

## 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

$$\begin{cases} \frac{d\mathbf{U}(t)}{dt} = \mathbf{F}(t, \mathbf{U}(t)) \\ \mathbf{U}(0) = \mathbf{U}_0 \end{cases} \quad (1)$$

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

$$\begin{cases} \frac{d\mathbf{U}(t)}{dt} = \mathbf{A}\mathbf{U}(t) \\ \mathbf{U}(0) = \mathbf{U}_0 \end{cases} \quad (2)$$

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

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

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  $\Delta t$ , and compute  $\mathbf{U}(t + \Delta t)$  as

$$\mathbf{U}(t + \Delta t) = \mathbf{U}(t) + \mathbf{G}(t, \Delta t, \mathbf{U}(t)) \quad (4)$$

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

$$\mathbf{G}(t, \Delta t, \mathbf{U}(t)) \simeq \int_t^{t+\Delta t} \mathbf{F}(t, \mathbf{U}(t)) \, dt \quad (5)$$

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

$$\mathbf{G}(t, \Delta t, \mathbf{U}(t)) = \mathbf{F}(t, \mathbf{U}(t)) \Delta t \quad (6)$$

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:

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

- Runge-Kutta 2 (RK2):

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

- Runge-Kutta 3 (RK3):

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

- Runge-Kutta 4 (RK4): (this scheme is only required for 4 credit-hours students)

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

**Solution:** See code attached in the appendix.

---

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  $\mathbf{F}(t, \mathbf{U}(t)) = \mathbf{A}\mathbf{U}(t)$ . This stability condition will be a function of  $\Delta t \lambda_n$ , with  $\lambda_n$  the  $n$ th eigenvalue of  $\mathbf{A}$ .

Q2.2 → Plot the corresponding stability region in the  $(\Delta t \operatorname{Re}(\lambda_n), \Delta t \operatorname{Im}(\lambda_n))$  plane.

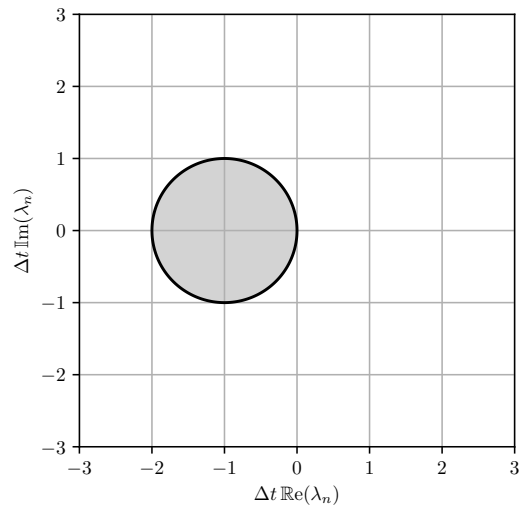
For instance: the stability condition for the first-order forward Euler scheme reads as

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

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

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

which produces the following figure:



**Solution:**

**Euler**

$$\begin{aligned}
 w_n^{m+1} &= w_n^m + \Delta t \lambda_n w_n^m = w_n^m (1 + \lambda_n \Delta t) \\
 \sigma_n &= 1 + \lambda_n \Delta t \quad \text{and} \quad |\sigma_n| \leq 1 \\
 &\quad |1 + \lambda_n \Delta t| \leq 1
 \end{aligned}$$

**Runge-Kutta 2**

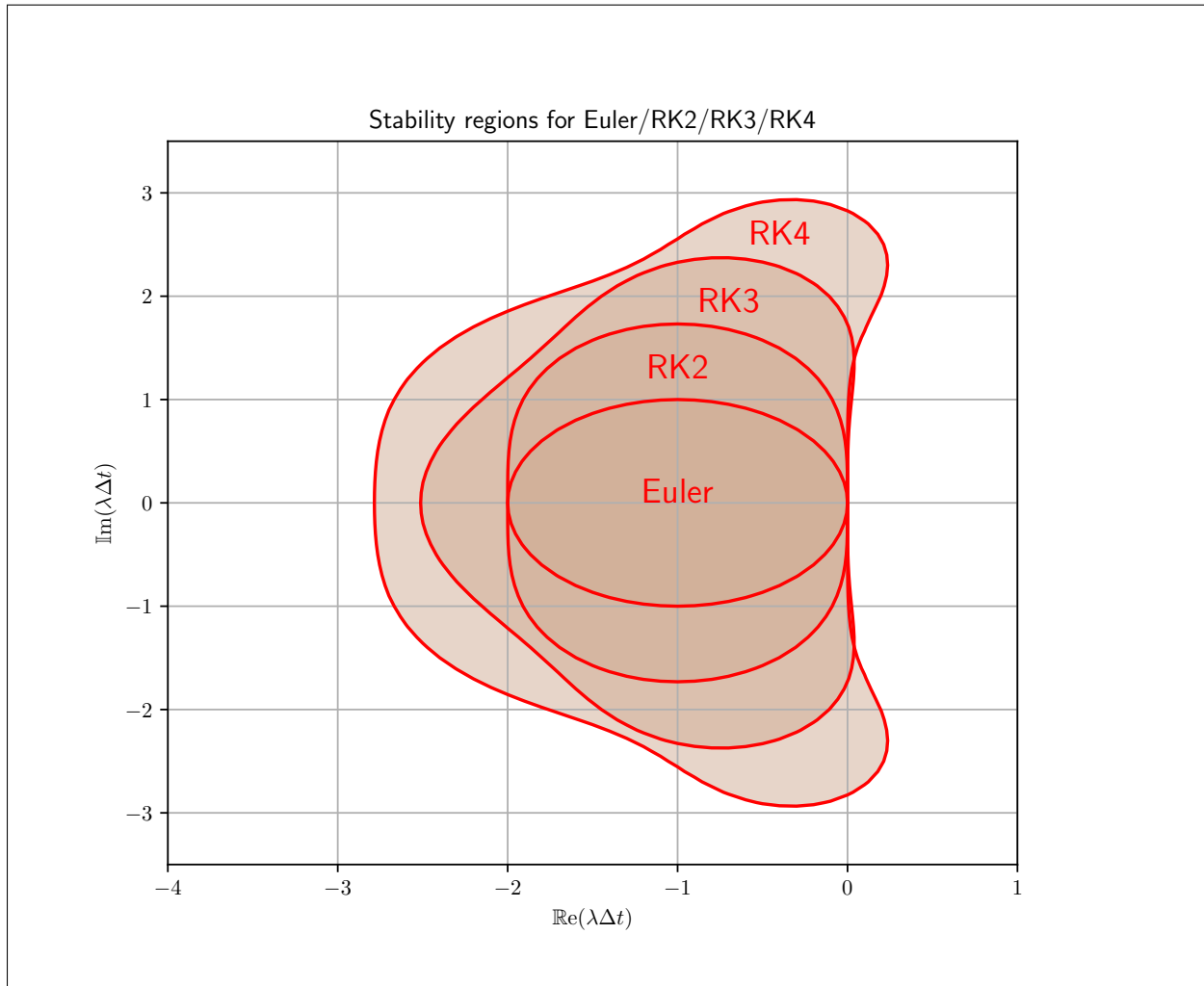
$$\begin{aligned}
 k_1 &= \lambda_n w_n^m \\
 k_2 &= \lambda_n (w_n^m + k_1 \Delta t) = w_n^m (\lambda_n + \lambda_n^2 \Delta t) \\
 w_n^{m+1} &= w_n^m + \frac{\Delta t}{2} (\lambda_n w_n^m + w_n^m (\lambda_n + \lambda_n^2 \Delta t)) \\
 &= w_n^m \left( 1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 \right) \\
 1 &\geq |\sigma_n| = \left| 1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 \right|
 \end{aligned}$$

