MTH 373/573:
Scientific Computing

Homework 3

IIIT-Delhi, Monsoon 2021
Kaushik Kalyanaraman

Due by: December 3, 2021 (Friday) by 23:59 IST

Total: 150 points

Please note that no late submission is allowed; turn in by above submission date and time.

Guidelines

- Please consult with and carefully read **IIIT-Delhi's Academic Dishonesty Policy** at this link.
- *This assignment should be answered individually.* You are not allowed to discuss specifics of your solutions, solution strategies, or coding strategies with any other student in this class. There are only two exceptions:
  1. You may post queries relating to any HW question to `#homeworks` channel of the class's Discord server.
  2. You may reach out to me or one of the Ph.D. TAs – Archana (archanaa@iiitd.ac.in) or Gaurav (gauravr@iiitd.ac.in) with your clearly phrased questions about any HW problem.
- Start working on your solutions early and don't wait until close to due date.
- Each problem should have a clear and concise write-up.
- Please clearly show all steps involved in your solution.
- A late submission can turned within an additional 2 days with grading rescaled to 0.6 of total points.
- You will need to write a separate code for each computational problem and sometimes for some subproblems too. You should name each such file as `problem_`*n*`.py` where *n* is the problem number. For example, your file names could be `problem_5.py` `problem_6a.py`, `problem_6b.py`, and so on.
- *Python tip:* You can import Python modules as follows:

```python
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import numpy.linalg as npla
import scipy.linalg as spla
```

- You can submit IPython Notebook files `*.ipynb` as well. Please use the same naming convention as above. If you choose to do this, please clean your output and also provide an exported HTML file. You can go to a Notebook's File menu options and choose `Kernel -> Restart & Run All` for a clean output. For exporting to HTML, you carry out `File -> Download as -> HTML (.html)`.

Problem 1: Newton's method in 1 dimension — 30 points

(a) Newton's method for solving a scalar nonlinear equation $f(x) = 0$ requires computation of the derivative of f at each iteration. Suppose that we instead replace the true derivative with a constant value d, that is, we use the iteration scheme:

$$x_{k+1} = x_k - f(x_k)/d.$$

i) Under what conditions on the value of d will this scheme be locally convergent?

ii) What will be the convergence rate in general?

iii) Is there any value of d that would still yield quadratic convergence?

(b) Write a Python function to implement Newton's method in 1d. Your function should take as inputs the 1d function, its derivative, an initial guess and a tolerance for checking convergence. Use this function to find the roots of the following functions using Newton's method with the provided initial guess.

i) $x^2 - 1 = 0$, $x_0 = 10^6$.

ii) $(x-1)^4 = 0$, $x_0 = 10$.

iii) $x - \cos x = 0$, $x_0 = 1$.

What is the observed convergence rate of Newton's method for each problem? Why does each problem exhibit this convergence rate?

You can compute the rate by using the fact that the error at each iteration $e_k = x^* - x_k$ satisfies $|e_{k+1}| / |e_k|^r = C$. Thus, $|e_{k+1}| / |e_k|^r = |e_k| / |e_{k-1}|^r$ from which you should derive an equation for obtaining r for each iteration. You can use the computed solution in the last iteration as the true solution x^* and the last of these computed rates to be the rate of convergence.

Problem 2: Newton's method for a system — 30 points

Consider the conversion formula from spherical to Cartesian coordinates:

$$x = r \sin\theta \cos\varphi$$
$$y = r \sin\theta \sin\varphi$$
$$z = r \cos\theta$$

(a) Write a function `newton(f, J, x0, tol=1e-12, maxit=500)` that implements Newton's method generically, with a starting guess of `x0`.

`f` is a function that accepts a `numpy` array x of the current state and returns the function value $f(x)$ as another `numpy` array. Write `f` so that $f(x) = 0$ at the sought solution.

`J` is the Jacobian of `f`, so given a `numpy` array `x`, it returns a Jacobian matrix of the appropriate dimensions.

Terminate the iteration once either the residual is smaller than `tol` or `maxit` iterations have been performed.

(b) Write a function that, given x, y, and z, finds r, θ and φ, using the Newton implementation from part (a). Do not reimplement Newton's method, simply pass in appropriate functions `f` and `J` to `newton` from part (a).

Find a starting guess that leads to convergence in your experiments.

The routine should accept x, y, z in *one* input vector and return r, θ and ϕ in *one* output vector.

(c) Test your work from (a) and (b) by drawing 10 random vectors from $x^* \in \mathbb{R}^3$ using `np.random.randn` and finding their spherical coordinates.

For each case, print the final relative residual $\|x - x^*\|_2 \,/\, \|x^*\|_2$, where x are the Cartesian coordinates corresponding to the spherical coordinates returned by your routine.

Also compare the polar coordinates $\widetilde{w} = \begin{bmatrix} \tilde{r} & \tilde{\theta} & \tilde{\varphi} \end{bmatrix}^T$ output by your routine with the true values computed using the formulas:

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arccos\left(\frac{z}{r}\right)$$
$$\varphi = \arctan\left(\frac{y}{x}\right)$$

Let $w = \begin{bmatrix} r & \theta & \varphi \end{bmatrix}^T$. For each example, print the relative error $\|w - \widetilde{w}\|_2 \,/\, \|w\|_2$ in your computed values. Is this small whenever the residual is small? Why/why not?

