

Diana Tutorial*

Version 0.8.2

Michael Krasnyk

August 21, 2009

Contents

| | | |
|----------|--------------------------------------|----------|
| 1 | Introduction | 2 |
| 1.1 | Some useful links | 2 |
| 2 | Model generation | 2 |
| 3 | Using Diana | 3 |
| 3.1 | Loading models and solvers | 3 |
| 3.2 | Dynamic simulation | 4 |
| 3.3 | Continuation | 5 |

Version Id: tutorial.tex 8180 2009-03-18 17:41:58Z miha (GMT) compiled August 21, 2009 4:32

*This is only draft version of the tutorial, that contains only some scripts without full explanation of methods and parameters. Complete version of the tutorial, an user guide and a reference manual will be provided after final release of the Diana

1 Introduction

Before using Diana it is necessary to update binary paths:

```
setenv PATH /usr/local/diana/bin:$PATH
setenv LD_LIBRARY_PATH /usr/local/diana/lib:$LD_LIBRARY_PATH
rehash
```

1.1 Some useful links

- the CAPE-OPEN Laboratories Network
- Python Tutorial

2 Model generation

Further examples will be based on the model of the continuous flow stirred tank reactor [1]. Let's copy model file to the local directory:

```
mkdir -p ~/work/models/cstr
cd ~/work/models/cstr
cp /usr/local/diana/tmp/build/diana/test/models/HafkeReactor/hafke.mdl .
```

Script `mdl2diana` generates C++ sources of the model described in the MDL file. It is possible to call script in two different ways:

```
mdl2diana -s <script>
```

or

```
mdl2diana <module> [-f <mdl-file>] [-d <diana-name>]
               [-g <generation-dir>] [-c] [-o <optflags>]
               [-sd <pairs of integers>]
```

Command line options of the script:

- `-s <script>` script containing all options to produce the model
- `<module>` name of the module to write.
- `-f <mdl-file>` optional mdl-file with definition of the module (default value is `<module>.mdl`).
- `-d <DianaName>` optional name for the model in Diana (default value is `<module>` translated to CamelCase).
- `-g <generation-dir>` directory to write files to (default is current work directory).
- `-c` compile generated sources with `dianac` immediately.
- `-o <optflags>` colon-separated list of processing options like `:eliminatehelps:assignmentpasses=2` (default values are taken from `$(HOME)/.promot-options`).
- `-sd <pairs of integers>` symbolic differentiation of the model (default value is 1 0). This option specifies orders of symbolic derivatives that will be generated by Maxima. For model $G(x, \dot{x}, t, p)$ pair of integers i, j corresponds to the symbolic derivative $\frac{\partial^{i+j} G(x, \dot{x}, t, p)}{\partial x^i \partial p^j}$.

For `hafke.mdl` possible to produce source files with command:

```
mdl2diana Hafke_Reactor -f hafke.mdl -sd 1 0
```

Compilation of the c++ sources possible with the script `dianac`:

```
dianac [options] [model_name]
```

where `model_name` is the name of the model source files generated by the script `mdl2diana`. And options are:

- `-version` show program version and exit
- `-h`, `-help` show help message
- `-c`, `-clean` clean model directory
- `-r`, `-rebuild` rebuild model
- `-O OPT` gcc optimization level (for example `-O 0`,..., `-O 3`)

So preparing model can be finished by command:

```
dianac HafkeReactor
```

Successfully generated and compiled model has this set of files:

```
HafkeReactor.cpp   HafkeReactor.jac.cpp   HafkeReactor.lisp   HafkeReactor.xml
HafkeReactor.hpp   HafkeReactor.jac1.cpp   HafkeReactor.so     hafke.mdl
```

3 Using Diana

3.1 Loading models and solvers

Diana script is the wrapper around Python and has possibility to specify Python options, so command

```
diana -h
```

produces help output for the Python. Also useful Python option `-i` that allows Python to stay in the interactive mode after running script.

For using Diana you need to import diana module:

```
1 import diana
```

`DianaMain` allows to get references on model manager and solver factory:

```
1 import diana
2 dmain=diana.GetDianaMain()
3 mmanger=dmain.GetModelManager();
4 sfactory=dmain.GetSolverFactory();
```

And finally possible to load model and integrator into Python:

```
1 import diana
2 dmain=diana.GetDianaMain()
3 # get Diana model manger
4 mmanger=dmain.GetModelManager();
5 # create model
6 model=mmanger.CreateModel(diana.CAPE_CONTINUOUS, "HafkeReactor.so");
7 model.Initialize();
8 # create solver
9 sfactory=dmain.GetSolverFactory();
10 solver=sfactory.CreateSolver(diana.CAPE_DAE, model, "ida.so");
11 solver.Initialize();
```

