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

Homework #3

Due date: February 29, 2024

Problem 1 · Develop a general purpose solver for systems of equations of the form

$$\frac{\mathrm{d}\mathbf{U}(t)}{\mathrm{d}t} = \mathbf{F}\left(t, \mathbf{U}(t)\right)$$

where $\mathbf{U}(t)$ is a vector of functions of time, and $\mathbf{F}$ is an operator that depends on the governing equations under consideration. Your code should take the following inputs:

- The ability to choose between periodic or non-periodic domains.
- The ability to choose the finite-difference spatial derivative operator.
- The ability to choose the governing equation.
- The length L of the physical domain, $x \in [0, L]$.
- The length T of the temporal domain, $t \in [0, T]$.
- The number N_x of discrete grid points $x_n, n \in \{0, 1, \ldots, N_x - 1\}$.
    - ↪ If the domain is periodic, then $\Delta x = L/N_x$ (end-point $x = L$ excluded).
    - ↪ If the domain is not periodic, then $\Delta x = L/(N_x - 1)$ (end-point $x = L$ included).
- The number N_t of discrete time levels $t_m, m \in \{0, 1, \ldots, N_t - 1\}$ (used only for reporting the solution!).
    - ↪ The time levels are separated by $\Delta t = T/(N_t - 1)$ (end-point $t = T$ included).
- The initial condition $\mathbf{U}(t = 0)$.

It should return:

- The set of discrete time levels $t_m, m \in \{0, 1, \ldots, N_t - 1\}$.
- The set of solutions $\mathbf{U}(t_m), m \in \{0, 1, \ldots, N_t - 1\}$.

You may use the following Python code, or develop your own:

```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()

```

```python
# 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 ('ForwardOrder1FirstDeriv',True):   # Periodic 1st-order forward differences
      D = D_operator_periodic(Nx,0,1,[-1/dx,1/dx])
    case ('BackwardOrder1FirstDeriv',True):  # Periodic 1st-order backward differences
      D = D_operator_periodic(Nx,1,0,[-1/dx,1/dx])
    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 that U has the format (Nt x Nx)
      U = sol.y.transpose()
      # Return outputs
      return t, U
    case _:
      raise Exception("The case '%s' is not yet implement!" % problem)
```

Problem 2 · Using the code of Problem 1, solve the problem that we have mainly considered in class so far, i.e., the periodic semi-discrete 1D advection with the wavespeed $a = 1$, domain length $L = 1$, final time $T = 10$, $N_x = 50$ discrete 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 each scheme listed below:

(1) Plot the eigenvalue spectrum of the matrix $\mathbf{A} = -a\mathbf{D}$.

(2) Compare your numerical solution to the exact expected solution at the times $t_m, m \in \{1, 2, \ldots, 10\}$.

(3) Name and explain the different phenomena you observe based on the previously computed eigenvalue spectra and the known properties of the scheme.

The list of schemes:

- First-order upwind:
$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u_i - u_{i-1}}{\Delta x}$$
- Second-order central:
$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x}$$
- Third-order upwind:
$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{2u_{i+1} + 3u_i - 6u_{i-1} + u_{i-2}}{6\Delta x}$$
- Fourth-order central:
$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{-u_{i+2} + 8u_{i+1} - 8u_{i-1} + u_{i-2}}{12\Delta x}$$
- Sixth-order Padé (**this scheme only needs to be studied by students taking AE 410/CSE 461 for four credit hours**):
$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u_{i+2} + 28u_{i+1} - 28u_{i-1} - u_{i-2}}{36\Delta x} - \frac{1}{3}\left(\left.\frac{\partial u}{\partial x}\right|_{i-1} + \left.\frac{\partial u}{\partial x}\right|_{i+1}\right)$$

Solution:

Using the code from Problem 1, the linear advection equation was solved on the domain $x \in [0, 1]$ with $N_x = 51$ points for the time frame $t \in [0, 10]$ using $u_0(x) = \exp[-(x - L/2)^2/2\sigma^2]$ as initial condition. For the spatial derivative, four difference schemes have been used and their properties based on an eigenanalysis have been investigated. The spectra for these four schemes can be seen in Fig. 1-5, and the following observations can be made:

1^{st} order upwind: all eigenvalues are complex and lie within the left half-plane → all eigenmodes are oscillatory and stable (damped), resulting in substantial amplitude decrease over time.

2^{nd} order central: all eigenvalues are purely imaginary → all eigenmodes are oscillatory and neutrally stable; substantial dispersion can be observed.

3^{rd} order upwind: all eigenvalues are complex and lie within the left half-plane and are closer to the imaginary axis than the 1^{st} order upwind scheme → all eigenmodes are oscillatory and stable (damped), resulting in some amplitude decrease over time.

4^{th} order central: all eigenvalues are purely imaginary but have a bigger range than the 2^{nd} order scheme thus resolving higher frequency content → all eigenmodes are oscillatory and neutrally stable; dispersion can be observed trailing the pulse.

6^{th} order Padé: all eigenvalues are purely imaginary but have a bigger range than the 4^{th} order scheme thus resolving higher frequency content → all eigenmodes are oscillatory and neutrally stable; no dispersion observed, resulting in very good solution quality.

