Phys 416/517 – Chapter 4

[Due by 1PM, Thursday March 12 at 1PM, 2020]

Linear equations

13. Consider the mass-spring system in the figure below (a simple model of a short polymer molecule - Figure 4.4 from the text). The blocks are free to move but only along the x-axis. The springs connecting adjacent blocks have spring constant k_1, while the two outer springs have stiffness k_2. All the springs have a rest length of one.

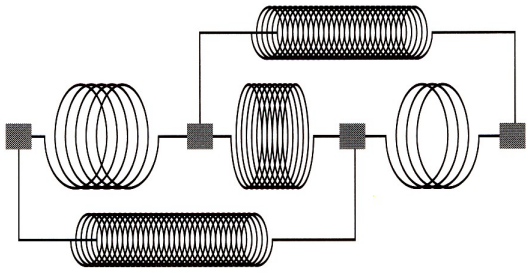


Figure 4.4 from Garcia

(a) Write out the matrix equation for the equilibrium positions of the blocks. [Pencil].

(b) Write a program that plots the total length of the system as a function of the ratio $\frac{k_1}{k_2}$ (for a range between 10^{-3} to 10^3 on a $\log_{10}$ scale.)

[*Computer, turn in the program and results, including plots.]*

Application of Newton's method

19. For blackbody radiation, the radiant energy per unit volume in the wavelength range λ to $\lambda+d\lambda$ is

$$u(\lambda)d\lambda = \frac{8\pi}{\lambda^5}\frac{hc}{\exp[hc/(\lambda kT)]-1}d\lambda$$

where T is the temperature of the body, c is the speed of light, h is Planck's constant, and k is Boltzmann's constant. Show that the wavelength in which $u(\lambda)$ is maximum may be written as

$$\lambda_{max} = \alpha hc/kT$$

where α is a constant. Determine the value of α numerically from the resulting transcendental equation. [Computer]

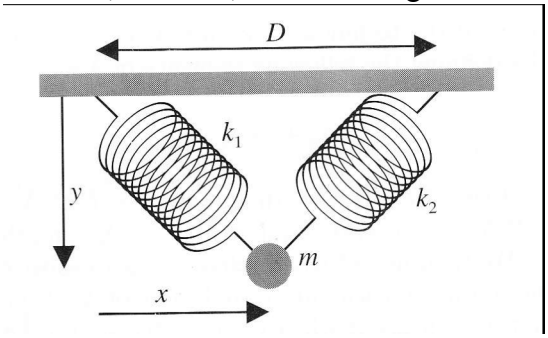
A. Multivariable Newton Problem.

24. For the simple, two-dimensional spring-mass system shown in the figure below, the potential energy is given by

$$V(x,y) = \frac{1}{2}k_1(\sqrt{x^2+y^2}-L_1)^2 + \frac{1}{2}k_2(\sqrt{(x-D)^2+y^2}-L_2)^2 - mgy$$

where k_1 and k_2 are the spring constants, L_1 and L_2 are the rest lengths of the springs, m is the mass of the object, and g=9.81 m/s^2. At a static equilibrium, $\vec{F} = -\nabla V = 0$. Write a program to

solve for the positions x and y at static equilibrium. What is the equilibrium position for k_1 =10N/m, k_2=20 N/m, $L_1 = L_2$ =0.1m, D=0.1m, and m=0.1 kg.



Note that for multidimensional problems, Newton's method takes the form:

$$\vec{f}(\vec{x}^*) = \vec{f}(\vec{x} + \delta\vec{x}) = \vec{f}(\vec{x}) + \delta\vec{x}\mathbf{D}(\vec{x}) + O(\delta\vec{x}^2) = 0$$

Where $\mathbf{D}(\vec{x})$ is the Jacobian matrix defined as $D_{ij}(\vec{x}) = \dfrac{\partial f_j(\vec{x})}{\partial x_i}$ (the index i corresponds the row and j the column). Solving for $\delta\vec{x}$ we get

$$\delta\vec{x} \approx -f(\vec{x})\mathbf{D}^{-1}(\vec{x})$$

To iterate and get a new solution we use the following

$$\vec{x}^* \approx \vec{x} + \delta\vec{x} = \vec{x} - f(\vec{x})\mathbf{D}^{-1}(\vec{x})$$

(In the convention used in the book, $\vec{x}$ is a row matrix, so we left multiply the matrix).

