Particle Filtering for Open-Loop Process Control

A Users' Guide for a MATLAB Toolbox

on Genealogical Decision Tree-Based Optimization

Enso Ikonen

February 24, 2006

Contents

Abstract

E. Ikonen: Particle Filtering for Open-Loop Process Control – A Users' Guide for a MATLAB Toolbox on Genealogical Decision Tree-Based Optimization

Advances in population-based evolutionary algorithms have introduced new tools for optimizing and controlling complex systems. This report considers a new algorithm that falls within the area of mathematical population genetics and interacting particle systems: the genealogical decision trees. These particle filter systems belong to stochastic search models. Their robustness and flexibility make them particularly attractive for applications to optimization and control of complex industrial processes.

The text is organized as follows. We start with a brief introduction to Bayesian filters and particle filters. In Section 3, the Genealogical Decision Tree (GDT) algorithm for open-loop control optimization is described. The MATLAB MGDT-Toolbox implementing the algorithm is presented in Section 4. A number of case studies with different types of processes, models and trajectories are reported in Section 5, so as to illustrate the performance and viability of the approach. To conclude, Section 6 discusses extensions of the algorithm and possible future directions.

Key words: automation, control engineering, control theory, evolutionary algorithms, genetic algorithms, Matlab, optimization, open-loop control, particle filtering, process engineering, programming, random search, stochastic processes, trajectory following

Tiivistelmä

E. Ikonen: Partikkelifiltterit prosessien avoimen piirin säädössä – Käyttäjän ohje genealogisen päätöksentekopuu -perustaisen optimoinnin MATLAB-työkaluun

Kehitys populaatioperustaisissa evoluutioalgoritmeissa on tuonut uusia työkaluja monimutkaisten systeemien optimointiin ja säätöön. Tässä raportissa käsitellään uutta algoritmia joka sijoittuu matemaattisen populaatiogenetiikan ja vuorovaikutteisten partikkelisysteemien alueelle. Nämä partikkelifiltterisysteemit kuuluvat stokastisiin hakumenetelmiin. Niiden robustisuus ja joustavuus tekee niistä erityisen houkuttelevia monimutkaisten teollisuusprosessien säädön ja optimoinnin sovelluksiin.

Tämä raportti on jäsennelty seuraavasti. Aloitamme lyhyellä johdannolla Bayesin filttereihin ja partikkelifilttereihin, Luku 2. Luvussa 3 kuvataan GDT algoritmi – Genealogical Decision Tree, eli genealoginen ('sukuhistoriallinen') päätöksentekopuu – avoimen piirin säätösekvenssin optimoimista varten. Menetelmän implentoiva Matlabin MGDT-kirjasto esitellään luvussa 4. Luvussa 5 esitellään lähestymistavan suorituskykyä ja soveltuvuutta lukuisalla määrällä esimerkkejä joissa käsitellään erityyppisiä prosesseja, malleja ja vasteita. Lopuksi pohditaan hieman menetelmän laajennuksia ja tulevaisuuden tutkimussuuntia.

Resumé

E. Ikonen: Méthodes particulaires pour la commande en boucle ouverte - Guide d'utilisateurs pour une bôite à outils MATLAB pour l'optimisation basée sur les arbres de décision généalogiques.

Les algorithmes évolutionnaires basés sur l'évolution des populations biologiques représentent des outils nouveaux pour l'optimisation et la commande des systèmes complexes. Ce rapport considère un nouvel algorithme de commande basé sur des processus historiques et arbres généalogiques pour la commande de systèmes complexes. Ces systèmes de filtre de particules appartiennent aux modèles stochastiques de recherche. Leur robustesse et flexibilité les rendent particulièrement attirants pour des applications à l'optimisation et à la commande des processus industriels complexes.

Ce rapport est organisé comme suit. Nous commençons par une brève introduction aux filtres Bayésiens et aux méthodes particulaires pour le filtrage, Section 2. Dans la troisième Section, un algorithme de commande basé sur des processus historiques et arbres généalogiques (Genealogical Decision Trees, GDT) pour la commande en boucle ouverte de systèmes complexes est décrit. La MGDT-boîte à outils Matlab qui permet la mise en ouevre de cet algorithme de commande est présentée dans la Section 4. Pour illustrer la faisabilité et la performance de cet algorithme de commande pour la régulation et la poursuite de trajectoires, divers exemples sont présentés dans la Section 5. Enfin, la Section 6 traite des futures directions de recherche.

Preface

In autumn 2003, Professor K. Najim told me that he had interesting developments together with Professor Del Moral, both in Toulouse at the time, related to genetic and random search techniques. I browsed the first drafts of the method in winter 2003–2004. My role in the co-operation was then to prepare some simulations. This report summarizes the work conducted based on these ideas since then.

The first results were published in the IFAC AFNC (Workshop on Advanced Fuzzy / Neural Control) in Oulu (spring 2004). Since then a series of manuscripts and conference papers has been prepared, illustrating the power of the approach with models of various industrial processes from process engineering, chemical engineering, energy conversion and robotics. The method searches for an optimal open-loop contol sequence, so as to follow a given trajectory. Extensions with additional feed-back control have also been considered.

The main purpose of this report is to provide a users' guide for the MATLAB-functions implementing the method, in order to popularize the approach. To enable those less familar with the area of particle filters to grasp the essential, a short introduction to particle filters (a.k.a. sequential Monte Carlo methods) is given. This is followed by the application of the techniques to the optimization of a control sequence in trajectory following. The few functions needed for implementing the algorithm are described in detail. Clear and easy-to-follow guidelines are suggested for initial tuning of the algorithm parameters. These are evaluated in various types of simulation examples. The source code (for MATLAB 6 – R12) is publically available at `http://cc.oulu.fi/~iko/MGDT.htm`.

Alltogether, a number of colleagues have participated to the reported research. All merits about the basic algorithm (Sections 3.1–3.2) should be addressed to Professor P. Del Moral (Nice, France) and Professor K. Najim (Toulouse, France). The role of E. Ikonen is more visible in process control applications: formulation in Section 3.3, introduction of incremental control, tuning rules, and feed-back control considerations. The MATLAB toolbox and all the simulations were prepared by E. Ikonen, with many of the models proposed by Prof. K. Najim. Applications to robotics were conducted in co-operation with Professor E. Gomez-Ramirez (Mexico D.F., Mexico).

Finally, I would like to address my thanks to Professor U. Kortela (Oulu, Finland) for a long term support on this line of research. Financial support from the Academy of Finland (projects 48545 and 203231) is gratefully acknowledged.

in Oulu, February 2006

Enso Ikonen
Docent (Adjunct Professor), Dr.

1 Introduction

Advances in population-based evolutionary algorithms have introduced new tools for optimizing and controlling complex systems. This report considers a new algorithm that falls within the area of mathematical population genetics and interacting particle systems (Del Moral 2004): the genealogical decision trees. These particle filter systems (Doucet *et al.* 2001)(Arulampalam *et al.* 2002) belong to stochastic search models (Najim *et al.* 2004). Their robustness and flexibility make them particularly attractive for applications to optimization and control of complex industrial processes.

Using the MGDT Toolbox, optimization of an open-loop control sequence for following a given trajectory for a given plant, consists of the following steps:

1. Create an M-function describing the plant, a *process model*:

$$\begin{cases} \mathbf{x}_n = f_n\left(\mathbf{x}_{n-1}, \mathbf{u}_{n-1}\right) \\ \mathbf{y}_n = h_n\left(\mathbf{x}_n\right) \end{cases}.$$

2. Specify the *cost function*:

$$J_T\left(\mathbf{u}_0, ..., \mathbf{u}_{T-1}\right) = \sum_{n=0}^{T-1} \|\mathbf{u}_n\|_{\mathbf{A}_n}^2 + \sum_{n=0}^{T-1} \left\|\mathbf{y}_n - \mathbf{y}_n^{\text{ref}}\right\|_{\mathbf{B}_n}^2$$

   by setting the trajectory to follow, covariances $\mathbf{A}_n$ and $\mathbf{B}_n$, and the type of cost function (penalties on magnitudes or increments of control).

3. Set algorithm *tuning parameters*: number of particles N (population size), and sharpness of the distribution, β.

4. *Run* the optimization algorithm and *validate* the results.

5. *Repeat* simulations (Monte Carlo + tuning variations)

These steps, the background behind the approach, as well as an ample amount of case studies will be explained in the remainder of this report.

The text is organized as follows. We start with a brief introduction to Bayesian filters and particle filters. In Section 3, the Genealogical Decision Tree (GDT) algorithm for open-loop control optimization is described. The MATLAB MGDT-Toolbox implementing the algorithm is presented in Section 4. A number of case studies with different types of processes, models and trajectories are reported in Section 5, so as to illustrate the performance and viability of the approach. To conclude, Section 6 discusses extensions of the algorithm and possible future directions.

2 Particle filtering

In this section we take a brief look at the backgound and essential ideas of particle filtering. For a better coverage, see, e.g., (Stramer 2006)(Ye 2001)(Doucet *et al.* 2001)(Arulampalam *et al.* 2002).

2.1 Bayes' rule

Let us start from the axiom of conditional probability. A conditional probability is a "belief that A happens, when we assume that B is known". Axiom:

$$p\left(A|B\right) = \frac{p\left(A,B\right)}{p\left(B\right)}.$$

We have

$$p\left(A|B\right) \times p\left(B\right) = p\left(A,B\right),$$

and due to symmetry, it also holds that

$$p\left(B|A\right) \times p\left(A\right) = p\left(A,B\right).$$

Consequently, we have

$$p\left(A|B\right) \times p\left(B\right) = p\left(B|A\right) \times p\left(A\right).$$

This results in the famous *Bayes' rule*:

$$p\left(A|B\right) = \frac{p\left(B|A\right) \times P\left(A\right)}{p\left(B\right)}$$

The Bayes' rule can be interpreted as an algorithm where our belief on hypothesis A is updated with new evidence B:

- The *posterior* (after event) belief, $p\left(A|B\right)$, is obtained by multiplying the *a priori* (before event) belief, $p\left(A\right)$, by the probability that B occurs if A is true, $p\left(B|A\right)$.

The Bayes' rule is useful when one wants to obtain $p\left(A|B\right)$, and

1. it is difficult to obtain it directly, and
2. there is direct information on $p\left(B|A\right)$.

Let us write the Bayes' rule for updating a hypothesis H, when there's new evidence E and a context I:

$$p\left(H|E,I\right) = \frac{p\left(E|H,I\right) \times p\left(H|I\right)}{p\left(E|I\right)}$$

where

- $p(H|E,I)$: *posterior* probability, probability of hypothesis H, when evidence E is examined in the context I.
- $p(H|I)$: *a priori* probability, probability of hypothesis H in context I (before new evidence E)
- $p(E|H,I)$: *likelihood*, probability of evidence E when hypothesis H and context I are assumed to hold.
- $p(E|I)$: scaling factor.

2.2 Bayesian state estimation

Assume that we want to estimate the state variable $\mathbf{x}(n)$ of a dynamic system at instant n, when all observations (measurements) $\mathbf{y}(n)$ up to and including instant n are available. In the Bayesian framework, we want to determine the conditional probability

$$p(\mathbf{x}(n)|\mathbf{Y}(n))$$

where $\mathbf{Y}(n) = \{\mathbf{y}(1), \mathbf{y}(2), ..., \mathbf{y}(n)\}$ is the entire set of observations. In a similar way, let us define the state history $\mathbf{X}(n) = \{\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(n)\}$, when $\mathbf{x}(0)$ is the *a priori* information on the system state (before any observations).

Let us use the Bayes rule, to obtain

$$p\left(\overbrace{\mathbf{x}(n)}^{H}|\overbrace{\mathbf{Y}(n)}^{E,I}\right) = \frac{p\left(\overbrace{\mathbf{y}(n)}^{E}|\overbrace{\mathbf{x}(n)}^{H},\overbrace{\mathbf{Y}(n-1)}^{I}\right) \times p\left(\overbrace{\mathbf{x}(n)}^{H}|\overbrace{\mathbf{Y}(n-1)}^{I}\right)}{p\left(\overbrace{\mathbf{y}(n)}^{E}|\overbrace{\mathbf{Y}(n-1)}^{I}\right)}$$

Make the following assumptions:

- $\mathbf{y}(n)$ are independent of each others:

$$p(\mathbf{y}(j), \mathbf{y}(i)) = p(\mathbf{y}(i)) \times p(\mathbf{y}(j)),$$

- $\mathbf{y}(n)$ is independent of the dynamic process:

$$p(\mathbf{y}(n)|\mathbf{X}(n)) = p(\mathbf{y}(n)|\mathbf{x}(n)),$$

- the system can be described as a Markov process:

$$p(\mathbf{x}(n)|\mathbf{X}(n-1)) = p(\mathbf{x}(n)|\mathbf{x}(n-1)).$$

Now the Bayes' rule is greatly simplified:

$$\begin{aligned}\underbrace{p\left(\mathbf{x}\left(n\right)|\mathbf{Y}\left(n\right)\right)}_{\text{new estimate}} &= \frac{p\left(\mathbf{y}\left(n\right)|\mathbf{x}\left(n\right)\right)\times p\left(\mathbf{x}\left(n\right)|\mathbf{Y}\left(n-1\right)\right)}{p\left(\mathbf{y}\left(n\right)\right)} \\ &= \underbrace{C\left(n\right)}_{\text{scaling}}\times \underbrace{p\left(\mathbf{y}\left(n\right)|\mathbf{x}\left(n\right)\right)}_{\text{likelihood of the observation}} \times \underbrace{p\left(\mathbf{x}\left(n\right)|\mathbf{Y}\left(n-1\right)\right)}_{\text{estimate before new observation}} \quad (1)\end{aligned}$$

where

$$p\left(\mathbf{x}\left(n\right)|\mathbf{Y}\left(n-1\right)\right)=\int \underbrace{p\left(\mathbf{x}\left(n\right)|\mathbf{x}\left(n-1\right)\right)}_{\text{system dynamics}}\times \underbrace{p\left(\mathbf{x}\left(n-1\right)|\mathbf{Y}\left(n-1\right)\right)}_{\text{old estimate}} d\mathbf{x}\left(n-1\right). \quad (2)$$

This is a recursive procedure to update $p\left(\mathbf{x}\left(n\right)|\mathbf{Y}\left(n\right)\right)$.

With the following initial information:

i) a model of the observation

$$p\left(\mathbf{y}\left(n\right)|\mathbf{x}\left(n\right)\right), \text{ i.e., } \mathbf{y}\left(n\right)=h_n\left(\mathbf{x}\left(n\right),\mathbf{e}\left(n\right)\right),$$

ii) a model of the system

$$p\left(\mathbf{x}\left(n\right)|\mathbf{x}\left(n-1\right)\right), \text{ i.e., } \mathbf{x}\left(n\right)=f_{n-1}\left(\mathbf{x}\left(n-1\right),\mathbf{v}\left(n\right)\right),$$

and

iii) *a priori* model

$$p\left(\mathbf{x}\left(0\right)\right) \text{ ,i.e., } \mathbf{x}\left(0\right)\sim p,$$

the desired probability can be determined in two stages:

prediction (2) and update (1).

In general, the three models i)–iii) are complex, perhaps non-analytic, so that the calculations are not easy. However, in the case of Gaussian distributions (densities) and linear models, the equations reduce to the well known *Kalman filter*. In a more general context, an alternative for solving the equations is provided by the so called *particle filtering* techniques.

2.3 Particle filters

The equations (1)–(2) can be solved using particle filters, where the posterior distributions (densities) $p(\mathbf{x}(n)|\mathbf{Y}(n))$ are described using a set of samples, a particle cloud.

