```
disp('hello world')
%This is a comment, they are useful
x = 2+5; %This comment is suppresed
y = 2+3 %This comment is not suppresed
x = 2 + 5, y = x+3
% First, the scalar 3 is placed in variable x
x = 3;
% Second, variable x (equal to 3) is squared, 2 is added, and the
 result
% is placed back into variable x (now equal to 11)
x = x^2 + 2;
% Third, variable x (equal to 11) and 5 are added and placed in
 variable y
y = x + 5
Pythagorean(1,1)
result = Pythagorean(1,1)
% Example code demonstrating the clear command. a is cleared from the
% workspace below
a = 5;
b = 3;
clear a
% Example of infinite loop. Try interrupting the code. Be careful with
% infinite loops!
a = 1;
while a ~= 1

end
help sin
a = sin(2*pi)
b = sum([3, 4, 5])
% Explicityly define an array with 5 numbers

x = [1, 3, 9, 11, -10.2];
% Create an array with numbers 1 though 15
x = 1:15;
% Create an array starting at 4 and increasing by 2 to 28
y = 4:2:28;
% Create an array starting at -1.3 and increasing by 0.1 to the value
 1.3
z = -1.3:0.1:1.3;
a = 1:2:6;
% Create a vector with 5 elements uniformly spaced between (and
 including) 1 and 5
a = linspace(1,5,5)
% Create a vector with 5 elements uniformly spaced logarithmically
 between
% (and including) 10^1 and 10^5
b = logspace(1,5,5)
c = []
d = ones(1,5)
f = zeros(1,5)
x = [5, 9, 4, 1, 7, 3, 4, 8];
```

```
y = x(1)
x(1:3)
x(6:8)
x(end)
x(end-1)
x([3, 6, 1])
x = [1, 2, 3]
x(1) = 5
x(1:2) = [8,9]
y = [8, 9, 3]
y([1,3]) = [5,1]
y(end-1:end) = [6,8]
z = [2, 6, 8]
z(end+1) = 4
z(end+1:end+2) = [5,7]
z(1:2:end) = 1
z(2:2:end) = 4
z(end+2) = 7
x = [5, 2, 1, 8]

x = [5, 3, 9, 4, 1, 6]
x(1) = []
x(1:2:end) = []
x = [9, 5, 8, 2]
x(1) = []
x(1:2:end) = []
x = [1 2 3 4 5];
y = [7 8 9 10 11];

% element-wise addition adds corresponding elements
z = x + y;
z1 = x.*y; % element-wise multiplication
z2 = x./y; % element-wise division
x = 1:10:1000;
% Compute the square root of each element in the array x
y = sqrt(x);
% create an array of values
x = 0:0.01:3.2;

% evaluate an expression on every element in % the array x producing a
 new array called y
y = exp(5*x).*sin(x);

% produce a plot
plot(x, y)
% Create an example vector
x = 1:2:8
% Call indices 1 and 4 of vector x
x([1,4])
% Delete the element associated with the index 4
x(4) = []
% Create new vectors via concatenation
x1 = [x,x]
x2 = [x,linspace(1,3,6)]
```

```
x = [1,4,2,5,6,4,3,1,4,2,5,6,4,3,1,4,2,5,6,4,3];
% Step 1
x(1:4:end) = [];
% Step 2
x(1:2:end) = x(1:2:end).^2;
% Step 3
%end is an inclusive range
%we concatinate a new vector that starts with the last 5 numbers and
%carries on to the first 5 numbers
x = [x(end-4:end), x(1:end-5)]
x = [5; 4; 6; 9]
% Creating a new column vector
y1 = (1:4)';
% Creating a new vector by taking the transpose of an existing one
y2 = x'
x = [5; 2; 1; 8]
% Replacing the first element
x(1) = 9
% Replacing the first two elements
x(1:2) = [4, 3]
% Replacing the second-to-last element
x(end-1) = 9
% Deleting the first and third elements
x([1,3]) = []
x = [5, 2, 1, 4, 9, 1]
L = length(x)
% Return the first half of the vector
x(1:length(x)/2)
x(1:length(x)/2) = 50;
x
1:length(x)/2
%Divides the original vector length in half
(1:length(x))/2
1:length(x)/2
1:(length(x)/2)
a = 1;
b = 2;
c = 3;
d = 4;
a+b*c/d
(a+b)*c/d
a+(b*c)/d
a1 = 1;
disp(a1)
a2 = [1,5,9];
disp(a2)
a3 = [1;5;9];
disp(a3)
%remember the semicolon makes the vector an array
a4 = [1,5,9;9,5,1];
disp(a4)
a5(:,:,1) = [1,5,9;9,5,1];
a5(:,:,2) = [2,3,4;4,3,2];
disp(a5)
```

```
length(a2)
length(a3)
length(a4)
size(a4)
size(a5)
numel(a4)
A = [2, 5, 9; 6, 8, 3]
a = [1, 2, 3];
b = [4, 5, 6];
% Concatenate vertically
A1 = [a;b]
% Concatenate horizontally
A2 = [a,b]
a = [1; 2; 3];
b = [4; 5; 6];
% Concatentate
B1 = [a;b]
% Concatenate vectors
a = [1 2 3; 4 5 6]
b = [7 8 9; 10 11 12]
% Concatenate matrices
C1 = [a;b]
C2 = [a,b]
A = [1:3;4:6;7:9]'
A(3,1)
A(1,3)
B = [1:3;4:6]
B'
A = [1:3;4:6;7:9]'
% Return a column vector of all rows in the first column
A(:,1)
%Colon returns all columns in that row?
% Return a row vector of all columns in the second row
A(2,:)
% Return a submatrix of entries in first and second rows and columns
A(1:2,1:2)
% Return a submatrix of entries in first and second rows and all
 columns
A(1:2,:)
A(5)
A(1:5)
A(:)
% Diagonal components of A
diag(A)
A = [1:3;4:6;7:9]'
% Replace the entry in the third row, first column
A(3,1) = 50
% Replace all entries in the "upper left of the matrix"
A(1:2,1:2) = 0
A(4,4) = 33
% C has not be previously defined
C(3,3) = 1
% Create new matrix
A = [1,5;2,3]
```

```
% Add 2 to every element

% Create new array by reordering the first and second row
B = [A(2,:);A(1,:)]
% Matrix addition

A


A
B
A = [1,4;2,3]
B = [1,2;3,4]
% Matrix multiplication
A*B
% Matrix multiplication with multiplication order reversed.
B*A
B = [2,1;5,3;4,6]
A = 1:3
% To perform matrix multiplication, ensure the columns of the first
 matrix
% match the rows of the second
A*B
A = [1 2; 4 8]
B = 16./A
% Define matrix A and vector b
A = [2,4;2,6]
B = [1,2;1,3]
% Perform right division

% Perform left division

% Define matrix A and vector b
A = [2,4;2,6]
b = [5;4]
% Perform left division
A\b
inv(A)*b
A = [2,4;2,6]
b = [5,4]
% Perform right division
b/A
b*inv(A)
% x vectors
x = 0:.1:10;
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,2*x-3,'r-')
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,x-3,'r-')
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,x,'r-')
% Step 1: convert to matrix format
```

```
A = [1,-1;1,-2]; %Coefficient matrix defined above
b = [0;-3]; % Constant matrix
% Step 2: solve for x and y

% Step 1: convert to matrix format
A = [1,3,1;2,2,-1;1,1,1]; %Coefficient matrix defined above
b = [1;2;-2]; % Constant matrix
% Step 2: solve for x

% Create same matrices as last time
A = [1,5;2,3]
B = [A(2,:);A(1,:)]
A
A*5
A
B
A.*B
A = [1,4;2,3];
B = [1,2;3,4];
% Matrix multiplication
A*B
% Matrix multiplication with multiplication order reversed.
