AME50541: Finite Element Methods
Homework 3: Due Friday, March 8, 2019

Be sure to email all code to the instructor and TA.

Problem 1: (25 points) Consider a single one-dimensional, $(p+1)$-node Lagrangian element with nodes located as $x_1^e, \dots, x_{p+1}^e$.

(a) What is the order of the element, i.e., its degree of completeness?

(b) Write the expressions for the element basis functions and their derivatives in terms of the nodal positions.

(c) Implement a function that evaluates all one-dimensional Lagrange polynomials and their derivatives associated with nodes $x_1^e, \dots, x_{p+1}^e$. Your function should have the following signature:

```
function [Q, dQ] = eval_interp_onedim_lagrange(xk, x)
%EVAL_INTERP_ONEDIM_LAGRANGE Evaluate Lagrange interpolation functions on
%interval (xk(1), xk(end)) associated with nodes xk (number of nodes = nv)
%at points x (number of points = nx).
%
%Input arguments
%----------------
%   XK : Array (nv,) : Nodes at which Lagrange polynomials interpolate
%    values.
%
%   X : Array (nx,) : Points at which Lagrange polynomials are evaluated.
%
%Output arguments
%----------------
%   Q : 2D array (nv, nx) : Lagrange interpolation functions (nv)
%    evaluated at each point in x.
%
%   DQ : 2D array (nv, nx) : Derivative of Lagrange interpolation
%   functions (nv) evaluated at each point in x
```

Starter code is provided on the course website in the Homework 3 code distribution: `eval_interp_onedim_lagrange.m`.

(d) Plot the basis functions assuming the 5 nodes are equally spaced in the domain $(-1, 1)$.

(e) Consider a one-dimensional domain, discretized by 3 of these 5-node Lagrangian elements. Create a global numbering for the mesh and write the `ldof2gdof` matrix.

Problem 2: (25 points) Hypercube finite elements (quadrilaterals in 2 dimensions, hexahedron in 3 dimensions, etc), also called tensor product elements, can be obtained as a tensor product of one-dimensional finite elements, where a tensor product is defined as

$$\boldsymbol{C} = \boldsymbol{a} \otimes \boldsymbol{b} \quad \Longleftrightarrow \quad C_{ij} = a_i b_j,$$

where $\boldsymbol{a} \in \mathbb{R}^m$, $\boldsymbol{b} \in \mathbb{R}^n$, and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$. Consider a singe d-dimensional, $(p+1)^d$-node Lagrangian tensor product element with nodes located at (x_i^e, y_j^e) for $i, j = 1, \dots, p+1$ for $d = 2$ (quadrilateral element) and (x_i^e, y_j^e, z_k^e) for $i, j, k = 1, \dots, p+1$ for $d = 3$ (hexahedral element).

(a) What is the order of the tensor product element, i.e., its degree of completeness?

(b) Write the expression for the element basis functions and their partial derivatives. You do not need to explicitly write the basis functions; rather, write them in terms of the 1D Lagrangian basis

$$\{\phi_i^{1d}(s; s_1, \dots, s_{p+1})\}_{i=1,\dots,p+1},$$

defined as the (unique) polynomial of degree p that is unity at s_i and zero at all other nodes, i.e., it is unity at the node defining it and zero at all other nodes, i.e., $\phi_i^{1d}(s_j; s_1, \dots, s_{p+1}) = \delta_{ij}$.

(c) Show the basis functions satisfy the Lagrange interpolation property (unity at the node defining it and zero at all other nodes).

