NTNU

TMA4215 Numerical Mathematics — Fall 2016

Student Project

Part 1: Implementation of a stiff ODE solver

1.1 Background

In order to solve a possibly stiff initial value problem

$$y' = f(t,y) \qquad y(t_0) = y_0, \qquad t_0 \le t \le t_{\text{end}}, \tag{1}$$

the following embedded Runge–Kutta scheme is suggested:

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
2\gamma & \gamma & \gamma & 0 & 0 \\
1 & \frac{-4\gamma^2+6\gamma-1}{4\gamma} & \frac{-2\gamma+1}{4\gamma} & \gamma & 0 \\
1 & \frac{6\gamma-1}{12\gamma} & \frac{-1}{12\gamma(2\gamma-1)} & \frac{-6\gamma^2+6\gamma-1}{3(2\gamma-1)} & \gamma \\
\hline
\widehat{\boldsymbol{b}}^\top & \frac{-4\gamma^2+6\gamma-1}{4\gamma} & \frac{-2\gamma+1}{4\gamma} & \gamma & 0 \\
\boldsymbol{b}^\top & \frac{6\gamma-1}{12\gamma} & \frac{-1}{12\gamma(2\gamma-1)} & \frac{-6\gamma^2+6\gamma-1}{3(2\gamma-1)} & \gamma
\end{array} \tag{2}$$

Here $\widehat{\boldsymbol{b}}$ refers to the error estimating method, $\boldsymbol{b}$ represents the advancing method, and γ is some real parameter. Using the fact that the first stage is explicit, and both the advancing and the error estimating methods are stiffly accurate, the RK-scheme can be formulated as follows (convince yourself that this is correct):

$$Y_1 = y_n, \tag{3a}$$

$$Y_i = y_n + h\sum_{j=1}^{i-1} a_{ij} f\left(t_n + c_j h, Y_j\right) + h\gamma f\left(t_n + c_i h, Y_i\right) \quad \text{for} \quad i = 2,3,4, \tag{3b}$$

$$y_{n+1} = Y_4, \tag{3c}$$

$$le_{n+1} = Y_4 - Y_3. \qquad \text{(Local error estimate)} \tag{3d}$$

Coefficients a_{ij} and c_j come from matrix $A = (a_{ij})$ and $\boldsymbol{c} = (c_1, \dots, c_4)$ in (2), respectively, assuming standard Butcher tableau format.

In this project, you will first investigate relevant properties of the RK-method, justify the theoretical results by numerical experiments, and finally use it to solve some given problems.

1.1.1 Solution of the nonlinear systems of equations

The only part of the implementation of an adaptive diagonally implicit RK-method which is not covered by the lecture note, is how the underlying nonlinear equations should be solved. Hence, this is explained here.

The stage values Y_i for $i = 2, 3$ and 4 are sequentially computed—first Y_2, then Y_3, and finally Y_4—by a *modified* Newton method of the form

$$\begin{aligned}(I - h\gamma J)\Delta Y &= h\gamma f\left(t_n + c_i h, Y_i^{(k)}\right) - Y_i^{(k)} + K_i,\\ Y_i^{(k+1)} &= Y_i^{(k)} + \Delta Y,\end{aligned} \tag{4}$$

where I is the identity matrix, $J = J(t_n, y_n)$ is the Jacobian, and $K_i = y_n + h\sum_{j=1}^{i-1} a_{ij} f\left(t_n + c_j h, Y_j\right)$. Noticeably, K_i has to be updated at each stage i, but remains constant through the iterations on k. Use $Y_i^{(0)} = Y_{i-1}$ as a starting value for the k-iteration, and stop the iteration when $\|\Delta Y\| \le \mathtt{TOL}_{\mathrm{it}}$ (success) or when $k = \mathtt{MAX}_{\mathrm{it}}$ (failure). In the latter case, reduce the stepsize h and try again. Observe that the Newton matrix $I - h\gamma J$ is only formed at the beginning of each step; it should neither be updated between stages or within stages (k-iterations).

