Open Solving Library for ODEs (OSLO) 1.0
User Guide

Introduction

OSLO is a .NET and Silverlight class library for the numerical solution of ordinary differential equations (ODEs). We wrote this library to provide open source access to established equation solving libraries in the .NET environment. This enables numerical integration to be performed in C#, F# and Silverlight applications. OSLO implements Runge-Kutta and back differentiation formulae (BDF) for non-stiff and stiff initial value problems for ordinary differential equations.

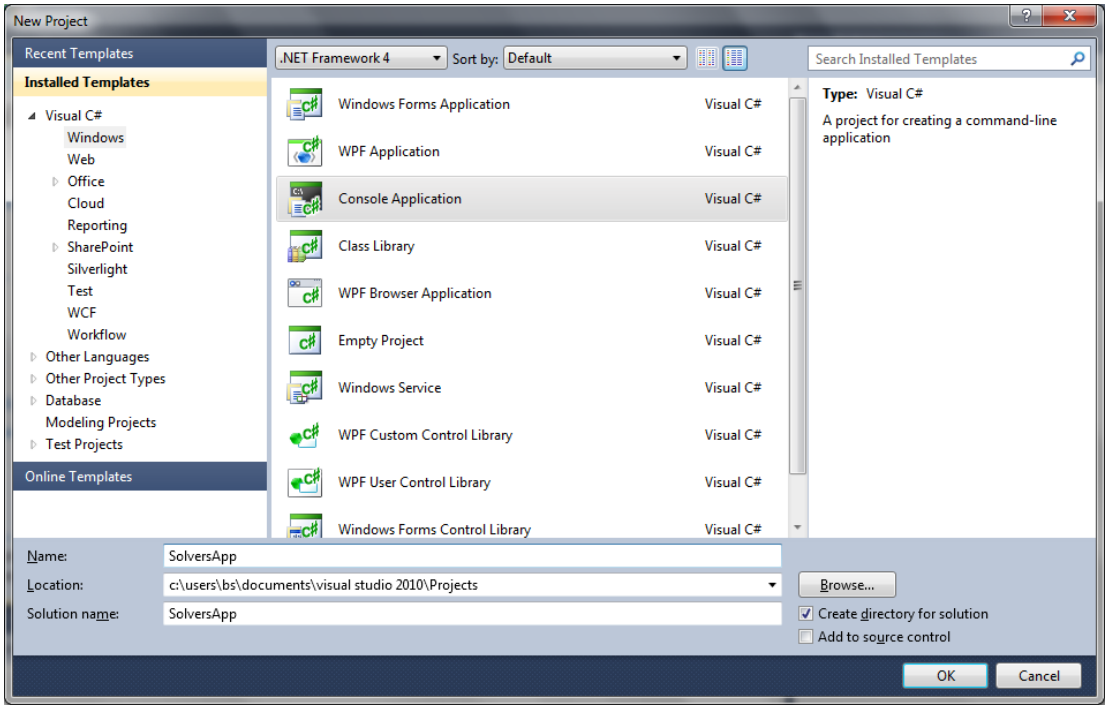
This User Guide provides instructions for installation and library usage in addition to some worked examples. The Getting Started section includes a simple example that shows the basic steps to solve ordinary differential equations using OSLO.

The OSLO library is distributed as a Visual Studio solution sample source code in C# and F#. Samples visualize results using DynamicDataDisplay library.

Getting Started

In this section, we demonstrate how to start working with the OSLO library

1. Download zip file with OSLO library and unpack it to the folder of your choice.
2. Start Visual Studio 2010 and create new project using File>New>Project command. Select Visual C# console application for Windows as shown on the picture below.



Step 2. Create new project dialog window

3. Open Solution Explorer window (you can press Ctrl+W,S). Right click on the References and choose 'Add Reference…' in context menu. Go to Browse tab and add Microsoft.Research.Oslo.dll from the folder you've selected at step 1.
4. Switch to the C# source code file. Go to the Solution Explorer window and double click Program.cs item.
5. Add using statements at the beginning of the file.

```
using Microsoft.Research.Oslo;
```

6. As an example, let's numerically solve the Lotka-Volterra (Predator-Prey) model as an initial value problem. The predator-prey interactions lead to a time evolution of these populations according to

$$\begin{cases} \frac{dx}{dt} = x - xy, \\ \frac{dy}{dt} = -y + xy. \end{cases}$$

where x(t) is the prey population and y(t) is the predator population. We consider the initial conditions x(0) = 5 and y(0) = 1.

