ENGR 105 – Introduction to Scientific Computing

Assignment 4

Due by 11:59 pm on Weds. 2/9/2022 on Gradescope

Problem 1 (30 pts): The Electronic Numerical Integrator and Computer (ENIAC), the first electronic general-purpose computer, was developed at the University of Pennsylvania to quickly compute artillery firing tables for the United States Army[1].

In homage to this system, you are to create a function that simulates the path of a projectile using time-stepped dynamics and a script that calls the function to investigate the trajectory of a projectile for a given firing velocity and variations in launch angle. Details of the physics, function, and script follow.

At any point in time the state of a 2D projectile is characterized by its position, (x,y), velocity, (v_x, v_y), and acceleration, (a_x, a_y). The projectile in the model you will develop is subject to gravity and air resistance.

In this model, the accelerations that the projectile experiences in the *x* and *y* directions are given by the following mathematical expressions.

$$a_x(t) = -cv_x(t)\sqrt{v_x^2(t)+v_y^2(t)}$$
$$a_y(t) = -g - cv_y(t)\sqrt{v_x^2(t)+v_y^2(t)}$$

where *g* is acceleration due to gravity and *c* is a measure of the damping caused by air resistance.

In this simulation, you will use a first order approximation to compute the next state of velocity and position of the projectile, as shown below.

$$v_x(t+\Delta t) = v_x(t) + a_x(t)\Delta t$$
$$v_y(t+\Delta t) = v_y(t) + a_y(t)\Delta t$$
$$x(t+\Delta t) = x(t) + v_x(t)\Delta t$$
$$y(t+\Delta t) = y(t) + v_y(t)\Delta t$$

where *t* represents the current state, $t+\Delta t$ represents the next state, and Δt (when it appears outside of parentheses) is the time step.

Develop a function that simulates the path of a projectile launched from an initial position of

[1] See this Wikipedia article on ENIAC and this ARL/Army site for descriptions of ENIAC, its history, and its use.

(x,y) = (`x0`,`y0`) and with an initial velocity of (v_x, v_y) = (`vx0`, `vy0`). The function should terminate when the projectile has hit the ground or just subtended the ground plan. You may assume that the projectile is always launched above the ground plane (`y0` > 0 always). Your function should have the following function declaration.

```
function [x,y] = projectile(g,c,x0,y0,vx0,vy0,tstep)
```

`tstep` is the time step of the simulation (i.e. Δt). All inputs are scalars.

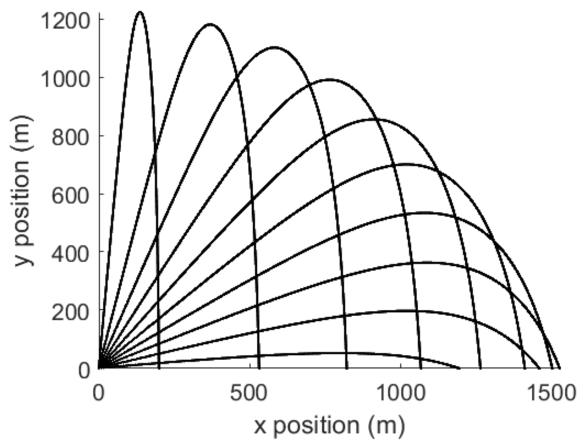
Your `projectile` function should return two row vectors, `x` and `y`, that contain the *x* and *y* positions of the projectile at every time step from launch to impact with the ground.

Create a script that repeatedly calls on function `projectile` to investigate the path of the projectile for launch angles with respect to horizontal of 5° ≤ θ ≤ 85° by 10 divisions with `g` = 9.81 m/s^2, `c` = 0.002 m^{-1}, `x0` = 0 m, `y0` = 1 m, `v0` = 820 m/s, and Δt = 0.01 s (i.e. `tstep` = 0.01 seconds).

The *x* and *y* components of velocity for a given firing angle are calculated according to the following mathematical expressions.

$$v_{x0} = v_0 \cos(\theta)$$
$$v_{y0} = v_0 \sin(\theta)$$

Your script should plot all trajectories on a single figure that looks like the following.



You will want to use the `hold on` command in your script to plot new paths over previous paths. The plot above was created using the command `plot(x,y,'k.')` and `axis tight` was specified after all plotting was completed.