3.2 Dynamic simulation

Dynamic simulation can be shown by the next script `dyn.py`.

```
1 import diana
2 try:
3     dmain=diana.GetDianaMain()
4     # get Diana model manger
5     mmanger=dmain.GetModelManager();
6
7     # create model
8     model=mmanger.CreateModel(diana.CAPE_CONTINUOUS , "HafkeReactor.so");
9     model.Initialize();
10    eso=model.GetActiveESO();
11    esopar=eso.GetParameters();
12
13    ri=dmain.CreateReportingInterface("basic");
14    ri.SetComponentName("data");
15
16    # create solver
17    sfactory=dmain.GetSolverFactory();
18    solver=sfactory.CreateSolver(diana.CAPE_DAE, model, "ida.so");
19    solver.Initialize();
20    solpar=solver.GetParameters();
21    solver.SetReportingInterface(ri);
22    ri.Add(solpar["T"]);
23    ri.Add(eso.GetStateVariables().ItemByName("tk"));
24
25    solpar["VerboseLevel"].SetValue(2);
26    solpar["Tend"].SetValue(16000);
27    solver.Solve();
28    # create directory and store results
29    import os
30    if not os.path.isdir('results'): os.mkdir('results');
31    ri.WriteData("results/dyn");
32    ri.WriteDataMatlab("results/dyn.m");
33 except diana.ECapeUser, exc:
34     print exc
```

Lines 2,4,6,9,10 initialise model as described above. Model has equation set object, that possible to get by `GetActiveESO()` method (line 11) On the line 12 ESO method `GetParameters()` returns collection of ESO parameters. Diana parameters are real, integer and structure parameters, help variables of the ESO and additional parameters for the Jacobian finite differences calculation. This parameters are:

- SymbolicJacobian boolean parameter. True value means calculation of Jacobi matrix by symbolically generated code. When the value equal False, then finite difference scheme will be used (default value True).
- FDDPartition boolean flag specifies whether matrix partitioning will be used (default value True).
- FDDOrder order of the finite difference scheme (possible values 1 and 2, default value 2).
- FDEpsilon difference size (default value 10^{-8}).

Parameters can be accessed by methods `ItemByName` and `ItemByIndex`, for example:

```

1 esopar.ItemByName("tkzu").GetValue()
2 esopar["tkzu"].SetValue(300.0)
3 esopar.ItemByIndex(5).GetComponentName()
4 esopar[5].Specification().DefaultValue()

```

The next lines are creating of the reporting interface and solver. After calling method `SetReportingInterface`, possible to initialise the reporting interface by observable parameters or state variables:

```

1 solver.SetReportingInterface(ri);
2 ri.Add(solpar["T"]);
3 ri.Add(eso.GetStateVariables().ItemByName("tk"));

```

After setting solver parameters possible to call method `Solve` and after save results:

```

1 solver.Solve();
2 ri.WriteDataMatlab("dyn.m");

```

Basic solver parameters:

- T current value of the independent variable.
- T0 initial value of the independent variable.
- Tend final value of the independent variable.
- Tout time interval for the equidistant output mode.
- RTol the relative error tolerance

It is possible to run `dyn.py` script with command:

```
diana dyn.py
```

Produced Matlab data file can be used to plot results:

```

clear('global');
run('dyn');
plot(data1(:,1),data1(:,2));
xlabel('T, sec'); ylabel('Tk, K');

```

3.3 Continuation

Steady state continuation with stability analysis can be presented by the following script:

```

1 import diana
2 try:
3     # get diana main and load Hafke reactor model
4     dmain=diana.GetDianaMain()
5     mmanger=dmain.GetModelManager();
6     model=mmanger.CreateModel(diana.CAPE_CONTINUOUS, "HafkeReactor.so");
7     model.Initialize();
8     eso=model.GetActiveESO();
9     # initialize model variables and parameters
10    esopar=eso.GetParameters();
11    eso.SetAllVariables([4.3796022e+01, 6.4482417e+01, 1.3579274e+03,
12                        5.6989992e-01, 3.6715772e+02, 3.2913593e+02]);
13    esopar["q"].SetValue(1.84e-06);
14    esopar["tkzu"].SetValue(2.9165e+02);
15    esopar["tzu"].SetValue(2.9365e+02);

