Prof. Fabien Evrard & Fabian Dettenrieder
AE 410/CSE 461 – Computational Aerodynamics
Spring 2024

Extra Homework – Optional

Due date: April 4, 2024

This homework is optional. You can receive a full grade for the course without completing this homework.

If you choose to submit a report for this homework, you will receive a grade. If superior to any of the grades you have obtained for HW1-4, it will replace the lowest of these grades.

In HW3, you have developed a general purpose solver for systems of ODEs of the form

$$\begin{cases} \dfrac{\mathrm{d}\mathbf{U}(t)}{\mathrm{d}t} &= \mathbf{F}\left(t, \mathbf{U}(t)\right) \\ \\ \mathbf{U}(0) &= \mathbf{U}_0 \end{cases} \tag{1}$$

and applied it to the solution of the semi-discrete problem

$$\begin{cases} \dfrac{\mathrm{d}\mathbf{U}(t)}{\mathrm{d}t} &= \mathbf{A}\mathbf{U}(t) \\ \\ \mathbf{U}(0) &= \mathbf{U}_0 \end{cases} \tag{2}$$

with $\mathbf{A} = -a\mathbf{D}$ a constant-coefficient matrix and with $\mathbf{D}$ a discrete differential operator. This semi-discrete problem produces an approximate solution to the linear 1D advection problem

$$\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} &= 0 \\ \\ u(x,0) &= u_0(x) \end{cases} \tag{3}$$

In your code from HW3, the system of equations (2) is solved using a ready-made Initial Value Problem (IVP) solver (e.g., `SciPy`'s `solve_ivp`, or `Matlab`'s `ode45` function). An example of such code is provided at the end of this document.

In this assignment, you will be asked to write your own implementation of an IVP solver that is an explicit discrete time integrator. This function, in effect, will discretize the temporal space $[0, T]$ into M time instances separated by a timestep Δt, and compute $\mathbf{U}(t + \Delta t)$ as

$$\mathbf{U}(t + \Delta t) = \mathbf{U}(t) + \mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) \tag{4}$$

where $\mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right)$ is the approximation of $\mathbf{F}\left(t, \mathbf{U}(t)\right)$ integrated from t to $t + \Delta t$,

$$\mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) \simeq \int_t^{t+\Delta t} \mathbf{F}\left(t, \mathbf{U}(t)\right)\ \mathrm{d}t \tag{5}$$

hence the name "discrete time *integrator*". For instance, the trivial approximation

$$\mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) = \mathbf{F}\left(t, \mathbf{U}(t)\right)\Delta t \tag{6}$$

leads to the first-order *(forward) Euler* method.

Problem 1 · In your code from HW3, replace the IVP solver by your own implementation of an explicit time integrator. This explicit time integrator must be able to consider the following schemes:

- Euler:

$$\mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) = \mathbf{F}\left(t, \mathbf{U}(t)\right) \Delta t$$

- Runge-Kutta 2 (RK2):

$$\begin{aligned} k_1 &= \mathbf{F}\left(t, \mathbf{U}(t)\right) \\ k_2 &= \mathbf{F}\left(t + \Delta t, \mathbf{U}(t) + k_1 \Delta t\right) \\ \mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) &= \left(\frac{k_1 + k_2}{2}\right) \Delta t \end{aligned}$$

- Runge-Kutta 3 (RK3):

$$\begin{aligned} k_1 &= \mathbf{F}\left(t, \mathbf{U}(t)\right) \\ k_2 &= \mathbf{F}\left(t + \Delta t, \mathbf{U}(t) + k_1 \Delta t\right) \\ k_3 &= \mathbf{F}\left(t + \Delta t/2, \mathbf{U}(t) + (k_1 + k_2)\Delta t/4\right) \\ \mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) &= \left(\frac{k_1 + k_2 + 4k_3}{6}\right) \Delta t \end{aligned}$$

