PPODE SUITE
Tutorial

Pascal A. Pieters
p.a.pieters@student.tue.nl

March 31, 2014

This document provides a small tutorial to get started with the PPODE SUITE. For more details on specific options or the internals of the suite, please read the PPODE SUITE Manual provided with the package.

1 Installation

The PPODE SUITE currently only runs on Linux/Unix and requires MATLAB and GCC to be installed.

Unpack the PPODE SUITE package, the created directory will be referred to as ⟨Install Path⟩ from hereon.

```
$ tar -xzf PPODESUITE-0.*.tar.gz
```

Open MATLAB and change directory to the ⟨Install Path⟩.

```
>> cd ⟨Install Path⟩
```

Build the libraries of the PPODE SUITE package.

```
>> PPODE_init
```

This concludes the installation, for next sessions it will suffice to change directory to the ⟨Install Path⟩ and run the addPaths script to add the PPODE SUITE paths to the MATLAB path variable.

```
>> PPODE_addPaths
```

2 Prepartion

In MATLAB, change directory to the path of the MATLAB ODE file that should be evaluated using the PPODE SUITE. Open the file that contains the ODE definitions, i.e. the function file that is usually supplied as first argument to the MATLAB ODE solver ("`ode15s(@⟨odefunc⟩, ...)`", which is defined in the file ⟨odefunc⟩.m). This function definition should have a header of the following form:

```
>> ⟨dx⟩ = func( ⟨t⟩, ⟨x⟩, ⟨par⟩, ⟨neq⟩, ⟨np⟩ )
```

Where ⟨t⟩ is the independent variable, ⟨x⟩ the dependent variable(s) and ⟨par⟩ the parameter values. The last two arguments are optional and represent the number of equations (⟨neq⟩) and number of parameters (⟨np⟩). For further restrictions on the ODE function, see the PPODE SUITE Manual.

If the function fullfills the requirements of the PPODE SUITE, it can be parsed to the programming language of the PPODE SUITE (Fortran 95).

```
>> PPODE_translate('⟨odefunc⟩')
```

This function will generate the file '⟨odefunc⟩.F', which can be build against the solver libraries provided.

```
>> PPODE_build('⟨odefunc⟩.F', '⟨execID⟩')
```

Where ⟨execID⟩ can be any identifier for the executable, e.g. 'odefunc_StiffSolver'. This command does not specify any options, so all default values will be used. Therefore also the default solver is used, which is a stiff solver, for the use of other solvers consult the PPODE SUITE Manual.

3 Execution

The ODEs can now be solved by calling the executable generated by the build command.

```
>> [⟨t⟩, ⟨y⟩] = ⟨execID⟩(⟨neq⟩, ⟨abstol⟩, ...
⟨reltol⟩, ⟨times⟩, ⟨par⟩, ⟨y0⟩)
```

Where ⟨neq⟩ is the number of equations, ⟨abstol⟩ and phreltol are the absolute and relative tolerances respectively, ⟨times⟩ is a vector of time points at which output is desired, ⟨par⟩ is a vector of parameter values and ⟨y0⟩ is a vector of the initial values of the states. The function returns the vector ⟨t⟩ with the time points at which the values of the states are calculated. The matrix ⟨y⟩ contains the values of all states at the time points specified by ⟨t⟩.

Note that the number of equations (⟨neq⟩) has a special role in the PPODE SUITE, since it was developed to work with large systems of ODEs that can have a variable number of equations. Therefore, it is important to always provide a correct number of equations. So if a system has a fixed number of equations, this number still has to be supplied. An easy way to do this is to use the size of the intial values, i.e. provide "`length(⟨y0⟩)`" as first argument.

4 Sparse Jacobian

One main feature of this suite is the ability to solve large systems of sparsely interlinked ODEs. These systems can be solved - within reasonable amount of time - by solvers that make use of a sparse Jacobian matrix implementation. These solvers require the user to provide a number of nonzero elements of the Jacobian matrix. To let PPODE SUITE calculate this number, a Jacobian function has to be generated from the ⟨odefunc⟩. This can be done by providing the 'AnaJac' option to the translate function.

```
>> PPODE_translate('⟨odefunc⟩', 'AnaJac', 1)
```

Subsequently, the ODE system has to be build against the correct solvers. The PPODE SUITE provides two solvers that make use of a sparse Jacobian matrix, namely 'LSODES' and 'MEBDFSO'. To build the system against for example the 'LSODES' solver, the following build command can be used:

```
>> PPODE_build('⟨odefunc⟩.F', '⟨execID⟩', ...
'Solver', 'LSODES')
```

All other steps stay the same with respect to what was described before.

5 Example

An example of this workflow can be found in the "examples/rigid" folder of the package. The function "`compareSolvers`" evaluates a simple stiff problem using various solvers. The ODEs are defined in the file "rigid.m". The "`compareSolvers`" function nicely shows the differences between the use of a MATLAB ODE solver and the PPODE SUITE.

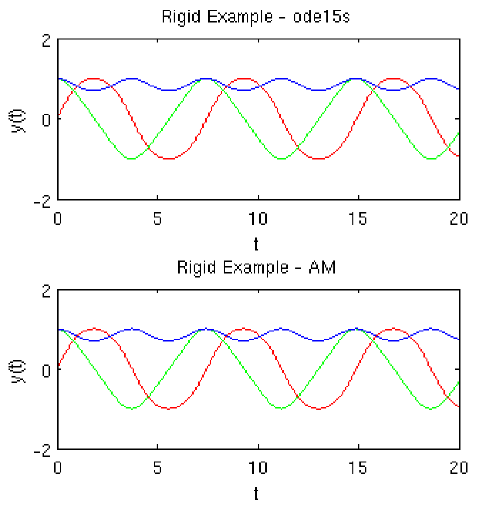
Listing 1: Code Snippet

```
PPODE_translate('rigid');
y0 = [0 1 1];
par = [-0.51];
t = 0:0.01:20;
options = odeset('RelTol', 1e-4, 'AbsTol', 1e-4);

% The ode15s solver
[t1, y1] = ode15s(@(t,y)(rigid(t, y, par)), ...
              t, y0, options);
% The Adams-Moulton solver
PPODE_build('rigid.F', 'rigid_AM', ...
            'Solver','Adams-Moulton');
[t2, y2] = rigid_AM(length(y0), ...
                    options.AbsTol, ...
                    options.RelTol, ...
                    t, par, y0);


figure(1);
subplot(2, 1, 1);
plot(t1, y1(:, 1), 'r-', t1, y1(:, 2), 'g-', ...
      t1, y1(:, 3), 'b-');
xlabel('t');
ylabel('y(t)');
title('Rigid␣Example␣-␣ode15s');
subplot(2, 1, 2);
plot(t2, y2(:, 1), 'r-', t2, y2(:, 2), 'g-', ...
      t2, y2(:, 3), 'b-');
xlabel('t');
ylabel('y(t)');
title('Rigid␣Example␣-␣AM');
```

The output of this code is:


Rigid Example - ode15s
y(t)
t
Rigid Example - AM
y(t)
t