CME292: Advanced MATLAB for Scientific Computing

Homework #4
Symbolic, Compiled, and Parallel MATLAB & ODEs/PDES

Due: Tuesday, May 5, 2015

Instructions

For this problem set, 2 problems out of 4 are required. You are free to choose the 2 problems you complete.

Before completing problem set, please see `HomeworkInstructions` *on Coursework for general homework instructions, grading policy, and number of required problems per problem set.*

Problem 1

The unsteady, two-dimensional Euler equations are

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x}(U) + \frac{\partial G}{\partial x}(U) = 0, \tag{1}$$

where $U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}$, $V = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix}$ $F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (e+p)u \end{bmatrix}$, $G = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ (e+p)v \end{bmatrix}$, $p = (\gamma - 1)\left(e - \rho\frac{u^2+v^2}{2}\right)$, and γ is a constant. The vector U is known as the *conservative* form of the flow variables and V is the *primitive* form. Re-write the conservation form of the Euler equations in (1) in wave speed form as

$$\frac{\partial V}{\partial t} + A\frac{\partial V}{\partial x} + B\frac{\partial V}{\partial x} = 0. \tag{2}$$

where

$$A = \left[\frac{\partial U}{\partial V}\right]^{-1}\frac{\partial F}{\partial V} \qquad B = \left[\frac{\partial U}{\partial V}\right]^{-1}\frac{\partial G}{\partial V}.$$

The *symbolic* eigenvalue decomposition of A and B plays an important role in the construction of some CFD schemes. Use MATLAB's symbolic tool box to perform the following tasks:

(1) Compute A and B symbolically. The expression for A and B should *only* involve the terms of ρ, u, v, p, γ

(2) Use the expression $\gamma p = \rho c^2$ to eliminate p and γ from these expressions for A and B.

(3) Finally, compute the eigenvalue decomposition of A and B symbolically.

Problem 2

In this problem, you will spatially discretize a Partial Differential Equation (PDE) using the Finite Difference Method (FDM) to yield a coupled system of Ordinary Differential Equations (ODEs). The resulting ODEs will be integrated using `ode45` and `ode23s`. Here we consider the *viscous* Burger's equation

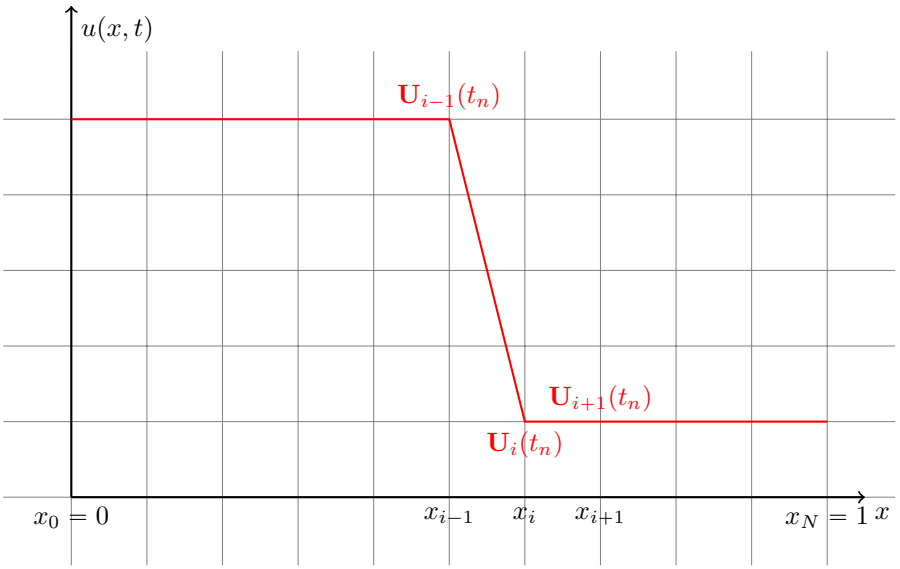


Figure 1: Discretized domain and solution for (3)

$$\frac{\partial u}{\partial t} + \frac{\partial \left(\frac{1}{2}u^2\right)}{\partial x} = \epsilon \frac{\partial^2 u}{\partial x^2} \tag{3}$$

for $x \in [0,1]$ and $t \in [0, 0.5]$, with the initial condition $u(x,0) = 1$ and boundary condition $u(0,t) = 5$. Using the Method of Lines, (3) is spatially discretized on the grid shown in Figure 1 using FDM to obtain a system of ODEs of the form

