MatDyn
Version 1.2

Stijn Cole
matdyn@gmail.com
ESAT-ELECTA
Katholieke Universiteit Leuven
Kasteelpark Arenberg 10
3001 Heverlee
Belgium

February 4, 2010

Contents

1   Introduction

MatDyn is a free Matlab based open source program to perform dynamic analysis of electric power systems. It is inspired by MATPOWER, a power flow and optimal power flow program in Matlab and shares its philosophy: "It is intended as a simulation tool for researchers and educators that is easy to use and modify." [1]. The source code of *MatDyn* is available [2]. Care has been taken to keep it well structured and easy to understand.

2   License

- *MatDyn* is free and open source software.

- *MatDyn* may be modified for personal use provided this license remains in force.

- *MatDyn* comes with no warranty whatsoever; not even the implied warranty of merchantability or fitness for a particular purpose.

- Any publications derived from the use of *MatDyn* must acknowledge *MatDyn*.

- *MatDyn* may not be redistributed without prior written permission.

- Modified versions of *MatDyn*, or works derived from *MatDyn*, may not be distributed without prior written permission.

Part I

User's Manual

3   Installation

The prerequisites for *MatDyn* are the following:

- Matlab must be installed

- MATPOWER must be installed and added to the Matlab path. It is available from [3].

- *MatDyn* must be installed and ideally added to the Matlab path. It is available from `www.esat.kuleuven.be/electa/teaching/matdyn/` [2].

You are now ready to run a first simulation.

4   Running a Simulation

Dynamic simulations are run by calling the `rundyn` function:

```
>> [Angles,Speeds,Eq_tr,Ed_tr,Efd,PM,Voltages,Stepsize,...
Errest,Time]=rundyn(PFFUN,DYNFUN,EVFUN,OPTIONS);
```

where PFFUN is a MATPOWER power flow data m-file or struct, DYNFUN an m-file or struct with dynamic data, and EVFUN an m-file or struct with the events such as faults. The OPTIONS vector is optional.

Examples: a simulation can be run by entering the following command at the Matlab prompt:

```
>> rundyn( 'case9','case9dyn','fault');
```

or

```
>> rundyn( 'casestagg','casestaggdyn','staggevent');
```

The EVFUN argument also accepts an empty matrix. The steady-state solution is then obtained:

```
>> rundyn( 'case9','case9dyn','[]');
```

5   Input Data

To perform a dynamic simulation, three m-files or structs have to be defined: one for power flow data, another for dynamic data and a last one for event data.

5.1   Power Flow Data

MatDyn uses the MATPOWER format for the power flow data. The matrices areas and gencost are not used by MatDyn. Consult the MATPOWER manual for more information on the power flow data format [1]. Alternatively, the command `help caseformat` gives a description of the MATPOWER power flow data format.

5.2   Dynamic Data

The dynamic data consist of general data, generator data, exciter data, and governor data. The m-file returns the matrices gen, exc, gov and the scalars freq, stepsize and stoptime. Alternatively, a struct can be defined as follows:

```
DYNFUN = struct( 'gen',gen,'exc',exc,'gov',gov,'freq',freq,...
'stepsize',stepsize,'stoptime',stoptime);
```

5.2.1   General Data

System frequency has to be defined, as well as the step size of the integration algoritm and the stoptime. The starttime is hard-coded and equal to $-0.02s$. The step size is only used by fixed step size algorithms (Table 1).

Table 1: General data

freq	network frequency [Hz]
stepsize	stepsize of the integration algorithm [s]
stoptime	stoptime of the simulation [s]

5.2.2   Generator Data

The generator data is defined in the matrix `gen`. For the 4th order model, model 2, the columns are defined in Table 2. For the classical generator model, model 1, the

columns are defined in Table 3. The generators are assumed to be entered from lowest to highest bus number: the generator parameters of the first row are thus from the generator connected at the bus with the lowest bus number.

Table 2: Generator data format

4th order model	
1	*genmodel*, generator model
2	*excmodel*, exciter model
3	*govmodel*, governor model
4	H, inertia constant (p.u.)
5	D, damping constant (p.u.)
6	x_d, d-axis reactance (p.u.)
7	x_q, q-axis reactance (p.u.)
8	x_d', d-axis transient reactance (p.u.)
9	x_q', q-axis transient reactance (p.u.)
10	T_d', d-axis time constant (s)
11	T_q', q-axis time constant (s)

Table 3: Generator data format

classical model	
1	*genmodel*, generator model
2	*excmodel*, exciter model
3	*govmodel*, governor model
4	H, inertia constant (p.u.)
5	D, damping constant (p.u.)
6	x, reactance (p.u.)
7	x', transient reactance (p.u.)

5.2.3 Exciter Data

The exciter data is defined in the matrix `exc`. The matrix has as many rows as there are generators. The definition of the columns depends on the exciter model. The data in Table 4 specifies the parameters of exciter model 2, the IEEE DC1A excitation system (Fig. 4). Exciter model 1 means constant excitation. Only the first column, the generator number, has to be specified.

