

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 12-3: Singly Diagonally Implicit Runge-Kutta Method

SDIRK methods (Singly Diagonally Implicit Runge-Kutta methods) are distinguished by Butcher schemes with a lower triangular coefficient matrix $\mathfrak{A}$ whose diagonal entries are all the same:

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|cccc} c_1 & \gamma & & \cdots & 0 \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} & \gamma \\ \hline & b_1 & \cdots & b_{s-1} & b_s \end{array}, \quad (12.3.1)$$

with $\gamma \neq 0$.

Familiarity with [Lecture → Section 12.3.3] is required. This problem has a focus on stability theory.

In this problem the scalar linear initial value problem of second order

$$\ddot{y} + \dot{y} + y = 0, \quad y(0) = 1, \quad \dot{y}(0) = 0 \quad (12.3.2)$$

should be solved numerically using a family of 2-stage SDIRK method described by the Butcher scheme

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-\gamma & 1-2\gamma & \gamma \\ \hline & 1/2 & 1/2 \end{array}. \quad (12.3.3)$$

(12-3.a)  (5 min.) Explain the benefit of using SDIRK-SSMs compared to using Gauss-Radau RK-SSMs as introduced in [Lecture → Ex. 12.3.4.18]. In what situations will this benefit matter much?

HINT 1 for (12-3.a): Recall that in every step of an implicit RK-SSM we have to solve a non-linear system of equations for the increments, see [Lecture → Rem. 12.3.3.7].

┘

SOLUTION of (12-3.a):

For SDIRK methods the Butcher matrix is lower triangular. In this case the increment equations/stage equations can be solved separately and sequentially, computing $\mathbf{k}_1 \rightarrow \mathbf{k}_2 \rightarrow \dots \rightarrow \mathbf{k}_s$ in that order.

So instead of solving a big $sd \times sd$ non-linear system of equations for a generic implicit Runge-Kutta single-step method, we need only solve s smaller $d \times d$ non-linear systems.

▲

(12-3.b)  (10 min.) State the equations for the increments $\mathbf{k}_1$ and $\mathbf{k}_2$ of the Runge-Kutta method (12.3.3) applied to the initial value problem corresponding to the differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$.

SOLUTION of (12-3.b):

The increments $\mathbf{k}_i$ are given by:

$$\begin{aligned}\mathbf{k}_1 &= f(t_0 + h\gamma, \mathbf{y}_0 + h\gamma\mathbf{k}_1), \\ \mathbf{k}_2 &= f(t_0 + h(1 - \gamma), \mathbf{y}_0 + h(1 - 2\gamma)\mathbf{k}_1 + h\gamma\mathbf{k}_2)\end{aligned}$$

▲

(12-3.c) 📏 (20 min.) Show that the stability function $S(z)$ of the SDIRK method (12.3.3) is given by

$$S(z) = \frac{1 + z(1 - 2\gamma) + z^2(1/2 - 2\gamma + \gamma^2)}{(1 - \gamma z)^2}$$

and plot the stability domain using the supplied MATLAB's function `stabdomSDIRK.m` or C++ code `stabdomSDIRK.cpp`.

SOLUTION of (12-3.c):

The stability function $S(z)$ of a method is derived by applying the method to the scalar, linear test equation

$$\dot{y}(t) = \lambda y(t) .$$

The solution can be written as

$$y_{k+1} = S(z)y_k ,$$

where $S(z)$ is the stability function and $z := h\lambda$.

In the case of the SDIRK method, we get:

$$\begin{aligned}k_1 &= \lambda(y_k + h\gamma k_1) \\ k_2 &= \lambda(y_k + h(1 - 2\gamma)k_1 + h\gamma k_2) .\end{aligned}$$

Therefore:

$$\begin{aligned}k_1 &= \frac{\lambda}{1 - h\lambda\gamma} y_k, \\ k_2 &= \frac{\lambda}{1 - h\lambda\gamma} (y_k + h(1 - 2\gamma)k_1) .\end{aligned}$$

Furthermore:

$$y_{k+1} = y_k + \frac{h}{2}(k_1 + k_2) .$$

With $z := h\lambda$ and by plugging in k_1 and k_2 we arrive at:

$$y_{k+1} = \underbrace{\left(1 + \frac{z}{2(1 - \gamma z)} \left(2 + \frac{z(1 - 2\gamma)}{1 - \gamma z}\right)\right)}_{=: S(z)} y_k .$$

Hence:

$$S(z) = \frac{2(1 - \gamma z)^2 + 2z(1 - \gamma z) + z^2(1 - 2\gamma)}{2(1 - \gamma z)^2}$$

and after collecting the powers of z in the numerator we get

$$S(z) = \frac{1 + z(1 - 2\gamma) + z^2(\gamma^2 - 2\gamma + \frac{1}{2})}{(1 - \gamma z)^2}.$$



(12-3.d) (25 min.) Find out whether for $\gamma = 1$ the SDIRK RK-SSM (12.3.3) is

- A-stable,
- L-stable.

