	3.7	3.4
451 s	10.0	6.3	3.8	3.4
651 s	10.0	6.5	3.6	3.4
711 s	10.0	7.5	3.2	3.4
901 s	10.0	7.0	3.4	3.4
1101 s	10.0	6.3	3.7	3.4
1251 s	10.0	7.0	3.5	3.4

A Gaussian noise of 1 % of the true variable value is added to the simulation results of the concentrations and the flowrate and a noise of 0.005 to the reduced temperatures. The initial conditions of the validation problem are listed in the next table 12.11:

Table 12.11: Stirred tank reactor with heat exchange: initial values (reduced variables)

Variables	Initial values	Units
C'_A	0.156	-
T'	4.606	-
q	10.0	cm^3/s
C'_{A0}	6.5	-
T'_0	3.50	-
T'_C	3.40	-

The parameters of the validation window are listed in table 12.12.

Table 12.12: Stirred tank reactor with heat exchange: window parameters

Parameters	Values
h_1	1
h_2	4
h_3	4
h_4	49
h_5	2
Order of Lagrange polynomials	2

12.2.1 Results

The figures 12.34 to 12.39 represent the profiles of the six variables of the reactor for the whole simulation time. For all variables, it can be seen that the validated values follow the profiles of the true values. Nevertheless, the validated values for the input variables have a greater tendency to follow the noise than the validated values for the differential state variables which are distinctly better corrected.

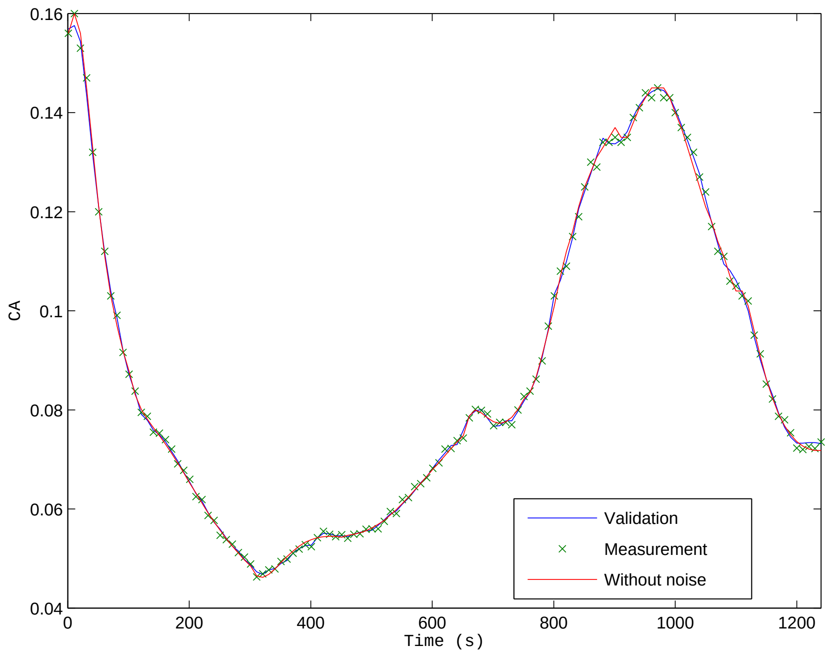


Figure 12.34: Stirred tank reactor with heat exchange: concentration: profile

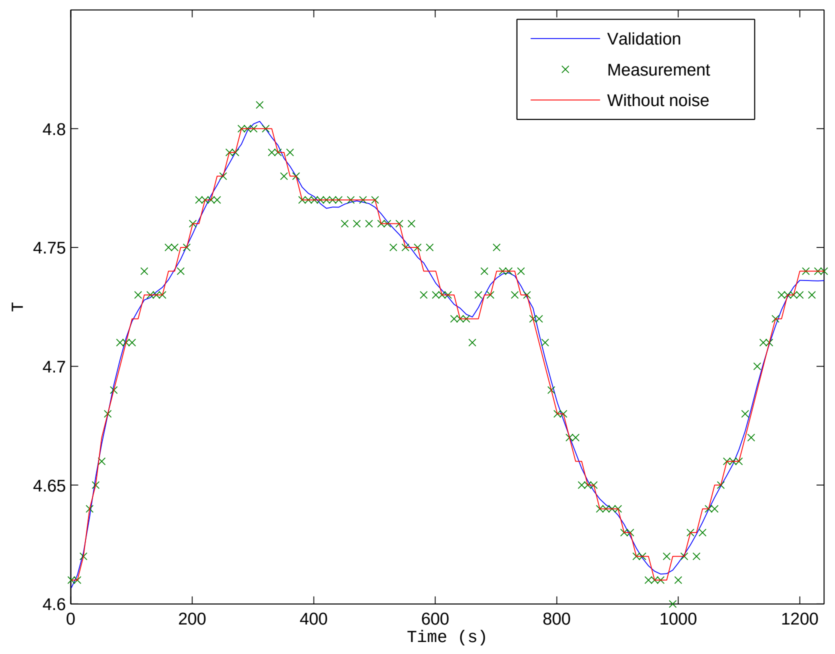


Figure 12.35: Stirred tank reactor with heat exchange: temperature: profile

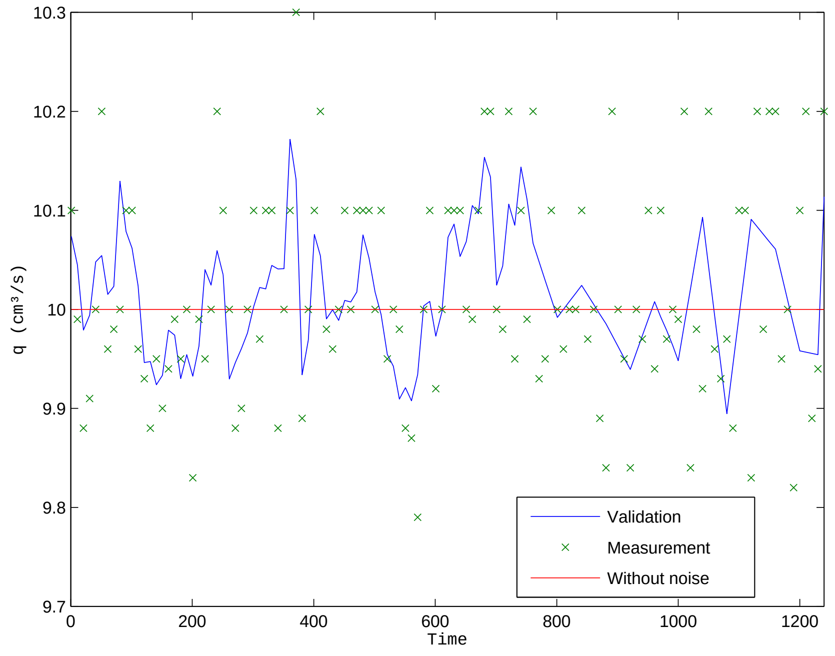


Figure 12.36: Stirred tank reactor with heat exchange: feed flowrate: profile

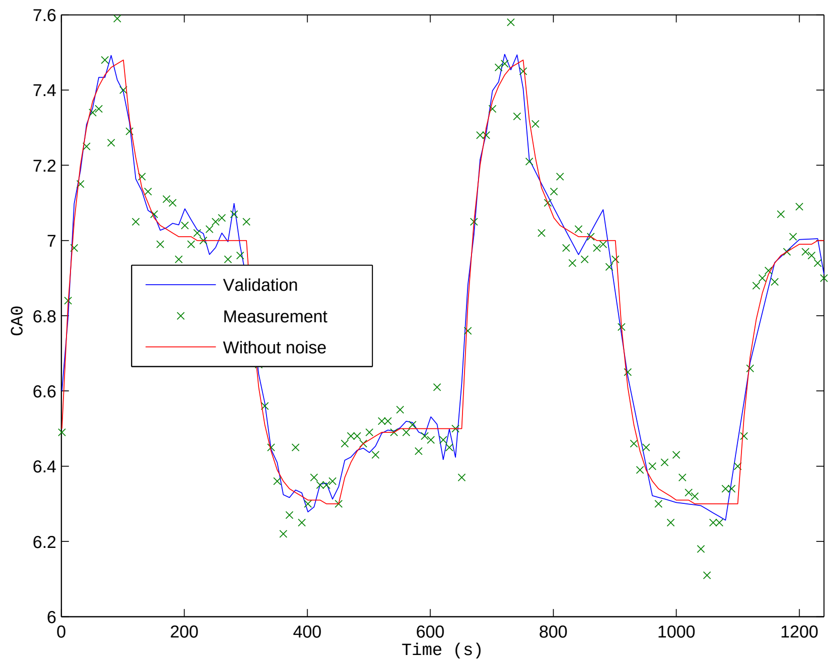


Figure 12.37: Stirred tank reactor with heat exchange: feed concentration: profile

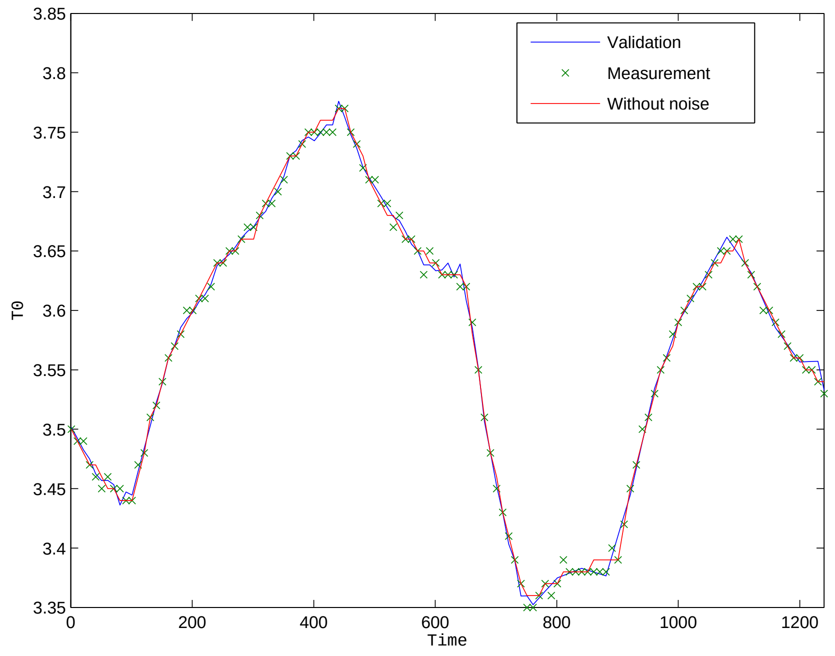


Figure 12.38: Stirred tank reactor with heat exchange: feed temperature: profile

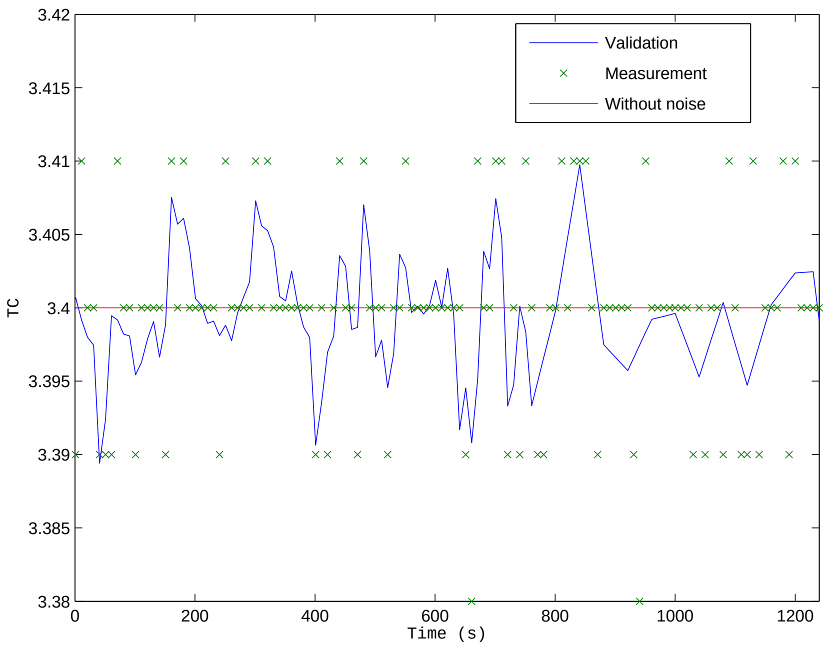


Figure 12.39: Stirred tank reactor with heat exchange: cooling temperature: profile

In the next six figures (figures 12.40 to 12.45), the a priori and a posteriori standard deviations are compared for the same validation window. On those figures, error bars represent once the standard deviation of the variables. For some measurement times of the validation window, a priori and a posteriori standard deviations are listed in tables 12.14 to 12.19.
It appears that a posteriori standard deviations are distinctly reduced compared to a priori ones for the six variables. As for the previous examples, one can note that the true values are usually but not always included within one a posteriori standard deviations of the validated values, as expected for a Gaussian distribution.
For all variables, standard deviations are less reduced for the first and the last measurements times of the validation window
In the next table 12.13, a posteriori standard deviation reduction mean factors are listed with their standard deviations. It appears that the mean reduction factors are of the same order except for the feed flowrate for which it is very high. The standard deviation of the reduction factors is about 5.5 times less than the mean reductions factors except for the feed flowrate (2.5 times).

Table 12.13: Stirred tank reactor with heat exchange: a posteriori standard deviation reduction mean factors

Variables	A posteriori standard deviation reduction factors: means	A posteriori standard deviation reduction factors: standard deviations
C_A	2.0	0.37
T	6.0	1.1
q	54	22
C_{A0}	2.1	0.37
T_0	2.1	0.37
T_C	2.1	0.37

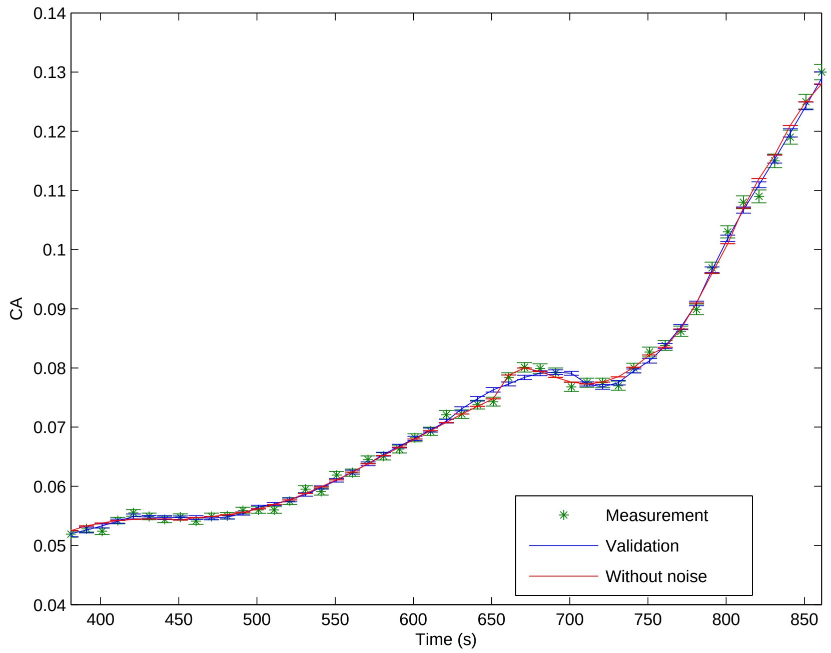


Figure 12.40: Stirred tank reactor with heat exchange: concentration: standard deviation comparison

Table 12.14: Stirred tank reactor with heat exchange: concentration: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	$5.28\ 10^{-4}$	$3.33\ 10^{-4}$	1.6
451	$5.48\ 10^{-4}$	$2.93\ 10^{-4}$	1.9

continued on next page

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
501	$5.60\ 10^{-4}$	$2.92\ 10^{-4}$	1.9
551	$6.19\ 10^{-4}$	$2.84\ 10^{-4}$	2.2
601	$6.82\ 10^{-4}$	$3.58\ 10^{-4}$	1.9
651	$7.43\ 10^{-4}$	$3.78\ 10^{-4}$	2.0
701	$7.68\ 10^{-4}$	$3.30\ 10^{-4}$	2.3
751	$8.27\ 10^{-4}$	$3.78\ 10^{-4}$	2.2
801	$1.03\ 10^{-3}$	$5.39\ 10^{-4}$	1.9
851	$1.25\ 10^{-3}$	$6.44\ 10^{-4}$	1.9

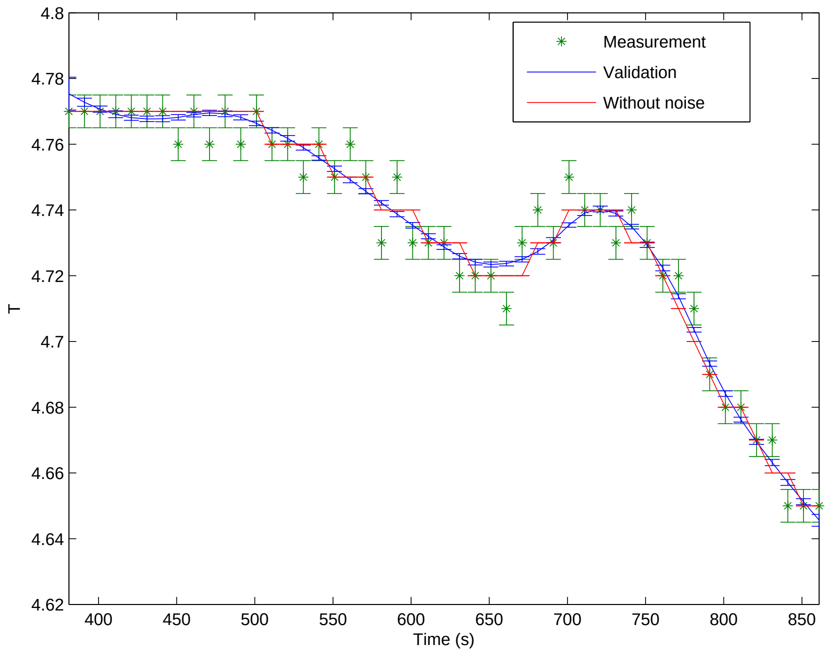


Figure 12.41: Stirred tank reactor with heat exchange: temperature: standard deviation comparison

Table 12.15: Stirred tank reactor with heat exchange: temperature: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	$5.0\ 10^{-3}$	$9.46\ 10^{-4}$	5.3
451	$5.0\ 10^{-3}$	$8.28\ 10^{-4}$	6.0
501	$5.0\ 10^{-3}$	$6.98\ 10^{-4}$	7.2
551	$5.0\ 10^{-3}$	$8.11\ 10^{-4}$	6.2

continued on next page

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
601	$5.0\ 10^{-3}$	$8.59\ 10^{-4}$	5.8
651	$5.0\ 10^{-3}$	$7.68\ 10^{-4}$	6.5
701	$5.0\ 10^{-3}$	$6.89\ 10^{-4}$	7.2
751	$5.0\ 10^{-3}$	$8.04\ 10^{-4}$	6.2
801	$5.0\ 10^{-3}$	$8.63\ 10^{-4}$	5.8
851	$5.0\ 10^{-3}$	$9.16\ 10^{-4}$	5.4

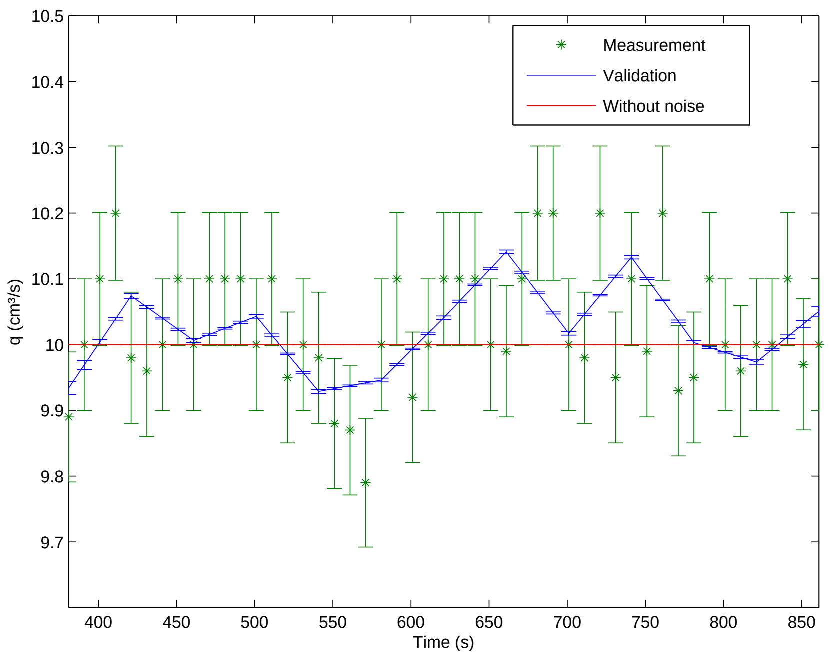


Figure 12.42: Stirred tank reactor with heat exchange: feed flowrate: standard deviation comparison

Table 12.16: Stirred tank reactor with heat exchange: feed flowrate: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	0.10	$3.75\ 10^{-3}$	27
451	0.10	$1.77\ 10^{-3}$	56
501	0.10	$2.93\ 10^{-3}$	34
551	0.10	$1.75\ 10^{-3}$	57
601	0.10	$1.11\ 10^{-3}$	90
651	0.10	$1.67\ 10^{-3}$	60

continued on next page

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
701	0.10	$2.76\ 10^{-3}$	36
751	0.10	$1.65\ 10^{-3}$	61
801	0.10	$1.20\ 10^{-3}$	83
851	0.10	$5.05\ 10^{-3}$	20

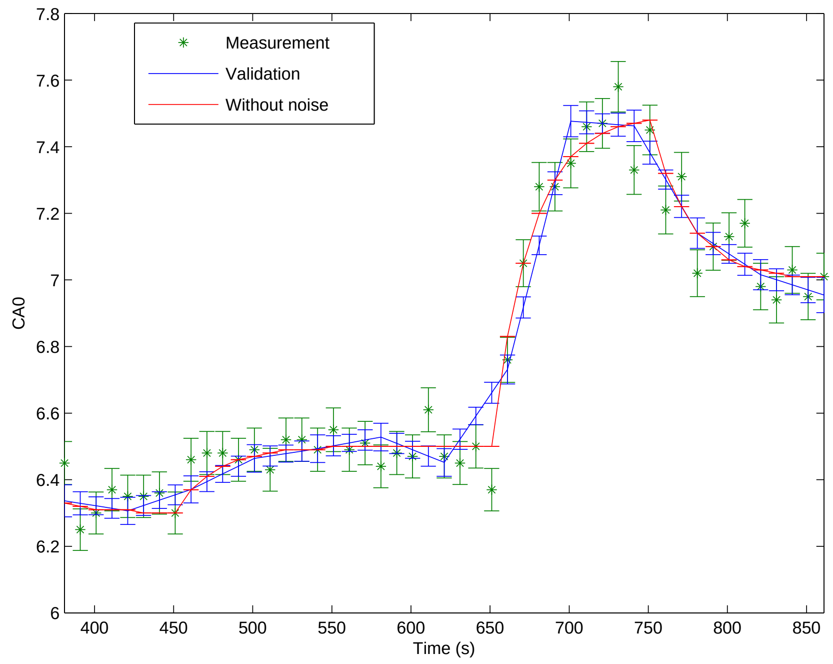


Figure 12.43: Stirred tank reactor with heat exchange: feed concentration: standard deviation comparison

Table 12.17: Stirred tank reactor with heat exchange: feed concentration: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	$6.30\ 10^{-2}$	$2.68\ 10^{-2}$	2.4
451	$6.30\ 10^{-2}$	$2.98\ 10^{-2}$	2.1
501	$6.49\ 10^{-2}$	$4.13\ 10^{-2}$	1.6
551	$6.55\ 10^{-2}$	$3.03\ 10^{-2}$	2.2
601	$6.47\ 10^{-2}$	$2.56\ 10^{-2}$	2.5
651	$6.37\ 10^{-2}$	$3.16\ 10^{-2}$	2.0
701	$7.35\ 10^{-2}$	$4.71\ 10^{-2}$	1.6
751	$7.45\ 10^{-2}$	$3.46\ 10^{-2}$	2.2

continued on next page

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
801	$7.13\ 10^{-2}$	$2.80\ 10^{-2}$	2.5
851	$6.95\ 10^{-2}$	$3.81\ 10^{-2}$	1.8

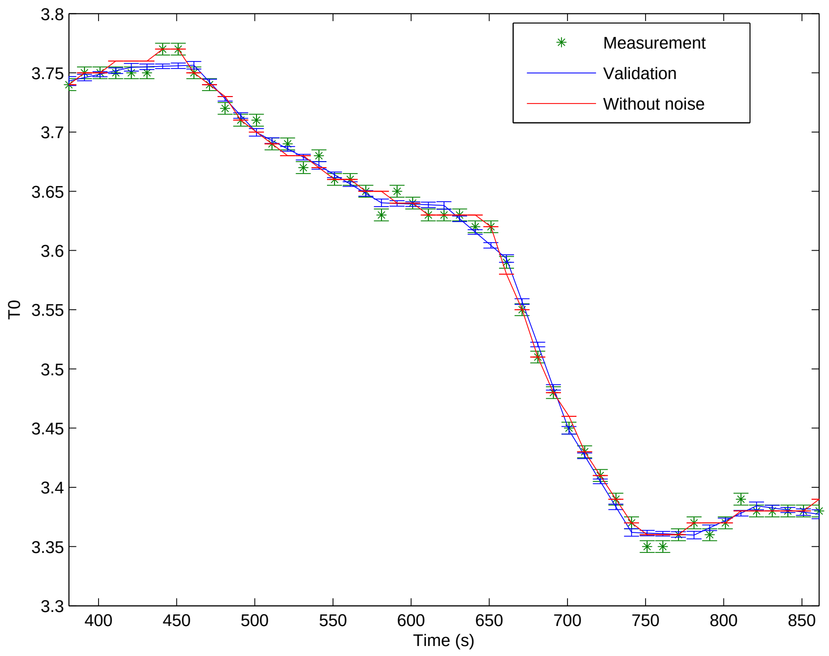


Figure 12.44: Stirred tank reactor with heat exchange: feed temperature: standard deviation comparison

Table 12.18: Stirred tank reactor with heat exchange: feed temperature: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	$5.0\ 10^{-3}$	$2.12\ 10^{-3}$	2.4
451	$5.0\ 10^{-3}$	$2.34\ 10^{-3}$	2.2
501	$5.0\ 10^{-3}$	$3.19\ 10^{-3}$	1.6
551	$5.0\ 10^{-3}$	$2.34\ 10^{-3}$	2.2
601	$5.0\ 10^{-3}$	$1.97\ 10^{-3}$	2.5
651	$5.0\ 10^{-3}$	$2.34\ 10^{-3}$	2.2
701	$5.0\ 10^{-3}$	$3.19\ 10^{-3}$	1.6
751	$5.0\ 10^{-3}$	$2.34\ 10^{-3}$	2.2
801	$5.0\ 10^{-3}$	$1.98\ 10^{-3}$	2.5
851	$5.0\ 10^{-3}$	$2.74\ 10^{-3}$	1.8

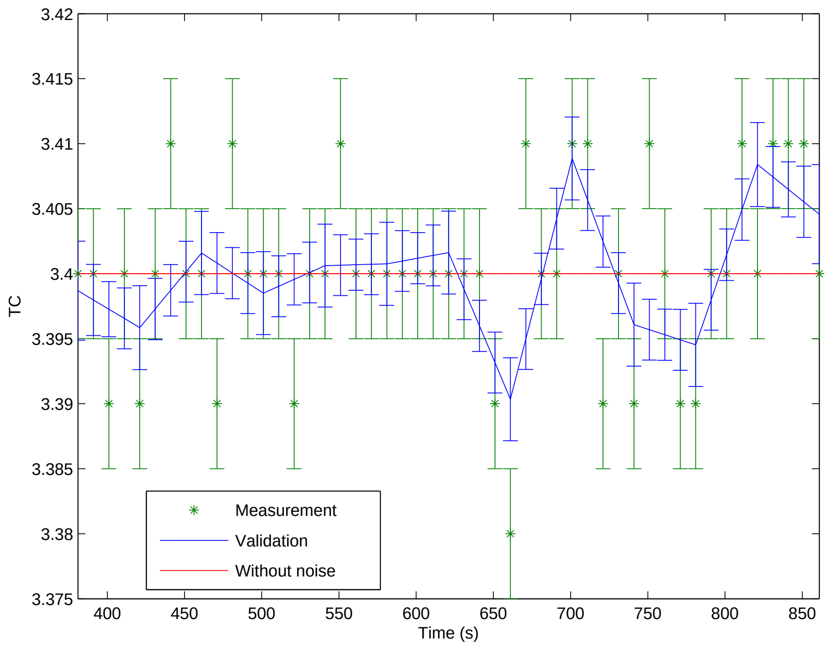


Figure 12.45: Stirred tank reactor with heat exchange: cooling temperature: standard deviation comparison

Table 12.19: Stirred tank reactor with heat exchange: cooling temperature: a posteriori standard deviation reduction factors

Times	A priori standard deviation	A posteriori standard deviation	Reduction factors
401	5.0 10−3	2.12 10^{-3}	2.4
451	5.0 10^{-3}	2.34 10^{-3}	2.2
501	5.0 10^{-3}	3.19 10^{-3}	1.6
551	5.0 10^{-3}	2.34 10^{-3}	2.2
601	5.0 10^{-3}	1.97 10^{-3}	2.5
651	5.0 10^{-3}	2.34 10^{-3}	2.2
701	5.0 10^{-3}	3.19 10^{-3}	1.6
751	5.0 10^{-3}	2.34 10^{-3}	2.2
801	5.0 10^{-3}	1.98 10^{-3}	2.5
851	5.0 10^{-3}	2.74 10^{-3}	1.8

The last six figures (figures 12.46 to 12.51) represent the distributions of the errors for all variables. The same four types of errors as before are compared. One can see that the normal noise with zero mean that was imposed for the simulation data is reproduced for the validation distributions (means of errors are nearly equal to zero).
It also appears that the distributions are larger for the measurements errors than the three other ones (except for the feed temperature for which it is quite the same as the three other distributions): error distributions are reduced while validating the data.

