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 4 credit-hours students**)

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

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 $\mathbf{F}\left(t, \mathbf{U}(t)\right) = \mathbf{A}\mathbf{U}(t)$. 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.

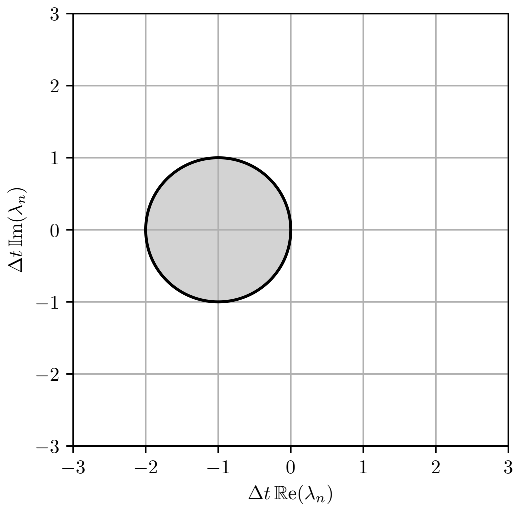
For instance: the stability condition for the first-order forward Euler scheme reads as

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

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

```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:



Solution:

Euler

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

Runge-Kutta 2

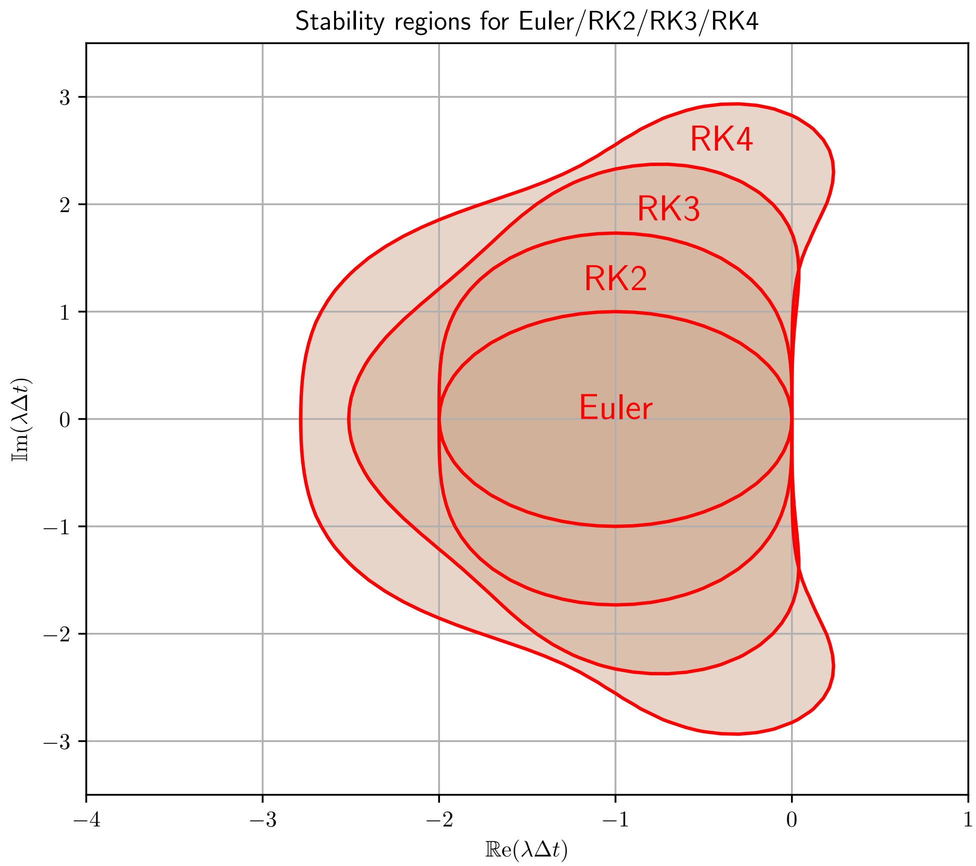
$$
\begin{aligned}
k_1 &= \lambda_n w_n^m \\
k_2 &= \lambda_n \left(w_n^m + k_1 \Delta t\right) = w_n^m \left(\lambda_n + \lambda_n^2 \Delta t\right) \\
w_n^{m+1} &= w_n^m + \frac{\Delta t}{2} \left(\lambda_n w_n^m + w_n^m \left(\lambda_n + \lambda_n^2 \Delta t\right)\right) \\
&= w_n^m \left(1 + \lambda_n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

Runge-Kutta 3

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

