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

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Homework #1

-- SOLUTIONS --

Due date: February 8, 2024

Differential matrix operators are key components of the computer programs developed in computational physics/computational aerodynamics. In this homework, you will develop your own code for constructing such an operator, and validate your implementation using test-functions with known derivatives.

Problem 1 · In the programming language of your choice, develop a piece of code that constructs the discrete operator $\mathbf{D}$, a $N \times N$ matrix corresponding to the finite-difference approximation

$$\left.\frac{\partial u}{\partial x}\right|_i \simeq \sum_{j=-L}^{R} a_j u_{i+j} \ , \quad i \in \{1, \dots, N\}$$

on a one-dimensional domain discretized into N consecutive nodes located at $x_i = (i-1)\Delta x$ with $\Delta x = 1/N$. Moreover, this domain is assumed periodic, i.e., $u_i = u_{(i+kN)}, \forall k \in \mathbb{Z}$.

The scalars $a_j, j \in \{-L, \dots, R\}$, are the coefficients of the finite-difference scheme that uses L left neighbors and R right neighbors to approximate the first spatial derivative at the location of any given discrete node. The ith row of the matrix $\mathbf{D}$ contains the coefficients for approximating the first spatial derivative at the location of the ith discrete node.

Your code/function should receive the following inputs:

- The positive integers L, R, and N (you may verify that $L + R + 1 \leq N$).
- The vector $\mathbf{a} = \begin{bmatrix} a_{-L} & a_{-L+1} & \cdots & a_R \end{bmatrix}^\intercal$ of length $L + R + 1$.

It should return:

- The matrix $\mathbf{D}$ of size $N \times N$.

Important remark: The discrete notations introduced in this problem correspond to a 1-based array indexing (i.e., all arrays start with the index 1). Depending on your choice of programming language, you may have to convert them into 0-based array indexing (i.e., all arrays start with the index 0). For reference, `Matlab` and `Fortan` use 1-based array indexing, whereas `Python` and `C/C++` use 0-based array indexing.

Solution:

Python (0-based indexing):

```python
# Required modules
import numpy as np
from scipy.linalg import circulant

```

```python
## OPTION 1
def D1_operator(N,L,R,a):
    # Initialize matrix with zeros
    D = np.zeros((N,N))
    # Fill left boundary
    for i in range(L):
        D[i][0:(i+R+1)] = a[(L-i):]; D[i][N-(L-i):] = a[0:(L-i)]
    # Fill interior
    for i in range(L,N-R):
        D[i][(i-L):(i+R+1)] = a
    # Fill right boundary
    for i in range(N-R,N):
        D[i][i-L:] = a[0:(N+L-i)]; D[i][0:(R-(N-i)+1)] = a[(N+L-i):]
    # Return matrix
    return D

## OPTION 2
def D1_operator(N,L,R,a):
    # Initialize first row with zeros
    first_row = np.zeros(N)
    # Place coefficients "a" in first row
    first_row[0:R+1] = a[L:]
    first_row[N-L:] = a[0:L]
    # Return circulant matrix for this first row
    return np.array(circulant(first_row)).transpose()

## OPTION 3
def D1_operator(N,L,R,a):
    # Initialize first row with zeros
    first_row = np.zeros(N)
    # First L+R+1 entries are initialize with a
    first_row[0:L+R+1] = a
    # Entries are shifted left by L columns
    first_row = np.roll(first_row,-L)
    # Return circulant matrix for this first row
    return np.array(circulant(first_row)).transpose()
```