Table 4: Exciter data format

IEEE DC1A model	
1	gen, number of the generator
2	K_a, amplifier gain
3	T_a, amplifier time constant
4	K_e, exciter gain
5	T_e, exciter time constant
6	K_f, stabilizer gain
7	T_f, stabilizer time constant
8	A_{ex}, parameter saturation function
9	B_{ex}, parameter saturation function
10	U_{rmin}, lower voltage limit
11	U_{rmax}, upper voltage limit

5.2.4   Governor Data

The turbine and governor data is defined in the matrix `gov`. The matrix has as many rows as there are generators. The definition of the columns depends on the turbine and governor model. The data in Table 5 specifies the parameters of the general IEEE speed governor system of Fig. 5). Governor model 1 means that the generator is driven be a turbine with constant mechanical output power. Only the first column, the generator number, has to be specified.

Table 5: Governor data format

General IEEE governor model	
1	gen, number of the generator
2	K, droop
3	T_1, time constant
4	T_2, time constant
5	T_3, servo motor time constant
6	P_{up}, upper ramp limit
7	P_{down}, lower ramp limit
8	P_{max}, maximal turbine output
9	P_{min}, minimal turbine output

5.3   Event Data

The event data file defines three matrices: event, buschange, and linechange. Alternatively, a struct can be used as input argument. It has to be defined as follows:

```
EVFUN = struct( 'event',event,'buschange',buschange,'linechange',...
'linechange');
```

The event matrix contains all events that take place during the simulation (Table 6). The first column defines the instant of the event, the second the type of the event.

Table 6: Event data format

Event data format
1   *time*, instant of change (s)
2   *eventtype*

Table 7 shows the available event types. The program can be called with an empty event matrix to obtain a steady-state solution.

Table 7: Event types

Event types	
1	change bus parameters
2	change line parameters

5.3.1   Bus change

The bus parameters can be changed during simulation by defining the buschange matrix (Table 8). Consult the MATPOWER manual for a list of bus parameters.

Table 8: Bus change

Buschange data format
1   *time*, instant of change (s)
2   *bus*, bus number
3   *parameter*, bus parameter to change
4   *newvalue*, new parameter value

5.3.2   Line change

The line parameters can be changed during simulation by defining the linechange matrix (Table 9). Consult the MATPOWER manual for a list of line parameters.

Table 9: Line change

Linechange data format
1   *time*, instant of change (s)
2   *line*, line number
3   *parameter*, line parameter to change
4   *newvalue*, new parameter value

5.3.3   Example: Bus faults

Three-phase bus faults can be simulated by changing the shunt susceptance of the bus in a bus change event. Table 10 gives the data for a zero impedance bus fault at bus 2 at $t = 0$. The fault can be cleared by resetting the susceptance to its original value.

Table 10: Bus fault

Buschange data
1   *time*, 0
2   *line*, 2
3   *parameter*, 6
4   *newvalue*, -1e10

5.4   Options

MatDyn accepts an option vector as an optional fourth input argument (Table 11). If the option vector is not specified, the default options are used.

- method
  Selects the integration method. 1: Modified Euler, 2: Runge-Kutta, 3: Runge-Kutta Fehlberg, 4: Higham and Hall, 5: Modified Euler with interface error control.

Table 11: Options

Default Options		
1	method	1
2	tol	1e-4
3	minstepsize	1e-3
4	maxstepsize	1e2
5	output	1
6	plots	1

- tol
  Specify the tolerance of the error. This argument is only used for the Runge-Kutta Fehlberg and Higham and Hall methods.

- minstepsize
  Sets the minimal step size. Only used by the adaptive step size algorithms: Runge-Kutta Fehlberg and Higham and Hall methods.

- maxstepsize
  Sets the maximal step size. Only used by the adaptive step size algorithms: Runge-Kutta Fehlberg and Higham and Hall methods.

- output
  Prints progress info if set to one, prints no progress info if set to zero. Errors are printed anyhow.

- plots
  Draws plots if set to one, draws nothing if set to zero.

The function mdoption returns the default option vector:

```
>> mdopt = mdoption

mdopt =

    1.0000
    0.0001
    0.0000
  100.0000
    1.0000
    1.0000
```

Remarks: The adaptive step size methods start with minimal step size. It is of interest to increase minimum step size as it speeds up the calculations. Generally, tolerance must be increased as well, or the integration will fail.

6   Output Data

`rundyn` returns the matrices given in Table 12.

Table 12: MatDyn output

Angles	Generator angles in degrees
Speeds	Generator speeds/synchronous speed
Eq_tr	Q-axis component of the voltage behind transient reactance in p.u.
Ed_tr	D-axis component of the voltage behind transient reactance in p.u.
Efd	Excitation voltage in p.u.
PM	Mechanical output power of the turbine in p.u.
Voltages	Network voltages $U \angle \delta$ in p.u. and radians
Stepsize	Step size of the integration
Errest	Estimation of the error (Only defined for the adaptive step size methods, Runge-Kutta Fehlberg and Higham and Hall. Set to zero for Modified Euler and standard Runge-Kutta methods.)
Time	Vector of time steps in seconds

7   User Defined Models

In *MatDyn* it is possible to include new models for generators, exciters, and governors. As an example we show how to include a simplified IEEE AC4A exciter in *MatDyn*. Only two files, `Exciter.m` and `ExciterInit.m`, must be changed.