Add the following code to the Main method to define the system we are going to solve using an explicit Runge-Kutta method. The first parameter of RK547M is the initial time, the second parameter is a 2D vector with the initial system state and the third parameter is a lambda expression that defines the right hand side of the equations. Note that we use the Vector constructor with two parameters to construct 2D vectors. More parameters will result in vectors of higher dimension.

```
var sol = Ode.RK547M(
                0,
                new Vector(5.0, 1.0),
                (t, x) => new Vector(
                     x[0] - x[0] * x[1],
                        -x[1] + x[0] * x[1]));
```

7. The previous line doesn't actually solve the ODEs. Instead it defines an enumerable sequence of solution points. The actual integration occurs when the variable 'sol' is being accessed. The next line will request solution points until time moment 20 and store them with step 1 in an array.

```
var points = sol.SolveFromToStep(0, 20, 1).ToArray();
```

8. Now we'll print solution points to the screen.

```
foreach (var sp in points)
    Console.WriteLine("{0}\t{1}", sp.T, sp.X);
```

9. After completing steps 6-8 the source code of Program.cs should look like this:

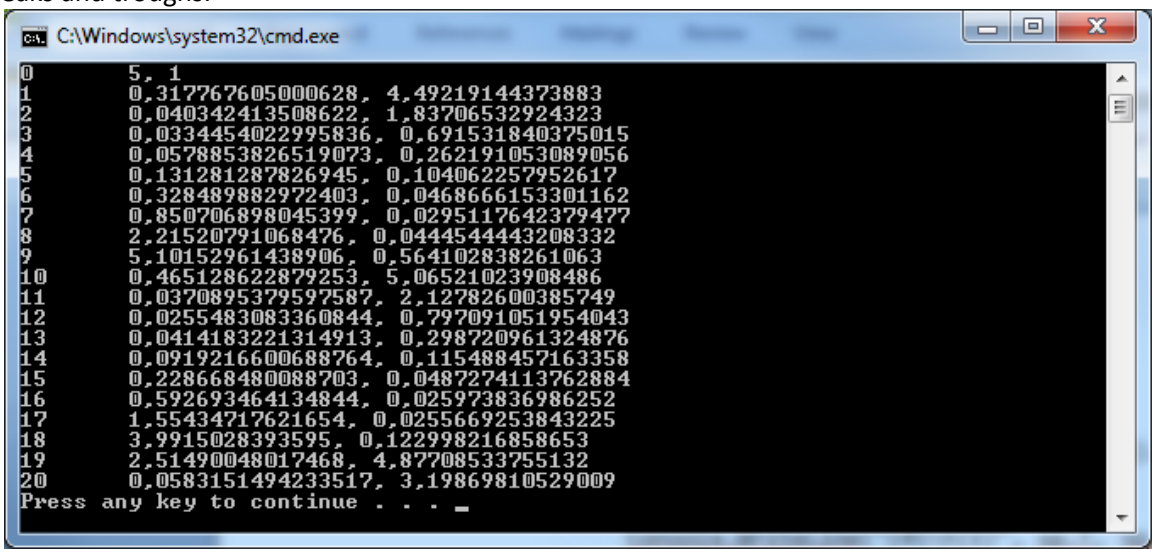
```
     static void Main(string[] args)
     {
         var sol = Ode.RK547M(
```

```
            0,
            new Vector(5.0, 1.0),
            (t, x) => new Vector(
                x[0] - x[0] * x[1],
                -x[1] + x[0] * x[1]));

        var points = sol.SolveFromToStep(0, 20, 1).ToArray();

        foreach (var sp in points)
            Console.WriteLine("{0}\t{1}", sp.T, sp.X);
    }
```

Compile and run the application. A console window appears with three columns of numbers. The first column is time ('T' property of solution point), the second and third columns are the solution vectors ('X' property of solution point). You may see how the population of predator and prey go through cycles of peaks and troughs.