Concerning the way of saving the data, the middle curves are better for the differential state variables. For the input variables, it depends of the variables. So, no rule can be inferred from this example.

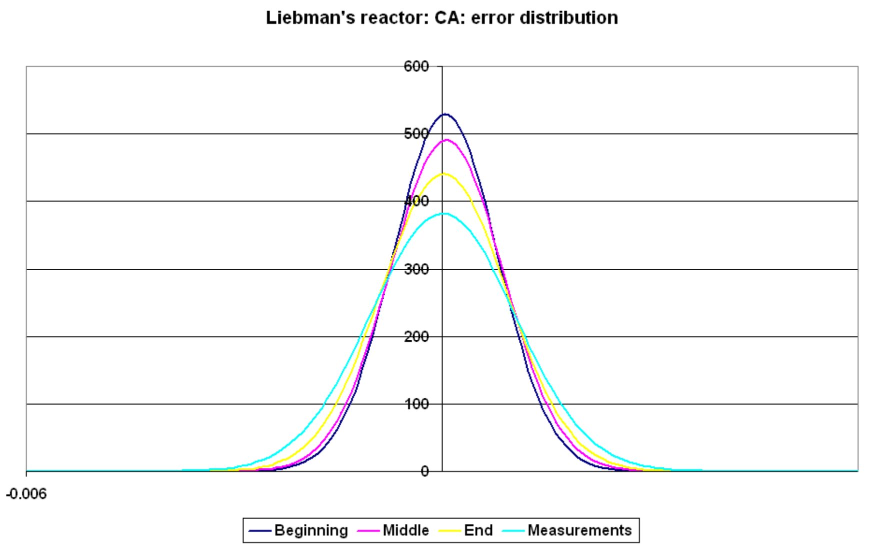


Figure 12.46: Stirred tank reactor with heat exchange: concentration: error distribution

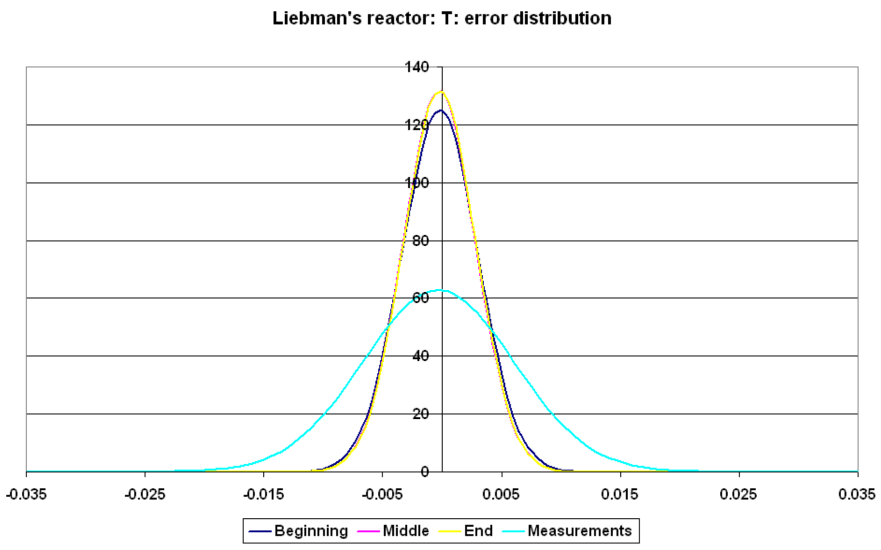


Figure 12.47: Stirred tank reactor with heat exchange: temperature: error distribution

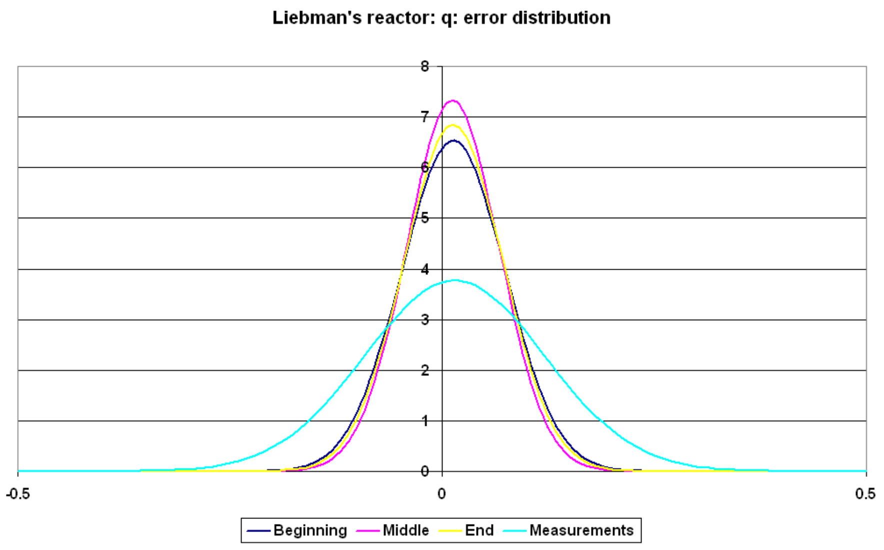


Figure 12.48: Stirred tank reactor with heat exchange: feed flowrate: error distribution

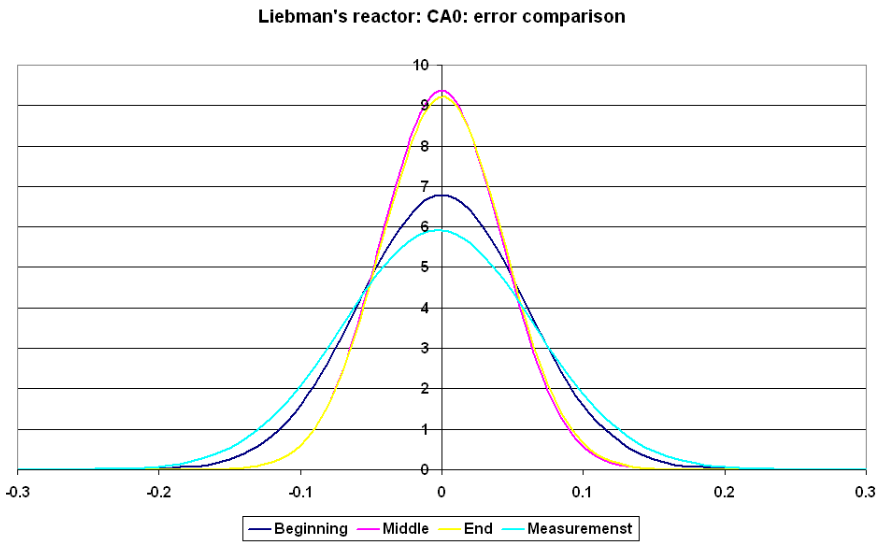


Figure 12.49: Stirred tank reactor with heat exchange: feed concentration: error distribution

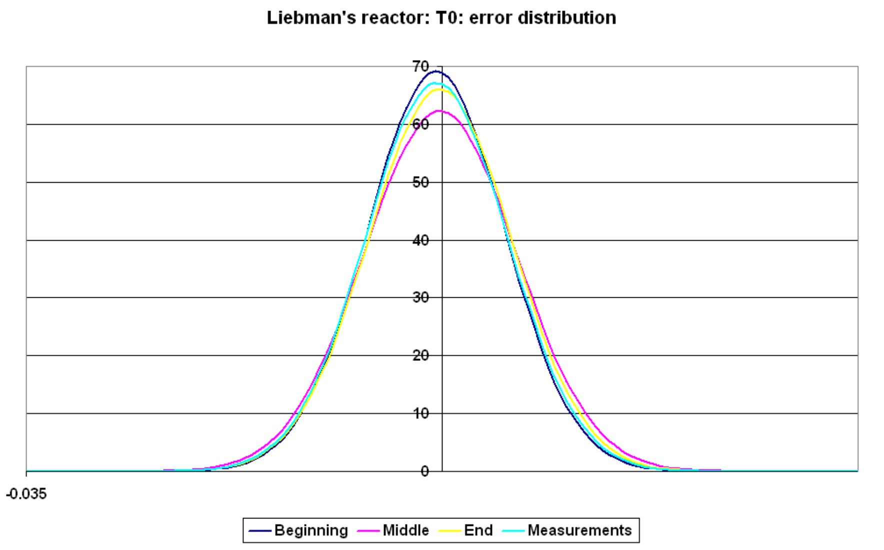


Figure 12.50: Stirred tank reactor with heat exchange: feed temperature: error distribution

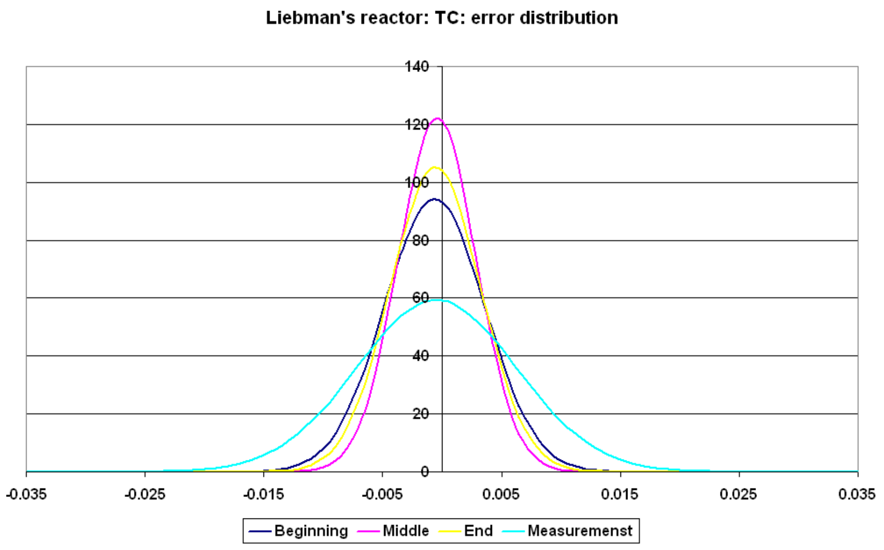


Figure 12.51: Stirred tank reactor with heat exchange: cooling temperature: error distribution

12.3 A network of five tanks

In this last example, five tanks are interconnected. The first tank is fed by stream F_{0A} and the recirculated stream F_{4B} coming from tank 4. The second tank receives the feed stream F_{0B} and the stream F_{1A} coming from tank 1. The third tank receives the feed stream F_{0C} and the stream F_{1B} coming from tank 1. The fourth tank is feeded by stream F_{3B} coming from tank 3. The fifth tank receives stream F_2 from tank 2 and stream F_{3A} from tank 3. This example is represented on figure 12.52.

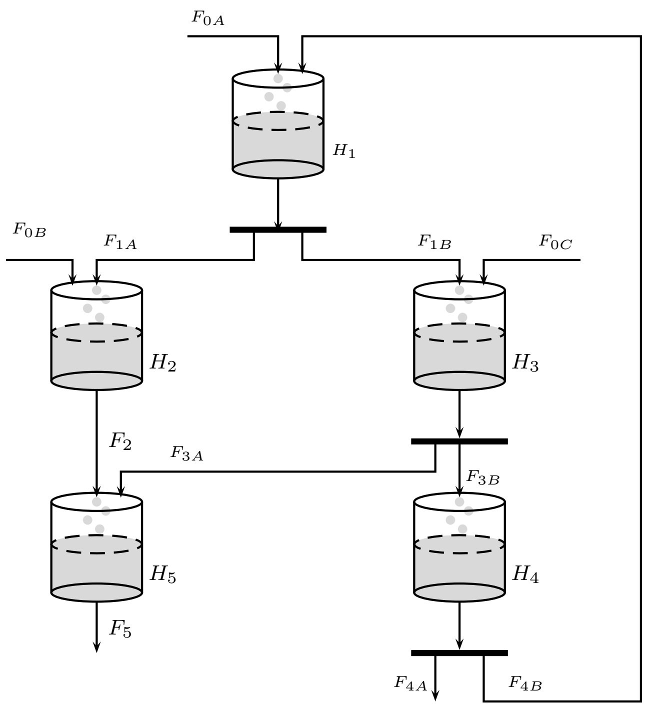


Figure 12.52: Flowsheet of the network of five tanks

The model is defined by 13 equations: 5 differential equations and 8 link equations:

$$\frac{dh_1}{dt} = \frac{F_{0A}+F_{4B}}{A_1} - \frac{F_{1A}+F_{1B}}{A_1} \tag{12.14}$$
$$\frac{dh_2}{dt} = \frac{F_{0B}+F_{1A}}{A_2} - \frac{F_2}{A_2} \tag{12.15}$$
$$\frac{dh_3}{dt} = \frac{F_{0C}+F_{1B}}{A_3} - \frac{F_{3A}+F_{3B}}{A_3} \tag{12.16}$$
$$\frac{dh_4}{dt} = \frac{F_{3B}}{A_4} - \frac{F_{4A}+F_{4B}}{A_4} \tag{12.17}$$
$$\frac{dh_5}{dt} = \frac{F_2+F_{3A}}{A_5} - \frac{F_5}{A_5} \tag{12.18}$$
$$F_{1A} = \beta_1\ H_1\ \alpha_1 \tag{12.19}$$
$$F_{1B} = \beta_1\ H_1\ (1-\alpha_1) \tag{12.20}$$
$$F_2 = \beta_2\ H_2 \tag{12.21}$$
$$F_{3A} = \beta_1\ H_3\ \alpha_3 \tag{12.22}$$
$$F_{3B} = \beta_3\ H_3\ (1-\alpha_3) \tag{12.23}$$
$$F_{4A} = \beta_4\ H_4\ \alpha_4 \tag{12.24}$$
$$F_{4A} = \beta_4\ H_4\ (1-\alpha_4) \tag{12.25}$$
$$F_5 = \beta_5\ H_5 \tag{12.26}$$

$$\Updownarrow$$

$$\frac{d\mathbf{x}_1}{dt} = \frac{\mathbf{u}_1+\mathbf{z}_7}{A_1} - \frac{\mathbf{z}_1+\mathbf{z}_2}{A_1} \tag{12.27}$$
$$\frac{d\mathbf{x}_2}{dt} = \frac{\mathbf{u}_2+\mathbf{z}_1}{A_2} - \frac{\mathbf{z}_3}{A_2} \tag{12.28}$$
$$\frac{d\mathbf{x}_3}{dt} = \frac{\mathbf{u}_3+\mathbf{z}_2}{A_3} - \frac{\mathbf{z}_4+\mathbf{z}_5}{A_3} \tag{12.29}$$
$$\frac{d\mathbf{x}_4}{dt} = \frac{\mathbf{z}_5}{A_4} - \frac{\mathbf{z}_6+\mathbf{z}_7}{A_4} \tag{12.30}$$
$$\frac{d\mathbf{x}_5}{dt} = \frac{\mathbf{z}_3+\mathbf{z}_4}{A_5} - \frac{\mathbf{z}_8}{A_5} \tag{12.31}$$
$$\mathbf{z}_1 = \beta_1\ \mathbf{x}_1\ \alpha_1 \tag{12.32}$$
$$\mathbf{z}_2 = \beta_1\ \mathbf{x}_1\ (1-\alpha_1) \tag{12.33}$$
$$\mathbf{z}_3 = \beta_2\ \mathbf{x}_2 \tag{12.34}$$
$$\mathbf{z}_4 = \beta_3\ \mathbf{x}_3\ \alpha_3 \tag{12.35}$$
$$\mathbf{z}_5 = \beta_3\ \mathbf{x}_3\ (1-\alpha_3) \tag{12.36}$$
$$\mathbf{z}_6 = \beta_4\ \mathbf{x}_4\ \alpha_4 \tag{12.37}$$
$$\mathbf{z}_7 = \beta_4\ \mathbf{x}_4\ (1-\alpha_4) \tag{12.38}$$
$$\mathbf{z}_8 = \beta_5\ \mathbf{x}_5 \tag{12.39}$$

This example counts sixteen variables:

- three input variables: the feed flowrates F_{0A}, F_{0B} and F_{0C};
- five differential state variables: the heights in the tanks , H_1, H_2, H_3, H_4 and H_5;
- eight algebraic variables: the flowrates F_{1A}, F_{1B}, F_2, F_{3A}, F_{3B}, F_{4A}, F_{4B} and F_5.

The model counts thirteen parameters fixed as follow:

Table 12.20: Network of five tanks: model parameters

Parameters	Values	Units
A_1	8	dm^2
A_2	10	dm^2
A_3	5	dm^2
A_4	10	dm^2
A_5	7	dm^2
β_1	1.2	-
β_2	0.9	-
β_3	1.0	-
β_4	1.5	-
β_5	1.1	-
α_1	0.5	-
α_2	0.3	-
α_3	0.8	-

A data set has been obtained by a dynamic simulation of the process. In this simulation differential equations are integrated by means of the fourth order Runge-Kutta method with a step of 1 s. The process has been simulated during 261 seconds and the simulated measurements are given every second. Feed flowrates evolve as follow during the simulation: transitions between feed flowrate set points have been modeled as first order responses with a 10 seconds time constant.

Table 12.21: Network of five tanks: feed flowrates evolution

Times (s)	F_{0A} (dm^3/s)	F_{0B} (dm^3/s)	F_{0C} (dm^3/s)
0 s	50	30	20
31 s	100	30	20
81 s	60	30	20
111 s	60	20	40

continued on next page

Times (s)	F_{0A} (dm^3/s)	F_{0B} (dm^3/s)	F_{0C} (dm^3/s)
151 s	40	35	20
211 s	60	40	10
261 s	80	35	20

A Gaussian noise (zero mean and standard deviation set at 2% of the true value) is added to the simulation results of the flowrates. Similarly, Gaussian noise (standard deviation set at 2 cm) is added to the simulated height before carrying out the data reconciliation. The initial conditions of the validation problem are listed in the next table 12.22:

Table 12.22: Network of five tanks: initial values

Variables	Initial values	Units
H_1	42.0	dm
H_2	12.0	dm
H_3	39.0	dm
H_4	33.0	dm
H_5	25.0	dm
F_{0A}	50.0	dm^3/s
F_{0B}	30.0	dm^3/s
F_{0C}	20.0	dm^3/s
F_{1A}	25.2.	dm^3/s
F_{1B}	25.2	dm^3/s
F_2	10.8	dm^3/s
F_{3A}	10.7	dm^3/s
F_{3B}	27.3	dm^3/s
F_{4A}	39.6	dm^3/s
F_{4B}	9.9	dm^3/s
F_5	27.5	dm^3/s

The parameters of the validation windows are listed in table 12.23.

Table 12.23: Network of five tanks: window parameters

Parameters	Values
h_1	1
h_2	4
h_3	4
h_4	49
h_5	2

continued on next page

Parameters	Values
Order of Lagrange polynomials	2

12.3.1 Results

For all variables, the validated values follow the curves of the simulation without noise and the measurements are corrected. The profiles of the differential state variables are represented on figures 12.53 to 12.57 for the whole time horizon. The values that are saved are the validated values of the measurements entering the validation window.

