Automation in Complex Systems EIEN35, 2022

Computer exercises (note: both exercises are included in this document)

Goal Task 1

The purpose of the simulations in Task 1 is to study some fundamental properties of differential equation solvers in Matlab. How does the stiffness of a system affect the simulation time and what can be done to overcome long simulation times? There are not very many questions to answer but you are expected to experiment using various solvers also on your own.

Goal Task 2

The goal of Task 2 is to introduce multivariate monitoring using multivariate statistical process control (MSPC) and to illustrate how it can be used to detect disturbances and deviations in the operation of an industrial process.

Download Files

For Task 2 you will first have to download a zipped archive (AKS_simu_task_2_code.zip) from http://www.iea.lth.se/aks/exercises

Report

These exercises are carried out either **individually** or in **groups of two**. The report should contain answers to the questions stated including your own comments, motivations and conclusions. When "(**Provide plot**)" is indicated, plots should be included in the answer to that particular question. You can write the reports either in Swedish or English. Note: remember to put your name and email address on a cover page. The reports can be submitted electronically (give the file a name of structure **AKS1_Familyname** (as a pdf-file)) or as a hard copy **not later than Friday April 8, 2022 for Task 1 and not later than Wednesday May 3, 2022 for Task 2** to: ulf.jeppsson@iea.lth.se

Feedback

We appreciate if you try to **estimate the time** you spend on this exercise and indicate the number of hours on the cover page. You can also provide other comments on the exercise if you wish to help us improve the tasks for future years.

Help

If you need help, send me an email: ulf.jeppsson@iea.lth.se. I will try to answer your questions as quickly as possible.

Good luck!

Ulf Jeppsson

TASK 1 – Solving stiff differential equation systems

Preparation

Read the Sections on "Numerical Solution of Differential Equations" and "Differential Algebraic Equations", Chapters 8.4 and 8.5 in the textbook Olsson-Rosen (2005 edition).

Simulation exercises

1. The system

The stiff system to simulate is described by Figure 1. Tank 1 is a small mixing device and tank 2 is a large storage tank.

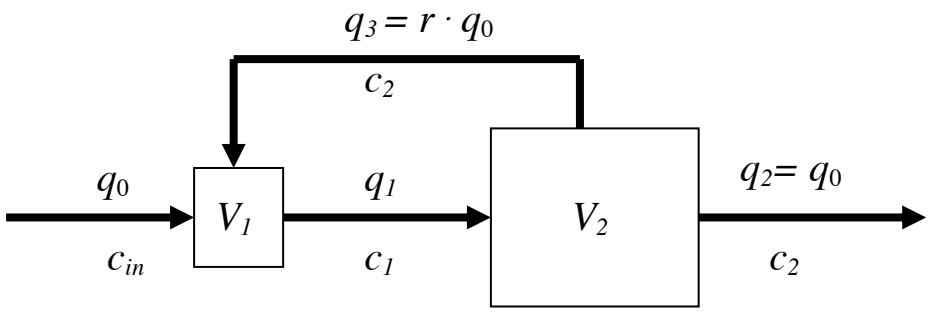

Figure 1. The system with recirculation.

The influent flow rate is q_0 with the concentration c_{in}. The concentrations in the tanks (that are completely mixed) are c_1 and c_2 respectively. There is a recycle flow q_3, and the flow rate is expressed as a fraction r of the feed rate q_0, i.e. $q_3 = r \cdot q_0$. The volumes of the tanks are V_1 and V_2 respectively. With $V_1 = 10$, $V_2 = 200$, $r = 0.6$ and $q_0 = 1$. The simulation results are shown in Figure 2. The simulation period is 2000 seconds, the influent concentration, c_{in}, is first 10 and when t > 1000, c_{in} is 20. Initial conditions [0 0].