Step 9. Computation results.

Ordinary differential equations (ODE)

Overview

The OSLO library provides subroutines to integrate initial value problems from time t_0 with initial conditions given by the vector $\mathbf{x}_0$.

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}), t \geq 0, \\ \mathbf{x}(t_0) = \mathbf{x}_0, \end{cases}$$

where $\mathbf{f}(t, \mathbf{x}) = \big(f_1(t, \mathbf{x}), \dots, f_s(t, \mathbf{x})\big)$.

Runge-Kutta and Gear backward differentiation formulae are supported. Both methods use automatic step size calculation procedures to satisfy accuracy conditions. The right-hand sides are specified as user defined methods or lambda expressions with two parameters: time and system state.

Programming model

Numerical integration is initiated by invoking a single method. It is Microsoft.Research.Oslo.Ode.RK547 for explicit Runge-Kutta method and Microsoft.Research.Oslo.Ode.GearBDF for implicit Gear BDF method suitable for stiff problems.

Both methods have the same set of parameters that include:

- Time moment to solve from
- Initial values vector
- Right part, specified either as method name or as lambda expression
- Additional options including initial suggested time step, desired accuracy, output time step and Jacobian matrix.

ODE solution methods return sequence of solution points represented by instances of SolPoint structures containing time moment T and system state X at this moment.

```
var sol = Microsoft.Research.Oslo.Ode.RK547M(
    0,
    new Vector(0.5,4.0),
    (t,x) => new Vector(
        x[1] – x[1]*x[0],
        -x[0] + x[1]*x[0]),
    new Options {
        AbsoluteTolerance = 1e-6,
        RelativeTolerance = 1e-6
    });
```

It is important to note that no integration in performed at the moment of RK547M invocation. Instead the method returns instance of IEnumerable<SolPoint> that performs actual computation when next point is requested. This allows to use full potential of LINQ subroutines when working with ODE solution.

For example, following code prints every point produces by numeric integration between time moments 1.0 and 2.5:

```
foreach(var p in sol.SkipWhile(sp => sp.T < 1.0).TakeWhile(sp => sp.T <= 2.5))
    Console.WriteLine(“{0}, {1}”, sp.T, sp.X);
```

In additional to standard LINQ methods, several methods are provided to manipulate the solution sequences produced by OSLO. The extension method WithStep(double dt) returns interpolated solution points at time moments n * dt , and AddTimeStep() appends a time step between the previous and current solution points as extra components of the system state.

Because the system is being integrated and every time solution sequence is enumerated, it is advisable to use ToArray() when the solution is going to be accessed multiple times. In following example numerical integration will take place twice, inside both 'foreach' loops.

```
var sol = Ode.RK547(0, new Vector(1), (t,x) => -x[0]).TakeWhile(p => p.T <=
5.0);
foreach(var p in sol) // Integrate ODE
    Console.WriteLine(p.X);
foreach(var p in sol) // Integrate ODE again!
    Console.WriteLine(p.X);
```

One can use the ToArray method to explicitly integrate the system when needed:

```
var sol = Ode.RK547(0, new Vector(1), (t,x) => -x[0]).TakeWhile(p => p.T <=
5.0).ToArray(); // Computations are performed here!
foreach(var p in sol) // Just enumeration of the array
    Console.WriteLine(p.X);
foreach(var p in sol) // Just enumeration of the same array
    Console.WriteLine(p.X);
```

Numerical methods

Runge-Kutta (Microsoft.Research.Oslo.Ode.RK547M method)

This method is most appropriate for solving non-stiff ODE systems. It is based on classical Runge-Kutta formulae with modifications for automatic error and step size control. Following Dormand and Prince [1], define method coefficients $c_i, a_{ij}, \widehat{b_i}, b_i$, where

c_i	a_{ij}						$\widehat{b_i}$	b_i
0							$\frac{35}{384}$	$\frac{5179}{57600}$
$\frac{1}{5}$	$\frac{1}{5}$						0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
1	$-\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$