Step 1: the differential equations

This example is for illustrative purposes only: the limiters are neglected here, which is wrong, but simplifies the example. For realistic results, limiters should be

modelled, as is done for the IEEE DC1A exciter model. The block diagram of the simplified IEEE AC4A exciter is shown in Fig. 1. It is easier to write down the equations when the block diagram is transformed to the equivalent block diagram shown in Fig. 2. The following three equations describe the dynamic behaviour of the exciter:

$$\dot{x} = \frac{1}{T_B}(U_{ref} - U - x), \tag{1}$$

$$U_r = x + \dot{x}T_C, \tag{2}$$

$$\dot{E_{fd}} = \frac{1}{T_A}(K_A U_r - E_{fd}). \tag{3}$$

Substituting (1) in (2) and the result thereof in (3) leads to the following two differential equations, that completely describe the system:

$$\dot{x} = \frac{1}{T_B}(U_{ref} - U - x), \tag{4}$$

$$\dot{E_{fd}} = \frac{1}{T_A}(K_A(x + \frac{T_C}{T_B}(U_{ref} - U - x)) - E_{fd}). \tag{5}$$

Step 2: the initial conditions

The initial conditions can be found by solving the equations obtained when the right-hand side of (4) and (5) are set to zero:

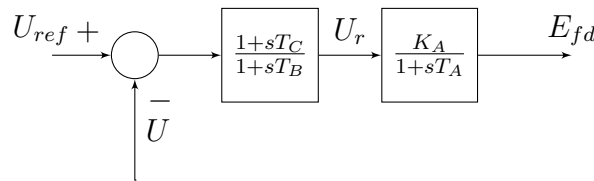$$0 = \frac{1}{T_B}(U_{ref} - U - x), \tag{6}$$
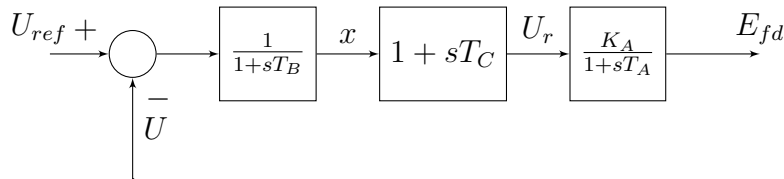

Figure 1: IEEE type AC4A excitation system

Figure 2: IEEE type AC4A excitation system

$$0 = \frac{1}{T_A}(K_A(x + \frac{T_C}{T_B}(U_{ref} - U - x)) - E_{fd}). \tag{7}$$

Solving for x and U_{ref} gives:

$$x = \frac{E_{fd}}{K_A}, \tag{8}$$

$$U_{ref} = U + x. \tag{9}$$

Step 3: insert initial conditions in *MatDyn*

Open `ExciterInit.m` and add a third exciter model:

```
%% Define exciter types
...
type3 = d(exctype==3);
```

The model has two state variables, E_{fd}, and x, and five parameters, K_A, T_A, T_B, and T_C. The excitation voltage, E_{fd}, must always be the first state variable. For governor models, P_m has to be the first state variable, and for generator models, δ must be the first state variable, and ω the second. The terminal voltage U is not a state variable nor a parameter, but needs to be accessible. It must therefore be appended to the variables matrix.

```
%% Exciter type 3: IEEE AC4A
Efd0 = Xexc(type3,1);

Ka = Pexc(type3,2);
Ta = Pexc(type3,3);
Tb = Pexc(type3,4);
Tc = Pexc(type3,5);

U = Vexc(type3,1);

x = Efd0./Ka;
Uref = U + x;

Xexc0(type3,1:2) = [Efd0, x];
Pexc0(type3,1:6) = [Pexc(type3,1), Ka, Ta, Tb, Tc, Uref];
```

Step 4: insert dynamic equations in *MatDyn*

Open `Exciter.m` and add a third exciter model:

```
%% Define exciter types
...
type3 = d(exctype==3);
```

The dynamic equations are then inserted:

```
%% Exciter type 3: IEEE AC4A
Efd = Xexc(type3,1);
x = Xexc(type3,2);

Ka = Pexc(type3,2);
Ta = Pexc(type3,3);
Tb = Pexc(type3,4);
Tc = Pexc(type3,5);
Uref = Pexc(type3,6);

U = Vexc(type3,1);

dx = 1./Tb.*(Uref - U - x);
dEfd = 1./Ta .* ( Ka .* (x + Tc./Tb.*(Uref - U - x)) - Efd);

F(type3,1:2) = [dEfd dx];
```

Step 5: insert parameter values in the dynamic data file

The parameters of the IEEE DC1A model are entered by appending the row [gen Ka Ta Tb Tc] to the *exc* matrix in the dynamic data file. Suppose the first and third generator are equipped with the IEEE DC1A exciter and the second one with the IEEE AC4A exciter. The *exc* matrix is then defined as follows:

```
exc=[1 Ka Ta Ke Te Kf Tf Aex Bex Urmin Urmax;
     2 Ka Ta Tb Tc 0  0  0   0   0     0;
     3 Ka Ta Ke Te Kf Tf Aex Bex Urmin Urmax]
```

New governor models can be added similarly. Adding generator models requires more changes because they contain algebraic equations. The algebraic equations are solved in `MachineCurrents.m` and `SolveNetwork.m`. These two files have to be changed additionally.

8   Errors and Warnings

Errors are fatal and cause the program to exit. If a warning occurs, the program will continue but it is strongly advised to recheck data.

- Warning: transient saliency not supported
  The d-axis transient reactance x_d' must be equal to the q-axis transient reactance x_q'. This condition is checked for all generators, except those represented by the classical model, and enforced by setting $x_q' = x_d'$.

- Error: Power flow did not converge
  The dynamic simulation is only started from a steady-state condition. The power flow data should be checked.