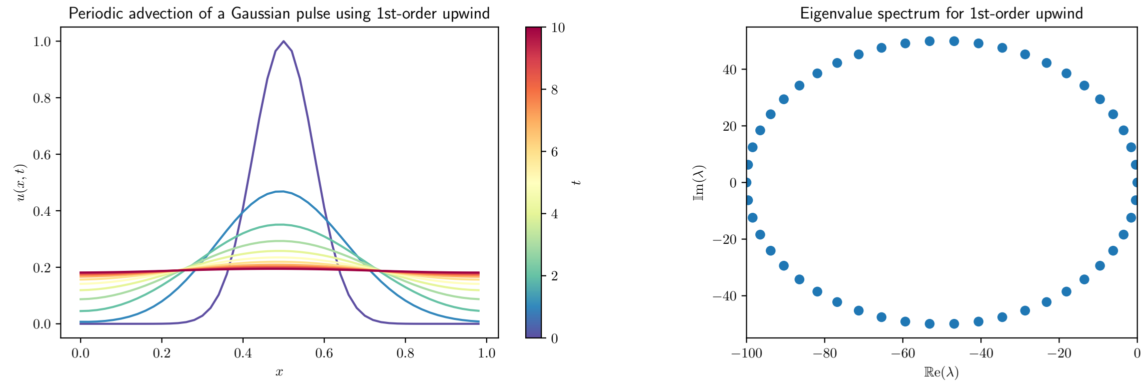

Figure 1: Unsteady solution (left) and spectrum of matrix A (right) for the 1^{st} order upwind scheme

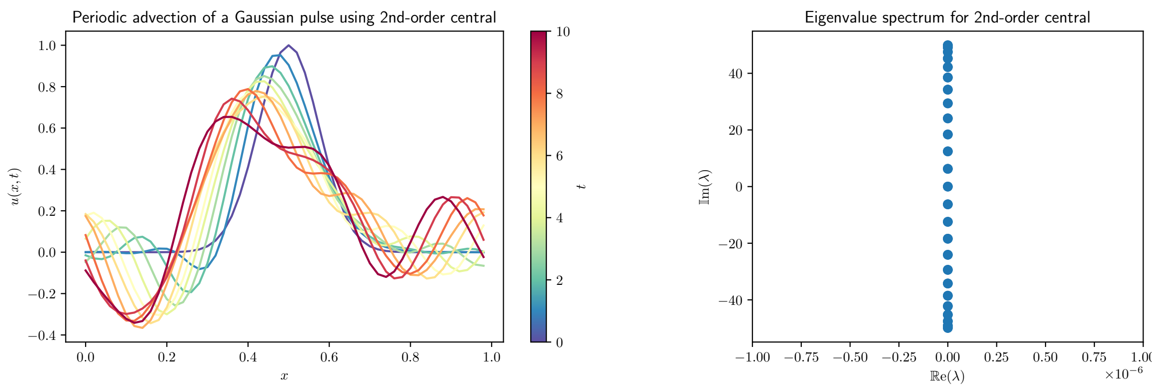

Figure 2: Unsteady solution (left) and spectrum of matrix A (right) for the 2^{nd} order central scheme.

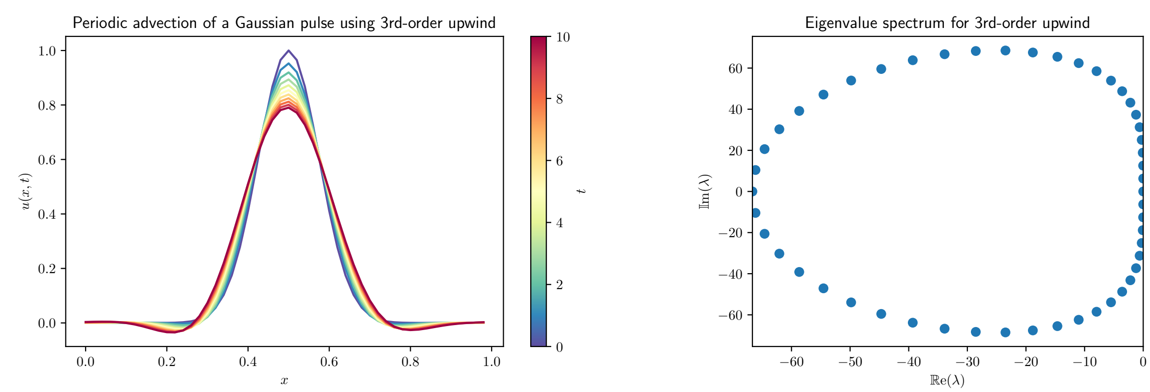

Figure 3: Unsteady solution (left) and spectrum of matrix A (right) for the 3^{rd} order upwind scheme.

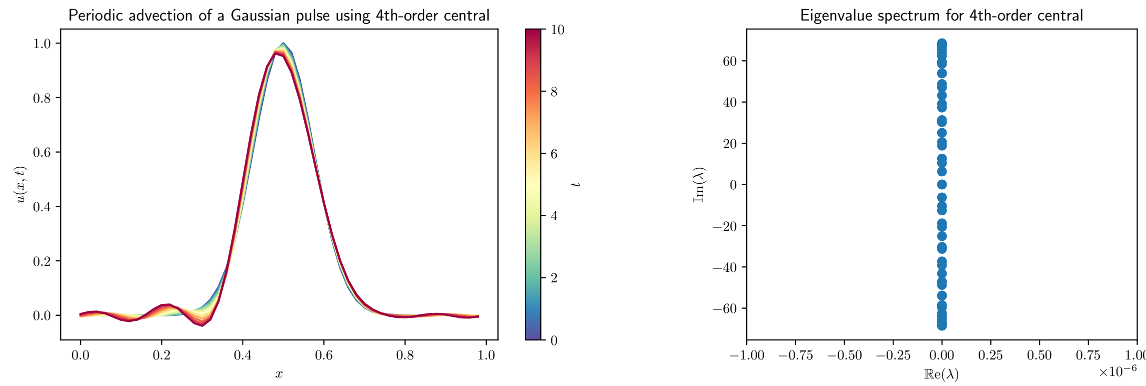

Figure 4: Unsteady solution (left) and spectrum of matrix A (right) for the 4^{th} order central scheme.