Matlab (1-based indexing):

```matlab
function [ D ] = D1_operator( n,R,L,a )
% computes a finite difference operator
%       n = number of grid points
%   [R,L] = right/left bound of stencil
%       a = finite-difference coefficients

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

Problem 2 · In order to verify the code developed in the previous problem, we can test **D** against known data of the form

$$\mathbf{f} = \begin{bmatrix} f(x_1) & f(x_2) & \cdots & f(x_N) \end{bmatrix}^\intercal$$

where $f(x)$ is a function that is (at least) three times differentiable and whose derivatives can be calculated analytically. If **d** is the vector of the exact derivatives of $f(x)$ at the discrete points, i.e.,

$$\mathbf{d} = \begin{bmatrix} f'(x_1) & f'(x_2) & \cdots & f'(x_N) \end{bmatrix}^\intercal$$

then the vector containing the errors between the exact and numerically estimated derivatives of $f(x)$ is given as

$$\boldsymbol{\epsilon} = \mathbf{D}\mathbf{f} - \mathbf{d}$$

The discrete ℓ_∞ and ℓ_2 norms of $\boldsymbol{\epsilon}$ are often used to study the accuracy of a numerical scheme. They are given as

$$\|\boldsymbol{\epsilon}\|_\infty = \max_i |\varepsilon_i|$$

$$\|\boldsymbol{\epsilon}\|_2 = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2}$$

Both $\|\boldsymbol{\epsilon}\|_\infty$ and $\|\boldsymbol{\epsilon}\|_2$ are scalar quantities, which depend on the type of finite-difference scheme used to construct **D**, as well as on $\Delta x = 1/N$.

(a) Choose a function $f(x)$ that is well suited for the testing of your code. Justify this choice.

Solution: In order to analyze the differentiation schemes at hand, we make use of the function

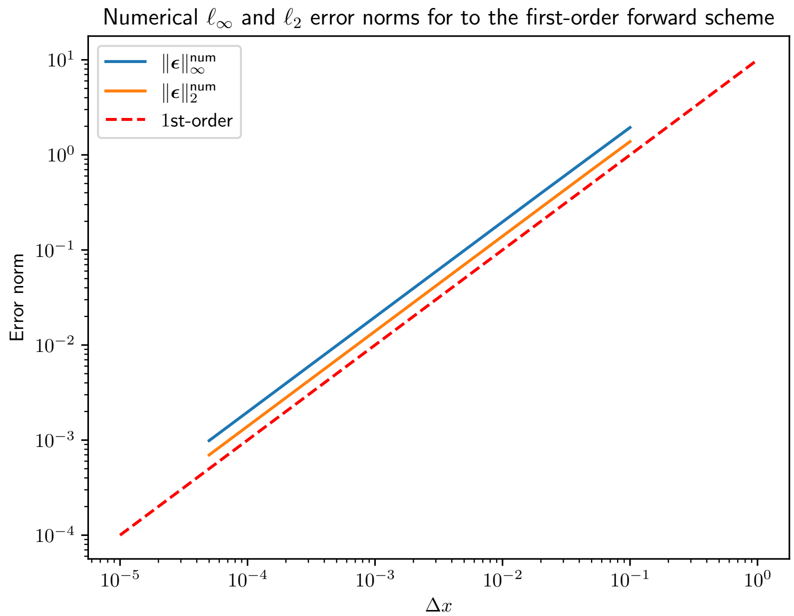
$$f(x) = \sin(2\pi x)$$

that is periodic with period $L = 1$ to analytically compute error norms and thus being able to verify our code and check for numerical convergence. This functions is infinitely differentiable, which will allow us to properly evaluate the error terms.