- Error: Generator\Exciter\Governor not in steady-state
  The dynamic simulation is only started from a steady-state condition. First of all, it should be checked that the initial conditions are calculated correctly. If this is the case, the error is caused by the violation of a limit in the definition of the equipment. The data in DYNFUN should be checked: a generator's reactance that is completely off could for instance lead to an unrealistical value of excitation voltage. The power flow solution should be checked as well, as an abnormally high voltage at a generator bus could for instance impose a very high excitation voltage.

- Error: No solution found with minimum step size
  The integration alghorithm failed. Try reducing minimum step size or using a fixed step size method.

Part II

Technical Reference

9 Program Flow

The power system is represented by a system of differential-algebraic equations (DAE).

$$\dot{X} = F(X, Y, P) \tag{10}$$

$$0 = G(X, Y, P) \tag{11}$$

Where X are the state variables, Y the algebraic variables, and P parameters. In $MatDyn$, the set of differential equations F consists of the dynamic equations of the generators, exciter, and governors, while the algebraic equations contain the power flow equations and the stator current equations of the generators.

The program flow is represented schematically in Fig. 3. MatDyn uses MATPOWER to obtain a power flow solution. It then calculates and factorises the augmented bus admittance matrix and proceeds with calculating the intial conditions of the generators, exciters, and governors if the power flow converged. If the system is in steady-state, the main loop is started. The set of differential equations F is integrated, and the set of algebraic equations G solved. If an event occurs, the augmented bus admittance matrix is refactorised and the algebraic equations G, consisting of the network equations and stator current equations, are recalculated. The variables of interest are saved and the time is advanced with the optimal stepsize for methods with step size control or with fixed step size for the fixed step size methods.

10 Solution of the DAE System

DAE systems can be solved simultaneously or partitioned. $Matdyn$ uses a partitioned solution scheme. Differential equations of the exciters, governors and generators are solved sequentially, followed by the algebraic equations, resulting in a high readability of the code. In a partitioned solution scheme, F is first solved for X and then G for Y. In solving for X, Y must be known and vice versa. However, these values may not be available accurately, causing a problem known as interface errors, aggravated by using extrapolated values for Y in order to reduce computations. In $MatDyn$, no extrapolation of the Y variables is performed, instead, the Y variables are solved for after every stage of the integration algorithm. This procedure leads to negligible interface errors. The user can verify this by selecting the Modified Euler 2 integration method, which eliminates interface errors.

11 Solving the Differential Equations

MatDyn comes with a number of ODE solvers. The first one is a second order Modified Euler method. The second the well-known fourth order Runge-Kutta method. The third
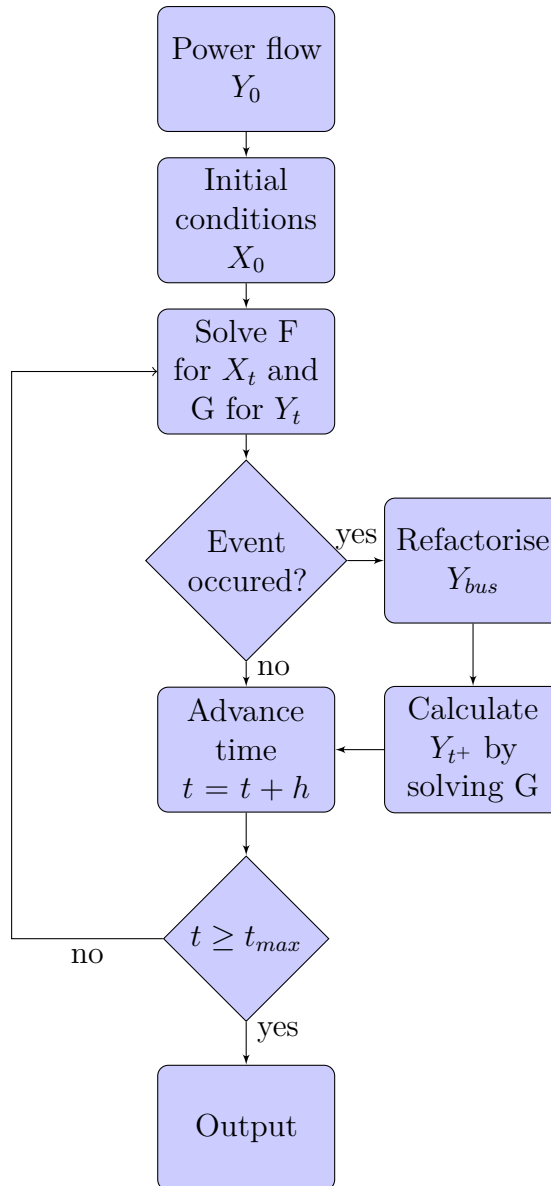
Figure 3: Program flow

and fourth method are variations of the standard Runge-Kutta method: the Runge-Kutta Fehlberg method is a 6 stage embedded method of order 4(5) with step size control, the Runge-Kutta method of Higham and Hall is a variable step size, 7 stage embedded method of order 6(5) with local extrapolation.