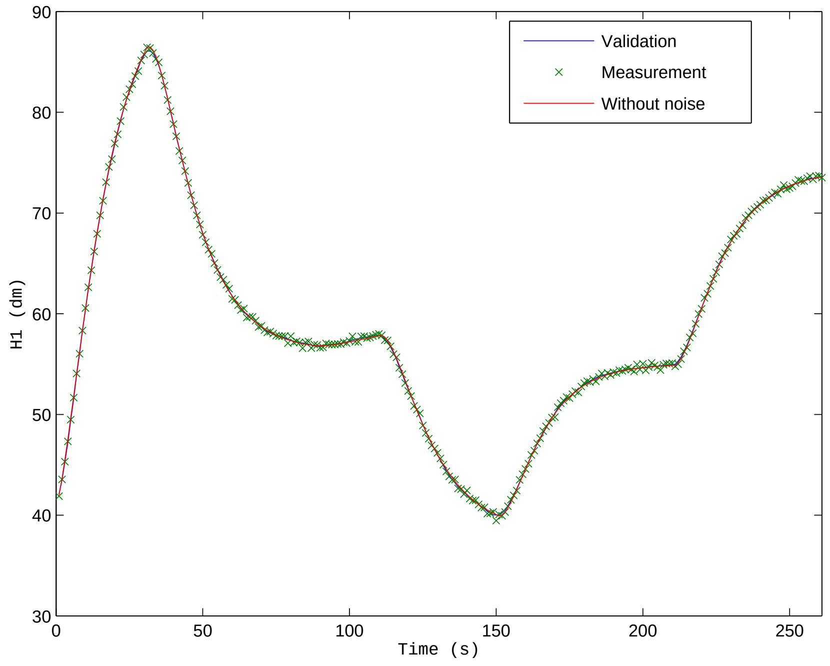


Figure 12.53: Network of five tanks: height H_1: profile

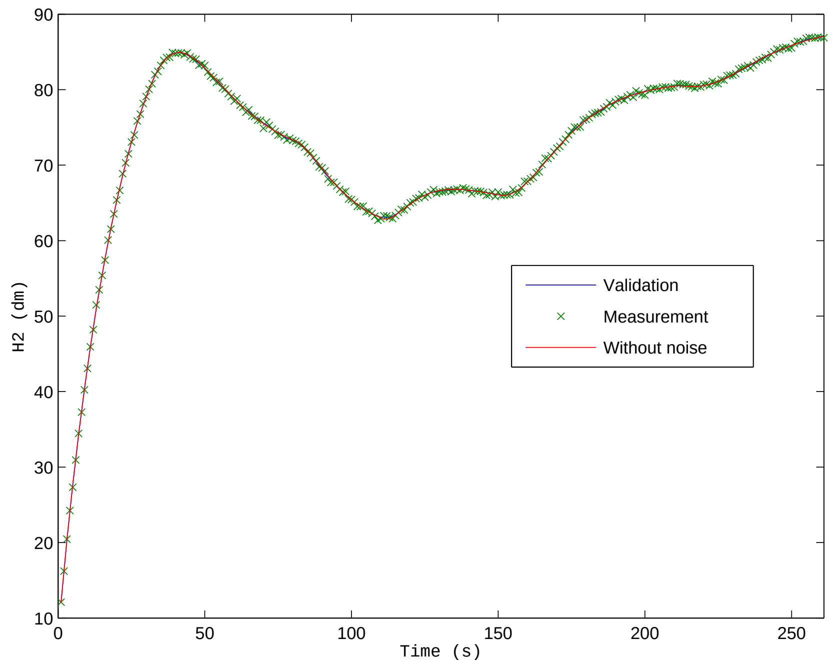


Figure 12.54: Network of five tanks: height H_2: profile

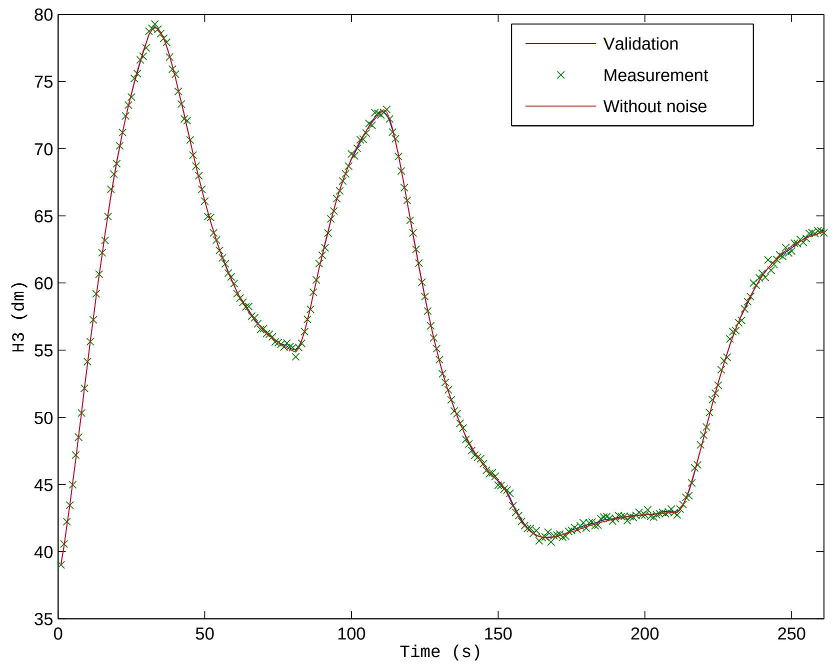


Figure 12.55: Network of five tanks: height H_3: profile

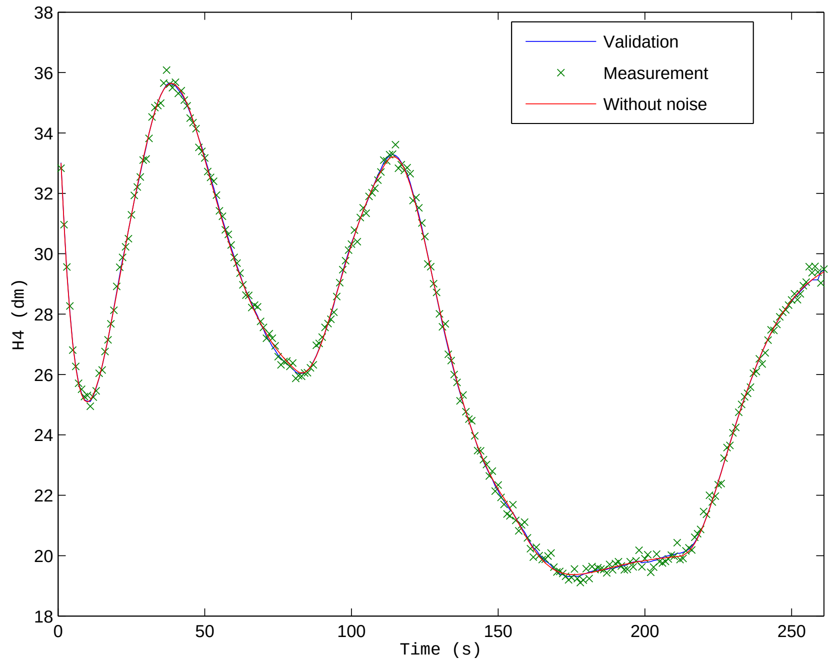


Figure 12.56: Network of five tanks: height H_4: profile

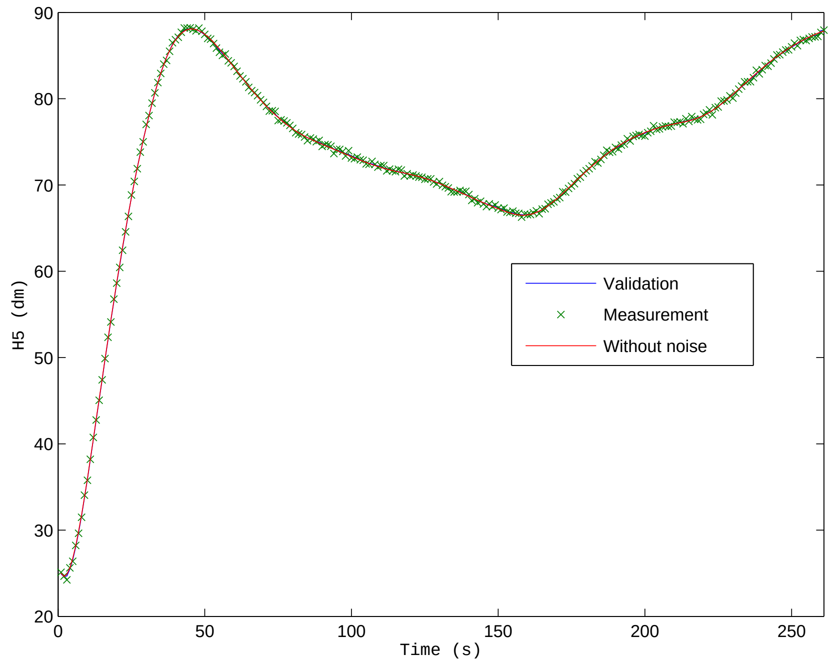


Figure 12.57: Network of five tanks: height H_5: profile

In the next table 12.24, a posteriori standard deviation reduction factors of all variables are listed with their standard deviations. In this example, it appears that the algebraic state variables have better a posteriori standard deviation reduction factors that input and differential state variables. In fact, since algebraic variables and differential variables are linked by the model equations 14.19 to 14.26, the standard deviations of their validated values are similarly linked. However level measurement are more precise (standard deviation = 2 cm and true value $\sim$ 500 cm) than flowrates (standard deviation = 2% of the true value). The reduction factor is better for flowrates.

Table 12.24: Network of five tanks: a posteriori standard deviation reduction mean factors

Variables	A posteriori standard deviation reduction factors: means	A posteriori standard deviation reduction factors: standard deviations
H_1	4.6	1.1
H_2	4.5	0.96
H_3	2.9	0.54
H_4	11	4.1
H_5	6.2	1.6
F_{0A}	6.0	2.5
F_{0B}	2.0	0.39
F_{0C}	2.0	0.39
F_{1A}	23	5.6
F_{1B}	23	5.8
F_2	30	6.6
F_{3A}	17	3.8
F_{3B}	17	3.7
F_{4A}	31	11
F_{4B}	31	11
F_5	44	11

As can been seen in table 12.25, the means of errors still correspond with the normal noises with zero mean that were imposed during the simulation.
Concerning the distributions, one can see they are wider for the measurements, that means that the errors are corrected during the validation. In the case of input variables, the validated values obtained at the end of the window have narrower distributions that the ones at the beginning and middle curves.
In the case of the differential state variables, no clear trend can be identified concerning the best time to keep the validated values.
As expected, the algebraic state variables have distributions similar to ones of the state variables they depend.
No rule concerning the best way of saving the data can be made from this example.

Table 12.25: Network of five tanks: means and standard deviations of errors

Variables	Means				Standard deviations			
	$=\frac{\sum error}{n}$				$=\frac{\sum(error\ -\ mean\ error)^2}{n-1}$			
	Beginning	Middle	End	Meas.	Beginning	Middle	End	Meas.
H_1	-0.003	-0.008	-0.018	-0.012	0.10	0.096	0.10	0.19
H_2	-0.018	-0.018	-0.014	-0.020	0.087	0.059	0.063	0.21
H_3	0.022	0.021	0.019	0.024	0.10	0.079	0.072	0.20
H_4	-0.010	0.009	0.003	-0.009	0.024	0.025	0.049	0.18
H_5	-0.012	-0.013	-0.005	-0.011	0.033	0.038	0.065	0.20
F_{0A}	0.062	0.040	0.037	0.080	0.71	0.59	0.54	1.2
F_{0B}	-0.007	-0.002	-0.015	-0.019	0.42	0.27	0.21	0.71
F_{0C}	0.024	0.027	0.024	0.012	0.28	0.28	0.23	0.42
F_{1A}	-0.002	-0.005	-0.010	0.042	0.063	0.058	0.060	0.70
F_{1B}	-0.002	-0.005	-0.012	-0.046	0.063	0.058	0.060	0.72
F_2	-0.016	-0.016	-0.006	-0.094	0.079	0.053	0.057	1.3
F_{3A}	0.007	0.006	0.038	0.038	0.030	0.024	0.022	0.35
F_{3B}	0.016	0.015	0.013	-0.033	0.071	0.056	0.050	0.85
F_{4A}	0.012	0.010	0.003	-0.0001	0.029	0.030	0.059	0.67
F_{4B}	0.003	0.003	0.001	-0.006	0.007	0.007	0.015	0.16
F_5	-0.013	-0.014	-0.006	-0.043	0.036	0.042	0.072	1.57

12.4 Conclusions

The validation techniques tested in this study allowed to validate the data of all the small examples we studied. The validated profiles match closely the simulated ones. However it appears that the validated profiles of input variables, are more affected by the noise than for the other variables.
Validation allows to reduce the uncertainty (σ) on the measured variables. However, for all studied examples, the procedure is less effective with respect to input variables. When measurement errors follow a normal, zero mean distribution, a similar distribution is observed for validation errors. As expected, the spread of validation errors is narrower, so that validation allows to better know the variables. There is no clearcut rule allowing to predict which validation estimate is the most reliable when a measurement is used several times in successive validation runs. Typically the results obtained by the first validation, as soon as the measurement becomes available, will be used for process control. Results calculated when the measurement is in the middle of the window might be more reliable and used for archival.
For all variables of all examples, a posteriori standard deviations are distinctly reduced compared to a priori ones, but true values are not always included in error bars drawn around the validated values. A posteriori standard deviation at the beginning and at the end of the validation window are larger that in the middle of the window where the values

of the variables are better known.
The validation being less efficient for input variables, we suggest to replace their linear interpolation by orthogonal collocations. Doing that, it should be decided if the order of the polynomials and the size of the discretization intervals should be the same as for the differential state variables.
One can also choose to place the nodes of orthogonal collocations at the measurement times, even if it could be less precise, since it would reduce the computing ressources needed.

Chapter 13

Conclusions part II

Several techniques have been tested to reconcile dynamic processes: the extended Kalman filter, the integration of the differential equation by means of Runge-Kutta method coupled with optimization algorithms (Davidon, 1975) or SQPIP (Kyriakopoulou, 1997), the discretization of differential equations by means of orthogonal collocations coupled with the same optimization methods.
From the cases that were studied, it appeared that, even if all methods allow reconciling the data, the simultaneous method using orthogonal collocations was the best one in our case: other methods were not able to achieve the minimum of the goal function, probably as a consequence of numerical noise or limited precision in the approximations of derivatives by numerical differentiation. Moreover, very non linear problems can not be solved properly using Kalman filters. With the design of sensor network for dynamic process as an objective, one has to know a posteriori uncertainty on input variables. The problem with filtering method is that they consider the input variables as perfectly known so that they are not corrected and their a posteriori uncertainty can not be estimate. Concerning the explicit integration, the main problem is that there does not exists a method allowing to estimate a posteriori variances. This could be obtained by integrating sensitivity equations in parallel with the model equations, but this strategy would be difficult to generalized and implement. In the case of orthogonal collocations, as the differential equations are discretized, we could transpose the method of a posteriori variances estimations developed for the stationary case (Heyen et al., 1996).
Good results have been obtained with orthogonal collocations for all small examples treated: the reconciled values follow the simulated ones, a posteriori variances are smaller than a priori ones (as expected) and are better inside the reconciliation window than for the first and the last measurement times, uncertainties are reduced.
Nevertheless there remains one problem with input variables, which follow more the noise than other variables. Using orthogonal collocations also for inputs variables could allow to address that problem.

Part III

Networks design for dynamic processes

Chapter 14

Design of sensor networks for dynamic processes

14.1 Introduction

The objective of this third part of the theses is to create an algorithm able to design the cheapest sensor network allowing to observe all process variables and to estimate process key variables within a prescribed accuracy.
Chouaib Benqlilou (Benqlilou et al., 2003), (Benqlilou, 2004), (Benqlilou et al., 2005) solved the problem of sensor placement for dynamic problem in the case where the dynamic reconciliation is made by means of Kalman filter. This implies that input variables are considered as being perfectly known. Only mass balances were solved in his study. He uses genetic algorithm to perform optimization.
The method proposed here after allows to solve more general problems involving mass and energy balances. It is based on the dynamic reconciliation results obtained with the method described in the second part of the theses. The formulation used in this reconciliation technique allows (as presented in chapter 11) to estimate a posteriori variances which are necessary to the sensor placement. One peculiarity of the dynamic version of the sensor placement problem is the consideration of an extra degree of freedom, namely the measurement frequency.

14.2 Observability and variable discretization

As for the stationary case, the observability condition implies that the sensitivity matrix is non-singular. This matrix is defined by the equations 11.15, 11.16 and 11.17. Indeed, if this condition is met, it is possible to estimate all a posteriori variances (see equations 11.23, 11.24 and 11.25). To estimate the variables one has to know the independent terms **-F**, **-F**^{c} and **-G** coming from the linearisation of the constraints in **A**, **A**^{c}, and **D** respectively (see equations 11.18, 11.19 and 11.20). The terms **-F**, **-F**^{c} and **-G** can be estimated by solving the constraints equations that are part of the optimality conditions :

$$
\begin{aligned}
\frac{\partial\ L}{\partial\ \Lambda_{A_{r,j}}} &= \sum_{i=1}^{n_x} \frac{\partial A_{r,j}}{\partial x_{i,j}}\ x_{i,j} + \sum_{i=1}^{n_z} \frac{\partial A_{r,j}}{\partial z_{i,j}}\ z_{i,j} + \sum_{i=1}^{n_u} \frac{\partial A_{r,j}}{\partial u_{i,j}}\ u_{i,j} + F_{r,j} = 0 \qquad \forall j,\ \forall r \\
\left[\frac{\partial\ L}{\partial\ \Lambda_{A^c_{r,j}}}\right]_{discr\ int\ s} &= \left[\sum_{i=1}^{n_{x^c}} \frac{\partial A^c_{r,j}}{\partial x^c_{i,j}}\ x^c_{i,j} + \sum_{i=1}^{n_{z^c}} \frac{\partial A^c_{r,j}}{\partial z^c_{i,j}}\ z^c_{i,j} \right. \\
&\quad \left. + \sum_{i=1}^{n_{u^c}} \frac{\partial A^c_{r,j}}{\partial u^c_{i,j}}\ u^c_{i,j} + F^c_{r,j}\right]_{discr\ int\ s} = 0 \qquad \forall j,\ \forall r,\ \forall s \\
\left[\frac{\partial\ L}{\partial\ \Lambda_{D_{i,k}}}\right]_{discr\ int\ s} &= \left[\sum_{r=0}^{n_\theta} \frac{\partial D_{i,k}}{\partial x^c_{i,r}}\ x^c_{i,r} + \frac{\partial D_{i,k}}{\partial z^c_{i,k}}\ z^c_{i,k} \right. \\
&\quad \left. + \frac{\partial D_{i,k}}{\partial u^c_{i,k}}\ u^c_{i,k} + G_{i,k}\right]_{discr\ int\ s} = 0 \qquad \forall i,\ \forall k,\ \forall s
\end{aligned}
\tag{14.1}
$$

Nevertheless, the optimization algorithm need some information about the initial values of input and differential state variables to start the optimization. Indeed, those values are necessary for the linearisation of the inputs and the creation of the Lagrange's polynomials used in the discretization of differential equations. So, the observability criterion described before can only be verified a posteriori.
If the SQP program starts the optimization with random initial values that can be very far from the solution, it can be very difficult or even impossible for the optimization program to find the solution, especially if the process has a tendency to become unstable. So, we have decided to impose the measurement of the input and differential variables at least once for each reconciliation window.
In the case of sensor network design, one has to ensure that the sensor network is evaluated for the minimum number of measurements that can be carried out with the chosen measurement tools. For example, if the measurement frequency of a sensor is five, a measurement is carried out every fifth time step. For a window containing 49 times, if the first, the second, the third or the fourth time of the window corresponds to the first measurement time, 10 measurements are carried out. But, if the fifth time of the window is the first measurement time of the widow, only 9 measurements are carried out. That's is why it has been decided that if the frequency of a sensor is k, the first measurement will take place at the k^{th} time of the window.

14.3 Method description

The optimal design of sensor network can be decomposed in five steps:

1. Formulation of the reconciliation model and model linearisation;

2. Specification of sensor database, precision requirements and sensor requirements;
3. Verification of the problem feasibility;
4. Optimization of the goal function;
5. Report generation.

The flow diagram of the algorithm is presented on figure 4.1.
The algorithm implemented for steady-state processes has been modified to take the dynamic into account. The modifications are described here after.

14.3.1 Formulation of the reconciliation model and model linearisation

First of all, a dynamic simulation of the process has to be carried out to obtain a data set. The differential equations are integrated by means of the fourth order Runge-Kutta's method. A noise is then added to the simulation values. This noise agrees with the measurement noise that would be expected if the values were plant measurements.
The second part of this step is the reconciliation of the noised data set. This reconciliation is carried out using the moving horizon method that was developed in section 10.2. In this dynamic reconciliation technique, differential equations are discretized by means of orthogonal collocations. At the beginning of the reconciliation, a small file is created. It contains the parameter of the reconciliation window: size of the window, size of the discretizetion and the interpolation intervals, displacement of the window, order of Lagrange's polynomials.
Once the solution is reached for a reconciliation window, the elements of the Jacobian matrix $\boldsymbol{\mathcal{E}}$ (see equation 11.17) of the process model are estimated. A file is created for each reconciliation window. It contains

- the number of process variables: the number of variables contained in the process model multiplied by the number of measurement times contained in the reconciliation window;
- the number of constraints of the reconciliation window: namely constraints $\mathbf{A}$, $\mathbf{A}^c$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{C}^c$, $\mathbf{D}$ and $\mathbf{E}$ defined in 10.2;
- the value, the physical unit and the type (concentration, temperature, level, mass flowrate...) of each process variable;
- the Jacobian matrix $\boldsymbol{\mathcal{E}}$ of the equation system.

To be sure that the sensor network will be able to handle all expected profiles, several of those files are generated. The files are chosen this way:

- the first file to be chosen is the file corresponding to the first reconciliation window;

- for the other files, it is imposed that at least the first measurement time of the chosen window is contained in the previous one. The amount of overlaps between successive windows will depend on the window displacement.

14.3.2 Specification of sensor database, precision requirements and sensor requirements

Besides the files coming from the reconciliation phase, the program needs three data files. Those files contain the sensor database, the precision requirements and the sensor requirements.

Sensor database

The sensor database is a list of sensor from which the algorithm must choose all the possible sensor placements for the studied plant. It must include, for each sensor, the following information:

- the name of the sensor;
- the annualized cost of the sensor. This cost must take into account:
  - the annualized purchase cost;
  - the annualized installation cost;
  - the annualized operating cost.

  All those costs depend on the studied process: indeed, for example, the cost of the sensor will be different if it is placed in a flow of water or if it must resists to acid or base. Moreover, costs evolves with the technologic progress and with the time so that annualized costs will be written in cost units for the studied cases in the next chapter.
- the parameters σ_{A_i} and σ_{B_i} that allow the estimation of the sensor standard deviation σ_j for the measured variable $X_j^{'}$:

$$\sigma_j = \sigma_{A_i} + \sigma_{B_i} * X_j^{'} \tag{14.2}$$

- the minimal and maximal values of the sensor measurement range;
- the type of variable the sensor is able to measure. For exemple:
  - TEMP for a temperature;
  - LEVEL for a level;
  - MASSF for a mass flowrate;

  - CONC for a concentration;

- the frequency of the measurement. If a sensor is listed in the database for several measurement frequencies the costs will decrease with decreasing measurement frequency so that a small frequency is preferentially chosen if it allows to reach the specified performances of the sensor network. This decrease in price can be justified by the fact that sensors used more frequently will generate more data to stock and treat. In some cases (for example chromatographs), they consume reactants or need to be service or calibrated more often.

Precision requirements and sensor requirements

Those files are exactly the same as for the stationary case (see section 4.2).

14.3.3 Verification of the problem feasibility

This step of the program begins by carrying out the list of all the sensors that can be located in the plant and used at each time step. Individuals are created as for the stationary case: the binary genes take the value 1 or 0 depending if the corresponding sensors are chosen or not. Afterwards, the algorithm checks whether there exists a solution to the problem for the first individual containing all possible sensors. If several sensors measure the same variables, only the variance of the most accurate one is taken into account.
To ensure there exists a solution to the studied problem, four conditions have to be met:

- The sensitivity matrix of the problem is non-singular for the ideal thermodynamic case. If this condition is not met, the program stops. This condition is not verified for the examples of the next chapter because thermodynamic models are not used in those examples.

- The sensitivity matrix of the problem is non-singular for the process specific thermodynamic.

- If one refers to the way the dynamic reconciliation program works, initial conditions on input and differential variables have to be known to carry out the dynamic data reconciliation. Thus all input and differential variables must be measured at least once for each chosen reconciliation window. If this is not the case for the first individual, the program may not continue.

- The accuracy target of all key parameters must be achieved for all times of all the reconciliation windows chosen by the program. If this condition is not net, the program may continue, but a penalty is added to the goal function.

There are different ways to cure those problems:

- adding more accurate sensors;

- adding sensors whose measurement frequency is higher;
- adding sensors able to measure other types of variables;
- adding more extra measurable variables with their link equations so that more variables can be measured.

14.3.4 Optimization of the sensor network

Knowing that a feasible solution exists, the search for the optimal sensor configuration can begin. The goal function to maximise is evaluated this way:

- If the sensitivity matrix is singular:

$$\text{Fitness} = -C_{max}\ \text{penalty}_{\text{singular matrix}} \tag{14.3}$$

  where

  - C_{max} is the cost relative to the most expensive sensor network (which corresponds to the first chromosome);
  - $\text{penalty}_{\text{singular matrix}}$ is the penalty factor for a singular sensitivity matrix and is generally chosen equal to 2.

- otherwise

$$\text{Fitness} = -\text{cost} - \sum_{i=1}^{N_{\text{key variables}} \cdot N^{\star}_{t_{mes}}} \begin{cases} -\frac{\sigma_i}{\sigma_i^{\text{target}}} & if \quad \frac{\sigma_i}{\sigma_i^{\text{target}}} \leq 1 \\ \frac{\sigma_i^2}{\left(\sigma_i^{\text{target}}\right)^2} \frac{C_{max}\ \text{penalty}_{\text{target}}}{N_{\text{key variables}}} & if \quad 1 < \frac{\sigma_i}{\sigma_i^{\text{target}}} < 10 \\ 10^2 \frac{C_{max}\ \text{penalty}_{\text{target}}}{N_{\text{key variables}}} & if \quad \frac{\sigma_i}{\sigma_i^{\text{target}}} \geq 10 \end{cases} \tag{14.4}$$

  where

  - $N_{\text{key variables}}$ is the number of process key variables;
  - $N^{\star}_{t_{mes}}$ is the number of measurement times on the moving horizon including the initial time of the horizon;
  - $\text{penalty}_{\text{target}}$ is the penalty factor for the targets on key parameters that are not respected and is fixed equal to 2;
  - σ_i is the accuracy obtained by the sensor network for the key variable i;
  - σ_i^{target} is the accuracy required for the key parameter i.

If the sensor network has to be designed for several reconciliation windows, the fitness becomes:

$$\text{Fitness} = \sum_{j=1}^{N_{\text{windows}}} \text{fitness}_j - (N_{\text{windows}} - 1)\ \text{cost} \tag{14.5}$$

where N_{windows} is the number of reconciliation windows chosen for the sensor network design.
If a measurement system observable in the case of one sensor failure has to be carried out, the fitness is evaluated for all configurations obtained by eliminating successively one of the sensors, and the worst one is kept.
The genetic algorithm developped by Carroll (Carroll, 2001) described in section A for the stationary case is used to perform this optimization.
Individuals of the first generation are chosen randomly, making sure that the number of chosen sensors is at least equal to the number of degrees of freedom of the problem. A high probability of selection if fixed for each sensor. A value of 80 % is typically chosen but this parameter appears not to be critical for the problem of optimal sensor design. For the other parameters of the genetic algorithm, the following values are recommended:

- the population size, generally chosen to 20 individuals;
- the probability of reproduction was fixed to 50 %;
- the probability of single-point cross-over was chosen to 50 %
- the probability of jump mutation after reproduction and cross-over is 1 %.

Once the population has been generated the goal function has to be evaluated to estimate reconciled variances of all variables, the sensitivity has to be inverted. This matrix is very sparse that's why one continues to use Belsim's routine to factorize the matrix.
At each generation, the best individual is kept and duplicated in the case it would mutate during the next generation.
If after a specified number of generations (n) the best chromosome remains unchanged, it is considered has the solution of the problem. There is no certainty that this solution is the best one, but it is available and much better than the first individual.
The computing times to reach the solution with the algorithm can be very important even for small problems likes the ones studied in the next chapter that is why a parallelized version of the code is necessary.

14.3.5 Report generation

The programme finally generates the same reports files as in the stationary case (see section 4.5).

Chapter 15

Case studies

In this chapter, the results of the sensor network design are given for the three cases studied in chapter 12:

- one tank;
- a network of five tanks
- a stirred tank reactor with heat exchange;.

.

15.1 One tank

The flowsheet of this example is represented on figure 15.1.

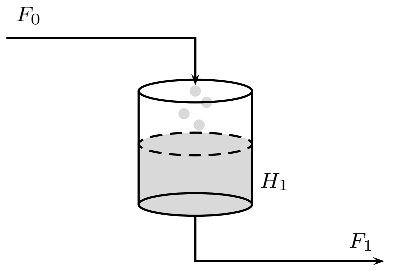


Figure 15.1: Flowsheet of the tank

The model of the example counts three variables (one differential state variable, one input variable and one algebraic variable) related by one differential equation and one link equation (see equations 12.1).
The parameters of the reconciliation window are listed in table 12.4.
For each reconciliation window, one has

- 147 process variables
- 84 collocations variables
- 216 constraints distributed this way:
  - 49 constraints **A**;
  - 24 constraints $\mathbf{A}^c$;
  - 48 constraints **B**;
  - 36 constraints **C**;
  - 24 constraints $\mathbf{C}^c$;
  - 24 constraints **D**;
  - 11 constraints **E**.

The definition of the constraints types is the same as in section 10.2.

The size of the sensitivity matrix is 447 x 447.
For the design of the sensor network, five files created during the reconciliation phase are processed by the programme as described in chapter 14.
The key variable of the process is the differential state variable H_1 corresponding to the level in the tank. Its prescribed accuracy is fixed to 2 cm. As described in the preceding chapter, the input and the state variables have to be measured at least once in the reconciliation window. This concerns the feed flowrate F_0 and the level in the tank H_1 in this case.
The sensor database for this example is listed in table 15.1. As for the stationary case, the costs are in cost units. The cost units have been chosen very expensive. Indeed as the size of the reconciliation window is quite large, many evaluations of a posteriori variances must be compared to the target one (In the present case, there are 147 variables per window, and five windows, thus 735 contributions to the goal function). So, one has to avoid that the absolute value of the sum of those contributions (coming from the variances ratios) is higher than the cost of the cheapest sensor of the sensor database. Indeed, in this case, the sensor network chosen would not necessary be the less expensive one who would allow to estimate key variables within the prescribed accuracy.

