Part 1

Linear systems that arise in many applications can become quite large, and it is often necessary to exploit any structure and/or sparsity in the matrices to reduce the computational burden. One such application arises in the solution of two-point boundary-value problems:

\[ u(t)y''(t) + v(t)y'(t) + w(t)y(t) = f(t), \quad \text{where} \ y(a) = \alpha \ \text{and} \ y(b) = \beta. \]

The functions \( u, v, w \) and \( f \) are assumed to be known, and the goal is to compute an approximation to the solution \( y \).

Such problems abound in nature and are frequently very hard to handle because it is often not possible to express \( y(t) \) in terms of elementary functions (as is done in undergraduate courses on differential equations). Numerical methods are employed in order to approximate \( y(t) \) at discrete points inside the interval \([a, b]\).

The approach we consider begins by subdividing the interval \([a, b]\) into \( n+1 \) equal subintervals, each of length \((b-a)/(n+1)\):

\[
t_0 = a, \ t_1 = a + h, \ t_2 = a + 2h, \cdots, t_n = a + nh, \ t_{n+1} = b;
\]

We approximate the first and second derivatives of \( y \) by the difference quotients

\[
y'(t) \approx \frac{y(t+h) - y(t-h)}{2h} \quad \text{and} \quad y''(t) \approx \frac{y(t-h) - 2y(t) + y(t+h)}{h^2}.
\]

These are called centered difference approximations for derivatives. One sided approximations, such as

\[
f'(t) \approx \frac{y(t+h) - y(t)}{h}
\]

could also be used, but the centered differences are preferred because they provide more accuracy. The points \( t_i = a + ih \) are called grid points, and the value \( h = (b-a)/(n+1) \) is called the step size. Smaller step sizes generally produce better approximations to the derivatives, so better accuracy requires smaller step size, and hence larger number of grid points.

In this project, we apply this approach to a canonical example, known as the one-dimensional Poisson equation:

\[-y''(t) = f(t) \quad \text{on} \ [0,1] \quad \text{with} \ y(0) = y(1) = 0.\]

1. Let \( y_i = y(t_i) \) and \( f_i = f(t_i) \), and show that by using the centered difference formula for \( y''(t) \), we can compute approximations to \( y_i \) by solving the linear system \( Ty = h^2f \), where

\[
T = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & -1 & 2 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}
\]

2. Verify that the matrix \( T \) has an \( LU \) factorization with

\[
L(i,i) = 1 \quad \text{and} \quad L(i+1,i) = -\frac{i}{i+1} \quad \text{and} \quad U(i,i) = \frac{i+1}{i} \quad \text{and} \quad U(i,i+1) = -1
\]

Is partial pivoting needed? Why or why not?
3. Find a formula for the determinant of the matrix $T$. Explain how you got this formula.

4. Write the following MATLAB functions:

   (a) $T = \text{PoissonMat}(n)$ which, given $n$, creates the $n \times n$ matrix $T$.

   (b) $y = \text{PoissonSolve}(f)$ which, given a vector $f$ of length $n$, computes the solution $y$ of $Ty = h^2 f$.

      Your function should not construct $T$ explicitly, but rather should exploit the special structure of $T$.

5. Now compare your solution scheme with a naive approach that constructs $T$ explicitly and uses the MATLAB backslash operator to solve for $y$. In particular, create a script file with the following commands:

   ```matlab
tic
   T = PoissonMat(n);
   h = 1/(n+1);
   b = h*h*f;
   y = T \ b;
   NaiveTime = toc

tic
   y = PoissonSolve(f);
   FastTime = toc
   ```

   You may want to put some other things in the script, such as generation of a function $f(t)$ (for example, $f(t) = \sin \pi t$). Experiment with various values of $n$, say $n = 10, 100, 500, 1000, 5000$, and see when your method becomes faster than the naive approach. Use a plot or a table to report your findings.

6. How many flops are required for your function $\text{PoissonSolve}$ to solve for $y$? How does this compare with the naive approach?

   Along with your write up, please include printouts of any functions and scripts used to produce your results. Please neatly organize the material you turn in.

**Matlab Remarks:**

- To generate the vector $f$, you may want to make use of the MATLAB commands `linspace`, `pi`, `sin`.
- In the function $\text{PoissonMat}$, you may want to make use of the MATLAB commands `eye`, `diag`.

**Part 2**

This part of the project concerns the notion of backward stability in the solution of linear systems by Gaussian elimination. All calculations are to be performed by hand, unless otherwise noted.

1. Solve the following system of equations:

   \[
   \begin{align*}
   -10^{-4}x + y &= 1 \\
   x + y &= 2
   \end{align*}
   \]  

   using Gaussian elimination, first using exact arithmetic, then using a finite precision arithmetic model with $\beta = 10$ and $t = 3$ significant digits. Compare the solutions. What can be said about the (backward) stability of GE on this particular problem?

2. Same as in 1, but now using partial pivoting (that is, interchanging the two equations in (1) before elimination). Comment on the (backward) stability of GEPP (= Gaussian Elimination with Partial Pivoting) in this case.
3. Same as in 1, but now the linear system is given by

\[
\begin{align*}
-10x + 10^5y &= 10^5 \\
x + y &= 2
\end{align*}
\]  
(2)

(this system is obtained from (1) by multiplying the first equation by $10^5$). Notice that here, partial pivoting is the same as no pivoting. Draw conclusions on the (backward) stability of GEPP in this case.

4. Consider the $n \times n$ Wilkinson matrix

\[
W_n = \begin{bmatrix}
1 & 0 & 0 & \cdots & 1 \\
-1 & 1 & 0 & \cdots & 1 \\
-1 & -1 & 1 & \cdots & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-1 & -1 & \cdots & -1 & 1
\end{bmatrix}.
\]

4a. Compute (by hand) the LU factorization of $W_5$.

4b. “Guess” the LU factorization of $W_n$, for any $n$.

4c. Write a MATLAB function $W = \text{wilkin}(n)$ that generates the $n \times n$ Wilkinson matrix.

4d. Perform the following MATLAB experiment:

4d.1 Generate $A = W_{60}$.

4d.2 Let $e \in \mathbb{R}^{60}$ be the column vector with all entries equal to 1. Form $b = Ae$.

4d.3 Use the backslash operator to solve $Ax = b$.

4d.4 Compare the computed solution with the exact solution, $x = e$.

4e. Repeat the experiment in 4.d for smaller values of $n$. What is the largest value of $n$ for which $W_n x = b$ can be solved accurately by GEPP?

4f. Use the MATLAB functions $\text{cond}$ (or $\text{condest}$) to compute or estimate the condition number of $W_n$ for $n = 10, 20, \ldots, 60$. Report your results in a table. Compare these results with the conditioning of the Hilbert matrix ($\text{hilb}(n)$). Based on this results, would you say that the Wilkinson matrix is ill-conditioned or well-conditioned?

4g. Comment on the backward stability of GEPPS (= GE with Partial Pivoting and Scaling) on this problem. (Note that here PP = no pivoting and that the matrix is already perfectly scaled).