Homework 7
APPM 4600 Numerical Analysis, Fall 2025

Due date: Friday, October 17, before midnight, via Gradescope. **Instructor**: Prof. Becker
Revision date: 10/10/2025

Theme: optimization and least-squares, Gauss-Newton, steepest descent, interpolation, curve-fitting

Instructions Collaboration with your fellow students is OK and in fact recommended, although direct copying is not allowed. The internet is allowed for basic tasks (e.g., looking up definitions on wikipedia) but it is not permissible to search for proofs or to *post* requests for help on forums such as `http://math.stackexchange.com/` or to look at solution manuals. Please write down the names of the students that you worked with. Please also follow our AI policy.

An arbitrary subset of these questions will be graded.

Turn in a PDF (either scanned handwritten work, or typed, or a combination of both) to **Gradescope**, using the link to Gradescope from our Canvas page. Gradescope recommends a few apps for scanning from your phone; see the Gradescope HW submission guide.

We will primarily grade your written work, and computer source code is *not* necessary (and you can use any language you want). You may include it at the end of your homework if you wish (sometimes the graders might look at it, but not always; it will be a bit easier to give partial credit if you include your code). For nicely exporting code to a PDF, see the APPM 4600 HW submission guide FAQ.

Problem 1: Curve-fitting, part 1 We'll consider finding parameters $\boldsymbol{\theta} = [a, b] \in \mathbb{R}^2$ for the curve $\varphi_{\boldsymbol{\theta}}(x) = ae^{bx}$. The data are observations $\{(x_i, y_i)\}_{i=1}^n$ and our goal is to find good values of parameters a and b such that $\varphi_{\boldsymbol{\theta}}(x_i) \approx y_i$ for all $i = 1, 2, \ldots, n$.

a) Derive $\nabla\varphi(x)$ (the gradient of φ with respect to $\boldsymbol{\theta}$, at a given point x)

b) If we have exactly two data points, we can write this as a root-finding problem: find $\boldsymbol{\theta}$ such that

$$\underbrace{\varphi_{\boldsymbol{\theta}}(x_1) - y_1}_{f_1(\boldsymbol{\theta})} = 0, \quad \text{and} \quad \underbrace{\varphi_{\boldsymbol{\theta}}(x_2) - y_2}_{f_2(\boldsymbol{\theta})} = 0. \quad \mathbf{F}(\boldsymbol{\theta}) = \begin{bmatrix} f_1(\boldsymbol{\theta}) \\ f_2(\boldsymbol{\theta}) \end{bmatrix}$$

Derive the Jacobian of $\mathbf{F}$.

c) Take $a = 2$ and $b = 3$ and $\boldsymbol{\theta}^\star = [2, 3]$. Consider exactly two nodes, $x_1 = 0$ and $x_2 = 1$. Generate y_1 and y_2 via $y_i = \varphi_{\boldsymbol{\theta}^\star}(x_i)$. Now pretend we don't know $\boldsymbol{\theta}^\star$ and our aim is to recover it. Do this by solving $\mathbf{F}(\boldsymbol{\theta}) = \mathbf{0}$ via Newton's method, starting at $\boldsymbol{\theta}_0 = [1, 1]$. *Turn in your code, and the output of the iterations of Newton's method — how many iterations does it take?* Note that we are doing this problem to make sure your code (and derivation of the Jacobian) is correct, since you can check your answer. From two data points, you should also be able to derive the parameters a and b via algebraic means.

d) Now let $x_1 = 0, x_2 = \frac{1}{19}, x_3 = \frac{2}{19}, \ldots, x_{20} = \frac{19}{19} = 1$ be 20 equispaced points on $[0, 1]$. Taking $\boldsymbol{\theta}^\star = [2, 3]$ still, this time generate $y_i = \varphi_{\boldsymbol{\theta}^\star}(x_i) + 0.1(-1)^i$. This added perturbation means that we cannot hope to simultaneously solve $f_i(\boldsymbol{\theta}) = 0$ for all $i = 1, 2, \ldots, 20$. So instead, attempt to solve the non-linear least squares problem $\min_{\boldsymbol{\theta}} \frac{1}{2}\sum_{i=1}^{20} f_i(\boldsymbol{\theta})^2$ using the Gauss-Newton method. Do this twice, once starting at $\boldsymbol{\theta}_0 = [1, 1]$ and once starting at $\boldsymbol{\theta}_0 = [1, 2]$. *Turn in your code, and the output of the iterations of the Gauss-Newton method — how many iterations does it take? Does it always converge? Does it come close to recovering the true $\boldsymbol{\theta}^\star$?*

