Mathematica Project 2

Below are two projects for Mathematica. You are expected to complete and turn in *at least* one of them. For extra credit you may do both and turn them in separately. How extra credit will be incorporated is to be determined, but it is better to do one project well than do two projects poorly.

You must turn in one assignment portion on the last day of class, **May 2nd 2018**.

If you wish to try for extra credit, you may attempt the second project and turn it in at the final on **May 9th, 2018**.

Both projects utilize new methods of numerical approximation not covered in class, and are introduced with new Mathematica commands, denoted by this typeface. If you so choose, you may do the project in a different programming language, although you may have difficulty finding analogues of some necessary commands.

- **Project 2A** (Finite differences and harmonic oscillators) introduces a method of approximation distinct from all previous ones, and applies it to a situation from physics. This method is unlike anything we have seen, but it is fairly straightforward.

- **Project 2B** (Runge-Kutta and chaos) builds upon the techniques of Euler's method and the improved Euler's method, and applies it to a famous system that has entered pop-culture parlance through the idea of a 'butterfly effect'. This project is more difficult, but should serve to be both more beautiful and more rewarding.

New Mathematica commands for these Projects

Both projects have accompnying *.nb* files that use functions necessary for these projects.
The new commands are as follows:

NSolve[ ] (Link to Documentation)

Solves systems of equations as replacement rules.

/. (Link to Documentation)

Evaluates a variable as per rules.

Graphics[Line[ ]] (Link to Documentation)

Draws a list of points pairwise connected by line segments in the plane.

Graphics3D[Line[ ]] (Link to Documentation)

Same as above but for 3D space.

Project 2A

Method: Finite Differences

While Euler's can be used to approximate first-order IVPs, we can use a new tool called finite difference approximation (dated to 1760s) to approximate solutions to second-order linear BVPs, $\frac{d^2y}{dx^2}+P(x)\frac{dy}{dx}+Q(x)y = f(x), y(a) = \alpha, y(b) = \beta$.
Let $a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$ be partition $[a, b]$ by $x_i = a + ih$, where $h = (b - a)/n$ (just like calculus!)
For every i except 0 and n, the ODE is approximated by the *finite difference equation*

$$\left(1 + \frac{h}{2}P(x_i)\right) y_{i+1} + \Big(- 2 + h^2Q(x_i)\Big)y_i + \left(1 - \frac{h}{2}P(x_i)\right) y_{i-1} = h^2 f(x_i)$$

Plugging everything in, we should achieve a system of $n - 1$ linear equations of real variables $y_1, \ldots y_{n-1}$.
Solving this system for the variables y_i gives us an approximation of $y(x)$ with global truncation error $\mathcal{O}(h)$.
The details of this method can be found in §9.5 On OnCourse is a mathematica notebook applying Finite Differences to

$$\frac{d^2y}{dx^2} + 3\frac{dy}{dx} + 2y = 4x^2, y(1) = 1, y(2) = 2 \text{ with } n = 10$$

Application: Forced Harmonic Oscillators with Dampening

The motion of a harmonic oscillator (think: spring) can be modeled by the second order ODE:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = g(t)$$

where m is the mass of the oscilator, making $m\frac{d^2x}{dt^2}$ the restorative force
$k > 0$ is the spring constant that depends on the material of the oscilator
$b > 0$ is a friction coefficient, making $b\frac{dx}{dt}$ the force due to drag
$g(t)$ is the driving force of the spring.
For this example, we will consider a spring being acted on by a sinusoidal driving force.

Your Assignment:

1. Program an Finite Differences approximation of a forced harmonic oscillator with dapening having parameters:
   $$m = 20 \qquad b = 1 \qquad k = 100 \qquad g(t) = 4\sin(2t) \qquad x(0) = x(100) = 0 \qquad n = 10$$
   and save the image of the approximation.
2. Repeat the process but instead increase the number of partitions to $n = 20$ and save the image.
3. Repeat the process but instead increase the number of partitions n to $n = 100$
4. Repeat the process but instead increase the number of partitions n to $n = 500$
5. Repeat the process but instead increase the number of partitions n to $n = 1000$
6. State any observations you have as you changed the n.

Note that for this one you must especially be careful in how you program it, because you will querie Matheamtica to solve a system of 1000 equations with 1000 unknowns. (And you thought 4×4 matrix in Math 221 was crazy)

Project 2B

Method: Runge-Kutta

We previously introduced Euler's method and the improved Euler's method as tools for numerically approximating solutions to an IVP given by $\frac{dx}{dx} = f(t,x)$, $x(t_0) = x_0$. We similarly define an m-th order Runge-Kutta method by

$$x_{n+1} = x_n + h(w_1k_1 + \ldots + w_mk_m),$$

where the k_i are evaluations of f, and the w_i are weights that satisfy $w_1 + \ldots w_m = 1$ and several other relationships that come from the taylor polynomial centered at x_0. The actual manner of finding k_i and w_i is very complicated, and discussed in §9.2 in the book. The truncation error of each term is $\mathcal{O}(h^{m+1})$, making the total truncation error $\mathcal{O}(h^m)$.