Runge-Kutta 4

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



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

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

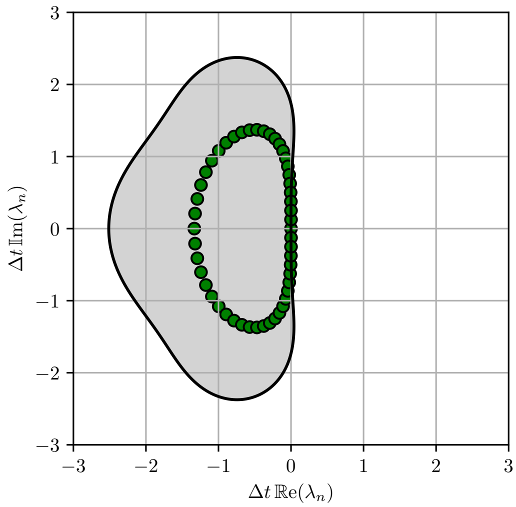
For the explicit time integration, consider the RK3 scheme with a CFL number

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

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.

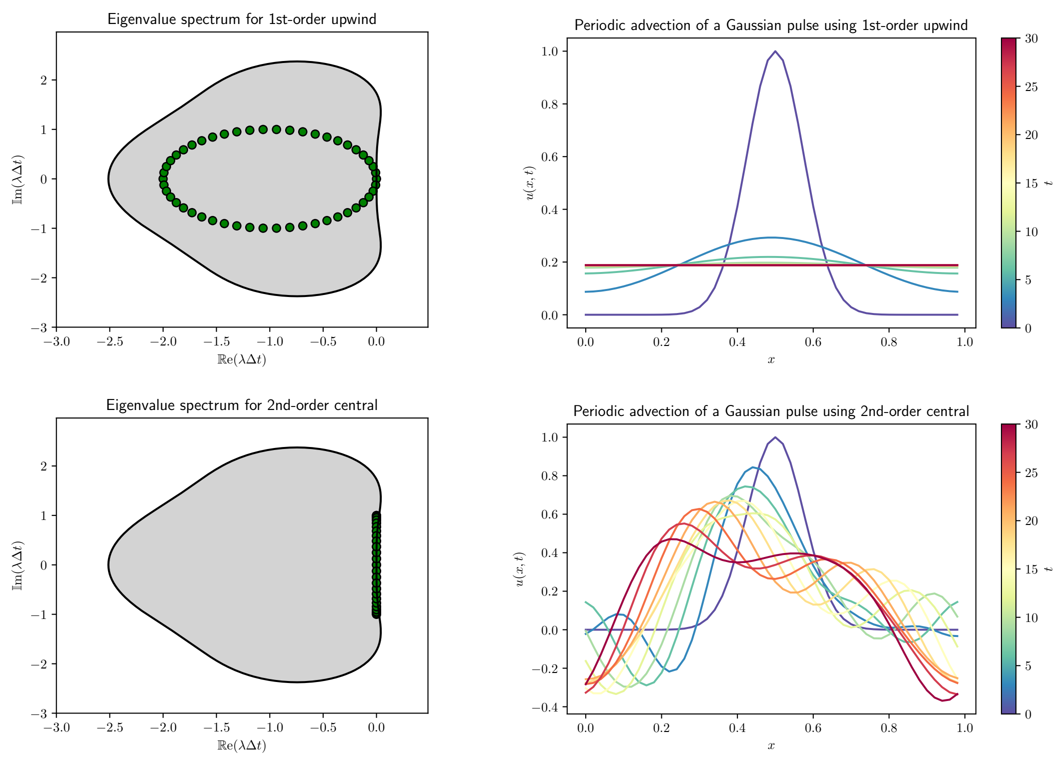
For instance: The eigenvalue spectrum of $\mathbf{A}$ for the third-order upwind scheme, plotted on top of the RK3 stability region, looks as follows:

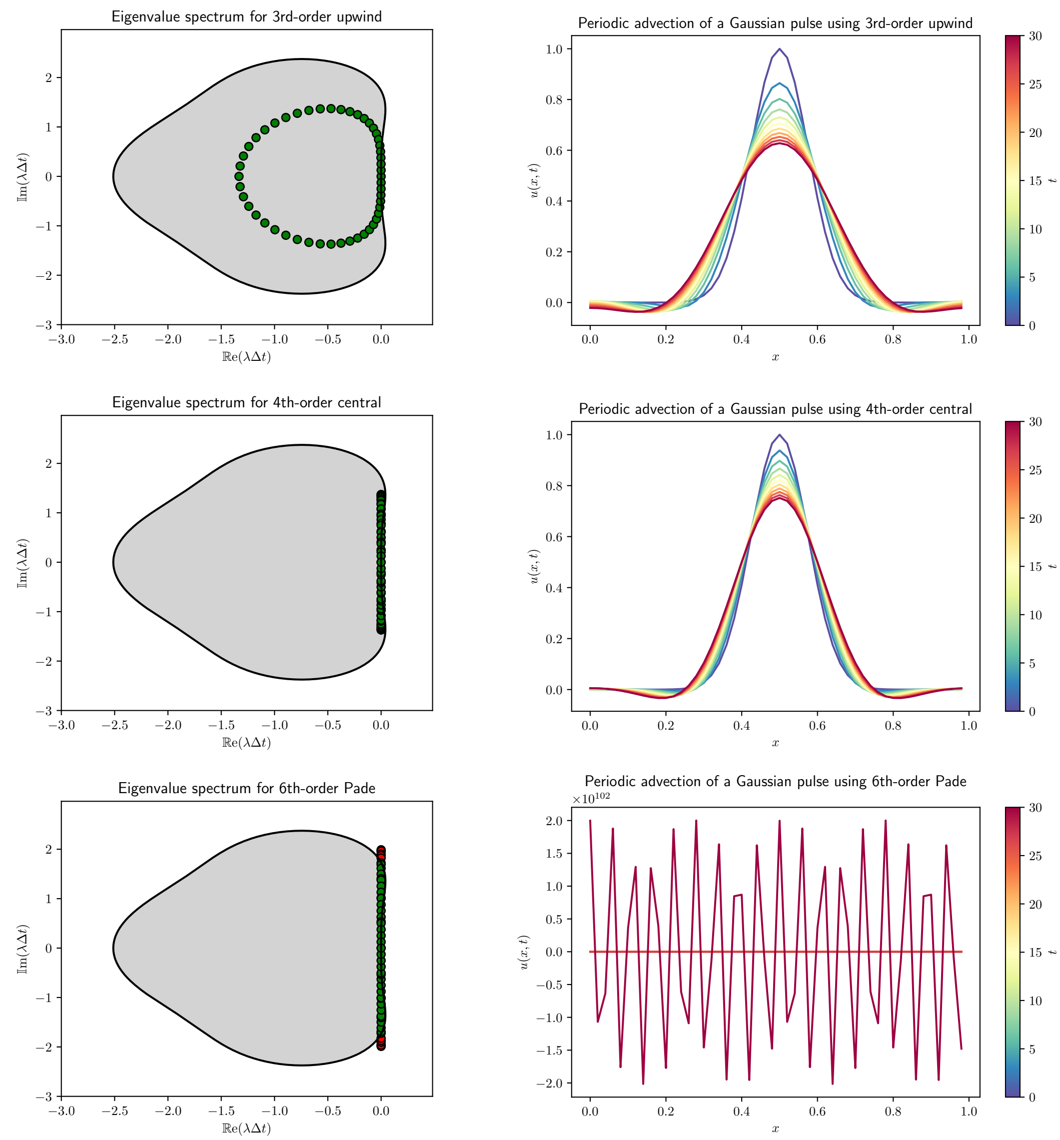


Q3.2 $\rightarrow$ Compare your numerical solution to the exact expected solution at the times $t_m, m \in \{3, 6, \ldots, 30\}$.

Q3.3 $\rightarrow$ Are your numerical solutions stable? If not, qualitatively explain why.

Solution:





As can be seen in above figures, all spatial discretization schemes whose eigenvalues are exclusively within the stability region of the time integration scheme (here Runge-Kutta 3) are unconditionally stable, as can be seen in the unsteady solution. Only the Padé scheme has eigenvalues that lie outside the stability region and consequently grow out of bounds, which can be seen in the unsteady solution.