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{F}(\mathbf{U}(t), t), \tag{4}$$

where $\mathbf{U}_i(t) = u(x_i, t)$ for $i = 1, \ldots, N$. As the value of u is known as x_0 from the boundary condition, the solution at x_0 is not included in the state vector.
For stability considerations, we apply *upwinding* to the advection term, namely

$$\frac{\partial \left(\frac{1}{2}u^2\right)}{\partial x}(x_i, t) \approx \frac{\frac{1}{2}u(x_i,t)^2 - \frac{1}{2}u(x_{i-1},t)^2}{\Delta x} \qquad i = 1, \ldots, N \tag{5}$$

where $\Delta x_i = x_i - x_{i-1} = \Delta x$ as the grid is assumed uniform. The standard second order approximation to the diffusive term is applied

$$\frac{\partial^2 u}{\partial x^2}(x_i, t) \approx \frac{u(x_{i+1},t) - 2u(x_i,t) + u(x_{i-1},t)}{\Delta x^2} \qquad i = 1, \ldots, N-1. \tag{6}$$

At the last equation, a first-order, leftward bias of the second order derivative is applied

$$\frac{\partial^2 u}{\partial x^2}(x_i, t) \approx \frac{u(x_N,t) - 2u(x_{N-1},t) + u(x_{N-2},t)}{\Delta x^2}. \tag{7}$$

The boundary condition is applied as

$$u(x_0, t) = u(0, t) \tag{8}$$

in (5) and (7). Combining (5)-(8) into the form of (4), we have

$$\mathbf{F}(\mathbf{U}(t), t)_i = \begin{cases} \frac{\epsilon}{\Delta x^2}\left(\mathbf{U}_2(t) - 2\mathbf{U}_1(t) + u(0,t)\right) - \frac{1}{2\Delta x}\left(\mathbf{U}_1(t)^2 - u(0,t)^2\right) & \text{for } i = 1 \\ \frac{\epsilon}{\Delta x^2}\left(\mathbf{U}_{i+1}(t) - 2\mathbf{U}_i(t) + \mathbf{U}_{i-1}(t)\right) - \frac{1}{2\Delta x}\left(\mathbf{U}_i(t)^2 - \mathbf{U}_{i-1}(t)^2\right) & \text{for } i = 2, \ldots, N-1 \\ \frac{\epsilon}{\Delta x^2}\left(\mathbf{U}_N(t) - 2\mathbf{U}_{N-1}(t) + \mathbf{U}_{N-2}(t)\right) - \frac{1}{2\Delta x}\left(\mathbf{U}_i(t)^2 - \mathbf{U}_{i-1}(t)^2\right) & \text{for } i = N. \end{cases} \tag{9}$$

Problem 2 continued on next page...

(1) Implement the above finite difference method for semi-discretizing (3)

- This involves creating a function that accepts t and $\mathbf{U}$ as an inputs (in that order) and returning $\mathbf{F}(\mathbf{U})$ as an output. There can be additional input arguments if desired. The input/output structure is for compatibility with MATLAB's ODE solvers (required for next part).

(2) Solve the resulting system of ODEs using (4) using `ode45` and `ode23s` for $\epsilon = [0, 0.001, 0.01, 0.1, 1, 2, 5, 10]$. I recommend using $N = 101$ (101 grid points, 100 degrees of freedom), but this is not required. Make note of the number of steps required to complete the time integration, as well as the CPU time. Comment on any trends.

(3) For $\epsilon = [0, 0.1, 1, 10]$ make an animation of the time steps. Include no more than 200 frames in each animation, regardless of time steps required for timestepping. Decide whether to use `ode45` or `ode23s` from your experience in the previous part. Save the animation to a movie file.

(4) (extra credit) Repeat (2) using MATLAB's `parfor` (the different simulations are independent). Don't worry about CPU timings for each value of ϵ, just report the CPU time to complete ALL simulations. How many CPUs did you use? How does the CPU time for all simulations compare between the serial and parallel versions?

