

CMSE 821: HOMEWORK 1
FALL 2025

PAGE LIMIT: 15 pages (single-sided).

NOTE: Please include a cover page – this will not count toward the 15 page limit.

NOTE: Don't forget to include **Python** code where appropriate (numpy, scipy, matplotlib, sympy) – this does count toward your 15 pages.

AI Collaboration Policy (Read First)

You are encouraged to use **ChatGPT5** to brainstorm and draft. However:

- **Verification is mandatory.** Every AI-produced formula or code snippet must be validated with sympy (symbolic checks), unit tests, and numerical experiments.
- **Provenance is required.** Include an appendix with (i) your exact prompts; (ii) model name (“GPT-5 Thinking”); (iii) date/time; (iv) a brief note on what you accepted or rejected and why.
- **Dual-sourcing.** For core derivations, obtain *two* distinct AI approaches (e.g., Taylor vs. moments/Vandermonde) and reconcile them, or explain the discrepancy.
- **Authorship.** Your final math, proofs, and commentary must be in your own words. Cite AI assistance where used.

Submission. One PDF (derivations, figures, **AI Appendix**) plus a repo/zip with runnable .py or .ipynb. Use sympy, numpy, matplotlib, scipy. *No Matlab.*

Code Documentation Requirements (applies to *all* code)

- **Every function/method** must begin with the header block below. Use clear type hints, document units and shapes, and list all dependencies.
- Keep subroutines short (≤ 1 page each). Split long logic into helpers.
- Comment *why* each nontrivial step is done (not just what).
- Provide a minimal usage example or unit test for each public API.

Header comment block template (paste into each subroutine)

```
1 """
2 Name:          <function_or_method_name>
3 Purpose:       <what this subroutine computes and why>
4 Author:        <Your Name>
5 Date written:  YYYY-MM-DD
6 Last modified: YYYY-MM-DD
7 Inputs:
8   - <arg1>: <type> ... <meaning/units/shape>
9   - <arg2>: <type> ... <meaning/units/shape>
10 Outputs:
```

```

11 - <ret>: <type> ... <meaning/units/shape>
12 Dependencies:
13 - <SubroutineA>, <SubroutineB>, ... (list all helpers called here)
14 """

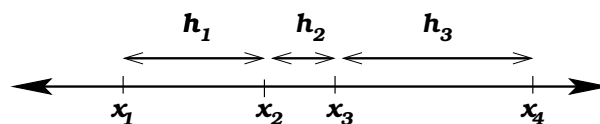
```

Part 1: AI-Assisted Symbolic Manipulation in Python

- Taylor series \rightarrow `lambdify` \rightarrow plots (complex-valued).** Let $f(x) = \frac{e^{(1+i)x}}{1+x^2}$ with $i^2 = -1$.
 - Compute Maclaurin polynomials of degrees 4, 8, 12 in `sympy`; strip $O(\cdot)$; simplify; `lambdify` (complex dtype).
 - AI-Assist:** ask for two derivations: (i) direct quotient expansion; (ii) product $e^{(1+i)x}(1+x^2)^{-1}$. Require explicit coefficients to x^{12} and stated assumptions.
 - Verify:** symbolically subtract AI series from your `sympy` series; show zero up to requested order.
 - Plot real/imag parts and error on $[-2, 2]$; report ℓ_∞ error for each degree.
- Method of moments: 5-point $f'(0)$ with offsets $s = \{-3, -2, -1, 0, 1\}$.**
 - Build the Vandermonde system for scaled weights $c_j = h d_j$ with moments $\sum c_j s_j^k = 0$ for $k = 0, 2, 3, 4$, and $= 1!$ for $k = 1$; solve in `sympy` for exact rationals.
 - AI-Assist:** ask for weights with pasted moment checks; insist on rationals.
 - Programmatically verify all moments; prove order 4 by identifying the first nonzero moment.
 - Numerical check with $f(x) = \sin x$: grid refinements $h = 2^{-k}$, $k = 3, \dots, 8$; report max error and observed order.
- One-sided 6-point f'' at a right boundary (offsets $s = \{-5, -4, -3, -2, -1, 0\}$).**
 - Set up and solve the 6×6 moment system for $c_j = h^2 d_j$ (zero moments for $k = 0, 1, 3, 4, 5$; $= 2!$ at $k = 2$).
 - AI-Assist:** request (i) rational weights; (ii) floating-point weights (12 sig. figs). Compare numerical stability on coarse vs. fine h .
 - Verify moment conditions symbolically; state the formal order $q = N - m$.
 - Test with $f(x) = e^x$ near a right boundary; confirm order.