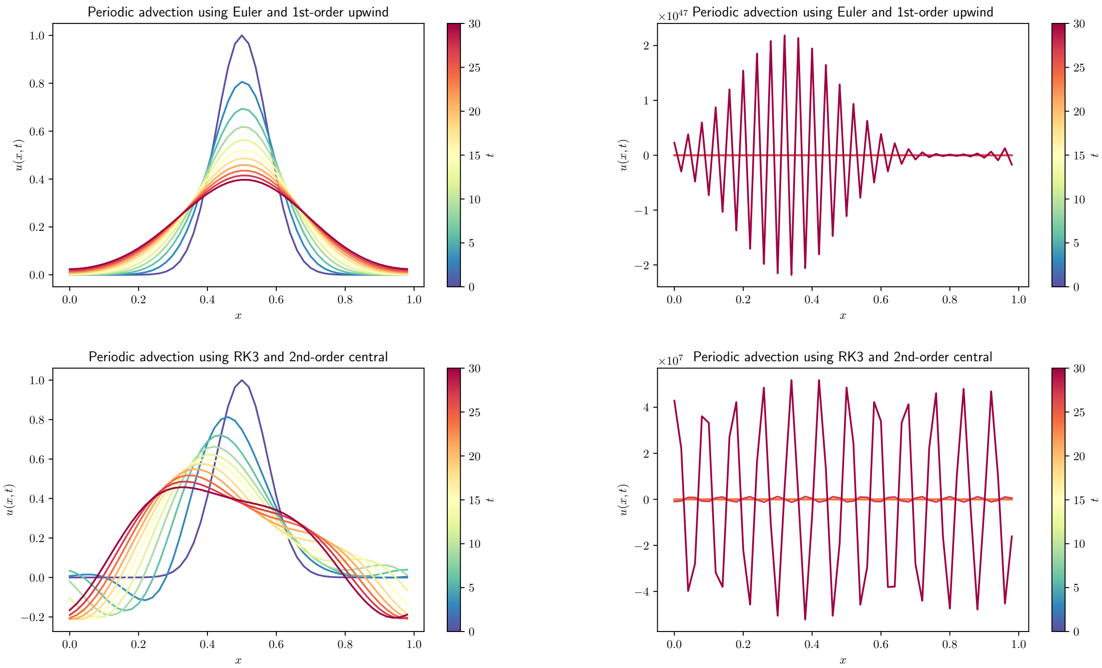
Problem 4 · In class, we have derived several stability conditions in the form $\mathrm{CFL} \leq \mathrm{CFL}_{\max}$ for combinations of discrete temporal and spatial differential operators applied to the numerical solution of (2).

For instance:

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

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

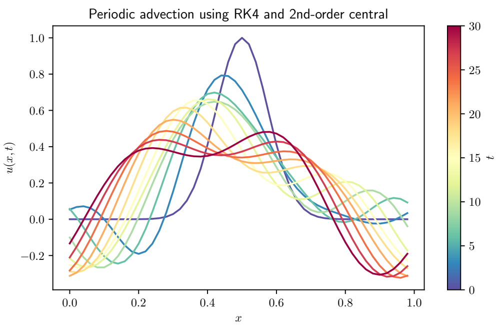
$$\Big|\mathbb{I}m\left(\lambda_n\right)\Big| \leq \frac{2\sqrt{2}}{\Delta t}$$

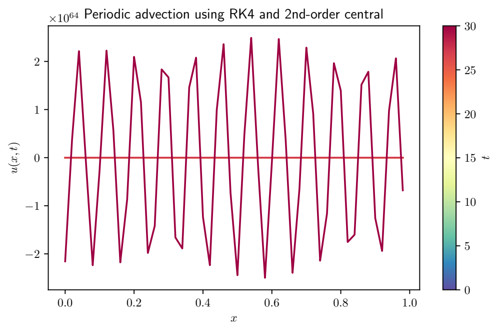
For the specific case of the second order central difference scheme, we know the analytical eigenvalues to be

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

therefore the stability condition fore the RK4 scheme becomes

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





As can bee seen in above pictures (CFL $= 0.95\cdot\text{CFL}_{\text{crit}}$ (left), 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)