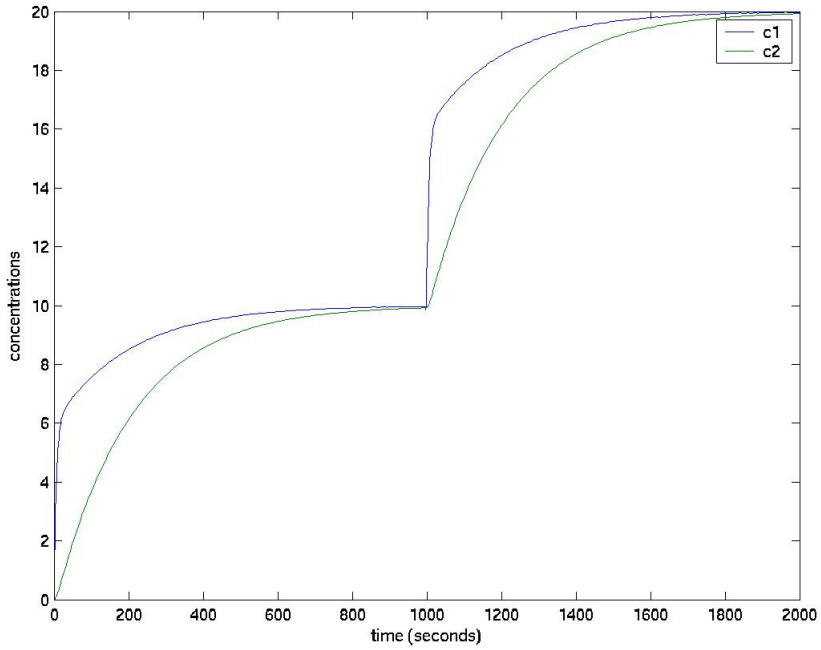

Figure 2. Simulation results.

2. Coding the model in Matlab

Matlab has a number of functions for solving differential equation systems. These are:

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	If using crude error tolerances or solving moderately stiff problems.
ode113	Nonstiff	Low to high	If using stringent error tolerances or solving a computationally intensive ODE file.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	If the problem is only moderately stiff and you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

More information on the different solvers are given at the end of this exercise.

However, there is a common way to access these functions and to solve an ODE system. Type for instance:

```
>> help ode45
```

to learn more on ode45. Read the help files for the other solvers as well.

In short, you need to define your model in an m-file with the derivatives of the equations as an output. The model is simulated by calling it from the solver function, for instance:

```
>>[t,x]=ode45(@yourmodel,[start_time stop_time],x0,[options],p1,p2,…)
```

where *yourmodel* is the filename of the m-file in which your model is defined. An example: let say we want to model the concentration in a tank. There is one input and one output. The effluent flow rate is equal to the influent flow rate (i.e. volume is constant). The influent concentration is c_{in} and there is no reaction taking place in the tank. Then the differential equation is:

$$\frac{dc}{dt} = \frac{q}{V}(c_{in} - c)$$

An m-file for this system could look like:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function dc=mymodel(t,c,q,V)

cin=1;
dc=q/V*(cin-c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

To simulate this model for 2000 time units using, for instance, ode45:

```
>>q=1;
>>V=1;
>>c0=0;
>>[t,c]=ode45(@mymodel,[0 2000],c0,[],q,V); % options are omitted
>>plot(t,c)
```

Questions and tasks:

2.1 Write an m-file for the two-tank system. V_1=10, V_2=200, r=0.6 and q_0=1. Let the influent concentration (c_{in}) initially be 10 and when t > 1000 seconds increase it to 20. Remember that the output of derivatives should be a column vector.

2.2 Simulate the model for 2000 seconds and plot the result. Let the initial tank concentrations be zero for both tanks. Make sure you obtain the same result as depicted in Figure 2.