Prediction: The particles evolve in the state-space according to the system model:

$$p(\mathbf{x}(n)|\mathbf{x}(n-1)) = \int \underbrace{\delta(\mathbf{x}(n) - f_{n-1}(\mathbf{x}(n-1), \mathbf{v}(n-1)))}_{\text{delta-function (a point)}} p(\mathbf{v}(n-1))\, d\mathbf{v}(n-1).$$

When the state $\mathbf{x}(n-1)$ of each particle is known and the noise $\mathbf{v}(n-1)$ has realized, we are left with a set of points (particles) which have evolved in the state-space according to the statistical properties of the function f_{n-1} and noise $\mathbf{v}(n-1)$. In a practical algorithm, each particle $\mathbf{x}^i(n-1)$, $i = 1, 2, ..., N$, is put through a system model:

$$\overline{\mathbf{x}}^i(n) \sim p\left(\mathbf{x}(n)|\mathbf{x}^i(n-1)\right), \text{ i.e., } \overline{\mathbf{x}}^i(n) = f_{n-1}\left(\mathbf{x}^i(n-1), \mathbf{v}^i(n-1)\right)$$

where $\mathbf{v}^i(n-1)$ are generated from the distribution assumed for the noise (system model).

Update: When the observation (measurement) $\mathbf{y}(n)$ becomes available, the conditional distribution $p(\mathbf{y}(n)|\mathbf{x}(n))$ can be evaluated according to the observation model:

$$p(\mathbf{y}(n)|\mathbf{x}(n)) = \int \underbrace{\delta(\mathbf{y}(n) - h_n(\mathbf{x}(n), \mathbf{e}(n)))}_{\text{delta-function (a point)}} p(\mathbf{e}(n))\, d\mathbf{e}(n).$$

In a practical implementation, an output for the i'th particle can be generated based on the state $\overline{\mathbf{x}}^i(n)$ and the observation model

$$\overline{\mathbf{y}}^i(n) \sim p\left(\mathbf{y}(n)|\overline{\mathbf{x}}^i(n)\right), \text{ i.e., } \overline{\mathbf{y}}^i(n) = h_n\left(\overline{\mathbf{x}}^i(n), \mathbf{e}^i(n)\right)$$

where $\mathbf{e}^i(n)$ are generated according to the observation model. What is of interest, is the likelihood probability of the observation for the particle in question: $p\left(\mathbf{y}(n)|\overline{\mathbf{x}}^i(n)\right)$. If the noise $\mathbf{e}(n)$ obeys a density $p_{e(n)}(e)$, then the following likelihood probability is obtained

$$p\left(\mathbf{y}(n)|\overline{\mathbf{x}}^i(n)\right) = p_{e(n)}\left(\mathbf{y}(n) - \overline{\mathbf{y}}^i(n)\right).$$

Let us use this as a weighing factor

$$w^i(n) \propto p\left(\mathbf{y}(n)|\overline{\mathbf{x}}^i(n)\right), \text{ where } \sum_{i=1}^{N} w^i(n) = 1.$$

The particles $\mathbf{x}^i(n-1)$ represent the distribution at $n-1$. The particle cloud is resampled using the weighting $w^i(n)$ from the Bayes' rule. In a practical mechanization of the approach, we select N times from the set $\overline{\mathbf{x}}^i(n)$, $i = \{1, 2, ..., N\}$, so that the probability for each i to be selected is $w^i(n)$. The selected particles make up the particle cloud at the next sampling instant.

Let us summarize the steps at instant n for Bayesian state estimation based on particle filtering:

Algorithm 1 Bayesian state estimation based on particle filtering:
1. For each particle (with state $\mathbf{x}^i(n-1)$), the compute a *prediction* ($\overline{\mathbf{x}}^i(n)$).
2. *Observe* $\mathbf{y}(n)$ from the system.
3. For each particle, evaluate the *likelihood* of the observation to occur, $p\left(\mathbf{y}(n) | \overline{\mathbf{x}}^i(n)\right)$.
4. *Resample* the cloud of particles using the normalized likelihood as a weighting factor. Resampling results in a new set of particles $\mathbf{x}^i(n)$, $i = 1, 2, ..., N$.
5. Set $n \rightarrow n+1$ and return to step 1.

2.3.1 Kalman filter

In the Kalman filter, the probability models are assumed as follows:

iii) according to the *a priori* model, the state $\mathbf{x}(0)$ obeys normal distribution,

$$\mathbf{x}(0) \sim \mathcal{N}\left(\widehat{\mathbf{x}}(0), \mathbf{P}(0)\right)$$

ii) the system is described as a linear (Markov) state-space model,

$$\mathbf{x}(n+1) = \mathbf{A}(n)\mathbf{x}(n) + \mathbf{B}(n)\mathbf{u}(n) + \mathbf{v}(n)$$

where the process noise is normally distributed (white zero-mean noise)

$$\mathbf{v}(n) \sim \mathcal{N}(\mathbf{0}, \mathbf{V}(n))$$

Notice, that the system control $\mathbf{B}(n)\mathbf{u}(n)$ is completely deterministic.

i) the measurement has the form:

$$\mathbf{y}(n) = \mathbf{C}(n)\mathbf{x}(n) + \mathbf{e}(n)$$

where the measurement noise is normally distributed (white zero-mean noise):

$$\mathbf{e}(n) \sim \mathcal{N}(\mathbf{0}, \mathbf{Y}(n)).$$

For the resulting algorithm, see, e.g., (Ikonen and Najim 2002).

3 Genealogical Decision Trees (GDT)

In the Genealogical Decision Tree (GDT) approach (Del Moral 2004) (Ikonen *et al.* 2004) (Najim *et al.* 2006), the open-loop control problem is solved using particle filtering techniques, by interpreting the noise in the stochastic filtering problem as the control input to a deterministic process, and the system observation as the reference trajectory. The essential idea then consists of associating Gaussian distributions to both the norms of the control actions and the tracking errors. The Bayesian prediction/evaluation steps are solved by sampling/resampling point approximations of the associated distributions. The resulting stochastic search model (optimization of an open loop control sequence for a deterministic state space model) can be interpreted as a simple genetic (evolutionary) particle population model, and has a natural birth and death interpretation. Converge in probability can be shown (Najim *et al.* 2006).

In the following, the control problem is formulated, followed by a description of the optimization and control algorithm under consideration.

3.1 Problem formulation

Consider any non-linear time-varying dynamic system, described by the following state-space equations:

$$\begin{cases} \mathbf{X}_n = F_n\left(\mathbf{X}_{n-1}, \mathbf{U}_n\right) \\ \mathbf{Y}_n = h_n\left(\mathbf{X}_n\right) \end{cases}, \tag{3}$$

where $\mathbf{X}_n = [X_{n,1}, X_{n,2}, ..., X_{n,S}]^T \in \mathcal{R}^S$, $\mathbf{U}_n = [U_{n,1}, U_{n,2}, ..., U_{n,P}]^T \in \mathcal{R}^P$, $\mathbf{Y}_n = [Y_{n,1}, Y_{n,2}, ..., Y_{n,Q}]^T \in \mathcal{R}^Q$. Index $n = 1, 2, ..., T$ represents the sampling instant. $\mathbf{X}_0$ represents the fixed initial state at instant $n = 0$. Let $\mathbf{A}_n$ and $\mathbf{B}_n$ be symmetric and semi-definite positive covariance matrices. The control objective in a finite horizon of length T is given by

$$J_T\left(\mathbf{U}_1, ..., \mathbf{U}_T\right) = \sum_{n=1}^{T} \left\|\mathbf{U}_n\right\|^2_{\mathbf{A}_n} + \sum_{n=1}^{T} \left\|\mathbf{Y}_n - \mathbf{Y}_n^{\mathsf{ref}}\right\|^2_{\mathbf{B}_n} \tag{4}$$

where $\mathbf{Y}_n^{\mathsf{ref}} \in \mathcal{R}^Q$ represents the reference trajectory (desired target outputs), and $\left\|\mathbf{U}\right\|^2_{\mathbf{A}} = \mathbf{U}^T\mathbf{A}^{-1}\mathbf{U}$.

Our objective is to find the sequence of control actions that will minimize this control objective for open-loop control. The essential idea is to associate Gaussian distributions to the norms of the control actions and the tracking errors, i.e.,

$$\left\|\mathbf{U}_n\right\|^2_{\mathbf{A}_n} \longrightarrow \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\beta}{2}\left\|\mathbf{U}_n\right\|^2_{\mathbf{A}_n}\right) \tag{5}$$

$$\left\|\mathbf{Y}_n - \mathbf{Y}_n^{\mathsf{ref}}\right\|^2_{\mathbf{B}_n} \longrightarrow \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\beta}{2}\left\|\mathbf{Y}_n - \mathbf{Y}_n^{\mathsf{ref}}\right\|^2_{\mathbf{B}_n}\right). \tag{6}$$

The parameter β is similar to the inverse temperature in simulated annealing optimization algorithms. For small values a flat probability distribution results, for large values of β the distribution will have the shape of a hair pin. Other than Gaussian distributions can also be considered.

3.2 Optimization of the control sequence

Let us consider the following scheme. At instant n, generate N independent and identically distributed normal vectors $\mathbf{U}_n^i \sim \mathcal{N}(\mathbf{0}, \mathbf{A}_n)$, $i = 1, 2, ..., N$:

$$\mathbf{U}_n^1, \mathbf{U}_n^2, ..., \mathbf{U}_n^N.$$

Using the model (F_n, h_n) we evaluate that these control values lead to N outputs:

$$\left\{ \begin{array}{c} \mathbf{X}_n^i = F_n\left(\widehat{\mathbf{X}}_{n-1}^i, \mathbf{U}_n^i\right) \\ \mathbf{Y}_n^i = h_n\left(\mathbf{X}_n^i\right) \end{array} \right.$$

for $i = 1, 2, ..., N$. At $n = 1$, the initial states $\widehat{\mathbf{X}}_0^i$ are given by $\mathbf{X}_0$.

In order to simplify the notations, let us introduce the following term

$$p_n^i = \frac{\exp\left(-\frac{\beta}{2}\left\|\mathbf{Y}_n^{\mathsf{ref}} - \mathbf{Y}_n^i\right\|_{\mathbf{B}_n}^2\right)}{\sum\limits_{j=1}^{N} \exp\left(-\frac{\beta}{2}\left\|\mathbf{Y}_n^{\mathsf{ref}} - \mathbf{Y}_n^j\right\|_{\mathbf{B}_n}^2\right)}. \tag{7}$$

We have $\sum_{i=1}^N p_n^i = 1$, and p_n can be interpreted as a probability measure. Let us then generate N independent and identically distributed random vectors $\widehat{\mathbf{U}}_n^1, \widehat{\mathbf{U}}_n^2, ..., \widehat{\mathbf{U}}_n^N$ according to the distribution

$$p_n(\mathbf{u}) = \sum_{i=1}^{N} p_n^i \delta_{\mathbf{U}_n^i},$$

where $\delta_{\mathbf{u}}$ is the Dirac measure at the control value $\mathbf{u} \in \mathcal{R}^P$.

In other words, for each $j = 1, 2, ..., N$, each random control $\widehat{\mathbf{U}}_n^j$ takes the value $\mathbf{U}_n^i$ with probability equal to p_n^i. This can be seen as a resampling procedure, where the probability for the survival of $\mathbf{U}_n^i$ depends on the performance of the associated $\mathbf{Y}_n^i$.

The implementation of these control actions leads to

$$\widehat{\mathbf{X}}_n^j = F_n\left(\widehat{\mathbf{X}}_{n-1}^i, \mathbf{U}_n^i\right) \Longrightarrow \widehat{\mathbf{Y}}_n^j = h_n\left(\widehat{\mathbf{X}}_n^j\right)$$

for $j = 1, 2, ..., N$. The procedure is repeated for all $n = 2, 3, ..., T$.

3.3 Algorithm

The algorithm can be now stated as a modification of the nonlinear filtering algorithm (compare with Algorithm 1):

Algorithm 2 Optimization of open-loop control sequence.
1. For each particle (with state $\mathbf{x}^i(n-1)$ and control $\mathbf{u}^i(n-1)$), compute a *prediction* ($\overline{\mathbf{x}}^i(n)$).
2. '*Observe*' $\mathbf{y}^{\text{ref}}(n)$ for the system.
3. For each particle, evaluate the *likelihood* of the reference target to occur.
4. *Resample* the cloud of particles using the normalized likelihood as a weighting factor. Resampling results in a new set of particles $\mathbf{x}^i(n)$ $i = 1, 2, ..., N$.
5. Set $n \rightarrow n+1$ and return to step 1.

A mechanization of the procedure for optimizing the open-loop control sequence is given in what follows.

Let us assume that the plant model is given by

$$\begin{cases} \mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n) \\ \mathbf{y}_n = h(\mathbf{x}_n) \end{cases} . \tag{8}$$

with initial condition $\mathbf{x}_0$. Note that the system equation is given by $\mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n)$, instead of $\mathbf{x}_n = f(\mathbf{x}_{n-1}, \mathbf{u}_n)$. This formulation is far more common in process modelling, as it ensures causality (the control at n can only have an effect on state at instant $n+1$. This formulation makes it easier to use 'standard' process models available in the literature. For simplicity, a time-invariant system is assumed. The associated cost function is given by

$$J_T(\mathbf{u}_0, ..., \mathbf{u}_{T-1}) = \sum_{n=0}^{T-1} \|\mathbf{u}_n\|^2_{\mathbf{A}_n} + \sum_{n=0}^{T-1} \left\|\mathbf{y}_n - \mathbf{y}_n^{\text{ref}}\right\|^2_{\mathbf{B}_n} \tag{9}$$

In this formulation, $\mathbf{y}_0$ is determined by the plant initial state $\mathbf{x}_0$, and only the first term on the right hand side depends on the control actions. For $n = 1, 2, ..., T-1$, both terms depend on the control sequence. Note that the reference targets are now set for $n = 0, 1, ..., T-1$.

The following pseudo-code implements the algorithm:

`for` $n = 1 : T$

`for` $i = 1 : N$

`if` $n == 1$, Initialize $\mathbf{x}^i = \mathbf{x}_0$, $\mathbf{y}^i = h(\mathbf{x}_0)$ and $J_T^i = 0$; `end`

Generate random $\mathbf{u}^i \sim N(\mathbf{0}, \mathbf{A}_n)$.

Store action to a list $\mathbf{v}_n^i = \mathbf{u}^i$.

Evaluate $J_Y = \left\|\mathbf{y}_n^{\text{ref}} - \mathbf{y}^i\right\|_{\mathbf{B}_n}^2$, $J_U = \left\|\mathbf{u}^i\right\|_{\mathbf{A}_n}^2$ and $J_T^i = J_T^i + J_Y + J_U$.

Set weight $p_{\text{un}}^i = \exp\left(\frac{-\beta}{2} J_Y\right)$.

`end`

`For` all $i = 1 : N$: Compute resampling probabilities: $p^i = p_{\text{un}}^i / \sum_{i=1}^N p_{\text{un}}^i$. `end`

Resample: `For` all $i = 1 : N$: Select $\widehat{I}_i = k$ such that $\Pr(k = j) = p^j$. `end`

`For` all $i \in \widehat{I}$: Compute model for next n: $\mathbf{x}^i = f\left(\mathbf{x}^i, \mathbf{u}^i\right)$; $\mathbf{y}^i = h\left(\mathbf{x}^i\right)$. `end`

Death and birth. `For` all $j = 1 : N$: Replace $\mathbf{x}^j$, $\mathbf{y}^j$, $\mathbf{v}_n^j$ and J_T^j by $\mathbf{x}^{\widehat{I}_j}$, $\mathbf{y}^{\widehat{I}_j}$, $\mathbf{v}_n^{\widehat{I}_j}$ and $J_T^{\widehat{I}_j}$. `end`

`end`

Find $i^* = \arg\min_i J_T^i$. The solution for the optimal control sequence is $\mathbf{v}^{i^*}$.

The code proceeds one sample at a time from the beginning to the end of the trajectory, $n = 1, 2, \ldots, T$[1]. At instant n, a random input $\mathbf{u}^i$ is generated for each particle at the population. Each particle i is evaluated based on its performance at n (deviation of $\mathbf{y}^i$ from output trajectory $\mathbf{y}_n^{\text{ref}}$), and a corresponding resampling weight is set. The particle population is resampled, and the particles that survived are simulated one sample forward. The loop is then continued at $n \to n + 1$.

The scalars J_T^i collect cumulatively the cost (4) associated with each particle, so that at $n = T$ these correspond to J_T in equation (4). The postponing of the model evaluations after resampling can significantly reduce the computational load of the algorithm (here advantage is taken of the fact that control at n will affect the plant output at $n + 1$ the earliest.)