Hints:

1. Your program will have to iterate until it converges. You will need to set a criteria for convergence.
2. You may find that successful convergence will depend on your initial guess, sometimes a poor guess will give a non-physical result.
3. One sign of a problem is if your matrix D is singular. This would suggest your programs or equations have errors.
4. Another check of your program is to set the spring constants and the lengths to be equal. That should yield a symmetric solution.
5. The matrix **D** should be symmetric.

[*Computer: turn in the program and a graph of the equilibrium position of the mass.]*

B. Euler's Collinear Solution to the 3 body problem

Euler and Lagrange found exact solutions to the three body equations. Referring to the document "nbody choreography.pdf", Lagrange's solution puts the masses at the edge of an equilateral triangle, while Euler's solution sets mass 2 to lie in between masses 1 and 3. We can satisfy equation (4) from *nbody* by assuming that

$$\vec{s}_1 = \lambda\vec{s}_3 \qquad\qquad \vec{s}_2 = -(1+\lambda)\vec{s}_3 \qquad (1)$$

where λ is a positive scalar to be determined. We can eliminate $\vec{G}$ from equations (6) in the *nbody* program to get

$$\ddot{\vec{s}}_1 + GMm\frac{\vec{s}_1}{|\vec{s}_1|^3} = \frac{m_1}{m_3}\left(\ddot{\vec{s}}_3 + GMm\frac{\vec{s}_3}{|\vec{s}_3|^3}\right)$$
$$\ddot{\vec{s}}_2 + GMm\frac{\vec{s}_2}{|\vec{s}_2|^3} = \frac{m_2}{m_3}\left(\ddot{\vec{s}}_3 + GMm\frac{\vec{s}_3}{|\vec{s}_3|^3}\right) \qquad (2)$$

Inserting (1) in (2) to eliminate the vectors $\vec{s}_1$ and $\vec{s}_2$ we get

$$(m_2 + m_3(1+\lambda)\ddot{\vec{s}}_3 = -(m_2 + m_3(1+\lambda)^{-2})\frac{GMm\vec{s}_3}{|\vec{s}_3|^3}$$
$$(m_1 - m_3\lambda)\ddot{\vec{s}}_3 = -(m_1 - m_3\lambda^{-2})\frac{GMm\vec{s}_3}{|\vec{s}_3|^3}$$

Which means then

$$\frac{m_2 + m_3(1+\lambda)}{m_1 - m_3\lambda} = \frac{m_2 + m_3(1+\lambda)^{-2}}{m_1 - m_3\lambda^{-2}}$$

Rearranging we get

$$(m_1+m_2)\lambda^5 + (3m_1+2m_2)\lambda^4 + (3m_1+m_2)\lambda^3 -$$
$$(m_2+3m_3)\lambda^2 - (2m_2+3m_3)\lambda - (m_2+m_3) = 0 \qquad (3)$$

Equation (3) is a 5th order polynomial that can be solved for λ given the masses.

1. Make another option for your `nbody` program that includes this solution. You are free to use the supplied version of `nbody`, or the program you used for chapter 3.

The basic procedure is as follows:

2. Write a Newton solver to solve for λ in equation (3) for a given set of masses.
3. Use this value to set the initial position of the planets using equations (5) from `nbody`.
4. Set the initial speed so that the motion is circular (You will have to work this out).

When this program is working properly, your masses should line up in a straight line.