(d) Implement a function that evaluates all the d-dimensional polynomials generated by the one-dimensional polynomials $\{\phi_i^{1d}(s; s_1, \dots, s_{p+1})\}_{i=1,\dots,p+1}$. Your function should have the following signature:

```
function [Q, dQ] = eval_interp_hcube_from_onedim(ndim, Q1d, dQ1d)
%EVAL_INTERP_HCUBE_FROM_ONEDIM Evaluate interpolation functions for
%hypercube from interpolation functions in one-dimension. The nodes of
%the interpolation functions (and points at which they are evaluated) are
%inherited from the tensor product structure.
%
%Input arguments
%---------------
%   NDIM : number : Number of dimensions of hypercube
%
%   Q1D : 2D array (nv, nx) : One-dimensional interpolation functions (nv)
%     evaluated at each point in x,
%
%   DQ1D : 2D array (nv, nx) : Derivative of one-dimensional interpolation
%     functions (nv) evaluated at each point in x
%
%Output arguments
%----------------
%   Q : 2D array (nv^ndim, nx^ndim) : Interpolation functions for the
%     hypercube element evaluated at each point inherited from the 1D
%     element.
%
%   DQ : 2D array (nv^ndim, ndim, nx^ndim) : Derivative of interpolation
%     functions evaluated at each point inherited from the 1D element.
```

Starter code is provided on the course website in the Homework 3 code distribution: `eval_interp_hcube_from_onedim.m`. For simplicity, you may only consider the special cases of $d = 2$ and $d = 3$.

(e) For the quadrilateral element, take $p = 2$ and number the nodes using a single index from $i = 1, \dots, 9$ (should be consistent with the ordering you chose in your code in the previous part). Draw the element, label the nodes, and plot each basis function. Assume the element covers the interval $(-1, 1) \times (-1, 1)$ and the nodes are uniformly spaced in each direction.

(f) Consider a two-dimensional domain, discretized by 4 of these 9-node Lagrangian quadrilaterals, configured as 2 elements in the x-direction and 2 elements in the y-direction. Create a global numbering for the nodes and elements in the mesh and write the `ldof2gdof` matrix.

Problem 3: (25 points) Derive the element stiffness matrix and force vector for the following PDE

$$\begin{aligned} -\frac{d^2u}{dx^2} - u + x^2 &= 0 \\ u(0) = 0, \left.\left(\frac{du}{dx}\right)\right|_{x=1} &= 1. \end{aligned} \tag{1}$$

Assume the element domain is $\Omega^e := (x_1^e, x_2^e)$ and linear Lagrangian basis functions are used:

$$\phi_1^e(x) = \frac{x_2^e - x}{x_2^e - x_1^e}, \qquad \phi_2^e(x) = \frac{x - x_1^e}{x_2^e - x_1^e}.$$

Be sure to consider two cases: one that includes the boundary term and one that does not. When should the element with the boundary term included be used? As always, feel free to use any symbolic mathematics software to ease the burden of the algebra/calculus manipulations. Finally, implement the element stiffness matrix and force vector using the starter code below (and provided on the course website).

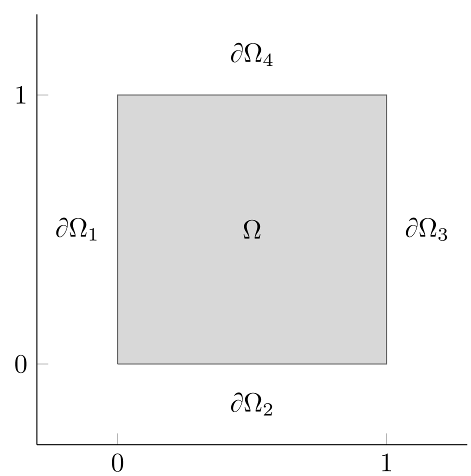
```
function [Ke, Fe] = eval_elem_contrib_pde0(elem_data)
%EVAL_ELEM_CONTRIB_PDE0 Evaluate the stiffness matrix and force
%vector for PDE0 element in 1D using linear Lagrange basis.
%
%   PDE : -u'' - u + x^2 = 0,   0 < x < 1
%   BCs :  u(0) = 0, u'(1) = 1
%
%   Basis: phi1 = (x2-x)/(x2-x1)
%          phi2 = (x-x1)/(x2-x1)
%
%Input arguments
%----------------
%   ELEM_DATA : structure : Element-specific fields
%      ELEM_DATA.x1 : number : position of local node 1
%      ELEM_DATA.x2 : number : position of local node 2
%
%Output arguments
%-----------------
%   KE : 2D matrix (ndof_per_node*nnode_per_elem,
%                   ndof_per_node*nnode_per_elem)
%      : Element stiffness matrix
%
%   FE : Array (ndof_per_node*nnode_per_elem,) : Element force vector
```

Starter code is provided on the course website in the Homework 3 code distribution: `eval_elem_contrib_pde0.m`. To run/test your function, use the function `create_unif_mesh_1d` provided on the course website to create a 1d mesh and `create_elem_structs_pde0` to setup the element data structures for this PDE. The latter function will create a structure array, one entry for each element; each entry can be passed to your function `eval_elem_contrib_pde0` to evaluate the corresponding element stiffness matrix and force vector.