[1] In MATLAB the vector indexes run starting from 1, therefore n is shifted by one in the code. Indexes $n = 1 : T$ correspond to $n = 0, 1, \ldots, T - 1$ in (9).

4 MGDT-Toolbox

This section explains how to use the algorithm package, and how to get started in solving a particular problem. The next section (Section 5) reports an extensive list of case studies. To make it short, the following steps are required:

---

1. Create an M-function describing the plant, a *process model*. It has to return the next output and state, given the current state and control (two output arguments, two input arguments).
2. Specify the *cost function*: set covariances $\mathbf{A}_n$ and $\mathbf{B}_n$, and select type of cost function (penalties on magnitudes or increments of control)
3. Set algorithm *tuning parameters*: number of particles N, and sharpness of the distribution β.
4. *Run* the optimization and *validate* the results.
5. *Repeat* simulations (Monte Carlo + tuning variations)

The MGDT-Toolbox was written using MATLAB 6 (Release 12) on a Windows XP platform.

4.1 Contents and variables

The MATLAB MGDT Toolbox contains the following files:

- `GDTopt.m` – contains essential parts of the algorithm
- `GDTsort.m` – extracts optimal solution from the GDT-structure
- `GDTplot.m` – plots information about the algorithm and its results
- `GDTODESampler.m` – sampling interface for ODE plant descritions
- `Urand.m` – generates a random vector distributed according to a given covariance
- `UGener.m` – an alternative resampling procedure (not used)

These M-files can be downloaded from `http://cc.oulu.fi/~iko/MGDT.htm`, along with a few sample files:

- `typicalplant.m` and `typicalrun.m` provide an example of a simple optimization problem.
- `typicaldiff.m`, `typicaldiffsampled.m` and `typicaldiffrun.m` provide an example using an ODE plant description.

To install the MGDT-Toolbox, create a new directory (e.g., 'c:\MGDT'), copy the above files to this directory, and add the diretory to Matlab's current search, .e.g., with the command `path(path,'c:\MGDT')`.

The `GDTopt`-file implements the essential parts of the algorithm. The `GDTopt` M-function needs the following input-arguments:

- `Yref` – a $T \times Q$ matrix containing the desired output trajectory
- `A_n`, `B_n` – covariance matrices $\mathbf{A}_n$ and $\mathbf{B}_n$ (matrices of size $P \times P$ and $Q \times Q$)
- `N`, `bet` – number of particles N and tuning factor β (scalars)
- `plantfunstr` – name of the M-file that simulates the plant under optimization
- `X0`, `Y0` – initial state and output of the plant (column vectors of length S and Q)
- `DELTAU` – a flag indicating optimization of a sequence of control increments (`DELTAU=1`), or absolute controls (`DELTAU=0`)
- `P1`, `P2`, `P3`, ..., `P6` – additional arguments to be passed to `plantfunstr`

The plant models (defined by `plantfunstr`, `P1`, `P2`, `P3`, ..., `P6`) are discussed in the next subsection.

As a result of the optimization, the M-file `GDTopt` returns a `GDT`-structure. This is a vector-structure, where each particle has its own structure, i.e., `GDT(1)` describes the first particle, `GDT(2)` the second, etc., up to `GDT(N)`. `GDT(i)` is the structure for the i'th particle, with the following fields:

- `U_hist` – past plant control sequence, a matrix of size $T \times P$. Note, that for incremental control (`DELTAU=1`) this contains the control *increments*, the absolute control values can be obtained by `cumsum(U_hist)+U0`, where `U0` would define an initial value for control (zero by default).
- `X_hist` – past plant state sequence, a matrix of size $T \times S$
- `Y_hist` – past plant output sequence, a matrix of size $T \times Q$
- `J_hist` – past cost history, a matrix of size $T \times 3$. The first column stores the total costs (cumulative), the second and third columsn store the deviation and control costs (instant).
- `parent_hist` – past indexes to parent particles, a T-vector
- `survived` – number of different resampled particles (number of particles that survived in selection at iteration n), a T-vector
- `X` – current plant state, S-vector
- `Y` – current plant output, Q-vector

- J_T – current total cost, a scalar
- Jy – current deviation cost, a scalar
- Ju – current control cost, a scalar
- parent – parent of current particle, a scalar
- p_un – unnormalized resampling weight of particle, a scalar
- Yref, A_n, B_n, bet, N, plantfunstr, X0, Y0, and DELTAU, as described already.

4.2 Plant models

The system model (8) is coded in an M-function whos name is indicated by plantfunstr. This M-file must have the following syntax:

$$[\texttt{Yn1},\ \texttt{Xn1}]\ =\ \texttt{fun}(\texttt{Xn},\ \texttt{Un},\ \texttt{P1},\ \texttt{P2},\ \texttt{P3},\ ...)$$

where

- Xn denotes the system state at instant n, $\mathbf{x}_n$;
- Un denotes the system input at instant n, $\mathbf{u}_n$
  - Note, that if DELTAU=1, then $\mathbf{u}_n = \sum \Delta\mathbf{u}_n$ will be automatically generated when evaluating the plantfunstr, while the sequence of $\Delta\mathbf{u}_n$ ($n = 0$, 1,..., $T-1$) is optimized.
- Yn1 denotes the system output at instant $n+1$, $\mathbf{y}_{n+1}$;
- Xn1 denotes the system state at instant $n+1$, $\mathbf{x}_{n+1}$;

The arguments P1, P2, P3, ... are optional, and not used anywhere else in the GDTopt M-file (except as arguments for plantfunstr). These may define a sampling time, a measurement matrix, an initial/nominal control value in optimization of controls, etc., to make it more convenient to examine different values/optimization settings.

Example 3 A typical plantfunstr M-file would look as follows.

```
function [yn1,xn1] = typicalplant(xn,un,Ts,u0)
% TYPICALPLANT
%    [yn1,xn1] = typicalplant(xn,un,Ts,u0)

% define a TITO system
sysc = tf({2,[-10 1];0,1},{[10 1],[100 20 1];1,[2 1]}); % plant
sysd = c2d(sysc,Ts);     % sampled system
```

```
[A,B,C,D]=ssdata(sysd)     %sampled state-space system
un = un+u0;    % add controls to initial/nominal controls
xn1=A*xn+B*un;    % state equations
yn1=C*xn1;    % measurement equations
```

This M-file implements a 2×2 linear system, but there are no constraints to use any other type of models, as long as the model is of form (8). The additional inputs (`P1`, `P2`) to this model are the sampling time `Ts` and initial control `u0`.

The `plantfunstr` M-file is typically evaluated very many times during optimization, so that efforts towards fast implementation will most likely pay off.

As a matter of coding style, it can be useful to attach reasonable initial values for all parameters into the plant description. The following code (if added in `plantfunstr` after the lines for function definition and help descriptions) gives an example.

Example 4 Code for initial values for plant and arguments

```
% Default arguments:
% proper initial values
if nargin==0, yn1 = zeros(2,1); xn1 = zeros(4,1); return; end
% reasonable additional argument values
if nargin<4, Ts = 1; u0 = zeros(2,1);
elseif nargin<3, Ts = 1;
end
```

When no input arguments are given to the M-file, initial values are returned. When third or fourth input arguments are missing, default values are used.

4.3 Optimization algorithm

This subsection explains how to run the algorithm and examine the results. Guidelines for initializing and tuning the algorithm parameters are also discussed.

4.3.1 Initializing parameter values

The search algorithm has five parameters to be specified: `DELTAU`, $\mathbf{A}_n$, $\mathbf{B}_n$, N and β.

`DELTAU` is a fundamental parameter:

- With `DELTAU=0` a sequence of control actions is searched for. This sequence attempts to minimize the cost function (9) consisting of weighted sums of squared deviations from desired trajectory and the weighted sums of squared magnitudes of the control actions required.
- With `DELTAU=1` a sequence of *increments* of control actions is searched for. The sequence should minimize the cost function (9) where the costs consists of deviations from desired trajectory and magnitudes of *changes* in control actions required.

- In Example 3, an initial/nominal control was defined. With this type of coding an initial control value can be given for an incremental control sequence, or the zero-level of an "absolute" control sequence can be justified. In both cases, the control sequence can be optimized around a predetermined operating point. Notice, that this has a great influence in the minimization of the costs.

The parameters `A_n` and `B_n` ($\mathbf{A}_n$ and $\mathbf{B}_n$) specify the cost function to be minimized. There's no unique way to define these parameters, unless the cost function fully includes all the control design specifications. As this is rarely the case in practice, a tentative scheme to find reasonable initial values/tuning is suggested in what follows:

- Select `A_n` as a diagonal matrix, where the elements at the diagonal are squares of a largest 'tolerable' output error for the output variable in question.
- Select `B_n` as a diagonal matrix, where the elements at the diagonal are squares of a largest 'tolerable' control action for the manipulated variable in question (with `DELTAU=0`). In incremental control, (`DELTAU=1`), specify a largest 'tolerable' control increment instead.
- Select `bet=1` (β) and `N=100` (N). This is a purely heuristic choice, supported by the arguments discussed in the next subsection.

Non-diagonal $\mathbf{A}_n$ and $\mathbf{B}_n$ are discussed in Section 6.

4.3.2 Running the simulation

The GDT optimization (simulation) is invoked by the following command:

$$\texttt{GDT} \;=\; \texttt{GDTopt}(\texttt{Yref}, \texttt{A\_n}, \texttt{B\_n}, \texttt{N}, \texttt{bet}, \texttt{plantfunstr}, \texttt{X0}, \texttt{Y0}, \texttt{deltaU})$$

This will run the optimization routine (generation of random controls, evaluations of system model, sampling of promising solutions, etc.) for $n = 0, 1, ..., T-1$. Any additional arguments (`P1`, `P2`, ...) to be passed to `plantfunstr` are added to the end of input argument list:

$$\texttt{GDT} \;=\; \texttt{GDTopt}(\texttt{Yref}, \texttt{A\_n}, \texttt{B\_n}, \texttt{N}, \texttt{bet}, \texttt{plantfunstr}, \texttt{X0}, \texttt{Y0}, \texttt{deltaU}, \texttt{P1}, \texttt{P2}, ...)$$

After each iteration n (or each three seconds if single iterations are faster) a plot of the state of the optimization is shown (see `GDTplot` below). The results of the optimization are returned in the output argument `GDT` (see Section 4.1).

Example 5 A typical simulation run would look as follows.

```
% Define plant M-file name
plantfunstr = 'typicalplant'
```

```
% reference trajectory is consequtive steps in both varibles
Yref = [0*ones(10,1) 0*ones(10,1);...
0*ones(10,1) ones(10,1);...
ones(10,1) ones(10,1)];
% algorithm parameters
A_n = [0.5^2 0;0 0.1^2];
B_n = [0.1^2 0;0 0.5^2];
N = 500;
bet = 1;
deltaU=1;
% initial values
[Y0,X0] = typicalplant;
% start optimization
GDT = GDTopt(Yref,A_n,B_n,N,bet,plantfunstr,X0,Y0,deltaU);
```

In the above, we take that in the first output variable an error of 0.1 is 'tolerable', in the second output a much larger error can be accepted (0.5),

$$\mathbf{B}_n = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.25 \end{bmatrix}.$$

These are to be in relation with the 'tolerable' control increments, judeged to be around 0.5 for the first control variable, while smoother manipulations are desired for the second input (0.1),

$$\mathbf{A}_n = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.01 \end{bmatrix}.$$

4.3.3 Plotting the results

The outcomes of the simulation can be examined using the M-file `GDTplot.m`. It plots the trajectories related to the best particle (out of the N particles), i.e., the one which minimizes the total costs J_T, eq. (9). The syntax is

`GDTplot(GDT)`

where `GDT` is the structure obtained as an output argument from `GDTopt`. The figure window title indicates the name of the `plantfunstr`. When invoked during optimization (from within `GDTopt`), it also gives an estimate of the time left for completion of simulations. The figure contains the following information for the optimal solution, i.e., the one among all N solutions that minimizes J_n:

- The *leftmost column* plots the system state trajectories $\mathbf{x}_n$, as a function of samples, from 0 to $n-1$. The first state is shown at the top, the other states follow consequtively. The x-axis label of the bottom plot indicates the value of n.

- The *second column* shows the control trajectories $\mathbf{u}_n$ (from 0 to $n-1$). The first input is shown at the top, the x-axis label shows the corresponding diagonal element of $\mathbf{A}_n$. The other inputs are shown below. In the case of incremental optimization, $\mathbf{u}_n$ are shown ($\sum_{i=0}^{n-1} \Delta\mathbf{u}_i$). Note, however, that any possible initial/nominal values passed to `plantfunstr` are not shown.
- The *third column* shows the desired (dotted line) and simulated output trajectories for each of the plant outputs. The complete reference trajectory is shown ($\mathbf{y}_0^{\text{ref}}$, $\mathbf{y}_1^{\text{ref}}$, ..., $\mathbf{y}_{T-1}^{\text{ref}}$), the simulated trajectory is shown up to n. The x-axis labels show the corresponding diagonal elements of $\mathbf{B}_n$.
- The *upper plot* in the *rightmost column* shows the evolution of costs. The total costs J_n are composed of the deviation term $\sum J_Y$ (red dotted line) and control term $\sum J_U$ (blue). The title text shows $J_n\left(\Delta U\right)$ for optimization of incremental costs, $J\left(U\right)$ otherwise; the total costs up to n are shown.
- The *lower plot* in the *right column* gives information on the population. The plot shows the survival percentage (upper, blue curve), and the number of different solutions in the population backwards in time (lower, red curve). If the `GDT` structure contains only one particle, the second curve is not shown. The y-axis scale is logarithmic, the y-axis is scaled such that the lower value corresponds to one sample ($\frac{100}{N}\%$), the upper to all samples (100%). The title text shows the average survival percentage, indicated by ${}^{n+1}/_{n}$. The x-axis label shows the N and β parameters.

An example of the plot produced for the `typicalrun`-example follows.

Example 6 Figure 1 illustrates the simulation, as plotted by the command `GDTplot(GDT)`, where `GDT` is obtained as in Example 5. Out of the four states (leftmost column), the first and fourth correspond to scaled output measurements (third column), according to the linear plant description in `typicalplant.m`. The target trajectories are shown together with plant outputs. In the design specifications, $\mathbf{B}_n$, a tight control for the first output was desired, while much larger deviations from the desired trajectory were accepted for the second output. This can be clearly observed from the output trajectories (the first output follows the trajectory much closer than the second). For the magnitudes in control increments, the specifications in $\mathbf{A}_n$ insisted on a relatively smooth control for the second input, while larger increments were allowed for the first. Again, this is clearly the case in the illustrated simulation. The diagonal components of $\mathbf{A}_n$ and $\mathbf{B}_n$ are shown in the plot labels, as well as other parameters N and β. The `DELTAU=1` can be deduced from the cost title: $J_n(\Delta U)$. The optimal control sequences are shown in the second column.

