GROUP PROJECT 2: GAUSSIAN ELIMINATION

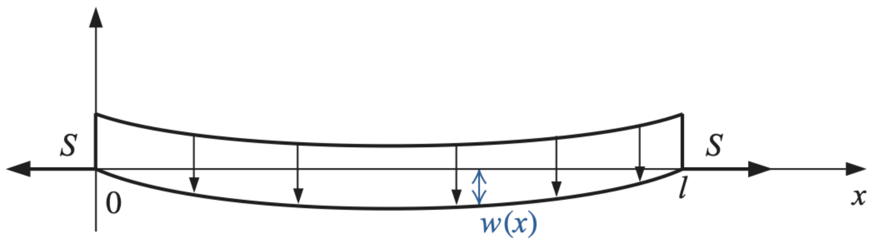
Project Rules

- Your class will be split into small groups (3-4 students). Please remember your group number (e.g. group 3A). Assign a scribe and a coordinator.
- **The scribe** should create a Google Docs file and share it with all group members and your instructors: Dr. Pencheva (gergina@utexas.edu) and Eboni (eboniwilliams@utexas.edu). Please share with us by adding us as "commentors" and restrict access so that only those people with access can open the link. The name of the file should be "GroupProject2 Group<your group number>" (for the example above, **"GroupProject2 Group3A"**). Please **do NOT change the file name.**
- The scribe should also reply to the Ed Discussion **post** under **category 'Group Projects'**, **subcategory 'GP2'**. The title of the post is "Section <your section number> Group Project 2 Groups". In the post include the names of all group members, your group number, and a link to the shared Google Docs file. Please **double-check that the GoogleDoc access has been restricted**.
- **The coordinator** will be responsible for collecting contact information from the group members and setting place/time for communication (GroupMe, Discord, Slack, Zoom, etc.) The coordinator should contact Dr. Pencheva in the case of unresponsive team member.
- The group project will be only **a part of one class period**. You are **not expected to finish the entire project during class**. In fact, group collaboration outside is strongly encouraged and even necessary. **I want to see evidence of group collaboration by the end of Saturday, March 9. Listing names and contact information only is NOT sufficient.** All team members might not get the same score. A component of the score will be determined based on the individual contribution.
- A list with tasks and questions is positioned in one place, at the bottom of the document. You are encouraged to provide only your answers at the top of your Google Docs file and keep the rest of the discussion separated. This will help for a quicker grading.
- Upload on Canvas the final version of your project report. **One submission per group is sufficient.** The Canvas assignment will be setup such that it shows submission for everyone in the group so long as one person in the group submits.

Project Description

We will experiment with solving linear systems using Gaussian Elimination (PA=LU) in a more realistic application and will observe some of the problems that might come with it.

Application A common problem in civil engineering concerns the deflection of a beam of rectangular cross-section subject to uniform loading while the ends of the beam are supported so that they undergo no deflection, like in a bridge.



Suppose that L, q, E, S, and I represent, respectively, the length of the beam, the intensity of the uniform load, the modulus of elasticity, the stress at the endpoints, and the central moment of inertia. The differential equation approximating the physical situation is of the form

$$\frac{d^2w}{dx^2}(x) = \frac{S}{EI}w(x) + \frac{qx}{2EI}(x - L) \tag{1}$$

where $w(x)$ is the deflection a distance x from the left end of the beam. Since no deflection occurs at the ends of the beam, there are two boundary conditions

$$w(0) = 0 \text{ and } w(L) = 0. \tag{2}$$

When the beam is of uniform thickness, the product EI is constant. In this case, the exact solution is easily obtained. When the thickness is not uniform, the moment of inertia I is a function of x, and approximation techniques are required.

The above system (1)-(2) is an example of a second-order linear boundary value problem (BVP). Discretization converts the differential equation model into a system of linear equations. The smaller the discretization size, the larger is the resulting system of equations. Here we will consider one of many discretization methods, namely the finite difference method. Techniques for discretizing derivatives are found in Section 5.1 of Sauer, where it will be shown that a reasonable approximation for the second derivative is

$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \tag{3}$$

for a small increment h. This formula is known as three-point centered Finite Difference for the second derivative. The discretization error of this approximation is proportional to h^2 (see equation (5.8) in Sauer). Our strategy will be to consider the beam as the union of many segments of length h, and to apply the discretized version of the differential equation on each segment.