```
# %% [markdown]
# Problem 2: Stability regions

# %%
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib as mpl
import numpy as np
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}')

xs = np.arange(-3.2,3.2,0.1)
```

```
ys = xs
X, Y = np.meshgrid(xs, ys)
lambda_dt = X+Y*1j
fig, ax = plt.subplots(1, 1, figsize=(7, 6))

plt.title('Stability regions for Euler/RK2/RK3/RK4')
plt.xlabel(r'$\mathbb{R}\mathrm{e}(\lambda \Delta t)$');
plt.ylabel(r'$\mathbb{I}\mathrm{m}(\lambda \Delta t)$');
plt.contourf(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3+1/24*lambda_dt**4),
    [0,1], cmap='pink', alpha=.5)
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],
    colors='r')
plt.contourf(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3), [0,1], cmap='pink',
    alpha=.5)
plt.contour(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2+1/6*lambda_dt**3), [1], colors='r')
plt.contourf(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2), [0,1], cmap='pink', alpha=.5)
plt.contour(X,Y,np.abs(1+lambda_dt+1/2*lambda_dt**2), [1], colors='r')
plt.contourf(X,Y,np.abs(1+lambda_dt), [0,1], cmap='pink', alpha=.5)
plt.contour(X,Y,np.abs(1+lambda_dt), [1], colors='r')
plt.grid()

plt.text(-1,0, 'Euler', dict(size=16), horizontalalignment='center', color='r')
plt.text(-1,1.2, 'RK2', dict(size=16), horizontalalignment='center', color='r')
plt.text(-0.7,1.85, 'RK3', dict(size=16), horizontalalignment='center', color='r')
plt.text(-0.4,2.5, 'RK4', dict(size=16), horizontalalignment='center', color='r')
ax.set_xlim([-4, 1])
ax.set_ylim([-3.5, 3.5])
plt.savefig('stabregions.pdf')
plt.show()

# %% [markdown]
# Problem 3: Explicit time integration of the 1D linear advection

# %%
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

# Time integrator
def TimeIntegration(functor, scheme, time_limits, dt, U0, D, t_eval):
  # Use Python's IVP solver; see documentation at:
  # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html
  if scheme == 'Exact':
    sol = solve_ivp(functor, time_limits, U0, args=(D,), t_eval=t_eval, rtol=1.0e-6, atol=1.0e
        -6, first_step=dt)
    return sol.y.transpose()
  # Use in-house IVP solver
  else:
    # Number of timesteps
    Nt = int(np.ceil((time_limits[1] - time_limits[0]))/dt)+1
    Uold = U0
    # Vector of stored solutions -- add initial condition
    sol = []; keval = 0
    if t_eval[keval] == time_limits[0]:
      sol.append(U0)
      keval = 1
    # Loop over all timesteps
```

```
    for k in range(Nt):
      told = time_limits[0] + dt * k
      tnew = told + dt
      # Compute RHS at told
      rhs = functor(told, Uold, D)
      # Update new solution with explicit scheme
      match scheme:
        case 'Euler':
          Unew = Uold + dt * rhs
        case 'RK2':
          K1 = functor(told, Uold, D)
          K2 = functor(told+dt, Uold+dt*K1, D)
          Unew = Uold + dt*(K1+K2)/2
        case 'RK3':
          K1 = functor(told, Uold, D)
          K2 = functor(told+dt, Uold+dt*K1, D)
          K3 = functor(told+0.5*dt, Uold+0.5*dt*0.5*(K1+K2), D)
          Unew = Uold + dt*(K1+K2+4*K3)/6
        case 'RK4':
          K1 = functor(told, Uold, D)
          K2 = functor(told+0.5*dt, Uold+0.5*dt*K1, D)
          K3 = functor(told+0.5*dt, Uold+0.5*dt*K2, D)
          K4 = functor(told+dt, Uold+dt*K3, D)
          Unew = Uold + dt*(K1+2*K2+2*K3+K4)/6
      # Update stored solution vector
      if (keval < len(t_eval) and t_eval[keval] >= told and t_eval[keval] <= tnew):
        sol.append(Uold + (Unew - Uold) * (t_eval[keval] - told) / dt)
        keval += 1
      # Move to next timestep
      Uold = Unew
    return sol

# General integrator function
def Integrator(periodic,operator,temporalscheme,problem,L,T,Nx,Nt,U0,dt):
  ######################## 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):
    #### All periodic cases
    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 Pade',True):         # Periodic 6th-order Pade
        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