The rightmost row (Fig. 1) illustrates the development of the total costs. Since the plant did not achieve its trajectory at T, and the control actions are not constants (zero increments), the costs do not seem to tend towards a fixed value. The lower plot in the righmost column shows the number of different solutions in the current

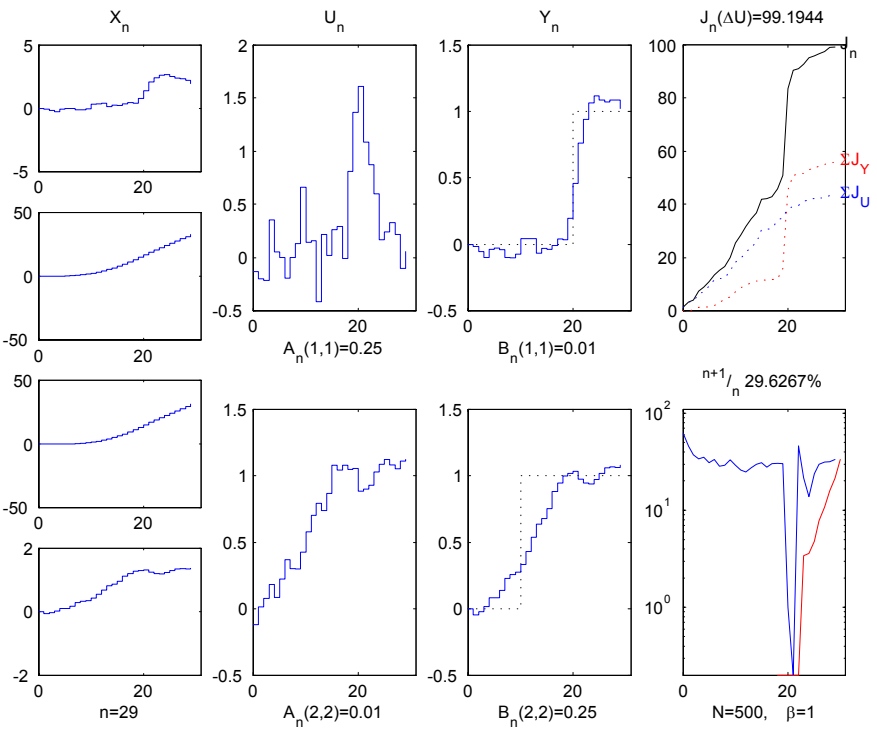

Figure 1: `GDTplot` shows the state of the optimization.

population, backwards in time. Notice, that the y-axis scale is logarithmic. We can see, for example, that the control sequences in the current population contains apprx. 30% different control sequences. However, if we examine the control sequences from first to 22'nd, only 0.2% of the sequences are different (i.e., each particle in the population contains the same sequence $\mathbf{u}_0, ..., \mathbf{u}_{12}$). Most of the differences within the sequences in the population are at the later stages, in particular at instants $24 - 29$, as can be expected.

4.3.4 Tuning algorithm parameters

Finding a proper set of GDT algorithm parameters can be tedious. However, if the guidelines given above can be followed, a good starting point is often obtained. Recall also that the search is stochastic, so that some kind of Monte-Carlo approach is usually recommendable (unless N is large).

Selecting $\beta = 1$ is a reasonable starting point since the initial tuning, suggested above, results in that the weighted norm on $\mathbf{y}_n - \mathbf{y}_n^{\text{ref}}$, (6), takes values less than one for all 'tolerable' deviations from the desired trajectory. The probability density of the surviving particles will have the shape shown in Fig. 2 as a function of the weighted norm $\left\|\mathbf{y}_n - \mathbf{y}_n^{\text{ref}}\right\|^2_{\mathbf{B}_n}$. For $\beta = 1$ (bold curve in Fig. 2) the resampling (selection) probability distribution will tend to focus on tolerable deviations, as roughly two out of three (67%) of the solutions will be within the range $[-1, 1]$ (indicated by dotted vertical lines in Fig. 2). However, the selectioning is based on the deviation at instant n only, cf. $\left\|\mathbf{y}_n^{\text{ref}} - \mathbf{y}_n^i\right\|^2_{\mathbf{B}_n}$ in equation (7). It may well be that a particular particle at instant n would be competitive in the long run, even if at instant n the deviation is large. Therefore it is reasonable to keep also some of these less appealing solutions in the population. If they turn out to be poor consistently, they will eventually die out.

Increasing β will narrow the resampling distribution, and the number of different solutions surviving in selection is diminished. The sharper the distribution, the likelier it is that only solutions with output deviations close to zero will survive. Therefore, if it appears that the solution is 'rough' (the output does not follow closely the target trajectory), increasing β may help. On the other hand, it may happen that the algorithm seems to fix a particular control in time too quickly because it runs out of alternatives in the population. This can be expected if plant dynamics have slow modes, or if the trajectory to follow is complex. Then decreasing β will increase the variety of alternatives sequences available for selection.

Increasing the number of particles will improve the quality of the solution (recall that with N approaching infinity, we are guaranteed to find the optimal solution..). In practice, available computing resources (memory/time) dictate the upper limit for N. In most cases, a major fraction of *computing time* is spent in evaluating the plant model. With a trajectory of length T and with N particles in the population, the maximum number of plant evaluations required is TN. Since the samples that die out (samples that are not selected in the resampling) need not be evaluated, the number of evaluations is typically much less than TN.

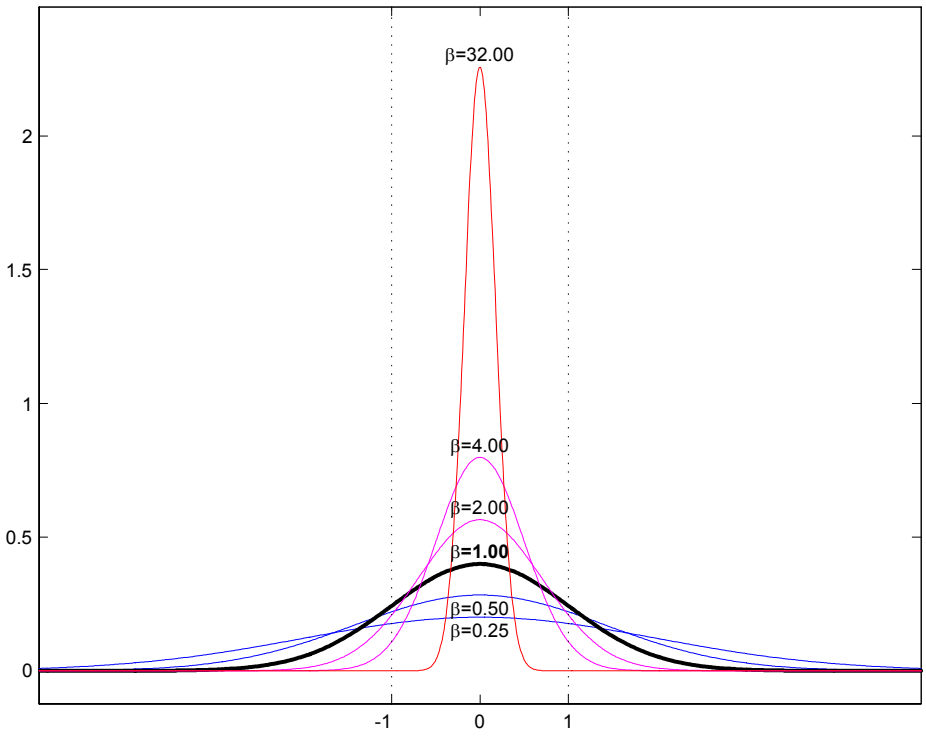

Figure 2: Normalized weight distribution $\exp\left(\frac{-\beta}{2}x\right)$.

If, however, each evaluation requires an iterative solution (such as a set of ODE, discussed in the next subsection), the underlying plant model will need to be evaluated many more times. The *memory* requirements are also mainly dictated by N and T, as well as by the system dimensions. Each particle stores its complete history (input, state and output trajectories), in addition to parameter values, costs, etc. With a standard office PC, a typical upper limit for reasonable population sizes, as set by memory requrements, is around some tens of thousands of particles. The time needed to solve the equations depends solely on the type of plant model. A 3×3 MIMO system described by seven ODE, with $T \approx 100$ and $N \approx 10000$, may require overnight computations, while the `typicalplant` is optimized in a few minutes.

To solve a problem, typically some tens of optimizations are needed (to find a proper set of parameters, to examine the effect of randomness in the solution, etc.). In Monte Carlo simulations, the syntax

$$\texttt{GDT\_new} \;=\; \texttt{GDTopt}(\texttt{GDT}, \texttt{P1}, \texttt{P2}, ...)$$

may be handy. It takes the algorithm parameters from a result of a previous optimization (structure `GDT`), and repeats the experiments with a new random seed.

4.4 Plant models given by ordinary differential equations

It is very common that dynamic plant descriptions are given by a set of ordinary differential equations (ODE). ODE can easily be used as plant models in the MGDT Toolbox. This requires, however, that an appropriate interface is built which simulates the equations in time (using zero order hold for control inputs at t, and MATLAB `ode23`, for example), and returns only the values of the state and output (sampled at $t + T_s$). An example follows.

Example 7 The differential equations of the system are given in `typicaldiff.m`.

```
function xdot = typicaldiff(t,x,u)
b_u = 1000;c_1 = 1.1e-10;c_2 = 0.8;c_3 = 1.5e-9;T_A = 20;
T_F = x(1); T_P = x(2);
T_Fdot = b_u*u - c_1*(T_F^4-T_P^4) - c_2*(T_F-T_A);
T_Pdot = c_3*(T_F^4-T_P^4);
xdot = [T_Fdot;T_Pdot];
```

These equations describe the dynamics of a rapid thermal processing plant, $\mathbf{x}' = f(\mathbf{x})$, see Section 5.2. The manipulated variable is u. The controlled variable is the second component in $\mathbf{x}$.

The M-file `GDTODESampler` implements the simulation/sampling procedure. Basically, it solves the system ordinary differential equations from time t to $t + T_s$ and returns the values of the state $\mathbf{x}$ and output $\mathbf{y} = \mathbf{C}\mathbf{x}$ at $t+T_s$. The syntax is as follows:

$$[\texttt{Yk1}, \texttt{Xk1}] \;=\; \texttt{GDTODESampler}(\texttt{Xk}, \texttt{Uk}, \texttt{diffeqstr}, \texttt{Ts}, \texttt{C})$$

where diffeqstr is the name of the M-file containing differential equations, Ts is the sampling time, and C is the measurement matrix. The diffeqstr-file should have the form: xdot = fun(t,x,u), see help ode23 for more details. The essential idea is that a new M-file can be specified and used as the planfunstr. An example illustrates this.

Example 8 In the previous example, we have defined the ode-file typicaldiff.m. We now define a new M-file which simulates and samples the ODE, and has the input-output arguments as required by any plantfunstr-file.

```
function [Yk1,Xk1]=typicaldiffsampled(Xk,Uk,Ts,U0,diffeqstr,C)
% TYPICALDIFFSAMPLED A pseudo-plant file
Uk = Uk+U0;
[Yk1,Xk1] = GDTODESampler(Xk,Uk,diffeqstr,Ts,C);
```

This provides the simulation/sampling interface for the ode-descrition. It can be seen as a plantfunstr with four additional arguments: Ts,U0,diffeqstr and C. These can be passed as P1, P2, P3 and P4 in the GDTopt command.

The script for optimizing the control sequence for the system (plant) described by ordinary differential equations the would look something like this

```
% plant
plantfunstr = 'typicaldiffsampling'
diffeqstr = 'typicaldiff'
Ts = 1.2/60; U0 = 0.22; C = [0 1]
% reference trajectory
Yref = [300*ones(50,1); [300:3:600]'; 600*ones(150,1)];
% algorithm parameters
A_n = 0.03^2; B_n = 20^2;
N = 500; bet = 1;
deltaU=1;
% initial values
X0 = [300 300]; Y0 = X0(2);
% start optimization
GDT = GDTopt(Yref,A_n,B_n,N,bet,plantfunstr,X0,Y0,deltaU,...
Ts,U0,diffeqstr,C);
```

If the system contains delays, it may be difficult to specify them using ode. For linear systems, Pade-approximation is often useful, perhaps this can be extended also for nonlinear systems. In the case of discrete-time sampled systems, delays are easier to handle, by expanding system state.

5 Case studies

In this section a number of case studies are reported, so as to illustrate the performance and viability of the approach. Many of the problems have been considered in earlier publications, including:

- a linear dynamic plant with nonlinear control manipulation, by Ikonen, Del Moral & Najim in IFAC AFNC Workshop 2004, Oulu, Finland (Ikonen *et al.* 2004);
- a RTP plant, by Ikonen, Najim & Del Moral in IFAC World Congress 2005, Prague, Czech (Ikonen *et al.* 2005);
- a single link manipulator by Ikonen, Najim & Del Moral in IFAC World Congress 2005, Prague, Czech (Ikonen *et al.* 2005);
- a 3×3 MIMO FBC plant with two SISO-PI feedback loops by Ikonen & Kovacs, to appear in Artificial Intelligence in Energy and Renewable Energy Systems, Edited by Kalogirou (Ikonen and Kovacs 2006);
- a 3×3 MIMO FBC plant described by seven ODE by Najim, Ikonen & Del Moral, to appear in Neural Computing & Applications (Najim *et al.* 2006);
- a two joint robot arm by Gomez-Ramirez, Najim & Ikonen, a submitted manuscipt.

In this report, two new additional examples are covered:

- a three-component batch CSTR (ABC-plant), and
- a continuous CSTR plant with van der Vusse reactions.

5.1 ABC-plant

Optimal operation of batch processes commonly involves following a pre-optimized batch trajectory, such as the temperature control in an exothermic batch reactor (Sjöberg and Agarwal 2002), or following a part temperature trajectory in wafer production (Gorinevsky 2002).

A simple batch CSTR (Continuos Stirred Tank Reactor) reactor with consecutive reactions

$$\mathsf{A} \overset{k_1}{\rightarrow} \mathsf{B} \overset{k_2}{\rightarrow} \mathsf{C}$$

can be simulated using mass balance equations (Sjöberg and Agarwal 2002)

$$\begin{aligned} \frac{dc_{\mathsf{A}}}{dt} &= -k_1(T)\, c_{\mathsf{A}}^2 \\ \frac{dc_{\mathsf{B}}}{dt} &= k_1(T)\, c_{\mathsf{A}}^2 - k_2(T)\, c_{\mathsf{B}} \\ \frac{dT}{dt} &= \gamma_1 k_1(T)\, c_{\mathsf{A}}^2 + \gamma_2 k_1(T)\, c_{\mathsf{B}} \\ &\quad + (a_1 + a_2 T) + (b_1 + b_2 T)\, u \end{aligned}$$

where c_{A} and c_{B} denote concentrations of components A and B, T is the reaction temperature. The control variable u represents a dimensionless scaling of two physical manipulated variables. The rate constants k_1 and k_2 follow the Arrhenius temperature dependence, $k_i(T) = A_i \exp\left(-\frac{E_i}{RT}\right)$, $i = 1, 2$.

The plant model equations were implemented as ordinary differential equations in an m-file `dx=abcdiff(t,x,u)`. The model parameters used in the simulations were the same as those used in (Sjöberg and Agarwal 2002): $A_1 = A_2 = 0.003$; $\frac{E_1}{R} = 0.1$; $\frac{E_2}{R} = 0.005$; $\gamma_1 = 3.33$; $\gamma_2 = 66.7$; $a_1 = 0.01$; $a_2 = -0.02$; $b_1 = 1$; $b_2 = -0.03$; initial conditions $c_{\mathsf{A}}(0) = 20$; $c_{\mathsf{B}}(0) = 0$; unit sampling time, as well as the desired output trajectory for temperature:

$$T^{\mathsf{ref}}(t) = 20 \exp(-0.02t).$$

A sampled model (`abcdiffsampled.m`) was coded using the `GDTODESampler`:

Example 9 MATLAB function `abcdiffsampled` describes the ode in form (3)

```
function [Yk1,Xk1] = abcdiffsampled(Xk,Uk,Ts,U0,diffeqstr)
Uk = Uk+U0; C=[0 0 1];
[Yk1,Xk1] = GDTODESampler(Xk,Uk,diffeqstr,Ts,C);
```

5.1.1 Initial values

First, a decision is needed weather a choice of incremental or absolute controls in the cost function would better reflect our specifications. Let us assume that absolute magnitude of the control manipulations (position of a mixing valve) has no particular cost

(e.g., input ingrediences have the same cost). Instead, we want to avoid large changes in control manipulations (to avoid wearing of equipment). Therefore, incremental control is chosen, `DELTAU=1`.

Next, 'tolerable' ranges for output and control variables are needed, so as to set initial values for $\mathbf{A}_n$ and $\mathbf{B}_n$. For the deviation in plant output (reactor temperature) a tight control is desired. Let us specify a range $[-0.2, 0.2]$ for a 'tolerable' deviation, $T^{\text{ref}} - T$, cf. Fig. 2 at x-axis $[-1, 1]$. Since design of a sequence of control increments was chosen, let us specify that a 'tolerable' Δu would be in the range $[-2, 2]$.

According to the tuning rules, we set initially $N = 100$ and $\beta = 1$.

We are now ready to implement the first test simulation (`abcrun0.m`) as a MATLAB script file.