(b) Using a log-log scale, plot the ℓ_∞ and ℓ_2 errors norms corresponding to the forward difference scheme with coefficients

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \frac{1}{\Delta x}\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

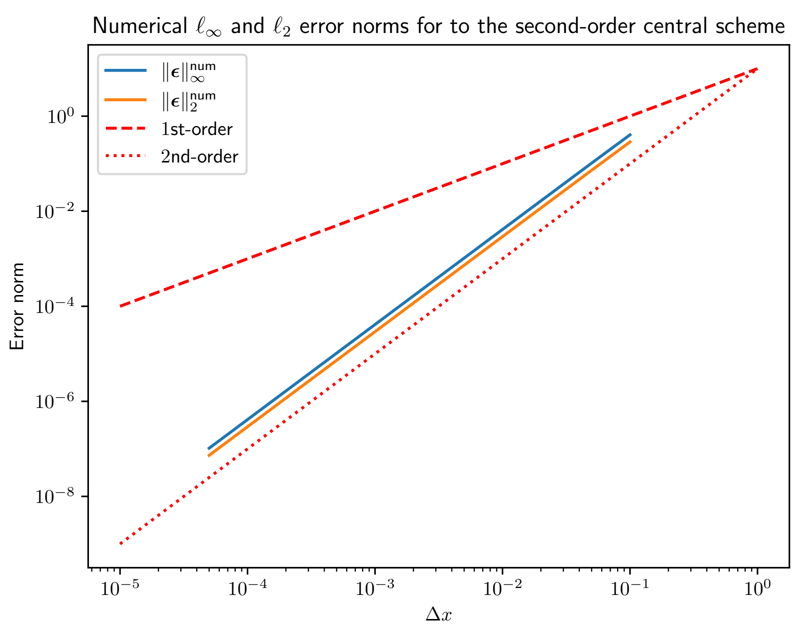
Solution: As can be seen in this figure, the finite difference scheme shows the expected convergence rate (first order) for the infinity norm and the 2-norm:



(c) Using a log-log scale, plot the ℓ_∞ and ℓ_2 errors norms corresponding to the central difference scheme with coefficients

$$\mathbf{a} = \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \end{bmatrix} = \frac{1}{2\Delta x}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Solution: As can be seen in this figure, the finite difference scheme shows the expected convergence rate (second order) for the infinity norm and the 2-norm:



(d) Verify that the slope of these plotted error norms matches their expected value.

Solution: Please see above figures for the convergence error with the appropriate first and second order lines that are perfectly parallel.

Problem 3 · **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.**

Using the Taylor series expansion of $f(x)$, calculate the analytically expected value of the discrete errors norms computed in Questions (b) and (c) of the previous problem.

Plot these analytically expected value alongside the corresponding previously computed errors norms.

Hint: You will need to use the midpoint rule

$$\int_0^1 g(x)\ \mathrm{d}x \simeq \sum_{i=1}^{N} g(x_i)\Delta x$$

Solution:
For the given stencil $[\tilde{a}_0, \tilde{a}_1]$, we can write the Taylor Series Expansion about x_i as

$$f_i = f_i$$
$$f_{i+1} = f_i + \Delta x \frac{\partial f}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 f}{\partial x^2} + \text{h.o.t.}$$

With the given stencils, we know that the exact derivative follows