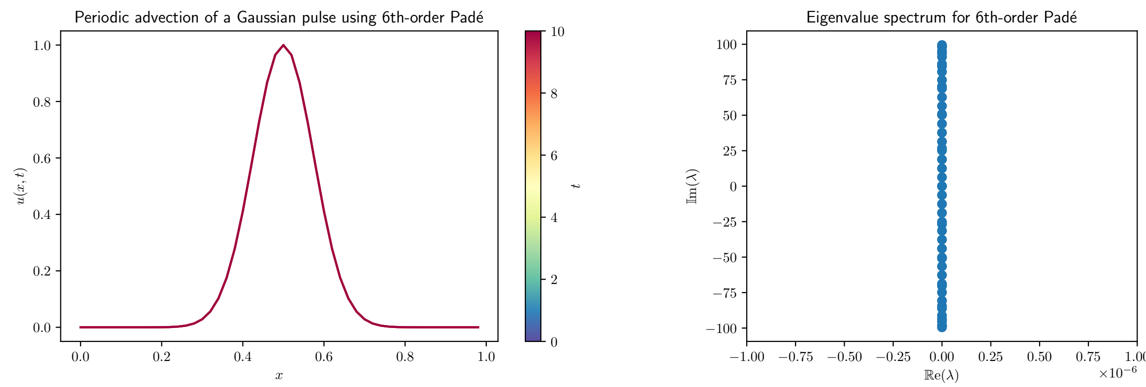

Figure 5: Unsteady solution (left) and spectrum of matrix A (right) for the 6^{th} order Padé scheme.

Problem 3 · For each of the schemes you have used in Problem 2:

(1) Derive the analytical expression of the modified wavenumber $\kappa^\star$ using Fourier error analysis.

(2) Plot the real and imaginary parts of $\kappa^\star \Delta x$ as a function of $\kappa \Delta x$, for $\kappa \Delta x \in [0, \pi]$.

(3) On the same plots as in (2), plot the discrete values of $\kappa_n^\star, n \in \{0, \ldots, N_x - 1\}$, obtained from the eigenvalues of the matrix $\mathbf{A}$.

Solution:

Question (1):

1^{st} **order upwind scheme**

$$\left.\frac{\delta f}{\delta x}\right|_n = \frac{f_n - f_{n-1}}{\Delta x} = \frac{e^{i\kappa x_n} - e^{i\kappa(x_n - \Delta x)}}{\Delta x} = e^{i\kappa x_n} \frac{1 - e^{-i\kappa \Delta x}}{\Delta x}$$
$$= e^{i\kappa x_n} \frac{1}{\Delta x} \left[i \sin(\kappa \Delta x) - \cos(\kappa \Delta x) + 1 \right]$$
$$= i\kappa^\star$$

Thus we get the relation

$$\kappa^\star \Delta x = \sin(\kappa \Delta x) + i\left(\cos(\kappa \Delta x) - 1\right) \tag{1}$$

2^{nd} **order central scheme**

$$\begin{aligned}\left.\frac{\delta f}{\delta x}\right|_n &= \frac{f_{n+1} - f_{n-1}}{2\Delta x} = \frac{e^{i\kappa(x_n+\Delta x)} - e^{i\kappa(x_n-\Delta x)}}{2\Delta x} = e^{i\kappa x_n}\frac{e^{i\kappa\Delta x} - e^{-i\kappa\Delta x}}{2\Delta x}\\ &= e^{i\kappa x_n}\frac{i}{2\Delta x}\sin(\kappa\Delta x)\\ &= i\kappa^\star\end{aligned}$$

Thus we get the relation

$$\kappa^\star \Delta x = \sin(\kappa\Delta x)$$

3^{rd} **order upwind scheme**

Similarly to the derivations above, we can get

$$\left.\frac{\delta f}{\delta x}\right|_n = e^{i\kappa x_n}\frac{1}{6\Delta x}\left[2e^{i\kappa\Delta x} + 3 - 6e^{-i\kappa\Delta x} + e^{-2i\kappa\Delta x}\right]$$

and thus

$$\kappa^\star\Delta x = \left(\frac{4}{3}\sin(\kappa\Delta x) - \frac{1}{6}\sin(2\kappa\Delta x)\right) + i\left(\frac{2}{3}\cos(\kappa\Delta x) - \frac{1}{6}\cos(2\kappa\Delta x) - \frac{1}{2}\right)$$

4^{th} **order central scheme**

$$\left.\frac{\delta f}{\delta x}\right|_n = e^{i\kappa x_n}\frac{1}{12\Delta x}\left[-e^{2i\kappa\Delta x} + 8e^{i\kappa\Delta x} - 8e^{-i\kappa\Delta x} + e^{-2i\kappa\Delta x}\right]$$

and thus

$$\kappa^\star\Delta x = \frac{4}{3}\sin(\kappa\Delta x) - \frac{1}{6}\sin(2\kappa\Delta x)$$

6^{th} **order Padé**

$$\kappa^\star\Delta x = \frac{1}{36}\frac{2\sin(2\kappa\Delta x) + 56\sin(\kappa\Delta x))}{1 + \frac{2}{3}\cos(\kappa\Delta x)}$$

Questions (2) and (3):

As seen in Fig. 6, the modified wave number plots show the previously observed characteristic properties of the differentiation schemes in a more physically relevant framework. Following properties can be observed:

- The wavenumbers obtained from Fourier error analysis or by direct calculation of the eigenvalues of the circulant matrix operator are identical.
- Both central schemes have a zero imaginary part and thus are neutrally stable.
- Both upwinding schemes have non-zero imaginary part and thus are dissipative.
- The upwinding schemes have the identical real part as the central schemes with one order higher, as has been proven for the 1^{st} order upwind scheme in Homework 1.

