You will need email a copy of your code to the TA.

In this assignment, we shall implement continuous piecewise linear finite elements in one spatial dimension. We assume that we are given a set of vertices
\[ \alpha = x_0 < x_1 < \ldots < x_m = \beta. \]
The resulting finite element space (without any boundary conditions imposed) is denoted by \( S_h \). We shall consider two cases:
(a) Dirichlet boundary conditions on both sides. The matrix problem has \( n = m - 1 \) degrees of freedom.
(b) Neumann boundary conditions on both sides. The matrix problem has \( n = m + 1 \) degrees of freedom.

In both cases, the stiffness matrix \( A \) is tri-diagonal.

Since the matrix is tri-diagonal, it can be stored as three vectors, \( L, D, U \) with \( L, D, U \) respectively containing the entries below, on and above the diagonal. The three vectors will have dimension \( n \). The \( i \)’th entry will be the entry in the \( i \)’th row. The first entry of \( L \) and the last entry of \( U \) can be set to zero as they are not involved in the calculations.

We shall have to solve problems of the form \( Ax = b \) with \( x, b \in \mathbb{R}^n \). This can be accomplished using a simple tri-diagonal solver:

**Fortran version:** trisol.f

**C version:** trisol.c

If you plan to implement in Matlab, you will need to rewrite the routine as a matlab file (the fortran version will be the simplest to modify). You are not allowed to build full \( n \times n \) matrices (**assignments using \( n \times n \) matrices in full representation will not be accepted**).

We shall consider the problem:
\[ -(ay')' + by' + qy = f \]
augmented by boundary conditions. The corresponding bilinear form is
\[ A_1(u, v) = \int_\alpha^\beta [au'v' + bu'v + quv] \, dx. \]
Task 1. Write a routine which builds vectors $U_1, D_1, V_1$ (of length $m + 1$) corresponding to the stiffness matrix for continuous linear finite elements and the above form, i.e., the tri-diagonals for the matrix

$$(\tilde{A}_1)_{ij} = A_1(\phi_j, \phi_i), \quad i, j = 0, \ldots, m.$$ 

Note that you shift the indexing when programming in Fortran or Matlab as arrays start at 1. The assembly of these vectors must be accomplished by running one loop through the elements, i.e., a loop from $j = 1, \ldots, m$ as discussed in recitation (assignments which implement this as a double loop will not be accepted). The integrals over elements are approximated by quadrature, i.e.,

$$(0.2) \quad \int_{x_i}^{x_{i+1}} w \, dx \approx \frac{h_i}{2} [w(x_{i,1}) + w(x_{i,2})].$$

Here $h_i = x_{i+1} - x_i$, $x_{i,1} = x_i + h_i(0.5 - \frac{1}{2\sqrt{3}})$, and $x_{i,2} = x_i + h_i(0.5 + \frac{1}{2\sqrt{3}})$. The functions $a, b, u$ are to be provided by function subroutines (for debugging purposes, you should alternate them between 0 and 1 so as to check each term in the form independently of the others and run small values of $m$).

Task 2. Write a routine which assembles a right hand side vector $F_1$ of length $m + 1$ defined by

$$(F_1)_j = \int_{\alpha}^{\beta} f \phi_j, \quad j = 0, \ldots, m.$$ 

Again $f$ is provided by a function subroutine and the integrals over elements is approximated by the quadrature (0.2). Debug your routine by using a simple $f$ and small $m$. The assembly of $f$ must involve only one loop ($i = 1, \ldots, m$) over the elements as discussed in recitation (assignments which implement this as a double loop will not be accepted).

We consider the case when $\alpha = 0, \beta = 1, a = (1 + x), b = q = 0$ and (non-homogeneous) Dirichlet boundary conditions. Moreover, we consider the (continuous) problem with solution $u = 2x^3$ and $f = -12x - 18x^2$. The finite element approximation is the function in $u_h \in S_h$ which satisfies $u_h(0) = 0, u_h(1) = 2$, and

$$(0.3) \quad A_1(u_h, \phi) = (f, \phi), \quad \phi \in S_0^h$$

where $S_0^h$ is the set of functions in $S_h$ which vanish at 0 and 1. Write $u_h = u_1^h + \theta$ where $u_1^h$ is the function in $S_h$ which equals 2 at 1 and vanishes
on all other nodes. Then \( \theta \in S_h^0 \) satisfies

\[
A_1(\theta, \phi) = (f, \phi) - A_1(u_h^1, \phi), \quad \phi \in S_h^0.
\]

The stiffness matrix for (0.4) involves an \((m-1) \times (m-1)\) tri-diagonal matrix which can be extracted from the matrix computed by Task 1. The \(m-1\) vector coming from \((f, \phi)\) can be extracted from that computed in Task 2. The final right hand side vector is obtained by modification to incorporated the \(A_1(u_h^1, \phi_j)\) term. The modification term can be easily computed from the matrix of Task 1.

**Task 3.** For each \( m = 16, 32, \ldots, 1024 \), use a grid of \( m + 1 \) equally spaced nodes and compute the finite element approximation given by (0.3). Denote the approximation by \( u_h \in S_h \). Next, compute the coefficients in the basis for \( S_h \) of the interpolant \( I_h u \) (into \( S_h \)) of the solution \( u \). Compute the tridiagonal vectors for the mass matrix for the subspace \( S_h \), i.e.,

\[
(M_h)_{ij} = (\phi_j, \phi_i), \quad i, j = 0, \ldots, m,
\]

using Task 1 with \( q = 1, a = b = 0 \). Compute the vector with entries \( v_i = u_h(x_i) - I_h u(x_i), \ i = 0, \ldots, m \). Show that

\[
\| u_h - I_h u \rVert_{L^2(\alpha, \beta)} = [(M_hv) \cdot v]^{1/2}.
\]

Using this, compute and report the values of \( \| u_h - I_h u \rVert_{L^2(\alpha, \beta)} \) for each \( m \) above.

We next consider the case when \( \alpha = 1, \beta = 2, a = b = q = 1 \) and non-homogeneous Neumann boundary conditions

\[
- u'(1) = c_0, \quad u'(2) = c_1.
\]

**Task 4.** Compute \( c_0, c_1 \) and \( f \) so that the problem (0.1),(0.5) has solution \( u = 2x^2 \). Also, show that \( A_1(u, v) \) is coercive on \( H^1(\alpha, \beta) \).

**Task 5.** For each \( m = 16, 32, \ldots, 1024 \) compute the finite element approximation \( u_h \in S_h \) satisfying

\[
A_1(u_h, \phi) = (f, \phi) + c_0\phi(1) + c_1\phi(2), \quad \phi \in S_h.
\]

The stiffness matrix comes from Task 1 while the right hand side comes from Task 2 with a suitable modification for the non-homogeneous Neumann conditions. As in Task 3, compute and report the values of \( \| u_h - I_h u \rVert_{L^2(\alpha, \beta)} \) for each \( m \) above.