C. Basin of Attraction.

Write a function that solves for the complex roots of the function $z^n - 1 = 0$. The function should return, for a given value of initial starting point, n, tolerance, and maximum number of iterations, the solution it found and the number of iterations it took to find the solution. If the solution fails to converge after a fixed number of iterations, it should return zero. The criteria for convergence is to satisfy both $|z_{n+1} - z_n| < \varepsilon$ and $|z^n{}_{n+1} - 1| < \varepsilon$ where z_n is the n-th iteration and ε is the desired error tolerance. Use a value of ε of 10^{-5} and a maximum number of iterations of 50.

Use this function as part of a program that generates 2 images:

1. The first will show the number of iterations it took to find the solution as a function of a starting point.
2. The second plot should plot the solution it found. Generating the solution figure may be a little tricky. For a given value of n, there should be *n+1* **distinct** solutions (including the zero found when it fails to find a solution). However due to numerical differences, the solutions will not be exactly the same and you will have to figure out how to classify the solution the program found as one of the n+1 solutions. The plot should show n+1 colors for each of the solutions it found.

The starting point should on be a mesh of points as defined by -1 ≤ Real(z) ≤ 1 and -1 ≤ Imaginary(z) ≤ 1. Use a mesh of size 1001 × 1001 points inside a square (but use a grid of size 11 × 11 for testing as the program will take a while to run). Both figures can made using the surf

function in *MATLAB* and viewing the figures from the top. (contour works fine for python, although there may be better alternatives.) Try running your program for n=3 and n=4.

Optional Extra Exercise – 1 point Extra Credit

D. Write your own Matrix Inverse Routine

Newton's method can be used to compute the inverse of a nonsingular matrix **A**. If we define the function F as

$$\mathbf{F}(\mathbf{X}) = \mathbf{I} - \mathbf{AX} \tag{D1}$$

where **X** is an n x n matrix, **I** is the identity matrix, then **F**(**X**) =0 when **X** = $\mathbf{A}^{-1}$.
Because **F**'(**X**) = -A, Newton's method for solving this equation has the form

$$\begin{aligned} \mathbf{X}_{k+1} &= \mathbf{X}_k - \left[\mathbf{F}'(\mathbf{X}_k)\right]^{-1}\mathbf{F}(\mathbf{X}_k) \\ &= \mathbf{X}_k + \mathbf{A}^{-1}(\mathbf{I} - \mathbf{AX}_k) \end{aligned} \tag{D2}$$

But $\mathbf{A}^{-1}$ is what we are trying to get, so instead we use the current guess for $\mathbf{A}^{-1}$ which is $\mathbf{X}_k$. Thus the scheme becomes

$$\boxed{\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{X}_k(\mathbf{I} - A\mathbf{X}_k) = 2\mathbf{X}_k - \mathbf{X}_k A\mathbf{X}_k} \tag{D3}$$

(a) If we define the residual matrix

$$\mathbf{R}_k = \mathbf{I} - \mathbf{AX}_k$$

and the error matrix

$$\mathbf{E}_k = \mathbf{A}^{-1} - \mathbf{X}_k$$

Show that

$$\mathbf{R}_{k+1} = \mathbf{R}_k^2 \text{ and } \mathbf{E}_{k+1} = \mathbf{E}_k A\mathbf{E}_k$$

from which we can conclude that the convergence is quadratic, despite only an approximate derivative.
(b) Write a function to compute the inverse of a given a random input matrix A using this iteration scheme (D3). A reasonable starting guess is

$$\mathbf{X}_0 = \frac{\mathbf{A}^T}{\|\mathbf{A}\|_1 \cdot \|\mathbf{A}\|_\infty} \tag{D4}$$

Test your program on some random matrix and compare its accuracy to the MATLAB inv routine.
Notes and Hints:

1. Write a function that takes a random matrix and returns the inverse.
2. In the function, you will need to check to see if the matrix has an inverse, if it does not the program needs to return an error.
3. Since the scheme is an iterative scheme, you will need to define a stopping criteria for convergence. If the method fails to converge, then the function should exit with an error.
4. For the starting point (B4), MATLAB has a norm function and a transpose ('). A similar function exists in python.