Take-home Exam 02686
Scientific Computing for Differential Equations
Spring 2025

John Bagterp Jørgensen

April 3, 2025

You can work on the problems in groups (2-3 persons). In the report you must state the contribution of each group member. The deadline for handing in the report is May 23, 2025, 16:00. The report must be uploaded to Learn and a printed copy of the pdf must be handed in at Office 303B-110. To Learn you must upload the following files: 1) a pdf of the report; 2) a zip file with a) all latex/word code used to generate the pdf; b) all matlab (Python, Julia) code used to generate the results. The grading will be based on this report.

1 Test Problems of ODE Solvers

In this problem you must implement a number of test problems for the solvers you will develop in the following questions. You should also demonstrate the solution of these problems using Matlab's ODE solvers (`ode45` and `ode15s`). We consider initial value problems of the type

$$\dot{x}(t) = f(t, x(t)) \qquad x(t_0) = x_0 \tag{1}$$

that may come with additional inputs, e.g.

$$\dot{x}(t) = f(t, x(t), p) \qquad x(t_0) = x_0 \tag{2a}$$

$$\dot{x}(t) = f(t, x(t), u, p) \qquad x(t_0) = x_0 \tag{2b}$$

$$\dot{x}(t) = f(t, x(t), u, d, p) \qquad x(t_0) = x_0 \tag{2c}$$

In these equations p may be a parameter vector (or structure), u may be a vector of manipulated inputs, and d may be a disturbance vector.

1.1 The Prey-Predator Model

1. Provide the mathematical equations for the Prey-Predator Problem

2. Provide Matlab code for the prey-predator problem. It should be in the form function `[f] = PreyPredatorModel(t,x,p)`.

3. Provide Matlab code for the Jacobian of the prey-predator problem. It should be in the form function `[J] = PreyPredatorJacobian(t,x,p)`.

4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian. It should be in the form function
   `[f,J] = PreyPredatorModelJacobian(t,x,p)`
5. Define a test simulation (and describe it) using ode45 and ode15s. You must provide the matlab code and the figures illustrating the numerical solution.

1.2 The Van der Pol Model

1. Provide the mathematical equations for the Van Der Pol Problem
2. Provide Matlab code for the Van der Pol problem.
3. Provide Matlab code for the Jacobian of the Van der Pol problem.
4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian.
5. Define a test simulation (and describe it) using `ode45` and `ode15s`. You must define a problem that is non-stiff and a problem that is stiff. You must provide the matlab code and the figures illustrating the numerical solution. You should discuss and illustrate which type of solvers are most suitable for non-stiff problems and which type of solvers are most suitable for stiff problems.

1.3 Chemical Reaction in adiabatic reactors

To have realistic test problems we consider the chemical system described in Wahlgreen et al. (2020) and Jørgensen et al. (2020). We consider the reaction in a continuous stirred tank reactor (CSTR) as well as in a dispersive plug flow reactor (PFR). The parameters are given in Table 1 and in Wahlgreen et al. (2020).

Let $C = [C_A; C_B; C_T] = [C_A; C_B; T]$ (3-state model) or $C = [C_T] = [T]$ (1-state model).

The reaction that occur is an exothermic reaction

$$A + 2B \rightarrow \frac{1}{2}C + \frac{1}{2}D + 2E \quad r = k(T)C_A C_B \quad \Delta H_r = -560 \text{ kJ/mol} \tag{3}$$

The structure for the CSTR reactor models is ($C = C(t)$)

$$\dot{C} = (C_{in} - C)F/V + R(C) \quad C(t_0) = C_0 \tag{4}$$

The structure for the PFR models is ($C = C(t,z)$)

$$\partial_t C = -\partial_z N(C) + R(C) \quad C(t_0, z) = C_0(z) \tag{5}$$

with the boundary conditions (no diffusion at the boundaries)

$$N(t,0) = N_{in} = vC_{in} \tag{6a}$$
$$N(t,L) = N_{out} = vC(t,L) \tag{6b}$$

The transport (flux) is modeled as advection (convection) and diffusion

$$N = N_a + N_d \quad N_a = vC \quad N_d = -D \odot \partial_z C \quad v = F/A \tag{7}$$

The chemical production can be expressed by the chemical reaction model

$$R = \nu' r,\ \nu = \begin{bmatrix} -1 & -2 & \beta \end{bmatrix},\ r = r(C) = k(T)C_A C_B \tag{8a}$$

Table 1: Parameters of the PFR models.