$$\frac{\partial f}{\partial x} = \frac{u_{i+1} - u_i}{\Delta x} + \frac{\Delta x}{2}\frac{\partial^2 f}{\partial x^2} + \text{h.o.t.} \tag{1}$$

and thus the leading truncation error term is

$$\varepsilon = \frac{\Delta x}{2}\frac{\partial^2 f}{\partial x^2} \tag{2}$$

which, for given test function $f = \sin(x)$, is

$$\varepsilon = -(2\pi)^2 \frac{\Delta x}{2} \sin(2\pi x) \tag{3}$$

For the ℓ_∞ norm of the error, we compute

$$\|\boldsymbol{\epsilon}\|_\infty = \max_i |\varepsilon_i| = \max\left((2\pi)^2\frac{\Delta x}{2}\sin(2\pi x)\right) = (2\pi)^2\frac{\Delta x}{2} = 2\pi^2 \Delta x \tag{4}$$

For the ℓ_2 norm, we compute

$$\|\boldsymbol{\epsilon}\|_2 = \sqrt{\Delta x \sum_i \varepsilon_i^2} = \frac{\Delta x}{2}\sqrt{\sum_i \left(\frac{\partial^2 f}{\partial x^2}\right)^2 \Delta x} = \frac{\Delta x}{2}\sqrt{\int \left(\frac{\partial^2 f}{\partial x^2}\right)^2 dx}$$
$$= \sqrt{2}\pi^2 \Delta x$$

where the step from summation of discrete values to the integral might involve another error due to approximation of the integrals by an finite summation. It can be shown that for this problem the error involved is negligible and effectively independent of the step size Δx. Thus, we can conclude to expect the ℓ_∞ and ℓ_2 norms of the error to decrease by order one.

For the central finite difference scheme, the derivation follows the same scheme as above and we gain the analytical error (leading term) as

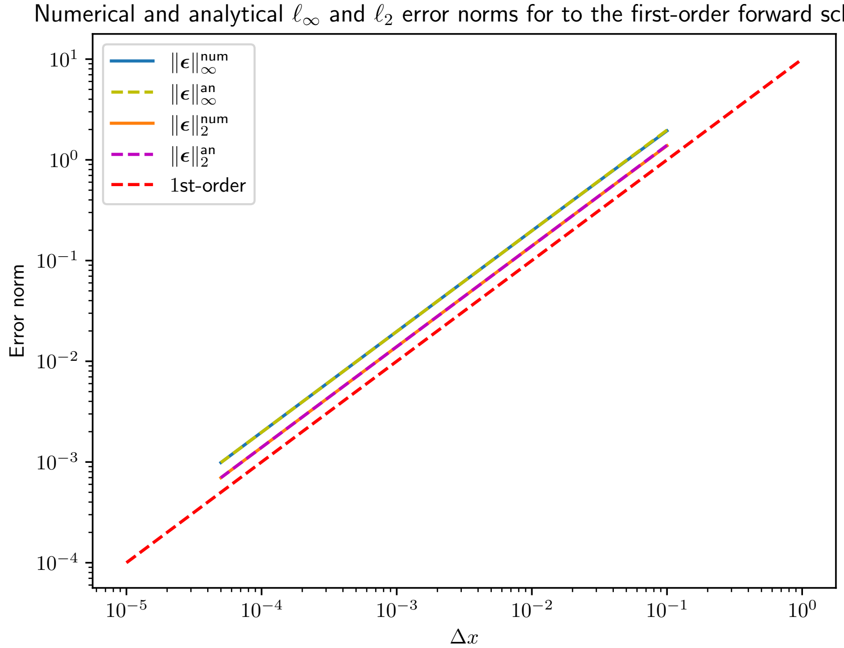
$$\varepsilon = \frac{\Delta x^2}{3!}\frac{\partial^3 f}{\partial x^3} \tag{5}$$

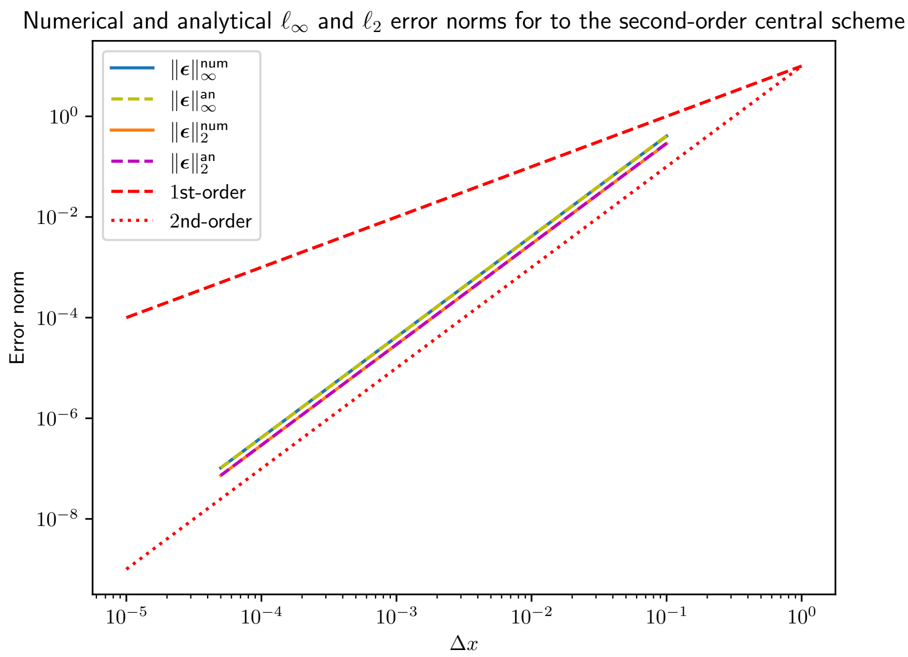
Thus, we get the ℓ_∞ and ℓ_2 norms following

$$\|\boldsymbol{\epsilon}\|_\infty = \frac{4\pi^3}{3}(\Delta x)^2$$
$$\|\boldsymbol{\epsilon}\|_2 = \frac{2\sqrt{2}\pi^3}{3}(\Delta x)^2$$

and thus have a second order convergence for the ℓ_∞ and ℓ_2 norms.

Numerical and analytical ℓ_∞ and ℓ_2 error norms for to the first-order forward scheme





Solution: Full codes:

Python (0-based indexing)