For a positive integer n, set $h = L/n$. Consider the evenly spaced grid $0 = x_0 < x_1 < \cdots < x_n < x_n = L$, where $h = x_i - x_{i-1}$ for $i = 1, \ldots, n$. Replacing the differential equation (1) with the difference approximation (3) to get the system of linear equations for the displacements $w_i = w(x_i)$, $i = 0, \dots, n$ yields

$$\begin{aligned} &\frac{w_{i-1} - 2w_i + w_{i+1}}{h^2} = \frac{S}{EI} w_i + \frac{q x_i}{2EI}(x_i - L), \quad i = 1, \dots, n-1 \\ &w_0 = w_n = 0 \end{aligned} \tag{4}$$

Eliminating w_0 and w_n and some rearrangements lead to the linear system $A\mathbf{w} = \mathbf{b}$, where A is the $(n-1) \times (n-1)$ tridiagonal matrix

$$A = \begin{bmatrix} 2 + \frac{S}{EI}h^2 & -1 & & & \\ -1 & 2 + \frac{S}{EI}h^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 + \frac{S}{EI}h^2 & -1 \\ & & & -1 & 2 + \frac{S}{EI}h^2 \end{bmatrix}, \tag{5}$$

$\mathbf{b}$ is the $(n-1)$-vector

$$\mathbf{b} = \frac{q}{2EI} h^2 \begin{bmatrix} x_1(L - x_1) \\ x_2(L - x_2) \\ \vdots \\ x_{n-1}(L - x_{n-1}) \end{bmatrix}, \tag{6}$$

and $\mathbf{w}$ is the $(n-1)$-vector of unknowns

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-1} \end{bmatrix}.$$

Finally, we are ready to model the "clamped-clamped" beam (the ends of the beam are supported). Suppose the beam is a W10-type steel I-beam with the following characteristics: length $L = 120$ in, intensity of uniform load $q = 100$ lb/ft (i.e. $q = 100/12$ **lb/in**), modulus of elasticity $E = 3.0 \times 10^7$ lb/in^2, stress at ends $S = 1000$ lb, and central moment of inertia $I = 625$ in^4.

The boundary value problem (1)-(2) actually has a known analytical solution:

$$w(x) = c_1 e^{ax} + c_2 e^{-ax} + bx(x - L) + c, \tag{7}$$

where $c = -\dfrac{qEI}{S^2}$, $a = \sqrt{\dfrac{S}{EI}}$, $b = -\dfrac{q}{2S}$, $c_1 = c\,\dfrac{(1 - e^{-aL})}{(e^{-aL} - e^{aL})}$, and $c_2 = c\,\dfrac{(e^{aL} - 1)}{(e^{-aL} - e^{aL})}$.

Goals

1. Compare the approximate (finite difference) solution with the true solution when we increase the number of segments n.

2. Study the error at the middle of the beam and verify that it is proportional to h^2, as claimed above.

3. Verify that the condition number of the matrix A in (5) is proportional to h^{-2}.

To achieve our goals, we will run several levels of refinement with number of subintervals/segment $n = 2^{k+1}$, for $k = 1, \ldots, 20$ with length $h = L/n$. For each level, you will do the following:

1. Generate the matrix A and the vector $\mathbf{b}$ in (5).

2. Solve the system using Gaussian Elimination (PA=LU).

3. Calculate and store the error (true - approximate solution) at the middle of the beam.

4. For the two coarsest levels ($k = 1$ and $k = 2$ **only**, plot on **the same grid** the **solution** of the linear system and the **true solution**. (one figure per level)

5. For **all levels** (values of k), plot the **error in the solution** along the beam (x axis).

6. Calculate and store the condition number of the matrix A.

Tools

On Canvas, under "Programming" tab on the Home page, there is a lot of information on how to setup MATLAB or its free alternative Octave, including online versions so you don't have to install anything. You could also use Python. I provided you with a few MATLAB snippets. **Feel free to modify them as needed or use another language to plot the graphs.** Just let me know if you did that. If you want, you can share these with us.