B*A
B = [2,1;5,3;4,6]
A = 1:3
% To perform matrix multiplication, ensure the columns of the first
 matrix
% match the rows of the second
A*B
A = [1 2; 4 8]
B = 16./A
% Define matrix A and vector b
A = [2,4;2,6];
B = [1,2;1,3];
% Perform right division
A./B
% Perform left division
A.\B
% Define matrix A and vector b
A = [2,4;2,6]
b = [5;4]
% Perform left division
A\b
inv(A)*b
A = [2,4;2,6]
b = [5,4]
% Perform right division
b/A
b*inv(A)
% x vectors
x = 0:.1:10;
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,2*x-3,'r-')
```

```
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,x-3,'r-')
% Create a figure window of a defined size
figure('Position',[0,0,300,300]);
plot(x,x,'k-',x,x,'r-')
% Step 1: convert to matrix format
A = [1,-1;1,-2] %Coefficient matrix defined above
b = [0;-3] % Constant matrix
% Step 2: solve for x and y

% Step 1: convert to matrix format
A = [1,3,1;2,2,-1;1,1,1]; %Coefficient matrix defined above
b = [1;2;-2]; % Constant matrix
% Step 2: solve for x
x = A\b
% Known physical parameters in this system
eta = 1*10^-3; %viscosity, Pa*s
L1 = 0.5*10^-2; % m
L2 = 1*10^-2; % m
L3 = 2*10^-2; % m
R = 20*10^-6; % m

% Calculate fluidic resistance of each channel segment
R1 = 8*eta*L1/(pi*R^4)*10^-12;   % kPa*s/mm^3
R2 = 8*eta*L2/(pi*R^4)*10^-12;   % kPa*s/mm^3
R3 = 8*eta*L3/(pi*R^4)*10^-12;   % kPa*s/mm^3
clear A b   % Clear previously used terms from workspace
% Define matrix A
A(4,1:7)=[1,-1,0,0,-R1,0,0];    % We specify columns 1:7 because the
 matrix A is not yet defined
A(5,:)=[0,1,-1,0,0,-R2,0];
A(6,:)=[0,1,0,-1,0,0,-R3];
A(7,:)=[zeros(1,4),1,-1,-1];
A([1,16,24]) = 1;               % Use single index notation to specify
 sparse 1 values
disp(A)
% Define matrix b
b = [110,100,100,zeros(1,4)]';  % Note the transpose: we need b to be
 a column vector
disp(b)
% Solve system of equations
x = A\b;
disp(x)
% Order of input matters when calling a function
m = 2;
n = 3;
q = 4;
[g1,h1] = myFunction(m,n,q)
[g2,h2] = myFunction(q,m,n)
% Define a vector to input into the function
u = linspace(1,10,8)
% Call the function and store output as variable.
v_odd = returnOddElements(u)
```

```
% Determine solution 1
sol_1 = fsolve(@eqnSet,[0,0])
% Determine solution 2
sol_2 = fsolve(@eqnSet,[1,1])
% Explicitly define a vector
a = [5,3,2,6,2,1]
% Create a vector with entries starting at 1 and incrementing by 2
 until 9
b = 1:2:9
% Create a vector consisting of 5 evenly spaced values between (and
% including) 1 and 8;
c = linspace(1,8,5)
% The first two entries are explicitly defined
x(1) = 0;
x(2) = 1;
% The next eight entries must be calculated individually based upon
 the
% previous two values
x(3) = x(1) + x(2);
x(4) = x(2) + x(3);
x(5) = x(3) + x(4);
x(6) = x(4) + x(5);
x(7) = x(5) + x(6);
x(8) = x(6) + x(7);
x(9) = x(7) + x(8);
x(10) = x(8) + x(9);
disp(x)
% Explicitly define a fibonacci vector, x
x = [0,1,1,2,3,5,8,13,21,34]
% Iterate for k equaling each value in the row vector
% [1,2,3,4,5,6,7,8,9,10]
for k = 1:10
    % For each iteration of the for loop, display the value of the
 index,
    % k, squared.
    disp(k^2)
end
% jj takes the values of 1 in the first iteration, 2 in the second
% iteration, etc.
for jj = 1:5
    % The value of the computation 2*jj is stored in the jj-th
 location of
    % vector v, which grows during each iteration of the for loop
    v(jj) = 2*jj
end
disp(v)
% An arbitrary vector
v0 = [5, -2, 4, 1];
for jj = 1:length(v0)
    % In the jj-th position of v1, store the value at the jj-th
 position of
    % v0 multiplied by 2.
   v1(jj) = 2*v0(jj)
end
```

```
disp(v0)
disp(v1)
v = 1:4;
% Return the last element of v
v(end)
v(end+1) = 1
v = [];
%Vector v is initialized as an empty set and is grown by adding to its
%end-ith + 1 index within the for loop
for jj = 1:.1:2
    %jj takes the values of 1 in the first iteration, 1.1 in the
 second
    %iteration, 1.2 in the third iteration, etc. – note that non-
integer
    %values cannot be used to index a vector
    v(end+1) = 2*jj;
end
v
v = 2*(1:.1:2)
n = 0;
% n is a counter the increments by 1 in every iteration of the for
 loop and
% identifies the vector index into which jj is placed
for jj = 1:.1:2
    %n is incremented by 1 in each iteration of the loop
    n = n+1; %counter
    v(n) = 2*jj;
end
v
%For the values it must take in a row verctor
for jj = (1:5)'
    jj
end
for jj = (1:5)
    jj
end
M = [5, 3; 1, 9]
for jj = M
    jj
end
for jj = M(:)'
    jj
end
v = [];
p = [];
for jj = 1:3
    for kk = 1:3
        v(end+1) = jj;
        p(end+1) = kk;
    end
end
```

```
disp(v)
disp(p)
% Populate an array containing the first ten numbers of the sequence.
n = 10;
% Empty the vector x - we used it previously
x = [];
x(1) = 0;
x(2) = 1;
% Create each subsequent entry by taking the sum of the previous two
% entries
for k = 3:n
    x(k) = x(k-1) + x(k-2);
end


% Display the vector
disp(x)
% Examples
3>4;
3<4;
3<= 0;
3<3;

true
~true
true == ~false
% Logical statement examples



% Relational operators can also operate on variables



s = true>false
b = true + true
c = 5 + (2==2)*5
ind = 1;
% ind is incremented within the loop and terminates the while loop
 after 4
% iterations, when ind < 5 is no longer true.
while ind<5
    % The incremetnation of ind is not suppressed with a semicolon so
 that
    % you can see the line of code that are run in each iteration.
    ind = ind+1
end

% Variable to store the number of rolls until the same dice reading is
% observed
r2s = 0;
% Initialize while loop logical. As long as contRoll is true, we will
```

```
% "continue rolling"
```

```
% Max value of entries in Fibonacci sequence
n = 100;
% Define first two entries
x(1) = 0;
x(2) = 1;
% Iterate as long as the last element is less than 100.
```

```
% Define vector containing the number of entries in each successive
% Fibonacci vector.