Problem 3

This this problem you will use MATLAB's MEX capabilities to speed up a 2D unstructured finite element code for solving unsteady heat flow provided for you in `heat_fem` (on course website). You will both write your own MEX-file and use MATLAB's Coder speedup the heat flow analysis. Similar to the nonlinear truss problem, no knowledge of heat flow is required. A very brief exposition is given next.
The heat flow equation is

$$\frac{\partial T}{\partial t} - \nabla \cdot (\boldsymbol{\kappa} \cdot \nabla T) = F(\mathbf{x}, t) \qquad \text{for} \qquad \mathbf{x} \in \Omega,\ t \in [0, T] \tag{10}$$

with Dirichlet boundary conditions of the form $T(\mathbf{x}, t) = \bar{T}(t)$ for $\mathbf{x} \in \Gamma_T$ and Neumann boundary conditions $(\kappa \cdot \nabla T) \cdot \mathbf{n} = 0$ for $\mathbf{x} \in \Gamma_q$, where $\Gamma_T \cup \Gamma_q = \partial\Omega$. The PDE in (10) is also equipped with an initial condition $T(\mathbf{x}, 0) = T_0(\mathbf{x})$.
Spatial discretization of (10) using the finite element method yields the linear system of ordinary differential equations

$$\mathbf{M}\frac{\partial \mathbf{T}}{\partial t} + \mathbf{K}\mathbf{T} = \mathbf{F} \tag{11}$$

where $\mathbf{M}$ is the "mass" matrix, $\mathbf{K}$ is the "stiffness" matrix, $\mathbf{F}$ is the forcing term, and $\mathbf{T}$ is the vector of nodal temperatures. The terms mass and stiffness have been adopted from the structural mechanics community and used in this context. Equation (11) is discretized in time using the backward Euler method to yield

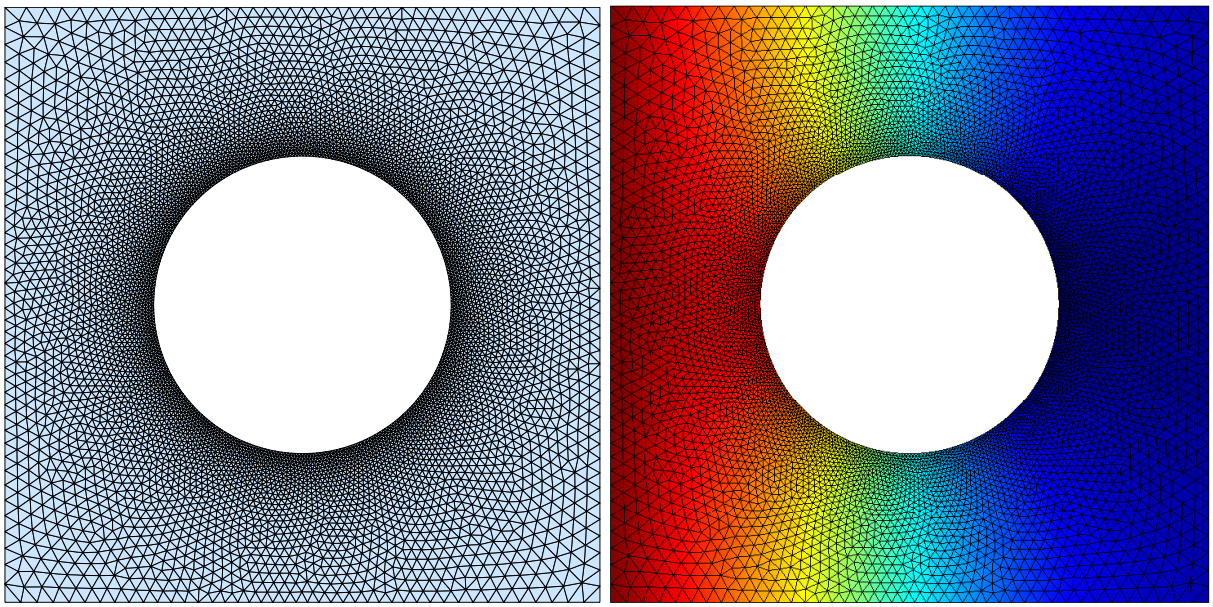
$$\mathbf{M}\left(\frac{\mathbf{T}^{n+1} - \mathbf{T}^n}{\Delta t}\right) + \mathbf{K}\mathbf{T}^{n+1} = \mathbf{F}^{n+1}. \tag{12}$$

If the solution is known at time t_n, the solution at time t_{n+1} is obtained by solving (12)

$$\mathbf{T}^{n+1} = (\mathbf{M} + \Delta t\mathbf{K})^{-1}\left(\Delta t\mathbf{F}^{n+1} + \mathbf{M}\mathbf{T}^n\right). \tag{13}$$

This is used to march in time from the initial condition to some final time. In this problem, we consider the a rectangular plate with a circular hole cut out of the center as depicted in Figure 2a. The initial condition is a uniform temperature of 0 throughout the domain. The boundary conditions are $T = 5$ on the left edge

Problem 3 continued on next page...