### Runge-Kutta 3

$$\begin{aligned}
 k_1 &= \lambda_n w_n^m \\
 k_2 &= \lambda_n (w_n^m + k_1 \Delta t) = w_n^m (\lambda_n + \lambda_n^2 \Delta t) \\
 k_3 &= \lambda_n \left( w_n^m + \frac{k_1 + k_2}{4} \Delta t \right) \\
 &= w_n^m \left( \lambda_n + \frac{1}{2} \lambda_n^2 \Delta t + \frac{1}{4} \lambda_n^3 \Delta t^2 \right) \\
 w_n^{m+1} &= w_n^m + \frac{k_1 + k_2 + 4k_3}{6} \Delta t \\
 &= w_n^m \left( 1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 + \frac{1}{6} (\lambda_n \Delta t)^3 \right) \\
 1 \leq |\sigma_n| &= |1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 + \frac{1}{6} (\lambda_n \Delta t)^3|
 \end{aligned}$$

### Runge-Kutta 4

$$\begin{aligned}
 k_1 &= \lambda_n w_n^m \\
 k_2 &= \lambda_n \left( w_n^m + \frac{k_1}{2} \Delta t \right) \\
 &= w_n^m \left( \lambda_n + \frac{1}{2} \lambda_n^2 \Delta t \right) \\
 k_3 &= \lambda_n \left( w_n^m + \frac{k_2}{2} \Delta t \right) \\
 &= w_n^m \left( \lambda_n + \frac{1}{2} \lambda_n^2 \Delta t + \frac{1}{4} \lambda_n^3 \Delta t^2 \right) \\
 k_4 &= \lambda_n (w_n^m + k_3 \Delta t) \\
 &= w_n^m \left( \lambda_n + \lambda_n^2 \Delta t + \frac{1}{2} \lambda_n^3 \Delta t^2 + \frac{1}{4} \lambda_n^4 \Delta t^3 \right) \\
 w_n^{m+1} &= w_n^m + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \Delta t \\
 &= w_n^m \left( 1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 \right. \\
 &\quad \left. + \frac{1}{6} (\lambda_n \Delta t)^3 + \frac{1}{24} (\lambda_n \Delta t)^4 \right) \\
 1 \leq |\sigma_n| &= |1 + \lambda_n \Delta t + \frac{1}{2} (\lambda_n \Delta t)^2 + \frac{1}{6} (\lambda_n \Delta t)^3 + \frac{1}{24} (\lambda_n \Delta t)^4|
 \end{aligned}$$



**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

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

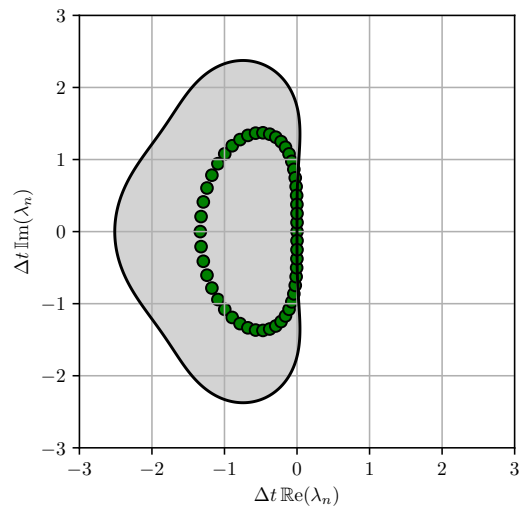
For the explicit time integration, consider the RK3 scheme with a CFL number

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

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  $(\Delta t \text{Re}(\lambda_n), \Delta t \text{Im}(\lambda_n))$  plane, on top of the RK3 stability region.

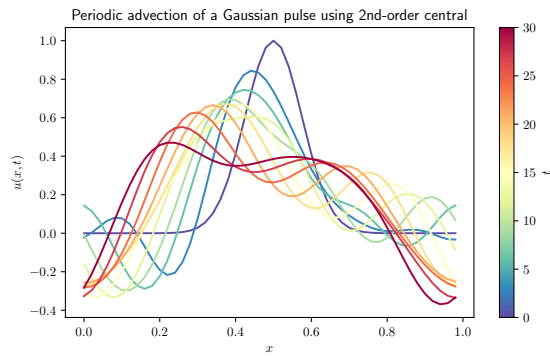
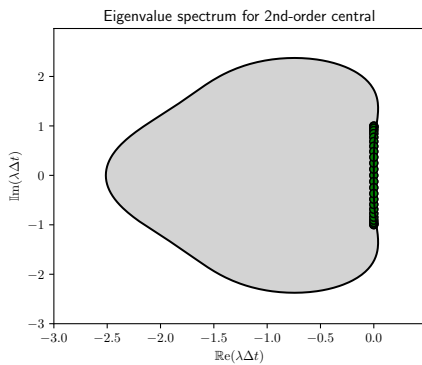
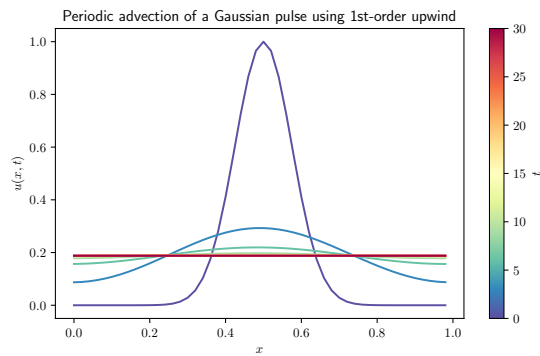
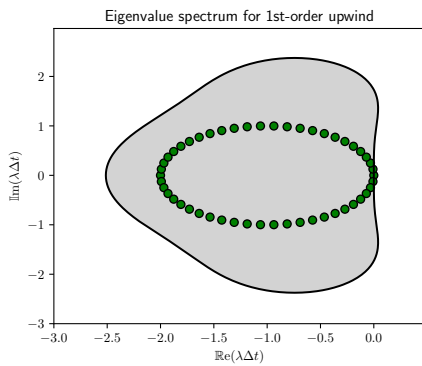
For instance: The eigenvalue spectrum of  $\mathbf{A}$  for the third-order upwind scheme, plotted on top of the RK3 stability region, looks as follows:

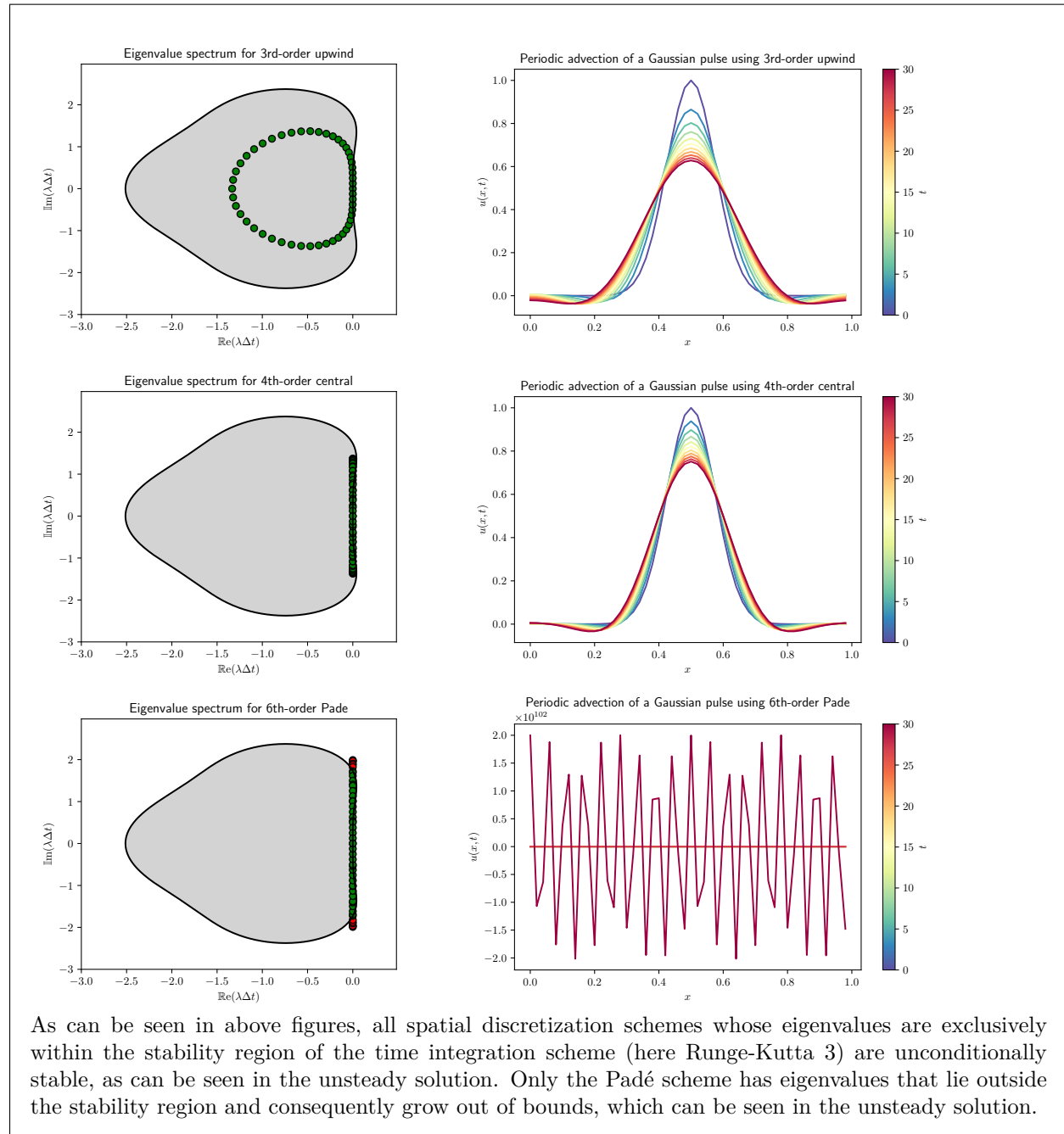


Q3.2 → Compare your numerical solution to the exact expected solution at the times  $t_m, m \in \{3, 6, \dots, 30\}$ .

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

**Solution:**



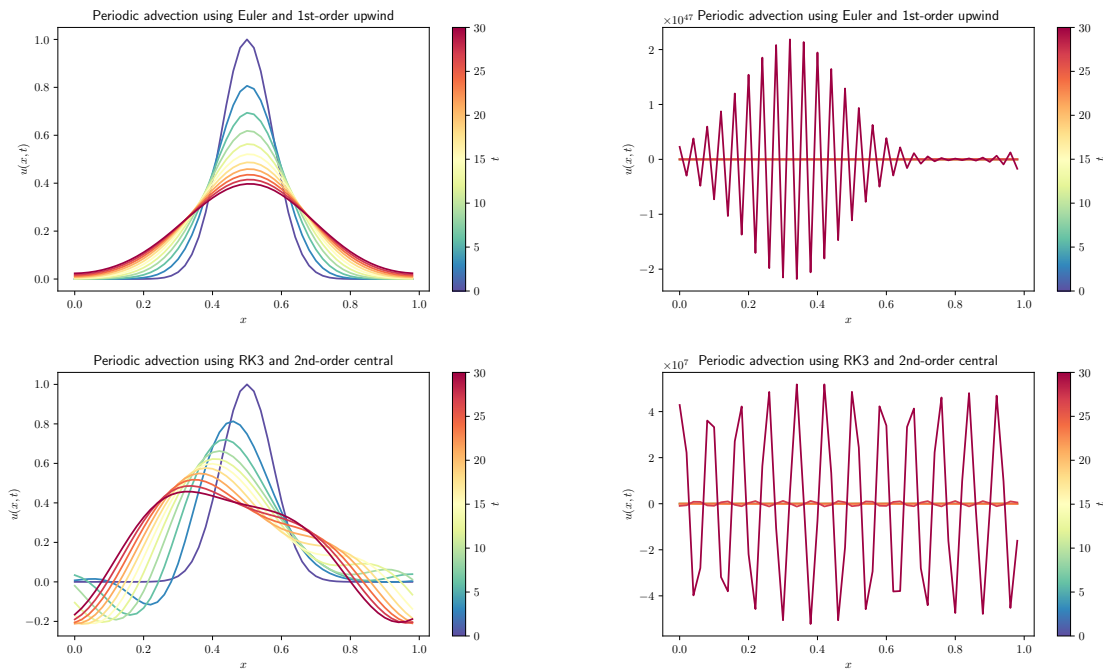


**Problem 4** · In class, we have derived several stability conditions in the form  $\text{CFL} \leq \text{CFL}_{\max}$  for combinations of discrete temporal and spatial differential operators applied to the numerical solution of (2).

For instance:

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

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

**Solution:**


The above figures clearly show that a deviation of only  $\pm 5\%$  from the CFL stability condition results in either stable or unstable simulations.

**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  $CFL = (1 \pm 0.05) \times CFL_{\max}$ . Are the results consistent with your expectations?

**Solution:**

For central finite differences, where the eigenvalues are purely imaginary, we can write  $\Delta t \lambda_n = \alpha i$  with  $\alpha \in \mathbb{R}$ , and the stability condition for Runge-Kutta 4 becomes

$$\begin{aligned}
 1 &\leq \left| 1 + \alpha i - \frac{1}{2}\alpha^2 - \frac{1}{6}\alpha^3 i + \frac{1}{24}\alpha^4 \right| \\
 &\leq \left| 1 - \frac{1}{2}\alpha^2 + \frac{1}{24}\alpha^4 + i \left( \alpha - \frac{1}{6}\alpha^3 \right) \right| \\
 &\leq \sqrt{\left( 1 - \frac{1}{2}\alpha^2 + \frac{1}{24}\alpha^4 \right)^2 + \left( \alpha - \frac{1}{6}\alpha^3 \right)^2} \\
 &\dots \\
 8 &\leq \alpha^2 \\
 -2\sqrt{2} &\leq \alpha \leq 2\sqrt{2}
 \end{aligned}$$



In other terms, if the chosen spatial discretization is neutrally stable, then the discrete time integration with Runge-Kutta 4 is stable if