N = round(logspace(0,8,20))
% Preallocate vectors to store time information
t_no = zeros(1,length(N));
t_yes = zeros(1,length(N));
% Calculate the time for each vector of length N(m)
for m = 1:length(N)

    % Calculate Fibonacci sequence vector without preallocation
    x = [];
    tic
    % Define first two entries
    x(1) = 0;
    x(2) = 1;
    % Create each subsequent entry by taking the sum of the previous
 two
    % entries
    for k = 3:N(m)
     x(k) = x(k-1) + x(k-2);
    end
    t_no(m) = toc;

    % Calculate Fibonacci sequence vector with preallocation
    tic;
    x = zeros(1,N(m));
    % Define first two entries
    x(1) = 0;
    x(2) = 1;
    % Create each subsequent entry by taking the sum of the previous
 two
    % entries
    for k = 3:N(m)
```

```
        x(k) = x(k-1) + x(k-2);
    end
    t_yes(m) = toc;
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot the functions
plot(N,t_no,'ko-',N,t_yes,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
% Graphical parameters
set(gca, 'XScale', 'log')   % "log" axes - x
set(gca, 'YScale', 'log')   % "log" axes - y
set(gca,'XMinorTick','on')
xlim([10^0,10^8]);  % range of x values on plot
ylim([10^-6,10^2]); % range of y values on plot
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('No preallocation','Preallocation','Location','northwest')
% Define vector containing number of entries in potential vector.
N = round(logspace(0,8,20))
% Preallocate vectors to store information
t_for = zeros(1,length(N));
t_vec = zeros(1,length(N));
% Calculate the time for each vector of length N(m)
for m = 1:length(N)
    % Create vector of random numbers with dimension 1xN(m)
    a = rand(1,N(m));

    % Calculate time for for loop
    tic
    % Preallocate for loop vector
    b_for = zeros(1,N(m));
    % Calculate each entry in b by looping through all of vector a
    for n = 1:N(m)
        b_for(n) = a(n)^2;
    end
    t_for(m) = toc;

    % Calculate time for vectorization
    tic
    % Calculate all entries in b by element-wise exponentiation
    b_vec = a.^2;
    t_vec(m) = toc;

    % Clear b vectors for next iteration
    clear b_for b_vec
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot call and graphical commands
plot(N,t_for,'ko-',N,t_vec,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
set(gca, 'XScale', 'log')
set(gca, 'YScale', 'log')
set(gca,'XMinorTick','on')
xlim([10^0,10^8]);
```

```
ylim([10^-6,10^2]);
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('for loop','Vectorization','Location','northwest')
% Populate an array containing the first ten numbers of the sequence.
n = 10;
% Empty the vector x - we used it previously
x = [];
x(1) = 0;
x(2) = 1;
% Create each subsequent entry by taking the sum of the previous two
% entries
for k = 3:n
 x(k) = x(k-1) + x(k-2);
end
% Display the vector
disp(x)
% Max value of entries in Fibonacci sequence
n = 100;
% Define first two entries
x(1) = 0;
x(2) = 1;
% Iterate as long as the last element is less than 100.
while x(end) < n
    % If the last element is less that 100, add the next element of
 the
    % Fibonacci sequence to the vector
 x(end+1)=x(end)+x(end-1);
end
x(end)=[];  % Remove last element
disp(x)
% Define vector containing the number of entries in each successive
% Fibonacci vector.
N = round(logspace(0,8,20))
% Preallocate vectors to store time information
t_no = zeros(1,length(N));
t_yes = zeros(1,length(N));
% Calculate the time for each vector of length N(m)
for m = 1:length(N)

    % Calculate Fibonacci sequence vector without preallocation
    x = [];
    tic
    % Define first two entries
    x(1) = 0;
    x(2) = 1;
    % Create each subsequent entry by taking the sum of the previous
 two
    % entries
    for k = 3:N(m)
     x(k) = x(k-1) + x(k-2);
    end
    t_no(m) = toc;
```

```
    % Calculate Fibonacci sequence vector with preallocation
    tic;
    x = zeros(1,N(m));
    % Define first two entries
    x(1) = 0;
    x(2) = 1;
    % Create each subsequent entry by taking the sum of the previous
 two
    % entries
    for k = 3:N(m)
     x(k) = x(k-1) + x(k-2);
    end
    t_yes(m) = toc;
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot the functions
plot(N,t_no,'ko-',N,t_yes,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
% Graphical parameters
set(gca, 'XScale', 'log')   % "log" axes - x
set(gca, 'YScale', 'log')   % "log" axes - y
set(gca,'XMinorTick','on')
xlim([10^0,10^8]);  % range of x values on plot
ylim([10^-6,10^2]); % range of y values on plot
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('No preallocation','Preallocation','Location','northwest')
% Define vector containing number of entries in potential vector.
N = round(logspace(0,8,20))
% Preallocate vectors to store information
t_for = zeros(1,length(N));
t_vec = zeros(1,length(N));
% Calculate the time for each vector of length N(m)
for m = 1:length(N)
    % Create vector of random numbers with dimension 1xN(m)
    a = rand(1,N(m));

    % Calculate time for for loop
    tic
    % Preallocate for loop vector
    b_for = zeros(1,N(m));
    % Calculate each entry in b by looping through all of vector a
    for n = 1:N(m)
        b_for(n) = a(n)^2;
    end
    t_for(m) = toc;

    % Calculate time for vectorization
    tic
    % Calculate all entries in b by element-wise exponentiation
    b_vec = a.^2;
    t_vec(m) = toc;

    % Clear b vectors for next iteration
```

```
    clear b_for b_vec
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot call and graphical commands
plot(N,t_for,'ko-',N,t_vec,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
set(gca, 'XScale', 'log')
set(gca, 'YScale', 'log')
set(gca,'XMinorTick','on')
xlim([10^0,10^8]);
ylim([10^-6,10^2]);
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('for loop','Vectorization','Location','northwest')
a = 20;
% The output of b is first established as NaN (nothing special about
 this
% choice)
b = NaN; % "not a number"

if a >10
    %b would be displayed if the condition is met
    b = 25;
end


disp(b)
% Declare vector and variable to store a value
v = [1, 5, 3];
res = 5;

% Overwrite a variable if the condition is true
if sum(v)>10
    res = 25;
end

disp(res)
var1 = 40;
var2 = 12;

if 10 <var1<30
    var2 = 2*var2;
end

disp(var2)
p = 6;
q = 15;
res = NaN;

% If condition one is true, "enter" the first if statement

if p>5
    if q ==15
        res = p*q
```

```
    end
end
disp(res)
var1 = 1;
var2 = 2;

% "If var1 equals var2, then assign "true" to out. Otherwise, assign
% "false" to out.

if var1 == var2
    out = true;
else
    out = false;
end
disp(out)
out = var1==var2;
disp(out)
% Classic problem: if my cereal is bad, the solution is clearly to add
% chocolate to it.
cerealGrade = 95;

if cerealGrade < 60
    chocolateAdded = 20;
elseif cerealGrade < 70
    chocolateAdded = 15;
elseif cerealGrade < 80
    chocolateAdded = 10;
elseif cerealGrade  <90
    chocolateAdded = 5;
else
    chocolateAdded = 0;
end
disp([cerealGrade,chocolateAdded])
cerealGrade = 95;
chocolateAdded = 0;
%reduces the amoung of code becasue the end statement does not need to
 be
%included

if cerealGrade < 60
    chocolateAdded = 20;
elseif cerealGrade < 70
    chocolateAdded = 15;
elseif cerealGrade < 80
    chocolateAdded = 10;
elseif cerealGrade  <90
    chocolateAdded = 5;
end
disp([cerealGrade,chocolateAdded])
% Examples using And
res1 = (1<2) && (4==5)
Alpha = (1<2) && (4<5)
% Examples using Or
var_2 = true || false
```

```
ind = (1<2) || (4==5)
% Examples using xOr
ip = xor(1<2,4<5)
a = [1,0,0,1]
b = [1,1,0,0]
a & b



A = true;
B = false;
C = false;

out
p = 6;
q = 15;
res = NaN;

if p > 5
    if q == 15
        res = p*q;
    end
end
disp(res)
if p>5 && q==15
    res = p*q
end
disp(res)
% Declare initial variable
res = false;

a = 15;
b = 20;
c = 30;

% If this condition is true, change the value of res
if a<20 && b==20 || c>20
    res = true;
end
disp(res)
% Assign initial values to variables variables
a = 15;
b = 17;

% As long as AT LEAST ONE of the current values of a or b is greater
 than
% 5, the loop will continue. That is, the loop will stop when BOTH a
 and b
% are less than 5.
while a(end)>5 || b(end)>5
    %generates random numbers from 1 to 20 and adds them to the
 vector.