```

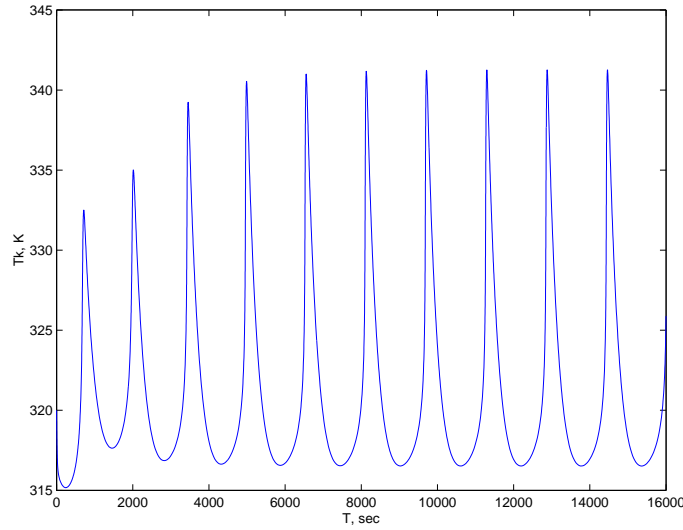


Figure 1: Plot of the dynamic simulation $Tk(t)$

```

16  esopar["c_zu"].SetValue([3.0e+03, 0.0e+0, 0.0e+0, 6.54e+0]);
17  esopar["c_f_ges"].SetValue(6.54e+0);
18  esopar["dhr"].SetValue([3.025e+02, 3.859e+02, 9.51e+01]);
19  esopar["e"].SetValue([1.055e+02, 1.262e+02, 8.833e+01,
20                        5.569e+01, 4.504e+01]);
21  esopar["k0"].SetValue([3.7e+13, 4.83e+14, 1.65e+10,
22                        3.72e+05, 3.8333e+04]);
23  esopar["rgas"].SetValue(8.314e-03);
24  esopar["kwfk"].SetValue(1.4e-01);
25  esopar["vk"].SetValue(8.0e-06);
26  esopar["rho_cp"].SetValue(4.106e+03);
27  esopar["vr"].SetValue(2.9e-03);
28  esopar["qknormal"].SetValue(8.0e-06);
29  # create steady state continuation solver
30  sfactory=dmain.GetSolverFactory();
31  conti=sfactory.CreateSolver(diana.CAPE_CONTI, model, "sstate.so");
32  conti.Initialize();
33  conpar=conti.GetParameters();
34  # create reporting interface and specify report variables
35  ri=dmain.CreateReportingInterface("basic");
36  conti.SetReportingInterface(ri);
37  ri.Add(esopar["qknormal"]);
38  ri.Add(eso.GetStateVariables().ItemByName("tk"));
39  ri.Add(conpar["Stability"]);
40  ri.Add(conpar["ConditionCurrent"]);
41  # set continuation parameters
42  conti.AddFreeParameter("qknormal", 0.5e-5, 5e-5);
43  conpar["VerboseLevel"].SetValue(3);
44  conpar["InitialDirection"].SetValue(1.0);
45  conpar["Parameterisation"].SetValue("PseudoArclength");
46  conpar["Predictor"].SetValue("Tangent");
47  conpar["InitialStepSize"].SetValue(0.1);
48  conpar["ConditionCheck"].SetValue(diana.SteadyStateNone);
49  conpar["StabilityCheck"].SetValue(True);
50  conpar["MaxStepSize"].SetValue(50.0);
51  conpar["MaxStepsAmount"].SetValue(2000);
52  # continuation

```

```

53     conti.Continue();
54     # create directory and store continuation results
55     import os
56     if not os.path.isdir('results'): os.mkdir('results');
57     ri.WriteDataMatlab("results/steadystate.m");
58 except diana.ECapeUser, exc:
59     print exc

```

Continuation curve is presented on the figure 2 (boxes are Hopf bifurcations and circles are limit-point bifurcations).

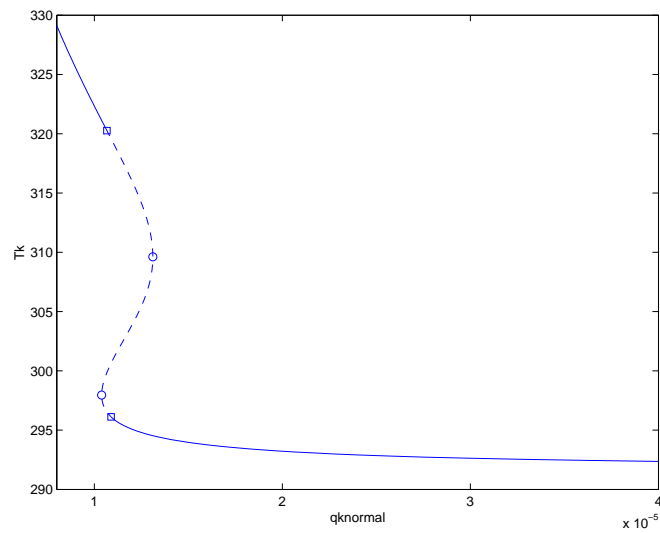


Figure 2: Plot of the continuation $Tk(qknormal)$

References

- [1] K.-P. Zeyer, M. Mangold, T. Obertopp, and E.D. Gilles. The iron(iii)-catalyzed oxidation of ethanol by hydrogen peroxide: a thermokinetic oscillator. *J. Phys. Chem.*, 103A:5515–5522, 1999.