Example 10 MATLAB script `abcrun0` sets the GDT optimization parameters and runs a simulation.

```
X0 = [20 0 5]'; Y0 = X0(3);
U0 = 0; Ts = 1;
t=[1:30]'; Yref = 20*exp(-0.02*t);
DELTAU=1;
A_n = 2^2; B_n = 0.2^2;
N = 100; bet = 1;
plantfunstr = 'abcdiffsampled'
diffeqstr = 'abcdiff'
GDT = GDTopt(Yref,A_n,B_n,N,bet,plantfunstr,X0,Y0,DELTAU,...
Ts,U0,diffeqstr);
```

Figure 3 illustrates results from a typical simulation with $N = 100$. Initially, the reactor temperature is at $T = 5$. During the first samples, the GDT scheme finds a sequence of control increments that bring the temperature close to the desired trajectory (starting from $T = 20$). After the first 10 samples, the temperature follows the desired trajectory. We observe that the control increments in the beginning of the sequence are large and then remain within the range ± 2. These can be compared with the 'tolerable' range set to ± 2. Similarly, the output deviations are large in the beginning. After the first 10 samples the standard deviation of the error between plant output and the desired trajectory is 0.34, to be compared with 'tolerable' range set to ± 0.2. The resulting control sequence bears resemblence to the simulations reported in (Sjöberg and Agarwal 2002) obtained using a neural linearization-based scheme.

5.1.2 Further simulations

In a sequence of consequtive runs, four parameter setting were examined: $N = 100$, $N = 500$, $N = 2500$ (all with $\beta = 1$), and $N = 2500$, $\beta = 0.5$. These are run by scripts `abcrun0`, `abcrun1`, `abcrun2` and `abcrun3`. The mean costs, standard deviation of the costs and the smallest cost among 165 test runs were observed as shown in the

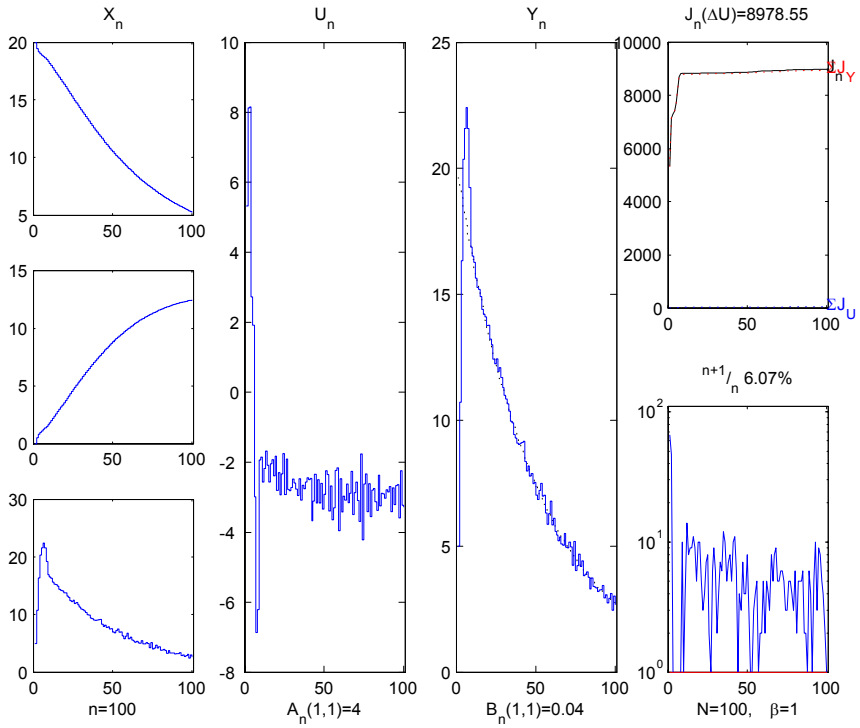

Figure 3: Simulation `abcrun0`.

following table.

test	sample mean	sample std	minimum cost	comment
$N = 100$	10252	3661	7058	163/165
$N = 500$	9932	8954	6785	164/165
$N = 2500$	8091	1788	6405	165/165
$N = 2500$, $\beta = 0.5$	7333	692	6703	165/165

From the results, it is clear that the sample mean within the test sets decreases as N increases. For the standard deviations, similar conclusions can be drawn: the deviation decreases as the population size increases.

Remark 11 Some comments are needed to explain interpretation of sample standard deviations (third column). The rightmost column 'comment' shows the number of simulations used for the statistic. With $N = 100$, two simulations were removed from the statistic (final costs were 4.75×10^{18} and 1.29×10^{6}). Similarly, for $N = 500$, one simulation was removed ($J_T = 7.55 \times 10^{13}$). These removals were due to failure of the algorithm to find a proper control sequence, due to an unrecoverable drift of the algorithm (at a particular instant all solutions were poor, the distributions became flat, and future steps were not able to recover the situation). Consequently, the sample std can be very sensitive, and results on 165 simulations may not be statistically significant. This may explain the large std in the case of $N = 500$.

The Monte Carlo simulations confirm the expectation that as the number of particles increases, the randomness in the solution gradually disappears. However, the optimization execution time increases linearily as a function of N.

For the largest population, decreasing β from 1 to 0.5 improved the sample mean. This can be explained by the "dynamic" aspects of the optimization (poorer solutions in the past may turn out to be useful in the long run). With a larger number of particles, a greater variety in the population can be tolerated. Fig. 4 illustrates a typical simultion with $N = 2500$ and $\beta = 0.5$. The control in the beginning of the sequence is aggressive and drives the plant output close to the desired trajctory in few samples.

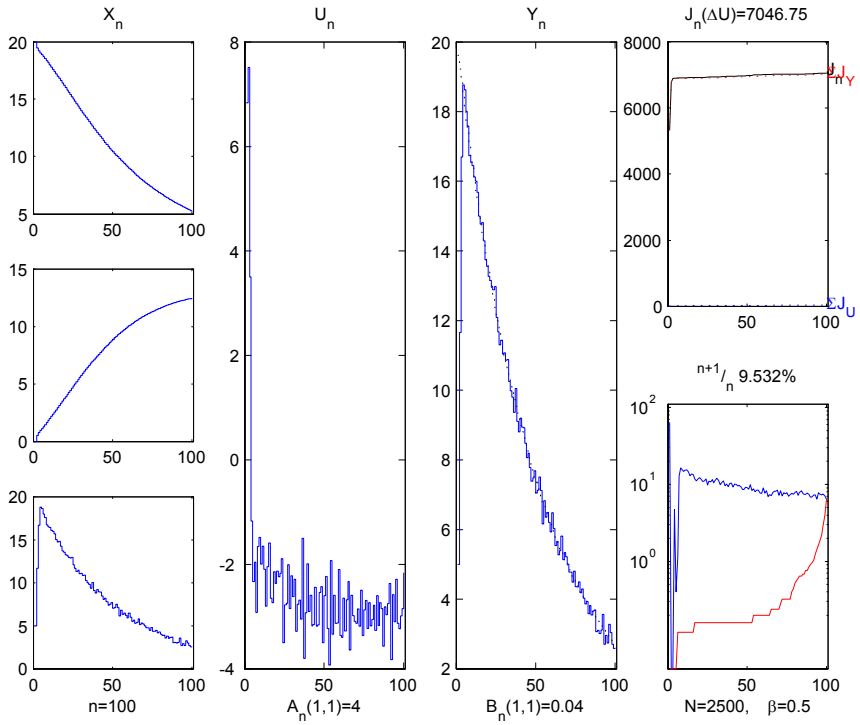

Figure 4: Simulation `abcrun4`.

5.2 RTP-plant

In order to achieve uniform processing and a high level of reproducibility of phenomena, the wafer temperature has to track a pre-specified temperature trajectory. A simple nonlinear continuous time model (Gorinevsky 2002) for such thermal processing has two states: furnace temperature T_{F} and part temperature T_{P}:

$$\begin{aligned} \dot{T}_{\mathsf{F}} &= b_{\mathsf{u}} u - c_1 \left(T_{\mathsf{F}}^4 - T_{\mathsf{P}}^4\right) - c_2 \left(T_{\mathsf{F}} - T_{\mathsf{A}}\right) \\ \dot{T}_{\mathsf{P}} &= c_3 \left(T_{\mathsf{F}}^4 - T_{\mathsf{P}}^4\right), \end{aligned}$$

where $u \geq 0$ is the heating intensity (control input), T_{P} is the part temperature (the system output to be controlled), and T_{A} is the ambient temperature. The parameter values were taken from (Gorinevsky 2002): $b_{\mathsf{u}} = 1000$, $c_1 = 1.1 \cdot 10^{-10}$, $c_2 = 0.8$ and $c_3 = 1.5 \cdot 10^{-9}$, $T_{\mathsf{A}} = 20$ °C.

The plant model equations were implemented as ordinary differential equations in an M-file `dx=rtpdiff(t,x,u)`. The requirement that $u \geq 0$ was implemented as a line `u=max(0,u)` within the `rtpdiff.m`. The plant was sampled using $T_{\mathsf{s}} = \frac{1.2}{60}$, implemented with `GDTODESampler` (`rtpdiffsampled.m`). The desired trajectory consisted of constant and ramp phases from $300°$C to $600°$C, to $900°$C and back to $600°$C. The steady-state initial conditions corresponding to part temperature $300°$C were solved from the differential equations, giving $T_{\mathsf{F}} = 300°$C and $u = 0.2240$.

5.2.1 Initial values

Let us now suppose that small control actions (heating intensity) are desired, i.e., `DELTAU=0`. We then specify that 'tolerable' controls reside within ± 1 from the nominal value ($\overline{u} = 0.224$), and that we want to follow the trajectory very closely, at $\pm 1°$C. We set $A_n = 1$, $B_n = 1$. An optimal balance between these contradicting requirements is obtained from the minimization of the cost function (9): The input to the RTP plant (u in plant ODE) is given by $U_n + \overline{u}$.

We set initially $\beta = 1$ and $N = 100$, and simulate (`rtprun0`) to see what happens.

Figure 5 illustrates a typical simulation.We observe that the results are very nice, even with such a small number of particles: the output trajectory is closely followed (largest deviations from the desired trajectory are around $10°$C), and the control actions rarely pass over 3 in magnitude.

5.2.2 Further simulations

In order to further focus on energy savings, more weight can be put on control costs. Let us consider 'tolerable' actions only within ± 0.5 around the nominal value, $A_n = 0.25$. Simulating with $N = 100$ resulted in a failure. Increasing N to 1000, however, gave excellent results (`rtprun1.m`). It appears that with $N = 100$ not enough good potential

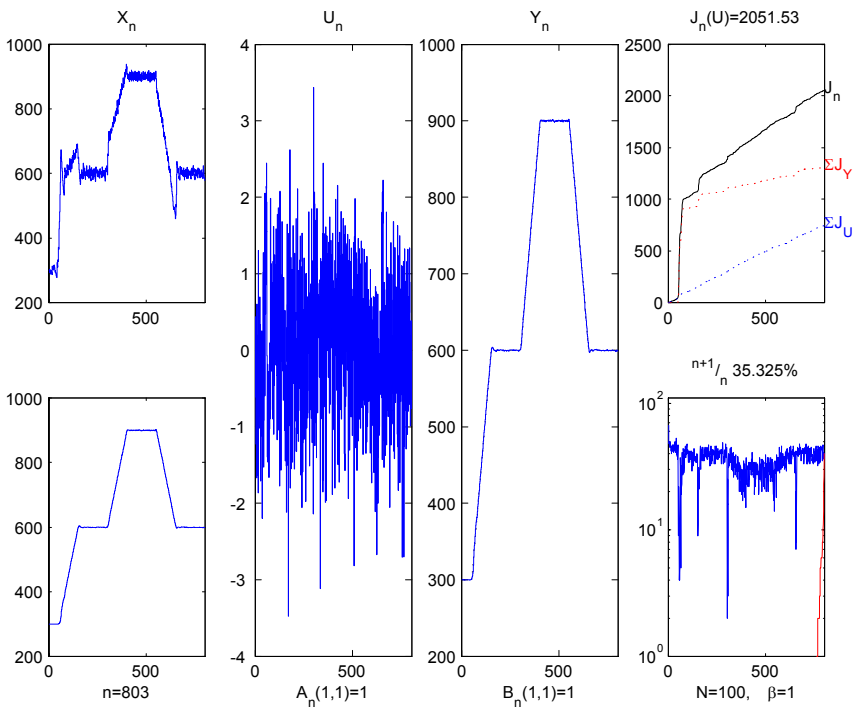

Figure 5: Simulation `rtprun0`.

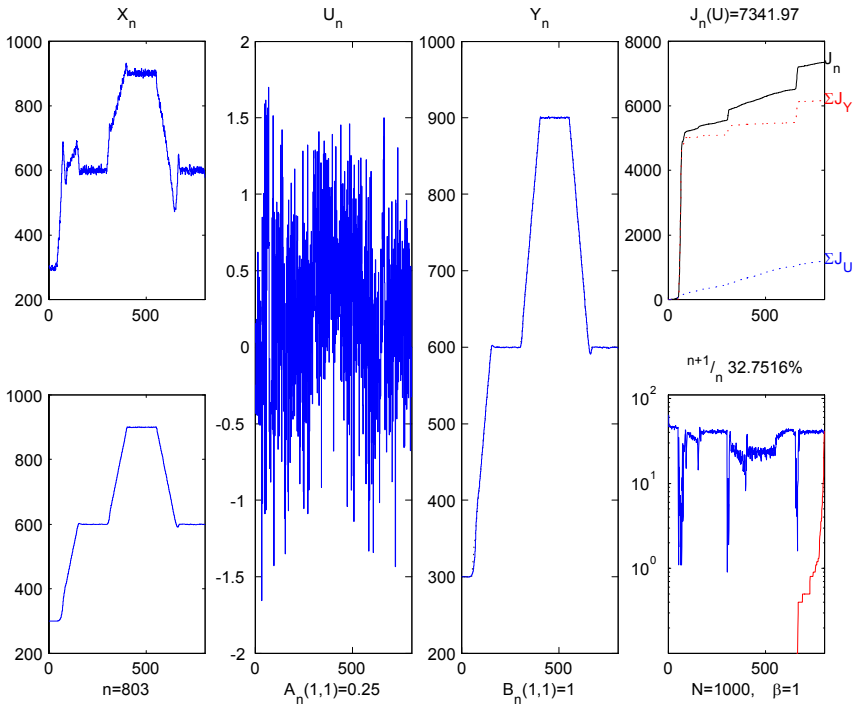

Figure 6: Simulation `rtprun1.m`.

candidates for u were generated, and setting $N = 1000$ fixes this problem. Recall that the generating distrubution for u is $\mathcal{N}(0, A_n)$, and any ability to follow the output trajectory (RTP part temperature) requires control actions (heating intensities) at the tails of this distribution. With a sufficiently large N, some samples are generated also at the ends of the distribution. Visual inspection (see Fig. 6) suggests that the control specifications are fulfilled, with significantly smaller control actions. It is worth noticing, however, that the deviation part of the costs is much larger than in `rtprun0`. This indicates that the improvement in control manipulations was obtained at the cost of output error.

Finally, recall that proper validation of the algorithm propertes would require extensive Monte Carlo simulations and examination of data statistics.

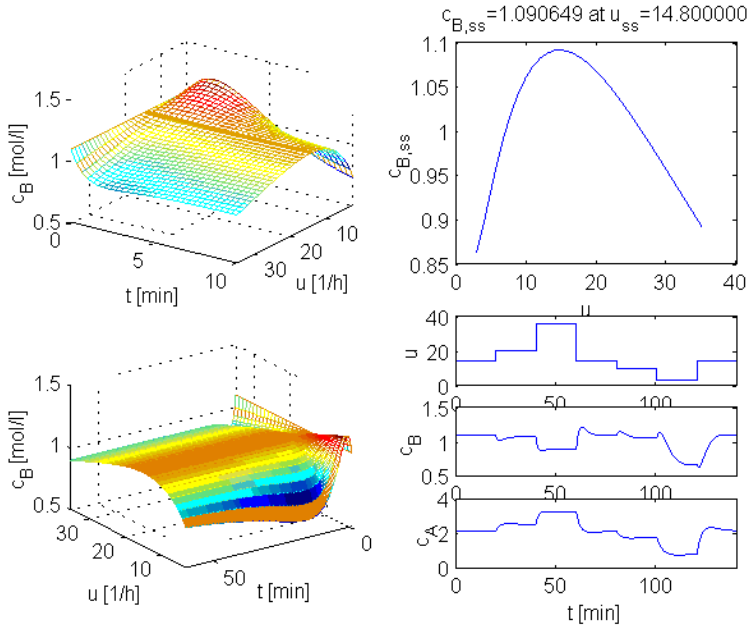

Figure 7: Behavior of the CSTR with van der Vusse reactions.