Parameter	Value	Unit	Description
ρ	1.0	kg/L	Density
c_P	4.186	kJ/(kg·K)	Specific heat capacity
k_0	exp(24.6)	L/(mol·s)	Arrhenius constant
E_a/R	8500.0	K	Activation energy
ΔH_r	-560.0	kJ/mol	Reaction enthalphy
L	10	m	Reactor length
A	0.1	m^2	Reactor cross-sectional area
D	[0.1; 0.1; 0.1]	m^2/s	Diffusion coefficients

where

$$k(T) = k_0 \exp\left(\frac{-E_a/R}{T}\right) \tag{8b}$$

$$\beta = \frac{-\Delta H_r}{\rho c_P} \tag{8c}$$

The differential equations for the CSTR (4) as well as the PFR (5) upon spatial discretization can be expressed as ordinary differential equations in the forms (1) and (2).

1.3.1 Model of the CSTR (3 state model)

Model an exothermic reaction in an adiabatic continuous stirred tank reactor (CSTR) as a system of ordinary differential equations (see papers). Simulate the system with the Matlab ODE solvers. Plot the results for different flow rates. Compute the steady states as function of the flow rate. Compute the corresponding eigenvalues of the system. Plot the steady state temperature as function of the flow rate.

1. Provide the mathematical equations
2. Provide Matlab code for the problem.
3. Provide Matlab code for the Jacobian of the problem.
4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian.
5. Define a test simulation (and describe it) using `ode45` and `ode15s`. You must provide the matlab code and the figures illustrating the numerical solution.

1.3.2 Model of the CSTR (1 state model)

Model an exothermic reaction in an adiabatic continuous stirred tank reactor (CSTR) as a 1-state system of ordinary differential equations (see papers). Simulate the system with the Matlab ODE solvers. Plot the results for different flow rates. Compute the steady states as function of the flow rate. Compute the corresponding eigenvalues of the system. Plot the steady state temperature as function of the flow rate.

1. Provide the mathematical equations
2. Provide Matlab code for the problem.

3. Provide Matlab code for the Jacobian of the problem.
4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian.
5. Define a test simulation (and describe it) using `ode45` and `ode15s`. You must provide the matlab code and the figures illustrating the numerical solution.

1.3.3 Model of the PFR (3 states per cell)

Model an exothermic reaction in an adiabatic plug flow reactor (PFR). The resulting system should be a advection-diffusion-reaction system with appropriate boundary conditions. Discretize the system using the methods of lines (FD/FV discretization). Simulate the resulting system using Matlab's ODE solvers (`ode45` and `ode15s`). Plot the results for different flow rates. Compute the steady states as function of the flow rate. Compute the corresponding eigenvalues of the system. Plot the steady state temperature as function of the flow rate.

1. Provide the mathematical equations
2. Provide Matlab code for the problem.
3. Provide Matlab code for the Jacobian of the problem.
4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian.
5. Define a test simulation (and describe it) using `ode45` and `ode15s`. You must provide the matlab code and the figures illustrating the numerical solution.

1.3.4 Model of the PFR (1 state per cell)

Model an exothermic reaction in an adiabatic plug flow reactor (PFR). The resulting system should be a advection-diffusion-reaction system with appropriate boundary conditions. Discretize the system using the methods of lines (FD/FV discretization). Simulate the resulting system using Matlab's ODE solvers (`ode45` and `ode15s`). Plot the results for different flow rates. Compute the steady states as function of the flow rate. Compute the corresponding eigenvalues of the system. Plot the steady state temperature as function of the flow rate.