- setup.m To run it, type the command line

  ```
  [A,b] = setup(4); ( or [A,b] = setup(n); )
  ```

  if n is defined already. This will generate the (sparse) matrix A and the vector $\mathbf{b}$. A is of size $(n-1) \times (n-1)$ and the dimension of $\mathbf{b}$ is $n-1$. Recall that for n subintervals there are $n+1$ unknowns $w_i = w(x_i)$, $i = 0, \ldots, n$ but we know already that $w_0 = w_n = 0$. The interior values are collected as the vector $\mathbf{w}$. Note: you have to **pad the solution** with zero at the start and the end so you get the complete numerical solution.

- To solve a linear system using PA=LU factorization you need the lu and backslash (\) commands:

  ```
  [L,U,P] = lu(A);  % This finds the matrices L,U,P

  x = A\b;  % This solves the system Ax=b
  ```

  Make sure you use **two-step back substitution** to solve the linear system.

- To plot a function in MATLAB $f(x)$ on the interval $[a, b]$, the command is

```
fplot(f,[a,b]);
```

Here f is a handle to a function. For example,

```
f = @(x) exp(3*x).*(1+x);
```

Using vectorized version of functions in MATLAB is more efficient and faster.

- To plot the numerical solution: assume it is stored in vector **y**. (The name might be a bit confusing. I chose it because this is the vector that will be plotted against vector **x**. Feel free to change it to something that "speaks" more to you.) Vector **y** is of size $n-1$ and before plotting it, two modifications are needed:
  - pad it with two zeros, one in front and one in back. In MATLAB, assuming vector **v** is a column vector, padding is achieved by "**v = [0; v; 0];**" If **v** is a row vector, then we need "**v = [0, v, 0];**" Note the difference in the delimiters, semicolon vs comma!
  - In MATLAB, **plot(y)** generates a 2d line plot **plot(x, y)** where **x** is the vector [1 2 3 ... length(**y**)] (i.e "**x = 1:length(y)**"). Recall: the meaning of the solution vector is an approximation of the deflection of the beam at a certain **distance x from the left end** of the beam. The length of the padded solution vector will be $n+1$, so you need $n+1$ equally spaced points starting at 0 and ending at L. One way to get this in MATLAB is

    ```
    x = linspace(0,L,n+1)';
    ```

    Note the prime symbol (') indicates transposition, we want **x** to match the shape of **y**.

- To get the error at the middle of the beam and assuming you defined the true solution as w(x), type

  ```
  err = abs(y((1+end)/2) - w(L/2))
  ```

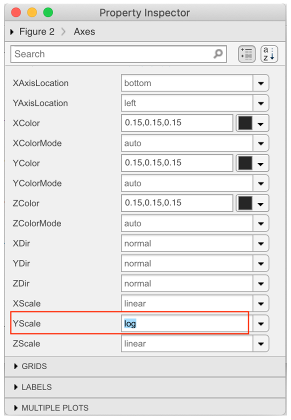
- To observe better the evolution of the error, you can plot on **the same figure** the errors at **all levels**. Assuming you saved all solution.m files, you can calculate all error vectors, e.g. for $k = 4$, solution4.m

  ```
  n = 2^(k+1);  x4 = linspace(0,L,n+1)';
  % solve the system Ay=b using PA=LU factorization.
  % You have to write the correct sequence of commands here!
  y4 = [0;y;0];
  w4 = w(x4);  err4 = abs(y4-w4);
  ```

  Plotting several graphs on the same figure is done the usual way:

  ```
  plot(x1,err1); hold on;
  plot(x2,err2);
  ...
  plot(x20,err20);
  ```

You will notice that after 4-5 levels, the errors are so small, relative to err1, that they appear as zero. You can further make the different error plots more distinguishable if you use a logarithmic y-scale. To achieve that you have (at least) two ways: 1. create another multi-plot figure using **"semilogy(x4,err4)"** instead of **"plot(x4,err4)"** command, for all levels. 2. Starting with the existing error plot, open the property inspector and change "YScale" from "linear" to "log"



