CME292: Advanced MATLAB for Scientific Computing

Homework #3
OOP, File Management, & System Interaction

Due: Thursday, October 16, 2014

Instructions

For this problem set, 1 problem out of 2 is required. You are free to choose the problem 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

In this problem, you are provided the completed `polynomial.m` class from Lecture 5. A brief outline overview of the code is shown below. Your task is to create a class `piecewise_polynomial.m` that will generalize `polynomial.m` to piecewise polynomial. A script `test_poly.m` has been provided to demonstrate use of the `polynomial` class.

```
classdef polynomial
    %POLYNOMIAL  Class for handling polynomials and operations

    properties (GetAccess=public,SetAccess=private)
        coeffs=0;
        order =0;
    end

    methods
        function  self = polynomial(arg1)

        function  [poly] = uplus(poly1)

        function  [poly3] = plus(poly1,poly2)

        function  [poly] = uminus(poly1)

        function  [poly3] = minus(poly1,poly2)

        function  [poly] = mtimes(a,b)

        function  [poly] = mpower(poly1,b)

        function  [iseq] = eq(poly1,poly2)

        function  [tf] = iszero(poly)

        function  [poly1] = integrate(poly,const)

        function  [poly1] = differentiate(poly)
```

```
        function  [y] = evaluate(poly,x)

        function  ax = plot_it(poly,x,pstr,ax)

        function [] = disp(poly)
end
```

A piecewise polynomial is a sequence of polynomials defined on neighboring intervals. To make this more precise, consider the domain $\Omega = [x_1, x_{N+1}] \subset \mathbb{R}$ (subset of the real number line) and define $x_i \in \Omega$ for $i = 2, \ldots, N$. Then, $\mathcal{I}_i = [x_i, x_{i+1}]$ for $i = 1, \ldots, N$ defines a partition of the domain Ω. In the remainder of this, the endpoints of these intervals (x_i) will be called *knots*. A piecewise polynomial $p(x)$ over Ω is defined as

$$p(x) = \begin{cases} p_1(x) & \text{for } x \in [x_1, x_2] \\ p_2(x) & \text{for } x \in [x_2, x_3] \\ \vdots & \vdots \\ p_N(x) & \text{for } x \in [x_N, x_{N+1}] \end{cases} \tag{1}$$

where $p_i(x)$ is a polynomial. In this problem, your tasks are

(1) Setup the class for `piecewise_polynomial.m`

- `piecewise_polynomial` should have three properties
  - `xi` - `double` array containing *knots*
  - `npoly` - `double` scalar containing number of polynomials (`= length(xi)-1`)
  - `polys` - `polynomial` array of length `npoly`
- the constructor of `piecewise_polynomial` should
  - accept zero, one or two argument
  - if zero arguments passed, create the *empty* piecewise polynomial (all properties set to `[]`)
  - if one argument passed of type
    - `piecewise_polynomial` - copy the properties of the input to the new object
    - `double` - assume the argument is `xi` and set the properties accordingly (`polys` empty)
  - if two arguments passed
    - assume the first is a `double` containing the knots `xi`
    - assume the second is a `polynomial` array containing the polynomials of the piecewise polynomial

(2) Implement the following methods, where $p(x)$ and $q(x)$ are piecewise polynomials

- `uplus`: $p(x) \mapsto +p(x)$
- `plus`: $p(x), q(x) \mapsto p(x) + q(x)$
- `uminus` $p(x) \mapsto -p(x)$
- `minus`: $p(x), q(x) \mapsto p(x) - q(x)$
- `mtimes`: $p(x), q(x) \mapsto p(x)q(x)$
- `mpower`: $p(x), c \mapsto p(x)^c$
  - c is a non-negative integer
- `eq`: $p(x), q(x) \mapsto \{0, 1\}$
  - return `true` if p and q are equal for all $x \in \mathbb{R}$ (coefficients identical)

Problem 1 continued on next page...

- `iszero`: $p(x) \mapsto \{0, 1\}$
  - return `true` if $p(x) = 0$ (all coefficients are zero)
- `integrate`: $p(x) \mapsto \int p(x)dx$
- `differentiate`: $p(x) \mapsto \frac{dp}{dx}(x)$
- `evaluate`: $p(x), \mathbf{v} \mapsto p(\mathbf{v})$
  - should accept vector inputs evaluate $p(x)$ at each entry and return the output as a vector
- `plot_it`
  - plot $p(x)$ over some specified
- *Hint: You should not have to implement any polynomial routines yourself. Your class should store an array of* `polynomial` *objects which have all polynomial functionality required for this assignment implemented. Use the methods in the* `polynomial` *objects to perform polynomial operations.* `piecewise_polynomial.m` *should really just be a* wrapper *for* `polynomial.m` *to implement the piecewise functionality.*