Part 2: Non-Uniform Grid

Consider the non-uniform grid:



- Derive a finite difference approximation to $u''(x_2)$ that is accurate as possible for smooth functions $u(x)$, based on the four values $U_1 = u(x_1), \dots, U_4 = u(x_4)$. Give an expression for the dominant term in the error.

In the next two questions, you will try to determine an order of accuracy for your method:

5. To get a better feel for how the error behaves as the grid gets finer, take 500 values of H , where H spans 2 or 3 orders of magnitude, and for each value of H , randomly generate three numbers, h_1 , h_2 , and h_3 , where each $h_i \in [0, H]$. For each value H , compute your approximation to $u''(x_2)$ using the randomly generated $h_i \in [0, H]$. Plot the error against H on a log-log plot to get a scatter plot of the behavior as $H \rightarrow 0$. **NOTE:** in **Python** the commands

```
rng = numpy.random.default_rng(); x = rng.uniform(0.0, H)
```

produce a single random number x in the range $[0, H]$.

6. Estimate the order of accuracy by doing a least squares fit of the form

$$\log(E(H)) = K + p \log(H)$$

to determine K and p based on the 500 data points. Recall that this can be done by solving the following linear system in the least squares sense:

$$\begin{bmatrix} 1 & \log(H_1) \\ 1 & \log(H_2) \\ \vdots & \vdots \\ 1 & \log(H_{500}) \end{bmatrix} \begin{bmatrix} K \\ p \end{bmatrix} = \begin{bmatrix} \log(E(H_1)) \\ \log(E(H_2)) \\ \vdots \\ \log(E(H_{500})) \end{bmatrix}.$$

NOTE: “In the least-squares sense” means that one should solve the rectangular system $Ax = b$, by solving the (square) normal equation: $A^T Ax = A^T b$. In **Python**, you may also use `numpy.linalg.lstsq(A, b, rcond=None)`.

Part 3: Mixed Boundary Conditions

Consider the following 2-point BVP:

$$\begin{aligned} u'' + u &= f(x), & \text{on } 0 \leq x \leq 10 \\ u'(0) - u(0) &= 0, & u'(10) + u(10) = 0. \end{aligned}$$

7. Construct a second-order accurate finite-difference method for this BVP. Write your method as a linear system of the form $A\vec{u} = \vec{f}$.
8. Show that this method is L_∞ -stable, by constructing an approximation to A^{-1} using a Green's function. **HINT:** You will need to construct the Green's function of the operator $\mathcal{L}(u) = u'' + u$ with the given BCs. Use this Green's function to obtain an approximation to A^{-1} . Finally, show that this approximated A^{-1} is bounded in the max-norm as $h \rightarrow 0$.
9. Construct the exact solution to this BVP with $f(x) = -e^x$.
10. Verify that your method is second order accurate by solving the BVP with $f(x) = -e^x$ at four different grid spacings h .

HINT 1: Use **Python** sparse matrices to create A – this will save storage and allow a fast solver. Modify the following commands, which generate a tri-diagonal matrix with a $[1, -2, 1]$ structure, to model your specific BVP:

```
import numpy as np
from scipy.sparse import diags
e = np.ones(n)
A = diags([e, -2*e, e], offsets=[-1,0,1], shape=(n,n))
```

HINT 2: Solve your linear system with a sparse direct solver in **Python**:

```
from scipy.sparse.linalg import spsolve; u = spsolve(A, f)
```

(This is the analogue of Matlab's $u = A \backslash f$.)