Table 15.1: One tank: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values	Measurement periods
1A Flowmeter	9000	2%	20 dm^3/s	75 dm^3/s	1
1B Flowmeter	10000	1%	20 dm^3/s	200 dm^3/s	1
1C Flowmeter	7000	5%	0 dm^3/s	100 dm^3/s	1
1D Flowmeter	12000	0.5%	30 dm^3/s	150 dm^3/s	1
1A Level	3000	0.5 dm	0 dm	200 dm	1
1B Level	4000	0.2 dm	0 dm	200 dm	1
1C Level	5000	0.1 dm	0 dm	200 dm	1
2A Flowmeter	8500	2%	20 dm^3/s	75 dm^3/s	2
2B Flowmeter	9500	1%	20 dm^3/s	200 dm^3/s	2
2C Flowmeter	6500	5%	0 dm^3/s	100 dm^3/s	2
2D Flowmeter	11500	0.5%	30 dm^3/s	150 dm^3/s	2
2A Level	2700	0.5 dm	0 dm	200 dm	2
2B Level	3700	0.2 dm	0 dm	200 dm	2
2C Level	4700	0.1 dm	0 dm	200 dm	2
3A Flowmeter	8200	2%	20 dm^3/s	75 dm^3/s	3
3B Flowmeter	9200	1%	20 dm^3/s	200 dm^3/s	3
3C Flowmeter	6200	5%	0 dm^3/s	100 dm^3/s	3
3D Flowmeter	11200	0.5%	30 dm^3/s	150 dm^3/s	3
3A Level	2500	0.5 dm	0 dm	200 dm	3
3B Level	3500	0.2 dm	0 dm	200 dm	3
3C Level	4500	0.1 dm	0 dm	200 dm	3
4A Flowmeter	7900	2%	20 dm^3/s	75 dm^3/s	4
4B Flowmeter	8900	1%	20 dm^3/s	200 dm^3/s	4
4C Flowmeter	5900	5%	0 dm^3/s	100 dm^3/s	4
4D Flowmeter	10900	0.5%	30 dm^3/s	150 dm^3/s	4
4A Level	2200	0.5 dm	0 dm	200 dm	4
4B Level	3200	0.2 dm	0 dm	200 dm	4
4C Level	4200	0.1 dm	0 dm	200 dm	4
5A Flowmeter	7700	2%	20 dm^3/s	75 dm^3/s	5
5B Flowmeter	8700	1%	20 dm^3/s	200 dm^3/s	5
5C Flowmeter	5700	5%	0 dm^3/s	100 dm^3/s	5
5D Flowmeter	10700	0.5%	30 dm^3/s	150 dm^3/s	5
5A Level	2100	0.5 dm	0 dm	200 dm	5
5B Level	3100	0.2 dm	0 dm	200 dm	5

continued on next page

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values	Measurement periods
5C Level	4100	0.1 dm	0 dm	200 dm	5
10A Flowmeter	7500	2%	20 dm^3/s	75 dm^3/s	10
10B Flowmeter	8500	1%	20 dm^3/s	200 dm^3/s	10
10C Flowmeter	5500	5%	0 dm^3/s	100 dm^3/s	10
10D Flowmeter	10500	0.5%	30 dm^3/s	150 dm^3/s	10
10A Level	2000	0.5 dm	0 dm	200 dm	10
10B Level	3000	0.2 dm	0 dm	200 dm	10
10C Level	4000	0.1 dm	0 dm	200 dm	10
24A Flowmeter	7300	2%	20 dm^3/s	75 dm^3/s	24
24B Flowmeter	8300	1%	20 dm^3/s	200 dm^3/s	24
24C Flowmeter	5300	5%	0 dm^3/s	100 dm^3/s	24
24D Flowmeter	10300	0.5%	30 dm^3/s	150 dm^3/s	24
24A Level	1800	0.5 dm	0 dm	200 dm	24
24B Level	2800	0.2 dm	0 dm	200 dm	24
24C Level	3800	0.1 dm	0 dm	200 dm	24
48A Flowmeter	7100	2%	20 dm^3/s	75 dm^3/s	48
48B Flowmeter	8100	1%	20 dm^3/s	200 dm^3/s	48
48C Flowmeter	5100	5%	0 dm^3/s	100 dm^3/s	48
48D Flowmeter	10100	0.5%	30 dm^3/s	150 dm^3/s	48
48A Level	1600	0.5 dm	0 dm	200 dm	48
48B Level	2600	0.2 dm	0 dm	200 dm	48
48C Level	3600	0.1 dm	0 dm	200 dm	48

With this sensor database, the maximum number of sensors that can be chosen for the process is 56, that is to say a solution space of $2^{56} = 7.2 * 10^{16}$ solutions. This first solution costs 394500 cost units.
The solution found by the program costs 16100 cost units and contains two sensors:

- a 48B Flowmeter is chosen for the feed flowrate;
- the level is measured by a 1B Level sensor.

This solution counts the minimum number of sensors and the maximum frequency of measurement that can be expected. It is the optimum solution of the problem.
The computing time per iteration is about the half of the one for the stationary case because one has only one matrix to invert for each individual (because we do not used thermodynamic models). To avoid very long computing times, we decided to fix the stop criterion to 100 and to restart the computation with new parameters. Indeed the first solution for this small problem is obtained after about 10 minutes on a 1.6 GHz computer. If one is not satisfied with that first sensor network, one can for example restart the optimization algorithm with a reduced sensor database. Indeed, one can already say from the first one that some types of sensor would never be chosen in the optimized solution because of their

cost, precision or measurement period. One can also restart the program with a modified restart file in which the initial worst solution would be replaced by alternative solutions based on the user's intuition and common sense.

15.2 A network of five tanks

The flowsheet of this second example is represented on figure 15.2.

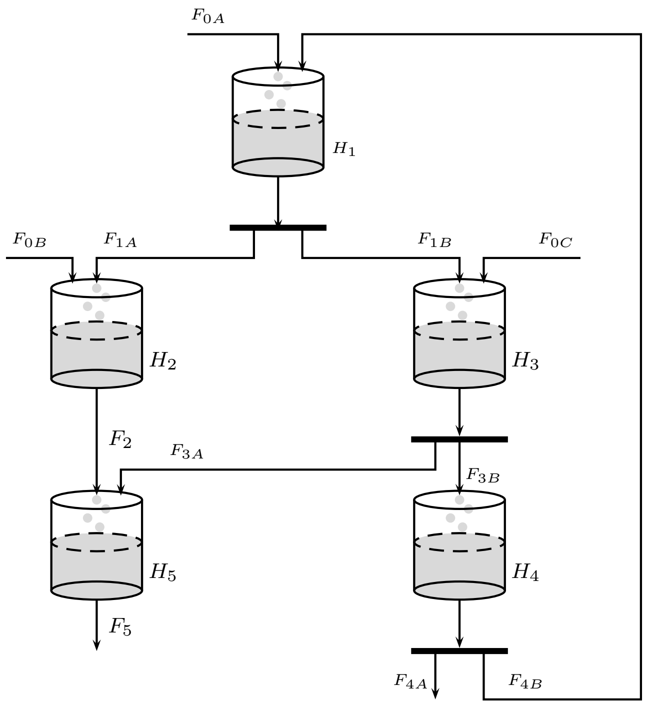


Figure 15.2: Flowsheet of the network of five tanks

The model of the example counts thirteen variables (five differential state variables, three input and eight algebraic variables) related by five differential equations and eight link equations (see equations 12.14).
The parameters of the reconciliation window are listed in table 12.23.
For each reconciliation window, one has

- 784 process variables
- 420 collocations variables
- 1179 constraints distributed this way:
  - 392 constraints $\mathbf{A}$;
  - 192 constraints $\mathbf{A}^c$;
  - 240 constraints $\mathbf{B}$;
  - 108 constraints $\mathbf{C}$;
  - 72 constraints $\mathbf{C}^c$;

  - 120 constraints **D**;
  - 55 constraints **E**.

The size of the sensitivity matrix is 2381 x 2381.
For the design of the sensor network for this last example, three data files created during the reconciliation phase are processed by the programme .
The key variables of the process are the levels in the tanks H_1, H_2, H_3, H_4 and H_5. Their prescribed accuracies are fixed to 2 cm. The levels in all the five tanks and the feed flowrates F_{0A}, F_{0B} and F_{0C} have to measured at least once in each reconciliation window. The sensor database is the same as for the first example (see table 15.1).
The maximum number of sensors that can be chosen for the process is 256, that is to say a solution space of $2^{256} = 1.2 * 10^{77}$ solutions. This first solution costs 1309700 cost units. The solution found by the program costs 91800 cost units and contains the thirteen following sensors:

- 1C Flowmeters are chosen for the flowrates of the streams F_{3A}, F_{3B} and F_{4A};
- 2C Flowmeters are chosen for the flowrates of the streams F_{0B} and F_{1A};
- 3C Flowmeters are chosen for the flowrates of the streams F_{0C} and F_{1B};
- 48C Flowmeters are chosen for the flowrates of the streams F_{0A}, F_2, F_{4B} and F_5;
- the levels in all tanks are measured by 1C Level sensors.

The number of sensors proposed in this solution is much higher than the minimum number of sensors required to measure at least one time for each reconciliation window each input and differential variables. The data files have been modified in a way to constrain the algorithm to choose sensors with smaller measurement periods but he found no cheapest solution than the presented one. This is probably due to the process dynamic. To reduce the computing time, this could be done in two times: first of all looking for sensor locations and then optimizing the measurement periods. The proposed solution is much better than the initial one but it can not be certified that it is the best one.

15.3 Stirred tank reactor with heat exchange

This third example is represented on figure 15.3.

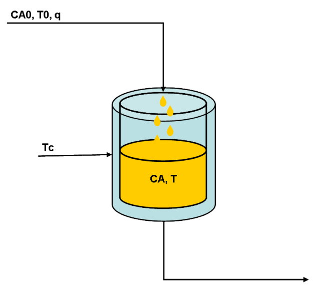


Figure 15.3: Flowsheet of the stirred tank reactor with heat exchange

The model of the example counts six variables (two differential state variables and four input variables) related by two differential equations (see equations 12.8).
The parameters of the reconciliation window are listed in table 12.12.
For each reconciliation window, one has

- 294 process variables
- 168 collocations variables
- 406 constraints distributed this way:
  - 0 constraints **A**;
  - 0 constraints $\mathbf{A}^c$;
  - 96 constraints **B**;
  - 144 constraints **C**;
  - 96 constraints $\mathbf{C}^c$;
  - 48 constraints **D**;
  - 22 constraints **E**.

The size of the sensitivity matrix is 868 x 868.
For the design of the sensor network for this last example, two data files created during the reconciliation phase are processed by the programme .
The key variables of the process are the differential state variables, namely the reduced temperature T and the reduced concentration C_A in the reactor. The prescribed accuracies are fixed to 0.005 and 1 % for the reduced temperature and the reduced concentration respectively. As the process variables are all differential state variables or input variables, all of them have to be measured at least once in each reconciliation window.
The sensor database for this example is listed on table 15.2. As the values of temperatures and concentrations are reduced ones in the reconciliation files, the precision and the measurement areas of the sensors have been changed accordingly.

Table 15.2: Stirred tank reactor with heat exchange: sensor database

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values	Measurement periods
1A Flowmeter	9000	2%	0 dm^3/s	20 dm^3/s	1
1B Flowmeter	10000	1%	0 dm^3/s	20 dm^3/s	1
1C Flowmeter	7000	5%	0 dm^3/s	20 dm^3/s	1
1D Flowmeter	12000	0.5%	0 dm^3/s	20 dm^3/s	1
1A Concentration (reduced)	250000	5%	0	10	1
1B Concentration (reduced)	300000	2%	0	10	1
1C Concentration (reduced)	350000	1%	0	10	1
1D Concentration (reduced)	400000	0.5%	0	10	1
1A Temperature (reduced)	5000	0.01	0	10	1
1A Temperature (reduced)	6000	0.005	0	10	1
1A Temperature (reduced)	7000	0.0025	0	10	1
2A Flowmeter	8500	2%	0 dm^3/s	20 dm^3/s	2
2B Flowmeter	9500	1%	0 dm^3/s	20 dm^3/s	2
2C Flowmeter	6500	5%	0 dm^3/s	20 dm^3/s	2
2D Flowmeter	11500	0.5%	0 dm^3/s	20 dm^3/s	2
2A Concentration (reduced)	245000	5%	0	10	2
2B Concentration (reduced)	295000	2%	0	10	2
2C Concentration (reduced)	345000	1%	0	10	2
2D Concentration (reduced)	395000	0.5%	0	10	2
2A Temperature (reduced)	4700	0.01	0	10	2
2A Temperature (reduced)	5700	0.005	0	10	2
2A Temperature (reduced)	6700	0.0025	0	10	2
3A Flowmeter	8200	2%	0 mol/dm^3	20 mol/dm^3	3
3B Flowmeter	9200	1%	0 mol/dm^3	20 mol/dm^3	3
3C Flowmeter	6200	5%	0 mol/dm^3	20 mol/dm^3	3
3D Flowmeter	11200	0.5%	0 mol/dm^3	20 mol/dm^3	3
3A Concentration (reduced)	242000	5%	0	10	3
3B Concentration (reduced)	292000	2%	0	10	3
3C Concentration (reduced)	342000	1%	0	10	3
3D Concentration (reduced)	392000	0.5%	0	10	3
3A Temperature (reduced)	4500	0.01	0	10	3
3A Temperature (reduced)	5500	0.005	0	10	3
3A Temperature (reduced)	6500	0.0025	0	10	3
4A Flowmeter	7900	2%	0 dm^3/s	20 dm^3/s	4

continued on next page

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values	Measurement periods
4B Flowmeter	8900	1%	0 dm^3/s	20 dm^3/s	4
4C Flowmeter	5900	5%	0 dm^3/s	20 dm^3/s	4
4D Flowmeter	10900	0.5%	0 dm^3/s	20 dm^3/s	4
4A Concentration (reduced)	239000	5%	0	10	4
4B Concentration (reduced)	289000	2%	0	10	4
4C Concentration (reduced)	339000	1%	0	10	4
4D Concentration (reduced)	389000	0.5%	0	10	4
4A Temperature (reduced)	4300	0.01	0	10	4
4B Temperature (reduced)	5300	0.005	0	10	4
4C Temperature (reduced)	6300	0.0025	0	10	4
5A Flowmeter	7700	2%	0 dm^3/s	20 dm^3/s	5
5B Flowmeter	8700	1%	0 dm^3/s	20 dm^3/s	5
5C Flowmeter	5700	5%	0 dm^3/s	20 dm^3/s	5
5D Flowmeter	10700	0.5%	0 dm^3/s	20 dm^3/s	5
5A Concentration (reduced)	237000	5%	0	10	5
5B Concentration (reduced)	287000	2%	0	10	5
5C Concentration (reduced)	337000	1%	0	10	5
5D Concentration (reduced)	387000	0.5%	0	10	5
5A Temperature (reduced)	4100	0.01	0	10	5
5B Temperature (reduced)	5100	0.005	0	10	5
5C Temperature (reduced)	6100	0.0025	0	10	5
10A Flowmeter	7500	2%	0 dm^3/s	20 dm^3/s	10
10B Flowmeter	8500	1%	0 dm^3/s	20 dm^3/s	10
10C Flowmeter	5500	5%	0 dm^3/s	20 dm^3/s	10
10D Flowmeter	10500	0.5%	0 dm^3/s	20 dm^3/s	10
10A Concentration (reduced)	235000	5%	0	10	10
10B Concentration (reduced)	285000	2%	0	10	10
10C Concentration (reduced)	335000	1%	0	10	10
10D Concentration (reduced)	385000	0.5%	0	10	10
10A Temperature (reduced)	3900	0.01	0	10	10
10B Temperature (reduced)	4900	0.005	0	10	10
10C Temperature (reduced)	5900	0.0025	0	10	10
24A Flowmeter	7300	2%	0 dm^3/s	20 dm^3/s	24
24B Flowmeter	8300	1%	0 dm^3/s	20 dm^3/s	24
24C Flowmeter	5300	5%	0 dm^3/s	20 dm^3/s	24
24D Flowmeter	10300	0.5%	0 dm^3/s	20 dm^3/s	24
24A Concentration (reduced)	233000	5%	0	10	24
24B Concentration (reduced)	283000	2%	0	10	24

continued on next page

Sensor types	Annualized costs	Accuracies	Minimum values	Maximum values	Measurement periods
24C Concentration (reduced)	333000	1%	0	10	24
24D Concentration (reduced)	383000	0.5%	0	10	24
24A Temperature (reduced)	3700	0.01	0	10	24
24B Temperature (reduced)	4700	0.005	0	10	24
24C Temperature (reduced)	5700	0.0025	0	10	24
48A Flowmeter	7100	2%	0 dm^3/s	20 dm^3/s	48
48B Flowmeter	8100	1%	0 dm^3/s	20 dm^3/s	48
48C Flowmeter	5100	5%	0 dm^3/s	20 dm^3/s	48
48D Flowmeter	10100	0.5%	0 dm^3/s	20 dm^3/s	48
48A Concentration (reduced)	231000	5%	0	10	48
48B Concentration (reduced)	281000	2%	0	10	48
48C Concentration (reduced)	331000	1%	0	10	48
48D Concentration (reduced)	381000	0.5%	0	10	2
48A Temperature (reduced)	3500	0.01	0	10	2
48B Temperature (reduced)	4500	0.005	0	10	2
48C Temperature (reduced)	5500	0.0025	0	10	2

The maximum number of sensors that can be chosen for the process is 168, resulting in a solution space of $2^{168} = 3.7 * 10^{50}$ solutions. This first solution costs 20739600 cost units. The solution found by the program costs 614100 cost units and contains the six following sensors:

- a 5C flowmeters is chosen for the feed flowrate Q;
- a 3A Concentration (reduced) is chosen for the feed concentration C_{A0} ;
- a 1C Concentration (reduced) is chosen for the concentration in the reactor C_A;
- the feed temperature T_0 is measured by a 2B Temperature (reduced) thermocouple;
- the cooling temperature T is measured by a 1B Temperature (reduced)thermocouple;
- the temperature in the reactor T_C is measured by a 2A Temperature (reduced) thermocouple.

As for the previous example, the solution presented before is the best one that has been obtained. The number of sensors is the smallest that can be expected (because of the fact that input and differential variables have to be measured at least once in each reconciliation window) but we have no guarantee that they represent the global optimal solution for this reactor.

Chapter 16

Conclusions part III

A method has been proposed to carry out the design of sensor networks for dynamic processes. It is based on the estimation of a posteriori variances that can be calculated when differential equations are discretized by means of orthogonal collocations.
This method provides also an observability criterion. To complete the requirements, initial conditions on input and differential variables have to be known to allow dynamic data reconciliation.
The proposed method goes a step further than the one described by Benqlilou, which is based on the state estimation by Kalman filter (Benqlilou et al., 2003), (Benqlilou, 2004), (Benqlilou et al., 2005). Indeed energy balances can be taken into account and inputs need not to be considered as perfectly known. Thus they can be estimated as well as their a posteriori variances. This implies that the variances of the input variables have a contribution to the variances of the other process variables to which they are related. The choice of the sensors that will measure them is thus of a great importance.
The time redundancy is also considered for each measured variable. The measurement frequency allowing to reach the imposed constraints can be optimized.
The method allowed to find a solution for all studied examples. Those solutions are always much better than the initial ones but there is no guarantee global optimal solution has been located.

Chapter 17

General conclusions and future work

This study proposed several variants of a technique allowing sensor network design based on data reconciliation and a posteriori variances estimation. The choice of a sensor being a binary decision, the problem is highly combinatorial. Furthermore, it is non derivable and often multimodal. So, the optimization is carried out with a genetic algorithm.
Important savings could be achieved if such a method was systematically applied before the construction of a new plant. Indeed, people who invest in industries always think to reduce the investment: the sensor networks that are installed are generally reduced so that the minimum control can be carried out. However, some variables whose knowledge seems to be without interest at the first look may reveal to be very important for the knowledge and control of the entire process. The installation of new sensors once the production has been started can be very expensive because of the loss of production due to the plant shut down that may be required during the installation. Moreover the global cost of the new sensor network can then reveal to be more important than the one that would have been determined by design method.
This method has first been developed for steady-state processes. In that case, the sensor network selected by the algorithm should be the cheapest one which allows the computation of all process variables and the estimation of all process key variables within a prescribed accuracy. Process equations are linearized to carry out the data reconciliation problem and the estimation of a posteriori variances, thus non linear processes can be treated as well as linear ones so that energy balances can also be treated. However, one should note that, because of the linearisation, the conclusions concerning the sensor network found by the program for a specified operating point can not be transposed to another operating point. So, when a process has several possible operating points, the sensor network is tested for all of them despite an increase of the computing time. The proposed method gives good results for all studied processes. However, as genetic algorithms are used to carry out the optimization, one can not guarantee that the solution obtained by the algorithm corresponds to the global optimum. One can just note that it is much better than the initial one. The only way to ensure that the optimal solution is reached should be to test all of them. Unfortunately, that would take more than a human being life time.
The computing time appears to be quite long for larger problems, and, due to the sensitivity

matrix inversion, it increases more quickly than the size of the process. That time can be reduced using parallelization techniques. Global parallelization and distributed genetic algorithms allow reducing the computing time. No method appears to outperforms the other for all studied processes. In the case of global parallelization, one can note that the computing time is inversely proportional to the number of processors but, as time devoted to communications increases with the number of processors, the efficiency also decreases when the number of processors increases. This loss of efficiency becomes quickly very important when the number of processors gets close to the number of goal function evaluations per generation. So, to optimize the computing ressources, one has to find a trade off between the computing time and the number of processors. Such phenomena does not appear for distributed genetic algorithm. Indeed, the way followed to find the solution is different for each number of sub-populations. Moreover, in the case of distributed genetic algorithms, when the size of sub-populations is small, it is sometimes better to relax the stop criterion and increase the size of sub-populations by the same factor. So, the user of the parallelized algorithm never knows in advance if he has chosen the fastest method but he is sure to reach the solution within a shorter computing time.
The option of the design of a sensor network that remains observable in the case of one sensor failure has been developed. The advantage of such network is that locating the failing sensor is made easier by the redundancy that has been introduced. Moreover the plant remains controllable despite the failure.

The sensor network design algorithm as been modified to solve the problem of process fault detection and localisation. In that case, the faults that should be detected are simulated and the chosen sensor network has to be able to detect and locate all of them. Leaks in storage tanks and pipes have been simulated in water networks but other faults can be simulated. To detect a fault, the residuals connected to that fault have to be higher than a lower threshold several consecutive times. The necessary number of measurement times to declare a fault is a trade off between the speed of detection and the necessity to differentiate between temporary measurement errors and process faults.

The last variant of the sensor network design algorithm is devoted to dynamic processes. This method being based on the estimation of a posteriori variances, a technique allowing to reconcile process variables at all measurement times and to estimates a posteriori variances in the case of dynamic processes was necessary.
Several dynamic data reconciliation approached were studied:

- one filtering technique: the extended Kalman filter;
- two moving horizon techniques:
  - the explicit integration of differential equations using the fourth order Runge-Kutta method;
  - the discretization of differential equations by orthogonal collocations.

  In those two methods, a successive quadratic programming algorithm is used to carry out the optimization. When orthogonal collocations are applied, process variables and variables coming from the discretization can be optimized sequentially or simultaneously.

All methods allow to reconcile the data but the simultaneous moving horizon algorithm based on orthogonal collocations gives the best reconciled values. Technically, Kalman filter and the explicit integration method can not be recommended in the case of our study. Indeed, filtering methods do not allow to estimate input variables, so that it is not possible to develop a method to estimate a posteriori variances taking precisions of input variables into account. Moreover, the extended Kalman filter is not able to solve properly strongly non linear problems. The explicit integration method can hardly be extended by an algorithm to estimate a posteriori variances. On the other hand methods used in steady-state for a posteriori variances computing can be transposed to the integration method based on orthogonal collocations.
Orthogonal collocation-based methods gives good results for all the studied examples: the profiles of reconciled values follow the simulated values, a posteriori variances are much smaller than a priori ones and are better inside the reconciliation window than at its extremities, and errors are distinctly reduced. There remains nevertheless a problem with input variables whose reconciled values follow more the noise than reconciled values of differential state and algebraic variables.
From the discretized differential equations and the Lagrangian formulation of the dynamic data reconciliation problem, the sensitivity matrix of the global system of equations (process equations and equations coming from the discretization) can be estimated. If that matrix is non singular, all process variables can be computed for all measurement times with their a posteriori standard deviations. So, the non singularity of the sensitivity matrix appears to be an observability criterion that can be verified after reconciliation. Nevertheless, that a posteriori observability criterion is not enough to ensure data reconciliation with orthogonal collocations. Indeed, initial conditions for input and differential variables have to be known to start the data reconciliation.

The sensor network design algorithm suggested here for the dynamic problems goes further than the one described by Benqlilou based on the state estimation by means of Kalman filter. Indeed, it allows to deal with energy balances and measured input variables. So, the precision of the input variables has an influence on a posteriori standard deviation of differential state and algebraic variables. The choice of sensors that will measure input variables is not innocent. The choice of frequency of the measurement has been introduced in the algorithm to take into account the temporal redundancy. The method gives satisfying results for all studied examples.

A way to improve the convergence of the successive quadratic programming in the case of the dynamic data reconciliation should be to deal with semi-analytical derivatives instead of numerical derivatives. So, one could compute the analytical derivative of the part of the objective function that is common for all processes and use numerical differentiation for the part related to constraints $\mathbf{A}$, $\mathbf{A}^c$, $\mathbf{D}$ which are different for each process. With such a method, one can also try the interior point optimization algorithm developed by Cervantes

et al. Indeed, this method is sensitive to numerical noise, and is known not to be able to give good results with numerical derivatives.
Another improvement that should be done in the dynamic data reconciliation algorithm is the modification of input variables representation. Indeed, in the proposed version, input variables are linearized inside the interpolation interval. Another approach could be the use of orthogonal collocations. In that case, one has to choose if the discretization intervals are the same for input and differential state variables or not. When input variables are simulated as constant for the whole discretization or interpolation interval, the best approach should be to set them constant on the interval instead of trying to model the measurements by a polynomial. So, a mixed version allowing to chose between those three representations in function of the context would probably be the best approach concerning input variables.

Publications

C.Gerkens, C.Ullrich, and G.Heyen (2009). Use of a moving horizon dynamic data validation method based on orthogonal collocations for the design of sensor networks. *Submitted to Computers and Chemical Engineering.*

Gerkens, C. (2002). Conception rationnelle de systèmes de mesure dans les procédés chimiques. Master's thesis, University of Liège, Applied Science Faculty, Laboratoire d'analyse et de synthèse des systèmes chimiques.

Gerkens, C. (2003). Synthèse optimale de réseaux de capteurs : résolution au moyen de calculateurs parallèles. d.e.a.'s thesis, university of liège, applied science faculty, laboratoire d'analyse et de synthèse des systèmes chimiques.

Gerkens, C. and Heyen, G. (2004a). Sensor network design using genetic algorithm. *Proceedings of MMM IFAC Symposium.*

Gerkens, C. and Heyen, G. (2004b). Use of parallel computers in rational design of redundant sensor networks. *Proceedings of Escape 14 Congress.*

Gerkens, C. and Heyen, G. (2005). Use of parallel computers in rational design of redundant sensor networks. *Computers and Chemical Engineering*, 29(6):1379–1387.