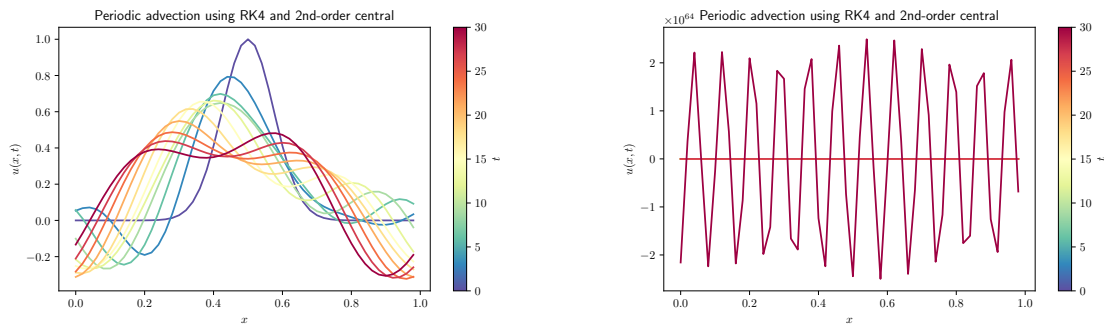
$$\left| \operatorname{Im}(\lambda_n) \right| \leq \frac{2\sqrt{2}}{\Delta t}$$

For the specific case of the second order central difference scheme, we know the analytical eigenvalues to be

$$\lambda_n = -\frac{ai}{\Delta x} \sin\left(\frac{2\pi n}{N}\right)$$

therefore the stability condition fore the RK4 scheme becomes

$$\begin{aligned} \frac{|a|}{\Delta x} &\leq \frac{2\sqrt{2}}{\Delta t} \\ \Rightarrow \text{CFL} = \frac{|a|\Delta t}{\Delta x} &\leq 2\sqrt{2} \end{aligned}$$



As can be seen in above pictures ( $\text{CFL} = 0.95 \cdot \text{CFL}_{\text{crit}}$  (left),  $\text{CFL} = 1.05 \cdot \text{CFL}_{\text{crit}}$  (right)), the unsteady solution is only stable when the CFL number is below its critical value, as expected.

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 · Full codes:

Python (0-based indexing)

```
1 # %% [markdown]
2 # Problem 2: Stability regions
3
4 # %%
5 import matplotlib.pyplot as plt
6 import matplotlib.lines as mlines
7 import matplotlib as mpl
8 import numpy as np
9 plt.rcParams['text.usetex'] = True
10 plt.rcParams['figure.dpi'] = 300
11 plt.rcParams['savefig.dpi'] = 300
12 plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{amssymb}')
13
14 xs = np.arange(-3.2,3.2,0.1)
```

```

15 ys = xs
16 X, Y = np.meshgrid(xs, ys)
17 lambda_dt = X+Y*1j
18 fig, ax = plt.subplots(1, 1, figsize=(7, 6))
19
20 plt.title('Stability regions for Euler/RK2/RK3/RK4')
21 plt.xlabel(r'$\mathbb{R}\backslash\mathrm{e}(\lambda \Delta t)$');
22 plt.ylabel(r'$\mathbb{I}\backslash\mathrm{m}(\lambda \Delta t)$');
23 plt.contourf(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3+1/24*lambda_dt**4),
24             [0,1], cmap='pink', alpha=.5)
25 plt.contour(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3+1/24*lambda_dt**4), [1],
26             colors='r')
27 plt.contourf(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3), [0,1], cmap='pink',
28             alpha=.5)
29 plt.contour(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3), [1], colors='r')
30 plt.contourf(X,Y,np.abs(1+lambda_dt), [0,1], cmap='pink', alpha=.5)
31 plt.contour(X,Y,np.abs(1+lambda_dt), [1], colors='r')
32 plt.grid()
33 plt.text(-1,0, 'Euler', dict(size=16), horizontalalignment='center', color='r')
34 plt.text(-1,1.2, 'RK2', dict(size=16), horizontalalignment='center', color='r')
35 plt.text(-0.7,1.85, 'RK3', dict(size=16), horizontalalignment='center', color='r')
36 plt.text(-0.4,2.5, 'RK4', dict(size=16), horizontalalignment='center', color='r')
37 ax.set_xlim([-4, 1])
38 ax.set_ylim([-3.5, 3.5])
39 plt.savefig('stabregions.pdf')
40 plt.show()
41
42 # %% [markdown]
43 # Problem 3: Explicit time integration of the 1D linear advection
44
45 # %%
46 from scipy.linalg import circulant
47 from scipy.integrate import solve_ivp
48
49 # Operator from HW1
50 def D_operator_periodic(N,L,R,a):
51     first_row = np.zeros(N); first_row[0:L+R+1] = a; first_row = np.roll(first_row,-L)
52     return np.array(circulant(first_row)).transpose()
53
54 # Functor for the 1D advection equation
55 def LinearAdv1D(t,U,D):
56     # Initialize velocity
57     a = 1
58     # Return F(t,U)
59     return (-a*D)@U
60
61 # Time integrator
62 def TimeIntegration(funcutor, scheme, time_limits, dt, U0, D, t_eval):
63     # Use Python's IVP solver; see documentation at:
64     # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html
65     if scheme == 'Exact':
66         sol = solve_ivp(funcutor, time_limits, U0, args=(D,), t_eval=t_eval, rtol=1.0e-6, atol=1.0e-6, first_step=dt)
67         return sol.y.transpose()
68     # Use in-house IVP solver
69     else:
70         # Number of timesteps
71         Nt = int(np.ceil((time_limits[1] - time_limits[0])/dt))+1
72         Uold = U0
73         # Vector of stored solutions -- add initial condition
74         sol = []; keval = 0
75         if t_eval[keval] == time_limits[0]:
76             sol.append(U0)
77             keval = 1
78         # Loop over all timesteps

```

```

79     for k in range(Nt):
80         told = time_limits[0] + dt * k
81         tnew = told + dt
82         # Compute RHS at told
83         rhs = functor(told, Uold, D)
84         # Update new solution with explicit scheme
85         match scheme:
86             case 'Euler':
87                 Unew = Uold + dt * rhs
88             case 'RK2':
89                 K1 = functor(told, Uold, D)
90                 K2 = functor(told+dt, Uold+dt*K1, D)
91                 Unew = Uold + dt*(K1+K2)/2
92             case 'RK3':
93                 K1 = functor(told, Uold, D)
94                 K2 = functor(told+dt, Uold+dt*K1, D)
95                 K3 = functor(told+0.5*dt, Uold+0.5*dt*0.5*(K1+K2), D)
96                 Unew = Uold + dt*(K1+K2+4*K3)/6
97             case 'RK4':
98                 K1 = functor(told, Uold, D)
99                 K2 = functor(told+0.5*dt, Uold+0.5*dt*K1, D)
100                K3 = functor(told+0.5*dt, Uold+0.5*dt*K2, D)
101                K4 = functor(told+dt, Uold+dt*K3, D)
102                Unew = Uold + dt*(K1+2*K2+2*K3+K4)/6
103            # Update stored solution vector
104            if (keval < len(t_eval) and t_eval[keval] >= told and t_eval[keval] <= tnew):
105                sol.append(Uold + (Unew - Uold) * (t_eval[keval] - told) / dt)
106                keval += 1
107            # Move to next timestep
108            Uold = Unew
109        return sol
110
111 # General integrator function
112 def Integrator(periodic,operator,temporalscheme,problem,L,T,Nx,Nt,U0,dt):
113     ##### Inputs of the function "Integrator" #####
114     #
115     # periodic : boolean flag to select periodicity (options: True or False)
116     # operator : string to select the spatial derivative operator
117     # problem : string to select the governing equations
118     # L : length of the physical domain, x runs from 0 to L
119     # T : length of the temporal domain, t runs from 0 to T
120     # Nx : number of points to use in x
121     # Nt : number of points to use in t (for reporting the solutions)
122     # U0 : initial condition
123     #
124     ##### Outputs of the function "Integrator" #####
125     #
126     # t : the discrete time levels (in a vector of size Nt)
127     # U : the solutions (in a matrix of size Nt x Nx)
128     #
129     #####
130
131     # Initialize spatial domain
132     x = np.linspace(0, L, Nx, endpoint=(not periodic))
133     dx = x[1] - x[0]
134
135     # Initialize temporal domain
136     t = np.linspace(0, T, Nt, endpoint=True)
137
138     # Construct spatial matrix operator
139     match (operator,periodic):
140         ### All periodic cases
141         case ('1st-order upwind',True): # Periodic 1st-order upwind
142             D = D_operator_periodic(Nx,1,0,[-1/dx,1/dx])
143         case ('2nd-order central',True): # Periodic 2nd-order central differences
144             D = D_operator_periodic(Nx,1,1,[-1/(2*dx),0,1/(2*dx)])
145         case ('3rd-order upwind',True): # Periodic 3rd-order upwind
146             D = D_operator_periodic(Nx,2,1,[1/(6*dx),-1/dx,1/(2*dx),1/(3*dx)])

