P. Grohs
W. Wu

Spring Term 2016

Numerical Analysis II

ETH Zürich
D-MATH

Problem Sheet 4

Problem 4.1 Averaged Vector Field Single Step Method

The *averaged-vector-field single step method* for the autonomous differential equation $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ with $\mathbf{f} : D \subset \mathbb{R}^d \mapsto \mathbb{R}^d$ smooth, is implicitly defined by

$$\mathbf{y}_1 = \mathbf{y}_0 + h \int_0^1 \mathbf{f}(\mathbf{y}_0 + \tau(\mathbf{y}_1 - \mathbf{y}_0))\,\mathrm{d}\tau. \tag{4.1.1}$$

(4.1a) Show that for every $\mathbf{y}_0 \in D$ the corresponding $\mathbf{y}_1$ is well-defined by (4.1.1), i.e. such $\mathbf{y}_1$ exists for $h > 0$ sufficiently small.

(4.1b) Show that (4.1.1) defines a single step method of order at least 2.

HINT: Replace $\mathbf{f}(\mathbf{y}_0 + \tau(\mathbf{y}_1 - \mathbf{y}_0))$ in (4.1.1) by its Taylor expansion about $\mathbf{y}_0$.

(4.1c) Write down the defining equation for the iteration of Newton's method for the solution of (4.1.1).

HINT: Rewrite the problem into "$F(y_1) = 0$" shape and apply Newton's method to this root-finding problem.

Problem 4.2 Implementation of the Gaussian Collocation Method

As explained in the section [NODE, Def. 2.2.1] of the lecture notes, a one-step collocation scheme $\mathbf{y}_1 = \Psi^{t_0,t_0+h}\mathbf{y}_0$ for the solution of the ODE $\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y})$, with collocation points

$$t_0 \le t_0 + c_1 h < \cdots < t_0 + c_s h \le t_0 + h = t_1, \qquad s \in \mathbb{N},$$

can be described by

$$\begin{aligned} \mathbf{k}_i &= \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j) \\ \mathbf{y}_1 &:= \mathbf{y}_h(t_1) = \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i \end{aligned} \qquad \text{with} \qquad \begin{aligned} a_{ij} &= \int_0^{c_i} L_j(\tau)\,\mathrm{d}\tau \\ b_i &= \int_0^1 L_i(\tau)\,\mathrm{d}\tau\ . \end{aligned} \tag{4.2.1}$$

Here

$$L_i(\xi) = \prod_{\substack{j=1 \\ j\neq i}}^{s} \frac{\xi - c_j}{c_i - c_j}, \qquad i = 1, \ldots s$$

are the Lagrange polynomials. The coefficients a_{ij}, $1 \le i, j \le s$ are collected in the matrix $\mathbf{A} \in \mathbb{R}^{s\times s}$.

(4.2a) Write a MATLAB function

```
function [A,b] = collCoeff(c)
```

which takes the relative positions $c_i \in [0,1]$ of the collocation points as a vector $\mathbf{c} \in \mathbb{R}^s$ and returns the coefficients of the matrix $\mathbf{A} \in \mathbb{R}^{s\times s}$ and the vector $\mathbf{b} \in \mathbb{R}^s$ with $(\mathbf{A})_{ij} = a_{ij}$ and $(\mathbf{b})_i = b_i$.

HINT: Familiarize yourself with the MATLAB functions `polyint` und `polyval`. `vander` may also be of use.

(4.2b) If the collocation points c_i are the roots of the Legendre polynomial of n^{th} degree for the interval $[0,1]$, the resulting method is called the *Gaussian collocation one-step method*. This method inherits the convergence properties of the Gaussian quadrature, meaning its convergence order is $2n$. Create a MATLAB function

```
function c = GaussNodes(n)
```

which returns the roots of the Legendre polynomial of n^{th} degree on the interval $[0,1]$.
HINT: The Golub-Welsch algorithm returns the roots of the Legendre polynomial of n^{th} degree on the interval $[-1,1]$. To be specific, The roots $c_1,\dots,c_n$ of the Legendre polynomial of degree n for the interval $[-1,1]$ are the eigenvalues of the matrix

$$\begin{pmatrix} 0 & b_1 & & \\ b_1 & 0 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & 0 \end{pmatrix},$$

where $b_j := j(4j^2-1)^{-1/2}$.