- Higher order schemes are able to capture higher wave numbers and thus higher frequency content.
- For low wave numbers, all schemes adequately represent the exact wave number.
- The 6^{th} order Padé scheme shows significantly better capturing of high wave number content while maintaining zero dissipation.

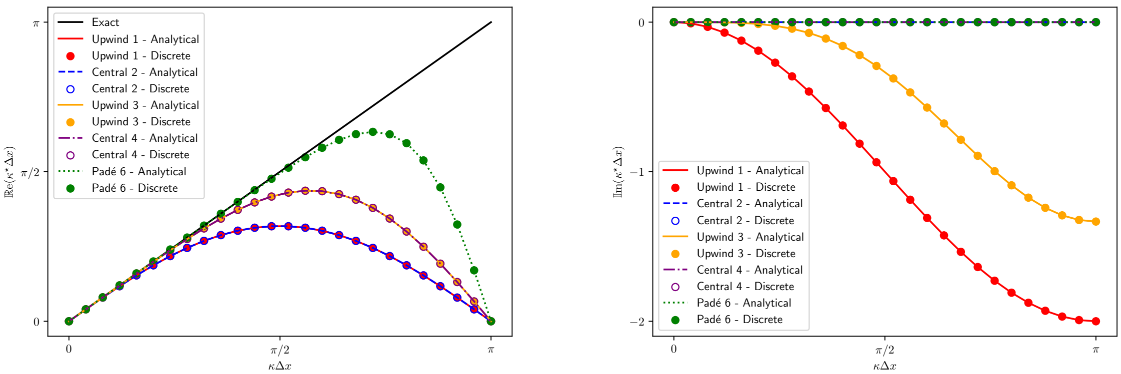

Figure 6: Modified wave number plots for both real (left) and imaginary (right) parts for varying differentiation schemes (symbols represent analytically derived distribution exploiting the circulant properties of the operators)

Solution: Full codes:

Python (0-based indexing)

```
# %% [markdown]
# Solver for the semi-discrete 1D avection problem:

# %%
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)

# %% [markdown]
# Problem 2:

# %%
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
A = []
for scheme in listofschemes:
  # Run solver and plot solution
  t, U, D = Integrator(True, scheme, 'LinearAdv1D',L, T, Nx, Nt, U0)
  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,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()

  # Plot eigenvalue spectrum
  fig, ax = plt.subplots(1, 1, figsize=(5, 4))
  eigenvalues, eigenvectors = LA.eig(-a*D)
  relambda = [ele.real for ele in eigenvalues]
  imlambda = [ele.imag for ele in eigenvalues]
  ax.scatter(relambda, imlambda)
  plt.xlim(min(-1e-6,np.min(relambda)),max(1e-6,np.max(relambda)))
  plt.title('Eigenvalue spectrum for %s' % scheme)
  plt.xlabel(r'$\mathbb{R}\mathrm{e}(\lambda)$');plt.ylabel(r'$\mathbb{I}\mathrm{m}(\lambda)$');
  plt.savefig('eigenvalues-%s.pdf' % scheme)
  plt.show()

# %% [markdown]
# Problem 3:

# %%
import cmath
# We only plot the first N/2+1 eigenvalues, corresponding to k*dx in [0,pi]
halfNx = int(0.5*Nx+1)
kndx = np.zeros(halfNx)

# Initialize abscissae
for n in range(halfNx):
  kndx[n] = 2*np.pi*n/Nx

# Exact modified wavenumbers
knmoddx_exact_upwind1  = np.sin(kndx) + (np.cos(kndx) - 1) * 1j
knmoddx_exact_central2 = np.sin(kndx)
knmoddx_exact_upwind3  = 4*np.sin(kndx)/3 - np.sin(2*kndx)/6 + (2*np.cos(kndx)/3 - np.cos(2*kndx)/6 - 1/2) * 1j
knmoddx_exact_central4 = 4*np.sin(kndx)/3 - np.sin(2*kndx)/6
knmoddx_exact_pade6    = (2*np.sin(2*kndx) + 56*np.sin(kndx))/(36*(1+2*np.cos(kndx)/3))

# Discrete modified wavenumbers
knmoddx_discrete_upwind1  = np.zeros(halfNx,dtype=complex)
knmoddx_discrete_central2 = np.zeros(halfNx,dtype=complex)
knmoddx_discrete_upwind3  = np.zeros(halfNx,dtype=complex)
knmoddx_discrete_central4 = np.zeros(halfNx,dtype=complex)
knmoddx_discrete_pade6    = np.zeros(halfNx,dtype=complex)

# First rows the matrix A
firstrow_upwind1 = A[0][0]
firstrow_central2 = A[1][0]
firstrow_upwind3 = A[2][0]
```

```python
firstrow_central4 = A[3][0]
firstrow_pade6 = A[4][0]

# Loop over all schemes to compute discrete eigenvalues
for n in range(halfNx):
  # n-th eigenvector of the circulant matrix
  Vn = np.zeros(Nx,dtype=complex)
  for k in range(Nx):
    Vn[k] = np.exp(2*np.pi*k*n*1j/Nx)
  # The n-th eigenvalue of A is the dot product of the n-th eigenvector with the first row
      of A (Eq 46 of notes 3)
  knmoddx_discrete_upwind1[n]  = dx*np.dot(firstrow_upwind1, Vn)*1j/a
  knmoddx_discrete_central2[n] = dx*np.dot(firstrow_central2, Vn)*1j/a
  knmoddx_discrete_upwind3[n]  = dx*np.dot(firstrow_upwind3, Vn)*1j/a
  knmoddx_discrete_central4[n] = dx*np.dot(firstrow_central4, Vn)*1j/a
  knmoddx_discrete_pade6[n]    = dx*np.dot(firstrow_pade6, Vn)*1j/a

# Plot real part
fig, ax = plt.subplots(1, 1, figsize=(7, 5))
ax.plot(kndx, kndx, 'k', label=r'Exact')
ax.plot(kndx, knmoddx_exact_upwind1.real, color='red', label=r'Upwind 1 - Analytical')
ax.scatter(kndx, knmoddx_discrete_upwind1.real, color='red', label=r'Upwind 1 - Discrete')
ax.plot(kndx, knmoddx_exact_central2.real, color='blue', linestyle='dashed', label=r'
    Central 2 - Analytical')
ax.scatter(kndx, knmoddx_discrete_central2.real, facecolors='none', edgecolors='blue',
    label=r'Central 2 - Discrete')
ax.plot(kndx, knmoddx_exact_upwind3.real, color='orange', label=r'Upwind 3 - Analytical')
ax.scatter(kndx, knmoddx_discrete_upwind3.real, color='orange', label=r'Upwind 3 -
    Discrete')
ax.plot(kndx, knmoddx_exact_central4.real, color='purple', linestyle='dashdot', label=r'
    Central 4 - Analytical')
ax.scatter(kndx, knmoddx_discrete_central4.real, facecolors='none', edgecolors='purple',
    label=r'Central 4 - Discrete')
ax.plot(kndx, knmoddx_exact_pade6.real, color='green', linestyle='dotted', label=r'Padé 6
    - Analytical')
ax.scatter(kndx, knmoddx_discrete_pade6.real, color='green', label=r'Padé 6 - Discrete')
plt.ylabel(r'$\mathbb{R}\mathrm{e}(\kappa^\star \Delta x)$');plt.xlabel(r'$\kappa\Delta x$
    ')
plt.legend()
tick_labels = [r'$0$',r'$\pi/2$',r'$\pi$']
ticks = [0,np.pi/2,np.pi]
ax.set_xticks(ticks); ax.set_yticks(ticks)
ax.set_xticklabels(tick_labels); ax.set_yticklabels(tick_labels)
plt.savefig('modwavenumber_real.pdf')
plt.show()

# Plot imaginary part
fig, ax = plt.subplots(1, 1, figsize=(7, 5))
ax.plot(kndx, knmoddx_exact_upwind1.imag, color='red', label=r'Upwind 1 - Analytical')
ax.scatter(kndx, knmoddx_discrete_upwind1.imag, color='red', label=r'Upwind 1 - Discrete')
ax.plot(kndx, knmoddx_exact_central2.imag, color='blue', linestyle='dashed', label=r'
    Central 2 - Analytical')
ax.scatter(kndx, knmoddx_discrete_central2.imag, facecolors='none', edgecolors='blue',
    label=r'Central 2 - Discrete')
ax.plot(kndx, knmoddx_exact_upwind3.imag, color='orange', label=r'Upwind 3 - Analytical')
ax.scatter(kndx, knmoddx_discrete_upwind3.imag, color='orange', label=r'Upwind 3 -
    Discrete')
ax.plot(kndx, knmoddx_exact_central4.imag, color='purple', linestyle='dashdot', label=r'
    Central 4 - Analytical')
ax.scatter(kndx, knmoddx_discrete_central4.imag, facecolors='none', edgecolors='purple',
    label=r'Central 4 - Discrete')
ax.plot(kndx, knmoddx_exact_pade6.imag, color='green', linestyle='dotted', label=r'Padé 6
    - Analytical')
ax.scatter(kndx, knmoddx_discrete_pade6.imag, color='green', label=r'Padé 6 - Discrete')
plt.ylabel(r'$\mathbb{I}\mathrm{m}(\kappa^\star \Delta x)$');plt.xlabel(r'$\kappa\Delta x$
    ')
```

```
plt.legend()
ytick_labels = [r'$0$',r'$-1$',r'$-2$']
yticks = [0,-1,-2]
ax.set_xticks(ticks); ax.set_yticks(yticks)
ax.set_xticklabels(tick_labels); ax.set_yticklabels(ytick_labels)
plt.savefig('modwavenumber_imag.pdf')
plt.show()
```