If time permits, you may improve the code further by keeping the Newton matrix over several steps.

1.1.2 Error control and stepsize selection

Use error per step (EPS) and local extrapolation. In addition, choose some maximum stepsize $h_{\max}$, and stop the integration with some error message if the stepsizes become unreasonable small.

1.2 Theoretical investigations

Before applying the method, you should confirm that it has the properties we want for a stiff ODE-solver. You should also choose the free parameter γ. More specifically:

$\boxed{1}$ Find the order of both the error estimating and advancing methods.

$\boxed{2}$ Find the stability functions $R(z)$ and $\widehat{R}(z)$ for both methods, where the hat refers to the error estimating method.

$\boxed{3}$ Find the stability constants $R(\infty)$ and $\widehat{R}(\infty)$ of both methods. (NB! *Matrix A is singular*).

$\boxed{4}$ For the implementation, let γ be the root of $R(\infty) = 0$ near 0.4. Find this root (numerically), and the corresponding $\widehat{R}(\infty)$.

$\boxed{5}$ Discuss the linear stability properties of both the advancing and the error estimating methods. Plot the stability regions.

Include the results and how you found them in the report, but do not include the intermediate calculus. You are free to use tools like MAPLE or WOLFRAM ALPHA, but simplify the expressions you get as much as possible.

1.3 Implementation

In this part, you should implement the method (2) with γ found in $\boxed{4}$, including error estimation—based on local extrapolation—and variable stepsizes. During the implementation process, we suggest that you use the following test problems:

- **Linear test problem:**

$$y' = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} y + \begin{bmatrix} t \\ t+3 \end{bmatrix}, \qquad y(0) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad t_{\text{end}} = 1. \tag{5}$$

  This is a problem for which the exact solution is easy to find. Moreover, it is very convenient for checking the nonlinear solver, since the Newton iteration will converge in one iteration (why?).

- **Van der Pol equation:**

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2, \\ y_2' &= \mu(1-y_1^2)y_2 - y_1, & y_2(0) &= 0. \end{aligned} \tag{6}$$

  In this case you may vary the stiffness of the problem. Try e.g. $\mu = 5$, $\mu = 50$ and $\mu = 500$. Typical integration intervals will be $t_{\text{end}} \approx \mu$.

- **The Robertson reaction:**

$$\begin{aligned} y_1' &= -0.04y_1 + 10^4 y_2 y_3, & y_1(0) &= 1, \\ y_2' &= 0.04y_1 - 10^4 y_2 y_3 - 3\times 10^7 y_2^2, & y_2(0) &= 0, \\ y_3' &= 3\times 10^7 y_2^2, & y_3(0) &= 0 \end{aligned} \tag{7}$$

  on the interval $0 \le t \le 40$.

Do the implementation of the ODE solver in the following steps:

$\boxed{6}$ Implement a function `function [A,c,g,s] = method` that returns the coefficient matrix A, the vector $\mathbf{c}$, the diagonal element γ and the number of stages s (which is 4 in this case).

Check that the function is correctly written by verifying the order conditions.

$\boxed{7}$ Implement a function `onestep`, doing one step of the RK-method using stepsize h. The function call should typically be:

```
function [tnext, ynext, le, iflag] = onestep(f, jac, tn, yn, h, Tolit)
% [tnext, ynext, le, iflag] = onestep(f, jac, tn, yn, h, Tolit)
% Do one step with an implicit RK—method method.
% Input arguments:
%    f, jac: the functions f(t, y) and Jac(t, y);
%    tn, yn: time and state variables
%    h: step size
%    Tolit: tolerance for the Newton iterations.
% Output arguments:
```

```
%    tnext, ynext: time and state variables after one step
%    le: Local error estimator.
%    iflag =  1: Iterations are successful
%          = −1: Iterations fail; t and y are not updated
```

The most tricky part here is the implementation of the Newton iterations (4) from Section 1.3. First verify it on the linear test problem. The Newton iteration should solve the linear problem in one iteration.