Argue, based on the following version of the maximum principle for analytic functions:

Lemma 12.3.4. Maximum principle for rational functions

If a *rational function* $z \in \mathbb{C} \mapsto R(z)$ has no pole in the left half plane $\mathbb{C}^- := \{z \in \mathbb{C} : \operatorname{Re} z \leq 0\}$, then **either**

$$\max\{|R(z)| : z \in \mathbb{C}^-\} = \max\{|R(z)| : \operatorname{Re} z = 0\}$$

or

$$\min\{|R(z)| : z \in \mathbb{C}^-\} = \min\{|R(z)| : \operatorname{Re} z = 0\},$$

that is, the function attains on the imaginary axis either its maximum or its minimum over the whole left half plane.

SOLUTION of (12-3.d):

Given the previous sub-problem, for $\gamma = 1$ the stability function is:

$$S_1(z) := \frac{1 - z - \frac{z^2}{2}}{(1 - z)^2} \quad (12.3.5)$$

Verification of the A-stability of (12.3.5):

By definition [Lecture $\rightarrow$ § 12.3.4.4], we need to show that $|S_1(z)| \leq 1$ for all $z \in \mathbb{C}^- := \{z \in \mathbb{C} : \operatorname{Re} z < 0\}$. In order to do this, we consider the stability function on the imaginary axis:

$$\begin{aligned} |S_1(iy)|^2 &= \frac{|1 - iy - (iy)^2/2|^2}{|1 - iy|^4} \\ &= \frac{|1 - iy + y/2|^2}{|1 - iy|^4} \\ &= \frac{(1 + y^2/2)^2 + y^2}{(1 + y^2)^2} \\ &= \frac{1 + 2y^2 + y^4/4}{1 + 2y^2 + y^4} \leq 1, \quad y \in \mathbb{R} \end{aligned}$$

Since the only pole ($z = 1$) of the rational function $S_1(z)$ lies in the positive half plane of $\mathbb{C}$, the function S_1 is holomorphic in the left half plane. Furthermore, S_1 is bounded by 1 on the boundary of this half

plane (i.e. on the imaginary axis). So, by the maximum principle for holomorphic functions (hint), S_1 is bounded on the entire left half plane by 1. This implies in particular that S_1 is A -stable.

Verification of the L-stability of (12.3.5):

S_1 is not L -stable (cf. definition [Lecture $\rightarrow$ § 12.3.4.11]), because:

$$\lim_{\operatorname{Re} z \rightarrow -\infty} |S_1(z)| = \lim_{\operatorname{Re} z \rightarrow -\infty} \left| \frac{1 - z - z^2/2}{1 - 2z + z^2} \right| = \frac{1}{2} \neq 0$$

▲

(12-3.e) 🕒 (5 min.) Formulate (12.3.2) as an initial value problem for a linear first order system for the function $\mathbf{z}(t) = (y(t), \dot{y}(t))^T$.

SOLUTION of (12-3.e):

Define $z_1 = y$, $z_2 = \dot{y}$, then the initial value problem

$$\ddot{y} + \dot{y} + y = 0, \quad y(0) = 1, \quad \dot{y}(0) = 0 \quad (12.3.6)$$

is equivalently to the first order system

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= -z_1 - z_2 \end{aligned} \quad (12.3.7)$$

with initial values $z_1(0) = 1$ and $z_2(0) = 0$.

▲

(12-3.f) 🕒 (20 min.) [depends on Sub-problem (12-3.e)]

Implement a C++ function

```
StateType sdirkStep(const Eigen::VectorXd &z0, double h, double gamma);
```

that realizes one step of the method (12.3.3) for the linear differential equation (12.3.2), starting from the initial state z_0 and returning the state after a single step of size h .

SOLUTION of (12-3.f):

Let $\mathbf{A} := \begin{pmatrix} 0 & 1 \\ -1 & -1 \end{pmatrix}$. Then:

$$\begin{aligned} k_1 &= \mathbf{A} \mathbf{y}_0 + h\gamma \mathbf{A} k_1 \\ k_2 &= \mathbf{A} \mathbf{y}_0 + h(1 - 2\gamma) \mathbf{A} k_1 + h \mathbf{A} k_2, \end{aligned}$$

so

$$\begin{aligned} k_1 &= (\mathbf{I} - h\gamma \mathbf{A})^{-1} \mathbf{A} \mathbf{y}_0 \\ k_2 &= (\mathbf{I} - h\gamma \mathbf{A})^{-1} (\mathbf{A} \mathbf{y}_0 + h(1 - 2\gamma) \mathbf{A} k_1). \end{aligned}$$

C++11-code 12.3.8: Solution of (12-3.f)

```

2  ///! \brief One step of autonomous IVP  $y'' + y' + y = 0$ ,  $[y(0), y'(0)] = z_0$  using
3  ///! SDIRK method Use SDIRK method for first order ode  $z' = f(z)$ . Steps of size
4  ///! h. \tparam Eigen::VectorXd type of solution space y and initial data y0
5  ///! \param[in] z0 initial data  $z(0)$  \param[in] h size of the step \param[in]
6  ///! gamma parameter \return next step z1
7  Eigen::Vector2d sdirkStep(const Eigen::Vector2d &z0, double h, double gamma) {
8      // Matrix A for evaluation of f
9      Eigen::Matrix2d A;
10     A << 0., 1., -1., -1.;
11
12     // Reuse factorization
13     auto A_lu = (Eigen::Matrix2d::Identity() - h * gamma * A).partialPivLu();
14     Eigen::Vector2d az = A * z0;
15
16     // Increments
17     Eigen::Vector2d k1 = A_lu.solve(az);
18     Eigen::Vector2d k2 = A_lu.solve(az + h * (1 - 2 * gamma) * A * k1);
19
20     // Next step
21     return z0 + h * 0.5 * (k1 + k2);
22 }