1. Provide the mathematical equations
2. Provide Matlab code for the problem.
3. Provide Matlab code for the Jacobian of the problem.
4. Provide Matlab code for simultaneous evaluation of the model and the Jacobian.
5. Define a test simulation (and describe it) using `ode45` and `ode15s`. You must provide the matlab code and the figures illustrating the numerical solution.

2 Explicit ODE solver

Consider the initial value problems (1) and (2). In this problem you must describe the explicit Euler method for solution of these problems, implement a solver in Matlab (or another programming language e.g. Python or Julia), and test it for the test problems described in Problem 1.

1. Provide a verbal and mathematical decription of the explicit Euler algorithm (i.e. provide an algorithm for it in your report and explain how you get from the differential equations to the numerical formulas). Discuss the order of the method and provide a stability plot for the method.
2. Implement an algorithm in Matlab for the explicit Euler method with fixed time-step and provide this in your report. Use a format that enables syntax highlighting.
3. Implement an algorithm in Matlab for the explicit Euler method with adaptive time step and error estimation using step doubling. Implement using a syntax similar to the Matlab ode-solver syntax (such that you can easily swithch between your own solvers and Matlab's solvers)
4. Test your algorithms for the test problems that you implemented in Problem 1 and compare the results from your algorithms with the results you get using some of Matlab's ODE solvers (`ode45` and `ode15s`)

The report should contain figures and a discussion of your algorithm for different tolerances and step sizes. Discuss the interfaces (the way you call) for the ODE solver that you made.

3 Implicit ODE solver

Consider the initial value problems (1) and (2). In this problem you must describe the Implicit Euler method for solution of these problems, implement a solver in Matlab (or another programming language e.g. Python or Julia), and test it for the test problems described in Problem 1.

1. Provide a verbal and mathematical description of the implicit Euler algorithm (i.e. provide an algorithm for it in your report and explain how you get from the differential equations to the numerical formulas). Discuss the order of the method and provide a stability plot for the method.
2. Implement an algorithm in Matlab for the implicit Euler method with fixed time-step and provide this in your report. Use a format that enables syntax highlighting.
3. Implement an algorithm in Matlab for the implicit Euler method with adaptive time step and error estimation using step doubling. Implement using a syntax similar to the Matlab ode-solver syntax (such that you can easily swithch between your own solvers and Matlab's solvers)
4. Test your algorithms for the test problems that you implemented in Problem 1 and compare the results from your algorithms with the results you get using some of Matlab's ODE solvers (`ode45` and `ode15s`).

The report should contain figures and a discussion of your algorithm for different tolerances and step sizes. Discuss the interfaces (the way you call) for the ODE solver that you made.

In particular you should discuss how you provide the Jacobian, $J = \frac{\partial f}{\partial x}(t,x)$ to the solver. You can consider computing it in a separate function as Matlab's ODE solvers do or you can have an interface of the type function `[f,J] = MyODEfun(t,x)`.

4 Solvers for SDEs

We consider now stochastic differential equations in the form (or another form compatible with the IVP problems described in Problem 1)

$$dx(t) = f(t,x(t),u,d,p_f)dt + g(t,x(t),u,d,p_g)d\omega(t) \quad d\omega(t) \sim N_{iid}(0,Idt) \qquad (9)$$

where $x \in R^{n_x}$ and ω is a stochastic variable with dimension n_w. p_f and p_g are parameters for $f : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_{p_f}} \mapsto \mathbb{R}^{n_x}$ and $g : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_{p_g}} \mapsto \mathbb{R}^{n_x \times n_w}$ (i.e. the result of g is a matrix of size $n_x \times n_w$).