$\boxed{8}$ Implement a constant-stepsize solver, using `onestep` from $\boxed{7}$. Make convergence plots, that is, loglog plots of error vs. stepsize—similar to what you did during the exercise sessions in the class—for both the advancing and the error estimating methods applied to all three test problems. Thus in total there should be six graphs, but save space by plotting more than one graph in a figure. Use $t_{\text{end}} = 1$ in all cases, and set $\mu = 50$ for the Van der Pol equation (6).

Remember that if the error satisfies $\|e_N\| \approx Ch^p$, then

$$\log \|e_N\| \approx \log C + p \log h.$$

As such, given two vectors `H` and `Err` containing the stepsizes and the corresponding global errors, respectively, an numerical approximation to the order p can be found by

```
    ord = polyfit(log(H),log(Err), 1);
    pnum = ord(1)
```

Explain why this works.

NB! For computation of the numerical order, remove points where the error clearly is dominated by rounding or iteration errors.

$\boxed{9}$ Write a complete code for solving ODEs by the RK-method in (2), including error estimation and variable stepsizes. The algorithms are similar to how this is done for explicit RK-methods. Use the function `onestep` which you have already implemented. The function call should be *exactly* like this:

```
function [t, y, iflag, nfun, njac] = RKs(f, jac, t0, tend, y0, Tol, h0)
```

where `iflag` indicates whether the solver was successful (`iflag` ≥ 1) or not, and `nfun` and `njac` are the numbers of function and Jacobian evaluations, respectively.

For the Newton iterations (4), let $\texttt{TOL}_{\text{it}} = 0.1 \cdot \texttt{Tol}$, where `Tol` is the user-defined tolerance. If the iterations fail to converge, reduce the stepsize by a factor of 0.5.

Test the code on all three test problems. Present plots of the solution together with the stepsizes. For the Van der Pol case (6), you will probably prefer to plot y_1 only, and not y_2—at least for large values of μ (try it).

$\boxed{10}$ Include counters `nfun` and `njac` that count all function evaluations and all Jacobian evaluations done during the integration of one problem.

Make work-precision diagrams. That means: solve one of the problems for different tolerances, e.g. $\texttt{Tol} = 10^{-2}, 10^{-3}, \ldots, 10^{-8}$, and in each case, find the error at t_{end}. For each value of `Tol`, let the corresponding work be computed as

$$\mathtt{W} = \mathtt{nfun} + \mathtt{m} * \mathtt{njac}.$$

Plot work vs. error. Compare with the results from some of MATLAB's solvers for stiff ODEs, and comment on the results. To set the tolerance and to get the statistics for the MATLAB solvers, you can write

```
sol = ode23s(f, tint, y0, options)
sol.stats
```

See the documentation for more information.

PART II: *Choose one, and only one, of the following problems*

2.1 Event detection and the woodpecker toy problem

Let $y(t)$ be a solution of the ODE $y' = f(t,y)$. Then find t_{ev} such that

$$g(t_{\text{ev}}, y(t_{\text{ev}})) = 0$$

Such functionality has quite a lot of useful applications: You might want to find e.g. maximum or minimum of your function, you want to know when a solution reaches a particular value. In this assignment, the event handler will be used to detect points of discontinuities.

Example: Let an elastic ball fall from height y_0. When the ball hits the floor, it bounces up again with a slightly smaller speed than before. The model becomes

$$y''(t) = -9.81, \qquad y(0) = y_0 > 0, \qquad y'(0) = 0$$

from 0 to when the ball hits the floor at t_{ev}, that is when

$$y(t_{\text{ev}}) = 0, \qquad y'(t_{\text{ev}}) < 0.$$

Continue the integration with new initial values (the ball bounces up again)

$$y'_+ = -0.9 y'_-,$$

where the y'_- is the velocity just before the event, and y'_+ the velocity afterwards. And continue the integration till the next time the ball hit the floor, etc.