  # Solve and return solutions!
  U = TimeIntegration(LinearAdv1D, temporalscheme, [0,T], dt, U0, D, t)
  return t, U, D

# %%
import cmath
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}')

############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3'
############## Choose CFL = a*dt/dx
CFL = 1

# Initialize case parameters
L = 1; T = 30; Nx = 50; Nt = 11; a = 1
dx = L/Nx
dt = CFL * dx / a
# 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:
listofspatialschemes = ['1st-order upwind', '2nd-order central', '3rd-order upwind', '4th-
    order central', '6th-order Pade']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
A = []
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  A.append(-a*D)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    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()

  # Plot stability region
  fig, ax = plt.subplots(1, 1, figsize=(5, 4))
  Ngrid = 200; xs = np.arange(-3,0.5,3.5/Ngrid); ys = np.arange(-3,3,6/Ngrid); X, Y = np.
      meshgrid(xs, ys)
  lambda_dt = X+Y*1j
  match temporalscheme:
    case 'Euler':
      ax.contourf(X,Y,np.abs(1+lambda_dt), [0,1], colors='lightgray')
      ax.contour(X,Y,np.abs(1+lambda_dt), [1], colors='k')
    case 'RK2':
      ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2), [0,1], colors='lightgray')
      ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2), [1], colors='k')
    case 'RK3':
      ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3), [0,1], colors='
          lightgray')
      ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3), [1], colors='k')
```

```
    case 'RK4':
      ax.contourf(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3+(1/24)*lambda_dt**4), [0,1], colors='lightgray')
      ax.contour(X,Y,np.abs(1+lambda_dt+0.5*lambda_dt**2+(1/6)*lambda_dt**3+(1/24)*lambda_dt**4), [1], colors='k')

  # Plot eigenvalue spectrum
  evalsdt, evecs = LA.eig(-a*D*dt)
  relambdadt = [ele.real for ele in evalsdt]
  imlambdadt = [ele.imag for ele in evalsdt]
  cmap = mpl.colors.ListedColormap(['green', 'red'])
  match temporalscheme:
    case 'Euler':
      ax.scatter(relambdadt, imlambdadt, c=np.abs(1+evalsdt), edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
    case 'RK2':
      ax.scatter(relambdadt, imlambdadt, c=np.abs(1+evalsdt+0.5*evalsdt**2), edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
    case 'RK3':
      ax.scatter(relambdadt, imlambdadt, c=np.abs(1+evalsdt+0.5*evalsdt**2+(1/6)*evalsdt**3), edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
    case 'RK4':
      ax.scatter(relambdadt, imlambdadt, c=np.abs(1+evalsdt+0.5*evalsdt**2+(1/6)*evalsdt**3+(1/24)*evalsdt**4), edgecolors='black', vmin=1e-6, vmax=2, cmap=cmap)
    case _:
      ax.scatter(relambdadt, imlambdadt, color='w', edgecolors='black')

  plt.title('Eigenvalue spectrum for %s' % scheme)
  plt.xlabel(r'$\mathbb{R}\mathrm{e}(\lambda \Delta t)$');plt.ylabel(r'$\mathbb{I}\mathrm{m}(\lambda \Delta t)$');
  plt.savefig('eigenvalues-%s.pdf' % scheme)
  plt.show()

# %% [markdown]
# Problem 4: Testing the stability of Euler and RK3

# %%
############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'Euler'
############## Choose CFL = a*dt/dx
CFL = 0.95 * 1
dt = CFL * dx / a
listofspatialschemes = ['1st-order upwind']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using Euler and %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-P4_Euler_095.pdf')
  plt.show()

############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'Euler'
############## Choose CFL = a*dt/dx
CFL = 1.05 * 1
dt = CFL * dx / a
listofspatialschemes = ['1st-order upwind']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
```

```
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using Euler and %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-P4_Euler_105.pdf')
  plt.show()

############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3'
############## Choose CFL = a*dt/dx
CFL = 0.95 * np.sqrt(3)
dt = CFL * dx / a
listofspatialschemes = ['2nd-order central']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using RK3 and  %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-P4_RK3_095.pdf')
  plt.show()

############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3'
############## Choose CFL = a*dt/dx
CFL = 1.05 * np.sqrt(3)
dt = CFL * dx / a
listofspatialschemes = ['2nd-order central']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0, dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using RK3 and %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-P4_RK3_105.pdf')
  plt.show()

# %% [markdown]
# Problem 5: Testing the stability of RK4

# %%
############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK4'
############## Choose CFL = a*dt/dx
CFL = 0.95 * 2 * np.sqrt(2)
dt = CFL * dx / a
listofspatialschemes = ['2nd-order central']

```

```python
# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using RK4 and %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-P5_095.pdf')
  plt.show()

############## Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK4'
############## Choose CFL = a*dt/dx
CFL = 1.05 * 2 * np.sqrt(2)
dt = CFL * dx / a
listofspatialschemes = ['2nd-order central']

# Plotting the eigenvalue spectra and numerical solutions for each scheme
for scheme in listofspatialschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, temporalscheme, 'LinearAdv1D',L, T, Nx, Nt, U0,dt)
  fig, ax = plt.subplots(1, 1, figsize=(7, 4))
  ax.plot(x,U0,color=plt.cm.Spectral_r(0))
  for j in range(1,len(U)):
    ax.plot(x,U[j],color=plt.cm.Spectral_r(t[j]/T))
  plt.title('Periodic advection using RK4 and %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-P5_105.pdf')
  plt.show()
```

Matlab (1-based indexing)