On the 1st integration step, $\mathbf{x}_1 = \hat{\mathbf{x}}_1 = \mathbf{x}_0$

Then, if $\mathbf{x}_\mathrm{n}$ is solution vector on n'st step, h_n is integration step size, we may find $\overline{\mathbf{x}}_{\mathrm{n}+1}$ in the following way:

1) Find $\mathbf{k}_1 = h_n\mathbf{f}(t_n, \hat{\mathbf{x}}_\mathrm{n}),\ \mathbf{k}_\mathrm{i} = h_n\mathbf{f}\left(\hat{\mathbf{x}}_\mathrm{n} + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_\mathrm{j}\right),\ i = 2,3,\dots,s$
2) Find $\hat{\mathbf{x}}_{\mathrm{n}+1} = \hat{\mathbf{x}}_\mathrm{n} + \sum_{i=1}^{s} \widehat{b}_i\mathbf{k}_\mathrm{i}$ and $\mathbf{x_{n+1}} = \hat{\mathbf{x}}_{\boldsymbol{n}} + \sum_{i=1}^{s} b_i\mathbf{k}_\mathrm{i}$
3) Compute error estimation $\varepsilon_{n+1} = \max\limits_{i=1,..,s} \frac{|\mathrm{x}_{\mathrm{n}+1,\mathrm{i}} - \hat{\mathrm{x}}_{\mathrm{n}+1,\mathrm{i}}|}{\max\left(AbsTol, RelTol\cdot\max\left(|\mathrm{x}_{\mathrm{n}+1,\mathrm{i}}|, |\hat{\mathrm{x}}_{\mathrm{n}+1,\mathrm{i}}|\right)\right)}$

   where *AbsTol* and *RelTol* are absolute and relative tolerance taken from solver options, *s* is a dimension of the system and *i* subscript stands for *i-th* component of vector.
4) Step size is decreased and n'th step is performed again if $\varepsilon_{n+1} \geq 1$. Otherwise $\overline{\mathbf{x}}_{\mathbf{n}+\mathbf{1}}$ is accepted as next solution point.
5) Step size is adjusted to keep ε_{n+1} below 1.0. Special procedure is used to avoid step size oscillations (see PI-filter definition in [2] for details).

Gear's backward differentiation formulae (Microsoft.Research.Oslo.Ode.GearBDF method)

It is implementation of Gear back differentiation method [4], a multistep implicit method for stiff ODE systems solving. General back differentiation formula of order *q* can be written as

$$\mathbf{x}_\mathrm{n} = \sum_{j=1}^{q} \alpha_j\mathbf{x}_{\mathrm{n}-\mathrm{j}} + h_n\beta_0\mathbf{f}(\mathbf{x}_\mathrm{n})$$

Coefficients α_j and β_0 are chosen to ensure that the above formula gives exact solutions for polynomials of order *q*.

Nordsieck representation [3,5] of Gear back differentiation formulae is used. On the nth integration step, we use Nordsieck's history matrix

$$\mathbf{Z}_\mathrm{n} = \begin{pmatrix} x_{1,n} & h_n\dot{x}_{1,n} & h_n^2\frac{\ddot{x}_{1,n}}{2!} & \cdots & h_n^q\frac{x_{1,n}^{(q)}}{q!} \\ x_{2,n} & h_n\dot{x}_{2,n} & h_n^2\frac{\ddot{x}_{2,n}}{2!} & \cdots & h_n^q\frac{x_{2,n}^{(q)}}{q!} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{s,n} & h_n\dot{x}_{s,n} & h_n^2\frac{\ddot{x}_{s,n}}{2!} & \cdots & h_n^q\frac{x_{s,n}^{(q)}}{q!} \end{pmatrix} \in R_{s\times q+1},$$

where *s* is dimension of the system, *q* is method maximal available order. It is easy to see that the first column of the Nordsieck matrix is the phase vector, and the second column is the first derivative of the phase vector multiplied by the time step.