```



(12-3.g) (25 min.) [depends on Sub-problem (12-3.f)]

Realize a C++ function

```
double cvgSDIRK(void);
```

to conduct a numerical experiment, which gives an indication of the order of the method (with $\gamma = \frac{3+\sqrt{3}}{6}$) for the initial value problem from (12.3.3). Choose $y_0 = [1, 0]^T$ as initial value, $T=10$ as end time and $N=20, 40, 80, \dots, 10240$ as numbers of equidistant timesteps. Track the maximal Euclidean error norm on the temporal mesh, tabulate it and use it as the basis for the estimation of the rate of algebraic convergence by means of the supplied function `polyfit`.

SOLUTION of (12-3.g):

C++11-code 12.3.9: Timestepping for (12.3.2) with SDIRK implicit RK-SSM

```

2  ///! \brief Solve autonomous IVP  $y'' + y' + y = 0$ ,  $[y(0), y'(0)] = z_0$  using SDIRK
3  ///! method Use SDIRK method for first order ode  $z' = f(z)$ , with N equidistant
4  ///! steps \tparam Eigen::VectorXd type of solution space  $z = [y, y']^T$  and initial
5  ///! data  $z_0 = [y(0), y'(0)]$  \param[in] z0 initial data  $z(0)$  \param[in] N number
6  ///! of equidistant steps \param[in] T final time of simulation \param[in] gamma
7  ///! parameter \return vector containing each step of  $z_k$  (y and  $y'$ )
8  std::vector<Eigen::Vector2d>
9  sdirkSolve(const Eigen::Vector2d &z0, unsigned int N, double T, double gamma) {
10     // Solution vector
11     std::vector<Eigen::VectorXd> res(N + 1);
12     // Equidistant step size
13     const double h = T / N;
14     // Push initial data
15     res.at(0) = z0;

```

```

16 // Main loop
17 for (unsigned int i = 1; i <= N; ++i) {
18     res.at(i) = sdirkStep(res.at(i - 1), h, gamma);
19 }
20 return res;
21 }

```

C++11-code 12.3.10: Testing convergence of SDIRK method for an IVP for (12.3.2)

```

2 // Initial data z0 = [y(0), y'(0)]
3 Eigen::Vector2d z0;
4 z0 << 1, 0;
5 // Final time
6 const double T = 10;
7 // Parameter
8 const double gamma = (3. + std::sqrt(3.)) / 6.;
9 // Mesh sizes
10 std::vector<int> N = {20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240};
11 // Exact solution (only y(t)) given z0 = [y(0), y'(0)] and t
12 auto yex = [&z0](double t) {
13     return 1. / 3. * std::exp(-t / 2.) *
14         (3. * z0(0) * std::cos(std::sqrt(3.) * t / 2.) +
15          std::sqrt(3.) * z0(0) * std::sin(std::sqrt(3.) * t / 2.) +
16          2. * std::sqrt(3.) * z0(1) * std::sin(std::sqrt(3.) * t / 2.));
17 };
18
19 // Store old error for rate computation
20 double errold = 0;
21 std::cout << std::setw(15) << "n" << std::setw(15) << "maxerr"
22     << std::setw(15) << "rate" << std::endl;
23 // Loop over all meshes
24 for (unsigned int i = 0; i < N.size(); ++i) {
25     int n = N.at(i);
26     // Get solution
27     auto sol = sdirkSolve(z0, n, T, gamma);
28     // Compute error
29     double err = std::abs(sol.back()(0) - yex(T));
30
31     // Print table
32     std::cout << std::setw(15) << n << std::setw(15) << err;
33     if (i > 0)
34         std::cout << std::setw(15) << std::log2(errold / err);
35     std::cout << std::endl;
36
37     // Store old error
38     errold = err;
39 }

```

N	maximal error
20	0.000283567
40	5.05272e-05
80	7.73715e-06
160	1.07013e-06
320	1.40565e-07
640	1.80048e-08
1280	2.27798e-09
2560	2.86467e-10
5120	3.59161e-11
10240	4.49619e-12

◁ Norm of maximal error on temporal mesh.

We observe an asymptotic (in the last steps) decrease of the error norm by a factor of ≈ 8 whenever the number of (equidistant) timesteps is doubled. Thus we observe *algebraic convergence* with rate 3 in the timestep size h :

$$\text{errornorm} = O(h^3) .$$

This hints that the SDIRK single-step method has order 3 .

The numerically estimated order of convergence is 3.



End Problem 12-3 , 110 min.