```matlab
clear all; close all; clc

xs = -3.2:0.1:3.2;
ys = xs;
[X, Y] = meshgrid(xs, ys);
lambda_dt = X + Y*1i;

figure(1); clf
contourf(X, Y, abs(1 + lambda_dt),[-1,0,1], 'EdgeColor', 'none', 'LineWidth', 2); hold on;
contour(X, Y, abs(1 + lambda_dt), [1,1.001], 'LineColor', 'r', 'LineWidth', 3);
caxis([-1,1])
title('Stability Diagram for Euler');
xlabel('Re(\lambda \Delta t)');
ylabel('Im(\lambda \Delta t)');
xlim([-4, 2]);
ylim([-3.1, 3.1]);
colormap pink;
%saveas(gcf, 'stabDiagsEuler.pdf');

%%
figure(2), clf;
contourf(X, Y, abs(1 + lambda_dt + lambda_dt.^2/2), [-1,0,1], 'EdgeColor', 'none', 'LineWidth'
    , 2); hold on
contour(X, Y, abs(1 + lambda_dt + lambda_dt.^2/2), [1,1.001], 'LineColor', 'r', 'LineWidth',
    2);
caxis([-1,1])
title('Stability Diagram for Runge-Kutta 2');
xlabel('Re(\lambda \Delta t)');
ylabel('Im(\lambda \Delta t)');
xlim([-4, 2]);
```

```
ylim([-3.1, 3.1]);
colormap pink;
%saveas(gcf, 'stabDiagsRK2.pdf');

%%
figure(3), clf;
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
contour(X, Y, abs(1 + lambda_dt + 1/2*lambda_dt.^2 + 1/6*lambda_dt.^3), [1,1.001], 'LineColor'
    , 'r', 'LineWidth', 2);
caxis([-1,1])
title('Stability Diagram for Runge-Kutta 3');
xlabel('Re(\lambda \Delta t)');
ylabel('Im(\lambda \Delta t)');
xlim([-4, 1]);
ylim([-3.1, 3.1]);
colormap pink;
%saveas(gcf, 'stabDiagsRK3.pdf');

%%
figure(4), clf;
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
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);
caxis([-1,1])
title('Stability Diagram for Runge-Kutta 4');
xlabel('Re(\lambda \Delta t)');
ylabel('Im(\lambda \Delta t)');
xlim([-4, 1]);
ylim([-3.1, 3.1]);
colormap pink;
%saveas(gcf, 'stabDiagsRK4.pdf');
```

```
clear all; close all; clc


% Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3';
% Choose CFL = a*dt/dx
CFL = 1; % 1.05 * sqrt(3)

% Initialize case parameters
L = 1; T = 30.6; Nx = 50; Nt = 11;
a = 1;
% Initialize solution at t=0
x = linspace(0, L, Nx);
dx = x(2) - x(1);
dt = CFL * dx / a;
sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
% Initialize list of schemes that will be tested:
listofspatialschemes = {'1st-order upwind', '2nd-order central', '3rd-order upwind', '4th-
    order central', '6th-order Pade'};

% Plotting the eigenvalue spectra and numerical solutions for each scheme
A = [];
for idx = 1:length(listofspatialschemes)
    scheme = listofspatialschemes{idx};
    % Run solver and plot solution
    [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
    %A = [A; -a * D];
    fig = figure('Position', [100, 100, 500, 400]); hold on
    colors = [linspace(0,1,length(t))',zeros(length(t),2)];
    plot(x, U0,'color',colors(1,:),'LineWidth',2); hold on
    for j = 2:length(t)
        plot( x, U(:,j), 'color', colors(j,:),'LineWidth',2);
    end
    hold off
```

```
    title(sprintf('Periodic advection of a Gaussian pulse using %s', scheme));
    xlabel( 'x');
    ylabel( 'u(x,t)');
    %saveas(fig, sprintf('solution-%s.pdf', scheme));
    %close(fig);

    % Plot stability region
    fig = figure('Position', [600, 100, 500, 400]);
    Ngrid = 200; xs = (-3:3.5/Ngrid:0.5); ys = (-3:3.5/Ngrid:3);
    [X, Y] = meshgrid(xs, ys);
    lambda_dt = X + 1i * Y;
    eigvals = eig(-a*D);
    eigvals = diag(eigvals);
    switch temporalscheme
        case 'Euler'
            %contourf(X, Y, abs(1 + lambda_dt), [0, 1], 'LineColor', 'none');
            %hold on
            contour(X, Y, abs(1 + lambda_dt), [0.999,1], 'LineColor', 'k'); hold on
            plot(real(eigvals)*dt,imag(eigvals)*dt,'r.','MarkerSize',15)
        case 'RK2'
            %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2), [0, 1], 'LineColor', 'none');
            %hold on
            contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2), [0.999,1], 'LineColor', 'k');hold on
            plot(real(eigvals)*dt,imag(eigvals)*dt,'r.','MarkerSize',15)
        case 'RK3'
            %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3), [0, 1], 'LineColor', 'none');
            %hold on
            contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3), [0.999,1], 'LineColor', 'k');hold on
            plot(real(eigvals)*dt,imag(eigvals)*dt,'r.','MarkerSize',15)
        case 'RK4'
            %contourf(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3 + (1/24) * lambda_dt.^4), [0, 1], 'LineColor', 'none');
            %hold on
            contour(X, Y, abs(1 + lambda_dt + 0.5 * lambda_dt.^2 + (1/6) * lambda_dt.^3 + (1/24) * lambda_dt.^4), [0.999,1], 'LineColor', 'k');hold on
            plot(real(eigvals)*dt,imag(eigvals)*dt,'r.','MarkerSize',15)
    end
    title(sprintf('Eigenvalue spectrum for %s', scheme));
    xlabel('Re(\lambda \Delta t)');
    ylabel('Im(\lambda \Delta t)');
    %saveas(fig, sprintf('eigenvalues-%s.pdf', scheme));
    %close(fig);
end