5.3 van der Vusse CSTR plant

A CSTR where van der Vusse reactions take place is a commonly used test plant for problems of nonlinear control (Chen *et al.* 1995). The plant is not only strongly nonlinear (non-monotonic, with maximum at $u = 14.8$), but its dynamic behaviour changes according to the operating point (in the vincinity of operating points with $u < 14.8$, non-minimum phase dynamics are observed). Figure 7 illustrates the behavior of the plant. Left plots illustrate the step responses of the plant for steps of size u from an initial state $c_\mathsf{A} = 2.14$ [mol/l], $c_\mathsf{B} = 1.09$ [mol/l], $v = 114.2$, [°C] and $v_\mathsf{K} = 112.9$ [°C]. The top right plot shows the steady-state curve for the plant output c_B as a function of control u. The plots at the lower right corner show the responses of c_B and c_A to a series of steps in u, as a function of time.

The plant equations are given by (Chen *et al.* 1995)

$$\begin{aligned}
\frac{d}{dt}c_{\mathsf{A}} &= \frac{V'}{V_{\mathsf{R}}}\left(c_{\mathsf{A}}^0 - c_{\mathsf{A}}\right) - k_1 c_{\mathsf{A}} - k_3 c_{\mathsf{A}}^2 \\
\frac{d}{dt}c_{\mathsf{B}} &= -\frac{V'}{V_{\mathsf{R}}}c_{\mathsf{B}} + k_1 c_{\mathsf{A}} - k_2 c_{\mathsf{B}} \\
\frac{d}{dt}v &= \frac{V'}{V_{\mathsf{R}}}\left(v^0 - v\right) - \frac{1}{\rho C_{\mathsf{p}}}\left(k_1 c_A \Delta H_{\mathsf{RAB}} + k_2 c_{\mathsf{B}} \Delta H_{\mathsf{RBC}} + k_3 c_{\mathsf{A}}^2 \Delta H_{\mathsf{RAD}}\right) \\
&\quad - \frac{k_{\mathsf{w}} A_{\mathsf{R}}}{\rho C_{\mathsf{p}} V_{\mathsf{R}}} \frac{v - v_{\mathsf{K}}}{1000} \\
\frac{d}{dt}v_{\mathsf{K}} &= \frac{1}{m_{\mathsf{K}} C_{\mathsf{PK}}}\left(Q'_{\mathsf{K}} + k_{\mathsf{w}} A_{\mathsf{R}}\left(v - v_{\mathsf{K}}\right)\right)
\end{aligned}$$

The reaction velocities are given by the Arrhenius equation: $k_i = k_{0i} e^{\frac{E_i}{v+273.15}}$. The manipulated variable is $u = \frac{V'}{V_{\mathsf{R}}}$. The values for the parameters used were the following (Chen *et al.* 1995):

- collision factors $k_{01} = 1.287 \times 10^{12}$ $[\frac{1}{\mathsf{h}}]$, $k_{02} = 1.287 \times 10^{12}$ $[\frac{1}{\mathsf{h}}]$, $k_{03} = 9.043 \times 10^{9}$ $[\frac{1}{\mathsf{mol\,h}}]$;
- activation energies $E_1 = -9758.3$, $E_2 = -9758.3$, $E_3 = -8560$ [K];
- enthalpies $\Delta H_{\mathsf{RAB}} = 4.2$, $\Delta H_{\mathsf{RBC}} = -11.0$, $\Delta H_{\mathsf{RAD}} = -41.85$ $[\frac{\mathsf{kJ}}{\mathsf{mol}}]$;
- density $\rho = 0.9342$ $[\frac{\mathsf{kg}}{\mathsf{l}}]$;
- heat capacity $C_{\mathsf{p}} = 3.01$ $[\frac{\mathsf{kJkg}}{\mathsf{K}}]$;
- heat transfer coefficient for cooling jacket $k_{\mathsf{w}} = 4032$ $[\frac{\mathsf{kJ}}{\mathsf{hm^2K}}]$;
- cooling jacket surface $A_{\mathsf{R}} = 0.215$ [m^2];
- reactor volume $V_{\mathsf{R}} = 0.01$ [m^3] ;
- coolant mass $m_{\mathsf{K}} = 5.0$ [kg];
- heat capacity of coolant $C_{\mathsf{PK}} = 2.0$ $[\frac{\mathsf{kJ}}{\mathsf{kgK}}]$;
- input concentration $c_{\mathsf{A}}^0 = 5.10$ $[\frac{\mathsf{mol}}{\mathsf{l}}]$;
- feed temperature $v_0 = 104.9$ [C];
- feed flow $V' = 14.19 V_{\mathsf{R}}$ $[\frac{\mathsf{m}^3}{\mathsf{h}}]$;
- heat removal $Q'_{\mathsf{K}} = -1113.5$ $[\frac{\mathsf{kJ}}{\mathsf{h}}]$.

The plant ode are invoked by `vdvdiff.m`, and the sampled interface is built in `vdvdiffsampled.m`. A sampling rate of 20 seconds is used.

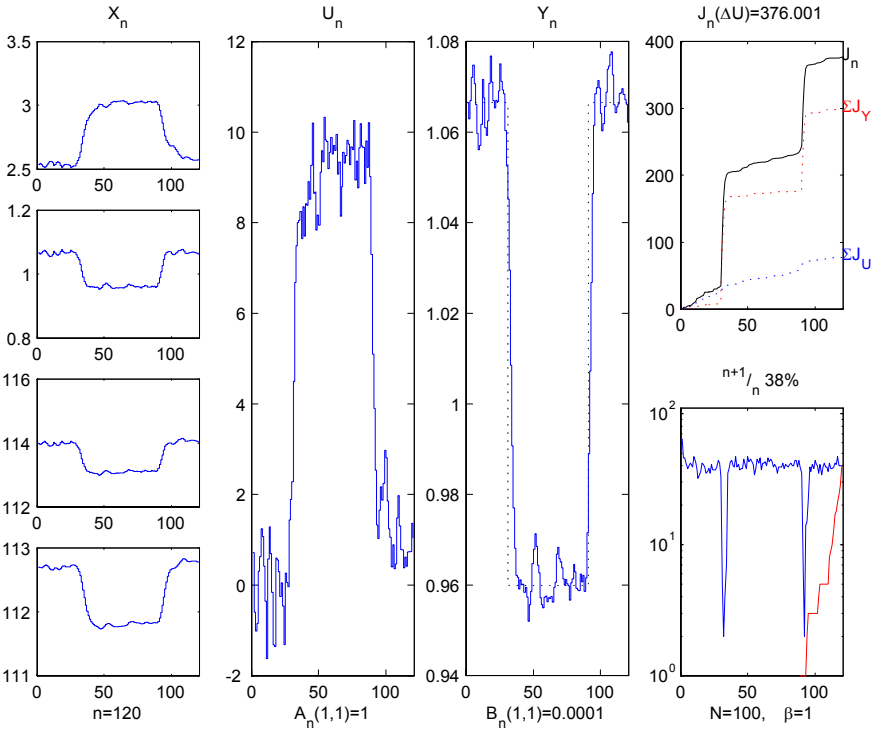

Figure 8: Simulation `vdvrun0`.

5.3.1 Initial values

For this plant, we consider incremental control (input u is related to flow, i.e., valve position, so that minimizing variability is of interest), `DELTAU=1`. We consider a SISO system, so that matrices $\mathbf{A}_n$ and $\mathbf{B}_n$ reduce to scalar constants. Based on 'tolerable' input increments and output deviations, set $\mathbf{A}_n = 1$, $\mathbf{B}_n = 0.0001$. As initial values, set $\beta = 1$ and $N = 100$. As output trajectory, an artificial sequence defined in terms of the plant intial state is considered, consisting of a 90% step downwards and back, from the initial steady state.

Let the initial steady state be at $u = 20$ (minimum phase dynamics). Figure 8 illustrates a typical simulation. Recall that the U_n shown in the plot is deviation from the nominal value ($u = 20$.) Results appear to follow the specifications.

A considerably more difficult optimization problem is the examination of a similar trajectory from $u = 5$ (in this operation region the plant exhibits strongly non-minimum phase dynamics). Figure 9 shows a typical simulation run with the initial values $N =$

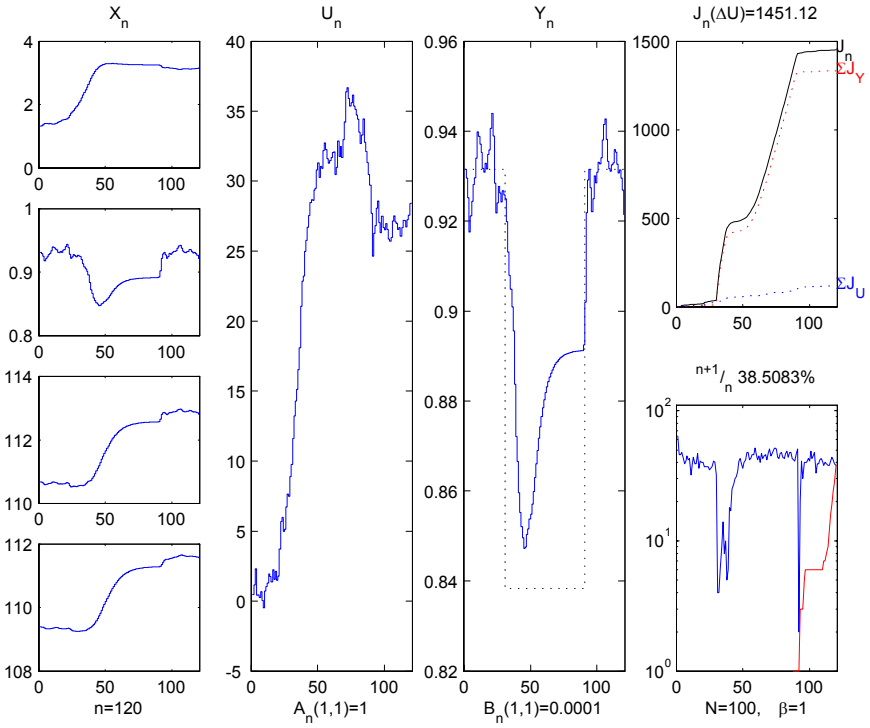

Figure 9: Simulation `vdvrun2.m`.

100 and $\beta = 1$. We observe that the solution is found by taking the process into the minimum phase region.

5.3.2 Further simulations

For the sequence from $u = 20$, a tighter control can be obtained by allowing larger control increments $\mathbf{A}_n = 4$, and decreasing the tolerated deviation from output trajectory to 0.001 $\mathbf{B}_n = 0.000001$ (`vdvrun1.m`).

For the sequence from $u = 5$ (non-minimum phase dynamics), in few simulations the following parameters were found to provide better results: $A_n = 0.1^2$, $B_n = 0.01^2$, $N = 5000$, $\beta = 0.1$. With this 'extended search space' and smoother actions, solutions were found which keep the plant in the non-minimum phase region throughout the sequence of steps, and result in significantly smaller a cost than with the initial tuning. Figure 10 illustrates a typical simulation.

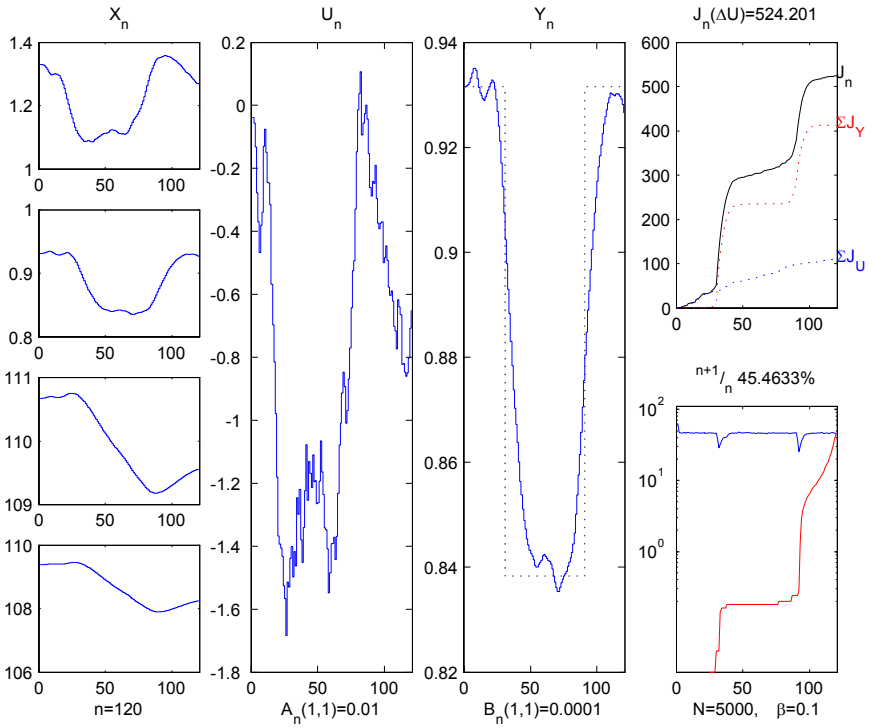

Figure 10: Simulation `vdvrun3.m`.

5.4 FBC plant

A simplified model for a FBC power plant can be formulated based on mass and energy balances. The model divides the furnace into two parts: the bed and the freeboard. The control inputs of the system are the fuel feed Q_{C} $[\frac{\mathsf{kg}}{\mathsf{s}}]$, and the primary and secondary air flows F_1 and F_2 $[\frac{\mathsf{Nm}^3}{\mathsf{s}}]$. Measurable system outputs are the flue gas O_2 content C_{F} $[\frac{\mathsf{Nm}^3}{\mathsf{Nm}^3}]$, the bed and the freeboard temperatures T_{B} and T_{F} [K], and the power output P [MW].