Matlab (1-based indexing)

```
clear all; clc; close all;

%% input parameter
T   = 10;
Nt  = 11;
L   = 1;
Nx  = 51;

%% compute unsteady solution and eigenvalues of differentiation operators
[t,u1,x,eigvals1] = integrator(L,Nx,T,Nt,'Upwind1FirstDeriv','LinearAdvection');
[t,u2,x,eigvals2] = integrator(L,Nx,T,Nt,'Central2FirstDeriv','LinearAdvection');
[t,u3,x,eigvals3] = integrator(L,Nx,T,Nt,'Upwind3FirstDeriv','LinearAdvection');
[t,u4,x,eigvals4] = integrator(L,Nx,T,Nt,'Central4FirstDeriv','LinearAdvection');
[t,u5,x,eigvals5] = integrator(L,Nx,T,Nt,'Pade6FirstDeriv','LinearAdvection');

%% plot unsteady solution
figure(1), clf; hold on
for i=1:length(t)
    plot(x,u1(i,:),'LineWidth',2,'Color',[i/length(t),0,0])
end
xlim([0,1.4])
ylim([-0.3,1.4])
set(gca,'FontSize',18)
xlabel('x','Interpreter','Latex','FontSize',30)
ylabel('u','Interpreter','Latex','FontSize',30)
leg=legend('$t=0$ ($u_{exact}$)','$t=1$','$t=2$','$t=3$','$t=4$','$t=5$','$t=6$',...
           '$t=7$','$t=8$','$t=9$','$t=10$');
set(leg,'Interpreter','Latex','FontSize',22)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 850 400])
fname = 'HW3P2_1stup';
print(gcf,fname,'-djpeg90')

figure(2), clf; hold on
for i=1:length(t)
    plot(x,u2(i,:),'LineWidth',2,'Color',[i/length(t),0,0])
end
xlim([0,1.4])
ylim([-0.3,1.4])
set(gca,'FontSize',18)
xlabel('x','Interpreter','Latex','FontSize',30)
ylabel('u','Interpreter','Latex','FontSize',30)
leg=legend('$t=0$ ($u_{exact}$)','$t=1$','$t=2$','$t=3$','$t=4$','$t=5$','$t=6$',...
           '$t=7$','$t=8$','$t=9$','$t=10$');
set(leg,'Interpreter','Latex','FontSize',22)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 850 400])
fname = 'HW3P2_2ndcentral';
print(gcf,fname,'-djpeg90')

figure(3), clf; hold on
for i=1:length(t)
    plot(x,u3(i,:),'LineWidth',2,'Color',[i/length(t),0,0])
end
```

```
xlim([0,1.4])
ylim([-0.3,1.4])
set(gca,'FontSize',18)
xlabel('x','Interpreter','Latex','FontSize',30)
ylabel('u','Interpreter','Latex','FontSize',30)
leg=legend('$t=0$ ($u_{exact}$)','$t=1$','$t=2$','$t=3$','$t=4$','$t=5$','$t=6$',...
           '$t=7$','$t=8$','$t=9$','$t=10$');
set(leg,'Interpreter','Latex','FontSize',22)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 850 400])
fname = 'HW3P2_3rdup';
print(gcf,fname,'-djpeg90')

figure(4), clf; hold on
for i=1:length(t)
    plot(x,u4(i,:),'LineWidth',2,'Color',[i/length(t),0,0])
end
xlim([0,1.4])
ylim([-0.3,1.4])
set(gca,'FontSize',18)
xlabel('x','Interpreter','Latex','FontSize',30)
ylabel('u','Interpreter','Latex','FontSize',30)
leg=legend('$t=0$ ($u_{exact}$)','$t=1$','$t=2$','$t=3$','$t=4$','$t=5$','$t=6$',...
           '$t=7$','$t=8$','$t=9$','$t=10$');
set(leg,'Interpreter','Latex','FontSize',22)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 850 400])
fname = 'HW3P2_4thcentral';
print(gcf,fname,'-djpeg90')

figure(5), clf; hold on
for i=1:length(t)
    plot(x,u5(i,:),'LineWidth',2,'Color',[i/length(t),0,0])
end
xlim([0,1.4])
ylim([-0.3,1.4])
set(gca,'FontSize',18)
xlabel('x','Interpreter','Latex','FontSize',30)
ylabel('u','Interpreter','Latex','FontSize',30)
leg=legend('$t=0$ ($u_{exact}$)','$t=1$','$t=2$','$t=3$','$t=4$','$t=5$','$t=6$',...
           '$t=7$','$t=8$','$t=9$','$t=10$');
set(leg,'Interpreter','Latex','FontSize',22)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 850 400])
fname = 'HW3P2_6thpade';
print(gcf,fname,'-djpeg90')

%% plot eigenspectra
figure(11), clf;
plot(real(eigvals1),imag(eigvals1),'k.')
set(gca,'FontSize',18)
xlabel('$\Re (\lambda)$','FontSize',30,'Interpreter','Latex')
ylabel('$\Im (\lambda)$','FontSize',30,'Interpreter','Latex')
xlim([-120,0])
ylim([-70,70])
leg=legend('$1^{st}$ order upwind');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 450 400])
fname = 'HW3P2_1stup_spectrum';
print(gcf,fname,'-djpeg90')

figure(12), clf;
plot(real(eigvals2),imag(eigvals2),'k.')
set(gca,'FontSize',18)
xlabel('$\Re (\lambda)$','FontSize',30,'Interpreter','Latex')
```

```
ylabel('$\Im (\lambda)$','FontSize',30,'Interpreter','Latex')
xlim([-1,1]*1e-10)
ylim([-70,70])
