Process model formulation and solution, 3E4

Computer software tutorial - Tutorial 1

Kevin Dunn, dunnkg@mcmaster.ca **September 2010**

Tutorial objectives

- Three questions that will help you get comfortable (again?) with MATLAB or Python
- Creating vectors and matrices
- Mathematical manipulation of variables
- Displaying simple plots of these vectors and matrices

Recap of tutorial rules

- Tutorials can be completed by yourself, or with one other person.
- Tutorials must be handed in by the end of the tutorial session in the lab.
- Please give the TA a paper submission, no electronic submissions - thanks!

Question 1 [1]

Note: This question is a recap of creating vectors and generating plots.

A pellet that is shot vertically upward with initial speed u will have vertical displacement, s, after a time, t, given by the formula:

$$s = s_{\text{up}} + s_{\text{down}} = ut - \frac{gt^2}{2}$$

where $g = 9.81$ m/s^2 is the acceleration due to gravity, and we ignore the effect of air resistance. We would like to plot a graph that shows the vertical displacement as a function of time. Also assume that the initial speed is 50 m/s.

1. Create a vector, called `t`, that contains the time steps from $t = 0$ to $t = 10.0$ in increments of 0.2 seconds.
2. How many elements in this vector? Write down the line of code used to calculate the vector's size.
3. Create a vector, `s_up` that contains the first term in the above equation.
   - How many elements in this vector?
   - What is the smallest value in the vector?
   - What is the largest value?
4. Also create a vector `s_down`, the downward displacement of the pellet. Write down the line of code used to create this vector.
5. Next create a vector `s`, as the **sum** of `s_up` and `s_down`. Note that we can add and subtract vectors as long as they have the same dimension.
6. Create a plot of the upward displacement vs time. Your solution should contain the line of code you use.

7. Create a plot of the the downward displacement vs time.
8. Finally, create a plot of the total displacement vs time.
    - label your x- and y-axes with labels that also contain the *units of measurement*,
    - give the plot a suitable title,
    - and make sure there is a grid in the background.

Your solution to this question must include the intermediate questions, but you will be graded on the lines of code and the plot from the last part.

Solution

MATLAB code

```
g = 9.81;                   % [m/s^2] Gravity constant
u = 50.0;                   % [m/s]   Initial speed of pellet

%1.1
t = [0.0:0.2:10]';          % [s]     Time

%1.2
%There are 51 elements in t
size_of_t = size(t);        % -The "size()" function returns a vector
                            %  containing the length of each dimension
size_of_t                   %  of the matrix in question

%1.3
%%%%There are 51 elements in s_up
%%%%The smallest value in s_up is 0
%%%%The largest value in s_up is 500

s_up = u*t;                 % -A constant multiplying a vector
                            %  multiplies each element in the vector
size_of_s_up = size(s_up);  % -Same as above
size_of_s_up                % -Telling MATLAB to print the contents
                            %  of size_of_s_up to command line
max_s_up = max(s_up)        % -The MATLAB function "max()" returns
                            %  the largest, non-absolute, element
                            %  of a vector
min_s_up = min(s_up)

%1.4
s_down = -(g/2)*(t.^2);     % -Notice the use of the dot operator
                            %  for the exponential. Also note that
                            %  a dot operator is not required for
                            %  the multiplcation because we are
                            %  multiplying by a constant, which
                            %  is already element by element

%1.5
s = s_up + s_down;          % -The addition of two equal dimension
                            %  vectors/matrices is performed
                            %  element by element
%1.6, 1.7 and 1.8

plot(t,s_up,'-or','MarkerEdgeColor','b','MarkerFaceColor','b', ...
            'MarkerSize',2); % -Plot(x,y,plot_specs)
title('Pellet Upward Displacement vs. Time');
```

```
xlabel('Time [s]'); ylabel('s [m]');
pause;                          % -The pause command holds a given plot
                                %  until the user presses any key at the
                                % command line
plot(t,s_down,'-or','MarkerEdgeColor','b','MarkerFaceColor', 'b', ...
              'MarkerSize',2);
title('Pellet Downward Displacement vs. Time');
xlabel('Time [s]'); ylabel('s [m]');
pause;
plot(t,s,'-or','MarkerEdgeColor','b','MarkerFaceColor','b', ...
               'MarkerSize',2);
title('Pellet Total Displacement vs. Time');
xlabel('Time [s]'); ylabel('s [m]');
grid on
% Solution prepared by Elliot
```