Part 4: Variable diffusivity

11. Show that the matrix A appearing in (2.50) of LeVeque's notes is negative definite provided that $\kappa > 0$ everywhere (a physically reasonable assumption). Recall that a matrix is negative definite if $U^T A U < 0$ for all vectors U that are not identically zero.
12. Show that (2.50) of LeVeque's notes satisfies a maximum principle in the homogenous case (i.e., $f(x) \equiv 0$). Recall that the maximum principle for the continuous problem states that

$$\begin{aligned} &\text{if } (\kappa u')' = 0 \text{ on } [a, b], \\ &\text{then } \min\{u(a), u(b)\} \leq u(x) \leq \max\{u(a), u(b)\}, \quad x \in (a, b). \end{aligned}$$

Part 5: Convergence verification

Assemble with `scipy.sparse.diags` and solve with `scipy.sparse.linalg.spsolve`. Confirm 2nd order in $\|\cdot\|_\infty$ for $N = 50, 100, 200, 400$.

13. **(Mixed BC 2D Poisson, 4th order) Interior & boundary closures, assembly, and solve.** On $\Omega = [0, 1] \times [0, 1]$, solve

$$-\Delta u = f(x, y), \quad f(x, y) = \begin{cases} 1, & x \in [1/3, 1/2], y \in [1/2, 2/3], \\ 0, & \text{otherwise,} \end{cases}$$

with homogenous *Dirichlet* BC on three sides and a homogenous *Neumann* BC on the remaining side (state clearly which side and the boundary data you choose).

- a) *Interior:* Derive and state the classic 4th-order 9-point Laplacian on a uniform grid ($h_x = h_y = h$):

$$\Delta_h^{(4)} u_{i,j} = \frac{-u_{i-1,j-1} - u_{i-1,j+1} - u_{i+1,j-1} - u_{i+1,j+1} + 16(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) - 60u_{i,j}}{12h^2},$$

and show it is $\mathcal{O}(h^4)$ by Taylor expansion.

- b) *Dirichlet sides:* State how to enforce given $u = g$ at boundary nodes and how these values modify the RHS adjacent to the boundary for a 4th-order scheme (i.e., incorporate known values from the 9-point stencil into b).
- c) *Neumann side:* Derive a 4th-order one-sided normal derivative to eliminate ghost points. For example, at a right boundary $x = 1$ with $u_x(1, y) = q(y)$,

$$u_x(x_N, y_j) \approx \frac{25u_{N,j} - 48u_{N-1,j} + 36u_{N-2,j} - 16u_{N-3,j} + 3u_{N-4,j}}{12h} = q(y_j),$$

and similarly for other sides. Show how to use these relations to express the needed ghost values (including diagonal ones for the 9-point stencil) in terms of interior/boundary unknowns and data, maintaining overall 4th order.

- d) **(Python)** *Assembly*: Build the sparse matrix A and vector b for the full grid using `scipy.sparse` (`diags`, `kron`, or explicit COO). Carefully handle rows adjacent to boundaries with the derived closures.
- e) **(Python)** *Right-hand side*: Discretize f by nodal sampling: set $f_{i,j} = 1$ if $(x_i, y_j) \in [1/3, 1/2] \times [1/2, 2/3]$, else 0. (Optionally, use cell-centered quadrature and map to nodes; document your choice.)
- f) **(Python)** *Solve*: Use `scipy.sparse.linalg.spsolve(A, b)` (the Python analogue of Matlab's $A \setminus b$) to obtain u . Visualize with `matplotlib` (`imshow` or 3D surface).
- g) **(Optional)** If you can construct an exact solution (e.g., by manufactured u and corresponding f & BCs), perform a grid refinement to confirm 4th order. Otherwise, at least confirm that changing only the Neumann-side closure from 2nd to 4th order noticeably reduces the observed error on a smooth manufactured problem.