MATH 408 — Mini-Project 1 — Due: Tuesday, September 8, 2020

1. (a) [3pts] Submit (in your PDF write-up) two different fourth-degree polynomial approximations to the function $u(x) = \mathrm{e}^{x/2}$:
   - using a *Taylor polynomial* T_4 centered at $x_0 = 5$, and
   - using a *Lagrange interpolating polynomial* P_4 with data sampled at $x_1 = 1$, $x_2 = 3$, $x_3 = 5$, $x_4 = 7$, and $x_5 = 9$.

   (b) [2pts] Use the MATLAB grader to plot the two polynomial approximations from Part (a) together with the exact function on the interval $[0, 12]$.

   (c) [1pt] Derive and submit (in your PDF write-up) the (absolute) error *estimates* for the two approximations when $x \in [0, 12]$.
   Hint: Use $\max_{\xi\in[0,12]} |u^{(5)}(\xi)|$ to obtain your estimates.

   (d) [1pt] Use the MATLAB grader to plot the (natural) logarithms of the two error estimates from Part (c).

   (e) [1pt] Use the MATLAB grader to plot the (natural) logarithm of the *actual* (absolute) errors together with their error estimates (from Parts (c) and (d)) for these two approximations on $[0, 12]$ and compare with the error estimates by computing the maximum of the ratios of values in the vector of error estimates to the values in the vector of actual errors (Hint: use elementwise division in MATLAB).

   (f) [2pts] Comment (in your PDF write-up) on the relation between the error estimates and the actual errors.

2. [8pts] If $x_1, x_2 \ldots, x_{n+1} \in \mathbb{R}$ are distinct, then the Lagrange basis polynomials L_k were defined in class to be

$$L_k(x) = \frac{1}{a_k} \prod_{\substack{j=1 \\ j\neq k}}^{n+1} (x - x_j), \qquad \text{where } a_k = \prod_{\substack{j=1 \\ j\neq k}}^{n+1} (x_k - x_j). \tag{1}$$

   These functions are the unique polynomial interpolants of degree n to the values 1 at x_k and 0 at all the other x_j, $j \neq k$.

   Show that the derivative of L_k is given by

$$L_k'(x) = L_k(x) \sum_{\substack{j=1 \\ j\neq k}}^{n+1} \frac{1}{(x - x_j)}.$$

   Hint: Take the logarithm of (1) and then differentiate.

3. [6pts] Prove that if $u(h) = \mathcal{O}(v(h))$ as $h \to 0$, then $\alpha u(h) = \mathcal{O}(v(h))$ as $h \to 0$ for any $\alpha \in \mathbb{R}$.

4. [8pts] Derive the FD approximation $D^2u(\bar{x})$ for $u''(\bar{x})$ from the notes by
   - forming the quadratic interpolating polynomial p to u at three points x_1, x_2, x_3,
   - differentiating p twice,
   - and then setting $x_1 = \bar{x} - h$, $x_2 = \bar{x}$ and $x_3 = \bar{x} + h$ in $p''(\bar{x})$.

5. Use of `fdcoeffF.m`

   (a) [3pts] Use the MATLAB code `fdcoeffF.m` (you can find it on Canvas or in the MATLAB grader) to compute the coefficients of a fourth-order accurate finite difference approximation to $u''(\bar{x})$ based on 5 equally spaced points,

   $$u''(\bar{x}) = c_1 u(\bar{x} - 2h) + c_2 u(\bar{x} - h) + c_3 u(\bar{x}) + c_4 u(\bar{x} + h) + c_5 u(\bar{x} + 2h) + \mathcal{O}(h^4).$$

   Hint: Call `fdcoeffF.m` with several different values of h to see how the coefficients $\boldsymbol{c} = [c_1, c_2, \ldots, c_5]^T$ depend on h.

   (b) [3pts] Use the MATLAB grader to test this finite difference formula to approximate $u''(1)$ for $u(x) = \sin(2x)$ with values of h from the array `hvals = logspace(-1, -4, 13)`. Make a table of the absolute error (i.e., the absolute value of the difference between your approximation from part (a) and the exact value $u''(\bar{x})$) vs. h for several values of h and compare against the predicted $\mathcal{O}(h^4)$ error. You may want to look at `FDApproximationRates.m` for guidance on how to make such a table.

   (c) [2pts] Use the MATLAB grader to produce a `loglog` plot of the absolute error vs. h.

   Note: You should observe the predicted accuracy for larger values of h. For smaller values, numerical cancellation in computing the linear combination of u values impacts the accuracy observed.

6. Consider the following system of second-order initial-value problems:

   $$u''(t) = -\frac{u(t)}{(u(t)^2 + v(t)^2)^{3/2}}, \qquad u(0) = 1,\ u'(0) = 0, \tag{2}$$

   $$v''(t) = -\frac{v(t)}{(u(t)^2 + v(t)^2)^{3/2}}, \qquad v(0) = 0,\ v'(0) = 1. \tag{3}$$

   These are Newton's equations of motion for the two-body problem, also known as the Kepler problem. Here the pair $(u(t), v(t))$ describes the trajectory of one of the bodies at time t. If we let t range from 0 to 2π, then the solution will be a circle.

   (a) [3pts] Transform the given problem into an appropriate system of first-order initial-value problems.

   (b) [2pts] Use the MATLAB grader to write a function `Twobody.m` that calculates the (vectorized) right-hand side of the system obtained in the previous step. The function should be of the form

   `function yprime = Twobody(t, y)`,

   where `yprime` and `y` are appropriate column vectors.

   (c) [2pts] Use the MATLAB grader to write a driver script that solves the equations of motion from Parts (a) and (b) with MATLAB's built-in ODE solver `ode23`. The calling sequence for this (and other) built-in ODE solver(s) is of the form

   `[t, y] = ode23(f, [tstart tend], y0)`

   with `f` the name of the right-hand side function (your function `Twobody`), `tstart` and `tend` starting and ending t-values, and `y0` a (column vector) of initial conditions.

(d) [3pts] Plot the computed solution in the uv-plane. For debugging/testing purposes you should also include plots of the components of the vector `y` against t. This can be accomplished in a single plot by using MATLAB syntax such as

```
subplot(2,2,1), plot(t,y(:,1))
subplot(2,2,2), plot(t,y(:,2))
subplot(2,2,3), plot(t,y(:,3))
subplot(2,2,4), plot(t,y(:,4))
```