    a(end+1) = randi([1 20]);
```

```
    b(end+1) = randi([1 20]);
end


disp([a',b'])
% Using comments to explain our code, we can perform basics tasks,
 look up
% information about functions, etc.
1 + 2
% We have learned how to use the resources in MATLAB to perform basic
% calculations
5*tan(pi/4)
a = 15; % Assigning a scalar variable
b = 13+2/a;  % Creating a statement based on the previous variable
disp(b) % Display the newly defined variable
% Create vectors
a = 1:5;
b = linspace(1,4,5);
c = zeros(3,3);
c(1:9) = randi(5,[1,9]);
disp(a)
disp(b)
disp(c)
% Access elements in the vectors and matrix
a([1:2,end])
b(randi(5,[1,10]))
c([1,3],[1,3])
% Perform array math
a+b     % vector addition
a.*b    % element-wise multiplication
(a')*b  % matrix multiplication
a(1:3)+c(1,:)   % Adding subvectors derived from original array
a = zeros(1,10);    % Preallocate vector to store values
% Perform 10 iterations, each iteration calculate square root of
 different
% random number
for k = 1:10
    a(k) = sqrt(randi(k));
end
disp(a)
% Create a vector containing all multiples of 13 less than 300;
a = 13; % vector to store integer multiples of 13. Start by populating
 first (obvious) case
num = 1;    % counter to determine next multiple
while a(end) < 300
    num = num + 1;
    a(end+1) = num*13;
end
a(end) = [];    % Delete the last entry because it will be over 200;
disp(a)
% Generate two random numbers
a = randi(5);
b = randi(5);
```

```
if a==b % if variables are equal
    disp(a*b*ones(1,a*b))
elseif a<b % if b is greater than a
    disp(b*ones(1,a))
else    % if a is greater than b
    disp(a*ones(1,b))
end
a = [5,2,1];
b = zeros(3,1);

% Goal: produce a plot
plot(a,b)
% Goal: create  matrix through matrix multiplication
a = 1:5;
b = ones(5,1);
a*b
% % Goal: calculate the terminal velocity of a falling object
 (positive is
% % down)
g = 9.81; % m/s^2
d = 0.001; % diameter, m
mu = 10^-3; % Pa*s
pm = 1000; % kg/m^3
ps = 1050; % kg/m^3

Vt = g*d^2/(12*mu)*(ps - pm)
%step 1: specify angle


%step 2: constants

% Repeat for all angles
%step 3: calculate distance for each theta

%step 4: calculate maximum height

% %step 1: specify angle
% th = 45; %in degrees
% th_r = th*pi/180;%radians
%
% %step 2: constants
% g = 9.81; %m/s^2
% h = 2; %height
% v0 = 30; %initial velocity
%
% %step 3: calculate distance for theta
% t_end = (-v0*sin(th_r) - sqrt(v0^2*sin(th_r)^2 + 2*g*h))/(-g);
% x_end = v0 *cos(th_r)*t_end
%
% %step 4: caculate max height
% t_peak = v0*sin(th_r)/g;
% y_peak = h + v0*sin(th_r)*t_peak - g/2*t_peak

%step 1: specify angle
```

```
th = 45; %in degrees
th_r = th*pi/180;%radians

%preallocationg
x_end = zeros(1,length(th));
y_peak = zeros(1,length(th));

%step 2: constants
g = 9.81; %m/s^2
h = 2; %height
v0 = 30; %initial velocity

for k = 1:length(th)
    %step 3: calculate distance for theta
    t_end = (-v0*sin(th_r(k)) - sqrt(v0^2*sin(th_r(k))^2 + 2*g*h))/(-
g);
    x_end(k) = v0 *cos(th_r(k))*t_end;

    %step 4: caculate max height
    t_peak = v0*sin(th_r(k))/g;
    y_peak(k) = h + v0*sin(th_r(k))*t_peak - g/2*t_peak^2;
end
plot(th,x_end,'k.')
% %step 1: specify angle
th = 45; %in degrees
th_r = th*pi/180;%radians

% %step 2: constants
g = 9.81; %m/s^2
h = 2; %height
v0 = 30; %initial velocity
%
% %step 3: calculate distance for theta
t_end = (-v0*sin(th_r) - sqrt(v0^2*sin(th_r)^2 + 2*g*h))/(-g);
x_end = v0 *cos(th_r)*t_end

% %step 4: caculate max height
t_peak = v0*sin(th_r)/g;
y_peak = h + v0*sin(th_r)*t_peak - g/2*t_peak;
plot(th,x_end,'k-')
% Step 1: specify 1 angle%
% Step 2: constants% Repeat for all angles%
% Step 3: calculate distance for each theta%
% Step 4: calculate maximum height
%Step 1: specify angle
th = 0:0.005:90; %degrees
th_r = th*pi/180; %radians

%Preallocate vector
x_end = zeros(1,length(th));
y_peak = zeros(1,length(th));

%Step 2: constants
g = 9.81; %m/s^2
```

```
h = 2; %m
v0 = 30; %m/s

%repeat for each theta
for k = length(th)
    %repeat for all angles
    %step 3: calculate distance for each theta

    t_end = (-v0*sin(th_r(k)) - sqrt(v0^2*sin(th_r(k))^2 + 2*g*h))/(-
g);
    x_end(k) = v0*cos(th_r(k))*t_end;

    %Step 4L calculate maximum height
    t_peak = v0*sin(th_r(k))/g;
    y_peak(k) = h + v0*sin(th_r(k))*t_peak - g/2*t_peak^2;
end

%plot the distance
plot(th,x_end,'k.')

% Plot the height of the ball
plot(th,y_peak,'k.')