- Runge-Kutta 4 (RK4): (**this scheme is only required for <u>4 credit-hours</u> students**)

$$\begin{aligned} k_1 &= \mathbf{F}\left(t, \mathbf{U}(t)\right) \\ k_2 &= \mathbf{F}\left(t + \Delta t/2, \mathbf{U}(t) + k_1 \Delta t/2\right) \\ k_3 &= \mathbf{F}\left(t + \Delta t/2, \mathbf{U}(t) + k_2 \Delta t/2\right) \\ k_4 &= \mathbf{F}\left(t + \Delta t, \mathbf{U}(t) + k_3 \Delta t\right) \\ \mathbf{G}\left(t, \Delta t, \mathbf{U}(t)\right) &= \left(\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\right) \Delta t \end{aligned}$$

Problem 2 · For each of the schemes listed in Problem 1:

Q2.1 → Derive the stability condition for the explicit time integration of the system of ODEs (2), i.e., when $\mathbf{F}\left(t, \mathbf{U}(t)\right) = \mathbf{A}\mathbf{U}(t)$. This stability condition will be a function of $\Delta t \lambda_n$, with λ_n the nth eigenvalue of $\mathbf{A}$.

Q2.2 → Plot the corresponding stability region in the $(\Delta t\, \mathbb{R}\mathrm{e}(\lambda_n), \Delta t\, \mathbb{I}\mathrm{m}(\lambda_n))$ plane.

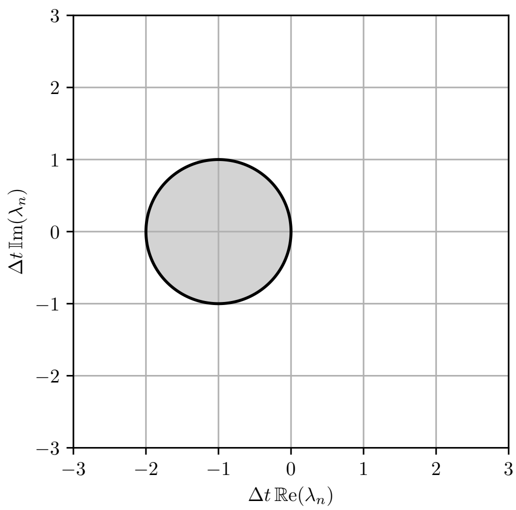
<u>For instance:</u> the stability condition for the first-order forward Euler scheme reads as

$$|1 + \Delta t \lambda_n| \leq 1, \quad \forall n$$

and its corresponding stability region can be plotted with the `Python` code

```python
import matplotlib.pyplot as plt
import numpy as np
s = np.linspace(-3,3,100,endpoint=True); re, im = np.meshgrid(s, s)
dt_lambda = re+im*1j
plt.contourf(re,im,np.abs(1+dt_lambda), [0,1], colors='lightgray')
plt.contour(re,im,np.abs(1+dt_lambda), [1], colors='k')
plt.xlabel(r'$\Delta t\,\mathrm{Re}(\lambda_n)$')
plt.ylabel(r'$\Delta t\,\mathrm{Im}(\lambda_n)$')
plt.grid()
plt.show()
```

which produces the following figure:



Problem 3 · Consider the periodic advection of a Gaussian pulse with the constant wavespeed $a = 1$, domain length $L = 1$, final time $T = 30$, $N_x = 50$ discrete grid points, and the initial condition

$$u_0(x) = \exp\left(\frac{-\left(x - \frac{L}{2}\right)^2}{2\sigma^2}\right) \quad , \quad \sigma = \frac{3}{40}$$

