Homework 6

Due: Wednesday 11:59pm, Dec 11, 2024, on Gradescope

Notes: Please read the syllabus for information on programming expectations, the late homework policy, and the collaboration policy. Please submit both the written and coding portions of your homework on Gradescope. The written portion should be submitted as a PDF (prepared however you like, but LaTeX is excellent), with plots and tables in the appropriate places. Results and plots requested in the problems should be placed in your write-up—you should not simply refer to your code. Please mark the location of each problem in your Gradescope submission—this makes it much easier for us to grade.

Plotting and formatting: Put thought into presenting your data and results in the clearest way possible. Think about what information you want to convey, and how to convey it most effectively. Sometimes, using a table can be useful, but at other times data is best presented using a plot. All tables and plots should be accompanied by some discussion of what we are intended to learn from them.

Choose proper ranges and scales (e.g., semilogx, semilogy, loglog), always label axes, and give meaningful titles. Make sure your axis numbering and labels are large enough to be readable—getting this right will require some exploration in MATLAB, or whatever language you are using. If you do print numbers, in MATLAB for example, use `fprintf` to format the output nicely. Use `format compact` and other format commands to control how MATLAB prints things. When you create figures using MATLAB (or Python or Julia), please try to export them in a vector graphics format (.eps, .pdf, .dxf) rather than raster graphics or bitmaps (.jpg, .png, .gif, .tif). Vector graphics-based plots avoid pixelation and thus look much cleaner.

Code: Try to write clean, concise, and easy-to-read code, with meaningful variable names. Your code must be well-commented. Every line of code should be explained by comments inside the code, unless it is absolutely self-explanatory. Commenting your code is like "showing your work", and grading will take this into account in a similar manner.

You should submit two code files for this assignment: `adapcheb_nodes.m` and `adapcheb_interp.m`.

---

1. **[5 pts]** Derive the Chebyshev nodes and the barycentric interpolation formula for Chebyshev interpolation on the interval $[a, b]$. The nodes and barycentric weights can be written in terms of the nodes x_k and barycentric weights w_k on the standard interval $[-1, 1]$.

2. This problem will explore a simple adaptive Chebyshev interpolation scheme on the interval $[0, 1]$. Note that it will require using your result from the previous problem. Consider an adaptive piecewise Chebyshev interpolant which is refined *dyadically* towards $x = 0$: we take panel endpoints $x_0 = 0$ and $x_k = 2^{-(n-k)}$ for $k = 1, \ldots, n$. Then we place p Chebyshev nodes on each panel $[x_{k-1}, x_k]$. The resulting piecewise Chebyshev interpolant can resolve certain functions which are localized near $x = 0$ at

different scales, since the panels become smaller and smaller approaching that point. Indeed, it is often important to resolve multiple functions with features localized at different scales on the same grid.

(a) [**10 pts**] Write a MATLAB function `adapcheb_nodes` which takes as input n and p, and outputs the np nodes for this composite Chebyshev grid (you can store these either in a vector, or in a $p \times n$ matrix, depending on what you find most convenient for the subsequent problems). Submit a plot showing the nodes for $n = 6$, $p = 4$. Submit your code in a file `adapcheb_nodes.m`.

(b) [**10 pts**] Write another function `adapcheb_interp` which takes as input n, p, a vector or matrix of the np values of a function $f(x)$ at the nodes produced by `adapcheb_nodes`, and a point $x \in [0, 1]$, and outputs the value of the piecewise Chebyshev interpolant of f at x. Your function should find the panel containing x (Hint: this can be done with simple operations rather than a brute force search), and then perform barycentric interpolation in that panel. Submit your code in a file `adapcheb_interp.m`.

(c) [**5 pts**] Consider the functions $f_1(x) = e^{-x}$ and $f_2(x) = e^{-10000x}$. To resolve $f_2(x)$, we can take n sufficiently large so that the smallest panel width is approximately $1/10000$, since $f_2(x)$ decays on that scale. How large should we choose n?