%Step 1: specify angle
th = 0:0.005:90; %degrees
th_r = th*pi/180; %radians

%Preallocate vector
x_end = zeros(1,length(th));
y_peak = zeros(1,length(th));

%Step 2: constants
g = 9.81; %m/s^2
h = 2; %m
v0 = 30; %m/s

% Step 3: Calculate distance thrown - element-wise
t_end = (-v0*sin(th_r) - sqrt(v0^2*sin(th_r).^2 + 2*g*h))/(-g);
x_end = v0*cos(th_r).*t_end;

% Step 4: calculate maximun height
t_peak = v0*sin(th_r)/g;
y_peak = h + v0*sin(th_r).*t_peak - g/2*t_peak.^2;

%plot the distance
plot(th,x_end,'k.')
xlabel('Angle thrown (degrees)')
ylabel('Distance thrown (m)')


% Plot the height of the ball
```

```
plot(th,y_peak,'k.')
xlabel('Angle thrown (degrees)')
ylabel('Heigth thrown (m)')

% Step 1: specify initial guess

% Step 2 L Iterate until xn -> xn+1;

% Step 3L xn + 1 from xn

% Step 4: compare xn and xn+1 to determine if loop should continue

%The final form of the practice problem from class

% Preallocate storage vectors
numTurns = zeros(1,1000);      % Store requred number of turns from k-
th game
diceFaceValue = zeros(1,1000); % Store final dice face value from the
 k-th game
for k = 1:1000
    [numTurns(k), diceFaceValue(k)] = diceRollSimulator(3,6);
end
histogram(diceFaceValue)
ylabel('Number of games')
xlabel('Final dice face value')
title('Result from 1000 simulations: histogram of final dice values
 that end the game')
% Plot 2 - plot distributuion of number of turns
histogram(numTurns)
ylabel('number of games')
xlabel('Number of turns required')
title('Result from 1000 simulations: histogram of final dice values
 that end the game')

%work while a conditional statement is true
n = 0;
a = 0;
while n<5
    %keep track of when n is odd to perform tasks
    if mod(n,2) == 1
        a = a+1;
    end
    n = n+1;

end

% Step 1: specify initial guess

% Step 2: iterate until xn -> xn+1

% Step 3: xn + 1 from xn

% Step 4: compare xn and xn+1 to dtermine if loop should continue
```

```
% "Rough Solution:" use a for loop to solve this problem

% Step 1: specify initial guess
xn = 3;

% Step 2: interate until xn -> xn+1
for k = 1:5
    %Step 3: evaluate function
    fn = xn^4 + 2*(xn-2)^2 - 20;

    % Step 3a: evaluate derivative
    f_prime_n = 4*xn^3 + 4*(xn-2);

    % Step 3b: calculate next x value
    xnplus1 = xn -fn/f_prime_n;

    % Step 4: update xn
    xn = xnplus1;
end
disp(xn)
% Improved solution: use a while loop to iterate until convergence
% We will start by revisiting on Monday. The first is the approach we
 were
% heading towards in class before we ran out of time: using an if
 statement
% inside the while loop.

% Step 1: specify initial guess and other variables
xn = 3; % initial guess
keepIterating = 1; % Variable that specifies if we should keep
 iterating.
% We will set the value to zero if we meet some stopping criteria.

% Step 2: iterate until xn -> xn+1
while keepIterating

    % Step 3: evaluate function
    fn = xn^4 + 2*(xn-2)^2 - 20;
    %Step 3a: evaluate derivative
    f_prime_n = 4*xn^3 + 4*(xn-2);

    %Step 3b: calculate the next value of xn
    xnplus1 = xn - fn/f_prime_n;

    % Step 4: Evaluate if we have reached a stopping criteria. Note
 the
    % form of this relational expression
    if abs(xn - xnplus1) < 10^-5
        % One approach: use an if statement
        keepIterating = 0;
    end
```

```
    % Step 5: update xn and repeat
    xn = xnplus1;
end
disp(xn)

% Define initial vector
v = [5 3 4 6 8]
% Define subsequent vectors through vector indexing
p = v([1 5 4 2])
q = v(1:2)
r = v(end-2:end)
s = v(end:-1:end-3)
t = v(1:2:end)
% Define initial matrix
M = [5 2 4; 3 7 5]
% Create subsequent matrices through indexing
B = M(1:2,1:2)
C = M(:,1)
D = M([1,2],[2,3])
F = M(1,end-1:end)
G = M(end:-2:1)
% Perform array math
B + D     % matrix addition
B.*D    % element-wise multiplication
B*M  % matrix multiplication
A = [pi/4,3*pi/4;5*pi/4,7*pi/4]
% Tangent operation on entire array
tan(A)
a = 1:3
sum(a)
% Create matrix
A = [1:3;4:6]
% Sum across each row of A
sum(A,1)
% Sum across each column of A
sum(A,2)
sum(sum(A))
sum(A(:))
a = 1:4
%adds to the next index as it goes through
cumsum(a)
A
cumsum(A,2)
% Vector example
a
prod(a)
% Matrix example
A
prod(A,2)
% Vector example
a
cumprod(a)
% Array example
A
```

```
cumprod(A,2)
% Create an array with all zeros
zeros(3,2)
% Create a matrix with all ones
ones(4)
% Create a matrix with equal row, column, and diagonal sums
magic(3)
% Create an identity matrix
eye(3)
% Create a matrix of random integers
randi(4,[2,3])
% Relational operator examples
an_1 = 5==2
c = 5==2
% Relational operators can also operate on variables
dog = 4;
cat = 9;
animal = dog==cat
% Define arrays
M = [1,5,8;9,4,3]
A = [7,2,6;3,0,5]
% Creating logical arrays by performing relational operations
V = M == 5
% Logical matrix based on A, specify which elements are greater than 6
r3 = A > 6
% Create logical matrix indicating which elements in M are greater
 than the
% corresponding elements in A
out = M>A
v = [6,4,0,1]
g2 = logical(v)
%choose x or o's to be 1 or zeros and then gerate random matrices
%in order to check how you win you need to check if all indices in a
%row/column/diagonal equal each other


%step 1: define initial variables
gameboard = zeros(3);
%create array to keep track of positions
availablePos = 1:9;

%Repeat: alternate between players
player = 1; %player 1
%checks who's turn it is
gameStillGoing = 1;
%iterate until someone has one, or game is a draw
availablePos
while gameStillGoing
    %Step 2: make a move, play position
    %populate using single index notation

    %logical corresponding to valid move

    invalidMove = 1;
```

```
    while invalidMove
        playPos = input("Select a position to play...")
        invalidMove = prod(availablePos ~= playPos)
    end
    %Update gameboard
    gameboard(playPos) = player;
    %Step 3: display gameboard
    disp(gameboard)

    %Step 4: evaluate stopping condition
    gameStillGoing = checkEndGame(gameboard, player);
    %step 5 change player
    player = mod(player,2) + 1;

end
% Define arrays
M = [1,5,8;9,4,3]
A = [7,2,6;3,0,5]
% Creating logical arrays by performing relational operations
v = M == 5
% Logical matrix based on A, specify which elements are greater than 6
r3 = A > 6
% Define vector
v = [5,2,4]
% Define logical vector
a = logical([1 0 1])
res = v(a) %since first and third entries are true you return those
res = v([1,3]) %enables us to select elemants that satisfy a condition
% Define two arrays
v = [6,5,2]
p = [9,5,3]
% Perform logical indexing
r = v(p==3)
s = v(v>3)
% Create new vector consisting only of entries of p less than 4
val2 = p(p<=4)
% Overwrite all entries whose value is greater than 3
v(v>3) = 7
% Delete entries from v based on logical indexing
v(p<6) = []
% Create example matrices
M = [3,7,6;5,4,2]
A = [1,8,0;5,9,2]
res = M(A>2)
%T = M(A)
out = M(logical(A))
% Two more examples
G = A(M==A)
vals = A(M==7 | A==9)
% Redefine matrices
M = [3,7,6;5,4,2]
A = [1,8,0;5,9,2]
% Set all elements less than 4 to be 9.