Gerkens, C. and Heyen, G. (2008). Sensor placement for fault detection and localisation. *Proceeding of Escape 18 Congress.*

Gerkens, C., Ullrich, C., Mateus, M., and G.Heyen (2006). Comparaison de techniques de validation dynamique de données. *Proceedings of SIMO 2006 Congress.*

Heyen, G. and Gerkens, C. (2002). Application d'algorithmes génétiques à la synthèse de systèmes de mesure redondants. *Proceedings of SIMO 2002 Congress.*

Ullrich, C., Heyen, G., and Gerkens, C. (2009). Variance of estimates in dynamic data reconciliation. *Proceeding of Escape 19 Congress.*

Bibliography

Abu-el zeet, Z. H., Becerra, V. M., and Roberts, P. D. (2002). Combined bias ansd outlier identification in dynamic data reconciliation. *Computers chem. Engng.*, 26:921–935.

Albuquerque, J. and Biegler, L. (1995). Decomposition algorithms for on-line estimation with nonlinear models. *Computers and Chemical Engineering*, 19(10):1031–1039.

Albuquerque, J. and Biegler, L. (1996). Decomposition algorithms for on-line estimation with nonlinear dae models. *Computers and Chemical Engineering*, 21(13):283–299.

Ali, Y. and Narasimhan, S. (1993). Sensor network design for maximizing reliaility of linear processes. *AICHE Journal*, 39(5):820–828.

Ali, Y. and Narasimhan, S. (1995). Redundant sensor network for linear processes. *AIChe Journal*, 41:2237–2249.

Almasy, G. (1975). Checking and correction of measurements on the basis of linear system model. *Porblem of Control and Information Theory*, 4:57.

Amand, T. (1999). Application de la validation en ligne à la détection et la localisation de pannes. Master's thesis, University of Liège, Applied Science Faculty, Laboratoire d'analyse et de synthèse des systèmes chimiques.

Amand, T., Heyen, G., and Kalitventzeff, B. (2001). Plant monitoring and fault detection: synergy between data reconciliation and principal component analysis. *Computers and Chemical Engineering*, 25(4-6):501–507.

Bagajewicz, M. (1997). *Process plant instrumentation : design and upgrade (Chapter 6).* Technomic Publishing Company, Lancaster, Pensiylvania.

Bagajewicz, M. and Jiang, Q. (1997). Integral approach to plant linear dynamic reconciliation. *AICHE Journal*, 43(10):2546–2558.

Bagajewicz, M. and Jiang, Q. (1998). Gross error modelling and detection in plant linear dynamic recooonciliation. *Computers and Chemical Engineering*, 22(12):1789–1809.

Bagajewicz, M. and Sanchez, M. (1999a). Design and upgrade of non redundant and redundant linear sensor networks. *AICHE Journal*, 45(9):1927–1938.

Bagajewicz, M. and Sanchez, M. (1999b). Duality of sensor network design models for parameter estimation. *AICHE Journal*, 45(3):661–664.

Bai, S., Thibault, J., and McLean, D. (2006). Dynamic data reconciliation: an alterntive to kalman filter. *Journal of Process Control*, 16:485–498.

Barbosa, V., Wolf, M., and Fo, R. (2000). Development of data reconciliation for dynamic nonlinear system: application the polymerisation reactor. *Computers and Chemical Engineering*, 24:501–506.

Belsim (2004). *Vali4 User's Guide*. B-4470 Saint-Gerges-sur-Meuse, Belgium.

Benqlilou, C. (2004). *Data reconciliation as a framework for chemical processes optimization and control*. PhD thesis, Universitat Politècnica de Catalunya.

Benqlilou, C., Musulin, E., Bagajewicz, M., and Puigjaner, L. (2003). Intrumentation design and upgrade for optimal kalman filtering. *Computer Aided Chemical Engineering*, 14:371–376.

Benqlilou, C., Musulin, E., Bagajewicz, M., and Puigjaner, L. (2005). Instrumentation design and upgrade for optimal kalman filtering. *Journal of process control*, 15(6):629–638.

Bhagwat, A., Srinivasan, R., and Krishnaswamy, P. (2003a). Fault detection during process transitions. *Chemical Engineering Science*, 58(2):309–325.

Bhagwat, A., Srinivasan, R., and Krishnaswamy, P. (2003b). Multi-linear model-based fault detection during process transitions. *Chemical Engineering Science*, 58(9).

Bhushan, M., Narasimhan, S., and Rengaswamy, . R. (2008). Robust sensor network design for fault diagnosis. *Computers and Chemical Engineering*, 32:1067–1084.

Biegler, L. (1984). Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Computers and Chemical Engineering*, 8(3/4):243–248.

Biegler, L. (2007). An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing*, 46(11):1043–1053.

Biegler, L., Cervantes, A., and Wächter, A. (2002). Advances in simultaneous strategies ofr dynamic process optimization. *Chemical Engineering Science*, 57:575–593.

Binder, T., Blank, L., Dahmen, W., and Marquardt, W. (1998). *Nonlinear model based process control*, chapter Towards multiscale dynamic data reconciliation, pages 623–665. NATO ASI Series. Kluwer Academic Publishers.

Binder, T., Blank, L., Dahmen, W., and Marquardt, W. (2002). On the regularization of dynamic data reconciliation problems. *Journal of Process Control*, 12:557–567.

Broyden, C. (1965). A class of methods for solving non linear simultaneous equations. *Mathematics of Compututations*, 19:577–593.

Bullnheimer, B., Kotsis, G., and Strau, C. (1997). Parallelization strategies for the ant system. http://www.ani.univie.ac.at/ gabi/papers/kluwer.ps.gz.

Carnero, M., Hernandez, J., Sanchez, M., and Bandoni, A. (2005). On the solution of the instrumentation selection problem. *Industrial and Engineering Chemical Research*, 44:358–367.

Carroll, D. (2001). Fortran genetic algorithm driver version 1.7. cuaerospace.com/carroll/ga.html, consulted on May 2004.

Cervantes, A., Wächter, A., Tütüncü, R., and Biegler, L. (2000). A reduce space interior point strategy for optimization of differential algebraic systems. *Computers and Chemical Engineering*, 24:39–51.

Chen, H. and Stadherr, M. (1981). A modification of powell'dogleg algorithm for solving systems of non-linear equations. *Computers and Chemical Engineering*, 5(3):143–150.

Chen, H. and Stadherr, M. (1984). On solving large sparse nonlinear equation systems. *Computers and Chemical Engineering*, 8(1):1–7.

Chen, J. and Romagnoli, J. (1998). A strategy for simultaneous dynamic data reconciliation and outlier detection. *Computers and Chemical Engineering*, 22(4/5):559–562.

Chen, T., Morris, J., and Martin, E. (2008). Dynamic data rectification using particle filters. *Computers and Chemical Engineering*, 32:451–462.

Colorni, A., Dorigo, M., Maffiolo, F., Maniezzo, V., Righini, G., and Trubian, M. (1996). Heuristics form nature for hard combinatorial optimization problems. *Int. Trans. Opl. Res.*, 3(1):1–21.

Crowe, M. (1989). Observalibilty and redundancy of process data for steady-state reconciliation. *Chemical Engineering Science*, 44:2909–2917.

Davidon, W. (1975). Optimally conditionned optimization algorithm without line search. *Math. Programming*, 9:1–30.

de Wouwer, A. V., Point, N., Porteman, S., and Remy, M. (2000). An approach to the selection of optimal sensor locations in distributed parameter systems. *Journal of Process Control*, 10:291–300.

Dorigo, M., Bonabeau, E., and Theraulaz, G. (2000). Ant algorithm and stimergy. *Future generation computers systems*, 16:851–871.

Feo, T. and Resende, M. (1995). Greedy randomized adaptive search procedure. *Journal of Global Optimization*.

Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer, Norwell, MA.

Goldberg, D. (1989). *Genetic algorithms in search, optimisation and machine learning*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Gropp, W. (2005). *MPICH2 User's Guide*. Mathematics and Computer Science Division Argonne National Laboratory, version 1.0.3 edition.

Gropp, W., Lust, E., and Skjellum, A. (1999a). *Using MPI: portabe parallel programming with the message passing interface*. The MIT Press, Cambridge, London, second edition.

Gropp, W., Lust, E., and Thakur, R. (1999b). *Using MPI-2: advanced features of the message-passing interface*. The MIT Press, second edition.

Gutjahr, W. (2000). A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16:873–888.

Harwell (1990). *Harwell subroutine library*. Oxfordshire.

Haseltine, E. and Rawlings, J. (2005). Critical evaluation of extended kalman filtering and moving-horizon estimation. *Industry and Engineering Chemical Research*, 44(8):2451–2460.

Herrera, F., Lozano, M., and Moraga, C. (1999). Hierarchical distributed genetic algorithms. *International Journal of Intelligent Systems*, 14(11):1099–1121.

Heyen, G., Dumont, M., and Kalitventzeff, B. (2002). Computer-aided design of redundant sensor networks. *Proceedings of Escape 12*.

Heyen, G. and Gerkens, C. (2002). Application d'algorithmes génétiques à la synthèse de systèmes de mesure redondants. *Proceedings of SIMO 2002 Congress*.

Heyen, G., Maréchal, E., and Kalitventzeff, B. (1996). Sensitivity calculations and variance analysis in process plant measurement reconciliation. *Computers and Chemican Engineering*, 20S:530–544.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Jang, S.-S., Joseph, B., and Mukai, H. (1986). Comparison of two approaches to on-line parameter and state estimation of nonlinear systems. *Ind. Eng. Chem. Process Des. Dev.*, 25:809–814.

Jia, F., Martin, E., and Morris, A. (1998). Non-linear principal components analysis for process fault detection. *Computers and Chemical Engineering*, 22(Suppl):S851–S854.

Joris, P. and Kalitventzeff, B. (1987). Process measurements analysis and validation. *Proceedings of Chemical Engineering Conference: use of computers in chemical engineering*, pages 41–46.

Kabouris, J. and Georgakavos, A. (1996). Parameter and state estimation of the activated sludge process: on-line algorithm. *Wat. Res.*, 30(12):3115–3129.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME-Jornal of Basic Engineering*, 82:35–45.

Kalman, R. and Bucy, R. (1961). New results in linear filtering and prediction problems. *Transactions of the ASME-Jornal of Basic Engineering*, 83:95–107.

Kameswaran, S. and Biegler, L. (2006). Simultaneous dynamic optimization strategies: recent advances and challenges. *Computers and Chemical Engineering*, 30:1560–1575.

Karjala, T. and Himmelblau, D. (1996). Dynamic rectification of data via recurrent neural nets and the extended kalman filter. *AICHE Journal*, 42(8):2225–2239.

Kim, I.-W., Liebman, M., and Edgar, T. (1990). Robust error-in-variables estimation using nonlinear programming techniques. *AICHE Journal*, 36(7):985–993.

Kim, I.-W., Liebman, M., and Edgar, T. (1991). A sequential error-in-variables method for nonlinear dynamic systems. *Computers and Chemical Engineering*, 15(9):663–670.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Kong, M., Chen, B., He, X., and Hu, S. (2004). Gross error identification for dynamic system. *Computers and Chemical Engineering*, 29(1):191–197.

Kong, M., Chen, B., and Li, B. (2000). An integral approach to dynamic data rectification. *Computers and Chemical Engineering*, 24:749–753.

Kretsovalis, A. and Mah, R. (1987). Effect of redundancy on estimation accuracy in process data reconciliation. *Chemical Engineering Science*, 42(9):2115–2121.

Kuehn, D. and Davidson, H. (1961). Computer control ii: mathematics of control. *Chemical Engineering Progress*, 16(10/11):1653–1672.

Kyriakopoulou, D. (1997). *Development and implementation of an interior point optimization algorithm for process engineering*. PhD thesis, Université de Liège.

Lang, Y.-D. and Biegler, L. (2007). A software environment for simultaneous dynamic optimization. *Computers and Chemical Engineering*, 31:931–942.

Liebman, M., Edgar, T., and Lasdon, L. (1992). Efficient data reconciliation and estimation for dynamic processes using nonlinear programing techniques. *Computers and Chemical Engineering*, 16(10/11):963–986.

Madron, F. (1992). *Process plant performance, measurement and data processing for optimization and retrofit (section 6.3)*. Ellis Horwood, New-York.

Mah, R., Stanley, G., and Downing, D. (1976). Reconciliation and rectification of process flow and inventory data. *Ind. Eng. Chem., Process Des. Dev.*, 15:175–183.

Mah, R. and Tamhane, A. (1982). Detection of gross error in process data. *AICHE Journal*, 28(5):828–830.

McBrayer, K. and Edgar, T. (1995). Bias detection and estimation in dynamic data reconciliation. *Journal of Process Control*, 5(4):285–289.

McBrayer, K., Soderstrom, T., Edgar, T., and Young, R. (1998). The application of non linear dynamic data reconciliation to plant data. *Computers and Chemical Engineering*, 22(12):1907–1911.

Moraal, P. and Grizzle, J. (1995). Observer design for nonlinear systems with discrete-time measurements. *IEEE Transaction on Automatic Control*, 40(3):395–404.

Muradore, R., Bezzo, F., and Barolo, M. (2006). Optimal sensor location for distributed-sensor systems using multivariate regression. *Computers and Chemical Engineering*, 30:521–534.

Narasimhan, S. and Jordache, C. (2000). *Data reconciliation and gross error detection: an intelligent use of process data*. Gulf Publishing Company, Houston, Texas.

Narasimhan, S. and Mah, R. (1987). Generalized likelihood ratio method for gross errors identification. *AICHE Journal*, 33(9):1514–1521.

Narasimhan, S. and Mah, R. (1988). Generalized likelihood ratios for gross errors identification in dynamic processes. *AICHE Journal*, 34(8):1321–1331.

Norgaard, M., Poulsen, N., and Ravn, O. (2000). New developpements in state estimation for nonlinear systems. *Automatica*, 36:1627–1638.

Raff, T., Ebenbauer, C., Findeisen, R., and Allgöer, F. (2005). *Control and observer design for non linear finite and infinite dimensional systems*, chapter Remarks on moving horizon state estimation with guaranteed convergence, pages 67–80. Springer.

Ragot, J. and Maquin, D. (2006). Fault measurement detection in an urban water supply network. *Journal of Process Control*, 16(9):887–902.

Ragot, J., Maquin, D., and Dibo, M. A. (2003). Sensor fault detection and isolation. a blind approach. *Proceedings of the Fifth international symposium on intelligent components and instruments for control.*

Rao, V. (2000). *Moving horizon strategies for the constrained monitoring and control of nonlinear discrete-time systems.* PhD thesis, University of Wisconsin-Madison.

Rechenberg, I. (1971). *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Technical University of Berlin. reprinted by Fromman-Holzboog (1973).

Reilly, D. and Carpani, R. (1963). Application of statistical theory of adjustment of material balances. *Proceedings of the 13th Canadian Chemical Engineering Congress.*

Renfro, J., Morshedi, A., and Asbjornsen, O. (1987). Simultaneous optimization and solution of systems described by differential/algebraic equations. *Computers and Chemical Engineering*, 11(5):503–517.

Romagnoli, J. and Stephanopoulos, G. (1981). Rectification of process measurement data in the presence of gross error. *Chemical Engineering Science*, 16(11):1849–1863.

Rousseaux-Peigneux, P. (1988). *Filtre de Kalman hierarchisé pour l'estimation d'état dynamique de grands systèmes électriques.* PhD thesis, Université de Liège.

Schmidt, S. (1980). The kalman filter: its recognition and development for aerospace applications. *Journal of Guidance and Control*, 4(1):4–10.

Schwefel, H.-P. (1974). *Numerische Optimierung von Computer-Modellen.* PhD thesis, University of Berlin. reprinted by Birkhäuser (1977).

Sen, S., Narasimhan, S., and Deb, K. (1998). Sensor network design of linear processes using genetic algorithms. *Computers and Chemical Engineering*, 22(3):385–390.

Shelokar, P., Jayaraman, V., and Kulkarni, B. (2004). An ant colony approach for clustering. *Analytica Chimica Acta*, 509:187–195.

Singh, A. and Hahn, J. (2005). Determining optimal sensor locations for state and parameter estimation for stable nonlinear systems. *Industry and Engineering Chemical Research*, pages 5645–5659.

Talbi, E., Roux, O., Fonlupt, C., and Robillard, D. (2001). Parallel ant colonies for the quadratic assignement problem. *Future Generation Computer Systems*, 17:441–449.

Vachhani, P., Narasimhan, S., and Rengaswamy, R. (2006). Robust and reliable estimation via unscented recursive nonlinear dynamic data reconciliation. *Journal of Process Control*, 16(10):1075–1086.

Vachhani, P., Rengaswamy, R., and Venkatasubramanian, V. (2001). A framework for integrating diagnostic knowledge with non linear optimisation for data reconciliation and parameter estimayion in dynamic systems. *Chemical Engineering Science*, 56:2133–2148.

Vaclavek, V. (1968). Studies on system engineering: on the application of the calculus of observations in calculuations of chemical engineering balances. *Coll. Czech. Chem. Commun.*, 34:3653–3660.

Vaclavek, V. (1969). Studies on system engineering: optimal choice of the balance measurements in complicated chemical engineering systems. *Chemical Engineering Science*, 24:947–955.

Verny, V. (1985). A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51.

Villadsen, J. and Michelsen, M. (1978). *Solution of differential equation models by polynomial approximation*. Prentice-Hall, Englewood Cliffs, New Jersey.

Villadsen, J. and Stewart, W. (1967). Solution of boundary-value problems by orthogonal collocation. *Chemical Engineering Science*, 22:1483–1501.

Wailly, O. and Héraud, N. (2005). Cost-optimal design of reliable sensor networks extended to multi-linear system. *Computers and Chemical Engineering*, 29:1083–1087.

Walfraff, W., Dochain, D., Bourrel, S., and Magnus, A. (1998). On the use of observability measures for sensor location in tubular reactor. *Journal of Process Control*, 8(5-6):497–505.

Wally, O., Héraud, N., and Malasse, O. (2008). Design of instrumentation in process plants using groëbner bases. *Computers and Chemical Engineering*, 32:2179–2188.

Wang, F., Jia, X., Zheng, S., and Yue, J. (2004). An improved mt-nt method for gross error detection and data reconciliation. *Computers and Chemical Engineering*, 28:2189–2192.

Wang, H., Song, Z., and Wang, H. (2002). Statistical process monitoring using improved pca with optimized sensor locations. *Journal of Process Control*, 12(6):735–744.

Welch, G. and Bishop, G. (2001). *Annual conference on computer graphics and interactive techniques*, chapter An introduction to Kalman Filter. ACM Press, Addison-Wesley, Los Angeles, Californie, USA, 2001 edition.

Wongrat, W., Srinophakun, T., and Srinophakun, P. (2005). Modified genetic algorithm for nonlinear data reconciliation. *Computers and Chemical Engineering*, 29(5):1059–1067.

Appendix A

Sensor network optimization method

A.1 Introduction

Many optimization problems dealing with engineering computer science, management, administration... can not be solved satisfactory or practically by a problem-specific algorithm. Metaheuristic methods have been created to solve such problems. They consist of heuristic methods for solving very large classes of computational problems by combining several black-box procedures given by the user in an efficient way. Those procedures are generally also heuristic methods. Metaheuristics are commonly used to solve combinatorial optimization problems or problems that can be translated into that form. Many methods of this class model processes that take place in nature.
The goal of combinatorial optimization is to find a discrete mathematical object that minimizes or maximizes an arbitrary objective function given by the user. Those objects are called *states* and the set of all candidate states, the *search space*. The nature of the states and the search space depends on the optimization problem. The objective function to be optimized is called the *goal function* and is given by the user in the form of a black-box procedure that evaluates the function of the studied state. More besides the objective function, it may be asked to the user to provide other black-box procedures that produce a new random state or several variants of a given state, choose one state between several, give upper and lower bounds for the objective function...
Metaheuristics can be classified between *iterative* and *constructive* heuristics:

- Iterative heuristics start with a complete feasible solution and change this current state thanks to an iterative process in order to improve the value of the goal function. A typical case of iterative heuristics is *local search*. In that method, the new state is derived from the current solution, which is always complete and feasible, by means of a *mutator* procedure. The set of new states produced by the mutator is called the *neighborhood* of the current state.

- In constructive heuristics the solution is generated "from scratch" by successive additions of certains elements or components, with or without backtracking (removal of the elements that have been added at an earlier step). A typical case of constructive heuristics is the *greedy heuristics*. In that method, the final solution is successively built up in a linear process without backtracking, governed by the gains of the components that are allowed to be added at a certain step.

An other classification can be made between *single-run* and *repetitive* heuristics:

- In the case of single-run heuristics, the algorithm stops as soon as a certain internal condition is satisfied. For example, this condition is reaching a local optimum in the case of local search and having finished the construction procedure in the case of greedy heuristics.

- In repetitive heuristics the user may control the amount of computation time he wants to invest so that the quality of the final solution is a function of the invested computation time. Genetic algorithms, simulated annealing, greedy randomized adaptive search procedure are representatives of this class.

Repetitive heuristics may reach solutions of very high quality but at a price of very large computational time, whereas single-run heuristics are generally faster but often provide solutions of moderate quality.

In this chapter, genetic algorithms are described. That algorithm has been chosen because the sensor network design problem is generally multimodal and involves many binary decisions. Moreover, as it will be shown in the examples of chapter 5, the computing time increases rapidly with the size of the studied process so that one had to chose an algorithm that could be parallelized.

A.2 What are evolutionary algorithms ?

Evolutionary algorithms are population-based metaheuristic optimization algorithms that use mechanisms inspired from Darwin's evolution theory. Their basic principle is to mimic the behavior of populations of living beings, which adapt to their surroundings thanks to phenomena like natural selection and genotype. The artificial version of Darwin's theory is extremely simplified and is also called artificial Darwinism.
The best known evolutionary algorithms are genetic algorithms, which will be described in the next section, but evolutionary strategies, evolutionary programming, genetic programming and learning systems classifiers are also largely used (they are briefly described in this section). All those algorithms have in common the fact that they manipulate an artificial population whose evolution is simulated thanks to two types of random operations:

- a selection of individuals that will be allowed to generate children based on the performance of the individuals, namely their more or less good correspondance with what is searched;

- genetic operators, generally cross-over and mutation, that produce new individuals in the population.

Those operations are repeated in loop, often in the form of generations, until the population converges.
The success of such methods comes from the fact that they represent optimization tools adapted to difficult, complex and irregular functions and problems. Nevertheless this evolutionary search has a computing cost that can be important; indeed this is an iterative method that proceeds by random directed trial and error. Those techniques are complementary to more standard optimization methods like deterministic optimization methods that often make regularly hypothesis on the functions to be optimized.
Evolutionary algorithms have been successfully applied in the fields of engineering, art, biology, economics, genetics, operations research, robotics, social sciences, physics, chemistry...
Evolutionary algorithms have also been used as an experimental framework to validate theories about biological evolution and natural selection, particularly in the field of artificial life. Techniques coming from evolutionary algorithms used to model biological evolution are generally limited to the modeling of microevolutionary processes. However some macroevolutionary dynamics have been modeled.
A limitation of evolutionary algorithms is that they do not clearly distinguish genotype and phenotype. For example, in nature, the fertilized egg cell undergoes a complex process called embryogenesis to become a mature phenotype. This indirect encoding is believed to make the genetic search more robust, namely it reduces the probability of fatal mutations, and may improve the evolvability of the organism. Some recent works try to take this phenomenon into account.

A.2.1 Evolution strategy

The evolution strategy was developed by I. Rechenberg (Rechenberg, 1971) and H.P. Schwefel (Schwefel, 1974) of the Technical University of Berlin. This evolutionary algorithm is based on the ideas of adaptation and evolution.
In its first form, this method was not population based: indeed, child created from a single individual, which contains a vector of design parameters, mutates by adding a random vector that is normally distributed with a mean of zero and a variance identical in all dimensions. The best individual between the parent and the child is used as the next individual. This strategy is called a $(1 + 1)$ evolution strategy, namely the current individual generate a child and the best individual between those two individuals generates the next individual. In this first version, no cross-over occurs between individuals. A rule for changing the variance of mutations was determined from the convergence rate theory developed by I. Rechenberg (Rechenberg, 1971): it appears that 20% of the children should replace their parent. If there are more good children, the search space is too small the variance should be increased and inversely if less children are able to replace their parents.
A second form of evolution strategy is the $(\mu+1)$ evolution strategy proposed by I. Rechenberg (Rechenberg, 1971). In this method, a population of μ individuals is used to generate a new child thanks to cross-over and mutations. This new individual replace the worst one in the population if it is better.

Schwefel (Schwefel, 1974) extended this second form in two ways in which it is possible to give a particular mutation variance to each individual:

- the (μ+λ) evolution strategy: in this strategy, the new population of μ individuals is made of the μ best individuals from the μ parents and the λ children.
- the (μ,λ) evolution strategy: in this case, the μ best individuals for the new population are chosen between the λ children only, so that $\lambda > \mu$. The best individuals of the parent population are not kept from generation to generation so that the convergence can not be guaranteed. Despite this disadvantage, this second extension is generally preferred because with the population evolution, the information contained in older individuals genes become quickly out of date so that those individuals can be removed without important loss of information.

There are several combination operators that can be used in evolution strategies. They are classified into two families:

- the sexual combination: the parameters are generated from two chosen parents;
- the panmitic combination: one fixed parent is chosen and for each parameter, another parent is chosen in the population for combination, so that the whole population contributes to each new individual.

Recently, two new parameters have been added to the (μ,λ) evolution strategy: κ the maximum lifespan of an individual and ρ the number of parents involved in the combination.

A.2.2 Evolutionary programming

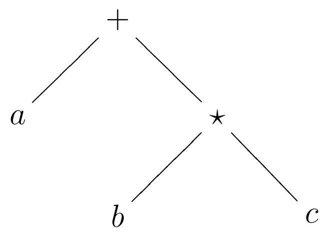
The evolutionary programming was developed by L.J. Fogel (Fogel et al., 1966) in the sixties in order to design state machines for predicting sequences and symbols. This method was criticized because the predicted sequences were relatively simple and penalty functions for mis-prediction were needed to obtain good results. In the eighties, the method was transformed to solve real-value optimization problems. This new version of the method is similar to evolution strategy but differs from it by two points:

- In evolutionary programming, there is no cross-over because in the sense of evolutionary programming, individuals represent entire species and not members of a single species.
- The selection of the new population is different: indeed in evolutionary strategy, the selection is deterministic, the best μ individuals are chosen for the next population, whereas in evolutionary programming, a stochastic method is used: the tournament selection. This selection mechanism is described in the next section with the genetic algorithm. Its advantage is that the set of potential solutions is more diverse. This means that the population will grow less frequently around a local minimum and there

is less likelihood that the algorithm converges prematurely to a local minimum. Its disadvantage is that sometimes the best individuals are lost making the convergence of the algorithm more time consuming.