```

```

147     case ('4th-order central',True):         # Periodic 4th-order central differences
148         D = D_operator_periodic(Nx,2,2,[1/(12*dx),-8/(12*dx),0,8/(12*dx),-1/(12*dx)])
149     case ('6th-order Pade',True):             # Periodic 6th-order Pade
150         DR = D_operator_periodic(Nx,2,2,[-1/(36*dx),-28/(36*dx),0,28/(36*dx),1/(36*dx)])
151         DL = D_operator_periodic(Nx,1,1,[1/3,1,1/3])
152         D = np.linalg.inv(DL)@DR
153
154     # Solve and return solutions!
155     U = TimeIntegration(LinearAdv1D, temporalscheme, [0,T], dt, U0, D, t)
156     return t, U, D
157
158 # %%
159 import cmath
160 from numpy import linalg as LA
161 plt.rcParams['text.usetex'] = True
162 plt.rcParams['figure.dpi'] = 300
163 plt.rcParams['savefig.dpi'] = 300
164 plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{amssymb}')
165
166 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
167 temporalscheme = 'RK3'
168 ##### Choose CFL = a*dt/dx
169 CFL = 1
170
171 # Initialize case parameters
172 L = 1; T = 30; Nx = 50; Nt = 11; a = 1
173 dx = L/Nx
174 dt = CFL * dx / a
175 # Initialize solution at t=0
176 x = np.linspace(0, L, Nx, endpoint=False); dx = x[1] - x[0]
177 sigma = 3/40; U0 = np.exp(-(x-0.5)**2/(2*sigma**2))
178 # Initialize list of schemes that will be tested:
179 listofspatialschemes = ['1st-order upwind', '2nd-order central', '3rd-order upwind', '4th-
    order central', '6th-order Pade']
180
181 # Plotting the eigenvalue spectra and numerical solutions for each scheme
182 A = []
183 for scheme in listofspatialschemes:
184     # Run solver and plot solution
185     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
186     A.append(-a*D)
187     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
188     ax.plot(x,U0,color=plt.cm.Spectral_r(0))
189     for j in range(1,len(U)):
190         ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
191     plt.title('Periodic advection of a Gaussian pulse using %s' % scheme)
192     plt.xlabel(r'$x$');plt.ylabel(r'$u(x,t)$');
193     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'),ax=ax
        , orientation='vertical', label=r'$t$')
194     plt.savefig('solution-%s.pdf' % scheme)
195     plt.show()
196
197 # Plot stability region
198 fig, ax = plt.subplots(1, 1, figsize=(5, 4))
199 Ngrid = 200; xs = np.arange(-3,0.5,3.5/Ngrid); ys = np.arange(-3,3,6/Ngrid); X, Y = np.
    meshgrid(xs, ys)
200 lambda_dt = X+Y*1j
201 match temporalscheme:
202     case 'Euler':
203         ax.contourf(X,Y,np.abs(1+lambda_dt), [0,1], colors='lightgray')
204         ax.contour(X,Y,np.abs(1+lambda_dt), [1], colors='k')
205     case 'RK2':
206         ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2), [0,1], colors='lightgray')
207         ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2), [1], colors='k')
208     case 'RK3':
209         ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3), [0,1], colors='
            lightgray')
210         ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3), [1], colors='k')

```

```

211     case 'RK4':
212         ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3+(1/24)*lambda_dt
213             **4), [0,1], colors='lightgray')
214         ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3+(1/24)*lambda_dt
215             **4), [1], colors='k')
216
217 # Plot eigenvalue spectrum
218 evalsdt, evecs = LA.eig(-a*D*dt)
219 relamdbadt = [ele.real for ele in evalsdt]
220 imlamdbadt = [ele.imag for ele in evalsdt]
221 cmap = mpl.colors.ListedColormap(['green', 'red'])
222 match temporalscheme:
223     case 'Euler':
224         ax.scatter(relamdbadt, imlamdbadt, c=np.abs(1+evalsdt), edgecolors='black', vmin=1e-6,
225             vmax=2, cmap=cmap)
226     case 'RK2':
227         ax.scatter(relamdbadt, imlamdbadt, c=np.abs(1+evalsdt+0.5*evalsdt**2), edgecolors='black
228             ', vmin=1e-6, vmax=2, cmap=cmap)
229     case 'RK3':
230         ax.scatter(relamdbadt, imlamdbadt, c=np.abs(1+evalsdt+0.5*evalsdt**2+(1/6)*evalsdt**3),
231             edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
232     case 'RK4':
233         ax.scatter(relamdbadt, imlamdbadt, c=np.abs(1+evalsdt+0.5*evalsdt**2+(1/6)*evalsdt
234             **3+(1/24)*evalsdt**4), edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
235     case _:
236         ax.scatter(relamdbadt, imlamdbadt, color='w', edgecolors='black')
237
238 plt.title('Eigenvalue spectrum for %s' % scheme)
239 plt.xlabel(r'$\mathbb{R}\mathrm{e}(\lambda \Delta t)$');plt.ylabel(r'$\mathbb{I}\mathrm{m}(\lambda \Delta t)$');
240 plt.savefig('eigenvalues-%s.pdf' % scheme)
241 plt.show()
242
243 # %% [markdown]
244 # Problem 4: Testing the stability of Euler and RK3
245
246 # %%
247 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
248 temporalscheme = 'Euler'
249 ##### Choose CFL = a*dt/dx
250 CFL = 0.95 * 1
251 dt = CFL * dx / a
252 listofspatialschemes = ['1st-order upwind']
253
254 # Plotting the eigenvalue spectra and numerical solutions for each scheme
255 for scheme in listofspatialschemes:
256     # Run solver and plot solution
257     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
258     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
259     ax.plot(x,U0,color=plt.cm.Spectral_r(0))
260     for j in range(1,len(U)):
261         ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
262     plt.title('Periodic advection using Euler and %s' % scheme)
263     plt.xlabel(r'$x$');plt.ylabel(r'$u(x,t)$');
264     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'),ax=ax
265         , orientation='vertical', label=r'$t$')
266     plt.savefig('solution-P4-Euler_095.pdf')
267     plt.show()
268
269 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
270 temporalscheme = 'Euler'
271 ##### Choose CFL = a*dt/dx
272 CFL = 1.05 * 1
273 dt = CFL * dx / a
274 listofspatialschemes = ['1st-order upwind']
275
276 # Plotting the eigenvalue spectra and numerical solutions for each scheme
277 for scheme in listofspatialschemes:

```

```

271 # Run solver and plot solution
272 t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
273 fig, ax = plt.subplots(1, 1, figsize=(7, 4))
274 ax.plot(x, U0, color=plt.cm.Spectral_r(0))
275 for j in range(1, len(U)):
276     ax.plot(x, U[j], color=plt.cm.Spectral_r(t[j]/T))
277 plt.title('Periodic advection using Euler and %s' % scheme)
278 plt.xlabel(r'$x$'); plt.ylabel(r'$u(x,t)$');
279 fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'), ax=ax,
280             , orientation='vertical', label=r'$t$')
281 plt.savefig('solution-P4_Euler_105.pdf')
282 plt.show()
283
284 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
285 temporalscheme = 'RK3'
286 ##### Choose CFL = a*dt/dx
287 CFL = 0.95 * np.sqrt(3)
288 dt = CFL * dx / a
289 listofspatialschemes = ['2nd-order central']
290
291 # Plotting the eigenvalue spectra and numerical solutions for each scheme
292 for scheme in listofspatialschemes:
293     # Run solver and plot solution
294     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
295     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
296     ax.plot(x, U0, color=plt.cm.Spectral_r(0))
297     for j in range(1, len(U)):
298         ax.plot(x, U[j], color=plt.cm.Spectral_r(t[j]/T))
299     plt.title('Periodic advection using RK3 and %s' % scheme)
300     plt.xlabel(r'$x$'); plt.ylabel(r'$u(x,t)$');
301     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'), ax=ax,
302                 , orientation='vertical', label=r'$t$')
303     plt.savefig('solution-P4_RK3_095.pdf')
304     plt.show()
305
306 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
307 temporalscheme = 'RK3'
308 ##### Choose CFL = a*dt/dx
309 CFL = 1.05 * np.sqrt(3)
310 dt = CFL * dx / a
311 listofspatialschemes = ['2nd-order central']
312
313 # Plotting the eigenvalue spectra and numerical solutions for each scheme
314 for scheme in listofspatialschemes:
315     # Run solver and plot solution
316     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
317     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
318     ax.plot(x, U0, color=plt.cm.Spectral_r(0))
319     for j in range(1, len(U)):
320         ax.plot(x, U[j], color=plt.cm.Spectral_r(t[j]/T))
321     plt.title('Periodic advection using RK3 and %s' % scheme)
322     plt.xlabel(r'$x$'); plt.ylabel(r'$u(x,t)$');
323     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'), ax=ax,
324                 , orientation='vertical', label=r'$t$')
325     plt.savefig('solution-P4_RK3_105.pdf')
326     plt.show()
327
328 # %% [markdown]
329 # Problem 5: Testing the stability of RK4
330
331 # %%
332 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
333 temporalscheme = 'RK4'
334 ##### Choose CFL = a*dt/dx
335 CFL = 0.95 * 2 * np.sqrt(2)
336 dt = CFL * dx / a
337 listofspatialschemes = ['2nd-order central']
338

```

```

336 # Plotting the eigenvalue spectra and numerical solutions for each scheme
337 for scheme in listofspatialchemes:
338     # Run solver and plot solution
339     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
340     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
341     ax.plot(x, U0, color=plt.cm.Spectral_r(0))
342     for j in range(1, len(U)):
343         ax.plot(x, U[j], color=plt.cm.Spectral_r(t[j]/T))
344     plt.title('Periodic advection using RK4 and %s' % scheme)
345     plt.xlabel(r'$x$'); plt.ylabel(r'$u(x,t)$');
346     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'), ax=ax,
347                 , orientation='vertical', label=r'$t$')
347     plt.savefig('solution-P5_095.pdf')
348     plt.show()
349
350 ##### Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
351 temporalscheme = 'RK4'
352 ##### Choose CFL = a*dt/dx
353 CFL = 1.05 * 2 * np.sqrt(2)
354 dt = CFL * dx / a
355 listofspatialchemes = ['2nd-order central']
356
357 # Plotting the eigenvalue spectra and numerical solutions for each scheme
358 for scheme in listofspatialchemes:
359     # Run solver and plot solution
360     t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt)
361     fig, ax = plt.subplots(1, 1, figsize=(7, 4))
362     ax.plot(x, U0, color=plt.cm.Spectral_r(0))
363     for j in range(1, len(U)):
364         ax.plot(x, U[j], color=plt.cm.Spectral_r(t[j]/T))
365     plt.title('Periodic advection using RK4 and %s' % scheme)
366     plt.xlabel(r'$x$'); plt.ylabel(r'$u(x,t)$');
367     fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, T), cmap='Spectral_r'), ax=ax,
368                 , orientation='vertical', label=r'$t$')
368     plt.savefig('solution-P5_105.pdf')
369     plt.show()

```

### Matlab (1-based indexing)

```

1 clear all; close all; clc
2
3 xs = -3.2:0.1:3.2;
4 ys = xs;
5 [X, Y] = meshgrid(xs, ys);
6 lambda_dt = X + Y*i;
7
8 figure(1); clf
9 contourf(X, Y, abs(1 + lambda_dt), [-1,0,1], 'EdgeColor', 'none', 'LineWidth', 2); hold on;
10 contour(X, Y, abs(1 + lambda_dt), [1,1.001], 'LineColor', 'r', 'LineWidth', 3);
11 caxis([-1,1])
12 title('Stability Diagram for Euler');
13 xlabel('Re(\lambda \Delta t)');
14 ylabel('Im(\lambda \Delta t)');
15 xlim([-4, 2]);
16 ylim([-3.1, 3.1]);
17 colormap pink;
18 %saveas(gcf, 'stabDiagsEuler.pdf');
19
20 %%
21 figure(2), clf;
22 contourf(X, Y, abs(1 + lambda_dt + lambda_dt.^2/2), [-1,0,1], 'EdgeColor', 'none', 'LineWidth',
23         , 2); hold on
24 contour(X, Y, abs(1 + lambda_dt + lambda_dt.^2/2), [1,1.001], 'LineColor', 'r', 'LineWidth',
25         , 2);
26 caxis([-1,1])
27 title('Stability Diagram for Runge-Kutta 2');
28 xlabel('Re(\lambda \Delta t)');
29 ylabel('Im(\lambda \Delta t)');
30 xlim([-4, 2]);