3. Solver performance

We will now try some different solvers to see how they perform. An obvious measure of performance is the simulation time. To make the comparison between the solvers fair, all simulations must be carried out on the same computer. To avoid influence of other tasks carried out by the computer, the cpu-time is used (hint: use a number of simulations and calculate the average cpu time). To clock the time it takes to simulate a model, type:

```
 >> cpu_t=cputime;[t,c]=ode45(@mymodel,[0 2000],c0,[options],q,V);cputime-
cpu_t
```

Another indication of the performance is how many time steps are needed to simulate the time period without deteriorating result precision (to think about: how many steps do the solver actually take?). So, for a fair comparison we set the tolerance of the method. The tolerance can be specified by typing:

```
>> options=odeset('RelTol',1e-3,'AbsTol',1e-6);
```

Questions and tasks:

3.1 Set the tolerance: `RelTol` =1e-3 and `AbsTol` = 1e-6.

3.2 Simulate the model using ode45. How much time and how many steps are needed? (Hint: `size(t)` or `length(t)` reveals how many steps used.)

3.3 Decrease the tolerance to `RelTol` =1e-6 and `AbsTol` = 1e-9.

3.4 How much time and how many steps are needed now?

You will now use the time and the number of steps obtained in 3.4 as the norm to which you compare the performance of the other solvers when the system is made increasingly stiffer.

Questions and tasks:

3.5 Simulate the model using ode15s, ode23s and ode23tb. What are the simulation times compared with ode45 (e.g. t_ode15s/t_ode45)?

3.6 How many steps are used in the simulations, respectively. Set up a table and provide all results for simulation time and steps for comparison.

4. Stiff system

With V_1=10 and V_2=200, the system is somewhat stiff but by decreasing V_1, the system can be made much stiffer.

Questions and tasks:

4.1 Let V_1 be 1, 0.1, 0.01 and see how this affects the simulation time for ode45, ode15s and ode23s. (Depending on which version of Matlab you use you may get different results and you may need to reduce V_1 much more to see the effects). Set up a table and show all results.

4.2 How does the stiffness of the system affect the ode45 simulations?

4.3 How does the stiffness affect the ode15s and ode23s simulations?

4.4 Does there seem to be a limit to how small V_1 can be when using ode15s and ode23s?

5. DAE system

It is possible to rewrite the system as a DAE system as described in "Differential Algebraic Equations". This will make the system a first order system.

Questions and tasks:

5.1 Write a new m-file with a DAE description of the system.

5.2 Compare the simulation times now (ode45, ode15s, ode23s). Set up a table and show all results. Which one is the best solver? Discuss your choice.

Description of the solvers in Matlab

`ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver - in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$. In general, `ode45` is the best function to apply as a "first try" for most problems.

`ode23` is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than `ode45` at crude tolerances and in the presence of moderate stiffness. Like `ode45`, `ode23` is a one-step solver.

`ode113` is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than `ode45` at stringent tolerances and when the ODE file function is particularly expensive to evaluate. `ode113` is a multistep solver - it normally needs the solutions at several preceding time points to compute the current solution.

`ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs). Optionally, it uses the backward differentiation formulas (BDFs, also known as Gear's method) that are usually less efficient. Like `ode113`, `ode15s` is a multistep solver. Try `ode15s` when `ode45` fails, or is very inefficient, and you suspect that the problem is stiff, or when solving a differential-algebraic problem.

`ode23s` is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems for which `ode15s` is not effective.

`ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping. `ode23t` can solve DAEs.

`ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like `ode23s`, this solver may be more efficient than `ode15s` at crude tolerances.

TASK 2 – Multivariate monitoring

The data used for the exercise come from a simulation model of a wastewater treatment plant. We use a model since this can easily be manipulated to incorporate faults and disturbances. The model is a non-linear model with 100s of states, which ensures that the exercise is not trivial but quite realistic. This exercise is more of a tutorial than a conventional computer exercise because we will only be able to be relatively shallow. If you find this interesting, the algorithms provided in the downloadable file can be used on any data.

Preparation

Read Section 11.9 on multivariate monitoring in the textbook Olsson-Rosen.

1. Initiation

Load the data by typing:

```
>> initiate_ex
```

The data being loaded to the workspace contain measurements from two periods of operation. The first period is considered as normal, that is, the process is producing desired outputs and no disturbances are present. The second period contains disturbances and faults.