```

```
M(M<4) = 99
% Specify element positions in M corresponding to those in which
 either A <
% 2 or A > 7 to equal 5
M(A<2 | A>7) = 5
% Delete elements from A that are greater than 5
A(A>5) = []
% Define vectors
v = [6,5,2]
p = [9,5,3]
% Create a vector with nonzero elements defined by the relational
 operator
res = (v>5).*p
% Note the importance of precedence!
res2 = v>5.*p
% time vector
t = -20:.1:20;
% Produce a plot of the causal signal y(t)
plot(t,sin(t).*(t>=0),'k-')
% Define plotting parameters
xlabel('t')
ylabel('y(t)')
axis square % The plot window becomes square
ylim([-1,1]) % Set the y limits
% time vector
t = -20:.1:20;
% Produce a plot of the causal signal y(t)
plot(t,sin(t).*(sin(t)>0),'k-')
% Define plotting parameters
xlabel('t')
ylabel('y(t)')
axis square % The plot window becomes square
ylim([-1,1]) % Set the y limits
x = (1:4)>3
% TRUE if any element is true
any(x)
x = (1:4)>3
% TRUE if all elements are true
all(x)
x = 1:4
% TRUE if any element is true
isscalar(x)
x = 1:4
% TRUE if any element is true
isempty(x)
x = nan(1,3)
% Element-wise evaluation -> a corresponding index is 1 if that entry
 is
% NaN
isnan(x)
% Create matrix
z = magic(3)
z'
% Display matrix
```

```
z
% Rotate matrix 90 degrees counter clockwise
rot90(z)
% Flip matrix "up and down," that is the last row becomes the first
flipud(z)
% This is equivalent to reversing the row order, but requires less
 code
z(end:-1:1,:)
% Flip matrix "left and right," that is the last column becomes the
 first
fliplr(z)
% This is equivalent to reversing the column order, but requires less
 code
z(:,end:-1:1)
% Flip the matrix along the specified dimension. This could work for
 higher
% order arrays
% 1: flip rows
% 2: flip columns
flip(z,1)
z
z2 = reshape(z,1,9)
% Create a test arrays
a = 1:7
% Use logical indexing to select all elements greater than 3
a(a>3)
% Use array math to set all elements that are NOT greater than 3 to 0.
a.*(a>3)
%b is a logical vector so only has 0 and 1
b = a > 1
% Evaluate if at least one entry is true
any(b)
% Evaluate if all entries are true
all(b)
% Evaluate if no entries are true
all(-b)
% Reshape array of elements larger than 3 into a 2x2 array.
b = reshape(a(a>3),2,2)
% Rotate matrix 90 degrees ccw
rot90(b)
% Reverse the rows
flipud(b)
% Reverse the columns
fliplr(b)
% Measured time points
t = 1:1000;
% Data points where each entry is the data point measured at the
% corresponding entry in the time vector.
dataPts = rand(1,length(t));

% Plot data
%plot for times less that 500
plot(t(t<500),dataPts(t<500),'k.')
axis square
```

```
ylim([0,1])          % y limit
xlim([0,length(t)]) % x limit
xlabel('time')       % data label for x axis
ylabel('data value')    % data label for y axis
% Plot data at t < 500
plot(t(t<500),dataPts(t<500),'k.')

axis square
ylim([0,1])          % y limit
xlim([0,length(t)]) % x limit
xlabel('time')       % data label for x axis
ylabel('data value')    % data label for y axis
% Plot data with values greater than 0.5
plot(t(dataPts>0.5),dataPts(dataPts>0.5),'k.')

axis square
ylim([0,1])          % y limit
xlim([0,length(t)]) % x limit
xlabel('time')       % data label for x axis
ylabel('data value')    % data label for y axis
% Plot data with values greater than 0.7 set to 0

plot(t,dataPts.*(dataPts<=0.7),'k.')
axis square
ylim([-0.1,1])          % y limit
xlim([0,length(t)]) % x limit
xlabel('time')       % data label for x axis
ylabel('data value')    % data label for y axis
% Create the time vector
t = -5:0.1:5;
% Create the signal vector
x = t.*(t>=0).*(t<1) + (2-t).*(t>=1).*(t<2);

% Plot
plot(t,x,'r-')
axis square
ylim([-5,5])
xlim([-5,5])
grid on   % Display the grid
xticks(-5:5) % Explicitly define the x tick marks
yticks(-5:5)
% Create vector, each entry corresponds to a different seller's packs
 sold
packsSold = [57, 299, 161, 429];
bonus = zeros(1,4);
% Calculate bonuses.
for jj = 1:length(packsSold)

    % If >=100 and <200 packs sold
    if packsSold(jj)>=100 && packsSold(jj)<200
        bonus(jj) = .02*packsSold(jj);

    % If >=200 and <300 packs sold
    elseif packsSold(jj)>=200 && packsSold(jj)<300
```

```
        bonus(jj) = .04*packsSold(jj);

    % If >300 packs sold
    elseif packsSold(jj)>=300
        bonus(jj) = .06*packsSold(jj);
    end
end
disp(bonus)
% Calculate bonus using logical indexing
%make a logic based function that only multiplies agains thtings that
 were
%true
bonus2 = ...
    0.02*packsSold.*(packsSold>= 100 & packsSold<200) + ...
    0.04*packsSold.*(packsSold>= 200 & packsSold<300) + ...
    0.06*packsSold.*(packsSold>= 300);
disp(bonus2)
%define verctor containing employee hours
employeeHours = [45, 21, 19, 61, 39];
%calculate pay in one statement
eployeePay = 10*employeeHours +
 19*0.5*(employeeHours-40).*(employeeHours>40)
%at the end there is a logical that only multiplies if there is more
 than
%40 hours
```

```
% Vector containing different numbers of barkers. We will calculate
 the
% time to calculate the bonus for increasing N using if-else vs.
 logical arrays
N = round(logspace(1,6,100));

% Preallocate vectors to store computational time
t_ifElse = zeros(1,length(N));
t_logicalMath = zeros(1,length(N));

% Calculate computational time for vector of different numbers of
 barkers
for k = 1:length(N)

    % Generate vector containing packs sold. Each entry corresponds to
 the
    % packs sold by a different employee.
    packsSold = randi(500,[1,N(k)]);

    % Calculate time using conditional statement
    tic;
    % Preallocate bonus array
    bonus1 = zeros(1,length(packsSold));
```

```
    % Calculate bonus received for each employee
    for jj = 1:length(packsSold)

        % If >=100 and <200 packs sold
        if packsSold(jj)>=100 && packsSold(jj)<200
            bonus(jj) = .02*packsSold(jj);

        % If >=200 and <300 packs sold
        elseif packsSold(jj)>=200 && packsSold(jj)<300
            bonus(jj) = .04*packsSold(jj);

        % If >300 packs sold
        elseif packsSold(jj)>=300
            bonus(jj) = .06*packsSold(jj);
        end
    end
    t_ifElse(k) = toc;


    % Calculate time using logical array operations and math
    tic;
    % Calculate bonus in one statement
    bonus2 = ...
        0.02*packsSold.*(packsSold>=100 & packsSold<200) + ...
        0.04*packsSold.*(packsSold>=200 & packsSold<300) + ...