return

%% Problem 4: First Set of Schemes

% Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'Euler';
% Choose CFL = a*dt/dx
CFL = 0.95 * 1; % sqrt(3)

% Initialize case parameters
L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
dx = L / Nx;
dt = CFL * dx / a;
% Initialize solution at t=0
x = linspace(0, L, Nx); dx = x(2) - x(1);
sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
% Initialize list of schemes that will be tested:
listofspatialschemes = {'1st-order upwind'};

% Plotting the eigenvalue spectra and numerical solutions for each scheme
```

```
A = [];
for idx = 1:length(listofspatialschemes)
    scheme = listofspatialschemes{idx};
    % Run solver and plot solution
    [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
    A = [A; -a * D];
    fig = figure('Position', [100, 100, 700, 400]);
    colors = [linspace(0,1,length(t))',zeros(length(t),2)];
    plot(x, U0, 'color', colors(1,:));
    hold on
    for j = 2:length(t)
        plot(x, U(:,j), 'color', colors(j,:));
    end
    hold on
    title(sprintf('Periodic advection using Euler and %s', scheme));
    xlabel('x');
    ylabel('u(x,t)');
    colorbar;
    %saveas(fig, 'solution-P4_Euler_095.pdf');
    %close(fig);
end

% Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'Euler';
% Choose CFL = a*dt/dx
CFL = 1.05 * 1; % sqrt(3)

% Initialize case parameters
L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
dx = L / Nx;
dt = CFL * dx / a;
% Initialize solution at t=0
x = linspace(0, L, Nx); dx = x(2) - x(1);
sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0 = U0';
% Initialize list of schemes that will be tested:
listofspatialschemes = {'1st-order upwind'};

% Plotting the eigenvalue spectra and numerical solutions for each scheme
A = [];
for idx = 1:length(listofspatialschemes)
    scheme = listofspatialschemes{idx};
    % Run solver and plot solution
    [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
    A = [A; -a * D];
    fig = figure('Position', [100, 100, 700, 400]);
    colors = [linspace(0,1,length(t))',zeros(length(t),2)];
    plot(x, U0, 'color', colors(1,:));
    hold on
    for j = 2:length(t)
        plot(x, U(:,j), 'color', colors(j,:));
    end
    hold('off');
    title(sprintf('Periodic advection using Euler and %s', scheme));
    xlabel('x');
    ylabel('u(x,t)');
    colorbar
    %saveas(fig, 'solution-P4_Euler_105.pdf');
    %close(fig);
end

%% Problem 4: Second Set of Schemes

% Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3';
% Choose CFL = a*dt/dx
CFL = 0.95 * sqrt(3);

% Initialize case parameters
```

```
L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
dx = L / Nx;
dt = CFL * dx / a;
% Initialize solution at t=0
x = linspace(0, L, Nx); dx = x(2) - x(1);
sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0 = U0';
% Initialize list of schemes that will be tested:
listofspatialschemes = {'2nd-order central'};

% Plotting the eigenvalue spectra and numerical solutions for each scheme
A = [];
for idx = 1:length(listofspatialschemes)
    scheme = listofspatialschemes{idx};
    % Run solver and plot solution
    [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
    A = [A; -a * D];
    fig = figure('Position', [100, 100, 700, 400]);
    colors = [linspace(0,1,length(t))',zeros(length(t),2)];
    plot(x, U0, 'color', colors(1,:));
    hold on
    for j = 2:length(t)
        plot(x, U(:,j), 'color', colors(j,:));
    end
    hold off
    title(sprintf('Periodic advection using RK3 and %s', scheme));
    xlabel('x');
    ylabel('u(x,t)');
    colorbar
    %saveas(fig, 'solution-P4_RK3_095.pdf');
    %close(fig);
end

% Choose temporal scheme between: Exact, Euler, RK2, RK3, RK4
temporalscheme = 'RK3';
% Choose CFL = a*dt/dx
CFL = 1.05 * sqrt(3);

% Initialize case parameters
L = 1; T = 30; Nx = 50; Nt = 11; a = 1;
dx = L / Nx;
dt = CFL * dx / a;
% Initialize solution at t=0
x = linspace(0, L, Nx); dx = x(2) - x(1);
sigma = 3 / 40; U0 = exp(-(x - 0.5).^2 / (2 * sigma^2)); U0=U0';
% Initialize list of schemes that will be tested:
listofspatialschemes = {'2nd-order central'};