e) Repeat the previous problem, but this time solve the non-linear least squares problem using steepest descent, with stepsize $\eta = 10^{-4}$. *Turn in your code, and the output of the iterations of the steepest descent method — how many iterations does it take? Does it always converge? Does it come close to recovering the true* $\boldsymbol{\theta}^\star$*?*

f) Repeat the previous problem, but this time using a third-party package for curve fitting. In Python, we suggest `scipy.optimize.curve_fit`; you have to give this the form of $\varphi(x, \theta_1, \theta_2)$, and ideally you would also give it the Jacobian — if you don't, then it will approximate it with finite differences (which is acceptable for this problem, since getting the Jacobian to work well requires some programming effort). Most of these methods use Levenberg-Marquardt, which is a robust version of Gauss-Newton. *Compare your results from the third-party package to your own results — are they similar?* [1]

Problem 2: Curve-fitting, part 2 Take the exact same 20 data points $x_1, \ldots, x_{20}$ you generated in problem 1(d), but this time we will interpolate these 20 points with a degree 19 polynomial $p(x)$, so that we have $p(x_i) = y_i$ for $i = 1, 2, \ldots, 20$. For this problem, you can use a third-party package to do the interpolation; in Python, we suggest `scipy.interpolate.BarycentricInterpolator`.

a) For $\boldsymbol{\theta}^\star = [2, 3]$, set $y_i = \varphi_{\boldsymbol{\theta}^\star}(x_i)$ for $i = 1, 2, \ldots, 20$, and interpolate the data. *Plot the data points and your interpolant, with an x-axis between* $[-1, 2]$. *Create two plots, one with a y-axis showing* $[-1000, 1000]$ *and another with a y-axis zoomed in to* $[0, 50]$.

b) Repeat the previous problem, but this time set $y_i = \varphi_{\boldsymbol{\theta}^\star}(x_i) + 0.1(-1)^i$. *Turn in the same two types of plots*

c) Comment on what you've observed.

Problem 3: Is the barycentric formula ill-conditioned?

a) Consider $f(x) = \frac{1}{1-x}$.

i. Find the relative condition number of evaluating f. Is this good or bad when $x \approx 1$?

ii. Actually evaluate f for $x = 1 - 10^{-13}$ on the computer, using any software/language that uses floating point numbers[2], and report your answer and the error.

iii. About how many correct digits do you have? Is this expected?

iv. If you evaluate f for $x = 1 - 2^{-43} \approx 1 - 1.13 \cdot 10^{-13}$, how many correct digits do you have now? (Don't forget your true answer has changed)

b) The barycentric formula for Lagrange interpolation looks like

$$p(x) = \sum_{i=0}^{n} \frac{\frac{w_i}{x - x_i} y_i}{\frac{w_i}{x - x_i}}$$

which might make you nervous when $x \approx x_i$ for some i.[3]

Let's analyze a simplified version of this formula that still retains all the interesting (and worrisome?) behavior:

$$f(x) = \frac{\frac{c}{1-x} + d}{\frac{1}{1-x} + a}$$

for some constants a, d and some $c \neq 0$.

[1] Comment: for this particular curve fitting problem we've been examining in Problem 1, one could do a log transform of the data y and turn this into a *linear* least squares problem, which can be solved via linear algebra and there's no issues with convergence or stepsizes. This does slightly change the problem, since the squared residuals are now in the log transformed space.

[2] Do *do not* use a symbolic computation framework like Mathematica unless you explicitly convert to floating point numbers by, e.g., writing $x = 1. - 10.^{-13.}$ where the decimal dots tell it to use floating point. Tools like graphing calculators and Wolfram Alpha are also probably to be avoided since they try to guess what you want, and could be doing it in either exact arithmetic or floating point.

[3] A full analysis of the stability of the barycentric formula is not elementary, and it does depend on the set of nodes $\{x_i\}$. If you're interested, see *The numerical stability of barycentric Lagrange interpolation*, by Nicholas J. Higham, in the IMA Journal of Numerical Analysis,24(4), 2004

i. Find $\lim_{x \to 1} f(x)$.

ii. Find the relative condition number κ_f for $x = 1$. *Hint: when evaluating this, plug in* $x = 1$ *as soon as possible to simplify the algebra.*