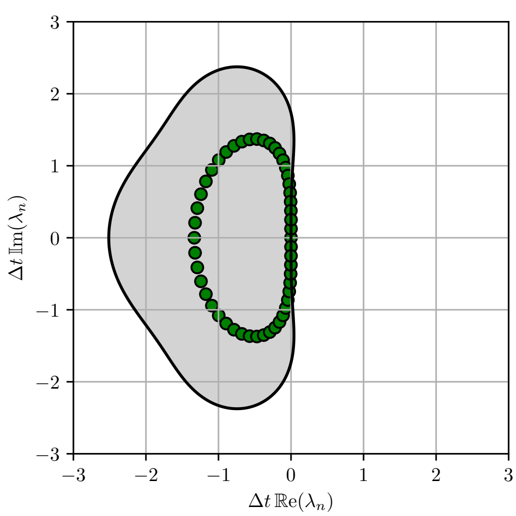
For the explicit time integration, consider the RK3 scheme with a CFL number

$$\text{CFL} = \frac{|a|\Delta t}{\Delta x} = 1$$

For each of the spatial schemes implemented in HW3 (i.e., first-order and third-order upwind, second-order and fourth-order central, and sixth-order Padé):

Q3.1 → Plot the eigenvalue spectrum of the matrix $\mathbf{A} = -a\mathbf{D}$ in the $(\Delta t\,\mathbb{R}\mathrm{e}(\lambda_n), \Delta t\,\mathbb{I}\mathrm{m}(\lambda_n))$ plane, on top of the RK3 stability region.

For instance: The eigenvalue spectrum of $\mathbf{A}$ for the third-order upwind scheme, plotted on top of the RK3 stability region, looks as follows:



Q3.2 → Compare your numerical solution to the exact expected solution at the times $t_m, m \in \{3, 6, \ldots, 30\}$.

Q3.3 → Are your numerical solutions stable? If not, qualitatively explain why.

Problem 4 · In class, we have derived several stability conditions in the form $\text{CFL} \leq \text{CFL}_{\text{max}}$ for combinations of discrete temporal and spatial differential operators applied to the numerical solution of (2).

For instance:

- For the combination of explicit Euler (time) and 1st-order upwind (space), $\text{CFL}_{\text{max}} = 1$.
- For the combination of RK3 (time) and 2nd-order central differences (space), $\text{CFL}_{\text{max}} = \sqrt{3}$.

Q4 → Solve the periodic advection problem described in Problem 3 with these two combinations of schemes, at $\text{CFL} = (1 \pm 0.05) \times \text{CFL}_{\text{max}}$. Are the results consistent with your expectations?

Problem 5 · **This problem is required for all students taking AE 410/CSE 461 for four credit hours. It is not required for students taking AE 410/CSE 461 for three credit hours.**

Q5.2 → Derive the expression of the stability limit CFL_{max} for the combination of RK4 (time) and 2nd-order central differences (space) applied to the system of equations (2).

Q5.2 → Solve the periodic advection problem described in Problem 3 with this combination of schemes, at $\text{CFL} = (1 \pm 0.05) \times \text{CFL}_{\text{max}}$. Are the results consistent with your expectations?

Submission guidelines · Instructions on how to prepare and submit your report are available on the course's Canvas page at https://canvas.illinois.edu/courses/43781/assignments/syllabus