(4.2c) The Gaussian collocation one-step method is implicit and is usually used with Newton's method. Let $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ be a function, of which we want to find the roots. Modify the code MATLAB function `newton(x0, F, DF)` provided in the template, so that the function performs `nNewton` steps of Newton's method.

(4.2d) Implement the implicit Gaussian collocation method of order 4: find the coefficients using the Matlab function `[A,b]=collCoeffs(c)` and the vector $\mathbf{b} \in \mathbb{R}^s$ with $(\mathbf{A})_{ij} = a_{ij}$ and $(\mathbf{b})_i = b_i$ and subproblem (4.2b) and rephrase the method as a root-finding problem. Apply your implementation of Newton's method from subproblem (4.2c) to it. Complete the template `impGauss.m` with inputs: the initial value `y0` of the IVP, the right hand side `f` of the initial value problem, the Jacobian of the right hand side `Df`, the end point `T`, the number of steps `Nh` sand `nNewton`, the number of iterations of Newton's method.

(4.2e) Consider the initial value problem

$$\dot{\mathbf{y}} = \exp(\mathbf{y})\sin(\mathbf{y}); \quad \mathbf{y}(0) = \pi/4.$$

Find the absolute error of the Gaussian collocation method at `T=0.5` by varying both the number of steps $N_h = 2^i$, $i = 4,\dots,8$ and the number of Newton iterations `nNewton`$= 1,2,3$. Plot

the error against the number of steps in logarithmic scale and estimate the algebraic convergence order with the MATLAB function `polyfit`. Use the template `GaussConv.m`.
HINT: You can find a reference solution with `ode45`. Set the relative and absolute tolerance to 10^{-12}.

Problem 4.3 Implicit Mid-Point Rule vs. Explicit and Implicit Euler Methods

We consider the ordinary differential equation of second order $\ddot{y} = -\sin y$ for the initial values $y(0) = \pi/2$, $\dot{y}(0) = 0$.

(4.3a) Rewrite the problem into an IVP of first-order ODE system. Try to solve the IVP using the implicit mid-point rule [NODE, Ex. 2.1.5],

$$\mathbf{y}_{k+1} = \mathbf{y}_k + hf\left(\tfrac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right). \tag{4.3.1}$$

There are two implementations of the implicit mid-point rule. The first version uses a fixed-point iteration analogous to the one implemented in `ImpEulerStep.m`. Finish the matlab template `ImpMidPointStep.m` by referring `ImpEulerStep.m`. The function `ImpMidPointStep` is called in `ImpMidPointSolve.m`.

The second version uses half an implicit Euler step and half an explicit Euler step. The two half-steps are given by

$$\begin{aligned} \boldsymbol{y}_{k+\frac{1}{2}} &:= \boldsymbol{y}_k + \frac{h}{2} f\left(\boldsymbol{y}_{k+\frac{1}{2}}\right), \quad \text{implicit Euler with step size } \tfrac{h}{2}, \\ \boldsymbol{y}_{k+1} &:= \boldsymbol{y}_{k+\frac{1}{2}} + \frac{h}{2} f\left(\boldsymbol{y}_{k+\frac{1}{2}}\right), \quad \text{explicit Euler with step size } \tfrac{h}{2}. \end{aligned}$$

Prove that the second version is equivalent to the first version, i.e. one step of the implicit mid-point rule

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + hf\left(\frac{\boldsymbol{y}_k + \boldsymbol{y}_{k+1}}{2}\right),$$

and implement it using template `ImpMidPointSolveWithEuler.m`.

(4.3b) Now try to solve the problem using classical Runge-Kutta method of fourth order (RK4, see [NODE, Ex. 2.3.7]. Complete the code using template `Rk4Solve.m`.

(4.3c) Now finish the template `s8a3Script.m` and run it. You will get two graphs, showing the solution you get from the explicit Euler method, the implicit Euler method, the implicit mid-point rule and classical Runge-Kutta method of fourth order. Describe the observed behavior of the solutions from these four different algorithms.

Published on 14 March 2016.

To be submitted by 22 March 2016.

References

[NODE] Lecture Notes for the course “Numerical Methods for Ordinary Differential Equations”.

Last modified on April 22, 2016