ESC 407H
Computational Methods in Engineering Science
Homework 1
Getting Up to Speed With MATLAB

Due Date: Tuesday, September 8 at 11:59 p.m.

Problem 1

Viète's formula for approximating π, which is

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots$$

is named after François Viète who published it in 1593. It is the first instance of an infinite product in mathematics and the first example of an explicit formula for the exact value of π. It can be interpreted as a variation of an Archimedes idea of approximating the length of a circle by the perimeter of a many-sided polygon. It is remarkable that it only uses the number 2 to approximate π as accurately as desired.

Write a MATLAB function called `viete` that approximates π using this formula. It should take as input the number of terms n to use in the product on the right-hand side of the formula. The output should be the corresponding approximation for π.

Next, write a script called `viete_run` that calls `viete` starting with $n = 1$ and then repeatedly calls `viete`, increasing n by one on each call up to $n = 25$. For each call, you should save the approximation for π and the true relative error between the approximation and MATLAB's built-in approximation for π, which is given by the built-in function `pi`. We will soon see that the *true relative error* is given by

$$e_{\text{tr}} = \frac{\text{true val} - \text{approx val}}{\text{true val}}.$$

Finally, plot the approximations for π versus n and plot the relative error versus n. For the error plot, use a semilog plot and comment on what you see.

Problem 2

Write a MATLAB function m-file called `days` that determines the elapsed days in any year. The function should be called as follows:

```
>> n_days = days(mo,da,yr);
```

where `mo` is the month, properly capitalized and entered as a string,* `da` is the numerical value of the day of the month, and `yr` is the numerical value of the year. Take into account leap years. Recall that a leap year occurs every year that is exactly divisible by four, except for years that are exactly divisible by 100. Years that are exactly divisible by 400 *are* leap years, even though they are also divisible by 100. A nice way to do this is with `for` and `switch` structures.

*It is February, not Febuary.

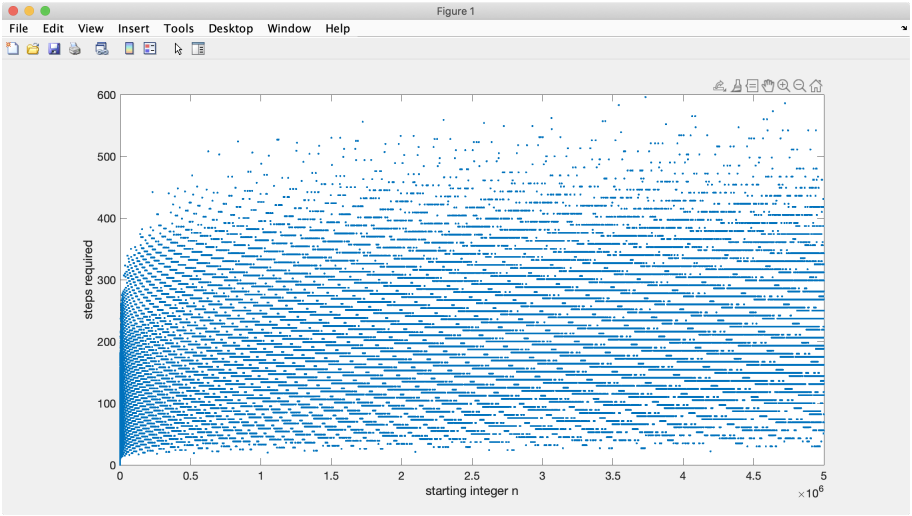
Problem 3

Generate a sequence in the following manner. Start with any positive integer n. If n is even, then divide it by two to generate the next term in the sequence, if n is odd, then generate the next term using $3n + 1$. Keep performing the same operations for each new term generated in the sequence. The German mathematician Lothar Collatz conjectured in 1937 that the sequence will *always* reach 1. I want you to write a script that proves that the Collatz conjecture is true for every integer up to and including $n = 5{,}000{,}000$.*

Your script should

- use MATLAB's `tic` and `toc` functions to time how long it takes to do the main part of the script (see `https://bit.ly/2MydsLS`),
- store the final value of the sequence for each n,
- store the number of steps in the sequence for each n,
- store the starting integer for each n that does not reach 1,
- report if all sequences reach 1,
- report all starting integers whose sequences do not reach 1.