---

Appendix · Solution code from HW3:

```python
import numpy as np
from scipy.linalg import circulant
from scipy.integrate import solve_ivp

# Operator from HW1
def D_operator_periodic(N,L,R,a):
  first_row = np.zeros(N); first_row[0:L+R+1] = a; first_row = np.roll(first_row,-L)
  return np.array(circulant(first_row)).transpose()

# Functor for the 1D advection equation
def LinearAdv1D(t,U,D):
  # Initialize velocity
  a = 1
  # Return F(t,U)
  return (-a*D)@U

# General integrator function
def Integrator(periodic,operator,problem,L,T,Nx,Nt,U0):
  ######################## Inputs of the function "Integrator" #########################
  #                                                                                    #
  # periodic : boolean flag to select periodicity (options: True of False)            #
  # operator : string to select the spatial derivative operator                        #
  #  problem : string to select the governing equations                                #
  #        L : length of the physical domain, x runs from 0 to L                       #
  #        T : length of the temporal domain, t runs from 0 to T                       #
  #       Nx : number of points to use in x                                            #
  #       Nt : number of points to use in t (for reporting the solutions)             #
```

```
  #       U0 : initial condition                                                     #
  #                                                                                  #
  ######################## Outputs of the function "Integrator" ########################
  #                                                                                  #
  #        t : the discrete time levels (in a vector of size Nt)                     #
  #        U : the solutions (in a matrix of size Nt x Nx)                           #
  #                                                                                  #
  ####################################################################################

  # Initialize spatial domain
  x  = np.linspace(0, L, Nx, endpoint=(not periodic))
  dx = x[1] - x[0]

  # Initialize temporal domain
  t  = np.linspace(0, T, Nt, endpoint=True)

  # Construct spatial matrix operator
  match (operator,periodic):
    case ('1st-order upwind',True):       # Periodic 1st-order upwind
      D = D_operator_periodic(Nx,1,0,[-1/dx,1/dx])
    case ('2nd-order central',True):      # Periodic 2nd-order central differences
      D = D_operator_periodic(Nx,1,1,[-1/(2*dx),0,1/(2*dx)])
    case ('3rd-order upwind',True):       # Periodic 3rd-order upwind
      D = D_operator_periodic(Nx,2,1,[1/(6*dx),-1/dx,1/(2*dx),1/(3*dx)])
    case ('4th-order central',True):      # Periodic 4th-order central differences
      D = D_operator_periodic(Nx,2,2,[1/(12*dx),-8/(12*dx),0,8/(12*dx),-1/(12*dx)])
    case ('6th-order Padé',True):         # Periodic 6th-order Padé
      DR = D_operator_periodic(Nx,2,2,[-1/(36*dx),-28/(36*dx),0,28/(36*dx),1/(36*dx)])
      DL = D_operator_periodic(Nx,1,1,[1/3,1,1/3])
      D = np.linalg.inv(DL)@DR
    case _:
      raise Exception("The %s operator '%s' is not yet implement!" % ('periodic' if periodic else 'non-periodic', operator))

  # Solve and return solutions!
  match problem:
    case 'LinearAdv1D':
      # Solve initial value problem; see documentation at:
      # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html
      sol = solve_ivp(LinearAdv1D, [0, T], U0, args=(D,), t_eval=t, rtol=1.0e-6, atol=1.0e-6)
      # Transpose solution vector so U has the format (Nt x Nx)
      U = sol.y.transpose()
      # Return outputs
      return t, U, D
    case _:
      raise Exception("The case '%s' is not yet implement!" % problem)

############# Periodic advection of a Gaussian pulse
import matplotlib as mpl
import matplotlib.pyplot as plt
from numpy import linalg as LA
plt.rcParams['text.usetex'] = True
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300
plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{amssymb}')

# Initialize case parameters
L = 1; T = 10; Nx = 50; Nt = 11; a = 1
# Initialize solution at t=0
x = np.linspace(0, L, Nx, endpoint=False); dx = x[1] - x[0]
sigma = 3/40; U0 = np.exp(-(x-0.5)**2/(2*sigma**2))
# Initialize list of schemes that will be tested:
listofschemes = ['1st-order upwind', '2nd-order central', '3rd-order upwind', '4th-order central', '6th-order Padé']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofschemes:
  # Run solver and plot solution
```

```
t, U, D = Integrator(True, scheme, 'LinearAdv1D',L, T, Nx, Nt, U0)
fig, ax = plt.subplots(1, 1, figsize=(7, 4))
ax.plot(x,U0,color=plt.cm.Spectral_r(0))
for j in range(1,Nt):
  ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
plt.title('Periodic advection of a Gaussian pulse using %s' % scheme)
plt.xlabel(r'$x$');plt.ylabel(r'$u(x,t)$');
fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'),ax=ax
    , orientation='vertical', label=r'$t$')
plt.savefig('solution-%s.pdf' % scheme)
plt.show()
```