ACM 11: Homework 2

Assigned Wednesday, Oct 8 2008. Due by Wednesday noon, October 15. 50 points.

Create a script file entitled `Firstname_Lastname_2.m`, where you replace `Firstname` and `Lastname` with your first and last names, respectively. Save your solution to this problem set in this **single** file; when finished, follow the directions on the course site to submit the file by FTP. If you need to submit revisions of your solution, follow the instructions provided on the course site.

Begin the script file with a title comment containing your name and email address. This information should also be displayed in the command window when your script runs. Output the answers to all questions to the command window, properly labeled, including the problem number; if you output a number, indicate what has been measured, and if you provide an explanation, indicate what phenomenon you are explaining. All values in the command windows should be the result of deliberate display statements (e.g. `disp` or `fprintf`) and not MATLAB default evaluations! This documentation is required! Put simply, your output should be easy to read and understandable without reference to your code. Your grade will be almost entirely determined by the script's **output**, not the code itself.

Each graph that is requested, or which you found useful while solving a problem (e.g. one which supports an explanation), should be created in a new figure (using the `figure` command), properly labeled, and referred to in your output by figure number. Be sure to put `clear`, `clc`, and `close all` as the first executable instructions in `main.m`.

1. Run the following commands. The sum of the variable **v1** is the 100-th order Taylor approximation of e^{-10} using double precision arithmetic, while the sum of **v2** and **v3** are the approximations using single precision arithmetic. Report the absolute errors for each approximation (using `exp` to find the true answer) and explain the results. 5 points.

   ```
   x = 0:100;
   ```

   (a) `v1=(-10).^x./factorial(x); sum(v1)`

   (b) `v2=single( (-10).^x./factorial(x) ); sum(v2)`

   (c) `v3=single( (-10).^x )./factorial(x); sum(v3)`

2. Fair dice. Download the file `diceData.mat` from the course website and load it into MATLAB using the `load` command. There are two vectors, `die1` and `die2`, which are the results of 5000 rolls of two dice. One die is a fair die, meaning that the integers $1, 2, \ldots, 6$ all have equal probability. The other die is not a fair die.

   Which die is fair, `die1` or `die2`? Explain what lead you to your conclusion, and what MATLAB commands you used. Hint: you might find the functions listed in `help datafun` useful. 5 points.

3. Pseudo-random number generators. Designing a good pseudo-random number generator is not easy, and MATLAB's default generators are some of the best. Go to the course website and download the file `randomNumbers.mat` and load it into MATLAB . The two variables, `rand1` and `rand2`, are both pseudo-random number sequences (meant to be like uniform random numbers in the range $[0, 1]$), but from different algorithms. One algorithm is the poorly designed RANDU algorithm, and the other is MATLAB's default. Which is which, and why? Hint: you may use the internet on this problem to research the weaknesses of RANDU. In MATLAB , you might find the command `plot3` useful. If, for example, you make a plot that is evidence for your conclusion, then include this plot with your homework. 5 points.

4. Vectorization and anonymous functions. Consider the function

$$f(x) = \begin{cases} \sin(2x) & \text{if} \quad 0 < x < 1 \\ \exp(-x^2) & \text{if} \quad 1 \leq x \leq 2 \\ 0 & \text{if} \quad x \leq 0 \quad \text{or} \quad x > 2 \end{cases}$$

Implement f in MATLAB using an anonymous function, and without using any `if` statements. Display your code for f (using `disp(f)`), as well as the output of f given the following input vector: `x = [-.3, .3, 1.5, 2, 200]`. Your code should be able to handle vector inputs, i.e. no need for `for` loops. 5 points.

5. Interpolation.

   (a) Trigonometric Polynomials. Consider the function $f(x) = \sin(4\pi x) \cdot \sin(\pi x)$ on the domain $x \in [a, b]$, and suppose we are interested in approximating f on a uniformly spaced grid with 512 points. To make this grid, use a command like `dX = (b-a)/512; grid = a:dX:b-dX` (we do NOT want to include the endpoint). Using $a = 0$ and $b = 2$, evaluate f on this grid using an anonymous function, and use this result as the true answer. For a brief introduction to interpolation, see Moler's online textbook.

      i. Now, subsample f on the smaller grid defined by `dX = (b-a)/N; a:dX:b-dX` (for $a = 0$ and $b = 2$ again), for some N smaller than 512. Using interpolating Fourier polynomials, interpolate (using the data from the small grid) onto the 512-point grid. In MATLAB , this can be done with the `interpft` command. Compute (and record) the error using the l_∞ norm on the large 512-point grid. How many points N are necessary to acheive nearly perfect reconstruction (i.e. the norm of the error is less than 10^{-14})? Is this surprising? 5 points.

      ii. Change the domain so that $b = 2.2$. Do the results from the previous question still hold? For $N = 50$, make a plot of the error on the 512-point grid, and give the plot an informative title. 5 points.

      iii. Now, interpolate using polynomials (instead of trigonometric polynomials), using both `pchip` and `spline`. These can also be called using `interp1`. Be careful: the syntax is different than for `interpft`. With $b = 2$ again, and $N = 20$, interpolate the same function. Record the l_∞ errors. Also, make a plot showing the true answer, the result given by `pchip`, and the result given by `spline`; this should all be in the same figure, and each line should have a different color. Use the command `legend` to label the different results. Are the results better or worse than with `interpft`? 10 points.

      iv. BONUS: can you increase the accuracy of the `pchip` and `spline` methods by *padding* the input with zero?

   (b) Polynomials.

      i. Consider a polynomial $p(x)$ with roots $\{.4, .7\}$, and consider the domain $x \in [0, 1]$; this polynomial is unique up to a multiplicative constant. Use MATLAB's `poly` function to create this polynomial (using MATLAB's default scaling constant). Evaluate p on a 512-point grid (created similarly via the method shown in the previous problem). To do this, type `help polyfun` to see the available MATLAB functions. Interpolate back to 512 points using `pchip` and `spline` for $N = 4$ and report the l_∞ errors. Is this surprising? 5 points.

      ii. Repeat the previous question, but for a polynomial $p(x)$ with roots $\{.2, .3, .8, .88\}$ (again, use MATLAB's default scaling), and use $N = 10$. Report the l_∞ errors. 5 points.