If you use a `while` loop, the script should check if a sequence becomes longer than 10,000 steps, and if it does, break out of the loop. Finally, your script should plot the number of steps in each sequence for each n. The plot should look like this



Make sure you preallocate all large arrays (see `https://bit.ly/2Bz2jGy`).

*This is an unsolved problem in mathematics. It has never been proven that this sequence ends at 1 for all positive integer n.

Problem 4

Let's do some infectious disease modeling. We will use the SIR model, which is sometimes referred to as a *compartmental model* since it separates the population into compartments. The compartments in the SIR model are

susceptible those individuals who are healthy but can still be infected,

infected obviously those who are infected,

recovered those who have recovered from the infection, but who cannot get infected again.

All of these are functions of time. Let $S(t)$, $I(t)$, and $R(t)$ represent the numbers of susceptable, infected, and recovered individuals, respectively. To make things a little nicer going forward, given that N is the total population, we will define

$$s(t) = S(t)/N = \text{the susceptible fraction of the population,} \tag{1}$$
$$i(t) = I(t)/N = \text{the infected fraction of the population,} \tag{2}$$
$$r(t) = R(t)/N = \text{the recovered fraction of the population.} \tag{3}$$

It should be obvious that $s(t) + i(t) + r(t) = 1$. In addition, assume that we have a closed population, that is, there are no births or immigration, so no one is added to the susceptible group.

Now, let β be the expected number of people an infected person infects per day. It should be clear that the number of days a person has and can spread the disease is important—let's call this number ε. Say $\beta = 2$ and $\varepsilon = 7$, then an infected person infects two people per day for seven days, so that person infects 14 other people. This number is called the *basic reproduction number*, R_0, so we see that $R_0 = \beta\varepsilon$.

Next, let $\gamma = 1/\varepsilon$, which you can think of *rate of recovery* or the proportion of infected people that recover each day. For example, if you have 40 infected people and $\varepsilon = 10$, then $1/10$ of those 40 people, or 4 people, will recover each day. This means that we can now write $R_0 = \beta/\gamma$. Let's now figure out the equations governing the spread of the disease.

Let's start by figuring out how $S(t)$ changes. Say, at time t, we have $N = 100$, $R = 20$, $I = 30$, $S = 50$, and $\beta = 1$. Then on that day, the number of newly infected people will be $30(1)(50)/100 = 15$. As a differential equation, this becomes

$$\frac{dS}{dt} = \dot{S} = -\beta IS/N \quad \Rightarrow \quad \dot{s} = -\beta is, \tag{4}$$

where I have divided both sides by N to get the last equation.

Now let's figure out how the number of infected people changes with time. The number of people leaving S become part of I, so the equation governing $I(t)$ looks just like the equation governing $S(t)$ with a change in sign, that is

$$\frac{dI}{dt} = \dot{I} = \beta IS/N \quad \Rightarrow \quad \dot{i} = \beta is. \tag{5}$$

This is missing something. Remember that some of the infected people recover, so if we use the $\gamma = 1/10$ that we used above, then the number of infect people *decreases* by $\gamma I = 30/10 = 3$. Thus, our differential in i becomes

$$\dot{i} = \beta i s - \gamma i. \tag{6}$$

The last differential equation governs how the number of recovered people changes with time. That's just the number we just calculated, so we have

$$\dot{r} = \gamma i. \tag{7}$$

Equations (4), (6), and (7) are a system of three, first-order, nonlinear ordinary differential equations.

Adding Exposed People and Dead People

Most infectious diseases have an incubation period during which the host cannot spread the disease. Let's call this group of people *exposed* and let $e(t)$ be the exposed fraction of the population. Let τ be the incubation period in days so that $\delta = 1/\tau$ will be the rate at which exposed people become infectious. Following the previous discussion, it's not hard to see that the governing equations now become

$$\begin{aligned}
\dot{s} &= -\beta i s, \\
\dot{e} &= \beta i s - \delta e, \\
\dot{i} &= \delta e - \gamma i, \\
\dot{r} &= \gamma i.
\end{aligned}$$

There are serious infectious diseases for which people do not survive. This component becomes important. It is easy to see that when people become infectious, they either recover or they die. For this new variable $d(t)$, let ρ be the rate at which people die. For example, if it takes eight days to die, ρ will be $1/8$. The last thing we need is the death rate or probability, which we will call α. That implies that the recovery probability is $1 - \alpha$. Introducing this final variable, out system of differential equations becomes

$$\begin{aligned}
\dot{s} &= -\beta i s, \\
\dot{e} &= \beta i s - \delta e, \\
\dot{i} &= \delta e - \gamma(1-\alpha) i - \alpha \rho i, \\
\dot{r} &= \gamma(1-\alpha) i, \\
\dot{d} &= \alpha \rho i.
\end{aligned}$$

These are the equations you will solve.

