

Assignment 3

Mathematics for Robotics

C. Gummaluru, A. Shoelling

University of Toronto

This assignment is part of the *Fall 2024* offering of *ROB310: Mathematics for Robotics* at the University of Toronto. Please review the information below carefully. Submissions that do not adhere to the guidelines below may lose marks.

Release Date: September 30, 2024 - 00:00
Due Date: October 06, 2024 - 23:59
Weight: 3.75% or 0%
Late Penalties: -10% per day and -100% after 3 days

Submission Information:

- Download this document as a PDF and fill in your information below.
- Answer the questions to the best of your ability.
- Typeset or **neatly hand-write** your answers, labelling the questions identically to this document. We recommend using L^AT_EX, but will accept the use of other typesetting software (such as Microsoft Word or Google Docs).
- Save each question as a separate PDF document. Do NOT include the questions.
- Submit your PDFs on Crowdmark titled `lastname-firstname-rob310f24-a3-q#.pdf`.

Problem 3.1

Let $x : \mathbb{R} \rightarrow \mathbb{R}$, which is defined according to the following ODE:

$$\dot{x} = -x.$$

Using a time-step of $T > 0$, write an implicit right-rectangular (backward Euler) equation for approximating x_{k+1} given x_k , where $x_k = x(kT)$.

Problem 3.2

Let $x : \mathbb{R} \rightarrow \mathbb{R}$, which is defined according to the following ODE:

$$\dot{x}(t) = f(x(t)),$$

where $f : \mathbb{R} \rightarrow \mathbb{R}$. Using a time-step of $T > 0$, consider solving the ODE numerically using the following scheme:

$$x_{k+1} = x_k + h(\theta f(x_k) + (1 - \theta)f(x_{k+1}))$$

which weighs the slope at t_k and t_{k+1} with θ and $1 - \theta$, respectively, where $0 \leq \theta \leq 1$.

Problem 3.2.a

Determine the order of the local error for $\theta = 0, 0.5$, and 1 . Provide a detailed derivation.

Problem 3.2.b

Using the test equation, $\dot{x}(t) = \lambda x(t)$, where $\lambda = \sigma + j\omega$, with $\sigma, \omega \in \mathbb{R}$ and $\sigma < 0$, determine the stability of the algorithm in Problem 3.2.a as a function of θ and $\kappa = \lambda T$.

Problem 3.3

Let $y : \mathbb{R} \rightarrow \mathbb{R}$ be defined according to the following ODE:

$$a^2 \ddot{y}(t) + y(t) = Ku(t).$$

For this problem, assume that $u \equiv 2$, $a = 2$, $K = 1$, $y(0) = 0$ and $\dot{y}(0) = 1$.

Problem 3.3.a

Solve the differential equation analytically to find a closed-form solution for y . Plot y as a function of t .

Problem 3.3.b

Determine the matrices, $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ in the standard-form representation of the ODE:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{Ax}(t) + \mathbf{Bu}(t) \\ y(t) &= \mathbf{Cx}(t) + \mathbf{Du}(t)\end{aligned}$$

Problem 3.3.c

Use Python to solve the ODE to determine y using each of the schemes below using the given step-sizes. Determine the error between the analytic solution and numerical solution. Plot the error as a function of time and compare how the error changes as the step-size and scheme varies. Make sure your plots are labelled appropriately.

- `ode1` with $T \in \{0.05, 0.1, 0.15\}$
- `ode2` with $T \in \{0.3, 0.4, 0.5\}$
- `ode4` with $T \in \{0.1, 0.2, 0.3, 0.4\}$

You may make use of the following code:

```
# Vectorized ODE1 Method
def ode1_vectorized(f, y0, t0, t_end, dt):
    t = np.arange(t0, t_end, dt)
    y = np.zeros((len(t), len(y0))) # Initialize an array for y
    y[0] = y0
    for i in range(1, len(t)):
        y[i] = y[i - 1] + f(t[i - 1], y[i - 1]) * dt
    return t, y

# Vectorized ODE2 Method
def ode2_vectorized(f, y0, t0, t_end, dt):
    t = np.arange(t0, t_end, dt)
    y = np.zeros((len(t), len(y0))) # Initialize an array for y
    y[0] = y0
    for i in range(1, len(t)):
        k1 = f(t[i - 1], y[i - 1])
        k2 = f(t[i - 1] + dt, y[i - 1] + k1 * dt)
        y[i] = y[i - 1] + (k1 + k2) * dt / 2
    return t, y

# Vectorized ODE4 Method
def ode4_vectorized(f, y0, t0, t_end, dt):
    t = np.arange(t0, t_end, dt)
    y = np.zeros((len(t), len(y0))) # Initialize an array for y
    y[0] = y0
    for i in range(1, len(t)):
        k1 = f(t[i - 1], y[i - 1])
        k2 = f(t[i - 1] + dt / 2, y[i - 1] + k1 * dt / 2)
```

```
k3 = f(t[i - 1] + dt / 2, y[i - 1] + k2 * dt / 2)
k4 = f(t[i - 1] + dt, y[i - 1] + k3 * dt)
y[i] = y[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4) * dt / 6
return t, y
```