The model is described by the following differential equations. For the fuel inventory W_{C} [kg], oxygen concentration C_{B} [$\mathsf{Nm}^3/\mathsf{Nm}^3$] and temperature T_{B} [K] in bed:

$$\begin{aligned}
\frac{dW_{\mathsf{C}}(t)}{dt} &= (1-V)\,Q_{\mathsf{C}}(t) - Q_{\mathsf{B}}(t) \\
\frac{dC_{\mathsf{B}}(t)}{dt} &= \frac{1}{V_{\mathsf{B}}}\left[C_1F_1(t) - X_{\mathsf{C}}Q_{\mathsf{B}}(t) - C_{\mathsf{B}}(t)\,F_1(t)\right] \\
\frac{dT_{\mathsf{B}}(t)}{dt} &= \frac{1}{c_{\mathsf{I}}W_{\mathsf{I}}}\{H_{\mathsf{C}}Q_{\mathsf{B}}(t) - a_{\mathsf{Bt}}A_{\mathsf{Bt}}\left[T_{\mathsf{B}}(t) - T_{\mathsf{Bt}}\right] \\
&\quad + c_1F_1(t)\,T_1 - c_{\mathsf{F}}F_1(t)\,T_{\mathsf{B}}(t)\}
\end{aligned}$$

Similarly, the freeboard dynamics are given by:

$$\begin{aligned}
\frac{dW_{\mathsf{V}}(t)}{dt} &= VQ_{\mathsf{C}}(t) - Q_{\mathsf{F}}(t) - Q_{\mathsf{T}}(t) \\
\frac{dC_{\mathsf{F}}(t)}{dt} &= \frac{1}{V_{\mathsf{F}}}\{C_{\mathsf{B}}(t)\,F_1(t) + C_2F_2(t) \\
&\quad - X_{\mathsf{V}}Q_{\mathsf{F}}(t) - C_{\mathsf{F}}(t)\left[F_1(t) + F_2(t)\right]\} \\
\frac{dT_{\mathsf{F}}(t)}{dt} &= \frac{1}{c_{\mathsf{F}}V_{\mathsf{F}}T_{\mathsf{F}\tau}}\{H_{\mathsf{V}}Q_{\mathsf{F}}(t) - a_{\mathsf{Ft}}A_{\mathsf{Ft}}\left[T_{\mathsf{F}}(t) - T_{\mathsf{Ft}}\right] \\
&\quad + c_{\mathsf{F}}F_1(t)\,T_{\mathsf{B}}(t) + c_2F_2(t)\,T_2(t) + c_1\left[F_1(t) + F_2(t)\right]T_{\mathsf{F}}(t)\}
\end{aligned}$$

The plant superheated steam power dynamics can be approximated by

$$\frac{dP(t)}{dt} = \frac{1}{\tau_{\mathsf{mix}}}\left[P_{\mathsf{T}}(t) - P(t)\right]$$

The combustion rates can be approximated by

$$Q_{\mathsf{B}}(t) = \frac{W_{\mathsf{C}}(t)}{t_{\mathsf{C}}}\frac{C_{\mathsf{B}}(t)}{C_1} \text{ and } Q_{\mathsf{F}}(t) = \frac{W_{\mathsf{V}}(t)}{t_{\mathsf{V}}}\frac{C_{\mathsf{F}}(t)}{C_2},$$

where t_{C} and t_{V} refer to the mean particle combustion time; $Q_{\mathsf{T}}(t) = \frac{W_{\mathsf{V}}(t)}{t_{\mathsf{T}}(t)}$, $t_{\mathsf{T}}(t) = \frac{V_{\mathsf{F}}}{F_1(t)+F_2(t)}$. The heat transfer is given by

$$P_{\mathsf{T}}(t) = a_{\mathsf{Bt}}A_{\mathsf{Bt}}\left[T_{\mathsf{B}}(t) - T_{\mathsf{Bt}}\right] + a_{\mathsf{Ft}}A_{\mathsf{Ft}}\left[T_{\mathsf{F}}(t) - T_{\mathsf{Ft}}\right] + c_{\mathsf{F}}\left(F_1 + F_2\right)\left(T_{\mathsf{F}} - T_{\mathsf{stack}}\right).$$

The parameter values were given by:

- $c_{\mathsf{I}} = 800$; Bed material specific heat [J / kgK]
  $W_{\mathsf{I}} = 15000$; Bed inert material [kg]
  $V_{\mathsf{B}} = 26.2$; Bed volume [m^3]
  $a_{\mathsf{Bt}} = 210$; Heat transfer coefficient in bed [W / m^2K]
  $A_{\mathsf{Bt}} = 65$; Heat exchange surface in freeboard [m^2]
  $T_{\mathsf{Bt}} = 273 + 300$; Cooling water temperature [K]
- $V_{\mathsf{F}} = 128$; Freeboard volume [m3]
  $a_{\mathsf{Ft}} = 210$; Heat transfer coefficient in freeboard [W / m^2K]
  $A_{\mathsf{Ft}} = 317$; Heat exchange surface in freeboard [m^2]
  $T_{\mathsf{Ft}} = 273 + 300$; Cooling water temperature [K]
  $T_{\mathsf{F}\tau} = 50$; Freeboard temperature time constant [s]
- $C_1 = 0.21$; Primary air O_2 [Nm^3/Nm^3]
  $c_1 = 1305$; Primary air specific heat [J / m^3K]
  $T_1 = 273 + 55$; Primary air temperature [K]
  $C_2 = 0.21$; Secondary air O_2 [Nm^3/Nm^3]
  $c_2 = 1305$; Secondary air specific heat [J / m^3K]
  $T_2 = 273 + 55$; Secondary air temperature [K]
  $c_{\mathsf{F}} = 1305$; Flue gas specific heat [J / m^3K]
- $X_{\mathsf{C}} = 0.42 \times 1.866$; O_2 consumed in combustion of char [Nm^3/kg]
  $H_{\mathsf{C}} = 0.27 \times 30 \times 10^6$; heat value of char [J/kg]
  $t_{\mathsf{C}} = 50$; Mean combustion rate of char [1/s]
- $V = 0.75$; Fraction of volatiles [kg/kg]
  $X_{\mathsf{V}} = 0.42 \times 1.866$; O_2 consumed in combustion of volatiles [Nm^3/kg]
  $H_{\mathsf{V}} = 50 \times 10^6$; Heat value of volatiles [J/kg]
  $t_{\mathsf{V}} = t_{\mathsf{C}}/1000$; Mean combustion rate of volatiles [1/s]
- $T_{\mathsf{stack}} = 273 + 55$; Stack gas temperature [K]
  $T_{\mathsf{mix}} = 15$; Time constant [s]

The model was implemented on MATLAB. The ODE are invoked by the command `xdot=fbcdiff(t,x,u)`. The function `[Yk1,Xk1] = fbcdiffsampled(Xk, Uk, Ts, U0, diffeqstr, C)` provides the sampled interface for the FBC plant model, where the additional parameters are the sampling time `Ts` (15 s), nominal control `U0`, name of the differential equation file (`diffeqstr = 'fbcdiff'`) and a measurement matrix `C`. The system states consisted of the three inputs (states 1–3) and the seven variables given by the ODE (states 4–10). All states were all constrained to be non-negative.

Figure 11 illustrates a simulation using the above model, tuned for a 25MW district heating power plant. The simulated values are compared with measurements from a real 25 MW district heating plant.

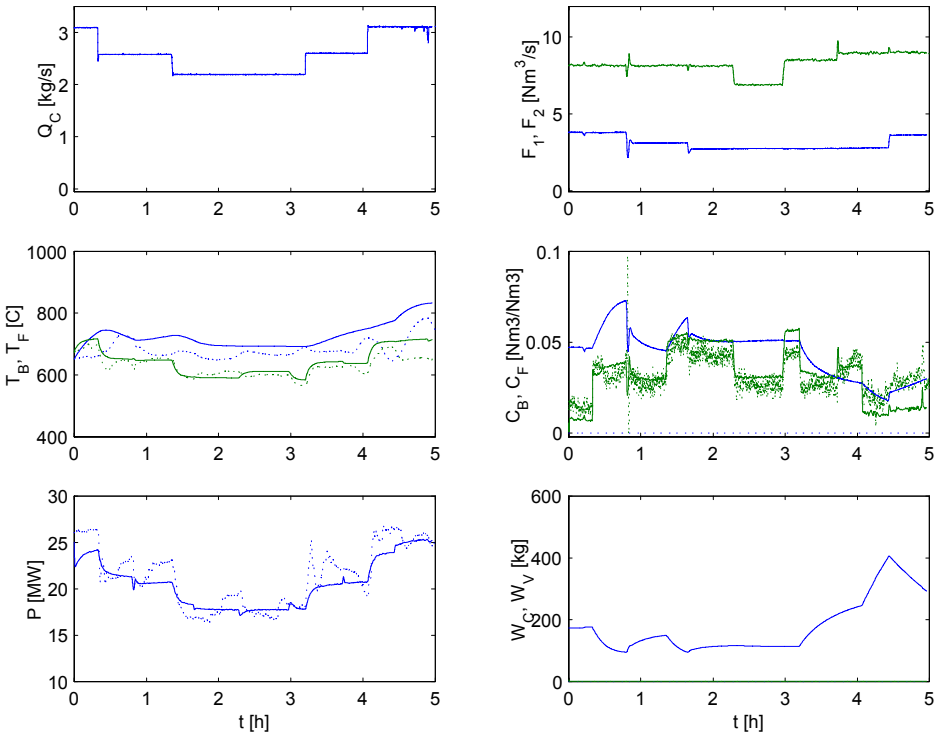

Figure 11: Comparison of the FBC plant model with data measured from a 25MW power plant.

5.4.1 Initial values

Let us consider a three-input three-output problem, with a trajectory consisting of an artificial step change (forth and back) between two steady states:

$$\mathbf{U} = \left[\begin{array}{ccc} 2.6 \ \frac{\mathsf{kg}}{\mathsf{s}} & 3.7 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} & 8.4 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} \end{array} \right]$$

and

$$\mathbf{U} = \left[\begin{array}{ccc} 3.1 \ \frac{\mathsf{kg}}{\mathsf{s}} & 3.7 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} & 11 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} \end{array} \right].$$

The flue gas oxygen, bed temperatures, and superheated steam power are the controlled variables, controlled using manipulations in the fuel feed rate, the primary flow and the secondary air flow.

Optimization of the sequence of control actions is considered, so as to minimize fuel/air flows, `DELTAU=0`. The nominal control was set to $\overline{\mathbf{U}} = \left[\begin{array}{ccc} 2 \ \frac{\mathsf{kg}}{\mathsf{s}} & 3 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} & 6 \ \frac{\mathsf{Nm}^3}{\mathsf{s}} \end{array} \right]$, slightly below the initial state. The cost function is further specified by setting the 'tolerable' deviations to 1% in oxygen content, 100K in bed temperatures and 0.5MW in superheated steam power:

$$\mathbf{A}_n = \left[\begin{array}{ccc} 0.01^2 & 0 & 0 \\ 0 & 100^2 & 0 \\ 0 & 0 & 0.5^2 \end{array} \right].$$

The 'tolerable' control actions (costs on magnitudes) were specified as 'reasonable', such that the trajectory targets could be attained by a reasonable amount of the generated controls:

$$\mathbf{B}_n = \left[\begin{array}{ccc} 2^2 & 0 & 0 \\ 0 & 2^2 & 0 \\ 0 & 0 & 4^2 \end{array} \right].$$

This setting will result in that the sequence $\mathbf{U}_n$ is optimized, where the actual plant input vector (Q_{C}, F_1 and F_2 in the ODE) is given by $\overline{\mathbf{U}} + \mathbf{U}_n$.

As always, set initially $N = 100$ and $\beta = 1$. Figure 12 illustrates a typical simulation of an optimization run (`fbcrun0.m`). The optimization results in a rough control sequence. The power trajectory (third column, bottom row) is somehow followed, even if with a large variance. The oxygen concentration (top row) is far beyond the design specifications, values above 10% are observed. For bed temperatures (middle row), the design specifications were very loose, and they are largely fulfilled.

It is obvious that a larger particle population is needed to properly solve this 3×3 problem.

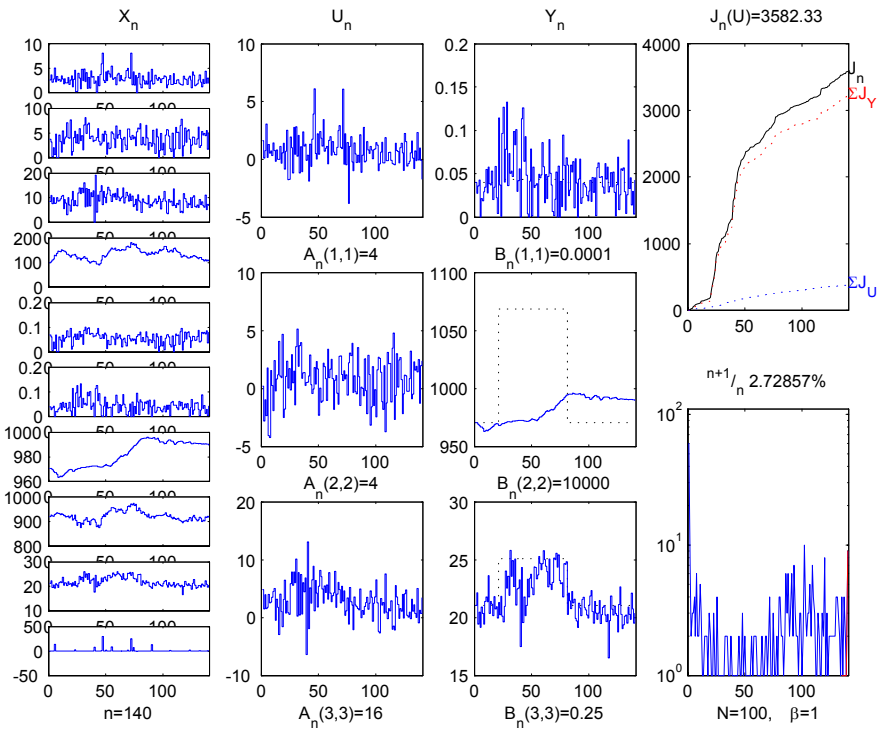

Figure 12: Simulation `fbcrun0.m`.

5.4.2 Further simulations

From tuning point of view, it is interesting to note that the percentage of survivors is very low (see Fig. 12). Very few particles in the population survive to the next generation. This suggests to loosen the specifications (increse $\mathbf{A}_n$ and/or $\mathbf{B}_n$) and/or to widen the distributions by the tuning factor β (decrease β). The first thing to try, however, is to increase N. Increasing the number of particles results in a significantly smaller cost. However, the sequences tend to remain rough as the cost function was specified using magnitudes of control actions.

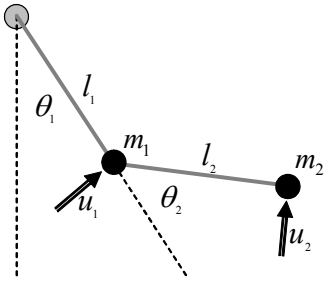

Figure 13: Rigid two-link robot.

5.5 Two-joint robot manipulator

Robotic systems are typical examples of complex systems for which accurate models can be derived. Therefore, they are particularily potential for model-based approches such as the genealogical decision trees. A frictionless double pendulum (rigid two-link robot manipulator) is depicted in Figure 13.

A TITO (two-input two-output) system is considered. The system outputs are given by the angular positions of the two links, the control manipulations are the control torques applied at the joints.

Let us assume that the links are massless, and denote the control torques by u_1 and u_2. The Euler–Lagrange equations lead to

$$\mathbf{A}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \mathbf{B}\left(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}\right) + \mathbf{C}(\boldsymbol{\theta}) - \mathbf{U} = \mathbf{0} \tag{10}$$

where

$$\ddot{\boldsymbol{\theta}} = \begin{bmatrix} \dfrac{d^2\theta_1}{dt^2} \\ \dfrac{d^2\theta_2}{dt^2} \end{bmatrix}, \mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

and

$$\begin{aligned}
\mathbf{A}(\boldsymbol{\theta}) &= \begin{bmatrix} m_1 l_1^2 + \sum_{i=1}^{2} m_2 l_i^2 + 2 m_2 l_1 l_2 \cos\theta_2 & m_2 l_2^2 + m_2 l_1 l_2 \cos\theta_2 \\ m_2 l_2^2 + m_2 l_1 l_2 \cos\theta_2 & m_2 l_2^2 \end{bmatrix} \\
\mathbf{B}\left(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}\right) &= \begin{bmatrix} -m_2 l_1 l_2 (\sin\theta_2)\, \dot{\theta}_2 \left(2\dot{\theta}_1 + \dot{\theta}_2\right) \\ m_2 l_1 l_2 (\sin\theta_2) \left(\dot{\theta}_1\right)^2 \end{bmatrix} \\
\mathbf{C}(\boldsymbol{\theta}) &= \begin{bmatrix} m_1 g l_1 \sin\theta_1 + m_2 g l_1 \sin\theta_1 + m_2 g l_2 \sin(\theta_1 + \theta_2) \\ m_2 g l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}.
\end{aligned}$$

For $l_1 = l_2 = 0.4$ [m], $m_1 = 3.0$ [kg], $m_2 = 2.0$ [kg], $g = 9.825$ $[\frac{\mathrm{m}}{\mathrm{s}^2}]$, we obtain the following 'coefficients'

$$\begin{aligned}
\mathbf{A}(\boldsymbol{\theta}) &= \begin{bmatrix} 1.12 + 0.64\cos\theta_2 & 0.32 + 0.32\cos\theta_2 \\ 0.32 + 0.32\cos\theta_2 & 0.32 \end{bmatrix} \\
\mathbf{B}\left(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}\right) &= \begin{bmatrix} -0.32 (\sin\theta_2)\, \dot{\theta}_2 \left(2\dot{\theta}_1 + \dot{\theta}_2\right) \\ 0.32 (\sin\theta_2) \left(\dot{\theta}_1\right)^2 \end{bmatrix} \\
\mathbf{C}(\boldsymbol{\theta}) &= \begin{bmatrix} 11.79 \sin\theta_1 + 7.86 \sin\theta_1 + 7.86 \sin(\theta_1 + \theta_2) \\ 7.86 \sin(\theta_1 + \theta_2) \end{bmatrix}.
\end{aligned}$$

The equations were coded in MATLAB (ODE invoked from `robdiff.m` and sampled plant interface in `robodiffsampled.m`). The M-function `robo_plot` produces an animated plot of the robot movements.