```

```

29 ylim([-3.1, 3.1]);
30 colormap pink;
31 %saveas(gcf, 'stabDiagsRK2.pdf');
32
33 %%
34 figure(3), clf;
35 contourf(X, Y, abs(1 + lambda_dt + 1/2*lambda_dt.^2 + 1/6*lambda_dt.^3), [-1,0,1], 'EdgeColor'
    , 'none', 'LineWidth', 2); hold on
36 contour(X, Y, abs(1 + lambda_dt + 1/2*lambda_dt.^2 + 1/6*lambda_dt.^3), [1,1.001], 'LineColor'
    , 'r', 'LineWidth', 2);
37 caxis([-1,1])
38 title('Stability Diagram for Runge-Kutta 3');
39 xlabel('Re(\lambda \Delta t)');
40 ylabel('Im(\lambda \Delta t)');
41 xlim([-4, 1]);
42 ylim([-3.1, 3.1]);
43 colormap pink;
44 %saveas(gcf, 'stabDiagsRK3.pdf');
45
46 %%
47 figure(4), clf;
48 contourf(X, Y, abs(1 + lambda_dt + 1/2*lambda_dt.^2 + 1/6*lambda_dt.^3 + 1/24*lambda_dt.^4),
    [-1,0,1], 'EdgeColor', 'none', 'LineWidth', 2); hold on
49 contour(X, Y, abs(1 + lambda_dt + 1/2*lambda_dt.^2 + 1/6*lambda_dt.^3 + 1/24*lambda_dt.^4),
    [1,1.001], 'LineColor', 'r', 'LineWidth', 2);
50 caxis([-1,1])
51 title('Stability Diagram for Runge-Kutta 4');
52 xlabel('Re(\lambda \Delta t)');
53 ylabel('Im(\lambda \Delta t)');
54 xlim([-4, 1]);
55 ylim([-3.1, 3.1]);
56 colormap pink;
57 %saveas(gcf, 'stabDiagsRK4.pdf');

1 clear all; close all; clc
2
3
4 % Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
5 temporalscheme = 'RK3';
6 % Choose CFL = a*dt/dx
7 CFL = 1; % 1.05 * sqrt(3)
8
9 % Initialize case parameters
10 L = 1; T = 30.6; Nx = 50; Nt = 11;
11 a = 1;
12 % Initialize solution at t=0
13 x = linspace(0, L, Nx);
14 dx = x(2) - x(1);
15 dt = CFL * dx / a;
16 sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
17 % Initialize list of schemes that will be tested:
18 listofspatialschemes = {'1st-order upwind', '2nd-order central', '3rd-order upwind', '4th-
    order central', '6th-order Pade'};
19
20 % Plotting the eigenvalue spectra and numerical solutions for each scheme
21 A = [];
22 for idx = 1:length(listofspatialschemes)
23     scheme = listofspatialschemes{idx};
24     % Run solver and plot solution
25     [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
26     %A = [A; -a * D];
27     fig = figure('Position', [100, 100, 500, 400]); hold on
28     colors = [linspace(0,1,length(t)), zeros(length(t),2)];
29     plot(x, U0, 'color', colors(1,:), 'LineWidth', 2); hold on
30     for j = 2:length(t)
31         plot(x, U(:,j), 'color', colors(j,:), 'LineWidth', 2);
32     end
33     hold off

```



```

34     title(sprintf('Periodic advection of a Gaussian pulse using %s', scheme));
35     xlabel('x');
36     ylabel('u(x,t)');
37     %saveas(fig, sprintf('solution-%s.pdf', scheme));
38     %close(fig);
39
40     % Plot stability region
41     fig = figure('Position', [600, 100, 500, 400]);
42     Ngrid = 200; xs = (-3:3.5/Ngrid:0.5); ys = (-3:3.5/Ngrid:3);
43     [X, Y] = meshgrid(xs, ys);
44     lambda_dt = X + 1i * Y;
45     eigvals = eig(-a*D);
46     eigvals = diag(eigvals);
47     switch temporalscheme
48     case 'Euler'
49         %contourf(X, Y, abs(1 + lambda_dt), [0, 1], 'LineColor', 'none');
50         %hold on
51         contour(X, Y, abs(1 + lambda_dt), [0.999,1], 'LineColor', 'k'); hold on
52         plot(real(eigvals)*dt, imag(eigvals)*dt, 'r.', 'MarkerSize', 15)
53     case 'RK2'
54         %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2), [0, 1], 'LineColor', '
55             none');
56         %hold on
57         contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2), [0.999,1], 'LineColor', 'k'
58             );hold on
59         plot(real(eigvals)*dt, imag(eigvals)*dt, 'r.', 'MarkerSize', 15)
60     case 'RK3'
61         %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3),
62             [0, 1], 'LineColor', 'none');
63         %hold on
64         contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3),
65             [0.999,1], 'LineColor', 'k');hold on
66         plot(real(eigvals)*dt, imag(eigvals)*dt, 'r.', 'MarkerSize', 15)
67     case 'RK4'
68         %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3 +
69             (1/24) * lambda_dt.^4), [0, 1], 'LineColor', 'none');
70         %hold on
71         contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3 +
72             (1/24) * lambda_dt.^4), [0.999,1], 'LineColor', 'k');hold on
73         plot(real(eigvals)*dt, imag(eigvals)*dt, 'r.', 'MarkerSize', 15)
74     end
75     title(sprintf('Eigenvalue spectrum for %s', scheme));
76     xlabel('Re(\lambda \Delta t)');
77     ylabel('Im(\lambda \Delta t)');
78     %saveas(fig, sprintf('eigenvalues-%s.pdf', scheme));
79     %close(fig);
80 end
81
82 return
83
84 %% Problem 4: First Set of Schemes
85
86 % Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
87 temporalscheme = 'Euler';
88 % Choose CFL = a*dt/dx
89 CFL = 0.95 * 1; % sqrt(3)
90
91 % Initialize case parameters
92 L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
93 dx = L / Nx;
94 dt = CFL * dx / a;
95 % Initialize solution at t=0
96 x = linspace(0, L, Nx); dx = x(2) - x(1);
97 sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
98 % Initialize list of schemes that will be tested:
99 listofspatialschemes = {'1st-order upwind'};
100
101 % Plotting the eigenvalue spectra and numerical solutions for each scheme

```

```

96  A = [];
97  for idx = 1:length(listofspatialschemes)
98      scheme = listofspatialschemes{idx};
99      % Run solver and plot solution
100     [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
101     A = [A; -a * D];
102     fig = figure('Position', [100, 100, 700, 400]);
103     colors = [linspace(0,1,length(t))',zeros(length(t),2)];
104     plot(x, U0, 'color', colors(1,:));
105     hold on
106     for j = 2:length(t)
107         plot(x, U(:,j), 'color', colors(j,:));
108     end
109     hold on
110     title(sprintf('Periodic advection using Euler and %s', scheme));
111     xlabel('x');
112     ylabel('u(x,t)');
113     colorbar;
114     %saveas(fig, 'solution-P4_Euler_095.pdf');
115     %close(fig);
116 end
117
118 % Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
119 temporalscheme = 'Euler';
120 % Choose CFL = a*dt/dx
121 CFL = 1.05 * 1; % sqrt(3)
122
123 % Initialize case parameters
124 L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
125 dx = L / Nx;
126 dt = CFL * dx / a;
127 % Initialize solution at t=0
128 x = linspace(0, L, Nx); dx = x(2) - x(1);
129 sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0 = U0';
130 % Initialize list of schemes that will be tested:
131 listofspatialschemes = {'1st-order upwind'};
132
133 % Plotting the eigenvalue spectra and numerical solutions for each scheme
134 A = [];
135 for idx = 1:length(listofspatialschemes)
136     scheme = listofspatialschemes{idx};
137     % Run solver and plot solution
138     [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
139     A = [A; -a * D];
140     fig = figure('Position', [100, 100, 700, 400]);
141     colors = [linspace(0,1,length(t))',zeros(length(t),2)];
142     plot(x, U0, 'color', colors(1,:));
143     hold on
144     for j = 2:length(t)
145         plot(x, U(:,j), 'color', colors(j,:));
146     end
147     hold('off');
148     title(sprintf('Periodic advection using Euler and %s', scheme));
149     xlabel('x');
150     ylabel('u(x,t)');
151     colorbar
152     %saveas(fig, 'solution-P4_Euler_105.pdf');
153     %close(fig);
154 end
155
156 %% Problem 4: Second Set of Schemes
157
158 % Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
159 temporalscheme = 'RK3';
160 % Choose CFL = a*dt/dx
161 CFL = 0.95 * sqrt(3);
162
163 % Initialize case parameters