2. Data pre-treatment

We start by plotting data from the first period, which will be used to identify a monitoring model:

```
>> plot(X1)
```

As can be seen, the data are very different in terms of amplitude. Some data are in the range [0 10] and some are in the range [0 10000]. This is a problem when the variables are compared. If a variable with large amplitudes is compared with a variable with small amplitudes, there is a risk that too much emphasis is put on the large variable. This is further illustrated by the arbitrary choice of engineering units. For instance 1000 m^3/h, 278 l/s and 6.33 MG/day (U.S. Gallons) all describes exactly the same flow rate but have significantly different amplitudes. The solution is to normalise the numbers so that they are comparable. The most basic form of normalisation and a method often used when no other information is available is so called autoscaling. This means that the variable is normalised using its standard deviation: $x_{scale} = x / std(x)$. Also, in multivariate monitoring it is an advantage if the data is mean-centred, i.e. the mean of the variable is subtracted from the values. You can do both manipulations by:

```
>> [ax1, mx1, stdx1]=auto(X1);
```

where `ax1` is the normalised data matrix, `mx1` is the mean values of the variables in `X1` and `stdx1` is the standard deviation of the variables in `X1`. You can check the results by typing:

```
>> mean(ax1)
>> std(ax1)
```

The mean values should be zeros and the standard deviation should be ones, respectively. If you now plot the data

```
>> plot(ax1)
```

you will see that all variables are in the same range and the means of all variables are zero.

3. Model identification

You will now identify a principal component model. The mathematics behind this is relatively simple and is described in the book (p. 426 ff). But to calculate the statistical limits and to plot the explained variability of each principal component (PC) is somewhat cumbersome. So, instead we use a pre-defined function:

```
>> [T1, P1, ssq1, Q1, Qlim1, HT2lim1, HT21] = pca(ax1);
```

The output of the procedure is: `T1` – the data projected on the PCs (also called scores); `P1` – the model (also called loadings); `Q1` – the model residual; `Qlim1` – statistical limit for the residual; `HT21` – Hotelling's T^2 measure; `HT2lim1` – statistical limit for T^2. `ssq1` is a matrix with basically the same information as in the table provided by the procedure as you will see later.

The first question asked in the procedure is how many PCs you would like to keep. The idea of PCA is that the multidimensional original data matrix (`X1`, or `ax1`) is projected onto a smaller coordinate system defined by the PCs. By looking at the plot and the table provided by the procedure, the number of PCs retained in the model must be decided. There are many ways to decide. One basic option when dealing with autoscaled data is the so-called quick-and-dirty method. This means that the first PC with an eigenvalue smaller than 1 is selected as the last PC.

Questions and tasks:

3.1 How many PCs should you retain if you use the quick-and-dirty method and what total percentage of explained variability does this number correspond to?

Type the number of PCs retained and the procedure will continue. Look at the plots provided by the procedure and hit space bar to go from plot to plot. To see how much of the variability of the original data is retained in the model, type:

```
>> plot(ax1,'r'), hold on
>> plot(ax1*P1*P1','k') % do not forget the transpose!
```

Zoom in on the result and note how much of the variability in the original data still remains.

4. Project new data

You now have a model, on which new data (`X2`) will be projected and used to monitor the succeeding two weeks of operation. There are some problems during this week and your task is to detect these problems and isolate the failures. But before you project the new data on the model, it must be pre-treated in the same way as the model data. This means that the mean and the standard deviation of `X1` is used to scale `X2`:

```
>> ax2=scale(X2, mx1, stdx1);
```

Questions and tasks:

4.1 What are the mean values and the standard deviations of `ax2`?

4.2 Why are they not zero and one, respectively?

New data can now be projected onto the model. This is a straightforward task:

```
>> T2 =X2*P1;
```

However, to view the new projections with statistical limits type (note that the input arguments of the procedure are, apart from the new data, the model and the statistical limits from the identification):

```
>> [T2, Q2, HT22]=pcapro(ax2, P1, ssq1, Qlim1, HT2lim1);
```

Hit the space bar to go from plot to plot (remember to type `>> hold off` after task 3 first).