The framework was developed around 1900, and is still in use today. We already know the simplest Runge-Kutta methods as Euler's method (first order) and improved Euler (second order). The gold standard of numeral approximation is the fourth-order Runge-Kutta method, denoted RK4, sometimes called *the* Runge-Kutta method. It is given by:

$$\begin{aligned}
x_{n+1} &= x_n + \frac{h}{6}\Big(k_1 + 2k_2 + 2k_3 + k_4\Big) \\
k_1 &= f(t_n, x_n) \\
k_2 &= f\left(t_n + \frac{1}{2}h,\ x_n + \frac{1}{2}hk_1\right) \\
k_3 &= f\left(t_n + \frac{1}{2}h,\ x_n + \frac{1}{2}hk_2\right) \\
k_4 &= f\left(t_n + h,\ x_n + hk_3\right)
\end{aligned}$$

On OnCourse is a mathematica notebook applying RK4 to the IVP $\frac{dx}{dt} = e^t\cos(xt)$, $x(0) = 1$ to approximate $x(4)$.

Application: Chaos Theory

In 1963, mathematician Edward Lorenz developed a system of ODEs to model convection and temperature variation in a layer of fluid uniformly warmed from below and cooled from above. His model is as follows:

$$\begin{aligned}
\frac{dx}{dt} &= \sigma(y - x) \\
\frac{dy}{dt} &= x(\rho - z) - y \\
\frac{dz}{dt} &= xy - \beta z
\end{aligned}$$

Where σ, ρ, β are constants that depend on the fluid itself and the physical shape of the layer. Such a non-linear system must be numerically approximated, but a curious thing arises when one does so – a small change to the parameters or initial conditions will, when t sufficiently large, output vastly different values. (This can be stated more rigourously via $\varepsilon - \delta$). This system is thus said to be *chaotic* or *sensitive to initial conditions*. Lorentz discussed this in a 1972 article, *Predictability: Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas* exploring the idea of whether the presence of minuscule change in atmospheric conditions (one flap of a butterfly's wing) could produce drastic different outcomes (a tornado across half-way the world) that otherwise would not have occurred . This idea of a 'butterfly effect' impacted the sciences, mathematics, philosophy, and science-fiction deeply.

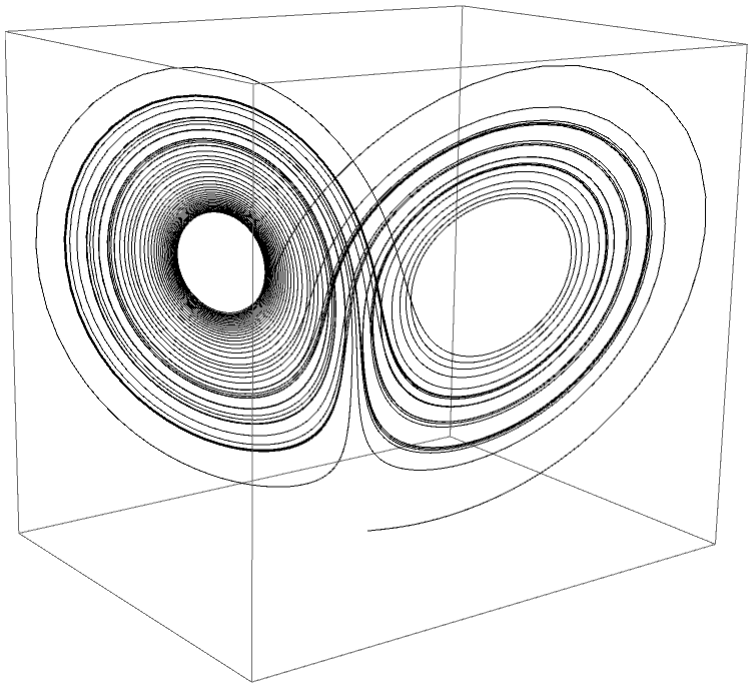
Your Assignment:

1. Program an RK4 approximation of a Lorenz system with parameters:

$$\sigma = 10 \qquad \rho = 25 \qquad \beta = 8/3 \qquad t_0 = 0 \qquad x_0 = y_0 = x_0 = 1 \qquad h = 0.01$$

   On OnCourse is a demonstration of RK4 applied to a single ODE. You should base your code on this demonstration, noting that each variable gets its own k_1, k_2, k_3, k_4 and that *each* k_i's depend on *all* of the k_{i-1}'s.

2. Using The command Graphics3D[ ], output a graph of your triples of points (x_n, y_n, z_n). It should look as such:



3. Approximate $(x(50), y(50), z(50))$ and take note of it.

4. Run the code again but with initial conditions for x changed to be $x(0) = 1.0001$. Record $(x(50), y(50), z(50))$.

5. Run the code yet again but with initial conditions for x changed to be $x(0) = 0.9999$. Record $(x(50), y(50), z(50))$.

6. Run the code one last time but changing $\rho = 23$. Record $(x(50), y(50), z(50))$.

7. Read the linked article *Predictability*, and make any observations about the outputs you had that you can. Feel free to play with the parameters and initial condiitions further.

When you turn it in, you should be handing me your code, ONE graphic similar to the one above, four triples of numbers, and some sentences on observations (it does not have to be a lot). If you are struggling, perhapse try to do the assignment with Euler's method first, and then, once it is working, beef up Euler's to RK4. §9.4 in the book may also prove helpful.