Problem 4: (35 points) Derive the element stiffness matrix and force vector for the following PDE

$$\begin{aligned} -\Delta T &= 0 && \text{in} \quad \Omega \\ \nabla T \cdot n &= 1 && \text{on} \quad \partial\Omega_1 \\ \nabla T \cdot n &= 0 && \text{on} \quad \partial\Omega_2 \\ T &= 0 && \text{on} \quad \partial\Omega_3 \cup \partial\Omega_4, \end{aligned} \tag{2}$$

where the domain is given in the figure below.



Square domain $\Omega = [0, 1] \times [0, 1]$ with boundary $\partial\Omega = \overline{\partial\Omega_1 \cup \partial\Omega_2 \cup \partial\Omega_3 \cup \partial\Omega_4}$

Assume the element domain is $\Omega^e := (x_1^e, x_2^e) \times (y_1^e, y_2^e)$ and linear Lagrangian basis functions are used:

$$\begin{aligned}
\phi_1^e(x,y) &= \left(\frac{x_2^e - x}{x_2^e - x_1^e}\right)\left(\frac{y_2^e - y}{y_2^e - y_1^e}\right)\\
\phi_2^e(x,y) &= \left(\frac{x - x_1^e}{x_2^e - x_1^e}\right)\left(\frac{y_2^e - y}{y_2^e - y_1^e}\right)\\
\phi_3^e(x,y) &= \left(\frac{x_2^e - x}{x_2^e - x_1^e}\right)\left(\frac{y - y_1^e}{y_2^e - y_1^e}\right)\\
\phi_4^e(x,y) &= \left(\frac{x - x_1^e}{x_2^e - x_1^e}\right)\left(\frac{y - y_1^e}{y_2^e - y_1^e}\right).
\end{aligned}$$

Be sure to consider two cases: one that includes the boundary term and one that does not. When should the element with the boundary term included be used? As always, feel free to use any symbolic mathematics software to ease the burden of the algebra/calculus manipulations. Finally, implement the element stiffness matrix and force vector using the starter code below (and provided on the course website).

```
function [Ke, Fe] = eval_elem_contrib_pde1(elem_data)
%EVAL_ELEM_CONTRIB_PDE1 Evaluate the stiffness matrix and force
%vector for PDE1 element in 1D using linear Lagrange basis.
%
%   PDE : -T_{,ii} = 0,   0 < x < 1, 0 < y < 1
%   BCs :  T(x=1, y) = T(x, y=1) = 0,
%          (dT*n)_{x, y=0} = 0, (dT*n)_{x=0, y} = 1
%
%   Basis: phi1 = (x2-x)(y2-y)/(x2-x1)/(y2-y1)
%          phi2 = (x-x1)(y2-y)/(x2-x1)/(y2-y1)
%          phi3 = (x2-x)(y-y1)/(x2-x1)/(y2-y1)
%          phi4 = (x-x1)(y-y1)/(x2-x1)/(y2-y1)
%
%Input arguments
%----------------
%   ELEM_DATA : structure : Element-specific fields
%      ELEM_DATA.x1, ELEM_DATA.x2 : number : x limits of element
%      ELEM_DATA.y1, ELEM_DATA.y2 : number : y limits of element
%
%Output arguments
%-----------------
%   KE : 2D matrix (ndof_per_node*nnode_per_elem,
%                   ndof_per_node*nnode_per_elem)
%      : Element stiffness matrix
%
%   FE : Array (ndof_per_node*nnode_per_elem,) : Element force vector
```

Starter code is provided on the course website in the Homework 3 code distribution: `eval_elem_contrib_pde1.m`. To run/test your function, use the function `create_unif_mesh_2d_rect` provided on the course website to create a 2d mesh and `create_elem_structs_pde1` to setup the element data structures for this PDE. The latter function will create a structure array, one entry for each element; each entry can be passed to your function `eval_elem_contrib_pde1` to evaluate the corresponding element stiffness matrix and force vector.

