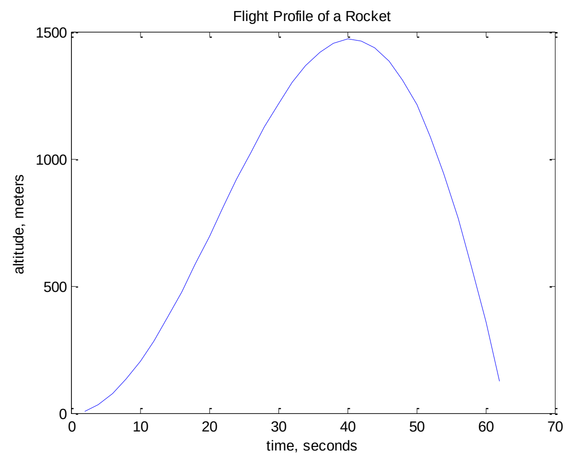
EE202 Lab#4

1. [6.1, 8.1] The altitude of a rocket (in meters) as a function of time from launch, can be modeled by the following equation:

   $a = 2.13t^2 – 0.0013t^4 + 0.000034t^{4.751}$

   Create a row vector of time from 0 to 100 at 2-second intervals.

   (a) Use the `find` function to determine when the rocket hits the ground to within 2 seconds. (Hint: The value of altitude will be positive for all values until the rocket hits the ground.)
   (b) Use the `max` function to determine the maximum altitude of the rocket and corresponding time. (Hint: Use the form `[m,i] = max(...)`)
   (c) Create a plot of the rocket from launch to impact. Place t on the horizontal axis and altitude on the vertical axis. Be sure to add a title and axis labels (Hint: `plot(t(x),a(x))` where `x` is the indices returned by the find function, `t` = time vector, and `a` = altitude vector).



2. [5.2, 7.1] Write an M-file function to find corners of a matrix and return it in a 2 by 2 matrix. Test it on a 5 by 5 and a 10 by 10 sequential matrix (1 2 3 4... etc). Hint: Use colon notation and the reshape function to quickly construct the matrices.

```
function [corners] = yourinitials_findCorners(m)
...Type your script starting here.
```

3. [5.2, 7.1] Create a frame matrix. A frame matrix is a matrix whose boundary is all ones and inside is all zeros (like a picture frame). You should write a M-file function so that when you enter n (order of a square matrix), then type the program name, you get a frame matrix. Try writing your script with the minimum number of lines and avoid loops. Turn in for n=5 and n=10.

```
function [m] = yourinitials_frameMatrix(n)
    Type your script starting here. You may use more than one line.
```

4. [5.2, 7.1]

Cramer's Rule
Source: http://en.wikipedia.org/wiki/Cramer%27s_rule\
Consider a system of linear equations represented in matrix multiplication form as

$$Ax = b$$

where the square matrix *A* is invertible and the vector $x = (x_1, \ldots, x_n)^\top$ is the column vector of the variables. Then the theorem states that:

$$x_i = \frac{\det(A_i)}{\det(A)} \qquad i = 1, \ldots, n$$

where A_i is the matrix formed by replacing the i^{th} column of *A* by the column vector *b*.

(a) Using a **for** loop, write a Matlab M-file function to solve Ax=b using Cramer's Rule. Save your answer in a column vector named xc.

```
function x = yourinitials_cramer(A,b)
% Demo of Cramer's Rule for solving A*x = b
for k=                       % 1 to number of rows, hint: type help size
    Ai = A;                  % build Ai matrix by
    ___________;             % replacing kth column with b
    x(k,1) = ______________;  % calculate kth row to the answer
end
```

(b) Using your new function solve the following three linear equations for x, y, and z.

```
3x + 4y + 6z =  1;
 x - 2y + 7z = 10;
2x + 3y - 9z = 15;
```

(c) [9.2] As introduced in my handout "Linear Algebra and Solving Linear Systems using MATLAB," the simplest way to solve a linear equation of this form for column vector x is to use the backslash operator, \. Solve the three linear equations above by using the backslash operator. Save your answer in a column vector named xb.

(d) [9.3] Another approach to solving this problem would be to use the inverse operator. Solve the three linear equations above by using the inverse operator. Save your answer in a column vector named xa.

(e) Verify the accuracy of all three methods using the norm function. For example, to find the accuracy of using Cramer's rule you would enter:

```
disp(['norm of Cramer''s Rule residual = ' num2str(norm(A*xc-b))]);
```

(f) Which of the three methods provide the best answer?

(g) Test your three methods of solution (`xc, xb, xa`) for a random 5 by 5 matrix `A` and a 5 by 1 vector b. The random matrix must be solved using all three methods (do not generate a new random matrix for each).

(h) Again check the accuracy of your three methods using the norm function and indicate which provides the best answer.

5. [7.1] Newton invented an algorithm for finding the square root of a number. Newton's algorithm approximates $\sqrt{a}$ as follows:

   Start with an initial guess $x = a$.

   $$x_{n+1} = \frac{\left(x_n + \frac{a}{x_n}\right)}{2}$$ For the first iteration ***n*** = 1.

   This calculation is repeated until some convergence criterion, *tol* is met.

   $$\frac{|x_{n+1} - x_n|}{|x|} \leq tol$$

(a) In Section 7.1 (starting on page 79) Higham implements Newton's iterative algorithm as a Matlab function. Copy Higham's function into Matlab.
(b) Calling sqrtn with only one argument (ex. `nsqrtn(5)`) sets the tolerance of eps. What line in Highman's script allows us to only send one argument and still have the tolerance defined.
(c) Test the sqrtn function by approximating the square root of 5 with a tolerance of eps and comparing it to the value calculated with the built-in Matlab function sqrt.

6. Edmund Halley (the astronomer famous for discovering Halley's comet) invented his own algorithm for finding the square root of a number. Halley's algorithm approximates $\sqrt{a}$ as follows:

   Start with an initial guess ***x*** provide by the user.

   $$y_n = \frac{1}{a}x_n^2$$ For the first iteration ***n*** = 1.

   $$x_{n+1} = \frac{x_n}{8}(15 - y_n(10 - 3y_n))$$

   These two calculations are repeated until some convergence criterion, *tol* is met.

   $$\frac{|x_{n+1} - x_n|}{|x|} \leq tol$$

   Comparing Newton's and Halley's algorithms you will see that programmatically they are almost the same. The biggest difference from a programming perspective is that Halley's algorithm requires the user to guess the answer. In fact if the user's guess is not close to the right answer, this algorithm will not converge. Newton on the other hand simply guesses the number whose square root is to be found (`x = a;`). Clearly, this guess is not close to the right answer, so this line must be deleted from Halley's function and replaced with a new argument to the function where the user can enter his guess.

To begin the translation from the Newton to the Halley algorithm, save your `nsqrt` M-file (previous problem) with a new M-file named `sqrth`. As mentioned, it should have an additional input:

```
function [x,iter] = sqrtn(a,x,tol)
```

Where **a** is the number whose square root is to be found, **x** is the user's guess, and **tol** is the tolerance. With the addition of this new argument, you will need to change the first line of code from Higham's script to read:

```
if nargin < 3, tol = eps; end
```

The user now guesses the answer so Higham's second line can be deleted

~~`x = a;`~~

Replace Newton's with Halley's algorithm and again test your function by approximating the square root of 5 with a tolerance of eps and comparing it to the value calculated with the built-in Matlab function sqrt.

7. Which square root algorithm is better Newton's or Halley's as defined by the number of iteration to reach convergence.