Numerically Solving the Differential Equations

Let v denote any one of the five dependent variables. We can approximate the $\dot{v}$ at every time step Δt using*

$$\dot{v}(t) = \frac{dv}{dt} \approx \frac{v(t+\Delta t) - v(t)}{\Delta t} \quad \Rightarrow \quad v(t+\Delta t) \approx v(t) + \dot{v}(t)\Delta t.$$

*Taking the limit as $\Delta t \to 0$ is the definition of the derivative.

For example, the five equations would look like

$$s(t+\Delta t) \approx s(t) - \beta i(t) s(t) \Delta t, \tag{8}$$
$$e(t+\Delta t) \approx e(t) + \left[\beta i(t) s(t) - \delta e(t)\right] \Delta t, \tag{9}$$
$$i(t+\Delta t) \approx i(t) + \left[\delta e(t) - \gamma(1-\alpha) i(t) - \alpha\rho i(t)\right] \Delta t, \tag{10}$$
$$r(t+\Delta t) \approx r(t) + \gamma(1-\alpha) i(t) \Delta t, \tag{11}$$
$$d(t+\Delta t) \approx d(t) + \alpha\rho i(t) \Delta t, \tag{12}$$

where the values of each of the variables at time t are the estimates of those variables at the previous time step (the initial conditions give the original estimate for the algorithm). With sufficiently small Δt, we can repeat these steps as many times as we like, starting with v at $t = 0$, to get approximations for $v(t)$. This method for solving differential equations is due to Leonhard Euler and is called *Euler's method*.

Now, one of the things that is nice about MATLAB is the ease with which it can handle vectors and vector operations. We can take advantage of this by defining a variable `v` with five columns, where the first column contains $s(t)$, the second $e(t)$, the third $i(t)$, the fourth $r(t)$, and the fifth $d(t)$. The first row of `v` will contain the initial condition for each of the variables. You can then create an anonymous function called `f` that contains as its five elements the five differential equations. Having done that, Eqs. (8)–(12) become one *vectorized* equation

$$v(t+\Delta t) \approx v(t) + f(t, v(t)) \Delta t.$$

We saw an example of this in the MATLAB Highlights we went through in class. Note that your anonymous function must explicitly contain t as one of its input arguments, so it will look like `odes = @(t,v) ...`.

MATLAB

Write a MATLAB function called `ode_solve` that implements Euler's method of solving ordinary differential equations. It should take as input arguments the anonymous function `odes` containing the differential equations, a vector containing the initial conditions on the dependent variables, the number of days for which you want the solution, and the step size Δt. If Δt is not specified, it should use 0.1 days as the default. The output arguments should be the vector containing the solution v at every time step and the execution time for the solution. Make sure that your function has comments at the top of your function so that it is well documented for future use and so that MATLAB's `help` command will provide useful information (see https://bit.ly/2MmTWTh). Use preallocation of vectors and arrays where you can.

Write a MATLAB script called `ode_run` that solves the differential equations using $\Delta t = 1$, $N = 1{,}000{,}000$, $\tau = 5$, $\varepsilon = 10$, $\alpha = 0.05$, and $\mu = 12$. Use $N_i = 50$ for the number of people initially infected. Vary R_0 from 2 to 10 in steps of 2 to see how the infectiousness of the disease affects how it plays out. You will need to vary the number of days that you solve the equations to see this. This will require a small amount of trial and error. For each of the five solutions, plot all five dependent variables versus time and comment (as comments in your the code) on how R_0 affects the dynamics of the disease. Your script should contain all five calls to `ode_solve` after you have refined the amount of time used for each solution. Each of your five plots should have a title, legend, and the axes should be labeled. You can increase the font size on MATLAB plots using the command `figure('DefaultAxesFontSize',14)`, where `14` is the font size in points.