(3) Use `piecewise_polynomial.m` to create the following piecewise polynomials over the domain $\Omega = [-2, 2]$. The code for this part of the problem should be in your driver (display output by not including semicolons `;`).

$$p_1(x) = \begin{cases} x^3 - 2x^2 & \text{for } x \in [-2, -1] \\ 0 & \text{for } x \in [-1, 1] \end{cases} \quad p_2(x) = \begin{cases} 0 & \text{for } x \in [-2, -1] \\ x & \text{for } x \in [-1, 1] \\ 0 & \text{for } x \in [1, 2] \end{cases} \quad p_3(x) = \begin{cases} 0 & \text{for } x \in [-2, 1] \\ x^2 - 3 & \text{for } x \in [1, 2] \end{cases} \tag{2}$$

- Compute and plot $p_4(x) = p_1(x) + p_2(x) + p_3(x)$
- Compute and plot $p_5(x) = p_1(x) - p_2(x) - p_3(x)$
- Compute and plot $p_6(x) = p_4(x)p_6(x)$
- Compute and plot $p_7(x) = p_2(x)^3$
- Compute and plot $p_8(x) = \int p_4(x)dx$
- Compute and plot $p_9(x) = \frac{dp_8}{dx}(x)$
  - Are $p_4(x)$ and $p_9(x)$ equal?

Problem 2

In this problem, you will gain experience with reading/writing files and making system calls by extending the class `dsg_elem_def` seen in lecture. Before defining the tasks for this problem, some background material is necessary. A design element is one of many concepts from the field of *shape parametrization*, commonly used in shape optimization, for using control nodes to deform a surface. Namely, it allows one to parametrize the *shape* of a surface as a function of the control nodes. In the remainder, we use the following definitions

- *design element nodes* - the nodes of the design element that will be used to *deform* the *surface* contained inside the design element (also known as *control nodes*)
- *surface nodes* - the nodes comprising the surface (to be deformed using the control nodes)

We will use the SDESIGN software to convert displacement of the control nodes into displacement of the *surface* nodes. As the SDESIGN software is not open source, I have compiled SDESIGN on corn and given you the path to the executable (/srv/zfs01/user_data/mzahr/sdesign.Linux.opt). *Therefore, you must use corn for this assignment.* You can call SDESIGN by

```
/srv/zfs01/user\_data/mzahr/sdesign.Linux.opt input_file.sdesign
```

where `input_file.sdesign` is an SDESIGN input file (notice that this is a system call - therefore to call SDESIGN from MATLAB, use `system`). The input file will define the control nodes and link them to the surface (already done for you), as well as the *displacement* of each control node. The output of SDESIGN will be two files: `input_file.vmo` and `input_file.der`. The output file `input_file.vmo` will contain the x, y, z displacement of each surface node (ignore `input_file.der` for this problem) - the file will contain a few header lines, followed by n_{surf} rows (one for each surface node) of 3 columns each (for the x, y, z displacements).

In its present form, `dsg_elem_def` accepts an input file (SDESIGN format [1, 2, 3] - it is not necessary to understand the SDESIGN format for this problem) that defines a *design element* and contains the name of a file defining the points on some surface (in our case, this surface is an airfoil, see Figure 1). `dsg_elem_def` plots the nodes of the design element (in red circles) and the surface (thick, blue line) as well as defines callback routines that allows one to move the design element nodes with the mouse (click on node, drag it to new location, and release) – try it. You will notice that moving the control nodes *does not affect the surface.* For this assignment, your task is to complete `dsg_elem_def` such that one can use the mouse to move the design element nodes *and cause the surface to deform.* All of the method that need to be implemented are already in `dsg_elem_def` (but some are empty). As usual, if you do not want to follow the structure of the starter code provided in `dsg_elem_def`, feel free to start from scratch.

The `dsg_elem_def` class has the following methods (some of which you need to complete)

- *Hint: Comments are provided throughout starter code (`dsg_elem_def.m`) – be sure to read them.*
- `dsg_elem_def` – the constructor – accepts a string `sdesign_template_in` that will contain the filename of a SEDESIGN template file (`naca0012.sdesign` in our case). This is called a *template* as it defines only the control node locations and the surface, but there are only *placeholders* for the control node *displacements* (this is necessary as the displacement of the control nodes are not known until the user starts moving them in the figure). These placeholders are named `<S0>`, `<S1>`, `<S2>` and they must be replaced by the actual value of the design element node displacements (when they are known). Specifically, `<S0>` and `<S1>` must be replaced by the x, y displacement of the first control node, `<S2>` and `<S3>` must be replaced by the x, y displacement of the second control node, and so on.
- `draw_surface` – this should set the `XData` and `YData` of the surface handle (`surfHan`) to the appropriate values.
- `select_node` – sets the `UserData` property of the control node to `true`. Used to identify which node is being moved (callback).
  - This was completed for you.
- `release_button` – this is the callback routine to be executed when the mouse button is release. It currently only moves the control node that is currently selected (by a mouse button down). You need to extend this method such that it
  - writes the control node displacement to the SDESIGN input file (more on this in `write_sdesign` below)
  - calls SDESIGN with the input file (system call)
  - reads the `naca0012.vmo` (filename stored in `displace_fname`) that contains the displacement of the surface nodes (for this problem, we only care about the x, y displacements, i.e. first two columns)
  - draw the new surface location

