Technische Universität Berlin **Summer Term 2014**
Fakultät II - Mathematik und Naturwissenschaften
Institut für Mathematik
Andreas Steinbrecher, Jeroen Stolwijk

Numerical Analysis II

Homework Sheet 7

Exercises

Tutorial on June 2

1. Problem
Prove the following theorem:

Theorem. *Multi-step methods of the type* (r, l)*, i.e.*

$$u_{m+k} - u_{m+r-l} = h\sum_{i=0}^{r} \beta_i f(t_{m+i}, u_{m+i}), \quad \textit{where} \quad \beta_i = \int_{-l}^{k-r} \prod_{j=0, j\neq i}^{r} \frac{r+s-j}{i-j} ds$$

are consistent of order (at least) $p = r + 1$*.*

Hint: When estimating the truncation error, you will need to use that the β_i are determined by an interpolating polynomial, whose error in each point is given by some formula (e.g. Numerical Analysis I).

2. Problem
Consider the implicit two-step method

$$u_{m+2} + \alpha u_{m+1} = h\left(\beta_0 f(t_m, u_m) + \beta_1 f(t_{m+1}, u_{m+1}) + \beta_2 f(t_{m+2}, u_{m+2})\right) \tag{1}$$

and choose the coefficients such that it is consistent of order $p = 2$. Start with the definition of the local discretization error $\tau(t, h, f)$ and use Taylor series expansions.

3. Problem
Consider again the implicit two-step method (1). Can we choose the coefficients such that we obtain the order of consistency $p = 3$?

Hint: Use a theorem given in the lecture.

Theoretical Homework

Due: June 10, during the lecture

1. Problem (12 Points)
Determine for $k = 2$ all linear multi-step methods with consistency order $p = 2$. Start with the definition of the local discretization error $\tau(t, h, f)$ and use Taylor series expansions.

2. Problem (8 Points)
Consider the multi-step method

$$u_{m+3} + \alpha_2 u_{m+2} + \alpha_0 u_m = h(\beta_1 f(t_{m+1}, u_{m+1}) + \beta_2 f(t_{m+2}, u_{m+2}) + \beta_3 f(t_{m+3}, u_{m+3})).$$

Determine the coefficients α_0, α_2, β_1, β_2, and β_3, such that the maximum order of consistency is achieved.

Hint: Use a theorem given in the lecture.

Total Points: 20

Programming Homework

Due: June 16 (first chance) or June 23 (second chance)

Attention: You can only submit your program on June 23 if you presented a programming approach on June 16!

Write a program that solves an ordinary differential equation $\dot{y}(t) = f(t, y(t))$, $y(t_0) = y_0$, on an interval $[t_0, t_0 + a]$ using the Gragg-Bulirsch-Stoer method. Your function should be called with the line

$$[\mathtt{h}, \mathtt{t}, \mathtt{u}] = \mathtt{gragg}(\mathtt{fun}, \mathtt{t0}, \mathtt{y0}, \mathtt{k}, \mathtt{a}, \mathtt{N}).$$

Here, `fun` should be a MATLAB function handle corresponding to the right hand side $f(t, y)$ of the differential equation. It should also be possible for `y` and `f` to be vectors of $\mathbb{R}^n$. The parameter $\mathtt{t0} = t_0$ is the lower interval bound, $\mathtt{y0} = y_0 \in \mathbb{R}^n$ is the initial value, `k` is the number of steps, $\mathtt{a} = a$ is the interval length, and `N` is the number of extrapolation stages.
The routine should return the basic step size $H = \mathtt{h} = \mathtt{a}/\mathtt{k}$, the vector of grid points $\mathtt{t} = [t_0, t_1, \ldots, t_k]$, and the corresponding approximated solution $\mathtt{u} = [u_0, u_1, \ldots, u_k]$.
You should compute the values $y_0 = S_0, S_1, S_2, \ldots, S_k$ by

$$\begin{aligned}
\tilde{u}_0 &= y_0 \\
\tilde{u}_1 &= \tilde{u}_0 + Hf(t_0, \tilde{u}_0) \\
\tilde{u}_{m+1} &= \tilde{u}_{m-1} + 2Hf(t_m, \tilde{u}_m), \qquad m = 1, 2, \ldots, k \\
S_m &= (\tilde{u}_{m-1} + 2\tilde{u}_m + \tilde{u}_{m+1})/4, \quad m = 1, 2, \ldots, k.
\end{aligned}$$

In order to extrapolate (given $N > 1$, else we have no extrapolation), for each step you should also compute additional approximations using the local step sizes $h_i = H/2^{i-1}$ for $i = 1, 2, \ldots, N$, followed by computing the extrapolated value $\tilde{u}_{m+1}$ using the Neville-Aitken scheme. Note that the sequence for the extrapolation is hence given by $(n_i)_{i=1}^N = (2^{i-1})_{i=1}^N$.
To test your program, use the initial value problems

1. $\dot{y}(t) = 2y(t) - e^t, \quad y(0) = 2, \quad t \in [0, 1]$,
   with exact solution $y(t) = e^t + e^{2t}$,

2. $\dot{y}(t) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} y(t), \quad y(0) = [1\ 0]^T, \quad t \in [0, 1]$,
   with exact solution $y(t) = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \end{bmatrix} e^{3t} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} e^t$,

3. $\dot{y}(t) = -\tan(t)y(t), \quad y(0) = 1, \quad t \in [0, 3]$,
   with exact solution $y(t) = \cos(t)$,

and compare the method without extrapolation to the one with extrapolation in terms of accuracy and arithmetic complexity. For the case **without** extrapolation ($N = 1$), use the numbers of steps $k = 10 \cdot 2^{i-1}$ for $i = 1, 2, \ldots, 10$. For the case **with** extrapolation ($N > 1$), use the fixed number of basic steps $k = 10$ and take N approximation stages for $N = 2, 3, \ldots, 10$ (i.e., use N different local step sizes $-$ the h_i defined above $-$ for each value of N). How can you explain your program's behavior for the third test problem?
Further, compare both methods to the classical Runge-Kutta method (without extrapolation) that you implemented in the last programming assignment using again the numbers of steps $k = 10 \cdot 2^{i-1}$ for $i = 1, 2, \ldots, 10$.

Last updated on May 23, 2014.