Assignment: Construct and implement an event locator. Include it in your code in such a way that the integration stops at that point, and the code returns to the main program. Test it on the bouncing ball problem (some maximum stepsize will be helpful in this case) to make sure it works, and then apply it on the following woodpecker toy model.

Hint: When an event is detected within a step, use Hermite interpolation to locate it more precisely.

Application: The woodpecker toy problem. The problem was first described in [2], and the following formulation is taken from [1]:

The woodpecker toy consists of

- a rod where the woodpecker glides down
- a sleeve which glides down the rod
- a woodpecker connected to the sleeve by a spring

The woodpecker has two degrees of freedom: one rotational with respect to the rod (angle θ) and one translational down the rod (height z of the point S). The sleeve hinders the woodpecker from moving down for angles $|\theta| > |\theta_{K1}|$. In that case, there are no movements in the vertical direction. The motion can be subdivided into 5 phases:

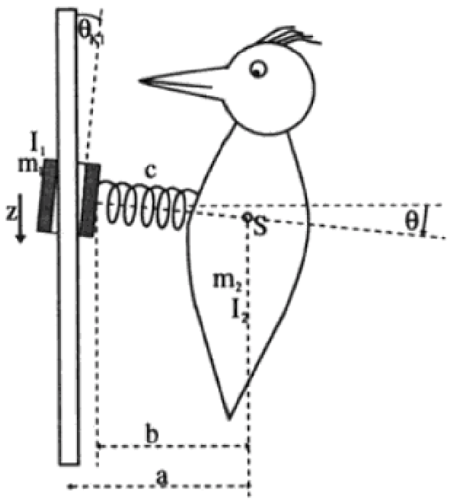


Figure 1: The woodpecker toy [1].

a) $\theta > \theta_{K1}$:
The woodpecker swings to the right until it reaches the maximal amplitude and then back to the left. The sleeve blocks the vertical movement. This phase ends when $\theta = \theta_{K1}$.

b) $-\theta_{K1} < \theta < \theta_{K1}$:
The woodpecker moves down due to gravitation. The rotation is counterclockwise ($\theta' < 0$). The phase ends when $\theta = -\theta_{K1}$, when the impact of the sleeve ends the movement in the z-direction.

c) $-\theta_{K2} < \theta < -\theta_{K1}$:
The woodpecker swings to the left until its beak hits the rod at $\theta = -\theta_{K2}$.

d) $-\theta_{K2} < \theta < -\theta_{K1}$:
The woodpecker bounces back again. The sleeve blocks vertical movement until $\theta = -\theta_{K1}$.

e) $-\theta_{K1} < \theta < \theta_{K1}$:
The self blocking phase ends, and the woodpecker moves down again, until $\theta = \theta_{K1}$ where the sleeve blocks, and the movements start from phase a) again.

The sleeve is modelled as a massless joint. Only small oscillations are considered. The following system of differential equations is obtained; see [2] for details:
Phases a), c) and d): the sleeve blocks.

$$(I_2 + m_2 b^2)\theta'' = -c\theta + m_2 b g_{gr} \tag{8}$$

Table 1: Constants in the woodpecker toy model.

a=0.025 [m]	b=0.015 [m]	
$m_1 = 0.0003$ [kg]	$m_2 = 0.0045$ [kg]	
$d_1 = 0.18335$	d_2=0.04766	
$\theta_{K1} = 10^\circ$	$\theta_{K2} = 12^\circ$	
$I_2 = 7 \cdot 10^{-7}$ [kg m^2]	$c = 0.0056$ [Nm]	$g_{gr} = 9.81$ [m/s^2]

Phases b) and e): with vertical movements.