Your code may not leverage built-in MATLAB ODE solvers.

Upload the projectile function, the script that produces the visualization of projectile trajectories, and a .jpg copy of the resulting plot. Identify the names of all files associated with this problem in your README.txt.

Problem 2 (25 pts): The Newton-Raphson root finding algorithm (Newton's method) is discussed in section 2.7.1 of Essential Matlab and the Topic 7 lecture slides. Use Newton's method to find the roots of the following equation.

$$y(x) = (x-2)^2 + (x-3) - 10$$

Instead of a `for` loop, your solution should employ a `while` loop that terminates after the absolute difference between the last guess (x_i) and the current guess (x_{i+1}) is less than scalar input `tol` ("tolerance"). Use the following function declaration.

```
function [N,x] = newtMethod(x,tol)
```

Scalar input `x` is the initial guess of the root location, scalar output `N` is the number of while loop iterations that were necessary to achieve the requested `tol`, and scalar output `x` is the reported root.

You may want to plot the equation $y(x)$ outside of the function to get a sense of its roots.

Call `newtMethod` with various values of scalar input `x` to find the two roots of function $y(x)$ for `tol = .5` and `tol = 1x10`$^{-9}$. Which `tol` gives a result closer to the known roots of function $y(x)$ and how many iterations `N` were required for convergence for each `tol`? You should check the roots of function $y(x)$ by hand or with another solver (e.g. Wolfram Alpha).

Submit your function as `newtMethod.m` and provide your answers and supporting remarks to the question above in your README.txt.

Problem 3 (25 pts): Simple algorithmic rules can yield interesting mathematical patterns. The Chaos game as described in this Numberphile video is one such example. The presenter creates variations of the Sierpiński triangle among other patterns. Similar complex systems based on simple algorithms have also provided insight into physics and nature.

Create a function that plots a Sierpiński triangle-like pattern using the algorithm outlined in the first two minutes of the Numberphile video linked above. Unlike the Numberphile algorithm, your algorithm can use a pseudo-random integer generation for integers 1, 2, and 3 instead of generating integers 1, 2, 3, 4, 5, and 6 as would occur with a six-sided die. The former is more convenient for indexing the 1x3 vectors that you will use to describe the vertices of the original triangle. The pseudo-random generation of an integer of value 1, 2, and 3 can be achieved using `randi([1 3])`.

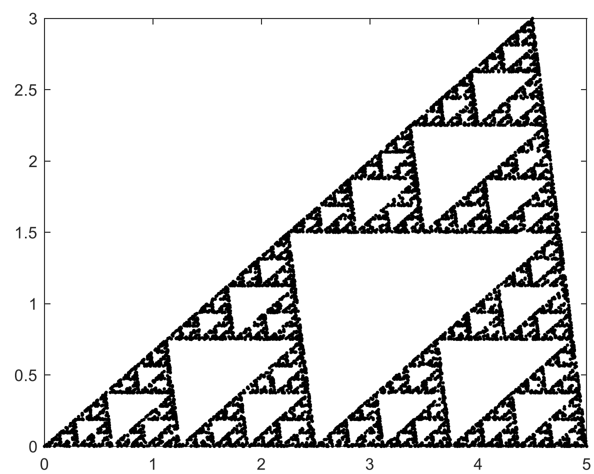
Your function should have the following function declaration.

```
sierpinski(x,y,xVertex,yVertex,N)
```

Inputs `x` and `y` are scalars and represent the x and y coordinate location, respectively, of the first point in the Sierpiński triangle-like pattern. Inputs `xVertex` and `yVertex` are 1x3 vectors and represent the x and y vertices of the triangle, respectively. Scalar input `N` is presumed integer and represents the number of additional points added to the Sierpiński triangle-like pattern.

Your function should plot the triangle vertices, initial point, and the additional `N` number of points that were generated.

As an example, the following figure was produced by invoking
`sierpinski(2,.5,[0, 5, 4.5],[0, 0, 3],10000)`



Submit your function as `sierpinski.m`. Produce an example plot based on sensible function inputs of your choice and using command line call to the function. Upload this plot in .jpg format and using a reasonable naming convention. Report the command line call and identify the function and plot filenames in your README.txt.