11.1 Modified Euler

The modified Euler method is a 2-stage method, which is often used in power system analysis computer programs. [4] The forward Euler method is used to obtain a first approximation $y_{n+1}^{(0)}$. In the second stage, a better approximation $y_{n+1}^{(1)}$ is found by using the average of the derivatives in the beginning and at the end of the interval using the first approximation.

$$y_{n+1}^{(0)} = y_n + hf(y_n) \tag{12}$$

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}(f(y_n) + f(y_{n+1}^{(0)})) \tag{13}$$

11.2 Modified Euler 2

The modified Euler 2 method checks after the second stage whether the interface errors are within bounds. If not, more stages are calculated until the desired tolerance is met.

$$y_{n+1}^{(0)} = y_n + hf(y_n) \tag{14a}$$

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}(f(y_n) + f(y_{n+1}^{(0)})) \tag{14b}$$

$$y_{n+1}^{(2)} = y_n + \frac{h}{2}(f(y_n) + f(y_{n+1}^{(1)})) \tag{14c}$$

$$\vdots$$

$$y_{n+1}^{(i)} = y_n + \frac{h}{2}(f(y_n) + f(y_{n+1}^{(i-1)})) \tag{14d}$$

11.3 Runge-Kutta

An s-stage explicit Runge-Kutta method can be defined as:

$$k_i = f(x_n + c_i h, y_n + \sum_{j=1}^{i-1} a_{i,j-1} k_j), \quad i = 1, \dots, s \tag{15}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \tag{16}$$

The coefficients are given by a Butcher tableau:

$$
\begin{array}{c|ccccc}
0 \\
c_2 & a_{21} \\
c_3 & a_{31} & a_{32} \\
\vdots & \vdots & \vdots & \ddots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

The most well-known Runge-Kutta method is the 4th order, 4-stage method defined by the following Butcher tableau:

$$
\begin{array}{c|cccc}
0 \\
\frac{1}{2} & \frac{1}{2} \\
\frac{1}{2} & 0 & \frac{1}{2} \\
1 & 0 & 0 & 1 \\
\hline
 & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & -\frac{1}{6}
\end{array}
$$

11.4   Runge-Kutta Fehlberg

The Runge-Kutta Fehlberg method is given by the following Butcher tableau:

$$
\begin{array}{c|cccccc}
0 \\
\frac{1}{4} & \frac{1}{4} \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} \\
\hline
y_1 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0 \\
\hat{y}_1 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}
$$

The idea is to calculate two approximations of the solution for every step: one 4th order approximation and one 5th order approximation. The difference between the two solutions is a measure for the local error of the lower order result. This error can be used to determine the optimal step size. If the local error is within specified bounds, the step is accepted and the step size enlarged. If the local error is too large, the step is rejected, the step size is reduced and the calculation is repeated with a smaller step size. The coefficients are optimized to minimize the error of the lower order result [5, p.178]. The lower order result is used in the further calculations, it is thus a 4th order method.

The new step size is calculated as

$$h^{new} = hq \tag{17}$$

with q usually

$$q = 0.84(\frac{tol}{|X_2 - X_1|})^{1/4} \tag{18}$$

To ensure that the step size does not change too fast, $h^{new} = hq$ is replaced by the following formula ([5, p.168]):

$$h^{new} = h \cdot min(facmax, max(facmin, q)) \tag{19}$$

Using variabele step solvers in dynamic power system simulations can lead to wrong results due to the discontinuous behaviour of events such as short-circuits. Since ODE solvers are usually not developed with discontinuities in mind, some code must be added to accomodate for discontinuities. Since large steps can be taken, the solver can jump over discontinuities. Therefore, after every step it is checked whether the optimal step size for the next step will not jump over an event. If this is indeed the case, the step size is reduced.

11.5   Higham and Hall

For stiff systems, the evolution of the step size is oscillating and lots of steps are rejected which wastes computer time. Higham and Hall sought a method with smooth step size changes for stiff systems. The method of Higham and Hall is of order 5. The coefficients seek "reasonable size of the stability domain, large parts of SC-stability and a small 6th order error constant." [6, p.27].

0							
$\frac{2}{9}$	$\frac{2}{9}$						
$\frac{1}{3}$	$\frac{1}{12}$	$\frac{1}{4}$					
$\frac{1}{2}$	$\frac{1}{8}$	0	$\frac{3}{8}$				
$\frac{3}{5}$	$\frac{91}{500}$	$-\frac{27}{100}$	$\frac{78}{125}$	$\frac{8}{125}$			
1	$-\frac{11}{20}$	$\frac{27}{20}$	$\frac{12}{5}$	$-\frac{36}{5}$	5		
1	$\frac{1}{12}$	0	$\frac{27}{32}$	$-\frac{4}{3}$	$\frac{125}{96}$	$\frac{5}{48}$	
y_1	$\frac{1}{12}$	0	$\frac{27}{32}$	$-\frac{4}{3}$	$\frac{125}{96}$	$\frac{5}{48}$	0
$\hat{y}_1$	$\frac{2}{15}$	0	$\frac{27}{80}$	$-\frac{2}{15}$	$\frac{25}{48}$	$\frac{1}{24}$	$\frac{1}{10}$

12 Solving the Algebraic Equations

The bulk of the algebraic equations consist of the power flow equations. After every event and after every stage of the integration algorithm, the network equations have to be solved. This leads to great computational effort, which can be reduced by extrapolating the Y variables but at the cost of aggravating interface errors.

MatDyn does not use extrapolation and computes the Y variables at every stage. However, two condition are imposed, that allow a substantially reduction of computation:

- Generators do not exhibit transient saliency

- Loads are represented by a constant admittance only

Under these two conditions, the network equations are all linear and the voltages can be solved for by solving a linear system of equations, which can be done efficiently in Matlab by LU factorisation of the augmented Y_{bus} matrix.