        0.06*packsSold.*(packsSold>=300);
    t_logicalMath(k) = toc;
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot the functions
plot(N,t_ifElse,'ko-',N,t_logicalMath,'ro-','LineWidth',1.5,'MarkerFaceColor','aut
% Graphical parameters
set(gca, 'XScale', 'log')   % "log" axes - x
set(gca, 'YScale', 'log')   % "log" axes - y
set(gca,'XMinorTick','on')
xlim([10^1,10^6]);  % range of x values on plot
ylim([10^-6,10^0]); % range of y values on plot
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('If-else','Array math','Location','northwest')
% Determine range of N
N = logspace(0,3,4);

% Preallocate vectors to store computational time
t_no = zeros(length(N),1);
t_yes = zeros(length(N),1);

% For each number of elements considered
for k = 1:length(N)

    % Calculate time without preallocation
    tic;
    % For each row
```

```
    for jj = 1:N(k)
        % For each column
        for mm = 1:N(k)
            % Each entry is the produce of the rows and columns of the
            % multiplication table
            multTable(jj,mm) = jj*mm;
        end
    end
    t_no(k) = toc;

    % Calculate time without preallocation
    tic;
    % Preallocate array
    multTable = zeros(N(k),N(k));

    % For each row
    for jj = 1:N(k)
        % For each column
        for mm = 1:N(k)
            % Each entry is the produce of the rows and columns of the
            % multiplication table
            multTable(jj,mm) = jj*mm;
        end
    end
    t_yes(k) = toc;

    % Delete multiplication table so that it can be created from
 scratch on
    % the next loop
    clear multTable
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot the functions
plot(N,t_no,'ko-',N,t_yes,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
% Graphical parameters
set(gca, 'XScale', 'log')   % "log" axes - x
set(gca, 'YScale', 'log')   % "log" axes - y
set(gca,'XMinorTick','on')
xlim([10^0,10^4]);  % range of x values on plot
ylim([10^-6,10^0]); % range of y values on plot
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('No preallocation','Preallocation','Location','northwest')
multTable = (1:10)'*(1:10)
A = zeros(1000,2000);
A(3,4) = 15;
A(100,1500) = 5;
A(1000,2000) = 9;
B = sparse([3,100,1000],...
    [4,1500,2000],...
    [15,5,9],1000,2000);
B = sparse(A);
whos A B
```

```
% Define a matrix intially with all zeros
A = zeros(100,100);

% Preallocate vector to store memory
mem_Full = zeros(1,numel(A));
mem_Sparse = zeros(1,numel(A));

% Repeat for all elements
for k = 1:numel(A)
    A(k) = randi(5);

    A_sparse = sparse(A);

    matrixInfo = whos('A','A_sparse');
    mem_Full(k) = matrixInfo(1).bytes;
    mem_Sparse(k) = matrixInfo(2).bytes;
end
% Plot the functions
plot(1:numel(A),mem_Full/100,'k-',1:numel(A),mem_Sparse/100,'r-')

% Graphical parameters
axis square
xlim([0,10^4]);  % range of x values on plot
ylim([0,2*10^3]); % range of y values on plot
xlabel('N - number of nonzero matrix elements');
ylabel('Memory required (kilobytes)');
legend('Full matrix','Sparse matrix','Location','northwest')
% Create a test matrix of containing simulated patient information
N = 10; % Number of patients

% Input matrix containing patient information
patient_rawData = [(1:N)',randi(50,[N,2])+100];
disp(patient_rawData)
% Step 1: Define output arrays - we will populate these as we
 sequentially
% go through each for loop.

patient_DrugA = [];
patient_DrugB = [];


% Step 2: For each patient, compare their blood pressure at time t1 to
% their average blood pressure to determine if they should receive
 meds

% Use a FOR loop because we know how many iterations to perform -> one
 for
% each row of patient_rawData because each row in patient_rawData
% corresponds to one patient
for k = 1:size(patient_rawData, 1)

    % NOTE: the columns contain the following information:
    % patient_rawData(k,1): four-digit patient ID
```

```
   % patient_rawData(k,2): P_t1: blood pressures of patients at time
 t1
   % patient_rawData(k,3): P_avg: average recent patient blood
 pressures

   % if P_t1 - P_avg >= 10, determine if a dose of 10 or 30 should be
   % applied
   if patient_rawData(k,2) - patient_rawData(k,3) >= 10

   % if P_t1 - P_avg >= 20 -> assign to patient_DrugA and presribe a
 dose
   % of 30
       if patient_rawData(k,2) - patient_rawData(k,3) >= 10

           % Add a row to patient_DrugA, with the first entry being
 the
           % patient ID and the second being a dose of 30
           patient_DrugA(end+1,1:2) = [patient_rawData(k,1), 30];


       % if P_t1 - P_avg >= 10 -> assign to patient_DrugA and
 presribe a dose
       % of 10
       else
           % Add a row to patient_DrugA, with the first entry being
 the
           % patient ID and the second being a dose of 10
           patient_DrugA(end+1,1:2) = [patient_rawData(k,1), 10];
       end


   % if P_t1 - P_avg < 10 -> assign to patient_DrugB and presribe a
 dose
   % of 20
   elseif patient_rawData(k,2) - patient_rawData(k,3) <= -10

       % Add a row to patient_DrugA, with the first entry being the
       % patient ID and the second being a dose of 10
       patient_DrugB(end+1, 1:2) = [patient_rawData(k,1), 20];

   end
end

disp(patient_DrugA)
disp(patient_DrugB)

% Step 1: Define output arrays - we will populate these as we
 sequentially
% go through each for loop.

patient_DrugA = zeros(size(patient_rawData,1),2);
patient_DrugB = zeros(size(patient_rawData,1),2);;
```

```
%create counters to keep track or rows for the patients
rowA = 1;
rowB = 1;

% Step 2: For each patient, compare their blood pressure at time t1 to
% their average blood pressure to determine if they should receive
 meds

% Use a FOR loop because we know how many iterations to perform -> one
 for
% each row of patient_rawData because each row in patient_rawData
% corresponds to one patient
for k = 1:size(patient_rawData, 1)

    % NOTE: the columns contain the following information:
    % patient_rawData(k,1): four-digit patient ID
    % patient_rawData(k,2): P_t1: blood pressures of patients at time
 t1
    % patient_rawData(k,3): P_avg: average recent patient blood
 pressures

    % if P_t1 - P_avg >= 10, determine if a dose of 10 or 30 should be
    % applied
    if patient_rawData(k,2) - patient_rawData(k,3) >= 10

    % if P_t1 - P_avg >= 20 -> assign to patient_DrugA and presribe a
 dose
    % of 30
        if patient_rawData(k,2) - patient_rawData(k,3) >= 10

            % Add a row to patient_DrugA, with the first entry being
 the
            % patient ID and the second being a dose of 30
            patient_DrugA(end+1,1:2) = [patient_rawData(k,1), 30];


        % if P_t1 - P_avg >= 10 -> assign to patient_DrugA and
 presribe a dose
        % of 10
        else
            % Add a row to patient_DrugA, with the first entry being
 the
            % patient ID and the second being a dose of 10
            patient_DrugA(end+1,1:2) = [patient_rawData(k,1), 10];
        end


    % if P_t1 - P_avg < 10 -> assign to patient_DrugB and presribe a
 dose
    % of 20
    elseif patient_rawData(k,2) - patient_rawData(k,3) <= -10

        % Add a row to patient_DrugA, with the first entry being the
        % patient ID and the second being a dose of 10
```

```
        patient_DrugB(end+1, 1:2) = [patient_rawData(k,1), 20];

    end
end

disp(patient_DrugA)
disp(patient_DrugB)



% Create matrix for patients recieiveing drug A

%create submatrix consisting only of patients that need Drug A
patient_DrugA = patient_rawData(patient_rawData(:,2) -
 patient_rawData(:,3) >= 10,:);

%Use logival array indexing to determine the dose of drugs each
 patient
%should recive. Create a new column to clculate doses based on the
%patient's blood pressure and avg blood pressure

patient_DrugA(:,4) = 10 + 20*(patient_DrugA(:,2) - patient_DrugA(:,3)
 >= 20);

%Delete the unnecessary columns -> we do not want columns
 corresponding to patient blood pressure or the avg blood pressure.