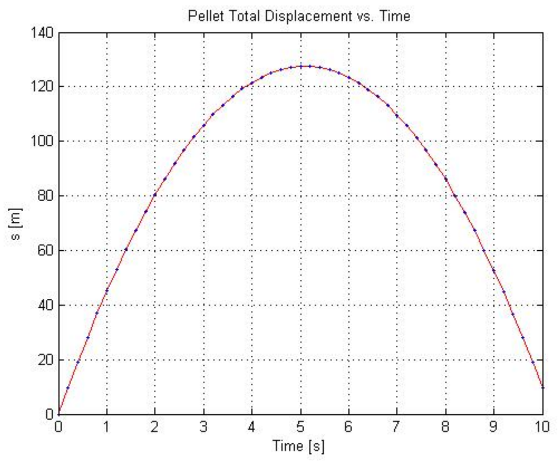
Python code

```
import numpy as np                  # Always start with this line
from matplotlib.pylab import *      # Include if you are going to plot

g = 9.81                            # [m/s^2] Gravity constant
u = 50.0                            # [m/s]   Initial speed of pellet

# Part 1.1
t = np.arange(0, 10.001, 0.2)       # type: help(np.arange)
                                    # to see why we did use "10.001"

# Part 1.2
size_of_t = len(t)
print(len(t))                   # 51


# Part 1.3
s_up = u * t                    # All Python operations are
                                # "element-by-element"

print(min(s_up))                # 0.0
print(max(s_up))                # 500.0

# Part 1.4
s_down = -(g * t**2)/2.0        # Python version 2.x peculiarity:
                                # make sure your divisor is "2.0",
                                # not just "2".  This is not an issue
                                # in Python vesion 3.x

# Part 1.5
s = s_up + s_down

# Part 1.6
plot(t, s_up, '.-')             # '-.' means "use dots, connected
                                #      by straight lines"
title('Upward pellet displacement')
xlabel('Time [s]')
ylabel('Displacement [m]')
grid('on')

# Part 1.7
figure(2)                       # Create a second figure window
plot(t, s_down, '.-')
```

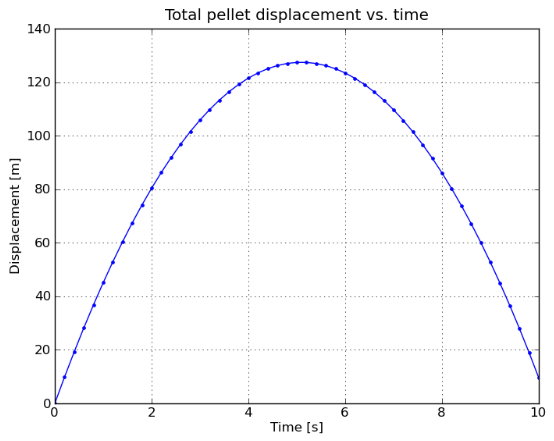
```
title('Downward pellet displacement')
xlabel('Time [s]')
ylabel('Displacement [m]')
grid('on')

# Part 1.8
figure(3)
plot(t, s, '.-')
title('Total pellet displacement vs. time')
xlabel('Time [s]')
ylabel('Displacement [m]')
grid('on')
savefig('tut-1-q1-8.png')

# Solution prepared by Kevin
```

Plots

MATLAB



Python



Question 2 [2]

Note: This question is a recap of for-loops and while-loops.

The square root of y, a positive number, can be approximated using Newton's method, a method we will learn about later in the course. The method requires an initial guess, and then iterates (repeats over and over) certain calculations, until the approximation does not improve any more. Here is some pseudo-code for the method:

1. Assign a value to y
2. Set the initial guess of the final answer to $x = y/2.0$
3. Repeat a few times, in this tutorial, repeat 5 times:
    - Replace x with $\dfrac{x + y/x}{2.0}$. In the future we will write such statements as: $x \leftarrow \dfrac{x + y/x}{2.0}$
    - Print the value of x to the screen
4. Stop and compare your answer with the built-in `sqrt()` command.

Answer these questions:

- Implement this pseudo code in either MATLAB or Python and show the printed output when calculating $\sqrt{13}$. MATLAB users: please type `format long` so that the output shows 15 digits of precision; Python shows the full precision by default.
- Change your code so that the 3rd step is replaced with a `while`-loop that will stop the code when the change in x from the previous iteration to the current iteration is smaller than 1×10^{-8}.
- How many iterations are required to calculate $\sqrt{65536.0}$ with this criterion?