5. Detection

In an on-line situation, the projection of new data onto the model can be done when each new sample of data is obtained. Since you do this off-line, all the data for the new time period are projected at once. However, in principle this is equivalent as long as you do not use "future" information to assess the "current" situation.

First, you will examine the new scores (`T2`). This can be done by using the function

```
>> plotscores(scores, [numbers], ssq1, scoreplot);
```

where *`numbers`* are the PCs you want to look at and *`scoreplot`* is an indicator (yes=1, no=0) if you want to plot the scores as scatter plots (score plots). You can look at the all the scores, but only three scores will be displayed as score plots, i.e. scores plotted against scores. For instance, if you want to plot the first two scores as a scatter plot:

```
>> plotscores(T2, [1 2], ssq1, 1);
```

If you want to plot three scores as time charts:

```
>> plotscores(T2, [1 2 3], ssq1, 0);
```

Another way of detecting deviations is by examining the summarising statistics in the model residuals (`Q2`) and Hotelling's T^2 (`HT22`). Investigate them by plotting them together with their limits, e.g. for `Q2`:

```
>> plot(Q2), hold on, plot([0 1345],[Qlim1 Qlim1]);
```

For both scores and the summarising statistics: as soon as the limits are violated by the measure, a detection is established. However, since these are statistical measures, outliers may exist. Therefore, it is wise to scrutinise the plot and look for systematic violations and other patterns.

Questions and tasks:

5.1 Make sure you look at all the scores by using the `plotscores` function. When would you say there are clear deviations or disturbances? (Hint, use time charts.)

5.2 Plot the first three scores as score plots (scatter plots). (**Provide plot**)

5.3 When you look at Q and T^2 do you find some other disturbances or do they confirm you suspicions from the investigation of scores? (**Provide plot**)

5.4 Is there any deviation or disturbance present in the scores that you cannot find in Q and T^2? Would you say that Q and T^2 are adequate indicators of what is happening in the process?

5.5 If you have to choose 5 significant events, which are these?

6. Isolation

You have detected some events. This is important but the task of finding the reason for the disturbances still remains. By using the model "backwards", the contributing variables to the deviations can be isolated. Since Q and T^2 are very effective measures for detection, it is common to only use these. When a detection in either Q or T^2 is established, one might use score plots to further investigate the problem. This is how you are going to isolate the deviations. Plot Q and its limits again (as above). Q is clearly violating its limits at sample numbers 133, 597, 848 and 1050. Now, plot T^2 and its limits. Here it is somewhat harder to find

distinct violations, but samples 139, 456, 605, 850 and 1051 look suspicious. Not surprisingly both measures indicate violation almost at the same time.

Questions and tasks:

6.1 Do the events listed above (i.e. samples 133, 597 and so on) agree with the 5 significant disturbances you detected?

The function `contrib` can be used to investigate how the variables contribute to the Q and T^2 at certain samples. For instance, type:

```
>> contrib(ax2,T2,P1,ssq1,[133 597],1)
```

to see what variables contribute to the deviation in Q at samples 133 and 597. The last argument is the type of plot you want. Contributions to Q = 1 and contributions to T^2 = 2.

Questions and tasks:

6.2 Use the function `contrib` to investigate the violations of the statistical limits for Q and T^2, respectively. (**Provide a plot** for Q contributions at sample 848)

6.3 Which variables seem to cause the 5 disturbances? Can you find an agreement between the contributions to Q and T^2? In some cases this may be difficult since the timing of the detections are not identical.

You have now used multivariate statistical process control (MSPC) for process monitoring. As you have probably figured out, it is far from trivial and experience is needed to make the most out of the method. However, in comparison with normal SPC, MSPC is far less complex when the number of variables is great.