A.2.3 Genetic programming

Genetic programming is a machine learning technique that uses a genetic algorithm to optimize a population of computers programs according to a fitness landscape determined by the ability of a program to perform a given computational task. The individuals are not anymore fixed-length character string as in genetic algorithm but they are programs. Those programs are expressed in parse trees rather than in lines of code. As an example, the program "$a + b \star c$" is represented by the following tree:



In genetic programming, the cross-over consists of taking randomly selected subtrees selected according to their fitness in the individuals and exchanging them.
Genetic programming is computationally intensive and was mainly used to solve simple problems in the nineties. Improvements in the genetic programming and exponential growth in CPU power in the last years allowed to obtain outstanding results in areas such as quantum computing, electronic design, game playing, sorting... In the nineties, genetic programming was considered as a sort of pariah amongst of the various techniques of search. In the 2000s, genetic programming developed rapidly so that it has been possible to build exact probabilistic models such as Markov chains or schema theories. Now, genetic programming is more general than genetic algorithms: it includes them.

A.2.4 Learning classifier systems

Learning classifier systems appeared at the beginning of the seventies and were described by John Holland (Holland, 1975). They are machine learning systems with close links to genetic algorithms and reinforcement leaning. A learning classifier system consists of a population of binary rules on which a genetic algorithm alters and selects the best rules. Rule utility is decided by a reinforcement learning technique instead of using a fitness function. Learning classifier systems are classified in two families depending upon where the genetic algorithm acts:

- the Pittsburgh-type: in those learning classifier systems, the population consists of separate rule sets and the genetic algorithm recombines and reproduces the best of these rule sets;

- the Michigan-type: in that type of learning classifier systems, there is only a single population and the genetic algorithm focuses on selecting the best classifiers within that rule set. They have two main types of reinforcement learning: fitness sharing and accuracy-based.

At the beginning, learning classifier systems used binary rules, but thanks to the recent research, they can now use populations of neural networks and other methods.
Learning classifier systems are not well defined in a mathematical point of view and remain an area of research even if they have been applied successfully to many different types of problems.

A.3 Genetic algorithms

Genetic algorithms are random research algorithms based on natural selection and genetic mechanisms. They were created by John Holland (Holland, 1975), his students and colleagues at the beginning of the sixties in the University of Michigan. His objectives were the abstraction and the rigorous explanation of the processes of natural systems adaptation as well as the design of artificial systems from a software using the same mechanisms than natural systems.
Genetic algorithms combine the survival of the best individuals of the parent population with a structured and random exchange of information between individuals. At each generation, a set of new individuals (children) is created using parts of the best parents. Genetic algorithm exploit efficiently information from the previous population to search for children with better goal function or *fitness*.
Robustness is one the the main advantages of genetic algorithms. Indeed, robustness implications for artificial systems are numerous: if artificial systems can be made more robust, the most expensive solutions can be eliminated or reduced; if higher level of adaptation can be achieved, systems can carry out their function during a longer time and more efficiently. Biological systems are robust, efficient and flexible. Laws governing those systems are selection, reproduction, cross-over and mutation. They appear only in a rough way in artificial systems so that nature always remains more efficient in the robustness point of view. Moreover, genetic algorithms have given proofs of their robustness as well in the mathematical point of view than in the empirical point of view.
Genetic algorithms differ from traditional optimization methods. Indeed:

- Genetic algorithms code the parameter set instead of the parameters themselves;
- The search carried out by genetic algorithms are made from a population of points instead of a single-point;
- Genetic algorithms use information coming from a goal function instead of derivatives or other auxiliary knowledge;
- Genetic algorithms use probabilistic transition rules instead of deterministic rules.

Applications of genetic algorithms are numerous and from very different domains: optimization of difficult numerical function (discontinuous, multimodal, with noise...), pictures processing, timetable optimization, design optimization, industrial systems control, learning of neural networks... Genetic algorithms can be used to control time involving systems (production lines, nuclear power plants) because the population can adapt to condition changes. They can also be used to find the parameter of a model from the experimental measurements, to optimize networks for water or gas distribution. They can be integrated in electronic chips so that they would be able to reconfigure themselves in function of their surroundings.

A.3.1 Individuals coding

Genetic algorithms use a population of points also called individuals or chromosomes. Each chromosome is made of a set of parameters that are translated in a binary form. Each coded parameter is named gene. There are three types of genes:

- boolean genes made of a single binary character. They translate parameters that have only two possible values, they translate binary decisions. In the case of sensor network design, they translate the decision to put or not a sensor at a specified location.
- integer genes made of several binary characters. They translate integer parameters;
- real genes made of several binary characters. They translate real parameters.

In our implementation of the algorithm, when the first individual is created, a gene is added for each parameter of the problem. In addition to the gene, different characteristic of the gene are also coded:

- the number of the gene in the chromosome;
- the name of the gene;
- the type of the gene;
- the length of the gene;

- the minimal value of the parameter represented by the gene;
- the position of the first binary character of the gene in the chromosome;
- the length of the chromosome after the insertion of the gene.

The values of the parameters of the first chromosomes are fixed arbitrarily while the values of the parameters of the other chromosomes of the first population are determined at random knowing that each parameter has a fixed probability to have the arbitrary values fixed for the first chromosome.

A.3.2 Natural mechanisms description

In this section, four useful genetic mechanisms are described: the selection, the reproduction, the cross over and the mutation. As in the case of sensor network design there are only binary choices (presence or absence of sensors in the plant), all those mechanisms are explained for binary decisions only, although those algorithms can be used for combinatorial problems for which the choice can be made between more than two solutions.

Selection

Individuals are selected in function of an objective function called *fitness*. Individuals with the best fitness have a higher probability to be chosen. This fitness operator is an artificial version of natural selection in which only the best individuals survive. In the nature, the fitness represents the capacity to survive to predators, diseases and all obstacles that prevent reaching the adult age and excluding reproduction. In the case of artificial systems, the fitness decides if individuals can live or if they have to die.
Two selection methods are described here after: the roulette wheel and the tournament selection.

6. **Tournament selection**
   In tournament selection mechanism, two individuals are chosen at random. The one with the best fitness is kept as the father individual. A mother individual is chosen in the same way. To make the choice of the pairs of individual random, all the individuals of the population are numbered at random before the first selection. They are then compared two by two following this new order: the first one with the second one to give the father, the third one with the fourth one to give the mother... Let N be the size of the population. Individuals are renumbered each time the number of chosen parents is $\frac{N}{2}$ if N is even and $\frac{N-1}{2}$ if N is odd. In this method, the individual with the lowest fitness, is never chosen and individuals with the best fitness have a great probability to be chosen.
   This technique can be used as well if the fitness has to be maximized as if it has to be minimized.
   This method has been chosen for the genetic algorithm used in this research.

7. **Roulette wheel**
   In roulette wheel selection method, the probability of selecting an individual in the population is proportional to the value of its fitness function. This technique is only used if the fitness has to be maximized.
   Let a population of five individuals composed of six binary genes (Figure A.1). Those individuals are represented on the wheel in function of the contribution of their fitness to the global fitness of the population. (Table A.1).

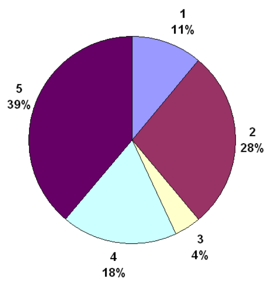


Figure A.1: Roulette wheel

Numbers	Individuals	Fitness	Proportions (%)
1	100110	143	11
2	101001	364	28
3	110010	52	4
4	101011	234	18
5	001011	507	39
Sums		1300	100

Table A.1: Roulette wheel: table of values

During selection mechanism, the roulette wheel rotates. When it stops, the individual situated in front of the wheel cursor is selected. Doing this way, individuals with the highest values of the fitness have a higher probability to be selected than those with the lowest values.

Reproduction

Reproduction is a mechanism during which a part of the parent individuals are copied to generate identical children. The parents that are copied are chosen in function of their fitness thanks to one of the selection mechanisms described before. In the algorithm that was used, the parents are selected thanks to a tournament and 50% of those selected individuals are copied.

Cross-over

Cross-over mechanism consists of combining the genotypes of a mother individual and a father individual in a way that the two children obtained genetic characteristics from both parents. Three cross-over methods are described here after: one-point cross-over, two-points cross-over and uniform crossover. Only the one-point cross-over is applied in the genetic algorithm used in this research.
Cross-over concern the parents pairs that have not given children by reproduction, that means the halve of the parent pairs.

8. One-point cross-over
   In the case of one-point cross-over, the crossing point is chosen randomly at the binary level. If the genes are made of more than one binary decision, the crossing point can be situated in the middle of a gene, whose binary characters are shared out between the two children. The distribution between the binary characters of the parents is made in the following way (figure A.2):

   - The first child is made with the binary characters of the father that are situated on the left of the crossing point and those from the mother that are on the right of that point;
   - The second child is made with the binary characters of the mother that are situated on the left of the crossing point and those from the father that are on the right of that point.

   Chromosomes are divided in 32 bits parts which are examined one after the other:

   - If the crossing point is on the left of the i^{th} part, the i^{th} part of the father is copied in the i^{th} place of the first child and the i^{th} part of the mother is copied in the i^{th} place of the second child;
   - If the crossing point is on the right of the i^{th} part, the i^{th} part of the father is copied in the i^{th} place of the second child and the i^{th} part of the mother is copied in the i^{th} place of the first child;
   - If the crossing point is inside the i^{th} part, the binary characters of the i^{th} part of the father that are on the left of the crossing point are copied in the left part of the i^{th} place of the first child and the ones situated on the right, in the right part of the i^{th} place of the second child. The binary characters of the i^{th} part of the mother that are on the left of the crossing point are copied in the left part of the i^{th} place of the second child and the ones situated on the right, in the right part of the i^{th} place of the first child.

9. Two points cross-over
   Two-points cross-over is similar to the one-point cross-over. It is generally said to be more efficient. The distribution between the binary characters of the parents is now made as can be seen on figure A.3:

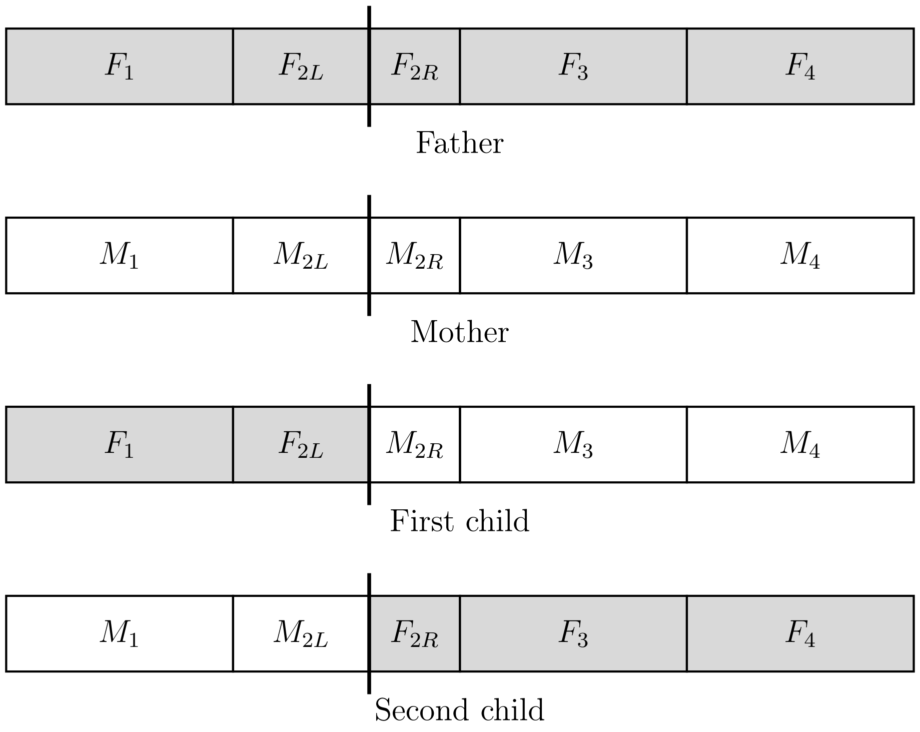


Figure A.2: One-point cross-over

10. Uniform cross-over
    In uniform cross-over, a random decision is taken for each binary character: either the i^{th} character of the father is copied at the i^{th} place of the first child and the i^{th} character of the mother is copied at the i^{th} place of the second child or the i^{th} character of the mother is copied at the i^{th} place of the first child and the i^{th} character of the father is copied at the i^{th} place of the second child (figure A.4).

Mutation

Mutations are alterations of the genetic material. In natural systems, three types of mutations take place: the substitution, the deletion and the insertion of one or several nucleotides in a gene. In genetic algorithms, mutations are occasional and random alterations of the value of one or several characters of a gene. It consists of the inversion of the value of one or several bits of a gene. Mutation plays the role of noise and prevent the evolution to be frozen. It allows to secure a research as well global than local since it can introduce in the population features that where not present in the previous generation. Furthermore, they guarantee mathematically that a global optimum can be reached after sometimes a very long computing time.
Mutations play another important role in genetic algorithm. Indeed, even if reproduction and cross-over are efficient research and recombination methods, they can drive to a genetic

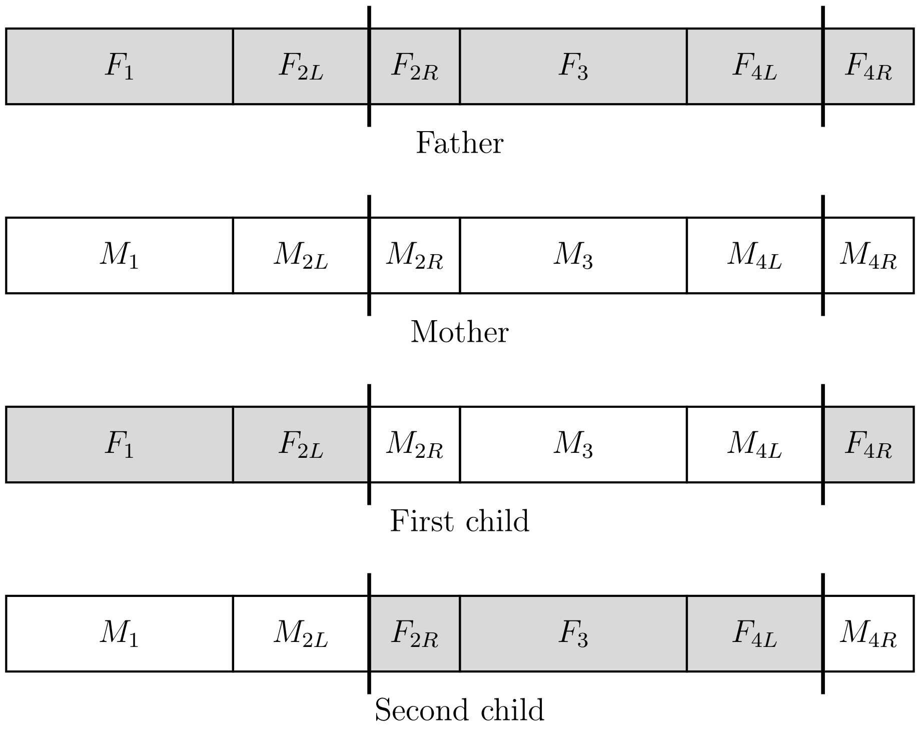


Figure A.3: Two-points cross-over

drift. Indeed, a too small population can get homogenized because of stochastic errors: genes that are favored by luck may spread in the population to the detriment of the others. Because of the genetic drift, the final solution may be not optimal. Mutations counterbalance this phenomenon by introducing regularly new genes in the population.
Empirical researches on genetic algorithms show that, to obtain good results, the frequency of mutation must be in the order of one mutation for thousand binary characters transferred to the children.

11. Jump mutation
    Jump mutation mechanism consists in the inversion of the value of one bit, chosen randomly in one gene of an individual. (figure A.5):

12. Creep mutation
    Creep mutation consists of the exchange of the values of two bits in a same gene. (figure A.6):

A.4 Others optimization methods

Other meta-heuristic algorithms could have been used to solve the sensor network design problem. One can cite ant colonies, simulated annealing, tabu search, greedy randomized

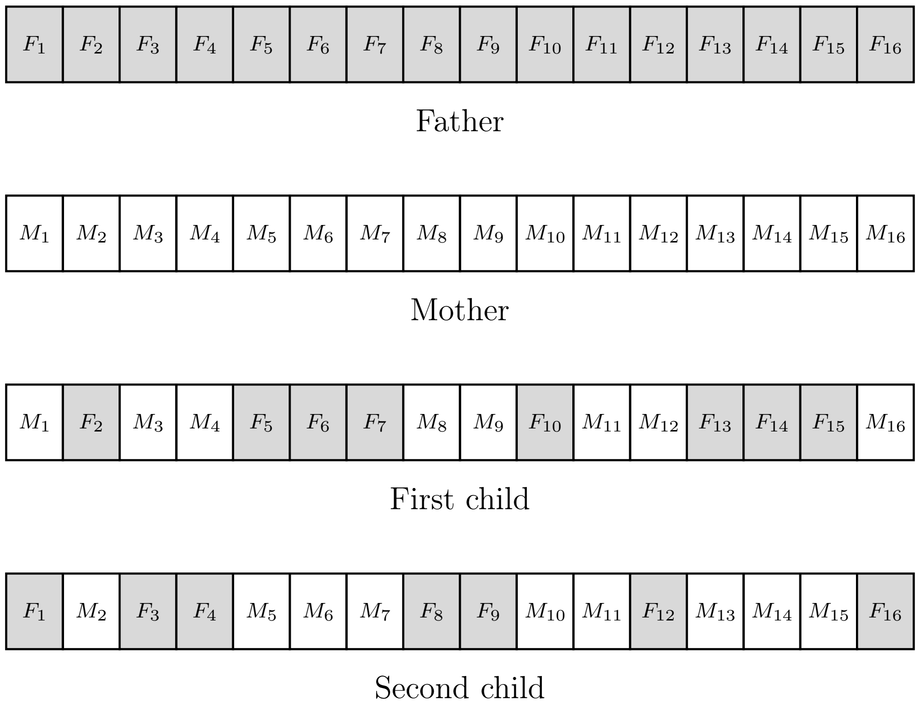

F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12} F_{13} F_{14} F_{15} F_{16}
Father
M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16}
Mother
M_1 F_2 M_3 M_4 F_5 F_6 F_7 M_8 M_9 F_{10} M_{11} M_{12} F_{13} F_{14} F_{15} M_{16}
First child
F_1 M_2 F_3 F_4 M_5 M_6 M_7 F_8 F_9 M_{10} M_{11} F_{12} M_{13} M_{14} M_{15} F_{16}
Second child

Figure A.4: Uniform cross-over

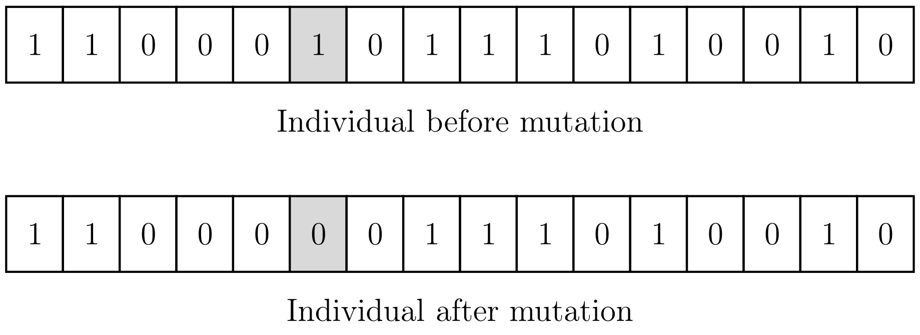


Figure A.5: Jump mutation

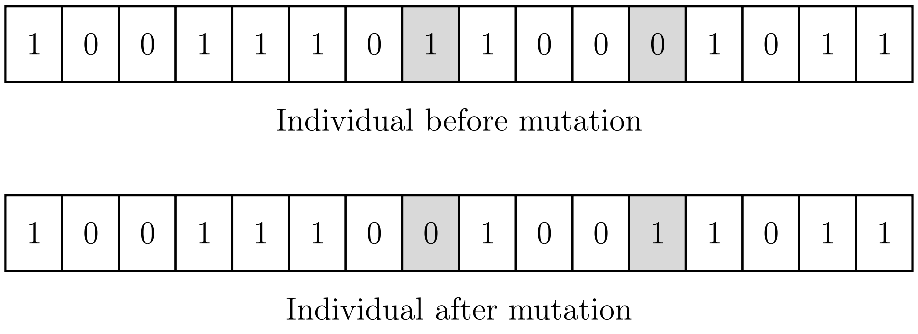


Figure A.6: Creep mutation

adaptive search procedures...
Those methods have not been used in this study. Indeed, as it will be illustrated in the case studies, the estimation of our goal function required the inversion of a large sparse matrix, which requires a rather long time. However simulated annealing is a sequential algorithm that can not be parallelized and it was estimated that it would not allow to reach the solution within a reasonable computing time for large size problems. The others algorithms were not tested because the results reached with genetic algorithm gave solutions that satisfied us.

Appendix B

Parallelization

B.1 Notions of parallelization

The development in sciences was traditionally based on a theoretical or on an experimental approach. Thanks to more and more powerful and fast computers, scientists can nowadays simulate problems that are too complex to be reliably predicted by theory, too dangerous or too expensive to be reproduced in the laboratory. The demand on supercomputers has strongly increased during the last years.

At the same time, parallel computers have evolved from theoretical concept to everyday tool to solve problems needing more and more computer ressources. Several factors have simulated this evolution: the speed of computers is limited by the speed of the light and the efficiency of heat dissipation, the cost of powerful computer increased more rapidly than their power, and the ratio between the cost and the performance is favorable if the computer ressources can be found instead of bought. This last factor has simulated laboratories to exploit existing computers in parallel and the development of clusters of computers. The rise in performance and capacity of wide-area networks has made it possible to write applications that span the world. The concept of a grid of computational ressources and connections is being explored. This concept is analogous to the electric power grid.

Thus, parallelization seems to be the best compromise between performance and the ratio between cost and performance. However there remains some obstacles to an universal use of parallelization coming from the material or from the software, but most of the time the main obstacle comes from an inadequate software environment. The ideal mechanism to communicate between a parallel algorithm and a parallel computer must be expressive, efficient and portable.

There exists several parallel computational models: data parallelism, shared memory, message passing, remote memory operations, threads and combined models.

For parallelizing the optimal sensor network design algorithm, a message passing model, the message passing interface, has been used. That kind of model consists of a set of processors that have only local memory and can communicate with other processors by sending and receiving messages. Data transfer from the local memory of one processor to the local memory of another one requires that both processors perform operations. This model is represented on figure B.1.

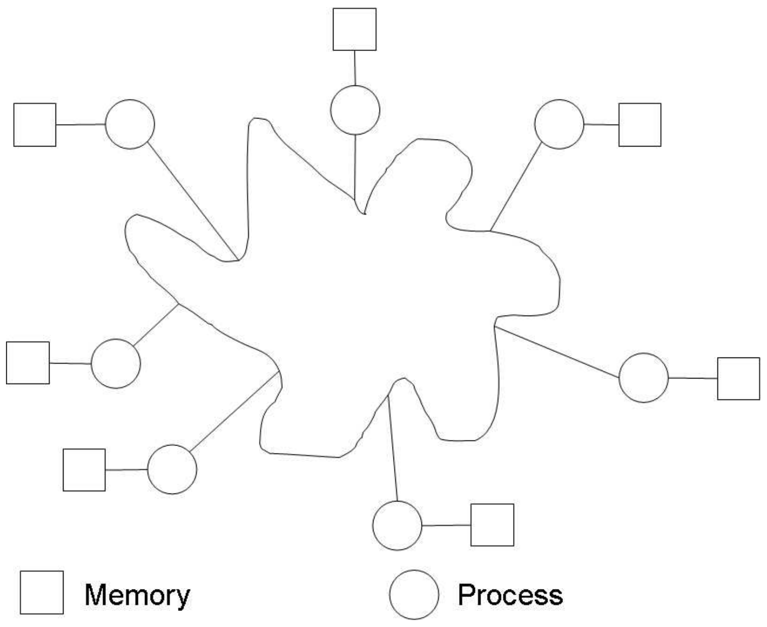


Figure B.1: Message passing model

MPI is a programming method on parallel computers (Gropp et al., 1999a),(Gropp et al., 1999b). It is a collection of the best features of many message passing systems that have been developed over the years. Those features have first been improved if necessary before being standardized. The main characteristics of MPI are the following ones: MPI is a library of routines, not a language, it is a specification and not a particular implementation and it addresses the message passing model. The advantages of MPI are the universality, the expressivity, the performance and the debugging relative easiness.
Mpi routines (Gropp, 2005) that have been used to parallelize the optimal sensor network design algorithm are of three types:

- The routines linked to the environment: the environment has to be initialized so that the different processors are created and a space of memory addresses is attributed to each of them. Inversely, another routine deactivates the environment. There exists also routines that allow to know the number of processors managed by the communicator or the rank of a given processor in a specified communicator.

- The point-to-point communications: those communications take place between two processors: the emitting processor and the receiving processor. Those two processors are identified by their rank in the communicator. Point-to point communications can be blocking: the emitting processor remains blocked until the receiving processor has received the message, or non-blocking: the emitting processor continues even if the receiving one has not received the message.

- The collective communications: those communications allow to carry out in a single operation a set of point-to-point communications. A collective communication concerns always all the processors of a given communicator. Three of them have been used in the parallelized version of the genetic algorithm:
  - the broadcast routine allows to give the same information to all processors of a given communicator (figure B.2);

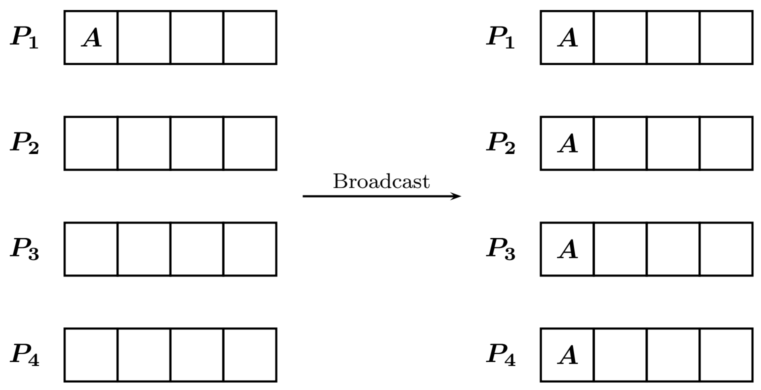


Figure B.2: Broadcast routine

  - the scatter routine allows to distribute a message between all the processors of the communicator. Each processor receives a message whose size is equal to the size of the original message divided by the number of processors in the communicator (figure B.3):

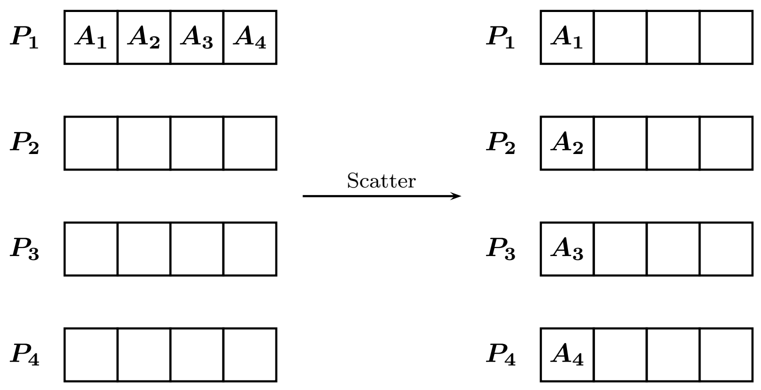


Figure B.3: Scatter routine

– the gather routine allows collecting data of all the processors of the communicator. The receiving processor receives from each emitting processor a message of the same size and of the same type of data (figure B.4).