```python
# %% [markdown]
# Import `numpy`, `matplotlib`, and `math`

# %%
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from math import pi, sqrt
# The following parameters are changed to make our figures prettier!
plt.rc('text', usetex=True)
plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{amssymb}')
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300

# %% [markdown]
# The function `D1_operator` takes `L`, `R`, `N`, and the vector `a` as inputs and returns the matrix operator `D`
#

# %%
def D1_operator(N,L,R,a):
    # Initialize matrix with zeros
    D = np.zeros((N,N))
    # Fill left boundary
    for i in range(L):
        D[i][0:(i+R+1)] = a[(L-i):]
        D[i][N-(L-i):] = a[0:(L-i)]
    # Fill interior
    for i in range(L,N-R):
        D[i][(i-L):(i+R+1)] = a
```

```
    # Fill right boundary
    for i in range(N-R,N):
        D[i][i-L:] = a[0:(N+L-i)]
        D[i][0:(R-(N-i)+1)] = a[(N+L-i):]
    # Return matrix
    return D

# %% [markdown]
# We choose the test-function
# $$ f(x) = \sin(2\pi x) $$
# and apply our discrete matrix operator to it

# %%
stencil_case = 'second-order central' # options: 'first-order forward'; 'second-order
    central'
Ns  = np.array([10,20,50,100,200,500,1000,2000,5000,10000,20000])
dxs = []; err_an_max =[]; err_an_rms = []; err_num_max =[]; err_num_rms = [];
# Loop over all grid sizes
for N in Ns:
    # Define discrete node locations and store mesh-spacing
    x  = np.linspace(0, 1, N, endpoint=False); dx = x[1]-x[0]; dxs.append(dx)
    # Initialize test-function at the node locations
    f  = np.sin(2*pi*x)
    # Compute analytical derivatives of test-function at the node locations
    fx = 2*pi*np.cos(2*pi*x)
    # Initialize finite-difference coefficients
    match stencil_case:
        case 'first-order forward':
            L = 0; R = 1; a = [-1,1]/dx
        case 'second-order central':
            L = 1; R = 1; a = [-1,0,1]/(2*dx)
    # Construct matrix operator
    D = D1_operator(N,L,R,a)
    # Compute approximate derivative by multiplying D with the vector f
    fx_approx = (D @ f)
    # Compute and store analytical error norms
    match stencil_case:
        case 'first-order forward':
            err_an = -dx*(2*pi)**2/2*np.sin(2*pi*x)
            err_an_max.append(np.max(np.abs(err_an)))
            err_an_rms.append(sqrt(np.sum(err_an**2)/N))
        case 'second-order central':
            err_an = -dx**2*(2*pi)**3/4*np.cos(2*pi*x);
            err_an_max.append(np.max(np.abs(err_an)))
            err_an_rms.append(sqrt(np.sum(err_an**2)/N))
    # Compute and store numerical error norms
    err_num = fx - fx_approx
    err_num_max.append(np.max(np.abs(err_num)))
    err_num_rms.append(sqrt(np.sum(err_num**2)/N))

# Plot numerical error norms
plt.title(r'Numerical $\ell_\infty$ and $\ell_2$ error norms for to the %s scheme' %
    stencil_case)
plt.xlabel(r'$\Delta x$');plt.ylabel(r'Error norm')
plt.loglog(dxs,err_num_max,label=r'$\|\boldsymbol{\epsilon}\|_\infty^\text{num}$')
plt.loglog(dxs,err_num_rms,label=r'$\|\boldsymbol{\epsilon}\|_2^\text{num}$')
plt.loglog([1e-5,1e0],[1e-4,1e1],'r--',label=r'$1$st-order')
if stencil_case == 'second-order central':
    plt.loglog([1e-5,1e0],[1e-9,1e1],'r:',label=r'$2$nd-order')
plt.legend()
match stencil_case:
    case 'first-order forward':  plt.savefig('error-norm-num-1st-order-FD.pdf')
    case 'second-order central': plt.savefig('error-norm-num-2nd-order-CD.pdf')
plt.show()

# %%
```

```python
# Plot numerical and analytical error norms
plt.title(r'Numerical and analytical $\ell_\infty$ and $\ell_2$ error norms for to the %s
    scheme' % stencil_case)
plt.xlabel(r'$\Delta x$');plt.ylabel(r'Error norm')
plt.loglog(dxs,err_num_max,label=r'$\|\boldsymbol{\epsilon}\|_\infty^\text{num}$')
plt.loglog(dxs,err_an_max,'g--',label=r'$\|\boldsymbol{\epsilon}\|_\infty^\text{an}$')
plt.loglog(dxs,err_num_rms,label=r'$\|\boldsymbol{\epsilon}\|_2^\text{num}$')
plt.loglog(dxs,err_an_rms,'m--',label=r'$\|\boldsymbol{\epsilon}\|_2^\text{an}$')
plt.loglog([1e-5,1e0],[1e-4,1e1],'r--',label=r'$1$st-order')
if stencil_case == 'second-order central':
    plt.loglog([1e-5,1e0],[1e-9,1e1],'r:',label=r'$2$nd-order')
plt.legend()
match stencil_case:
    case 'first-order forward':  plt.savefig('error-norm-num+an-1st-order-FD.pdf')
    case 'second-order central': plt.savefig('error-norm-num+an-2nd-order-CD.pdf')
plt.show()
```

Matlab (1-based indexing)