Problem 5: (40 points) In this problem, you will implement a basic FEM code that we will enhance over the next several homework assignments. Before proceeding, carefully review the starter code, including all the comments, that has been provided on the course website in `ame50541-hwk03-code-starter.zip`. I have provided the following functions:

- `create_unif_mesh_1d`: create `xcg`, `e2vcg` for uniform 1D mesh
- `create_unif_mesh_2d_rect`: create `xcg`, `e2vcg` for uniform 2D mesh
- `create_map_ldof_to_gdof`: create `ldof2gdof` matrix

- `visualize_fem`: visualize FE mesh and solution.

You are welcome to use your own version of `create_map_ldof_to_gdof` that you implemented in Homework 1 if you made it work for an arbitrary number of degrees of freedom per node.

Problem 5.1 Implement a function that evaluates and stores the element stiffness matrix and force vector for a generic element defined by the structures `elem` and `elem_data`. See the starter code for `solve_fem_dense` for a description of `elem` and the starter code for `eval_elem_contrib_pde0` for a description of `elem_data`. Your function should have the following signature:

```
function  [Ke, Fe] = eval_unassembled(elem, elem_data)
%EVAL_UNASSEMBLED Evaluate/store element stiffness matrix and
%force vector for each element.
%
%Input arguments
%----------------
%   ELEM, ELEM_DATA : See description in CREATE_ELEM_STRUCTS_*
%
%Output arguments
%----------------
%   KE : 3D array (ndim*nnode_per_elem, ndim*nnode_per_elem, nelem) :
%     unassembled element stiffness matrices (Ke(:, :, e) is the stiffness
%     matrix of element e).
%
%   FE : 2D array (ndim*nnode_per_elem, nelem) : unassembled force vector
%     unassembled element stiffness matrices (Fe(:, e) is the force vector
%     of element e).
```

Starter code is provided on the course website in the Homework 3 code distribution: `eval_unassembled.m`. Be sure to test your function, e.g., using problems 3 or 4. This function should be *nearly identical* to the function you created in Homework 1 for the direct stiffness method. The only difference is we need to extract both element stiffness matrices and force vectors, instead just the element stiffness matrices.

Problem 5.2 Implement a function that assembles the element stiffness matrices and force vectors into the global stiffness matrix and force vector without applying Dirichlet boundary conditions. It should have the following signature:

```
function [K, F] = assemble_nobc_dense(Ke, Fe, ldof2gdof)
%ASSEMBLE_NOBC_DENSE Assemble element stiffness matrices and force
%vector into the global quantities without applying Dirichlet boundary
%conditions.
%
%Input arguments
%----------------
%   KE, FE : See defintion in EVAL_UNASSEMBLED*
%
%   LDOF2GDOF : See definition in CREATE_MAP_LDOF_TO_GDOF
%
%Output arguments
%----------------
%   K : 2D array (ndof, ndof) : assembled stiffness matrix PRIOR to static
%     condensation.
%
%   F : Array (ndof,): assembled force vector PRIOR to static condensation.
```

Starter code is provided on the course website in the Homework 3 code distribution: `assemble_nobc_dense.m`. This function should be *nearly identical* to the function `assemble_stiff_nobc` you created in Homework 1 for the direct stiffness method. The only difference is we need to assemble both the stiffness matrix and force vector, instead of just the stiffness matrix.