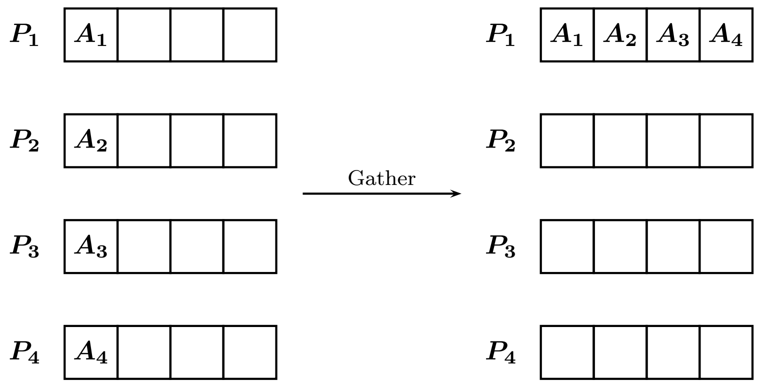


Figure B.4: Gather routine

The parallelization methods allow to reduce the computing time by sharing the computing work between several processors. However, this gain in solution time must be done without increasing the computing ressources too much. The *speed up* is the gain in time obtained when a sequential code is parallelized. If T_1 is the time required to solve the problem B on a sequential computer and T_p the time required to solve the same problem on a cluster of p processors, the speed up is given by the ratio:

$$\text{Speed up} = \frac{T_1}{T_p} \tag{B.1}$$

Parallelization techniques can be compared thanks to the efficiency, which is estimated this way:

$$\text{Efficiency} = \frac{\text{Speed up}}{p} = \frac{T_1}{T_p}\frac{1}{p} \tag{B.2}$$

The computing work is given by:

$$\text{Work} = p\ T_p \tag{B.3}$$

The speed up can be linear, under-linear or over-linear (figure B.5):

- The speed up is linear if the processors have an activity level equal to 100 %. The corresponding gain of time is equal to the number p of processors.
- The speed up is under-linear if the processors have an activity level lower than 100 %. The speed up can be under-optimal if some parts of the program are sequential or because of the time required for the communications between processors.

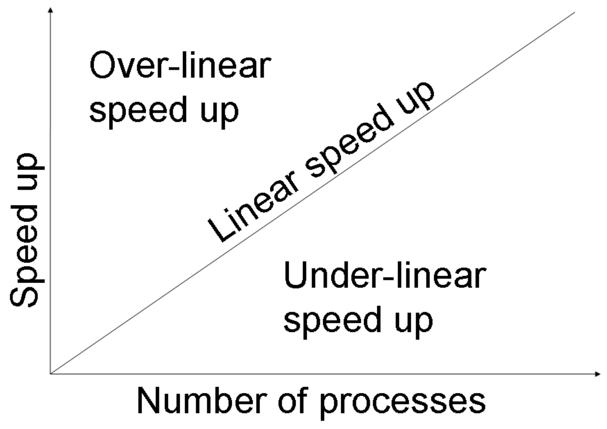


Figure B.5: Speed up

- The speed up is over-linear if the processors have an activity level higher than 100 %. That seems impossible under Amdhal's rule.
  Let B a problem implemented in parallel. f is the part of the program that has to be executed sequentially and p the number of processors. The Amdhal's rule is the following one:
  $$\text{Speed up} \leq \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \tag{B.4}$$
  This rule is optimist in the sense that it does not take into account the additional work due to parallelization (communications between processors). It is pessimist in the sense that it does not take into account the algorithm that is used, the use of hierarchic memory and the part of sequential time of the application, which is a decreasing function of the global time.
  An over-linear speed up can nevertheless occurs: indeed, in the case of a multi-processors computer, the parallel algorithm can allow a better use of hierarchic memory. It can also occurs when the parallel algorithm look for different ways to solve the problem more quickly.

B.2 Parallelization of the optimal sensor network design algorithm

To reduce the computing time, the optimal sensor network design algorithm has been parallelized in two ways, which are explained in this section: global parallelization and distributed genetic algorithms.
Both parallelized versions of the algorithm have been tested on two different computer clusters:

- the first cluster was placed to our disposal during some months by the company Hemeris (http://www.hemeris.com). It was composed 64 nodes (Apple Power Mac

G4 bi-processors).

- the second one was composed of 12 computers of the network available for students in the chemical engineering department. Eight of them had Celeron 2.2 GHz processors while the four others had Pentium 4.3 GHz processors.

B.2.1 Global parallelization (GP)

The objective of parallelization is to decrease the computing time while keeping an optimal efficiency. In our case, the limiting task is the evaluation of the fitness function of each individual of the population. This evaluation requires the factorization of at least two square matrices (whose size is equal to the sum of the number of equations and the number of variables of the problem):

- the first matrix that is factorized is the sensitivity matrix of the model, evaluated assuming ideal thermodynamic. This matrix is used to ensure that the incidences of the pressures on the enthalpies are not taken into account in the analysis of the occurrence matrix of the problem.

- the other matrices are the incidence matrices of all the operating points. The inversion of those matrices allows to estimates a posteriori variances for the candidate sensor network.

The fitness evaluations can be shared between several processors, each of them calculating the goal function of one or several individuals of the population. The other tasks are carried out by the master processor only. They consists of the population evolution, the distribution of the individuals between the different processors, the search of the optimal fitness value between those of the current population, the report generation. Those operations take very little time in comparison to matrix inversions, so that their sequential treatment only introduces very small losses of efficiency. Another loss of efficiency comes from the communications between the master processor and his slaves.
In a first time, the task of fitness evaluation has been shared between a number of processors p equal to the size of the population N_{pop}. If the master processor remains idle while each slave is evaluating the fitness of one individual, there will be a great loss of efficiency, so that is was decided that the master processor would also estimate the fitness of one individual. Thus there is one master processor and $N_{pop} - 1$ slaves.
Once the environment has been initialized, all processors execute several identical operations:

- reading the Vali4 reports model files and the data files;

- determination of all possible sensors for the processor;

- evaluation of the fitness of the first individual;

- creation of the first population of individuals.

This list of operations is executed by all the processors because all the slaves need to know most of the information contained in the files and the information about the potential sensors. If all processors did not carry out all those operations, the information should be transferred to each of them by means of collective communications. Moreover, those communications between the master and the slaves would introduce losses of efficiency. While the master was alone to carry out those operations, the slaves would remain idle. Requiring of the slaves to carry out those operations in parallel does not change the elapsed computing time and should not create losses of efficiency.
Once all those operations have been completed, each processor evaluates the goal function of one individual and sends the value to the master that receives it. To avoid to carry out one point-to-point operation by slave, the gather routine is used.
After reception of the values of the fitness, the master processor compares those values to the value corresponding to the current best solution. If one of them is better, the chromosome corresponding to this new best network is kept in memory.
Then, the master prepares the next generation by means of selection, reproduction, crossover and mutation. New chromosomes are stocked in a vector which is distributed to all processors by means of the scatter routine. Each processor evaluates the fitness of the received chromosome and sends the value to the master, and so on until the solution is achieved. When the convergence is detected, the master reevaluates the best individual and generates the report before the environment is deactivated.
A flow diagram of this parallelization mode of the programme is drawn in figure B.6.
The global parallelization of the optimal sensor network design algorithm allows a gain of time smaller than the number of active processors, so that the efficiency is lower than 100 %. As it was mentioned before, the losses of efficiency come from the communications between the master processors and the slaves and from the sequential parts of the program. However, another source of efficiency loss comes from the fact that some individuals correspond to singular sensitivity matrices. In that case, the inversion of the sensitivity matrices relative to the operating points is not carried out and the corresponding processor remains idle until the next call of the gather routine, which allows to collect the fitness values of all individuals. Moreover, if all machines were not identical, the gain should be smaller because they would not have the same execution speed so that some machines should wait a longer time before the call of the gather routine.

In a second attempt, the program has been transformed so that the number of processors could be smaller than the number of individuals of the population. In this case, each processor estimates the fitness of a number k_{chrom} of chromosomes determined as follow:

- if the number of processors is a divisor of the size of the population N_{pop}:

$$k_{chrom} = \frac{N_{pop}}{p} \tag{B.5}$$

- if the number of processors is not a divisor of the size of the population and if r_{chrom} is the remainder of the division of the number of chromosomes by the number of

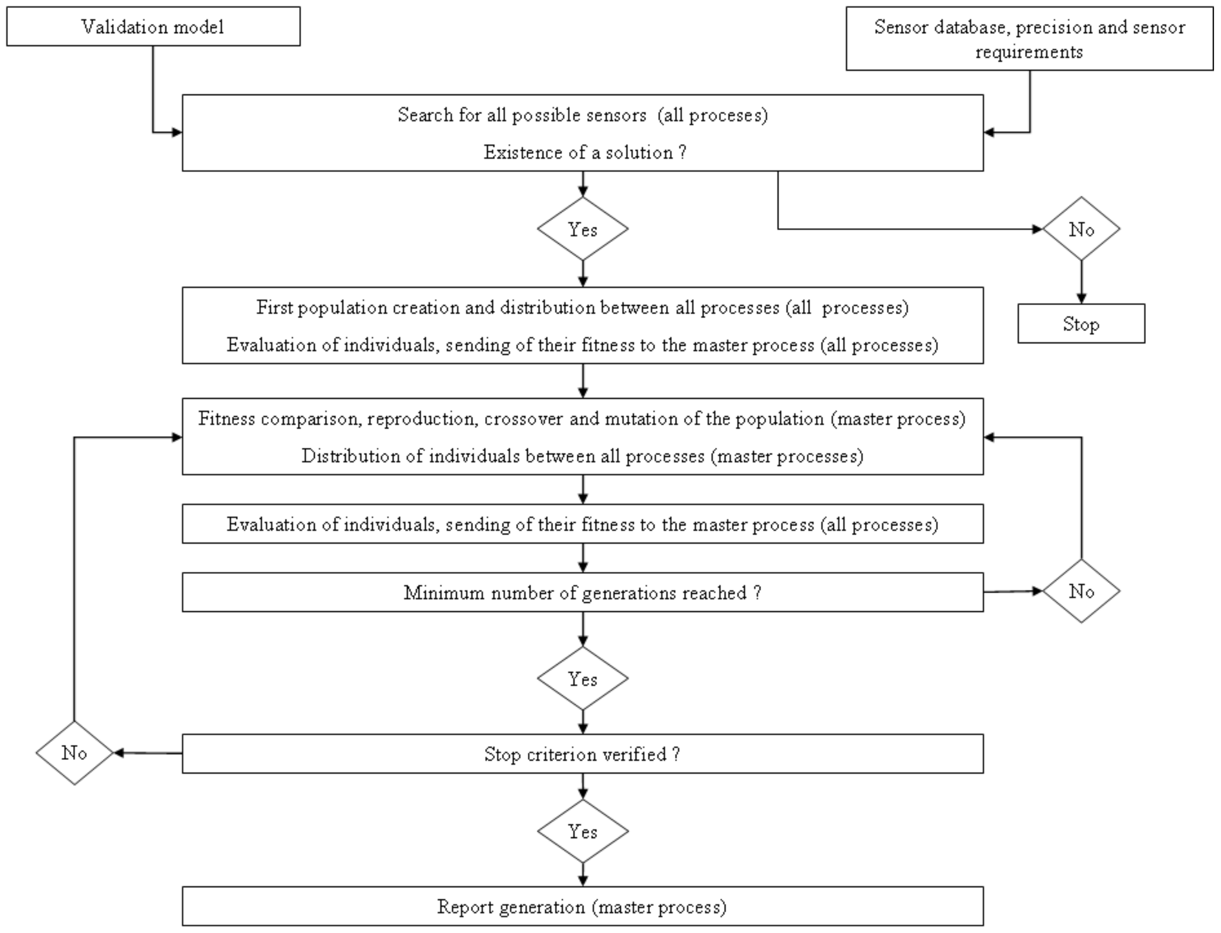

Validation model
Sensor database, precision and sensor requirements
Search for all possible sensors (all proceses)
Existence of a solution ?
Yes
No
Stop
First population creation and distribution between all processes (all processes)
Evaluation of individuals, sending of their fitness to the master process (all processes)
Fitness comparison, reproduction, crossover and mutation of the population (master process)
Distribution of individuals between all processes (master processes)
Evaluation of individuals, sending of their fitness to the master process (all processes)
Minimum number of generations reached ?
No
Yes
Stop criterion verified ?
No
Yes
Report generation (master process)

Figure B.6: Flow diagram of global parallelization

processors:

$$k_{chrom} = \frac{N_{pop}}{p} + 1 \tag{B.6}$$

for the r_{chrom} first processors and

$$k_{chrom} = \frac{N_{pop}}{p} \tag{B.7}$$

for the other processors.

To achieve the best possible efficiency, the number of processors must be a divisor of the number of chromosomes. In that case, each processor has the same work if it is not taken into account that some individuals correspond to a singular sensitivity matrix. The losses of efficiency have the same origines has explained before.

In a last attempt, the case of a number of processors higher than the size of the population has been studied. This version of the program can only be used if the objective is to determine a sensor network that remains observable in the case of one sensor failure. Indeed, the number of fitness evaluation per chromosome is then equal to the number of sensors in the network plus one, so that several processors can carry out those evaluations. Let n_{sensors} be the number of sensors in the network and r_{sensors} the remainder of the division of the number of sensors in the chromosomes plus one by the number of processors. The number of evaluations n_{eval} that are carried out by one processor for one chromosome is determined this way:

$$n_{eval} = \frac{n_{\text{sensors}} + 1}{p} + 1 \tag{B.8}$$

for the r_{sensors} first processes and

$$n_{eval} = \frac{n_{\text{sensors}}}{p} + 1 \tag{B.9}$$

for the other processors.
Information is transferred this way: the first processor that is in charge of one chromosome, receives the fitness values from the other processors in charge of this chromosome and look for the worst one. The worst fitness values of each chromosome are then collected by the master processor, which compares them as before.
This version of the algorithm needs a lot of computing ressources to be applied, so that is is not the most interesting one.

B.2.2 Distributed genetic algorithms (DGA)

Another way to parallelize the genetic algorithm is to distribute the individuals of the population between several sub-populations. This parallelization method is called *distributed genetic algorithms*. It allows to avoid premature convergence to a local minimum that is different from the optimal one and should be faster than the global parallelization.

This method has been implemented this way: after the verification of the existence of a solution for the problem, the individuals of the first generation are shared between the sub-populations, so that each sub-population starts with different individuals. The first individual of each sub-population is then replaced by the chromosome corresponding to the sensor network containing all possible measurement tools. This way, we ensure that each sub-population contains at least one individual that corresponds to a feasible solution for the problem. Each sub-population evolves independently from one another tanks to the genetic algorithm mechanisms (tournament selection, reproduction, single-point cross-over and jump mutation). As the individuals are different at the beginning of the search, each sub-population evolves in a different way. After a number $n_{\text{generation}}$ of generations, the fitness value of the best solution of each sub-population is collected by the master processor, which compares all of them. If this value remains unchanged after N collects of the master processor ($N * n_{\text{generation}}$ generations in the sub-populations), the stop criterion is satisfied and the solution is reached. If the stop criterion is not satisfied, a fixed number of individuals are chosen at random in each sub-populations and are sent to the master processor, which redistributes them at random in the different sub-populations. The two chromosomes that are replaced by the moving ones in each sub-populations are also chosen at random. If one of the replaced chromosome corresponds to the best solution of the sub-population, it is not replaced and the sub-population has only one new individual. The move of individuals is called *migration*. Instead of being chosen at random, the moving chromosomes could be the best ones of each sub-population, and the replaced individuals, the worst ones of each sub-population. This method for choosing chromosomes has not been adopted, even if it would converge faster, because it would have been the same as keeping only the best individuals of each sub-population and would have lead to a loss of information contained in the global population. Indeed, even if they have a worse fitness than the best individuals, those chromosomes that would be eliminated contain genes that can make the sub-populations evolve favorably. A flow diagram of distributed genetic algorithms is drawn on figure B.7.

There are four parameters in distributed genetic algorithms that can be modified to obtain more quickly the solution and with as few computing ressources as possible:

- The size of the sub-populations: it must be high enough to allow a continuous evolution of the individuals of each sub-population until the obtention of the solution, independently of the migrations. Moreover, if it is too small, the algorithm converges prematurely to a local minimum. A size of sub-populations comprises between 10 and 20 seems to be reasonable as the size of the population was fixed to 20 for the initial non-parallelized algorithm. In the next chapter, results are discussed for sub-populations of 10 and 20 individuals in the case of an ammonia synthesis loop.

- The number of sub-populations: this parameter is also studied in the next chapter;

- The number of generations in the sub-populations between two migrations. In literature, a value of five can be found. To confirm this value, tests have been done on

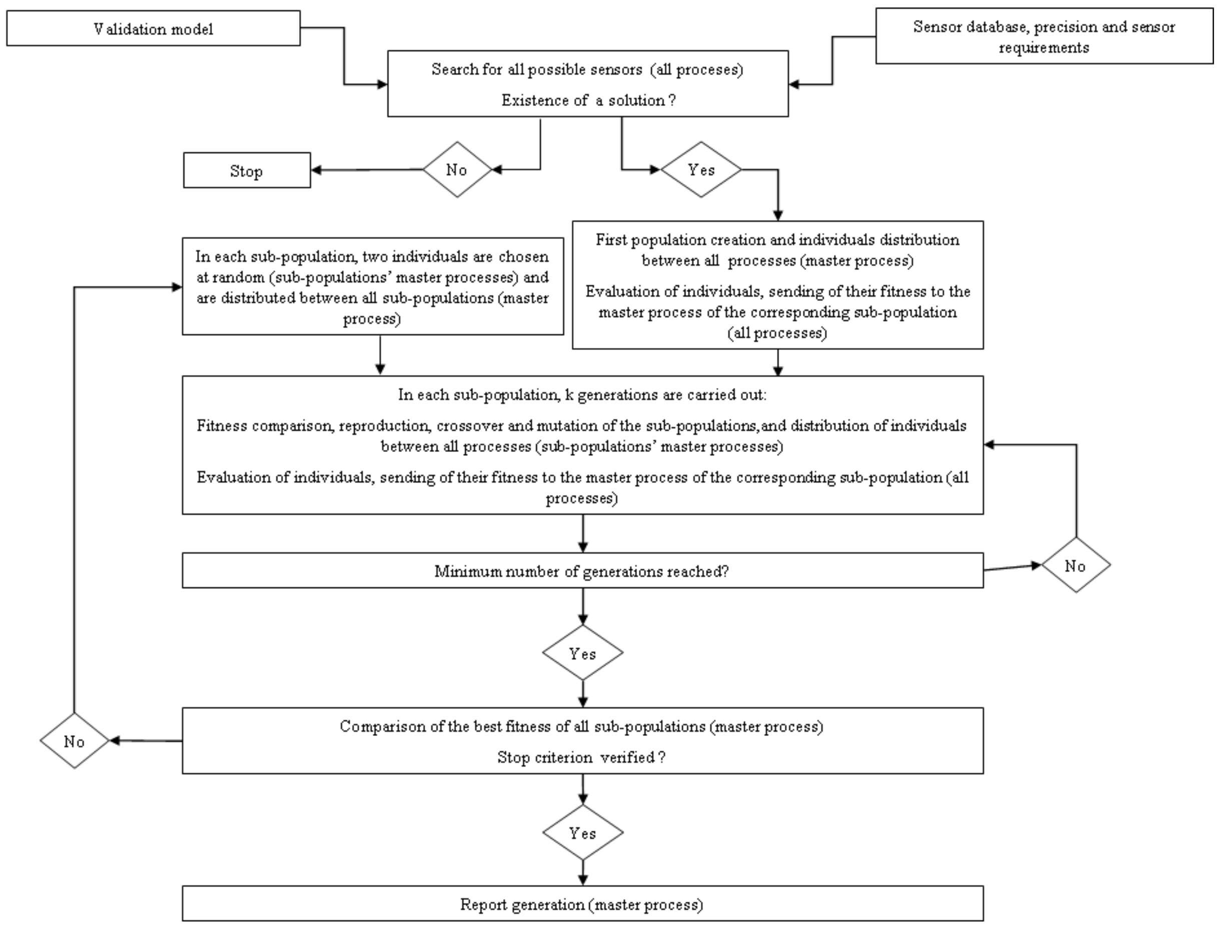

Validation model
Sensor database, precision and sensor requirements
Search for all possible sensors (all proceses)
Existence of a solution ?
Stop
No
Yes
In each sub-population, two individuals are chosen at random (sub-populations' master processes) and are distributed between all sub-populations (master process)
First population creation and individuals distribution between all processes (master process)
Evaluation of individuals, sending of their fitness to the master process of the corresponding sub-population (all processes)
In each sub-population, k generations are carried out:
Fitness comparison, reproduction, crossover and mutation of the sub-populations, and distribution of individuals between all processes (sub-populations' master processes)
Evaluation of individuals, sending of their fitness to the master process of the corresponding sub-population (all processes)
Minimum number of generations reached?
No
Yes
Comparison of the best fitness of all sub-populations (master process)
Stop criterion verified ?
No
Yes
Report generation (master process)

Figure B.7: Flow diagram of distributed genetic algorithms

the ammonia synthesis loop and the results are presented in the next chapter.

- The number of migrating chromosomes was fixed to 2 what seems to be reasonable.

Appendix C

Orthogonal collocations

In this chapter, some polynomial approximations are discussed before the choice of collocation points (Villadsen and Stewart, 1967), (Villadsen and Michelsen, 1978).

C.0.3 Polynomial approximations

Taylor series

To illustrate this method, we apply it to the solution of a differential equation of the form:

$$\frac{d^2y\left(t\right)}{dt^2}+\frac{1}{t}\frac{dy\left(t\right)}{dt}-\Phi^2\ y\left(t\right)^n=0 \tag{C.1}$$

This equation has for boundary conditions:

$$\begin{array}{ll} \frac{dy(t)}{dt}=0 & \text{for } t=0 \\ y\left(t\right)=1 & \text{for } t=1 \end{array} \tag{C.2}$$

The derivatives of the Taylor series take a simpler form if equation C.1 is rewritten in terms of $u=t^2$ for $0\leq t\leq 1$:

$$\frac{dy}{dt}=\frac{dy}{du}\ \frac{du}{dt}=\frac{dy}{du}\ 2\ \sqrt{u}\qquad \text{for } u\geq 0 \tag{C.3}$$

$$\frac{d^2y}{dt^2}=\frac{d}{dt}\ \left(2\ \sqrt{u}\ \frac{dy}{du}\right)=2\ \frac{d}{du}\left(\sqrt{u}\ \frac{dy}{du}\right)\ \frac{du}{dt}=2\ \frac{dy}{du}+4\ u\ \frac{d^2y}{du^2} \tag{C.4}$$

Thus one obtains:

$$u\ \frac{d^2y}{du^2}+\frac{dy}{du}=\frac{\Phi^2}{4}\ y=py \tag{C.5}$$

The first boundary condition

$$\left.\frac{dy}{dt}\right|_{t=0}=0 \tag{C.6}$$

is satisfied in equation C.5 as $y=f(u=t^2)$.

Thus the boundary conditions of equation C.5 are

$$y|_{u=1} = 0 \tag{C.7}$$

$$\left.\frac{d^k y}{du^k}\right|_{u=0} = y^{(k)}\big|_{u=0} \text{ finite} \qquad \text{for k=0,1,...} \tag{C.8}$$

If one differentiates equation C.5 k times with respect to u, one obtains:

$$\begin{aligned} u\ y^{(3)} + 2\ y^{(2)} &= p\ y^{(1)} \\ u\ y^{(4)} + 3\ y^{(3)} &= p\ y^{(2)} \\ u\ y^{(k+2)} + (k+1)\ y^{(k+1)} &= p\ y^{(k)} \end{aligned} \tag{C.9}$$

The following recurrence formula allows thus to build derivatives of any order at $u = 0$:

$$y^{(k+1)}\big|_{u=0} = y_0^{(k+1)} = \frac{p}{k+1}\ y_0^{(k)} = \frac{p^2}{(k+1)\ k}\ y_0^{(k-1)} = ... = \frac{p^{k+1}}{(k+1)!}\ y_0 \tag{C.10}$$

The Taylor series for $y(u)$ from $u = 0$ is defined by:

$$y(u) = \sum_{k=0}^{\infty} \frac{1}{k!}\ y_0^{(k)}\ u^k = \sum_{k=0}^{\infty} b_k\ u^k = \sum_{k=0}^{\infty} \frac{1}{k!}\ \frac{p^k}{k!}\ y_0\ u^k = \sum_{k=0}^{\infty} \frac{p^k}{(k!)^2}\ y_0\ u^k \tag{C.11}$$

The boundary condition $y|_{u=1} = 1$ allows to determine y_0 :

$$y_0 = \frac{1}{\sum_{k=0}^{\infty} \frac{p^k}{(k!)^2}} \tag{C.12}$$

Finally, one obtains for $y_N(u)$:

$$y_N(u) = \sum_{k=0}^{N} b_k\ u^k = \frac{\sum_{k=0}^{N} \frac{p^k}{(k!)^2}\ u^k}{\sum_{k=0}^{N} \frac{p^k}{(k!)^2}} \qquad N = 1, 2, ..., \infty \tag{C.13}$$

The way the coefficients b_k have been built shows that the residual $R_n(u)$ and its $(N-1)^{st}$ derivatives are zero at $u = 0$. Conversely the Taylor series is a method in which the N constants b_i are the solution of the following equations:

$$R_N^{(k)}\left[y_N(u)\right]\Big|_{u=0} = R_N^{(k)}(u)\Big|_{u=0} = 0 \qquad k = 0, 1, ..., N-1 \tag{C.14}$$

The approximations by Taylor series are unsatisfactory in the way that other approximation methods converge more quickly.

