CSC 336H1S Assignment Four Due 3:00pm, April 6, 2011

Please staple a copy of the assignment cover sheet to the front of your assignment solutions. This cover sheet is available on the course website.

See the Course Information Sheet for the policy on late assignments.

For questions where you are asked to write a program, you **must** hand in your program and all output results, including plots.

1. Consider the function $f(t) = \int_0^t e^{-x^2}\,dx$ for $0 \le t \le 1$.
   (a) Compute the polynomial $p_2(t)$ that interpolates $f(t)$ at $t = 0$, 0.5 and 1. Next calculate an approximation to $f(0.336)$ by evaluating the interpolant p_2 at $t = 0.336$. Determine the absolute error in the approximation. You may use `matlab`'s `erf` function to determine the "true" value.
   (b) Determine a good upper bound for the error in approximating $f(t)$ by $p_2(t)$ on the interval $[0, 1]$.

2. Interpolating the data points

t	0	1	4	9	16	25	36	49	64
y	0	1	2	3	4	5	6	7	8

   should give an approximation to the square root function.
   (a) Use `matlab`'s `polyfit` function to construct the polynomial of degree eight that interpolates these nine data points. Let's call this polynomial $p_8(t)$. To see how well the interpolant fits the data, graph the functions $y = p_8(t)$ and $y = \sqrt{t}$ in a single plot using `matlab`'s `plot` function. Since `matlab` uses piecewise linear interpolation when drawing plots, construct your graphs by evaluating $p_8(t)$ and $\sqrt{t}$ at 1001 evenly spaced points in the interval $[0, 64]$. In a separate figure, graph the absolute error in the approximation, $|\sqrt{t} - p_8(t)|$, over the interval $[0, 64]$. Use the `matlab` function `semilogy` to produce this graph so that you can observe the size of the error.
   The `matlab` function `sqrt` calculates a square-root.
   (b) Use `matlab`'s `spline` function to compute a not-a-knot cubic spline interpolant of the same data. Let's call this piecewise polynomial $S_{\text{n-a-k}}(t)$. Graph $y = S_{\text{n-a-k}}(t)$ and $y = \sqrt{t}$ in a single plot, using the procedure used in part (a). Also graph the absolute error in the approximation.
   (c) Which one of the two interpolants is more accurate over most of the interval $[0, 64]$?
   (d) Which one of the two interpolants is more accurate over the interval $[0, 1]$? (You may find it helpful to plot the error over $[0, 1]$.)

3. Given a set of data $D_0 = \{(t_i, f_i)\}_{i=1}^n$, it is desired to find an approximation for $f''(t)$. This is to be done by generating a new set of data $D_2 = \{(t_i, f_i'')\}_{i=1}^n$ and then interpolating it with a cubic spline. Two possible ways of generating the set D_2 are as follows.

(a) Interpolate D_0 with a cubic spline $S(t)$ and evaluate its second derivative at each t_i. That is, $f_i'' = \left(\dfrac{d^2S}{dt^2}\right)_{t=t_i}$.

(b) Use a "spline-on-spline" technique. Interpolate the data D_0 as before but, this time, generate the data $D_1 = \{(t_i, f_i')\}_{i=1}^n$, where $f_i' = \left(\dfrac{dS}{dt}\right)_{t=t_i}$. Now interpolate this set of data with another cubic spline $\widehat{S}(t)$ and then generate D_2 by setting $f_i'' = \left(\dfrac{d\widehat{S}}{dt}\right)_{t=t_i}$.

Compare the accuracy of these two methods with the following data D_0 :

t_i	0	1	2	3	4	5
f_i	0.0000	0.8415	0.9093	0.1411	−0.7568	−0.9589
t_i	6	7	8	9	10	
f_i	−0.2794	0.6570	0.9894	0.4121	−0.5440	

The data is derived from the function $f(t) = \sin(t)$ so it is easy to check the accuracy of the approximation for $f''(t)$.

A good comparison would include plots of $f''(t)$ and the two cubic spline approximations to $f''(t)$, and also plots of their error. The plots can be generated by sampling the functions at say 1001 equally spaced points in the interval $[0, 10]$ and using `matlab`'s `plot` function. **In addition**, report the maximum observed errors in the approximations over $[0, 10]$.

Use `matlab`'s `spline` function to compute not-a-knot cubic splines. You will want to use the `spline` function in its `pp = spline(t,y)` form, so that the cubic spline is returned to you as a piecewise-polynomial that you may differentiate. Use the functions `unmkpp` and `mkpp` to manipulate the representation of the piecewise polynomials, and the function `ppval` to evaluate piecewise polynomials. You may use the function `polyder` to differentiate polynomials. Use `matlab`'s `help` command to learn more about these functions. You may also want to use the `type` command to see how these functions are implemented in `matlab`.

4. Consider the integral

$$\int_{-1}^{1} \sqrt{|x|}\, dx.$$

Use `matlab`'s `quad` function to approximate this integral for `TOL = 1.0e-2, 1.0e-4, 1.0e-6` and `1.0e-8`. What is the absolute error in each of the approximations? At how many points was the integrand evaluated for each setting of `TOL`? Comment on your observations.

For each of the four `TOL` settings, generate a plot that shows the integrand together with the values of `x` where the integrand was evaluated. Comment on your observations.

One way to keep a record of the integrand evaulations (for plotting after the call to `quad`) is to use `global` variables. If you put the statement

```
global xNodes fAtxNodes;
```

in both your main `matlab` script and in your `matlab` integrand function, and then add the lines

```
% append x and f to global save variables
%  (assumes the argument to function is named x
%   and the integrand values are named f)
xNodes = [ xNodes x ];
fAtxNodes = [ fAtxNodes f ];
```

at the end of your `matlab` integrand function, then the evaluation points will be available in your main `matlab` script after the `quad` function has returned. You will want to `clear` the variables `xNodes` and `fAtXnodes` before each call to `quad`.

Finally, repeat the above, but this time use the integration function `quadl` instead of `quad`. Comment on any observations.

Execute the `matlab` commands `help quad`, `help quadl` and `help function` to learn about `quad` and `quadl`, and declaring functions in `matlab`. See also the results from `help plot` to learn about plotting unconnected points.