Problem 5.3 Implement a function that applies essential/Dirichlet boundary conditions via static condensation to the global stiffness matrix and solves for the unknown solution at the FE nodes. It should have the following signature:

```
function [u] = apply_dbc_solve(K, F, dbc_idx, dbc_val)
%APPLY_DBC_SOLVE Apply Dirichlet boundary conditions via static
%condensation.
%
%Input arguments
%---------------
%   K : 2D array (ndof, ndof) : assembled stiffness matrix
%
%   F : Array (ndim*nnode,) : assembled force vector
%
%   DBC_IDX, DBC_VAL : See defintion in SOLVE_FEM_*
%
%Output arguments
%----------------
%   U : See defintion in SOLVE_FEM_*
```

Starter code is provided on the course website in the Homework 3 code distribution: `apply_dbc_solve.m`. This function should be *nearly identical* to the function `apply_bc_solve` you created in Homework 1 for the direct stiffness method. The only difference is we directly input the force vector to the function, instead of constructing it from the force boundary conditions as in the direct stiffness method.

Problem 5.4 Implement a function that uses the finite element method to solve for the uknown PDE solution at nodes using the functions created in Problems 5.1-5.3. It should have the following signature:

```
function [u] = solve_fem_dense(e2vcg, elem, elem_data, dbc_idx, dbc_val)
%SOLVE_FEM_DENSE Approximate the solution of a PDE by solving for the nodal
%degrees of freedom on a mesh using the finite element method.
%
% Input arguments
% ---------------
%   XCG : 2D array (ndim, nnode) : The position of the nodes in the mesh.
%     The (i, j)-entry is the position of node j in the ith dimension. The
%     global node numbers are defined by the columns of this matrix, e.g.,
%     the node at xcg(:, j) is the jth node of the mesh.
%
%   E2VCG : 2D array (nnode_per_elem, nelem): The connectivity of the
%     mesh. The (:, e)-entries are the global node numbers of the nodes
%     that comprise element e. The local node numbers of each element are
%     defined by the columns of this matrix, e.g., e2vcg(i, e) is the
%     global node number of the ith local node of element e.
%
%   ELEM : structure : Element metadata
%     ELEM.NDIM : number : Number of dimensions
%     ELEM.ETYPE : string : Element type (usually 'simp' for simplex or
%       'hcube' for hypercube)
%     ELEM.PORDER : number : Polynomial order
%     ELEM.NDOF_PER_NODE : number : Number of degrees of freedom per node
%     ELEM.NNODE_PER_ELEM : number : Number of nodes per element
%     ELEM.STIFF_FORCE : function : Function that takes a single entry of
%       the ELEM_DATA structure array as input and returns the element
%       stiffness matrix and force vector.
%
%   ELEM_DATA : structure array (nelem,) : Element-specific fields
%
%   DBC_IDX : Array (ndbc,) : Indices into array defined over global dofs
%   (size = ndim*nnode) that indicates those with prescribed displacements
%   (Dirichlet BCs).
%
%   DBC_VAL : Array (ndbc,) : Value of the prescribed displacements such
%   that U(DBC_IDX) = DBC_VAL (see definition of U below).
```

```
%
% Output arguments
% ----------------
%   U : Array (ndof_per_node*nnode,) : The coefficients corresponding to
%   each DOF in the mesh.
%
% Note: The ordering of one-dimensional vectors over all/some of the
% degrees of freedom will ALWAYS be ordered first by the dofs at a fixed
% node and then across all nodes. For example, let U be the vector of size
% ndof_per_node*nnode containing the displacements at all nodes, then its
% components are U = [Ux_1; Uy_1; ... ; Ux_nnode; Uy_nnode], where Ux_i,
% Uy_i are the x- and y- displacements at node i.
```

Starter code is provided on the course website in the Homework 3 code distribution: `solve_fem_dense.m`.

Problem 5.5 Use the element developed in Problem 3 to approximate the solution of (1) using the finite element method. Use a mesh consisting of three linear elements and plot against the exact solution

$$u(x) = \frac{2\cos(1-x) - \sin(x)}{\cos(1)} + x^2 - 2.$$

What do you notice about the accuracy of the FEM solution at the nodes vs. interior to elements? Repeat the analysis using a finite element mesh with 25 linear elements and plot the solution.

Problem 5.6 Use the element developed in Problem 4 to approximate the solution of (2) using the finite element method. Use a mesh consisting of 3×3 linear elements and plot the solution. Repeat the analysis using a finite element mesh of 25×25 linear elements and plot the solution.