Lowest-order MWR (Weighted residuals method): $y_1(t)$ approximation

To illustrate this technique, let us solve equation C.1 in which $n = 1$. One obtains

$$y(t) = \frac{I_0(\Phi t)}{I_0(\Phi)} \tag{C.15}$$

where I_0 is a first order Bessel function :

$$I_0(x) = \sum_{i=0}^{\infty} \frac{\left(\frac{1}{4}x^2\right)^i}{(i!)^2} = 1 + \frac{1}{4}x^2 + \frac{1}{4}\left(\frac{1}{4}x^2\right)^2 + \frac{1}{36}\left(\frac{1}{4}x^2\right)^3 + ... \tag{C.16}$$

So, equation C.15 becomes:

$$y(t) = \frac{1 + \left(\frac{\Phi^2 t^2}{4}\right) + \frac{1}{4}\left(\frac{\Phi^2 t^2}{4}\right)^2 + \frac{1}{36}\left(\frac{\Phi^2 t^2}{4}\right)^3 + ...}{1 + \left(\frac{\Phi}{2}\right) + \frac{1}{4}\left(\frac{\Phi}{2}\right)^4 + \frac{1}{36}\left(\frac{\Phi}{2}\right)^6 + ...} \tag{C.17}$$

In first approximation, equation C.1 can be written

$$y_1(t) = 1 + a_1 \ \left(1 - t^2\right) \tag{C.18}$$

The term $a_1 (1 - t^2)$ is a perturbation of the known boundary value $y(t) = 1$ at $t = 1$. The equation C.18 satisfies both boundary conditions C.2. For the preceding serie C.17, $y_1(t)$ takes the form:

$$y_1(t) = \frac{1 + \frac{\Phi \ t^2}{2}}{1 + \frac{\Phi^2}{2}} \tag{C.19}$$

By transforming $y_1(t)$:

$$\begin{aligned} y_1(t) &= \frac{4 + \Phi^2 - \Phi^2 + \Phi^2 \ t^2}{4 + \Phi^2} \\ &= 1 - \frac{\Phi^2 \ (1 - t^2)}{4 + \Phi^2} \end{aligned} \tag{C.20}$$

one obtains for a_1 :

$$a_1 = -\frac{\Phi^2}{4 + \Phi^2} \tag{C.21}$$

Replacing $y_1(t)$ from equation C.18 in equation C.1 for $n = 1$, the following residual is obtained:

$$R(y_1(t)) = R(a_1, t) = -4 \ a_1 - \Phi^2 \ \left[1 - a_1 \ \left(1 - t^2\right)\right] \tag{C.22}$$

a_1 is determined so that the weighted residual $W * R$ has a zero mean value, or in an equivalent way :

$$\int_0^1 R_1(a_1, t) \ W(t) \ dt^2 = 0 \tag{C.23}$$

where $W(t)$ is a weighting function which depends on the MWR method that is chosen. Some of those methods are described later in this section.

Higher-order MWR: $y_N(t)$ approximation

If one approximates $y_N(t)$ by the following series :

$$y_N(t) = T_0 + \sum_{i=1}^{n} a_i \; T_i \tag{C.24}$$

where,

- T_0 satisfies the boundary conditions C.2 of the differential equation C.1;
- les T_i are boundary conditions satisfying the boundary conditions of the problem, namely:

  $$\left.\frac{dT_i}{dt}\right|_{t=0} = \left.T_i\right|_{t=1} = 0 \qquad \forall i > 0 \tag{C.25}$$

  The choice of those functions is generally free as long as they satisfy the conditions C.25 and are linearly independent. The following polynomial functions satisfy the conditions C.25 for equation C.1:

$$\begin{aligned} T_0 &= 1 \\ T_i &= \left(1+t^2\right) \; t^{2i-2} \qquad i = 1, 2, ... \end{aligned} \tag{C.26}$$

$y_N(t)$ can thus be written :

$$y_N(t) = 1 + \left(1+t^2\right) \; \sum_{i}^{n} a_i \; t^{2i-2} \tag{C.27}$$

If a change of variable is carried out in equation C.1:

$$u \; \frac{d^2 y(u)}{du^2} + \frac{dy(u)}{du} - p \; y(u) = 0 \tag{C.28}$$

with

$$p = \frac{\Phi^2}{4} \tag{C.29}$$

and with boundary conditions :

$$\begin{aligned} &\left.y(u)\right|_{u=1} = 1 \\ &\left.y_k(u)\right|_{u=0} \text{ finite} \qquad \text{for } k \geq 0 \end{aligned} \tag{C.30}$$

y_N can then be written in the following way:

$$y_N(u) = 1 + (1-u) \; \sum_{i=1}^{N} a_i \; u^{i-1} \tag{C.31}$$

The first and the second derivatives of equation C.31 are calculated this way:

$$\frac{dy_N(u)}{du} = \sum_{i=1}^{N} a_i \; (i-1) \; u^{i-2} - \sum_{i=1}^{N} a_i \; i \; u^{i-1} \tag{C.32}$$

$$u \, \frac{d^2y_N(u)}{du^2} = \sum_{i=1}^{N} a_i \; (i-1) \; (i-2) \; u^{i-2} - \sum_{i=1}^{N} a_i \; i \; (i-1) \; u^{i-1} \tag{C.33}$$

Replacing equations C.31, C.32 et C.33 in equation C.28, the residual becomes :

$$\begin{aligned} R_N(\mathbf{a}, u) &= u \, \frac{d^2y_N(u)}{du^2} + \frac{dy_N(u)}{du} - p \; y_N(u) \\ &= \sum_{i=1}^{N} a_i \; \left[(i-1)^2 \; u^{i-2} - i^2 \; u^{i-1} \right] - p \; \left[1 + (1-u) \; \sum_{i}^{n} a_i \; u^{i-1} \right] \end{aligned} \tag{C.34}$$

The N following relations allow to determine the coefficients a_i:

$$\int_0^1 R_N(\mathbf{a}, u) \; W_j(u) \; du = 0 \qquad j = 1, 2, ..., N \tag{C.35}$$

Each MWR is characterized by a different choice of the sequence of the N (N^{st} order) weighting functions $W_j(u)$. Five MWR are described here after:

13. **The collocations method**: in this method, the weighting functions are taken from the family of the Dirac δ functions in the domain:

$$W_j(u) = \delta(u - u_j) \qquad j = 1, 2, ..., N \tag{C.36}$$

where

$$\delta(u - u_j) = \begin{cases} 1 & u = u_j \\ 0 & \text{otherwise} \end{cases} \tag{C.37}$$

The integration of the weighted residual statement results in the forcing of the residual to zero at specific points u_j in the domain ($[0,1]$). This is equivalent to:

$$R_N(\mathbf{a}, u_j) = 0 \qquad j = 1, 2, ..., N \tag{C.38}$$

This first weighted residuals method is used for the dynamic data reconciliation algorithm described in this chapter.

14. **The sub-domains method**: this method does not use weighting factor explicitly but can be considered as a modification of the collocations method. In this method, the weighted residuals are fixed to zero not only at fixed points of the domain but over various subsections of the domain. So, the weight functions are fixed to unity and the integral over the entire domain can be broken into a number of integral equal to the number of sub-domains so that all unknown parameters can be estimated:

$$\int_{domain} R_N(\mathbf{a}, u) \, du = \sum_{i=1}^{N} \int_{u_{j-1}}^{u_j} R_N(\mathbf{a}, u) \, du = 0 \tag{C.39}$$

15. **The least squares method**: this method consists of the continuous minimization of the sum of squared residuals:

$$S = \int_{domain} R_N(\mathbf{a}, u)\ R_N(\mathbf{a}, u)\ du = \int_{domain} R_N^2(\mathbf{a}, u)\ du \tag{C.40}$$

The minimum of this function is reached when its derivative with respect to the unknown parameters a_j is zero:

$$\frac{\partial S}{\partial a_j} = 0 = 2 \int_{domain} R_N(\mathbf{a}, u)\ \frac{\partial R_N(\mathbf{a}, u)}{\partial a_j} du \qquad j = 1, 2, ..., N \tag{C.41}$$

So that the weighting functions are:

$$W_j = 2\ \frac{\partial R_N(\mathbf{a}, u)}{\partial a_j} du \qquad j = 1, 2, ..., N \tag{C.42}$$

The "2" factor can be dropped so that the weighting functions for this method are just the derivatives of the residuals with respect to the unknown parameters a_j:

$$W_j = \frac{\partial R_N(\mathbf{a}, u)}{\partial a_j} du \qquad j = 1, 2, ..., N \tag{C.43}$$

16. **The Galerkin method**: this method can be considered as a modification of the least squares method in which the derivative of the approximating function with respect to the unknown parameters a_j is used instead of the derivative of the residual:

$$W_j(u) = T_j = \frac{\partial y_N}{\partial a_j} = (1 - u)\ u^{j-1} \tag{C.44}$$

This is equivalent to:

$$\int_0^1 R_N(\mathbf{a}, u)\ (1 - u)\ u^{j-1}\ du = 0 \qquad j = 1, 2, ..., N \tag{C.45}$$

17. **The method of moments**: in this last method, the weighting functions are chosen from the family of polynomials:

$$W_j(u) = u_j^{j-1} \qquad j = 1, 2, ..., N \tag{C.46}$$

This is equivalent to:

$$\int_0^1 R_N(\mathbf{a}, u)\ u^{j-1}\ du = 0 \qquad j = 1, 2, ..., N \tag{C.47}$$

Non linear problem

Let equation C.28 with $n \neq 1$:

$$u \, \frac{d^2 y(u)}{du^2} + \frac{dy(u)}{du} - p \, y^n(u) = 0 \tag{C.48}$$

with $y|_{u=1}$=1.
As before, the first order approximation is written:

$$y_1(u) = 1 + a_1 \ (1-u) \tag{C.49}$$

It has for residual:

$$R_1(a_1, u) = -a_1 - p \ [1 + a_1 \ (1-u)]^n \tag{C.50}$$

If the collocations method is used:

$$R_1(a_1, u) = -a_1 - p \ [1 + a_1 \ (1-u)]^n = 0 \tag{C.51}$$

This equation is non linear and must be solved numerically. It is generally not possible to obtain a parametric solution in p.
If Galerkin method is used:

$$\int_0^1 R_1(a_1, u) \ (1-u) \ du = \int_0^1 [-a_1 - p \ [1 + a_1 \ (1-u)]^n] \ (1-u) \ du = 0 \tag{C.52}$$

If n is not an integer, this integral must be evaluated numerically and it is not possible to obtain an explicit equation in a_1. Even if n is an integer (n=2,3,...), complex algebraic manipulations are needed to evaluate the integral. A choice must be done between the collocations method which is easily applicable but somewhat less precise and the more precise Galerkin method which is more difficult to use. A compromise could be to approximate the integral C.52 thanks to a numerical quadrature, like the Gauss-Jacobi quadrature or the Radau-Lobatto quadrature.

Optimal Fourier type expression

For the following approximation

$$y_N(u) = 1 + (1-u) \sum_{i=1}^{N} a_i \ u^{i-1} \tag{C.53}$$

the boundary condition $y(u) = 1$ for $u = 1$ has been used to eliminate one of the $N+1$ parameters a_i that are necessary to the determination of a polynomial of degree N. A polynomial of degree $N-1$ can then be extracted:

$$g_{N-1}(u) = \frac{y_N(u) - 1}{1-u} = \sum_{i=1}^{N} a_i \ u^{i-1} = \sum_{i=1}^{N} b_i \ P_{i-1}(u) \tag{C.54}$$

The $P_i\left(u\right)$ are often chosen as orthogonal polynomials of degree N in u and satisfying the following equation:

$$\int_0^1 u^{\beta}\ \left(1-u\right)^{\alpha}\ u^j\ P_N^{\alpha,\beta}\left(u\right)\ du = 0 \qquad j = 0, 1, ..., N-1 \tag{C.55}$$

Those polynomials can also take the form:

$$P_i^{\alpha,\beta}\left(u\right) = \gamma_i\ u^i - \gamma_{i-1}\ u^{i-1} + \gamma_{i-2}\ u^{i-2} - ... + \left(-1\right)^i \tag{C.56}$$

They can also be normalized so that the leading coefficient is 1:

$$P_i^{\alpha,\beta}\left(u\right) = u^i - \gamma_{i-1}^{'}\ u^{i-1} + \gamma_{i-2}^{'}\ u^{i-2} - ... + \left(-1\right)^i\ \gamma_0^{'} \tag{C.57}$$

In equations C.56 and C.57, all γ_i and $\gamma_i^{'}$ are positives.
The advantage of this reformulation of equation C.31 is that the coefficients b_i allow to obtain a more rapid convergence than the a_i series that decreases less quickly.

Lagrange interpolation polynomials

Two kinds of polynomial approximations of degree N have been discussed previously:

$$y_N = \sum_{i=0}^{N} a_i\ x^i \tag{C.58}$$

$$y_N = \sum_{i=0}^{N} b_i\ P_i\left(t\right) \tag{C.59}$$

In this paragraph, a third type of approximations is discussed: the Lagrange interpolation polynomials:

$$y_N = \sum_{i=0}^{N} y_i\ l_i\left(t\right) \tag{C.60}$$

In this formula:

- the y_i are the values of y_N at the interpolation points t_i;
- the $l_i\left(t\right)$ are Lagrange interpolation polynomial of degree N defined as follow:

$$l_i\left(t\right) = \frac{p_N\left(t\right)}{\left(t-t_i\right)\ p_N^{(1)}\left(t\right)} \tag{C.61}$$

- $p_N\left(t\right)$ is a polynomial of degree N with a leading coefficient equal to 1, called the *node polynomial.*

$$p_N\left(t\right) = \left(t-t_0\right)\ \left(t-t_2\right)...\left(t-t_N\right) \tag{C.62}$$

This equation can be transformed:

$$\prod_{\substack{j=0\\ j\neq i}}^{N} (t-t_j) = \frac{p_N(t)}{t-t_i} \qquad \text{if } t \neq t_i \tag{C.63}$$

$$\prod_{\substack{j=0\\ j\neq i}}^{N} (t_i-t_j) = \frac{dp_N(t)}{dt} \qquad \text{if } t = t_i \tag{C.64}$$

The $l_i(t)$ can then be written:

$$l_i(t) = \prod_{\substack{j=0\\ j\neq i}}^{N} \frac{(t-t_j)}{(t_i-t_j)} \tag{C.65}$$

with

$$l_i(t_j) = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{C.66}$$

As a reasonable approximation of the y_i at the collocation times is quite easy to obtain if enough measurements are carried out during the chosen time interval, this method will be preferred to the two other ones previously studied. Nevertheless, it should be noted that the $l_i(t)$ can not be derived or integrated as easily as the t^i or the P_i :

$$\frac{d^k y_N}{dt^k} = \sum_{i=0}^{N} a_i \left[\frac{d^k}{dt^k} t^{i-1}\right] = \left[\frac{d^k}{dt^k}\mathbf{t}\right]^T \mathbf{a} \tag{C.67}$$

$$\int_0^1 W(t)\; y_N dt = \sum_{i=0}^{N} a_i \left[\int_0^1 W(t)\; t^{i-1} dt\right] = \left[\int_0^1 W(t)\; \mathbf{t}\; dt\right]^T \mathbf{a} \tag{C.68}$$

Similar formulas can nevertheless be obtained for the Lagrange interpolation polynomials:

$$\frac{d^k y_N}{dt^k} = \left[\frac{d^k}{dt^k}\mathbf{l}(t)\right]^T \mathbf{y} \tag{C.69}$$

$$\int_0^1 W(t)\; y_N dt = \left[\int_0^1 W(t)\; \mathbf{l}(t)\; dt\right]^T \mathbf{y} \tag{C.70}$$

Demonstration of the derivative formula C.69 is described in the following paragraph.

Derivative of the Lagrange interpolation polynomials

The first derivative of equation C.60 gives:

$$\frac{d}{dt}\left[y_N\left(t\right)\right] = \sum_{i=0}^{N} \frac{dl_i\left(t\right)}{dt}\ y\left(t_i\right) = \left[\mathbf{l}^{(1)}\right]^T\ \mathbf{y} \tag{C.71}$$

One has for the following orders:

$$\frac{d^k y_N\left(t\right)}{dt^k}\left[\mathbf{l}^{(k)}\right]^T\ \mathbf{y} \tag{C.72}$$

The interesting derivatives are the ones obtained at the $N+1$ interpolation points x_j. If one transforms the following formula

$$l_i = \frac{p_N\left(t\right)}{\left(t-t_i\right)\ p_N^{(1)}\left(t_i\right)} \tag{C.73}$$

one obtains :

$$\frac{p_N\left(t\right)}{p_N^{(1)}\left(t_i\right)} = \left(t-t_i\right)\ l_i\left(t\right) \tag{C.74}$$

Deriving with respect to t:

$$\frac{p_N^{(1)}\left(t\right)}{p_N^{(1)}\left(t_i\right)} = \left(t-t_i\right)\ l_i^{(1)}\left(t\right) + l_i\left(t\right) \tag{C.75}$$

$$\frac{p_N^{(2)}\left(t\right)}{p_N^{(1)}\left(t_i\right)} = \left(t-t_i\right)\ l_i^{(2)}\left(t\right) + 2\ l_i^{(1)}\left(t\right) \tag{C.76}$$

$$\frac{p_N^{(k)}\left(t\right)}{p_N^{(1)}\left(t_i\right)} = \left(t-t_i\right)\ l_i^{(k)}\left(t\right) + k\ l_i^{(k-1)}\left(t\right) \tag{C.77}$$

So one obtains for the Lagrange polynomial derivatives:

$$l_i^{(k-1)}\left(t_i\right) = \frac{1}{k}\ \frac{p_N^{(k)}\left(t_i\right)}{p_N^{(1)}\left(t_i\right)} \qquad \text{if } t = t_i \tag{C.78}$$

$$l_i^{(k)}\left(t_i\right) = \frac{1}{t_j - t_i}\ \left[\frac{p_N^{(k)}\left(t_i\right)}{p_N^{(1)}\left(t_i\right)} - k\ l_i^{(k-1)}\left(t_j\right)\right] \qquad \text{if } t = t_j \neq t_i \tag{C.79}$$

For most of differential equations, only the two first orders are needed. They are:
If $t = t_i$

$$l_i^{(1)}\left(t_i\right) = \frac{1}{2}\frac{p_N^{(2)}\left(t_i\right)}{p_N^{(1)}\left(t_i\right)} \tag{C.80}$$

$$l_i^{(2)}\left(t_i\right) = \frac{1}{3}\frac{p_N^{(3)}\left(t_i\right)}{p_N^{(1)}\left(t_i\right)} \tag{C.81}$$

Otherwise, $t = t_j \neq t_i$

$$\begin{aligned}
l_i^{(1)}(t_j) &= \frac{1}{t_j - t_i} \qquad \text{as } l_i(t_j) = 0 && \text{(C.82)}\\
l_i^{(2)}(t_j) &= \frac{1}{t_j - t_i}\left[\frac{p_N^{(2)}(t_i)}{p_N^{(1)}(t_i)} - 2\, l_i^{(1)}(t_j)\right] && \text{(C.83)}\\
&= l_i^{(1)}(t_j)\left[\frac{p_N^{(2)}(t_i)}{p_N^{(1)}(t_i)} - 2\,\frac{1}{t_j - t_i}\right] \qquad \text{as } l_i^{(1)}(t_j) = \frac{1}{t_j - t_i} && \text{(C.84)}\\
&= 2\, l_i^{(1)}(t_j)\left[\frac{p_N^{(2)}(t_i)}{2\, p_N^{(1)}(t_i)} - \frac{1}{t_j - t_i}\right] && \text{(C.85)}\\
&= 2\, l_i^{(1)}(t_j)\left[l_j^{(1)}(t_j) - \frac{1}{t_j - t_i}\right] && \text{(C.86)}
\end{aligned}$$

The next formula allows to calculate the derivatives of the node polynomial at the interpolation points t_j

$$p_N(t) = \prod_{i=0}^{N}(t - t_i) \tag{C.87}$$

From there, one has the following recurrence formula:

$$\begin{aligned}
p_0(t) &= 1\\
p_j(t) &= (t - t_j)\; p_{j-1}(t) \qquad j = 0, 1, 2, ..., N && \text{(C.88)}
\end{aligned}$$

Deriving the equation C.88, one obtains

$$\begin{aligned}
p_j^{(1)}(t) &= (t - t_j)\; p_{j-1}^{(1)}(t) + p_{j-1} && \text{(C.89)}\\
p_j^{(2)}(t) &= (t - t_j)\; p_{j-1}^{(2)}(t) + 2\, p_{j-1}^{(1)} && \text{(C.90)}\\
p_j^{(3)}(t) &= (t - t_j)\; p_{j-1}^{(3)}(t) + 2\, p_{j-1}^{(2)} && \text{(C.91)}
\end{aligned}$$

with

$$p_0^{(1)}(t) = p_0^{(2)}(t) = p_0^{(3)}(t) = 0$$

If t is replaced by t_i and if the interpolation points are reordered in a way that the t_i at which the derivatives are calculated is placed in first position, the preceding equations become:

$$\begin{aligned}
p_0(t_i) &= 1 && \text{(C.92)}\\
p_0^{(k)}(t_i) &= 0 \qquad k = 1, 2, 3 && \text{(C.93)}\\
p_1(t_i) &= (t_i - t_i)\; p_0(t_i) = 0 && \text{(C.94)}\\
p_1^{(1)}(t_i) &= (t_i - t_i)\; p_0^{(1)}(t_i) + p_0(t_i) = 1 && \text{(C.95)}\\
p_1^{(2)}(t_i) &= (t_i - t_i)\; p_0^{(2)}(t_i) + 2\, p_0^{(1)}(t_i) = 0 && \text{(C.96)}\\
p_1^{(3)}(t_i) &= (t_i - t_i)\; p_0^{(3)}(t_i) + 3\, p_0^{(2)}(t_i) = 0 && \text{(C.97)}
\end{aligned}$$

For $j = 1, 2, 3, ..., i-1, i+1, ..., N$, one has:

$$\begin{aligned}
p_j(t_i) &= (t_i - t_j)\ p_j(t_i) = 0 && \text{(C.98)}\\
p_j^{(1)}(t_i) &= (t_i - t_j)\ p_j^{(1)}(t_i) + p_j(t_i) = 1 && \text{(C.99)}\\
p_j^{(2)}(t_i) &= (t_i - t_j)\ p_j^{(2)}(t_i) + 2\ p_j^{(1)}(t_i) = 0 && \text{(C.100)}\\
p_j^{(3)}(t_i) &= (t_i - t_j)\ p_j^{(3)}(t_i) + 3\ p_j^{(2)}(t_i) = 0 && \text{(C.101)}
\end{aligned}$$

C.0.4 Determination of the collocation nodes

There exists several possible choices for the collocation nodes: the nodes can be chosen by the user of the method, at random, equidistant, at the measurements times, at the zeros of orthogonal polynomials,... Villadsen and Michelsem (Villadsen and Michelsen, 1978) have shown that the collocation nodes chosen at the zeros of orthogonal polynomials give the best results for the solving of many types of differential equations. Other authors like Mingfang et al. (Kong et al., 2000) proposed to choose the measurement times as collocation nodes in the case of dynamic data reconciliation. In the case of this study, the collocation points are chosen at the zeros of the Jacobi orthogonal polynomials. Those zeros are determined using Newton's method with suppression of the previously determined zeros.

Jacobi orthogonal polynomials are written this way:

$$P_N^{(\alpha,\beta)}(t) = \sum_{i=0}^{N} (-1)^{N_i}\ \gamma_i\ t^i \qquad \text{(C.102)}$$

In this equation, γ_0 is chosen equal to 1 and the N other coefficient γ_i are determined thanks to the orthogonality property:

$$\int_0^1 t^\beta\ (1-t)^\alpha\ P_j(t)\ P_N(t)\ dt = 0 \qquad j = 0, 1, ..., N-1 \qquad \text{(C.103)}$$

or equivalently:

$$\int_0^1 t^\beta\ (1-t)^\alpha\ t^j\ P_N(t)\ dt = 0 \qquad j = 0, 1, ..., N-1 \qquad \text{(C.104)}$$

where the t_j are linear combinations of the P_k with $k = 0, 1, ..., N-1$.
If P_N is replaced in equation C.104 by its value in formula C.102:

$$\int_0^1 t^\beta\ (1-t)^\alpha\ t^j \sum_{i=0}^{N} (-1)^{N_i}\ \gamma_i\ t^i\ dt = 0 \qquad j = 0, 1, ..., N-1 \text{ and } i = 0, 1, ..., N \qquad \text{(C.105)}$$

Let, for fixed i:

$$\int_0^1 t^{\beta+i+j}\ (1-t)^\alpha\ (-1)^{N_i}\ \gamma_i\ dt = 0 \qquad j = 0, 1, ..., N-1 \text{ and } i = 0, 1, ..., N \qquad \text{(C.106)}$$

Knowing that:

$$\int_0^1 t^m \ (1-t)^n = \frac{\Gamma(m+1) \ \Gamma(n+1)}{\Gamma(m+n+2)} \tag{C.107}$$

one obtains:

$$(-1)^{N_i} \ \gamma_i \int_0^1 t^{\beta+i+j} \ (1-t)^{\alpha} = (-1)^{N_i} \ \gamma_i \ \frac{\Gamma(\beta+i+j+1) \ \Gamma(\alpha+1)}{\Gamma(\alpha+\beta+i+j+2)}$$
$$j = 0, 1, ..., N-1 \text{ and } i = 0, 1, ..., N \tag{C.108}$$

If $\alpha = 0$ and $\beta = 0$, Legendre polynomials are obtained :

$$(-1)^{N_i} \ \gamma_i \int_0^1 t^{i+j} \ (1-t)^0 = (-1)^{N_i} \ \gamma_i \ \frac{\Gamma(i+j+1) \ \Gamma(1)}{\Gamma(i+j+2)}$$
$$j = 0, 1, ..., N-1 \text{ and } i = 0, 1, ..., N \tag{C.109}$$

Now, by definition:

$$\Gamma(1) = 1 \tag{C.110}$$
$$\Gamma(n+1) = n \ \Gamma(n) = \Gamma(1) \ (n-1)! \tag{C.111}$$

Thus one obtains:

$$(-1)^{N_i} \ \gamma_i \int_0^1 t^{i+j} \ (1-t)^0 = (-1)^{N_i} \ \gamma_i \ \frac{(i+1)!}{}\Gamma(i+j+1)!$$
$$j = 0, 1, ..., N-1 \text{ and } i = 0, 1, ..., N \tag{C.112}$$

The next formula allows to obtain the γ_i :

$$\gamma_i = \frac{N!}{i! \ (N-i)!} \ \frac{\Gamma(N+\alpha+\beta+i+1) \ \Gamma(\beta+1)}{\Gamma(N+\alpha+\beta+1) \ \Gamma(\beta+i+1)} \tag{C.113}$$

From there, the following recurrence formula is obtained:

$$\gamma_i = \frac{N-i+1}{i} \ \frac{N+\alpha+\beta+i}{}\beta + i\gamma_{i-1} \qquad \gamma_0 = 1 \text{ and } i = 1, 2, ..., N \tag{C.114}$$