(a) Plate with hole mesh (b) Heat distribution at final time

Figure 2: Problem 3 - Heat Flow

of the plate and $(\kappa \cdot \nabla T) \cdot \mathbf{n} = 0$ (insulated) on all other boundaries (including the hole). The time interval considered is $[0, 1]$. The solution of heat equation at time $T = 1$ is shown in Figure 2b.
See the *comments* of the code in `heat_fem` for any additional information. In this problem, your tasks are to:

(1) Write and compile your own MEX-file to build

- `build_matrices.m`
  - This function is pretty fast in MATLAB so you won't see significant speedup.
  - I recommend naming the source MEX-file `build_matrices_mex.c` to avoid naming conflict with M-file.

(2) Use MATLAB's Coder to generate MEX-files for

- `create_sparsity_structure.m`
- `get_temp_fem.m`
- I recommend naming the projects `create_sparsity_structure_mex.prj` and `get_temp_fem_mex.prj`, respectively, to avoid naming conflicts with M-files.

Recall from lecture that the bulk of the work required for using the Coder lies in defining the inputs for each function. The comments in `get_temp_fem.m` and `create_sparsity_structure.m` should prove helpful, as well as the code in `setup_problem.m`.

(3) Run `run_heatflow.m` to solve heat flow problem using MATLAB. Make note of execution time of each portion of the code.

(4) Copy `run_heatflow.m` to `run_heatflow_mex.m` and modify it such that it runs the compiled version of the heat flow code. How does the execution time of each component of the program compare?

Problem 4

In this problem, you will gain experience with MATLAB's `parfor` functionality to construct a Pareto front in the context of multiobjective optimization by solving multiple (independent) optimization problems in parallel. In the context of competing objectives, a Pareto front is defined as a set of states such that it is impossible to improve one objective without worsening the other. This problem will use the *nonlinear* 2D truss code from Homework 2. Prior knowledge of truss analysis or mechanics is not necessary. A brief exposition of the truss analysis framework is provided below (repeated from Homework 2).

First, we begin with some terminology and nomenclature. A truss structure will be composed of n_v nodes or vertices connected by n_{el} truss elements. Each element e has an initial length L_e, cross-sectional area A_e, elastic modulus E_e, density ρ_e, and deformed length (i.e. after loads applied) ℓ_e. A truss element can only carry *normal* forces, not shear or moments. This means the direction of the force within an element must align with the element. By definition, the intersection of two truss elements will be a *pinned* connection (otherwise the members would carry shear/moments). The force carried by element e will be denoted $\mathbf{N}_e$. Each vertex will be subject to forces that are applied externally or transferred from the truss elements. Let $\mathbf{f}_i^{\text{int}} \in \mathbb{R}^2$ denote the force on vertex i due to the truss elements, usually called the *internal* force. Also, let $\mathbf{f}_i^{\text{ext}} \in \mathbb{R}^2$ denote the force on vertex i due to external loads. Finally, $\mathbf{u}_i \in \mathbb{R}^2$ denotes the displacement of vertex i induced by the loads.

Boundary conditions for truss elements can either be *displacement* or *force* boundary conditions. Force boundary conditions correspond to external loads, while displacement boundary conditions indicate prescribed displacements. Every node must have *exactly one* boundary condition for *each* degree of freedom (i.e. the $x-$ and $y-$directions).

Static equilibrium at the nodes leads to the governing equation

$$\mathbf{f}^{\text{int}}(\mathbf{u}) = \mathbf{f}^{\text{ext}}. \tag{14}$$

There are several important quantities that will be used in this assignment as objective or constraints.

- Stress
$$\sigma_e = E_e \frac{\ell_e}{L_e} \tag{15}$$
- Weight
$$W = \sum_{e=1}^{n_{el}} \rho_e A_e \ell_e \tag{16}$$
- Compliance
$$C = \sum_{i=1}^{n_v} \mathbf{f}_i^T \mathbf{u}_i \tag{17}$$

In the truss program you are given, there will be three relevant data structures.