% Plotting the eigenvalue spectra and numerical solutions for each scheme
A = [];
for idx = 1:length(listofspatialschemes)
    scheme = listofspatialschemes{idx};
    % Run solver and plot solution
    [t, U, D] = Integrator(true, scheme, temporalscheme, 'LinearAdv1D', L, T, Nx, Nt, U0, dt);
    A = [A; -a * D];
    fig = figure('Position', [100, 100, 700, 400]);
    colors = [linspace(0,1,length(t))',zeros(length(t),2)];
    plot(x, U0, 'color', colors(1,:));
    hold on
    for j = 2:length(t)
        plot(x, U(:,j), 'color', colors(j,:));
    end
    hold off
    title(sprintf('Periodic advection using RK3 and %s', scheme));
    xlabel('x');
    ylabel('u(x,t)');
    colorbar
    %saveas(fig, 'solution-P4_RK3_105.pdf');
    %close(fig);
```

```
end


%% Auxiliary Functions
function [ D ] = D_operator_periodic( n,L,R,a )
    %computes a finite difference operator
    %   n = number of grid points
    %   dx = step size
    %   [R,L]  = right/left bound of stencil
    %   a    = stencils

    % initialize
    D = zeros(n);
    % fill interior domain
    for i=1+L:n+1-R-1
        D(i,i-L:i+R) = a;
    end
    % fill left boundary
    for i=1:L
        D(i,1:i+R)          = a(L-i+2:end);
        D(i,end-L+i:end)    = a(1:L-i+1);
    end

    % fill right boundary
    for i=n+1-R:n
        D(i,end-L+0+(i-n):end)  = a(1:L+(n+2-i)-1);
        D(i,1:(i-n+R-0))        = a(L+(n+2-i):end);
    end
end

% Functor for the 1D advection equation
function dUdt = LinearAdv1D(t, U, D)
    % Initialize velocity
    a = 1;
    % Compute dU/dt
    dUdt = -a * (D * U);
end

% Time integrator
function sol = TimeIntegration(scheme, time_limits, dt, U0, D, t_eval)
    % Exact solution using MATLAB's ODE solver
    if strcmp(scheme, 'Exact')
        options = odeset('RelTol', 1.0e-6, 'AbsTol', 1.0e-6);
        [~, sol] = ode45(@(t, U) LinearAdv1D(t, U, D), t_eval, U0, options);
        sol = sol';
        return;
    else
        % Number of timesteps
        Nt = ceil((time_limits(2) - time_limits(1)) / dt) + 1;
        Uold = U0;
        % Vector of stored solutions -- add initial condition
        sol = [];
        keval = 1;
        if t_eval(keval) == time_limits(1)
            sol(:, keval) = U0;
            keval = keval + 1;
        end
        % Loop over all timesteps
        for k = 1:Nt
            told = time_limits(1) + dt * (k - 1);
            tnew = told + dt;
            % Compute RHS at told
            rhs = LinearAdv1D(told, Uold, D);
            % Update new solution with explicit scheme
            switch scheme
                case 'Euler'
                    Unew = Uold + dt * rhs;
                case 'RK2'
```

```
                    K1 = dt*LinearAdv1D(told,Uold,D);
                    K2 = dt*LinearAdv1D(told+dt,Uold+K1,D);
                    Unew = Uold+1/2*(K1+K2);
                case 'RK3'
                    K1 = dt*LinearAdv1D(told,Uold,D);
                    K2 = dt*LinearAdv1D(told+dt/2,Uold+K1/2,D);
                    K3 = dt*LinearAdv1D(told+dt,Uold+2*K2-K1,D);
                    Unew = Uold+1/6*(K1+4*K2+K3);
                case 'RK4'
                    K1 = LinearAdv1D(told, Uold, D);
                    K2 = LinearAdv1D(told + 0.5 * dt, Uold + 0.5 * dt * K1, D);
                    K3 = LinearAdv1D(told + 0.5 * dt, Uold + 0.5 * dt * K2, D);
                    K4 = LinearAdv1D(told + dt, Uold + dt * K3, D);
                    Unew = Uold + dt * (K1 + 2 * K2 + 2 * K3 + K4) / 6;
            end
            %Update stored solution vector
            if keval <= length(t_eval) && t_eval(keval) >= told && t_eval(keval) <= tnew
                sol(:, keval) = Uold + (Unew - Uold) * (t_eval(keval) - told) / dt;
                keval = keval + 1;
            end
            % Move to next timestep
            Uold = Unew;
        end
    end
end

function [t, U, D] = Integrator(periodic, operator, temporalscheme, problem, L, T, Nx, Nt, U0, dt)
    % Initialize spatial domain
    x = linspace(0, L, Nx);
    dx = x(2) - x(1);

    % Initialize temporal domain
    t = linspace(0, T, Nt);

    % Construct spatial matrix operator
    switch operator
        % All periodic cases
        case '1st-order upwind'
            D = D_operator_periodic(Nx, 1, 0, [-1/dx, 1/dx]);
        case '2nd-order central'
            D = D_operator_periodic(Nx, 1, 1, [-1/(2*dx), 0, 1/(2*dx)]);
        case '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'
            D = D_operator_periodic(Nx, 2, 2, [1/(12*dx), -8/(12*dx), 0, 8/(12*dx), -1/(12*dx)]);
        case '6th-order Pade'
            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 = inv(DL) * DR;
        otherwise
            error("The %s operator '%s' is not yet implemented!", ...
                ifelse(periodic, 'periodic', 'non-periodic'), operator);
    end

    U = TimeIntegration(temporalscheme, [0, T], dt, U0, D, t);
end
```