We also use the following matrix of coefficients [5] where each row corresponds to specified method order q:

$$\mathbf{L} = \begin{pmatrix} 1 & & & & & & \\ \frac{2}{3} & 1 & \frac{1}{3} & & & & \\ \frac{6}{11} & 1 & \frac{6}{11} & \frac{1}{11} & & & \\ \frac{24}{50} & 1 & \frac{35}{50} & \frac{10}{50} & \frac{1}{50} & & \\ \frac{120}{274} & 1 & \frac{225}{274} & \frac{85}{274} & \frac{15}{274} & \frac{1}{274} & \\ \frac{720}{1764} & 1 & \frac{1624}{1764} & \frac{735}{1764} & \frac{175}{1764} & \frac{21}{1764} & \frac{1}{1764} \end{pmatrix}$$

At initial time moment $t = t_0$, we take the Nordsieck matrix as

$$\mathbf{Z}_0 = \begin{pmatrix} x_1(t_0) & h_0\dot{x}_1(t_0) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_s(t_0) & h_0\dot{x}_s(t_0) & 0 & \cdots & 0 \end{pmatrix}, \text{ where } \dot{\mathbf{x}}(t_0) = \mathbf{f}(\mathbf{x}(t_0), t_0).$$

At each step of integration, we implement the predictor-corrector scheme:

Predictor:

$$\begin{cases} \mathbf{Z}_n^{[0]} = \mathbf{A}\mathbf{Z}_{n-1}, A_{i,j} = \binom{i}{j}, \\ \mathbf{e}_n^{[0]} = 0. \end{cases}$$

Corrector (multiple iterations):

$$\begin{cases} \mathbf{g}_n^{[m]} = h_n \mathbf{f}\left(\mathbf{x}_n^{[m]}\right) - h_n\dot{\mathbf{x}}_n^{[0]} - \mathbf{e}_n^{[m]}, \\ \mathbf{e}_n^{[m+1]} = \mathbf{e}_n^{[m]} + (\mathbf{I} - h_n\beta\mathbf{J})^{-1}\mathbf{g}_n^{[m]}, \\ \mathbf{x}_n^{[m+1]} = \mathbf{x}_n^{[0]} + L_{0,q}\mathbf{e}_n^{[m+1]}. \end{cases}$$

After M iterations of the corrector step we compute the Nordsieck matrix for the next time instance as

$$\mathbf{Z}_n = \mathbf{Z}_n^{[0]} + \mathbf{C}, C_{i,j} = e_{n,i}^{[M]} L_{j,q}.$$

Iterational corrector algorithm uses inverted Jacobian **J** of the system right-hand side. If a Jacobian isn't supplied in the Options structure, the following numerical form of the Jacobian is used [5]:

Let $\boldsymbol{\delta x} = \left(\sqrt{10^{-6}\, max(10^{-5}, |x_1|)}, \ldots, \sqrt{10^{-6}\, max(10^{-5}, |x_s|)}\right)$ is numerical analogue of x variation.

Then $\boldsymbol{J} = \begin{pmatrix} \frac{f_1(t,\boldsymbol{\delta x}^1) - f_1(t,\mathbf{x})}{\delta x_1} & \cdots & \frac{f_1(t,\boldsymbol{\delta x}^S) - f_1(t,\mathbf{x})}{\delta x_s} \\ \vdots & \ddots & \vdots \\ \frac{f_s(t,\boldsymbol{\delta x}^1) - f_s(t,\mathbf{x})}{\delta x_1} & \cdots & \frac{f_s(t,\boldsymbol{\delta x}^S) - f_s(t,\mathbf{x})}{\delta x_s} \end{pmatrix}$,

where $\boldsymbol{\delta}\mathbf{x}^i = (x_1, \dots, x_i + \delta x_i \dots, x_s), i = 1, \dots s.$

On every step of integration, the solution accuracy is controlled and step size is changed according to convergence of corrector iterations. Note that method order can also decrease and increase in the range 1 to 3. For details about Nordsieck matrix transformations when changing order and/or step size see [5].

References

[1] Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics 6 (1): 19–26

[2] Soderlind, G.. Digital Filters in Adaptive Time Stepping. ACM Transactions on Mathematical Software.

[3] E.Hairer, S.P. Nørsett, G.Wanner, Solving ordinary differential equations, Springer, 1993.

[4] C.W. Gear. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, Englewood Cliffs 1971.

[5] K. Radhakrishnan and A. C. Hindmarsh, "Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations," LLNL report UCRL-ID-113855, December 1993