The power flow Y_{bus} matrix is augmented by adding the generator's stator admittance and the load admittance to its diagonal elements.

13 Models

13.1 Generator

MatDyn uses the well-known classical generator model and a fourth order generator model, considered to be sufficiently accurate for stability studies. [7, p.181] In this model,

damper windings are ignored, but the effect of damping can be accounted for by increasing the damping constant D. The differential equations are:

$$\dot{\omega} = \frac{\pi f}{H}(-D(\omega_0 - \omega) + P_m - P_e), \tag{20}$$

$$\dot{\delta} = \omega_0 - \omega, \tag{21}$$

$$\dot{E_q} = \frac{1}{T'_{d0}}(E_{fd} - E'_q + (x_d - x'_d)I_d), \tag{22}$$

$$\dot{E_d} = \frac{1}{T'_{q0}}(-E'_d - (x_q - x'_q)I_q). \tag{23}$$

The algebraic equations are:

$$I_d = (v_q - E'_q)/x'_d, \tag{24}$$

$$I_q = -(v_d - E'_d)/x'_q, \tag{25}$$

$$P_e = E'_q I_q + E'_d I_d + (x'_d - x'_q)I_d I_q, \tag{26}$$

with

$$v_d = -U \sin(\delta - \theta), \tag{27}$$

$$v_q = U \cos(\delta - \theta). \tag{28}$$

The algebraic equations are part of the set G, and are solved after the network equations.

13.2 Exciter

An IEEE DC1A excitation system representation is used [8]. The time constant T_r of the regulator input filtering is considered to be zero as it is usually very low. The resulting block diagram is shown in Fig. 4. Saturation is represented by the exponential function

$$S(E_{fd}) = A_{ex} \cdot e^{(B_{ex}E_{fd})}. \tag{29}$$

13.3 Turbine and Governor

The governor is represented by a simplified model (Fig. 5). It is an IEEE model of a general speed-governing system for steam turbines. It can represent a mechanical-hydraulic or electro-hydraulic system by the appropriate selection of parameters [9, p.1906].
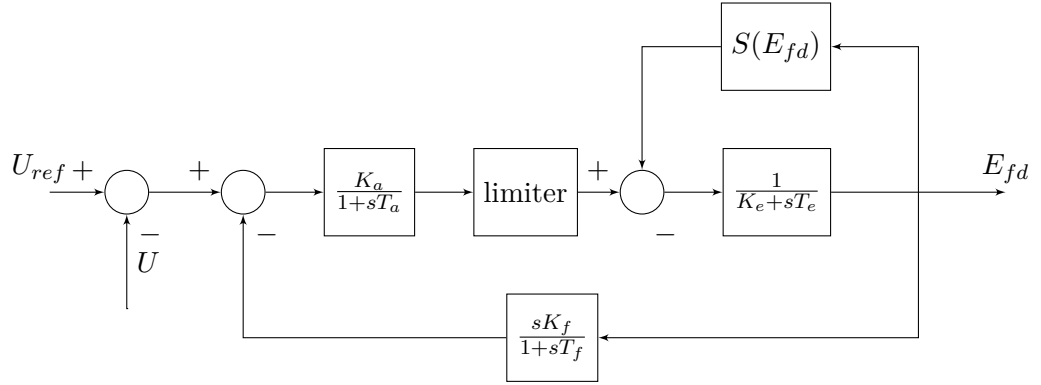
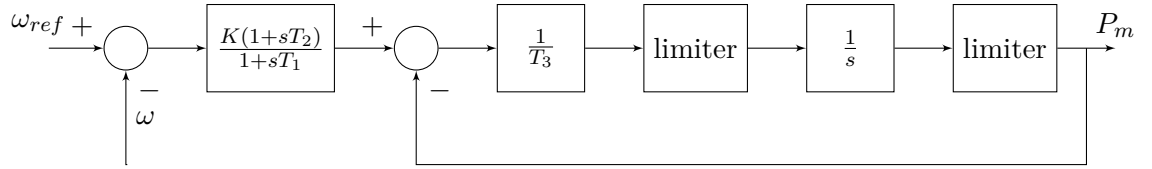Figure 4: IEEE excitation system

Figure 5: IEEE general speed-governing system

13.4   Loads

All loads are modelled as constant admittances.

Part III

Examples

14 Validation

MatDyn is validated by comparing its results with those obtained by using the commercial power system software PSS/E and the open source software package PSAT [10]. A five bus network, 2 generator network has been simulated. The full description of the system, including one-line diagram, system data, and the results can be found in [4]. The three software packages show a very good correspondence. The results do however differ very slightly. This is due to some design decisions. For instance: in PSS/E, speed voltages are taken into account, in *MatDyn* and PSAT, they are not.

