Ph 20.5 – Introduction to Symbolic Algebra with *Mathematica*

Due: Week 8 -v20170314-

This Assignment

In the next two assignments we return to scientific computing, but this time using *Mathematica* instead of Python. As you become familiar with them you will learn that the two programs have different strengths and weaknesses; if you continue in scientific computing you will almost certainly use both regularly for different tasks. Compared to Python *Mathematica* really shines when it comes to doing symbolic or algebraic work, much of which is simply impossible using Python. On the other hand Python/`numpy`/`pyplot` are typically more convenient for making good-looking plots of numerical data and for doing anything where writing a loop is necessary. When it comes to numerical analysis such as numerical analysis of differential equations you will likely find that Mathematica is superior when you can use a built-in function to do the job but much harder to work with when you need to write your own code from scratch.

This assignment shows off some of the areas that *Mathematica* is best at. If you have already worked with *Mathematica* before much of it will be very straightforward; if you finish the assignment quickly you may want to get a head start on next week's assignment. Before moving on you should make sure you are comfortable with the use of replacement rules (`/.` notation), which will be used repeatedly in 20.6.

One further advantage of *Mathematica* is that a notebook can combine code, text, and comments into one document. You are welcome to submit the notebook itself rather than a PDF for this assignment and the next provided you include commentary. If you are interested in doing something similar in Python look up "ipython notebooks."

Introduction

The techniques of computational science have been developing rapidly, driven by advances in computer hardware and software. In particular, the ready availability of increasingly higher performance of laptops and workstations has allowed sophisticated numerical and graphical techniques to be used routinely for computational analysis in both theory and experimentation. In this assignment you are introduced to *Mathematica*, a *computing environment* developed to do pretty much any kind of mathematics on a computer. Among other things, *Mathematica* allows you to manipulate symbols, numbers, data, and graphics. Thus, it is a very general program; however, speed and ease of use (and yes, elegance) sometimes leave something to be desired.

Such computing environments are already used quite extensively by researchers for a wide range of serious scientific calculations. More immediately, you may find *Mathematica* useful for a wide range of homework sets. Beware: using *Mathematica* does not mean that you can turn your brain to idle; you will still need to understand your problem in depth, and you will need to adapt to a different style of work from what you would do with pencil and paper.

This assignment gives you a feel for the range of capabilities found in *Mathematica*: the best way to learn about it is to try out a few examples. You can run *Mathematica* on the physlab workstations by simply typing `mathematica` into a terminal. The best documentation for particular functions in *Mathematica*is found in the program itself; Help - Wolfram Documentation or just hit F1. The trick then becomes knowing which function to use for a given job.

The Assignment, Part 1

1. Take a quick tour of *Mathematica*: visit `http://www.wolfram.com/products/mathematica/tour`, or (even better) run `mathematica` at the shell prompt, then explore the Help bar menu, particularly the "Virtual Book". You could google "Hands-on Start to Mathematica" to get a video introduction to various topics, or you might also google the older "A Practical Introduction to *Mathematica*", which provides a shortened version of the material contained in the Virtual Book.

2. Explore the functionality of *Mathematica* on your own, inventing your own simple problems to solve. Be a little creative with your home-made problems, and try to experiment with a large range (or even expand the range) of the functions you have seen in part 1. Don't spend longer than fifteen or twenty minutes on this part before moving on to the rest of the assignment.

Series Expansion

You have already encountered series expansions earlier in Ph20, in the assignment on Runga-Kutta methods. *Mathematica* provides numerous tools for approximating functions with series expansions.

The Assignment, Part 2

This part of the assignment will require you to use several functions and features of *Mathematica* including possibly:

- **Series**[ ]
- **Normal**[ ]
- **NumberForm**[ ]
- **Plot**[ ]
- The difference between := and = in making function definitions
- Replacement rules (**ReplaceAll**[ ] **or /.**)

1. Create a *Mathematica* notebook. Define functions SerCos[x,n] and SerSin[x,n] as the series expansions around the point x=0 of Cos[x] and Sin[x] out to n terms. Make sure you have written the functions so you can both evaluate them for a particular value of x and plot them as a function of x—if this doesn't work the first time, read more about replacement rules! Evaluate the difference between SerSin[x,n] and Sin[x] for a number of discrete values. Plot the difference as a function of x.

2. We all know that $\cos^2[x]+\sin^2[x] = 1$. How does (SerCos$[x, n]^2$+SerSin[x, n]^2) behave? What goes wrong, graphically and algebraically? Define functions SerCosSq[x,n] and SerSinSq[x,n] as the series expansions of Cos^2[x] and Sin^2[x]. How does (SerCosSq[x,n] + SerSinSq[x,n]) behave?

Euler Angles

Sooner or later, most scientists and engineers run into the problem of rotations in 3-dimensions. Calculations involving such rotations can often become quite tedious and involved: clever application of trigonometric identities is essential and avoiding "sign errors" is a significant challenge. Symbolic manipulations programs such as *Mathematica* are extremely useful in such cases as they are quite diligent in employing identities for simplification (although sometimes they have to be "coaxed" in the right direction). Also, symbolic manipulation programs don't make sign errors. This part of the assignment is meant to illustrate the capability of *Mathematica* to carry out tedious algebraic calculation.