- `msh` - contains all mesh information
- `bcs` - contains all boundary condition information
- `mat` - contains all material information (including area)

Given a new vector of elemental areas, you can obtain the solution of the truss problem as follows

```
% Toy mesh
mesh = 'mesh1';

% Determine number of elements in mesh and define vector of areas
```

Problem 4 continued on next page...

```
nel = eval([mesh,'()']);
A = ones(nel,1);

% Setup mesh, BCs, and materials
[msh,bcs,mat] = setup_problem(A,mesh);

% Solve nonlinear truss equations (and sensitivity equations)
[U,dUdA,F,dFdA] = solve_truss(msh,mat,bcs);

% Plot mesh and deformation
% Vary last argument to scale deformation
visualize_truss_loads(msh,bcs,mat,U,10);

% Compute deformed lengths and sensitivities
[l,dldA] = compute_element_lengths(U,dUdA,msh)

% Compute weight and sensitivities
[W,dWdA] = compute_weight(U,dUdA,msh,mat);

% Compute compliance and sensitivities
[C,dCdA] = compute_compliance(U,dUdA,F,dFdA);

% Compute element force and sensitivities
[N,dNdA] = compute_internal_force(U,dUdA,msh,mat);

% Compute stress and sensitivities
[s,dsdA] = compute_stress(U,dUdA,msh,mat);
```

(1) Construct a Pareto front to quantify the tradeoff between weight and compliance minimization for the mesh in Figure 3 (`mesh = 'mesh1'`). This can be done by solving the minimization problem

$$\begin{aligned} \underset{\mathbf{A}\in\mathbb{R}^{n_{el}}}{\text{minimize}} \quad & W(\mathbf{A}) + \theta C(\mathbf{A}) \\ \text{subject to} \quad & |\sigma_e(\mathbf{A})| \leqslant \sigma_{\text{fail}} \\ & 0.01 \leqslant \mathbf{A} \leqslant 1, \end{aligned} \tag{18}$$

for $\theta \in [0, 1]$ (the bounds on θ are usually determined by trial and error, which was done for you). For each optimization problem, use an initial guess of $\mathbf{A} = 0.1$.

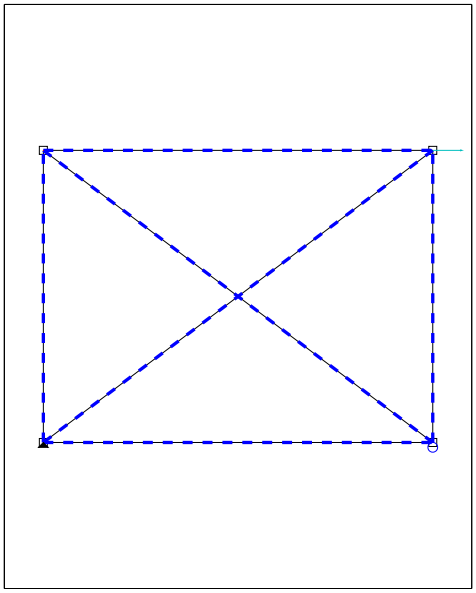
- Use `fmincon` with an interior point algorithm and *user-defined gradients*
- Generate the Pareto front by solving (18) for 50 values of θ in $[0, 1]$ and plotting $W(\mathbf{A}^*(\theta))$ vs $C(\mathbf{A}^*(\theta))$

(2) As all optimization problems are independent, solve in parallel using `parfor`

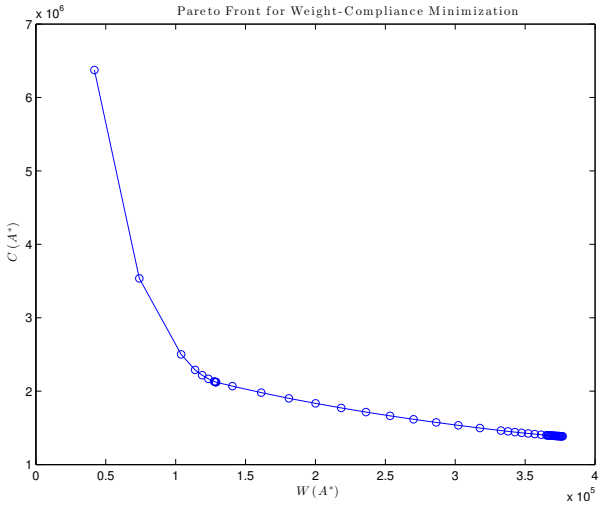
- Use 1, 2, 4, 8 processes and record time for each (do not time plot generation)
  - Do this portion of the problem on `corn.stanford.edu` to ensure Parallel Computing toolbox available and sufficient number of processors to solve problem
- Comment briefly on speedup obtained by using multiple processors

(3) If you are feeling ambitious, try to repeat the previous study using `mesh2`

- I recommend considering only 4 and 8 processors



(a) Mesh



(b) Pareto Front (Weight/Compliance)

Figure 3: Simple Truss (`mesh1`)