$$\left(I_2 + m_2 b^2 \left(1 - \frac{m_2}{m_1 + m_2}\right)\right)\theta'' = -c\theta, \tag{9a}$$

$$(m_1 + m_2)z'' + m_2 b\theta'' = (m_1 + m_2)g_{gr}. \tag{9b}$$

In addition, we have the following impacts:

- Impact of the sleeve at the top:
$$\theta'_+ = (1 - d_2)\left(\theta'_- + \frac{m_2 b}{I_2 + m_2 b^2} z'_-\right).$$
- Impact of the sleeve at the bottom:
$$\theta'_+ = (1 - d_1)\left(\theta'_- + \frac{m_2 b}{I_2 + m_2 b^2} z'_-\right).$$
- Impact when the beak hits the rod:
$$\theta'_+ = -\theta'_-.$$

The constants are given in Table 1.

2.2 Differential-algebraic equations and the differentiator circuit

Consider a system of differential equations on the form

$$Mv' = F(t, v), \qquad v(t_0) = v_0, \tag{10}$$

where M is a constant, singular $m \times m$ matrix of rank d. We know that there exist nonsingular matrices P and Q such that

$$PMQ = \begin{bmatrix} I_d & 0 \\ 0 & 0 \end{bmatrix},$$

where I_d is the identity matrix of dimension d. By the transformations

$$\begin{bmatrix} y \\ z \end{bmatrix} = Q^{-1}v, \qquad \begin{bmatrix} f(t,y,z) \\ g(t,y,z) \end{bmatrix} = PF(v),$$

the system can be rewritten in semi-explicit form:

$$y' = f(t,y,z), \qquad y(t_0) = y_0, \tag{11a}$$

$$0 = g(t,y,z), \qquad z(t_0) = z_0. \tag{11b}$$

If g_z (the Jacobian of g with respect to z) is nonsingular, then, according to the implicit function theorem, the second equation $g = 0$ can be solved with respect to z. Thus $z(t) = G(t, y(t))$ for some function G. Inserting this into the first equation yields the system

$$y'(t) = f(t,y,G(t,y)) = \widetilde{f}(t,y), \tag{12a}$$

$$z(t) = G(t,y), \qquad \text{where } z \text{ is the solution of } g(t,y,z) = 0. \tag{12b}$$

The first equation is just an ODE. Equation (10) is called a *differential-algebraic equation (DAE)*, and when g_z is nonsingular, it is of *index 1*. The initial value $v_0 = Q \cdot [y_0^\top \;\; z_0^\top]^\top$ is *consistent* if it satisfies the algebraic constraint

$$g(t_0, y_0, z_0) = 0.$$

Example: The transformation (10) ⇒ (11) ⇒ (12) can be illustrated with the following example:

$$\begin{aligned} v_1' - v_2' &= -v_3 \\ -v_1' + v_2' &= -4v_2 \\ 0 &= v_1 + \sin t \end{aligned} \quad \Rightarrow \quad \begin{aligned} y' &= -z_2 \\ 0 &= 4y - 4z_1 - z_2 \\ 0 &= z_1 + \sin t \end{aligned} \quad \Rightarrow \quad \begin{aligned} y' &= -4y - 4\sin t \\ z_1 &= -\sin t \\ z_2 &= 4y + 4\sin t \end{aligned}$$

where $y = v_1 - v_2$, $z_1 = v_1$ and $z_2 = v_3$. The initial values $v(0) = [0 \;\; -1 \;\; 4]^\top$ are consistent.

But, even if a formulation (12) in principle exists, it is not necessarily easy to find an analytic expression like we have here. Hence, we would prefer to solve the DAE in its original form (10). Method (3) can be applied to (10) by

$$V_1 = v_n, \tag{13a}$$

$$MV_i = Mv_n + h\sum_{j=1}^{i-1} a_{ij} F\left(t_n + c_j h, V_j\right) + h\gamma F\left(t_n + c_i h, V_i\right) \qquad \text{for} \quad i = 2,3,4, \tag{13b}$$

$$v_{n+1} = V_4, \tag{13c}$$

$$le_{n+1} = V_4 - V_3. \qquad \text{(Local error estimate)} \tag{13d}$$

Assignment: Let consistent values $v_n = Q \cdot [y_n^\top \;\; z_n^\top]^\top$ be given. Moreover, let v_{n+1} be the solution found by (13), and y_{n+1}, z_{n+1} be the solutions when (3) is applied to the ODE in (12). Show the similar result for the error estimate.