A 2-dimensional rotation through an angle θ in the (x,y) plane is well known:

$$\begin{aligned} x' &= x\ \cos[\theta] - y\ \sin[\theta] \\ y' &= x\ \sin[\theta] + y\ \cos[\theta] \end{aligned}$$

in matrix notation this is written as

$$\begin{bmatrix} \cos[\theta] & -\sin[\theta] \\ \sin[\theta] & \cos[\theta] \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

For 3 dimensions the general transformation of a vector into a new vector looks like:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \\ a_{31}x + a_{32}y + a_{33}z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

This general vector transformation can include translations, rotations, scalings, Lorentz boosts (relativistic transformations), etc. Here we will concentrate on rotations. In 3 dimensions we can represent rotations around the x, y, and z axes as:

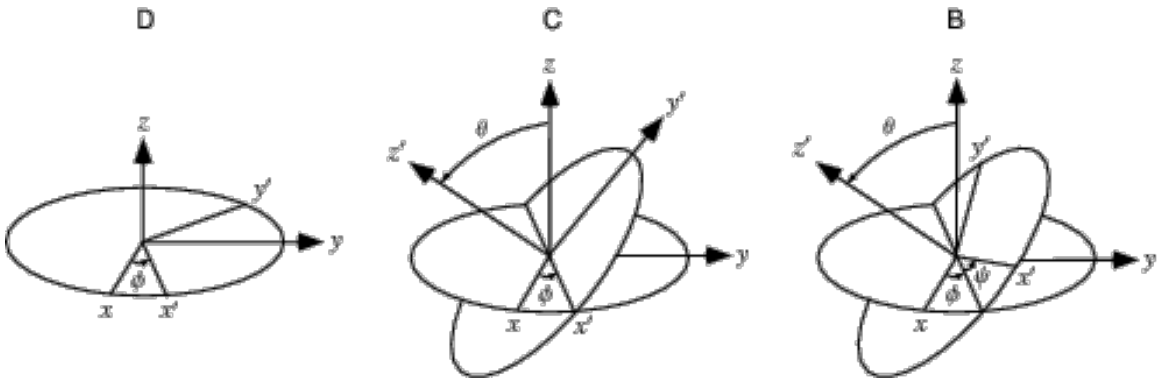
$$\begin{aligned} R_x[\theta] &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos[\theta] & -\sin[\theta] \\ 0 & \sin[\theta] & \cos[\theta] \end{pmatrix} \\ R_y[\zeta] &= \begin{pmatrix} \cos[\zeta] & 0 & \sin[\zeta] \\ 0 & 1 & 0 \\ -\sin[\zeta] & 0 & \cos[\zeta] \end{pmatrix} \\ R_z[\phi] &= \begin{pmatrix} \cos[\phi] & -\sin[\phi] & 0 \\ \sin[\phi] & \cos[\phi] & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

A rotation of a vector about the z-axis by an angle ϕ can thus be written as:

$$R_z[\phi] \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

or in *Mathematica* notation as:

Figure 1: Euler Angles



$$R_z[\phi].\{x,\ y,\ z\}$$

A rotation of a vector about the z-axis followed by a rotation around the x-axis can be written as:

$$R_x[\theta].\left(R_z[\phi].\{x,\ y,\ z\}\right) = R_x[\theta].R_z[\phi].\{x,\ y,\ z\}$$

Euler proved that any rotation in three dimensions can be written as a sequence of 3 individual rotations represented by the following matrix product:

$$R_z[\psi].R_x[\theta].R_z[\phi].\{x,\ y,\ z\}$$

The angles ψ, θ, ϕ are called Euler angles. A graphical representation of the rotations is shown in Figure 1. See e.g. `http://mathworld.wolfram.com/EulerAngles.html` for additional details.

The Assignment, Part 3

This part of the assignment will require you to use several functions and features of *Mathematica* including possibly:

- **Inverse**[ ]
- **IdentityMatrix**[ ]
- **Simplify**[ ] **and FullSimplify**[ ]
- Matrix multiplication using **.**
- **MatrixForm**[ ]

1. Define $R_x[\theta]$, $R_y[\zeta]$, and $R_z[\phi]$ rotation matrices in *Mathematica*.
2. Find an expression for Rot3$[\psi, \theta, \phi]$, the general rotation matrix: $R_z[\psi].R_x[\theta].R_z[\phi]$. See if it simplifies.

3. An expression for the inverse rotation is obviously the product of the rotations that undoes the initial rotations one-by-one: Rot3Inverse$[\psi, \theta, \phi] = R_z[-\phi].R_x[-\theta].R_z[-\psi]$. Compute Rot3Inverse$[\psi, \theta, \phi]$.Rot3$[\psi, \theta, \phi]$. Simplify it using **Simplify[ ]**.

4. Another expression for the inverse rotation matrix of *Rot*3 can be found using the *Mathematica* function **Inverse[ ]**. Compute an expression for Inverse[Rot3$[\psi, \theta, \phi]$].Rot3$[\psi, \theta, \phi]$ in terms of the angles $\{\psi, \theta, \phi\}$. Use **Simplify[ ]** to simplify the expression. The two different expressions for the inverse of Rot3[ ] must be identical. Show this by computing the difference between the two.