Project 2: GPS, Solving Systems of Equations, Conditioning, and Nonlinear Least Squares.
Due date: 11/23/2022

Statement of the Project

Solve the "Reality Check 4" project that is presented in the textbook *Numerical Analysis* by Timothy Sauer (3rd edition, pages 248 – 251). More specifically, **solve activities 1,2,4,5,6** (activity 3 is not necessary), with the following modifications:

- You don't need to use MATLAB. You can use jupyter/python instead. In this case, you may disregard all MATLAB hints in the text.
- In activity 1, make sure that your implementation of Multivariate Newton's Method does **not** compute the inverse of the Jacobian matrix.
- In activity 1, add the following questions:
  - What stopping condition did you use in your solver? How accurate is the obtained solution (root)?
  - It can be shown that if at a neighborhood of an isolated root x^* the Jacobian matrix $DF(x)$ has a bounded inverse and continuous derivatives, then Newton's method converges locally **quadratically**, i.e., there is a constant M such that
    $$\|x^* - x_{k+1}\| \leq M\|x^* - x_k\|^2,$$
    provided $\|x^* - x_k\|$ is already small enough.
    Demonstrate the quadratic converge of Multivariate Newton's Method by tabulating $\|F(x_k)\|_\infty$ for $k = 0, 1, 2, \ldots$.
- In activity 2, add the following questions:
  - How many solutions are there to system (4.37)? How did you choose the physically correct one among them?
  - Why is it better to solve system (4.37) using the quadratic formula instead of the Multivariate Newton Method? For activities 4 and 5 below, use the quadratic formula instead of the Multivariate Newton Method.
- In activity 4, use the following equispaced values for the spherical coordinates:
  $$(\phi_i, \theta_i) = \left(i\frac{\pi}{8}, \quad (i-1)\frac{\pi}{2}\right), \qquad \text{for } i = 1, 2, 3, 4.$$
  For illustration, you can plot the position of the 4 satellites.
  Use the following variations of Δt_i's:
  $$(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4) = (+\epsilon, +\epsilon, +\epsilon, -\epsilon)$$
  $$(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4) = (+\epsilon, +\epsilon, -\epsilon, -\epsilon)$$
  $$(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4) = (+\epsilon, -\epsilon, +\epsilon, -\epsilon)$$
  $$(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4) = (+\epsilon, -\epsilon, -\epsilon, -\epsilon)$$
  with $\epsilon = 10^{-8}$.

- In activity 5, use the following tightly grouped values for the spherical coordinates:

$$(\phi_i, \theta_i) = \left(\frac{\pi}{2} + (i-1)\frac{5}{100}\frac{\pi}{2}, \quad (i-1)\frac{5}{100}2\pi \right), \qquad \text{for } i = 1, 2, 3, 4.$$

  For illustration, you can plot the position of the 4 satellites.

- Activity 6 is **the most important one**, since it's the only activity involving least squares.

- In activity 6, add the following questions:

  - In general, Gauss-Newton is only locally convergent (starting from an initial condition close enough to the minimum). What is a good initial vector in this particular problem? If you use a good initial vector, Gauss-Newton should converge (because it is *locally* convergent).
  - Even when it converges, it does not necessarily converge to a minimum: it could converge to a maximum, or to a saddle point. Briefly explain how one could check that it actually converged to a minimum. (You don't need to implement this).

- In activity 6, use the following equispaced values for the spherical coordinates, corresponding to eight unbunched satellites:

$$(\phi_i, \theta_i) = \left(i\frac{\pi}{16}, \quad (i-1)\frac{\pi}{4} \right), \qquad \text{for } i = 1, 2, \ldots, 8.$$

  Use the following variations of Δt_i's:

$$\begin{aligned}
(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4, \Delta t_5, \Delta t_6, \Delta t_7, \Delta t_8) &= (+\epsilon, +\epsilon, +\epsilon, -\epsilon, +\epsilon, +\epsilon, +\epsilon, -\epsilon) \\
(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4, \Delta t_5, \Delta t_6, \Delta t_7, \Delta t_8) &= (+\epsilon, +\epsilon, -\epsilon, -\epsilon, +\epsilon, +\epsilon, -\epsilon, -\epsilon) \\
(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4, \Delta t_5, \Delta t_6, \Delta t_7, \Delta t_8) &= (+\epsilon, -\epsilon, +\epsilon, -\epsilon, +\epsilon, -\epsilon, +\epsilon, -\epsilon) \\
(\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4, \Delta t_5, \Delta t_6, \Delta t_7, \Delta t_8) &= (+\epsilon, -\epsilon, -\epsilon, -\epsilon, +\epsilon, -\epsilon, -\epsilon, -\epsilon)
\end{aligned}$$

  with $\epsilon = 10^{-8}$.

  Do results change much if you try other combinations of signs for the Δt's?

The deliverables consist of a Project Report and a Code Listing.

Project Report

This is a typed-up report (does not need to be long, 5 pages is okay) that carefully discusses and presents your **solutions to the project questions**. **A solution with no explanation may not receive full credit**.

If the activity asks for a numerical solution, always include the numerical solution in the report. (The numerical solution may be a number, a vector, a list of numbers, a table, ...). Be sure to use enough significant digits!

If the activity asks for a plot, always include the plot in the document.

For instance:

Please explain any other important methodological decision that you made. For instance (these are just hypothetical examples), if you

- decided to try several methods and compare their performance;
- combined several methods into one super-method;
- tried several stopping conditions, and decided to use this one or another, because ...;
- timed the running time of your code and report it in a table;
- performed a fantabulous check to validate your results are correct;
- greatly optimized your code using such and such trick;
- etc, etc, etc

The report should be submitted to the instructor **via Canvas as a PDF** file by the due date.

Code Listing

Your code should be clearly written, consistent with best practices, and **reproducible**. In particular:

- **Write your own numerical code**. The goal of this course is to learn numerical methods. You must program the **numerical methods explained in class** yourselves. Don't use readily-available numeric libraries, such as SciPy's `fsolve` or MATLAB's `vpasolve` to find roots of an equation. Don't use python's `solve_ivp` to solve an ODE. And so on.

  Of course, you may use libraries for all other things (e.g. for data manipulation, plotting, etc.). You may also use libraries for standard numerical computations (e.g. standard functions like trigonometric functions, matrix algebra, ...)

- **Reproducibility**. Your code must be **ready to run** and **produce the same results** documented in the project report. Points will be deduced otherwise (e.g. I press "Run" and it hangs...).

  I recommend to add a brief README.txt file explaining how to run your code (e.g. "Run the jupyter notebooks activity1.ipynb, activity2.ipynb", ... Or: "First check that that package TDA.R is installed in the system. Then execute the command `Rcmd activity1.r` from the command line").

- **Structure your code**. Divide your code into files with meaningful names (e.g. activity1.m, activity2.m, ...). Use **functions** to encapsulate relevant pieces of code. Use meaningful names for constants/variables (*don't* use "Dracula", "foo", "mystring", ...). If you like, use classes / Object Oriented Programming (but it's not necessary for this course).

- **Comment your code**. Be as verbose as necessary, adding comments profusely whenever code is complex.

- **Test key functions**. If a function performs some complex task and is very important for your project, test it on a few known examples. The more complex the function, the more exhaustive the testing.

- **Reference your sources**. Do not "borrow" code from other people (websites, textbooks, research papers ...) unless you cite them. No exceptions!

The code should be submitted to the instructor **via Canvas as a Zip (compressed)** file by the due date.