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 \( y \).

Differential equations like this arise in many applications, and usually they are very hard to solve because it is not possible to express \( y(t) \) in terms of elementary functions (as is done with very simple problems in undergraduate courses on differential equations). Numerical methods are employed in order to approximate \( y(t) \) at discrete points inside the interval \([a,b]\). Although we do not formally consider solutions to differential equations in this course, in this homework we will explore one approach that leads to a linear system of equations.

The approach we consider begins by generating \( n+2 \) points \( t_0, t_1, \ldots, t_n, t_{n+1} \) in the interval \([a,b]\), spaced equally by the amount \( h = (b-a)/(n+1) \). That is,

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

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. 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\(^1\).

\(^1\)This statement about \( h \) being small needs to be qualified. Recall from the first homework assignment that a difference quotient approximation of a derivative does improve as \( h \) gets small, but only up to some limit that depends on the floating point precision.
In this homework, we apply centered difference approximations to a canonical example, known as the one-dimensional Poisson equation:

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

1. (Paper and pencil problem): 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 \(Ty = h^2 f\), where

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

To do this, try \(n = 5\). If you do it for \(n = 5\) it should be obvious that it generalizes to larger values of \(n\).

2. (Paper and pencil problem): Verify, with a \(5 \times 5\) example, that the matrix \(T\) has an \(LU\) factorization with

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

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

3. (Paper and pencil problem): Derive algorithms to solve

\[
Lz = b \quad \text{and} \quad Uy = z
\]

for the \(L\) and \(U\) given in problem 2. The algorithm should require only one \(for\) loop.
4. (MATLAB problem): Write a MATLAB function \( y = \text{PoissonSolve}(f) \) which, given a vector \( f \) of length \( n \), computes the solution \( y \) of \( Ty = b \), where \( b = h^2f \). This function should not construct \( T \) explicitly, but rather should use the algorithms you derived in problem 3 to compute a solution for \( y \) very efficiently. The beginning of this function should look like:

```matlab
function y = PoissonSolve( f )
%
% y = PoissonSolve(f);
%
% Use finite difference approximations to solve
% Poisson's equation \(-y''(t) = f(t), \ 0 \leq t \leq 1\)
% \( y(0) = y(1) = 0 \)
%
% Input: f - vector whose values are \( f_i = f(t_i) \),
% where \( t_i \) are equally spaced points in the
% interval \([0,1]\).
%
% Output: y - vector containing the approximate solution.
% That is, \( y_i = y(t_i) \).
%
% The basic idea of this code is to solve \( Ty = h^2f \)
% where \( T = \text{tridiag}(-1, 2, -1) \), and \( h = 1/(n+1) \). The solution to the
% linear system is computed using the LU factorization of \( T \) (which
% is known, and does not need to be computed explicitly).
```

5. Consider \( f(t) = \sin(\pi t) \).

   (a) (Paper and pencil problem): With \( f(t) = \sin(\pi t) \), use basic calculus techniques to solve
   \[-y''(t) = f(t) \text{ on } [0, 1] \text{ with } y(0) = y(1) = 0.\]
   for \( y(t) \). Show your work.

   (b) (MATLAB problem): Use this simple example for \( f(t) \) to test your MATLAB code for \text{PoissonSolve} to make sure it is working correctly. You could do this, for example, by plotting \( y(t) \) you found in part (a) of this problem, and plotting the vector \( y \) computed by your \text{PoissonSolve} code to make sure they look the same.

6. (MATLAB problem) Write a MATLAB function, \( T = \text{PoissonMat}(n) \) which, given \( n \), explicitly creates the \( n \times n \) matrix \( T \). Your function should include some documentation. The code for this function should be very short – use the MATLAB built-in functions \text{eye} and \text{diag}.
7. (MATLAB problem): Use the following script to test the speed of PoissonSolve.

```matlab
% Script: PoissonTest
%
% This script compares the time it takes to solve a linear system
% using two methods:
%   Naive Method = Matlab's backslash operator
%   Fast Method = My efficient code.
% The linear system we solve arises from finite difference
% discretization of a 1-dimensional Poisson equation:
%   -y''(t) = f(t), 0 <= t <= 1,
%   y(0) = y(1) = 0.
%
% To test the methods, we use f(t) = sin(pi*t)
%
% The script will compute average timings for several values of n (i.e.,
% dimension of the linear system), and will produce a plot to show
% the difference in timings.
%
% n = 100:50:1500;
AvgNaiveTime = zeros(length(n),1);
AvgFastTime = zeros(length(n),1);
for i=1:length(n)
    t = linspace(0,1,n(i)+2)';
    f = sin(pi*t(2:end-1));

    NaiveTime = zeros(10,1);
    for k = 1:10
        tic;
        T = PoissonMat(n(i));
        h = 1/(n(i)+1);
        b = h*h*f;
        y = T \ b;
        NaiveTime(k,1) = toc;
    end
    AvgNaiveTime(i,1) = mean(NaiveTime);

    FastTime = zeros(10,1);
    for k = 1:10
        tic
        y1 = PoissonSolve(f);
        FastTime(k,1) = toc;
    end
    AvgFastTime(i,1) = mean(FastTime);
end
```
You need to do two additional things related to this script:

(a) Explain the purpose of the \texttt{for k = 1:10} loops.
(b) After the final \texttt{end} statement you should include some additional MATLAB code that will produce a plot that looks similar to the following:

![Plot Description]

The work you turn in should include a printout of your version of the above plot (the timings might be slightly different than what I obtained on my computer). Please include printouts of all MATLAB code used to produce your results. Please neatly organize the material you turn in.