ODDtool Documentation

Leonhard Lücken, Dmitry Puzyrev, Markus Kantner and Serhiy Yanchuk

November 5, 2015

Summary

ODDtool ("ODD" stands for "Ordinary Delay Differential equations") allows the integration of high dimensional delay differential equations involving many different or very large delays. In particular, spatio-temporal phenomena in systems with large delay can take place on a very slow timescale and require in an efficient way to deal with large amounts of data representing the solutions history. Up to date this is not achieved by any other DDE-solver known to the authors. ODDtool has been used for simulations in the papers [1, 2, 3, 4, 5].

The header `ODDtool.h` provides an interface for the integration of delay differential equations with constant point delays. The class which implements the integration is called `ODD_integrator<CKStepper>` and possesses a method `integrate()` which generates solution data and writes it to output-files. The template parameter `CKStepper` denotes a class which implements a CashKarp adaptive-stepwidth method for the actual computation of integration steps. Other integration methods can be added by implementing different stepper classes (which should be derived from the abstract class `ODD_Stepper`). This template design is inspired from the `Odeint` class given in [6].

These instructions presuppose that you are working on a unix system with gcc4.9 or higher and the `boost` libraries installed[1].

Usage

Providing the right hand side

To use the integrator, you include the header `ODDtool.h` and provide the right hand side of the equation

$$\dot{x}(t) = f(p; t, x(t), x(t-\tau_1), ..., x(t-\tau_{n_\tau})) \in \mathbb{R}^N, \tag{1}$$

you wish to integrate. Here, the vector $p \in \mathbb{R}^{n_p}$ contains the system's parameters, $t \in \mathbb{R}$ is a scalar value of the time, and $x(t-\tau_j) \in \mathbb{R}^N$ is the systems state at time $t-\tau_j$. The function f must be implemented as a `C++` function with the following signature:

```
void f(double, const vector<double>&, ODD_delayed_values&, const vector<double>&, vector<double>&)
```

Here, the type `ODD_delayed_values` is a data structure which contains the needed values $x_i(t-\tau_j)$. From an object `xd` of this type the value $x_i(t-\tau_j)$ is retrieved by the expression `xd[j][i]` (note that i and j are reversed in some sense). A call to `f(t, x, xd, p, result)` writes the value of f(`p;t,x,xd[0],...,xd[`n_τ`]`) to the vector `result`.

For instance, if you wish to integrate two delay-coupled Mackey-Glass systems

$$\dot{x}_1(t) = \frac{\sigma x_2(t-\tau)}{1+x_2(t-\tau)^n} - \gamma x_1(t), \tag{2}$$

$$\dot{x}_2(t) = \frac{\sigma x_1(t-\tau)}{1+x_1(t-\tau)^n} - \gamma x_2(t), \tag{3}$$

you could implement the right hand side function as:

```
   void f(double t,...,vector<double>& result) {
      result[0] = p[0] * xd[0][1] / (1 + pow(xd[0][1], p[1])) - p[2] * x[0]; // (2)
      result[1] = p[0] * xd[0][0] / (1 + pow(xd[0][0], p[1])) - p[2] * x[1]; // (3)
   }
```

[1]check the boost linkage in the `Makefile` if compilation does not work

where the parameters are σ =`p[0]`, n =`p[1]` and γ =`p[2]`, and the delayed terms are $x_1(t-\tau)$ =`xd[0][0]` and $x_2(t-\tau)$ =`xd[0][1]`.

Providing parameters and delays

The parameters $p = (p_1, ..., p_{n_p})$ and the delays $\tau_1, ...\tau_{n_\tau}$ in (1) are specified in the parameter file `ODD_parameters.txt`. In this file, the user can as well control various parameters to the integration procedure such as starting time and ending time, desired tolerance, minimal and maximal stepwidth, etc. – see the file `ODD_default_parameters.txt` for further documentation. If a parameter is not provided by the user in `ODD_parameters.txt`, a default value specified in `ODD_default_parameters.txt` is used. The later file should not be modified by the user.

To make the above example (2) and (3) work, assuming $\sigma = 2.2$, $n = 10$, $\gamma = 1$, $\tau = 20$, and that you wish to integrate for $t \in [0, 100]$, you should assure that the parameter file contains the following lines:

```
N = 2
name = TwoMackeyGlass
p0 = 2.2
p1 = 10
p2 = 1
tau0 = 20
t_start = 0
t_end = 100
```

Here, the specification of the system's name is optional. However, it will influence the naming of the output files. The default value is `name = noname`.