leg=legend('$2^{nd}$ order central');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 450 400])
fname = 'HW3P2_2ndcentral_spectrum';
print(gcf,fname,'-djpeg90')

figure(13), clf;
plot(real(eigvals3),imag(eigvals3),'k.')
set(gca,'FontSize',18)
xlabel('$\Re (\lambda)$','FontSize',30,'Interpreter','Latex')
ylabel('$\Im (\lambda)$','FontSize',30,'Interpreter','Latex')
xlim([-80,0])
ylim([-80,80])
leg=legend('$3^{rd}$ order upwind');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 450 400])
fname = 'HW3P2_3rdup_spectrum';
print(gcf,fname,'-djpeg90')

figure(14), clf;
plot(real(eigvals4),imag(eigvals4),'k.')
set(gca,'FontSize',18)
xlabel('$\Re (\lambda)$','FontSize',30,'Interpreter','Latex')
ylabel('$\Im (\lambda)$','FontSize',30,'Interpreter','Latex')
xlim([-1,1]*1e-10)
ylim([-90,90])
leg=legend('$4^{th}$ order central');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 450 400])
fname = 'HW3P2_4thcentral_spectrum';
print(gcf,fname,'-djpeg90')

figure(15), clf;
plot(real(eigvals5),imag(eigvals5),'k.')
set(gca,'FontSize',18)
xlabel('$\Re (\lambda)$','FontSize',30,'Interpreter','Latex')
ylabel('$\Im (\lambda)$','FontSize',30,'Interpreter','Latex')
xlim([-1,1]*1e-10)
ylim([-90,90])
leg=legend('$6^{th}$ order Pade');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 450 400])
fname = 'HW3P2_6thpade_spectrum';
print(gcf,fname,'-djpeg90')
```

```
clear all; close all; clc

%% parameters
L            = 1;
nx           = 51;
x            = linspace(0,1,nx+2);
x            = x(1:end-1);
dx           = L/nx;

%% analytical eigenvalues of FD schemes
kdx              = linspace(0,pi,100);
firstupwind      = sin(kdx) + i*(cos(kdx)-1);
secondcentral    = sin(kdx);
thirdupwind      = -i/6*(3 - 4*cos(kdx) + cos(2*kdx) - 2*i*(-4 + cos(kdx)).*sin(kdx));
```

```
fourthcentral    = 4/3*sin(kdx)-1/6*sin(2*kdx);
sixthpade        = 1/36*( 2*sin(2*kdx)+56*sin(kdx) )./( 1+2/3*cos(kdx) );

%% eigenvalues of circulant matrix for 1st order updwind
D    = D1_operator( nx,dx,0,1,[-1,1] ); % first upwind
[ k_tilde_1up,dxkm_1up ] = circD_wavenum( D );
D    = D1_operator( nx,dx,1,1,[-1,0,1]/2 ); % second central
[ k_tilde_2cfd,dxkm_2cfd ] = circD_wavenum( D );
D    = D1_operator( nx,dx,1,2,[1,-6,3,2]/6 ); % third upwind
[ k_tilde_3up,dxkm_3up] = circD_wavenum( D );
D    = D1_operator( nx,dx,2,2,[1,-8,0,8,-1]/12 ); % fourth central
[ k_tilde_4cfd,dxkm_4cfd ] = circD_wavenum( D );
D    = D1_operatorPade( nx,dx ); % sixth Pade
[ k_tilde_6pade,dxkm_6pade ] = circD_wavenum( D );

%% plot
line     = 3;
labelsize    = 24;
legendsize   = 20;

figure(1), clf;
a(1) = subplot(1,2,1); hold on
h(1) = plot(kdx,real(firstupwind),'k-','LineWidth',line,'LineStyle','-');
h(2) = plot(kdx,real(secondcentral),'b','LineWidth',line,'LineStyle','--');
h(3) = plot(kdx,real(thirdupwind),'m','LineWidth',line);
h(4) = plot(kdx,real(fourthcentral),'c','LineStyle','--','LineWidth',line);
h(5) = plot(kdx,real(sixthpade),'g','LineStyle','--','LineWidth',line);
plot(dxkm_1up,real(k_tilde_1up*dx),'k','LineStyle','none','Marker','d')
plot(dxkm_2cfd,real(k_tilde_2cfd*dx),'b','LineStyle','none','Marker','x')
plot(dxkm_3up,real(k_tilde_3up*dx),'m','LineStyle','none','Marker','>')
plot(dxkm_4cfd,real(k_tilde_4cfd*dx),'c','LineStyle','none','Marker','p')
plot(dxkm_6pade,real(k_tilde_6pade*dx),'g','LineStyle','none','Marker','p')
h(6) = plot(kdx,kdx,'k','LineWidth',line);
xlim([0,pi]); ylim([0,pi])
set(gca,'FontSize',20)
xlabel('$k_m \Delta x$','Interpreter','Latex','FontSize',labelsize)
ylabel('Re$(\tilde{k}_m \Delta x)$','Interpreter','Latex','FontSize',labelsize)
leg=legend(h,'$1^{st}$ order updwind','$2^{nd}$ order central',...
           '$3^{rd}$ order updwind','$4^{th}$ order central','$6^{th}$ order Pad\''e','exact');
set(leg,'Interpreter','Latex','FontSize',legendsize,'Location','NorthWest')

a(2) = subplot(1,2,2); hold on
plot(kdx,imag(firstupwind),'k','LineWidth',line,'LineStyle','-')
plot(kdx,imag(secondcentral),'b','LineWidth',line,'LineStyle','--')
plot(kdx,imag(thirdupwind),'m','LineWidth',line)
plot(kdx,imag(fourthcentral),'c','LineStyle','-.','LineWidth',line)