(d) [**10 pts**] Take this value of n, and $p = 16$. Plot f_1 and f_2, along with their piecewise Chebyshev interpolants, on a single plot. Then, on another plot, plot the absolute error of your piecewise Chebyshev interpolants of $f_1(x)$ and $f_2(x)$ versus x, using a dense grid of 10000 points on $[0, 1]$ (you can use a loglog plot in order to resolve the error near $x = 0$). Finally, plot the error of a global Chebyshev interpolant of f_1 and f_2 of the same total number np of nodes, on the same grid (you can simply use the same code, replacing n by 1 and p by np). (You might find the `subplot` function useful to make a single figure of three plots.) Interpret and discuss the results.

3. (a) [**5 pts**] Using Taylor expansions, derive a fourth-order accurate central finite difference scheme to compute $f'(x)$ for some smooth $f(x)$. A central difference scheme uses values of $f(x)$ on a grid of points $x \pm kh$ placed symmetrically around x. (Hint: your scheme will require values $f(x \pm h)$ and $f(x \pm 2h)$.)

   (b) [**5 pts**] Show that the same scheme can be obtained using Richardson extrapolation of the second-order central difference scheme.

   (c) [**5 pts**] Estimate the rounding error of this scheme, and use it to obtain a rough estimate of the minimum error achievable by the scheme.

   (d) [**10 pts**] Consider $f(x) = \sin(x)$. Denote the finite difference step size by h. Generate a plot of the error of your finite difference scheme in computing $f'(0.1)$ versus h. Demonstrate fourth-order accuracy by including an additional line on your plot. How do your results compare to your prediction?

4. This problem will explore Gauss-Legendre quadrature. Surprisingly, MATLAB does not have a built-in function to generate the Gauss-Legendre nodes and weights, so we will

use an open-source code, which I have made available in the folder `lgwt` in the course Github repository. You will need to put the file `lgwt.m` in the same directory as your code for this problem. Read the documentation at the top of that file to understand how to use the function `lgwt`.

(a) **[10 pts]** Compute $\int_0^1 x^k \, dx$ using a Gauss-Legendre quadrature of $n = 3$ nodes for $k = 0, \ldots, 10$, and report the error for each k. Report your observation, and explain.

(b) **[5 pts]** It can be shown that $\int_{-\infty}^{\infty} e^{-x^2} \, dx = \sqrt{\pi}$. We truncate the integral to $\int_{-6}^{6} e^{-x^2} \, dx$; the error of this truncation is below the double machine precision. Plot the error of the integral approximated using an n-node Gauss-Legendre quadrature on $[-6, 6]$ for increasing n. Do you observe algebraic, exponential, or super-exponential convergence? How many quadrature nodes are needed to achieve an error below 10^{-12}?

5. **[10 pts]** Consider the n-node left-endpoint rule approximation of $\int_{-\pi}^{\pi} \exp(\cos(x)) \, dx$. The exact integral is given by the MATLAB command `2*pi*besseli(0,1)`. Plot the error versus n. Why is the convergence so rapid, even though we are only using the simple left endpoint rule?

6. **[10 pts]** This problem will explore the method of Monte Carlo integration. It is typically used for very high-dimensional integrals, for which traditional quadrature rules are intractable, but we will demonstrate the method for $\int_{-\infty}^{\infty} e^{-x^2} \, dx = \sqrt{\pi} \approx \int_{-6}^{6} e^{-x^2} \, dx$. The basic Monte Carlo method simply approximates an integral as

$$\int_a^b f(x) \, dx \approx \frac{b-a}{n} \sum_{i=1}^{n} f(x_i),$$

where x_i are uniform random numbers on the interval $[a, b]$. Note that a vector of n uniform random numbers on $[a, b]$ can be generated using the MATLAB command

```
x = a + (b-a)*rand(n,1);
```

Plot the error of the Monte Carlo approximation of $\int_{-6}^{6} e^{-x^2} \, dx$ versus n, on a `loglog` plot, for $n = 10^k$ for $k = 1, \ldots, 8$. Also plot a curve $C/\sqrt{n}$, demonstrating the roughly $\mathcal{O}(n^{-1/2})$ convergence of the Monte Carlo method (Note: the convergence might be a bit erratic, due to the randomness). What is the error for $n = 10^8$ points? Compare this with your result from Problem 4b. Indeed, they say that the Monte Carlo method should only be used when no other method will work—but for high-dimensional functions, this is often the case.