Figure 14 illustrates the movement of the pendulum from an initial position (marked by 'o'). It is easy to observe that the end of the second link shows chaotic behavior.

5.5.1 Initial values

The target was to drive the robot arm from down position $(0, 0)$ to an upright position $(\pi, -2\pi)$, and remain there for 30 samples. The sampling frequency was set to 40 Hz ($T_\mathrm{s} = 0.025$ s). The cost function was specified as `DELTAU=0`, $\mathbf{B}_n = \mathrm{diag}\left(\begin{bmatrix} \left(\frac{\pi}{32}\right)^2 & \left(\frac{\pi}{32}\right)^2 \end{bmatrix}\right)$, $\mathbf{A}_n = \mathrm{diag}\left(\begin{bmatrix} 10^2 & 10^2 \end{bmatrix}\right)$. This corresponds to a 'tolerable' output deviation of apprx. $\pm 6^\circ$, while large control actions were tolerated. The algorithm parameters were set to $N = 100$, $\beta = 1$. Figure 15 illustrates a typical robot trajectory in the (x,y)-space with this parameter setting (`roborun0.m`). The optimization was not succesful, as typically the arm followed the trajectory to the upright position but continued its angular movement (recall that the joints are frictionless).

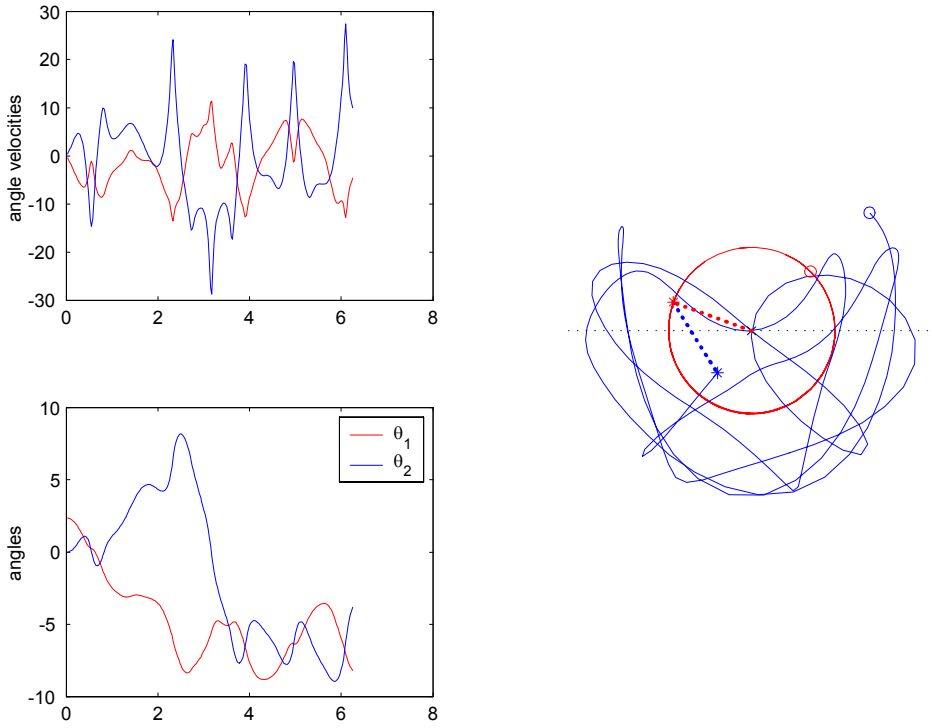

Figure 14: Behavior of the two-joint robot arm.

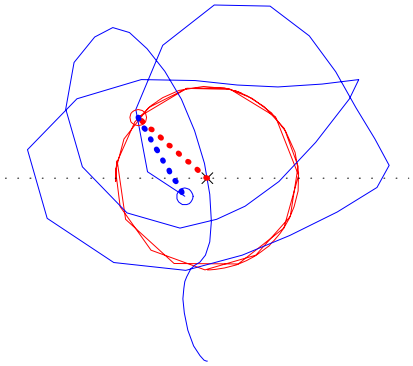

Figure 15: (x,y)-plot of robot trajectory (`roborun0.m`).

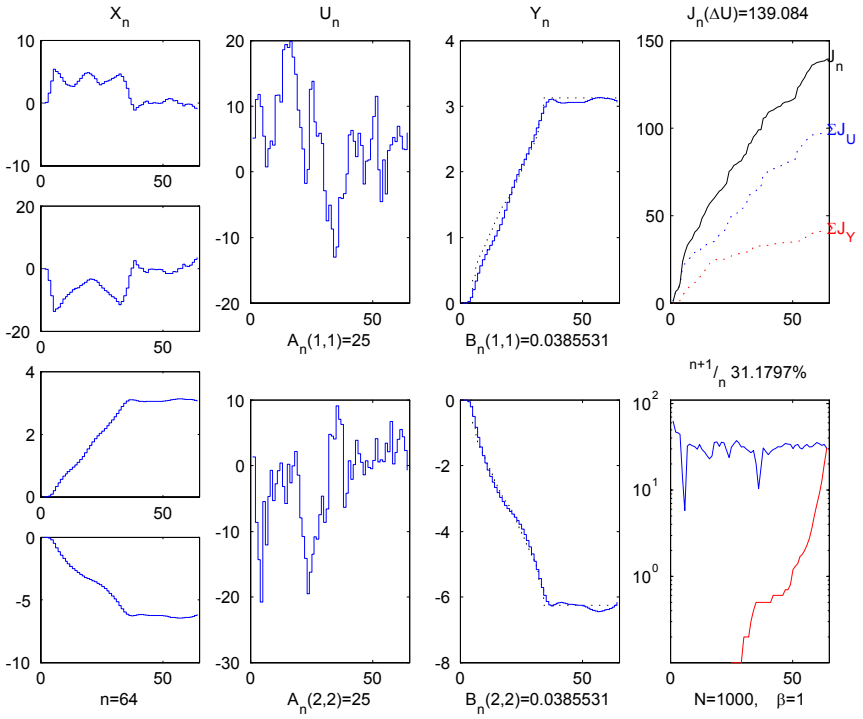

Figure 16: Simulation `roborun1.m`.

5.5.2 Further simulations

After a few test runs, the control increments were slightly reduced $\mathbf{A}_n = \text{diag}([5^2, 5^2])$, and the population size N increased ($N = 1000$). A simulation is illustrated in Fig. 16, an (x,y)-plot is shown in Fig. 17. We observe that the two-link robot is driven succesfully to an upright position, and remains there.

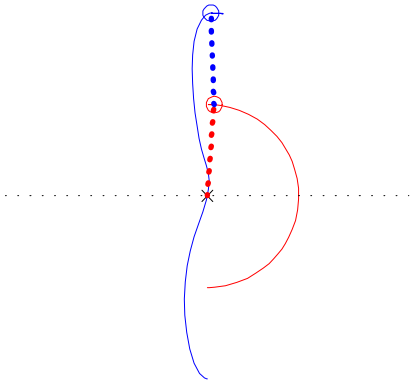

Figure 17: (x,y)-plot of robot trajectory (`roborun1.m`).

5.6 Conclusions

In this Section, a number of different types of processes, models and trajectories in GDT-based optimization of open-loop control sequences were examined. The first example considered a batch CSTR (with the $A \rightarrow B \rightarrow C$) reaction taking place. A sequence of control increments was optimized so as to follow an exponential batch trajectory. The second example considered a repetitive RTP plant where the wafer temperature trajectory was to be followed, by optimizing the plant input control sequence around a nominal value. The control design for batch plants is problematic because in the absence of a well-determined operating point a linearization approach around an equilibrium point cannot be used. Therefore non-linear plant descriptions are needed. The GDT algorithm can easily deal with nonlinearities, such as in the 'ABC' and RTP batch plants. This was clearly illustrated by the successful simulations. We also showed that the tuning rules for both initial and fine-tuning resulted in control sequences with good performance.

The third example studied the van der Vusse CSTR. This plant is strongly nonlinear, with diffult non-minimum phase dynamics. These pose a particular challenge for 'dynamic' optimization algorithms, such as the GDT. We illustrated that the algorithm was able to overcome these difficulties, and produced reasonable results in all operating regions. The FBC plant provided an example of a nonlinear MIMO plant, as a three-input three-output optimization problem was considered. Again, we found that the initial tuning rules provided a good starting point for the optimization. In the final example, a frictionless two joint robot arm was considered. This TITO plant exhibits chaotic dynamics. The algorithm was able to drive the robot (model) in an upright position.

Full descriptions (ODE) were provided for all plants, as well as complete sets of model parameters.

6 Extensions and future directions

For reasons of both practical applications and academic challenge, many extensions of the genealogical decision tree approach can be considered. In this Section, some possible directions are briefly drafted.

6.1 Other cost functions and distributions

The duality between cost/performance functions and probability measures can be extended to more general situations. For instance, for SISO bang/bang control problems the cost function is rather given by an expression of the form

$$J_T\left(U_1,...,U_T\right)=\sum_{n=1}^{T}\alpha_n U_n+\sum_{n=1}^{T}\left\|Y_n-Y_n^{ref}\right\|_{B_n}^2$$

with control sequences $U_1,...,U_T$ taking values in $\{0,1\}$, and for some strictly positive sequence of parameter α_n. In this context the duality is given as above by replacing the Gaussian distribution in (5) by the distribution of a random variable U_n which takes the value 0 with probability

$$p_n=\frac{1}{1+\exp\left(\alpha_n\right)}$$

and 1 with probability $1-p_n$. In other words, the Gaussian distribution is replaced by $\Pr\left(U_n=0\right)=1-\Pr\left(U_n=1\right)=p_n$.

6.2 Correlated and time-varying specifications

In the examples in Section 5, diagonal matrices $\mathbf{B}_n$ and $\mathbf{A}_n$ were specified. In some cases, off-diagonal elements can be useful. An example illustrates this.

Example 12 In the FBC plant the control inputs were chosen as fuel feed, primary air and secondary air. In practice it is common that a minimum level for the primary air is set, based on the fuel feed rate. For safety reasons, and to ensure beneficial combustion conditions, proper stoichiometric conditions are to be maintained in the bed. Based on simple chemical and physical information, a linear relation can be established between the primary air and the fuel feed rate, $F_1=CQ_C$, where C is a coefficient. This suggests the use of off-diagonal elements C in $\mathbf{A}_n$, e.g.,

$$\mathbf{A}_n=\begin{bmatrix} a_n\left(1,1\right) & C & 0 \\ C & a_n\left(2,2\right) & 0 \\ 0 & 0 & a_n\left(3,3\right) \end{bmatrix}$$

This has the practical consequence that the generating distribution for particle control actions will produce correlated random vectors. The use of off-diagonal elements is supported by the `Urand.m` in the MGDT-Toolbox

Also time-varying models/parameters/specifications may be handy. These are not supported by the MGDT-Toolbox, but it should be simple to include this feature if so desired.

Example 13 In the case of the two-link robot, the target trajectory consisted of two different stages. The first task was to drive the arm to an upright position; the second task was to keep the two poles balanced at this position. It can be expected that different parameters for these two tasks would make it easier to find proper solutions. In the case of 'abc'-plant, it is clear that the system gain increases as the batch proceeds. Given that we always have only a finite number of particles in our population, a more efficient search would use time-varying parameters, for example in A_n.

6.3 Feed-back control

In real applications, feed-back control is required to diminish the effect of disturbances acting on the process. A straightforward way to use the genealogical decision tree approach is to pre-optimize a control sequence using a plant model (off-line), and add feed-back controllers to compensate for the disturbances (on-line). The output signal from the feed-back controller(s) is simply summed with the current value of the the pre-optimized sequence.

In many cases, *nonlinear MIMO* open-loop optimization (such as the GDT) is able to take care of the essential plant nonlinearities. The remaining deviations and small disturbances can then be dealt with *linear SISO* feedback. In general, SISO loops are simple to tune, robust, and easy to implement. Alternatives for model-based tuning include the Ziegler–Nichols rules, the SIMC rules (Skogestad's Internal Model Control), pole-placement, etc. Linear models can often be derived from the models required by the GDT via linearization (analytically or by numerical approximation). It seems completely plausable to apply adaptive control (gain scheduling, for example) if disturbances act in a severily nonlinear fashion. Similarly, the application of the RGA-method or decoupling techniques for severely interacting multivariable disturbances would seem to be straightforward.

In (Ikonen and Kovacs 2006), feed-back control of a 3×3 GDT-optimized FBC was considered using 2 SISO PI-controllers, tuned using the SIMC rules. The third output was not measurable, but was controlled in open-loop via the GDT and the plant model.

6.4 Computational efficiency

The GDT method is computationally intensive. This is particularily true for complex systems. However, for such systems few other optimization techniques are available, if any. The straightforward MATLAB implementation of the MGDT Toolbox can easily be improved by proper *vectorization*. For example, solving several sets of ODE simultaneously (in parallel) has been found to greatly increase the speed of iterations. This presupposes, however, that both the ODE and the `GDTopt` function are adjusted

appropriately. It is obvious that any possibilities for truly *parallel computing* can be very advantageous.

6.5 Constraints

In general, a major advantage of random search techniques is the easiness to deal with nonlinearities, incontinuities, etc. This includes handling of *constraints* (whether input, output, rate, etc.). From this point of view, the genealogical decision tree approach should not make a difference. The investigation of the many types of constraints of significance in the industrial practice is a potential future direction to take.

References

Arulampalam, M S, S Maskell, N Gordon and T Clapp (2002). A tutorial on particle filters for Online Nonlinear/Non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing* **50**(2), 174–188.

Chen, H, A Kremling and F Allgöwer (1995). Nonlinear predictive control of a benchmark CSTR. In: *Proceedings of the European Control Conference, Rome, Italy*. pp. 3247–3252.

Del Moral, P (2004). *Feynman-Kac Formulae - Genealogical and Interacting Particle Systems with Applications*. Springer Verlag. Berlin.

Doucet, A, N Freitas and N Gordon (2001). *Sequential Monte Carlo Methods in Practice*. Springer.

Gorinevsky, D (2002). Loop-shaping for iterative control of batch processes. *IEEE Control Systems Magazine* **22**(6), 55–65.

Ikonen, E and J Kovacs (2006). Artificial intelligence in energy and renewable energy systems. Chap. Learning control of fluidized bed combustion processes for power plants. Nova Publishers.

Ikonen, E and K Najim (2002). *Advanced Process Identification and Control*. Marcel Dekker Inc. New York.

Ikonen, E, K Najim and P Del Moral (2005). Application of genealogical decision trees for open-loop tracking control. In: *16th IFAC World Congress, Prague, Czech*.

Ikonen, E, P Del Moral and K Najim (2004). A genealogical decision tree solution to optimal control problems. In: *IFAC Workshop AFNC'04*. Oulu, Finland.

Najim, K, E Ikonen and D Ait-Kadi (2004). *Stochastic Processes: Estimation, Optimization and Analysis*. Kogan Page Science. New York.

Najim, K, E Ikonen and P Del Moral (2006). Open-loop regulation and tracking control based on a genealogical decision tree. *Neural Computing and Applications*.

Sjöberg, J and M Agarwal (2002). Trajectory tracking in batch processes using neural controllers. *Engineering Applications of Artificial Intelligence* **15**, 41–51.

Stramer, O (2006). Probability and statistics for engineering and physical sciences. Technical report.

Ye, M (2001). Aerial point target detection and tracking - a motion-based Bayesian approach. Technical report. ISL.

Index