- `write_sdesign` – needs to write the displacement of the control nodes to the SDESIGN file. The way that I recommend doing this is by looping through the SEDESIGN template file line-by-line, replacing the *placeholders* (`<S0>`, `<S1>`, ...) with their appropriate values (if they exist in a given line) – recall the placeholders should be replaced by the x or y *displacement* of a control node – and copying the modified line to the SDESIGN input file (`sdesign_iter`).
- `run_sdesign` – make a system call to run SDESIGN (path to SDESIGN stored in `sdesign_exec`) with the input file (stored in `sdesign_iter`)
- `get_design_element_nodes` – reads the design element nodes from SDESIGN file
  - This was completed for you.
- `get_surfacenode_file` – reads the name of the file containing the surface nodes from the SDESIGN file
  - This was completed for you.
- `read_surfacenodes` – reads the coordinates of the surface nodes (undeformed)
- `read_surfacenode_disp` – reads the SDESIGN output file (`naca0012.vmo` stored in `displace_fname`) to extract the *displacement* of the surface nodes

```
classdef  dsg_elem_def < handle

    properties (SetAccess=private,GetAccess=public)
        figHan  = []; % Figure handle
        axHan   = []; % Axes handle
        ptHan   = []; % Node handles
        surfHan = []; % Surface handle

        sdg_name  = []; % Name of SDESIGN file
        sdg_nodes = []; % Design element nodes (control nodes)
        surf_nodes= []; % Location of surface nodes on airfoil
        sdg_nodes_disp=[]; % Displacement of design element nodes
    end

    properties (Hidden)
        sdesign_template = '';
        sdesign_exec     = '/srv/zfs01/user\_data/mzahr/sdesign.Linux.opt';
        sdesign_iter     = '';
        surfnodes_fname  = '';
        displace_fname   = '';
    end

    methods
         function [self] = dsg_elem_def(sdesign_template_in)
         function [] = draw_surface(self,x,y)
         function [] = select_node(self,hobj,¬)
         function [] = release_button(self,hobj,¬)
         function [] = set_node_position(self,node_num,pos)
         function [] = write_sdesign(self)
         function [] = run_sdesign(self)
    end
    methods (Static=true)
        function [dsgnodes]  = get_design_element_nodes(sdesign_file)
        function [surfnodes] = get_surfacenode_file(sdesign_file)
        function [nodes] = read_surfacenodes(fname)
```

Problem 2 continued on next page...

```
        function [disp] = read_surfacenode_disp(fname)
    end
end
```

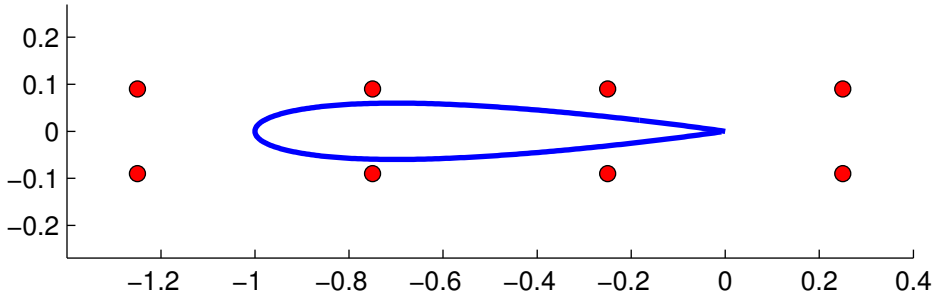

Figure 1: Airfoil surface and deformation control nodes

References

[1] K. Maute and M. Raulli, "Fem—optimization module and sdesign user guides, 0.," 2006.

[2] K. Maute, M. Nikbay, and C. Farhat, "Sensitivity analysis and design optimization of three-dimensional nonlinear aeroelastic systems by the adjoint method," *The International Journal for Numerical Methods in Engineering*, vol. 56, pp. 911–933, 2003.

[3] K. Maute, M. Nikbay, and C. Farhat, "Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems," *AIAA Journal*, vol. 39, pp. 2051–2061, 2001.