Solution

MATLAB code

```
%2.1
format long % -Sets MATLAB's default output to its full precision
y = 13.0;
%2.2
x = y/2.0;
%2.3
fprintf('sqrt(%d) = %d \n\n',y,sqrt(y));  % -'\n' is the carriage
                                           % return character, %d is a
                                           % format string specifier
                                           % for the inputting variables
                                           % into the string

fprintf('for loop results: \n');
fprintf('%-20.15e \n',x);    % -"%-20.15e" is another form of format
                             %  string specifier (search "fprintf" in
                             %  the MATLAB help menu for more details)

for(i=1:1:5)                 % -for loop -> i starts at 1 and moves in
                             % steps of 1 to a final value of 5
                             % (thus completing 5 steps)
    x = ( x + ( y / x ) )/2.0;
    fprintf('%-20.15e \n',x);
end
```

```
tol = 1e-008;                 % -tolerance to be used in the while loop
x = y/2.0;
x_dif = 1e-004;               % -Initial guess for the relative
                              %  difference that ensures the while loop
                              %  executes at least once
x_hold = x;                   % -x_hold will always hold the value of the
                              % previous step, thus allowing a relative
                              % comparison between subsequent steps

i = 0;                        % -Initializes the loop iteration counter
fprintf('\n while loop results: \n');
fprintf('%-20.15e \n',x);
while(abs(x_dif) > tol)      % -While loop keeps executing until the
                              % absolute difference between consecutive
                              % estimates of x is below the tolerance
    x = ( x + ( y / x ) )/2.0;
    fprintf('%-20.15e \n',x);
    x_dif = x_hold - x;      % -Calculats the difference between the
                              % current and previous estimate of x
    x_hold = x;               % -resets the previous stage holder
    i = i + 1;                % -increments the iteration counter
end
fprintf('\n Number of while loop iterations: %d \n\n',i)

tol = 1e-008;
y = 65536.0;
x = y/2.0;
x_dif = 1e-004;
x_hold = x;
i = 0;
fprintf('sqrt(%d) = %d \n\n',y,sqrt(y));
fprintf('\n while loop results: \n');
fprintf('%-20.15e \n',x);
while(abs(x_dif) > tol)
    x = ( x + ( y / x ) )/2.0;
    fprintf('%-20.15e \n',x);
    x_dif = x_hold - x;
    x_hold = x;
    i = i + 1;
end
fprintf('\n Number of while loop iterations: %d \n\n',i)
%The number of iterations required to calculate sqrt(65536.0) to the
%desired tolerance is 12

% Solution prepared by Elliot
```

Python code

```
import numpy as np

# Part 2.1: using a for-loop
#------------------------------
y = 13.0
x = y / 2.0                   # Again, in Python 2.x, always
                              # divide by a float, not an integer
for i in range(5):
    x = (x + y/x) / 2.0
    print(x)

print('The actual answer is %f' % np.sqrt(y))
```

```
# Part 2.2: using a while-loop
#------------------------------
tol = 1e-8                    # Tolerance level for the while loop
x = y / 2.0
x_prev = 0.0
iter = 0
while abs(x - x_prev) > tol:
    x_prev = x
    x = (x + y/x) / 2.0
    print(x)
    iter += 1

# One way to print the output:
print('Used ' + str(iter) + ' iterations to calculate sqrt(' + \
      str(y) + ') = ' + str(x) + '; true value = ' + str(np.sqrt(y)))

# Better way to get the same output
print('Used %d iterations to calculate sqrt(%f) = %.15f; true '
      'value = %.15f ' % (iter, y, x, np.sqrt(y)))

# Part 2.3: using a while-loop
# ----------------------------
y = 65536.0
tol = 1e-8                    # Tolerance level for the while loop
x = y / 2.0
x_prev = 0.0
iter = 0
while abs(x - x_prev) > tol:
    x_prev = x
    x = (x + y/x) / 2.0
    print(x)
    iter += 1

print('Used %d iterations to calculate sqrt(%f) = %f; true '
      'value = %f ' % (iter, y, x, np.sqrt(y)))

# It is more efficient to create a function to calculate this.
# See the "Software tutorial" on the course website for details

# Solution prepared by Kevin
```

Selected MATLAB output:

```
sqrt(13) = 3.605551e+000

 while loop results:
6.500000000000000e+000
4.250000000000000e+000
3.654411764705882e+000
3.605877914546100e+000
3.605551290258318e+000
3.605551275463990e+000
3.605551275463990e+000

 Number of while loop iterations: 6

sqrt(65536) = 256

 while loop results:
3.276800000000000e+004
```

```
1.638500000000000e+004
8.194499877937138e+003
4.101248718697424e+003
2.058614121064604e+003
1.045224565257789e+003
5.539624834270534e+002
3.361332618959818e+002
2.655517766166163e+002
2.561717865301000e+002
2.560000575992625e+002
2.560000000000065e+002
2.560000000000000e+002

 Number of while loop iterations: 12
```