Providing an initial history

The initial data for (1) can be provided in two ways. Either you define a function `hist(double t, vector<double>& result)` within the program similarly as you do for the right hand side function `f(...)`. For each t it should assign the value of the initial function to the vector `result`.

A second way to provide the history function is to store it in a table containing timepoints, the corresponding states of the system and the corresponding derivates of the function. (The derivatives are required until a higher order Newton interpolation is implemented.) If you provide only one time point, the programm treats the corresponding statepoint as a constant initial function (derivatives are not required in this case). The table of initial data can be provided in two ways: in seperate files or in a single file. This is done via the parameter file `ODD_parameters.txt` by adding either a line

```
history_file = name_of_some_file.txt
```

or adding the lines

```
history_t_file = name_of_some_file.txt
history_x_file = name_of_some_other_file.txt
history_f_file = name_of_some_another_file.txt
```

In both cases each row corresponds to one time point, different components are seperated by spaces. If all data is contained in a single file, one row should look like this:

$$\underbrace{992.88}_{\mathtt{t}}\ \underbrace{5.10\ 8.74}_{\mathtt{x}}\ \underbrace{71.69\ -21.91}_{\mathtt{f}}$$

Building a program using `ODDtool`

If you wish to integrate the example above, you have to prepare the parameter file `ODD_parameters.txt`, and a file `main.cpp`, which includes `ODDtool.h` and contains (or includes) the definition of your right hand side function and the method `main()`, which constructs an instance of `ODD_integrator<CKStepper>` and calls `integrate()`. Further, you may want to include an explicit definition of a history function. Otherwise, you must provide files containing information about the history. The `C++` file might look like this:

```
// file main.cpp
#include "ODDtool.h"
using namespace std;

// user-supplied history function
void myHistory(double t, vector<double>& result) {
     result[0] = abs(cos(t)) + 1;
     result[1] = 1 + sin(t);
  }

  // Two coupled Mackey-Glass systems
  void MG2Rhs(double t, const vector<double>& x, ODD_delayed_values& xd, const vector<double>& p,
vector<double>& result) {
     result[0] = p[0] * xd[0][1] / (1 + pow(xd[0][1], p[1])) - p[2] * x[0];
     result[1] = p[0] * xd[0][0] / (1 + pow(xd[0][0], p[1])) - p[2] * x[1];
  }

  int main() {
     // create an integrator
     ODD_integrator<CKStepper> integrator(MG2Rhs, myHistory);
     // start integration over time interval specified in parameter file (ODD_parameters.txt)
     integrator.integrate();
  }
```

If you don't want to specify the history explicitely but prefer to provide it in a file, you can simply drop the corresponding argument in the constructor for the integrator and type instead:

```
ODD_integrator<CKStepper> integrator(MG2Rhs);
```

Finally, you compile an executable. To this end you copy your files to the base directory of a copy of oddtool and you type `make` to the command line. This should compile the executable `oddtool` in this directory. When you run the executable by typing `./oddtool`, files are created, which contain the calculated solution.

Output

The output-files generated by oddtool are named `t.txt`, `x.txt` and, optionally, `f.txt`. They are contained in the local directory `system_name_data/`, if the system's name is specified as `name = system_name` in the file `ODD_parameters.txt`. Moreover, `integrate()` generates the files `system_name_temp/continuation_*.txt` containing information necessary to resume the integration from the last calculated point.

References

[1] D. Puzyrev, S. Yanchuk, A. Vladimirov, and S. Gurevich. Stability of plane wave solutions in complex Ginzburg-Landau equation with delayed feedback. *SIAM J. Appl. Dyn. Syst.*, 13:986–1009, 2014.

[2] Serhiy Yanchuk and Giovanni Giacomelli. Pattern formation in systems with multiple delayed feedbacks. *Phys. Rev. Lett.*, 112:174103, 2014.

[3] S. Yanchuk and G. Giacomelli. Dynamical systems with multiple long-delayed feedbacks: Multiscale analysis and spatiotemporal equivalence. *Phys. Rev. E.*, (accepted), 2015.

[4] Vladimir Klinshov, Leonhard Lücken, Dmitry Shchapin, Vladimir Nekorkin, and Serhiy Yanchuk. Multistable jittering in oscillators with pulsatile delayed feedback. *Phys. Rev. Lett.*, 114:178103, 2015.

[5] Markus Kantner, Eckehard Schöll, and Serhiy Yanchuk. Delay-induced patterns in a two-dimensional lattice of coupled oscillators. *Sci. Rep.*, 5(8522), 2015.

[6] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.