15 Event file

The following event file uses all possible events. At time $t = 0.2$, a fault is applied to bus 6. It is cleared at $t = 0.3$. A second fault is applied to bus 5 at $t = 1$, and cleared at $t = 1.1$. At $t = 1.2$ the line parameters of the second branch are changed, and at $t = 1.4$, a new load power is specified at bus 7.

```
function [event,fault,linechange,loadchange] = testall

% event = [time type]
event=[0.2      1
```
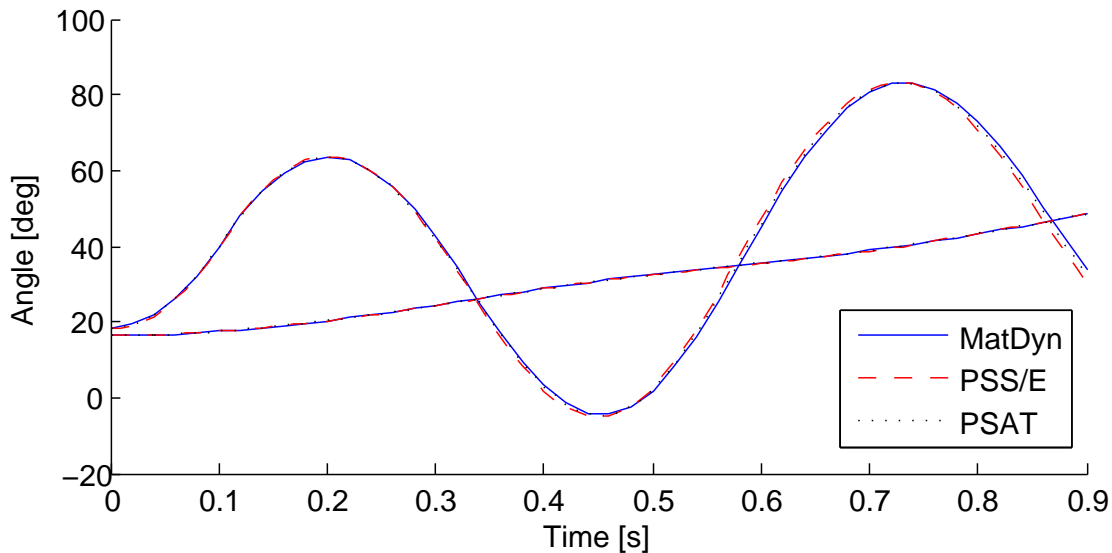

Figure 6: Comparison between *MatDyn*, PSS/E, and PSAT: generator angles

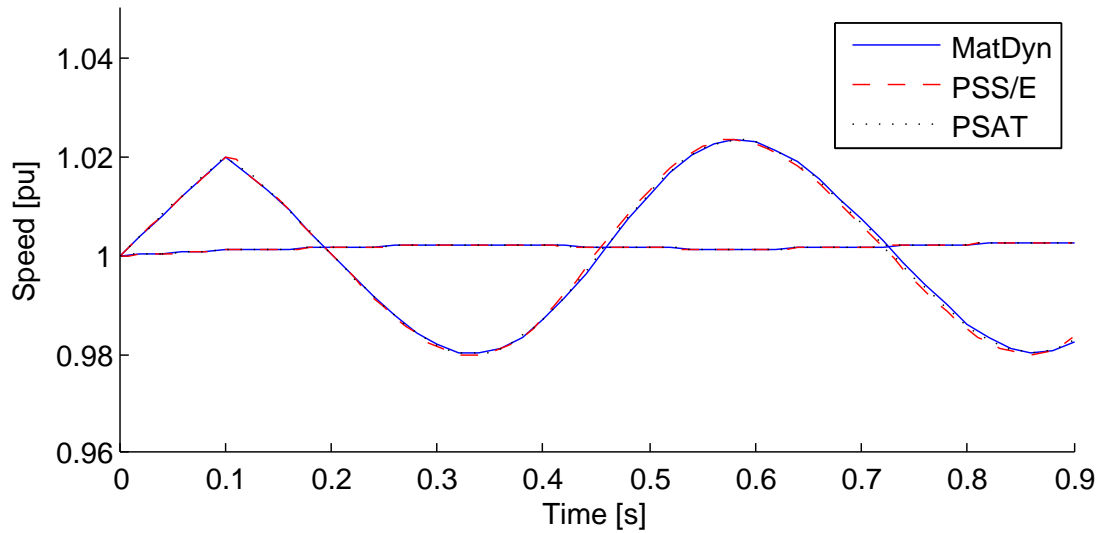Figure 7: Comparison between $MatDyn$, PSS/E, and PSAT: generator speeds

```
        0.3       2
        1         1
        1.1       2
        1.2       3
        1.4       4];

% fault = [time bus]
fault   = [0.2  6
           1    5];

% linechange = [time    branch    R   X   B]
linechange = [1.2   2   0.1 0.2 0.3];

% loadchange = [time    bus     P   Q]
loadchange = [1.4   7   0.9 .5];

return;
```

16 Test

The test files, Test1.m and Test2.m show how Matdyn can be used in an m-file. They can be run by typing 'Test1' or 'Test2' at the Matlab prompt. Test1.m runs sequentially a

simulation with Modified Euler, Runge-Kutta, Runge-Kutta Fehlberg and Runge-Kutta Higham-Hall. The results are displayed on a graph. Test2.m compares the Runge-Kutta Fehlberg method with tolerance 1e-4 and 1e-3 (Fig. 8 and 9).