Hints:

- Read the documentation for and use `np.arctan2` in implementing the formulas to find r, θ, and ϕ.
- It is instructive to watch the conditioning of the Jacobian. A singular Jacobian will cause the method to break down or compute inaccurate results. Some starting guesses are more prone to this than others.

Problem 3: Chebyshev polynomials, Vandermonde matrices 50 points

(a) A function $F_n(t)$ is known to satisfy the *Chebyshev three-term recurrence* if the following properties hold:

$$\begin{aligned} F_0(t) &= 1 \\ F_1(t) &= t \\ F_{n+1}(t) &= 2tF_n(t) - F_{n-1}(t) \end{aligned}$$

Show that the function

$$F_n(t) = \cos(n \arccos(t))$$

satisfies the Chebyshev three-term recurrence.

Hint: Remember the addition formula for cosines:

$$\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta).$$

How would this apply to $\cos((n+1)\arccos(t))$?

(b) Deduce that $F_n(t)$ is a polynomial. This is called the *Chebyshev polynomial*.

(c) The *generalized Vandermonde matrix* V for a set of points $x_1, \dots, x_n$ with a set of functions $\phi_1, \dots, \phi_n$ is given by

$$V_{ij} = \phi_j(x_i).$$

Notice that this matrix captures an essential bit of interpolation: it maps coefficients with respect to the functions (ϕ_j) to point values at the points (x_i):

$$\sum_{j=1} \alpha_j \phi_j(x_i) = \sum_{j=1} V_{ij}\alpha_j = (V\alpha)_i.$$

Clearly, V^{-1} describes the reverse process, and the conditioning of V can tell us quite a bit about how well-behaved of an operation interpolation is with respect to the given sets of functions and nodes.

Perform the following steps:

i) Construct $n \times n$ generalized Vandermonde matrices for $n = 5, 10, 15, \dots, 100$.

ii) Plot the condition number of the matrix with respect to n. for each of the cases below:

  i. Equispaced nodes x_i in the interval $[-1, 1]$ (endpoints included), with the monomials $\phi_j(x) = x^j$.

  ii. Chebyshev nodes in the range $[-1, 1]$ are given by

$$x_i = \cos\left(\frac{2i+1}{2n}\pi\right),$$

  with the monomials.

  iii. Equispaced nodes x_i in the interval $[-1, 1]$ (again, endpoint included), with the Chebyshev polynomials F_j.

  iv. Chebyshev nodes in the interval $[-1, 1]$ (see equation above) with the Chebyshev polynomials F_j.

Your output should include one (clearly labeled) plot as described above showing (and comparing) the behavior of the condition number for all four cases.

Hint: Use `matplotlib.pyplot.semilogy()` to plot the values with a linear scale in n on the x axis and a logarithmic scale for the condition number on the y axis.

(d) Which of the combinations performs best?

Problem 4: Interpolation, Newton and Cubic Spline 40 points

(a) The following formula is called the *divided differences* approach to computing the coefficients of an interpolating polynomial using Newton's polynomials as a basis:

$$f[t_1, t_2, \dots, t_k] := \frac{f[t_2, t_3, \dots, t_k] - f[t_1, t_2, \dots, t_{k-1}]}{t_k - t_1}$$
$$f[t_j] := f(t_j)$$

Prove that, using mathematical induction, that indeed this approach gives the coefficient of the j^{th} basis function using the Newton interpolation polynomial.

(b) Given the three data points $(-1, 1)$, $(0, 0)$, $(1, 1)$, determine the interpolating polynomial of degree 2 (using hand calculations alone and showing all necessary steps) using:

i) Using the monomial basis.

ii) Using the Lagrange basis.

iii) Using the Newton basis.

iv) Show that all three representations give the same polynomial.

(c) Consider interpolating a given set of data points (x_i, y_i), $i = 1, \dots, n$ using natural cubic splines. Write a code to set up and solve the linear system that performs this interpolation. Plot the resulting cubic spline along with the data. For the data, pick $n = 6$ random points $(x_i)_{i=1}^n$ on $[0, 1)$ with values $(y_i)_{i=1}^n$ in $[0, 1)$.

Hint: Make sure you sort the x_i's after you draw the random numbers and before you start constructing the spline, to avoid confusing your spline construction code.

Submission Notes

1. The answer to all theoretical problems, output of Python code for computational problems including figures, tables and accompanying analyses should be provided in a single PDF file along with all your code files.
2. You can typeset (please consider using LaTeX if you choose to do so), or write by hand and scan/take photographs of solutions for theoretical questions. For scanning or photography, please put in the extra effort and provide a single PDF file of your submission.
3. For Problem 2, submit your code in one file: `problem_2.py`. You can use the Python file provided as a boilerplate.
4. For Problem 3, submit your code in one file again: `problem_3.py`. You will of course, copy-paste the functions written in `problem_2.py` for Gaussian elimination without and with partial pivoting into this file.