```matlab
clear all; close all; clc
stencil_case    = 2;
ns              = [10,20,50,100,200,500,1000,2000,5000,10000,20000];
%% iterate over all grids
for i=1:length(ns)
    n   = ns(i);
    % define function's and its derivative's values
    x  = linspace(0,1,n+1); x = x(1:end-1);
    y  = sin(2*pi*x)';
    dy = 2*pi*cos(2*pi*x)';
    dx = x(2)-x(1);
    dxs(i) = dx;
    switch stencil_case
        case 1
            L = 0;
            R = 1;
            a = [-1,1]/dx;
        case 2
            L = 1;
            R = 1;
            a = [-1/2,0,1/2]/dx;
    end
    % compute derivative operator
    [ D ]   = D1_operator( n,R,L,a );
    % compute derivative
    Dy      = D*y;
    % compute error norms
     switch stencil_case
        case 1
            err_an          = -(2.0*pi)^2*dx/2*sin(2.0*pi*x);
            err_an_inf(i)   = max(abs(err_an));
            err_an_L2(i)    = sqrt(sum(err_an.^2)/n);

         case 2
            err_an          = -(2.0*pi)^3*dx^2/4*cos(2.0*pi*x);
            err_an_inf(i)   = max(abs(err_an));
            err_an_L2(i)    = sqrt(sum(err_an.^2)/n);
     end
    err        = dy-Dy;
    err_inf(i) = max(abs(err));
    err_L2(i)  = sqrt(sum((dy-Dy).^2)/n);
    xs{i}      = x;
end
%% plot results
figure(1), clf;
```

```
loglog(dxs,err_inf,'k-','LineWidth',3); hold on
loglog(dxs,err_L2,'LineStyle','-','Color','b','LineWidth',3)
plot([1e-5,1e0],[1e-4,1e1],'r--','LineWidth',2)
if(stencil_case==2)
    plot([1e-5,1e0],[1e-9,1e1],'r:','LineWidth',2)
end
leg=legend('$\vert \vert \epsilon \vert \vert_{\infty}^{num}$',...
           '$\vert \vert \epsilon \vert \vert_{2}^{num}$',...
           '$1$st-order','$2$nd-order');
set(gca,'FontSize',20)
xlabel('$\Delta x$','Interpreter','Latex','FontSize',25)
ylabel('Error norm','Interpreter','Latex','FontSize',25)
set(leg,'Interpreter','Latex','FontSize',20,'Location','SouthEast')
set(gcf,'Position',[100 400 400 400])
set(gcf,'PaperPositionMode','Auto')
fname = ['plot_P2_case',num2str(stencil_case)];
print(gcf,[fname,'.jpg'],'-djpeg90')

figure(2), clf;
loglog(dxs,err_inf,'k-','LineWidth',3); hold on
loglog(dxs,err_an_inf,'LineStyle','--','Color','g','LineWidth',3);
loglog(dxs,err_L2,'LineStyle','-','Color','b','LineWidth',3)
loglog(dxs,err_an_L2,'LineStyle','--','Color','c','LineWidth',3)
plot([1e-5,1e0],[1e-4,1e1],'r--','LineWidth',2)
if(stencil_case==2)
    plot([1e-5,1e0],[1e-9,1e1],'r:','LineWidth',2)
end
leg=legend('$\vert \vert \epsilon \vert \vert_{\infty}^{num}$',...
           '$\vert \vert \epsilon \vert \vert_{\infty}^{an}$',...
           '$\vert \vert \epsilon \vert \vert_{2}^{num}$',...
           '$\vert \vert \epsilon \vert \vert_{2}^{an}$',...
           '$1$st-order','$2$nd-order');
set(gca,'FontSize',20)
xlabel('$\Delta x$','Interpreter','Latex','FontSize',25)
ylabel('Error norm','Interpreter','Latex','FontSize',25)
set(leg,'Interpreter','Latex','FontSize',20,'Location','SouthEast')
set(gcf,'Position',[100 400 400 400])
set(gcf,'PaperPositionMode','Auto')
fname = ['plot_P3_case',num2str(stencil_case)];
print(gcf,[fname,'.jpg'],'-djpeg90')
```

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