plot(kdx,imag(sixthpade),'g','LineStyle','-.','LineWidth',line)
plot(dxkm_1up,imag(k_tilde_1up*dx),'k-','LineStyle','none','Marker','d')
plot(dxkm_2cfd,imag(k_tilde_2cfd*dx),'b','LineStyle','none','Marker','x')
plot(dxkm_3up,imag(k_tilde_3up*dx),'m','LineStyle','none','Marker','>')
plot(dxkm_4cfd,imag(k_tilde_4cfd*dx),'c','LineStyle','none','Marker','p')
plot(dxkm_6pade,imag(k_tilde_6pade*dx),'c','LineStyle','none','Marker','p')
xlim([0,pi])
set(gca,'FontSize',20)
xlabel('$k_m \Delta x$','Interpreter','Latex','FontSize',labelsize)
ylabel('Im$(\tilde{k}_m \Delta x)$','Interpreter','Latex','FontSize',labelsize)

set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 1200 400])
fname = 'HW3P3';
print(gcf,fname,'-djpeg90')
```

```
function [t,u,x,eigvals] = integrator(L,Nx,T,Nt,spacedef,casedef)
%
% Inputs
```

```
%
% L :: length of physical domain, x runs from 0 to L
% Nx :: number of points to use in x
% T :: length of temporal domain, t runs from 0 to T
% Nt :: number of points to use in t (for reporting solution, ode45 chooses dt internally)
% spacedef :: string to select spatial derivative operator
% casedef :: string to select governing equation
%
% Output
%
% t :: time vector
% u :: solution vector
% spatial domain
dx = L/(Nx+1);
x = dx*linspace(0,Nx,Nx+1).';
% temporal domain
dt = T/(Nt-1);
t = dt*linspace(0,Nt-1,Nt);
% spatial operator
if (strcmp(spacedef,'Central2FirstDeriv'))
  D1 = (D1_operator(Nx,dx,1,1,[-0.5 0 0.5]));
elseif (strcmp(spacedef,'Upwind1FirstDeriv'))
  D1 = (D1_operator(Nx,dx,0,1,[-1,1]));
elseif (strcmp(spacedef,'Upwind3FirstDeriv'))
  D1 = (D1_operator(Nx,dx,1,2,[1,-6,3,2]/6));
elseif (strcmp(spacedef,'Central4FirstDeriv'))
  D1 = (D1_operator(Nx,dx,2,2,[1,-8,0,8,-1]/12));
elseif (strcmp(spacedef,'Pade6FirstDeriv'))
  D1 = (D1_operatorPade(Nx,dx));
else
  error(sprintf('Unknown spacedef = %s',spacedef));
end
[~,eigvals] = eig(-D1);
eigvals     = diag(eigvals);
% initial condition
u0 = exp(-(x-0.5).^2/(2*(3/40)^2));
% run
if (casedef == 'LinearAdvection')
  [t,u] = ode45(@(t,y) LinearAdvection(t,y,D1), t, u0);
else
  error(sprintf('Unknown casedef = %s',casedef));
end
```

```
function udot = LinearAdvection(t,u,D)
  udot = -D*u;
  return
```

```
function [ D ] = D1_operator( n,dx,R,L,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+1);
% fill interior domain
for i=1+L:n+1-R
    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+2-R:n+1
    D(i,end-L-1+(i-n):end)  = a(1:L+(n+2-i));
    D(i,1:(i-n+R-1))        = a(L+(n+3-i):end);
end
D = 1/dx*D;
end
```

```
function [ D ] = D1_operatorPade( n,dx)
%computes th order Pade scheme operator for first derivative
%   n = number of grid points
%   dx = step size
%   [R,L]  = right/left bound of stencil
%   a   = stencils

a = [-1,-28,0,28,1]/36;
R = 2;
L = 2;

% initialize
RHS = zeros(n+1);

% fill interior domain
for i=1+L:n+1-R
    RHS(i,i-L:i+R) = a;
end
% fill left boundary
for i=1:L
    RHS(i,1:i+R)          = a(L-i+2:end);
    RHS(i,end-L+i:end)    = a(1:L-i+1);
end
% fill right boundary
for i=n+2-R:n+1
    RHS(i,end-L-1+(i-n):end)  = a(1:L+(n+2-i));
    RHS(i,1:(i-n+R-1))        = a(L+(n+3-i):end);
end
RHS = RHS/dx;

% assemble left matrix
a2  = [1/3,1,1/3];
R   = 1;
L   = 1;
% fill interior domain
for i=1+L:n+1-R
    LHS(i,i-L:i+R) = a2;
end

% initialize
LHS = zeros(n+1);
R   = 1;
L   = 1;

% fill interior domain
for i=1+L:n+1-R
    LHS(i,i-L:i+R) = a2;
end

% fill left boundary
for i=1:L
    LHS(i,1:i+R)          = a2(L-i+2:end);
    LHS(i,end-L+i:end)    = a2(1:L-i+1);
end
% fill right boundary
for i=n+2-R:n+1
    LHS(i,end-L-1+(i-n):end)  = a2(1:L+(n+2-i));
    LHS(i,1:(i-n+R-1))        = a2(L+(n+3-i):end);
end
```

```
D = inv(LHS)*RHS;

end
```

```
function [ k_tilde,dxkm ] = circD_wavenum( D )
%computes the wavenumber and the modified wavenumber for given
%circulant matrix D

d   = D(1,:);
n=length(D);
for k=1:n
    sum = 0;
    for j=1:n
        sum = sum + d(j)*exp(i*2*pi*(j-1)*(k-1)/n);
    end
    lambda(k)    = sum;
    dxkm(k)      = 2*pi*(k-1)/(n-1);
end

k_tilde = -i*lambda;

end
```

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