Numerical Methods in Physics and Astrophysics (2022-23)

Kostas Kokkotas

October 27, 2022

Contents

1 Non-Linear Equations & Systems **2**
1.1 Root Finding: Bisection, Linear Interpolation, $x = g(x)$ and Newton 3
1.2 Nonlinear Systems of Equations 4
1.3 Fractals through Newton-Raphson 4

1 | Non-Linear Equations & Systems

1

[1] This set of problems for the course developed over the last 12 years with the help of my collaborators:

- Dr. Wolfgang Kastaun (2007-9)
- Dr. Buchart Zink (2010-13)
- Dr. Tanja Bode (2013-15)
- Dr. Daniela Doneva (2015-17)
- Dr. Praveen Manoharan (2021-22)

together with many valuable comments and suggestions from my ex-students.

1.1 | Root Finding: Bisection, Linear Interpolation, $x = g(x)$ & Newton

PROBLEM I

We will consider specifically the following function:

$$f(x) = e^{\sqrt{5}x} - 13.5\cos(0.1x) + 25x^4 \tag{1.1}$$

1. Write a program which uses the **bisection method** to find the positive root of the function above with an accuracy of 10^{-14}.
2. Find this root again, using **linear interpolation**.
3. Implement **Newton's method**.
4. Examine the convergence of the three methods.
5. Explore the robustness of the methods with varying initial guesses and tolerances. What happens when you specify a tolerance below machine precision?

PROBLEM II

Find the eigenvalues λ of the differential equation $y'' + \lambda^2 y = 0$ with the following boundary conditions $y(0) = 0$ and $y(1) = y'(1)$.

Tips, Hints, and Suggestions:

As first step you should try to solve the problem analyticaly by considering the general solution of this wave equation.
Then use the boundary conditions to simplify the solution and the problem will be reduced in finding the roots of a non-linear equation.

1.2 | Nonlinear Systems of Equations

PROBLEM III

We first consider the following nonlinear system of equations

$$f_1(x, y) = xy - 0.1 \tag{1.2}$$
$$f_2(x, y) = x^2 + 3y^2 - 2 \tag{1.3}$$

A1. Solve the nonlinear system of equations using the generalized Newton method. Make sure you find all solutions (there are 4!). Under what conditions would the solver create problems? What checks can you put on/in your solver to avoid diverging from the solution or continuing through with NaNs?

A2. Reformulate the equations and add a function for the "improved" $x = g(x)$ method to the above program. What are the requirements for convergence? How does this affect solving the above system of equations? How is the relative performance for the various roots? How does the program behave if you guess $(x, y) = (2, 1)$?

1.3 | Fractals through Newton - Raphson

PROBLEM IV

Fractals and chaos theory are intricately entwined. In a nutshell, chaos theory amounts to the sensitive, diverging nature of a solution with small changes in the initial conditions. We will go through another chaos problem later in these exercises, but here we delve deeper into the fractal nature of root-finding via the Newton-Raphson method. For these solutions, as 1-dimensional fractals are not as pretty, we will look at solutions to simple functions in the complex plane. The iteration scheme in Newton-Raphson here is the same as for functions in real space:

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}$$

For each function $f(z)$ we consider in this lab, we create a map of convergence in the (x_0, y_0)-plane where $z_0 = x_0 + iy_0$ is the initial guess to our solver. For which z_0 does the method converge? How fast does it converge? To which root does it converge?

1. Consider first the solution of the function

$$f(z) = z^3 - 1 = 0 \tag{1.4}$$

   - ► Learn how to handle complex numbers in your language of choice.
   - ► Write a function `solve_cnewton` which takes an initial guess and returns the root (if any), the number of iterations need to converge, and the quality of the solution $f(z)$. This should be as simple as grabbing your old newton solver from Lab 1 and upgrading to use complex numbers and interface with the function `get_fdf`.
   - ► Write a program that takes a square $N \times N$ grid in the complex plane and, for each point, uses the above functions to find a root to which the method converges. For a convergence criteria, look for when the change in z is less than ϵ with a certain number of maximum steps M. That is when within M steps $|z_{n+1} - z_n| < \epsilon$. Start with $M = 200$, $\epsilon = 10^{-9}$, and $N = 400$. Define the range of the grid by two gridpoint coordinates (e.g. upper left and lower right). If you know how, take these points as arguments to the main function to ease exploring a given function; otherwise hardcode them initially at $|x_0| < 2$ and $|y_0| < 2$.
   - ► Have the main program print out or save, for every point z_0 of your $N \times N$ grid, a line of format

$$x_0 \quad y_0 \quad k(z) \quad f(z) \quad \log_{10}(n_{\texttt{itns}})$$

     with one blank line after each row of the matrix (if you are working in C in linux). Here $k(z)$ is either the complex root to which the method converges, or zero. n_{itns} is the number of iterations taken to converge, if it does.
   - ▼ Plot $\Im(k(z))$ on the (x, y) plane with a colorscaling in map mode (see hint). Find an interesting region and look at it closer. Repeat. Choose an interesting region and save the plot (with full labels of axes of course!).
   - ▼ Plot $\log(n_{\text{itns}})$ in the (x, y) plane, focusing around a region where the solver does not converge. Comment on what you see.

2. Look at the website http://www.chiark.greenend.org.uk/~sgtatham/newton/ and use your basic code with different functions to explore and understand the convergence and sensitivities of Newton-Raphson. Consider particularly solutions to the function

$$f(z) = 35z^9 - 180z^7 + 378z^5 - 420z^3 + 315z \tag{1.5}$$

   Extra: Choose another function of your choice to present to and discuss with the class.

Tips, Hints, and Suggestions:

- *If you are on a linux machine and you choose to print out the results to stdout, the program* `gnuplot` *can be used to study the results as follows:*

  > gnuplot
  gnuplot> set pm3d map
  gnuplot> splot "<./ComplexRoots" u 1:2:3

  where `ComplexRoots` *is the program name. (If your on another architecture ... perhaps write to a file and read into your favorite visualizer?).*

- *There was no standard for complex numbers in C until the C99 standard. The compiler* `gcc` *(and others) know a data type* `double complex`*, which requires including the extra header "complex.h". A complex number* $z = 3 + 5i$ *is then set by*
  `double complex z = 3.0 + 5*I`.
  The absolute value, real part, imaginary part, and complex phase of a complex number can be calculated by the similarly provided functions `cabs(z)`, `creal(z)`, `cimag(z)`, *and* `carg(z)` *respectively. C++ has <complex> containers, akin to <vector>. Matlab assumes any number could be complex.*

- *The roots of* $f(z) = z^3 - 1$ *are* $\{1, e^{\pm\frac{2}{3}\pi i}\}$*. Or* 1 *and* $(-0.5, \pm 0.8660254040\, i)$.