```

```

164 L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
165 dx = L / Nx;
166 dt = CFL * dx / a;
167 % Initialize solution at t=0
168 x = linspace(0, L, Nx); dx = x(2) - x(1);
169 sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0 = U0';
170 % Initialize list of schemes that will be tested:
171 listofspatialschemes = {'2nd-order central'};
172
173 % Plotting the eigenvalue spectra and numerical solutions for each scheme
174 A = [];
175 for idx = 1:length(listofspatialschemes)
176     scheme = listofspatialschemes{idx};
177     % Run solver and plot solution
178     [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
179     A = [A; -a * D];
180     fig = figure('Position', [100, 100, 700, 400]);
181     colors = [linspace(0,1,length(t))', zeros(length(t),2)];
182     plot(x, U0, 'color', colors(1,:));
183     hold on
184     for j = 2:length(t)
185         plot(x, U(:,j), 'color', colors(j,:));
186     end
187     hold off
188     title(sprintf('Periodic advection using RK3 and %s', scheme));
189     xlabel('x');
190     ylabel('u(x,t)');
191     colorbar
192     %saveas(fig, 'solution-P4_RK3_095.pdf');
193     %close(fig);
194 end
195
196 % Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
197 temporalscheme = 'RK3';
198 % Choose CFL = a*dt/dx
199 CFL = 1.05 * sqrt(3);
200
201 % Initialize case parameters
202 L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
203 dx = L / Nx;
204 dt = CFL * dx / a;
205 % Initialize solution at t=0
206 x = linspace(0, L, Nx); dx = x(2) - x(1);
207 sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
208 % Initialize list of schemes that will be tested:
209 listofspatialschemes = {'2nd-order central'};
210
211 % Plotting the eigenvalue spectra and numerical solutions for each scheme
212 A = [];
213 for idx = 1:length(listofspatialschemes)
214     scheme = listofspatialschemes{idx};
215     % Run solver and plot solution
216     [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
217     A = [A; -a * D];
218     fig = figure('Position', [100, 100, 700, 400]);
219     colors = [linspace(0,1,length(t))', zeros(length(t),2)];
220     plot(x, U0, 'color', colors(1,:));
221     hold on
222     for j = 2:length(t)
223         plot(x, U(:,j), 'color', colors(j,:));
224     end
225     hold off
226     title(sprintf('Periodic advection using RK3 and %s', scheme));
227     xlabel('x');
228     ylabel('u(x,t)');
229     colorbar
230     %saveas(fig, 'solution-P4_RK3_105.pdf');
231     %close(fig);

```

```
232 end
233
234
235 %% Auxiliary Functions
236 function [ D ] = D_operator_periodic( n,L,R,a )
237     %computes a finite difference operator
238     % n = number of grid points
239     % dx = step size
240     % [R,L] = right/left bound of stencil
241     % a = stencils
242
243     % initialize
244     D = zeros(n);
245     % fill interior domain
246     for i=1+L:n+1-R-1
247         D(i,i-L:i+R) = a;
248     end
249     % fill left boundary
250     for i=1:L
251         D(i,1:i+R) = a(L-i+2:end);
252         D(i,end-L+i:end) = a(1:L-i+1);
253     end
254
255     % fill right boundary
256     for i=n+1-R:n
257         D(i,end-L+0+(i-n):end) = a(1:L+(n+2-i)-1);
258         D(i,1:(i-n+R-0)) = a(L+(n+2-i):end);
259     end
260 end
261
262 % Functor for the 1D advection equation
263 function dUdt = LinearAdv1D(t, U, D)
264     % Initialize velocity
265     a = 1;
266     % Compute dU/dt
267     dUdt = -a * (D * U);
268 end
269
270 % Time integrator
271 function sol = TimeIntegration(scheme, time_limits, dt, U0, D, t_eval)
272     % Exact solution using MATLAB's ODE solver
273     if strcmp(scheme, 'Exact')
274         options = odeset('RelTol', 1.0e-6, 'AbsTol', 1.0e-6);
275         [~, sol] = ode45(@(t, U) LinearAdv1D(t, U, D), t_eval, U0, options);
276         sol = sol';
277         return;
278     else
279         % Number of timesteps
280         Nt = ceil((time_limits(2) - time_limits(1)) / dt) + 1;
281         Uold = U0;
282         % Vector of stored solutions -- add initial condition
283         sol = [];
284         keval = 1;
285         if t_eval(keval) == time_limits(1)
286             sol(:, keval) = U0;
287             keval = keval + 1;
288         end
289         % Loop over all timesteps
290         for k = 1:Nt
291             told = time_limits(1) + dt * (k - 1);
292             tnew = told + dt;
293             % Compute RHS at told
294             rhs = LinearAdv1D(told, Uold, D);
295             % Update new solution with explicit scheme
296             switch scheme
297                 case 'Euler'
298                     Unew = Uold + dt * rhs;
299                 case 'RK2'
```

```

300         K1 = dt*LinearAdv1D(told,Uold,D);
301         K2 = dt*LinearAdv1D(told+dt,Uold+K1,D);
302         Unew = Uold+1/2*(K1+K2);
303     case 'RK3'
304         K1 = dt*LinearAdv1D(told,Uold,D);
305         K2 = dt*LinearAdv1D(told+dt/2,Uold+K1/2,D);
306         K3 = dt*LinearAdv1D(told+dt,Uold+2*K2-K1,D);
307         Unew = Uold+1/6*(K1+4*K2+K3);
308     case 'RK4'
309         K1 = LinearAdv1D(told, Uold, D);
310         K2 = LinearAdv1D(told + 0.5 * dt, Uold + 0.5 * dt * K1, D);
311         K3 = LinearAdv1D(told + 0.5 * dt, Uold + 0.5 * dt * K2, D);
312         K4 = LinearAdv1D(told + dt, Uold + dt * K3, D);
313         Unew = Uold + dt * (K1 + 2 * K2 + 2 * K3 + K4) / 6;
314     end
315     %Update stored solution vector
316     if keval <= length(t_eval) && t_eval(keval) >= told && t_eval(keval) <= tnew
317         sol(:, keval) = Uold + (Unew - Uold) * (t_eval(keval) - told) / dt;
318         keval = keval + 1;
319     end
320     % Move to next timestep
321     Uold = Unew;
322 end
323 end
324 end
325
326 function [t, U, D] = Integrator(periodic, operator, temporalscheme, problem, L, T, Nx, Nt, U0,
    dt)
327     % Initialize spatial domain
328     x = linspace(0, L, Nx);
329     dx = x(2) - x(1);
330
331     % Initialize temporal domain
332     t = linspace(0, T, Nt);
333
334     % Construct spatial matrix operator
335     switch operator
336     % All periodic cases
337     case '1st-order upwind'
338         D = D_operator_periodic(Nx, 1, 0, [-1/dx, 1/dx]);
339     case '2nd-order central'
340         D = D_operator_periodic(Nx, 1, 1, [-1/(2*dx), 0, 1/(2*dx)]);
341     case '3rd-order upwind'
342         D = D_operator_periodic(Nx, 2, 1, [1/(6*dx), -1/dx, 1/(2*dx), 1/(3*dx)]);
343     case '4th-order central'
344         D = D_operator_periodic(Nx, 2, 2, [1/(12*dx), -8/(12*dx), 0, 8/(12*dx), -1/(12*dx)
345         ]);
346     case '6th-order Pade'
347         DR = D_operator_periodic(Nx, 2, 2, [-1/(36*dx), -28/(36*dx), 0, 28/(36*dx), 1/(36*
348         dx)]);
349         DL = D_operator_periodic(Nx, 1, 1, [1/3, 1, 1/3]);
350         D = inv(DL) * DR;
351     otherwise
352         error('The %s operator %s is not yet implemented!', ...
353             periodic, 'periodic', 'non-periodic'), operator);
354 end
355
356 U = TimeIntegration(temporalscheme, [0, T], dt, U0, D, t);
357 end

```