XAxisLocation	bottom
YAxisLocation	left
XColor	0.15,0.15,0.15
XColorMode	auto
YColor	0.15,0.15,0.15
YColorMode	auto
ZColor	0.15,0.15,0.15
ZColorMode	auto
XDir	normal
YDir	normal
ZDir	normal
XScale	linear
YScale	log
ZScale	linear

- To calculate the condition number of a matrix A, type

```
KN = condest(A) % use KN = cond(A) if A is a full matrix
```

- Since we will need to process the midpoint error and condition number of A for each level, you should collect them in vectors, say **E** and **KN**. You can initialize them at the beginning as (for example)

```
E = zeros(20,1); KN = E;
```

  and then for each level k (from 1 to 20), have instead

```
E(k) = abs(y((1+end)/2) - w(L/2)); KN(k) = condest(A);
```

- To verify the second order of approximation, $\mathbf{E} = O(h^2)$, store the length of the segments in another vector, say **H**, and follow the work from Group Project 1:

```
x = log(H); y = log(E);
plot(x,y,'bo-'); xlabel('log(h)'); ylabel('log(err)'); grid on
dx = x(2:end) - x(1:end-1); dy = y(2:end) - y(1:end-1);
slope = dy'./dx'
```

  The slopes should be roughly equal to 2 but don't get alarmed if you see some variation. In fact, you should see that **at some point the errors stop decreasing and start increasing! This is because the condition number of matrix A becomes worse as we refine the discretization and the solution is losing accuracy.** To confirm, modify the above script to show that $\mathbf{KN} = O(h^{-2})$. Recall that the relative error satisfies

$$\frac{||x - x_a||}{||x||} \sim \text{cond}(A)\, \epsilon_{\text{mach}}$$

So, the approximate solution deviates more and more from the true one, i.e. the error (in particular, the error at the midpoint) increases when more subintervals are used!

This behavior is inherent to (all) finite difference methods. It stems from the fact that dividing by very small numbers leads to a loss of accuracy. Sadly, this is not a problem we can avoid by applying "tricks" like multiplying by the conjugate, etc.

Tasks and Questions

1. Write the MATLAB or Python code defining the true solution $w(x)$. Make sure the value you use for the intensity of uniform load, q, has units lb/in. Otherwise, you will get a big mismatch between the numerical and the "true" solution.
2. For **each refinement level** $k = 1, \ldots, 20$:
   - (a) Generate the matrix A and for the vector $\mathbf{b}$.
   - (b) Write the commands needed to solve the system $Ay = b$ using Gaussian Elimination (PA=LU). (It is sufficient to write them just once.)
   - (c) For $k = 1, 2$, plot **on the same figure** (one figure per level) the true and the approximate solution. You can plot the numerical solution either as a piecewise line function or as a scatter plot. **Annotate** the figures correspondingly.
   - (d) For **each level** (value of k), plot the **error in the solution**. **Annotate** the figure correspondingly.
   - (e) (Bonus) Plot **on the same figure all error plots** using logarithmic y-scale.
   - (f) Calculate and store the error at the middle of the beam and the condition number of matrix A in vectors $\mathbf{E}$ and $\mathbf{KN}$, respectively. (These are just suggestions for names.)
   - (g) Note: depending on your computer, you might not be able to run all twenty levels and that's OK. Just report as many levels as you can.
3. Verify that $\mathbf{E} = O(h^2)$:
   - (a) Include the graph and the **vector with slopes** described above.
   - (b) Does the slope match your expectations?
4. Verify that $\mathbf{KN} = O(h^{-2})$:
   - (a) Include the graph and the **vector with slopes** described above.
   - (b) Does the slope match your expectations?
5. What conclusions can you draw about this practical application?

6. (Bonus) Using the general meaning of "well-conditioned problem", is finding a numerical approximation of this BVP a well-conditioned problem? Why or why not?

7. Give feedback.
   (a) Did you struggle with any part of the group project?
   (b) What do you think we should change so our next sessions run better?
   (c) Do you have any conceptual questions that didn't get answered?
   (d) Feel free to add any comments that you want to make but I haven't listed.

8. Submit on Canvas your **report** (.pdf, .doc, .docx, or .odt file).