Rewrite your code to solve index 1 DAEs of the form (10). Use the example above as a test problem, and confirm that the two formulations are equivalent. Then use your rewritten code to solve the differentiator circuit:

Application: The differentiator circuit:

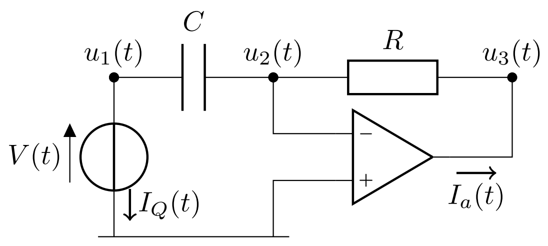


Figure 2: The differentiator circuit.

The differentiator circuit in Figure 2 consists of an operational amplifier, a capacitance, and a resistor. The purpose of the circuit is to differentiate the input signal $V(t)$ (with a change of sign).

Circuit models consist of models for each element in the circuit, glued together by Kirchoff's current and voltage laws. Several techniques for automatic generation of the circuit equations exist. One of the more popular methods is modular nodal analysis, which applied to the differentiator circuit results in the following DAE:

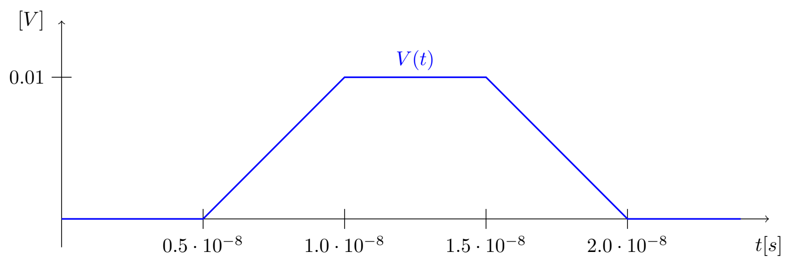
$$\begin{aligned} Cu_1' - Cu_2' &= -I_Q, \\ -Cu_1' + Cu_2' &= -\frac{1}{R}(u_2 - u_3), \\ 0 &= \frac{1}{R}(u_2 - u_3) + I_a, \\ 0 &= -Au_2 - u_3, \\ 0 &= -u_1 + V. \end{aligned}$$

Confirm that this is an index 1 DAE.

The parameters of the circuit are chosen as $C = 10^{-12}$ [F], $R = 10^4$ [Ω] and $A = 300$, as suggested in [3]. A typical time scale for the problem is 10^{-8} [s]. As input signal $V(t)$ you can use

$$V(t) = 0.01 \cdot \sin\left(2\pi \cdot 10^8 t\right)\ [V],$$

or the signal



or you may try your own signal. Choose initial values yourself, but make sure they are consistent.

NB! This is a small and linear system, so it can quite easily be rewritten as an ODE with 4 constraints. You may want to do this for testing, but your code should work on the original problem.

References

[1] Edda Eich-Soellner and Claus Führer. *Numerical methods in multibody dynamics*. European Consortium for Mathematics in Industry. B. G. Teubner, Stuttgart, 1998.

[2] F. Pfeiffer. Mechanische systeme mit unstetigen übergängen. *Ingenieur-Archiv*, 54:232–240, 1984.

[3] O. Schein and G. Denk. Numerical solution of stochastic differential-algebraic equations with applications to transient noise simulation of microelectronic circuits. *J. Comput. Appl. Math.*, 100(1):77–92, 1998.