Selected Python output:

```
Used 6 iterations to calculate sqrt(13.000000) = 3.605551275463990; true value = 3.605551275463989

16385.0
8194.49987794
4101.2487187
2058.61412106
1045.22456526
553.962483427
336.133261896
265.551776617
256.17178653
256.000057599
256.0
256.0
Used 12 iterations to calculate sqrt(65536.000000) = 256.000000; true value = 256.000000
```

Bonus question [1.5]

The population count of the United States can be modelled using the formula:

$$P(t) = \frac{197\,252\,000}{1 + e^{-0.03134(t-1913.25)}}$$

where the time, t, is given in years. Write some code that will calculate the population count every **ten** years, from 1800 to 2010. Plot the results and label the plot appropriately.

Does the population ever reach steady-state? If so, roughly when, and what is the steady-state value?

Comment on whether or not this model is realistic. Is it useful?

Solution

Note: Please note that Elliot's solution is much more than was required to get full grade for this question. But please read his solution to gain some extra insight.

By the very definition of the model the population will approach but never reach the steady state value of 197252000 (the carrying capacity). Extrapolating the model, however, would seem to indicate that the steady state will be reached around **2010 - 2050** (it's hard to get a firm number).

In terms of the model's "usefulness" and "realism", as with anything in modelling, the answer really depends on what you are trying to achieve. The model is the classic Verhulst-Pearl population equation or simply the "Logistic equation" (for more information consult http://en.wikipedia.org/wiki/Logistic_growth). The basic form of the equation is $\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right)$, which has an analytical solution of:

$$P(t) = \frac{K}{1 + e^{-rt - \ln \frac{P_0}{K - P_0}}}$$

where:

- K = carrying capacity (in our case 197252000)
- r = growth rate (in our case 0.03134)
- P_0 = the initial population

The logistic equation is an ideal case where growth is assumed proportional only to the current population as well as the level of renewable resources available. The steady state achieved by the model represents the hypothetical population that could be sustained indefinitely given a constant resource level. Of course this is a gross over simplification and so the model is not very realistic.

However, depending on the accuracy of prediction one desires, this model could be useful in making short term predictions of population growth. Given that the population and infrastructure of a nation is a dynamic thing one would expect K and r to change over time (as various resources are discovered, exhausted, improved, etc.) but in the short term (assuming up to date estimates of K, r, and P_0) this could be a useful tool for mapping population growth. As with any model though, always remember that the wider you make your extrapolation window, the more erroneous your approximation is likely to be.

As a famous statistician once said, *Essentially, all models are wrong, but some are useful.* (George E .P. Box)

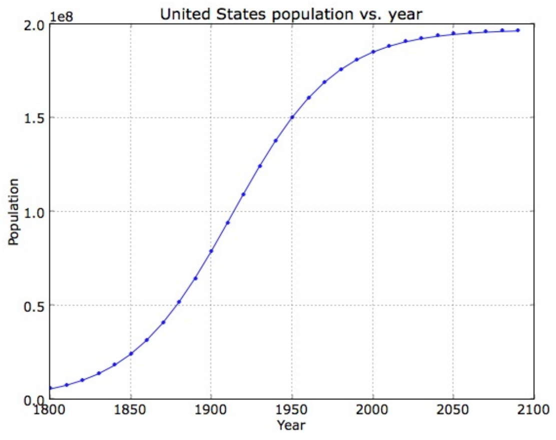
MATLAB code

```
t = [1800:10:2100];
P = 197252000./(1+exp(-0.03134*(t-1913.25)));
plot(t,P,'-or','MarkerEdgeColor','b','MarkerFaceColor','b','MarkerSize',2);

title('United States Population vs. Year');
xlabel('Year');
ylabel('Population');
grid on
```

Python code

```
import numpy as np
from matplotlib.pylab import *

t = np.arange(1800, 2100, 10)
P = 197252000 / (1 + np.exp(-0.03134 * (t-1913.25)))

plot(t, P, '.-')
title('United States population vs. year');
xlabel('Year')
ylabel('Population')
grid('on')
savefig('tut-1-qb1-Python.jpg')
```

Python figure



END