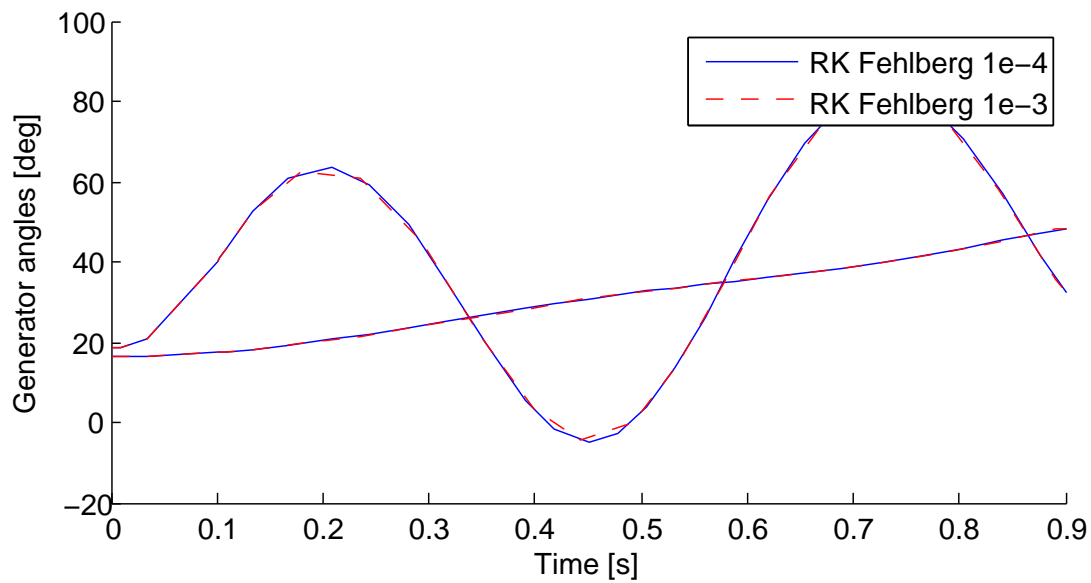Figure 8: Generator Angles Test2.m

17 Casestagg

The command,

```
>> rundyn( 'casestagg','casestaggdyn','fault');
```

simulates a fault on $t = 0$, cleared at $t = 0.1$.
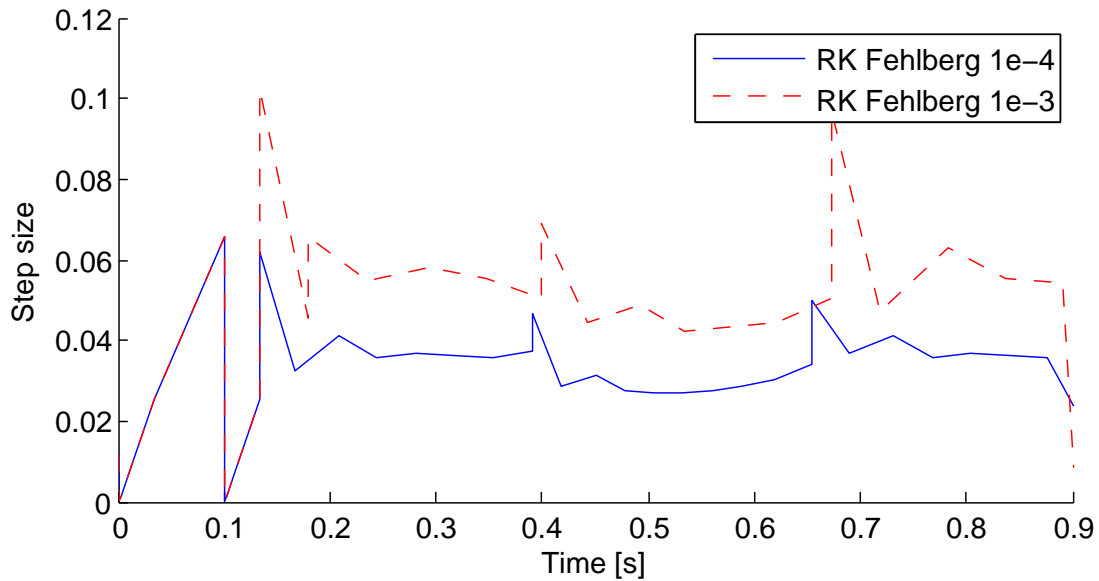
Figure 9: Step Sizes Test2.m

Bibliography

[1] R. D. Zimmerman and C. E. Murrilo-Sánchez. (2007) MATPOWER, version 3.2,
    user's manual. Power system engineering research center, Cornell Univ. Ithaca, NY.
    [Online]. Available: http://www.pserc.cornell.edu/matpower/manual.pdf

[2] MatDyn website. [Online]. Available: http://www.esat.kuleuven.be/electa/teaching/
    matdyn/

[3] MATPOWER website. [Online]. Available: http://www.pserc.cornell.edu/matpower/

[4] G.W.Stagg and A. El-Abiad, *Computer Methods in Power System Analysis*, ser.
    McGraw-Hill Series in Electronic Systems. Tokyo, Japan: McGraw-Hill Kogakusha,
    1968.

[5] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I:
    Nonstiff Problems*, 2nd ed., ser. Springer Series in Computational Mathematics.
    Berlin, Germany: Springer, 2000, vol. 8.

[6] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and
    Differential-Algebraic Problems*, 2nd ed., ser. Springer Series in Computational
    Mathematics. Berlin, Germany: Springer, 2002, vol. 14.

[7] K.R.Padiyar, *Power System Dynamics: Stability and Control.* Kent, UK: Anshan, 2004.

[8] I. C. Report, "Excitation system models for power system stability studies," no. 2, pp. 494–509, Feb. 1981.

[9] ——, "Dynamic models for steam and hydro turbines in power system studies," no. 6, pp. 1904–1915, Nov. 1973.

[10] F. Milano, "An open source power system analysis toolbox," vol. 20, no. 3, pp. 1199–1206, Aug. 2005.