1. Make a function in Matlab that can realize a multivariate standard Wiener process
2. Implement the explicit-explicit method with fixed step size
3. Implement the implicit-explicit method with fixed step size
4. Describe your implementations and the idea you use in deriving the methods. Provide matlab code for your implementations.
5. Make SDE extensions of your test problems in Problem 1.
6. Test your SDE solver on the SDE test problems.

5 Classical Runge-Kutta method

Consider the initial value problems (1) and (2). In this problem you must describe the classical Runge-Kutta method for solution of these problems, implement a solver in Matlab (or another programming language e.g. Python or Julia), and test it for the test problems described in Problem 1.

1. Describe the classical Runge-Kutta algorithm (i.e. provide an algorithm for it in your report and explain how you get from the differential equations to the numerical formulas). Discuss the order of the method and provide a stability plot for the method.
2. Implement an algorithm in Matlab for the classical Runge-Kutta method with fixed time-step and provide this in your report. Use a format that enables syntax highlighting.
3. Implement an algorithm in Matlab for the classical Runge-Kutta method with adaptive time step and error estimation using step doubling.

4. Test your algorithms for the test problems that you implemented in Problem 1 and compare the results from your algorithms with the results you get using some of Matlab's ODE solvers (`ode45` and `ode15s`)

The report should contain figures and a discussion of your algorithm for different tolerances and step sizes. Discuss the interfaces (the way you call) for the ODE solver that you made.

6 Dormand-Prince 5(4)

Consider the initial value problems (1) and (2). In this problem you must describe the Dormand-Prince 5(4) method for solution of these problems, implement a solver in Matlab (or another programming language e.g. Python or Julia), and test it for the test problems described in Problem 1.

1. Describe the Dormand-Prince method (DOPRI54) method with adaptive time step size. Discuss the order of the method and provide a stability plot for the method.
2. Implement an algorithm in Matlab for the DOPRI54 method with adaptive time step size. Provide the code in your report. Use a format that enables syntax highlighting. Comment on the code.
3. Test your algorithms for the test problems that you implemented in Problem 1 and compare the results from your algorithms with the results you get using some of Matlab's ODE solvers (`ode45` and `ode15s`)

The report should contain figures and a discussion of your algorithm for different tolerances and step sizes. Discuss the interfaces (the way you call) for the ODE solver that you made.

7 ESDIRK23

Consider the initial value problems (1) and (2). In this problem you must describe the ESDIRK23 method for solution of these problems, implement a solver in Matlab (or another programming language e.g. Python or Julia), and test it for the test problems described in Problem 1.

1. Using the order conditions and other conditions derive the ESDIRK23 method.
2. Plot the stability region of the ESDIRK23 method. Is it A-stable? Is it L-stable? Discuss the practical implications of the stability region of ESDIRK23.
3. Implement ESDIRK23 with variable step size.
4. Test your algorithms for the test problems that you implemented in Problem 1 and compare the results from your algorithms with the results you get using some of Matlab's ODE solvers (`ode45` and `ode15s`)

The report should contain figures and a discussion of your algorithm for different tolerances and step sizes. Discuss the interfaces (the way you call) for the ESDIRK23 ODE solver that you made. In particular you should test the ESDIRK23 solver on some of the test problems that arise as a discretization of partial differential equations (the PFR test problems).

References

J. B. Jørgensen, T. K. S. Ritschel, D. Boiroux, E. Schroll-Fleischer, M. R. Wahlgreen, M. K. Nielsen, and J. K. Huusom. Simulation of NMPC for a Laboratory Adiabatic CSTR with an Exothermic Reaction. In *Proceedings of 2020 European Control Conference*, pages 202–207, 2020.

M. R. Wahlgreen, E. Schroll-Fleischer, D. Boiroux, T. K. S. Ritschel, H. Wu, J. K. Huusom, and J. B. Jørgensen. Nonlinear Model Predictive Control for an Exothermic Reaction in an Adiabatic CSTR. *IFAC PapersOnLine*, 53-1:500–505, 2020.