patient_DrugA(:,2:3) = [];


%Step 2: Create matrix for patients recieving drub B

%Create a column vector consisting of only patients that will need
 drub B
patient_DrugB = patient_rawData(patient_rawData(:,2) -
 patient_rawData(:,3) <= 10,1);

%use logival array indexing to deremine the dose of drugs each person
%should receive. Create a new column containing the dose 20 -- all
patient_DrubB(:,2) = 20;

disp(patient_DrugA)
disp(patient_DrugB)
% Determine range of N
N = logspace(0,3,4);

% Preallocate vectors to store computational time
t_no = zeros(length(N),1);
t_yes = zeros(length(N),1);

% For each number of elements considered
for k = 1:length(N)

    % Calculate time without preallocation
    tic;
```

```
    % For each row
    for jj = 1:N(k)
        % For each column
        for mm = 1:N(k)
            % Each entry is the produce of the rows and columns of the
            % multiplication table
            multTable(jj,mm) = jj*mm;
        end
    end
    t_no(k) = toc;

    % Calculate time without preallocation
    tic;
    % Preallocate array
    multTable = zeros(N(k),N(k));

    % For each row
    for jj = 1:N(k)
        % For each column
        for mm = 1:N(k)
            % Each entry is the produce of the rows and columns of the
            % multiplication table
            multTable(jj,mm) = jj*mm;
        end
    end
    t_yes(k) = toc;

    % Delete multiplication table so that it can be created from
 scratch on
    % the next loop
    clear multTable
end
% Define dimensions of plot
figure('Position',[0,0,500,300]);
% Plot the functions
plot(N,t_no,'ko-',N,t_yes,'ro-','LineWidth',1.5,'MarkerFaceColor','auto')
% Graphical parameters
set(gca, 'XScale', 'log')   % "log" axes - x
set(gca, 'YScale', 'log')   % "log" axes - y
set(gca,'XMinorTick','on')
xlim([10^0,10^4]);  % range of x values on plot
ylim([10^-6,10^0]); % range of y values on plot
xlabel('N - number of vector elements');
ylabel('Compuatational time (s)');
legend('No preallocation','Preallocation','Location','northwest')
multTable = (1:10)'*(1:10)
A = zeros(1000,2000);
A(3,4) = 15;
A(100,1500) = 5;
A(1000,2000) = 9;
B = sparse([3,100,1000],...
    [4,1500,2000],...
    [15,5,9],1000,2000);
B = sparse(A);
```

```
whos A B
% Define a matrix intially with all zeros
A = zeros(100,100);

% Preallocate vector to store memory
mem_Full = zeros(1,numel(A));
mem_Sparse = zeros(1,numel(A));

% Repeat for all elements
for k = 1:numel(A)
    A(k) = randi(5);

    A_sparse = sparse(A);

    matrixInfo = whos('A','A_sparse');
    mem_Full(k) = matrixInfo(1).bytes;
    mem_Sparse(k) = matrixInfo(2).bytes;
end
% Plot the functions
plot(1:numel(A),mem_Full/100,'k-',1:numel(A),mem_Sparse/100,'r-')

% Graphical parameters
axis square
xlim([0,10^4]);  % range of x values on plot
ylim([0,2*10^3]); % range of y values on plot
xlabel('N - number of nonzero matrix elements');
ylabel('Memory required (kilobytes)');
legend('Full matrix','Sparse matrix','Location','northwest')
A = 1:4
B = A > 2
% Sum over all rows
sum(((A>2) | A == 1),1)
val1 = 1;
val2 = 2;

% Call function
x = square_add1(val1,val2)
% Create a function handle
f1 = @square_add1;
% Use the function handle. Note the scalar inputs are passed to the
% function in the same way as before.
y = f1(1,2)
% Create function handle for new function
f2 = @operateOnKnownValues
% Input square_add1 into new function
z = f2(@sum)
% Create a vecotr
A = 1:4
% Create function handle for sum() function
f1 = @sum;
f1(A)
% Function handle for example A
f2 = @exampleA
% Function handle for example B
```

```
f3 = @exampleB
% Call example function, input sum and A. The following two are
 equivalent
out1 = f3(@sum,A)
out2 = f2(f1,A)
% NOTE: an error will occur if you input a function and not a function
% handle
% out3 = f2(sum,A)
% Create function handle
a = @springForce1;
% Call function
myForce = a(4)
% Evaluate the total force when spring 1 and 2 are arranged in
 parallel
force = parallelSprings(@springForce1,@springForce1,5)
% Create function handle with product function
a = @prod;
a([3,2,1,2])
b = @mod;
% Create function handle with modulus function
b(3,2)

% Create a function handle
f1 = @sum;
% Define a vector
c = 1:4
% Call function that takes the handle and vector as inputs
fxn1(f1,c)
t = 0:0.1:5;       % Time vector
g = t + exp(-t)-4; % Function g(t)
plot(t,g,'k-')
axis square
xlabel('t')
ylabel('g(t)')
% Script for using the Newton-Raphson method to solve for f(x)

% Step 1: specify initial guess
xn = 5; % initial guess
xprev = inf; % Variable to store previous xn value -> ensure it is
           % not xn to enter while loop

% Step 2: iterate until xn -> xn+1
while abs(xn - xprev) > 10^-5

    % Step 3: evaluate function at xn
    fn = xn^4 + 2*(xn-2)^2 - 20;
    % Step 3a: evaluate derivative at xn
    f_prime_n = 4*xn^3 + 4*(xn-2);

    % Instead of calculating the "next" x value, store the previous xn
    % value
    xprev = xn;

    % Step 3b: update xn to next value for the iteration
```

```
    xn = xn - fn/f_prime_n;
end
disp(xn)
%Determine root of original f(x)
rootF = newtonSolve(@originalEqn, 5, 10^-5)
rootG = newtonSolve(@originalEqn, 5, 10^5)
newEqn(rootG)
% Anonymous function of a single input operating on a
% scalar.
myFunc = @(x) x^2 + 2*x + 1;
res1 = myFunc(1)
% Anonymous function of a single input operating on a
% vector.
myFunc = @(x) x.^2 + 2.*x + 1;
res2 = myFunc([1:4])
% Anonymous function of a multiple inputs operating on
% scalars.
%x and y are the inputs
myFunc = @(x,y) x^2 + 2*y +1;
res3 = myFunc(1,3)

 % Anonymous function of a multiple inputs operating on
% vectors.
myFunc = @(x,y) x.^2 + 2.*y +1;
res4 = myFunc([1,2,2,1],4:7);


mod2 = @(x) mod(x,2);
mod2(3)
% Plot x vector
x = 0:0.1:10;
% Define function handle
f = @(a) exp(-a*x)

% Plot function for each a
plot(x,f(0.5),'k-')
hold on     % plot on the same set of axis
plot(x,f(1),'r-')
hold on     % plot on the same set of axis
plot(x,f(5),'b-')
hold off    % turn "off" plotting on the same set of axis

axis square
legend('a = 0.5','a = 1', 'a = 5')
xlabel('x')
ylabel('f(x)')
% Call the recursive function
x = recursiveAdd(3)
% Max value of entries in Fibonacci sequence
n = 50;
% Define first two entries
x(1) = 0;
x(2) = 1;
% Iterate as long as the last element is less than 100.
```

```
while x(end) < n
    % If the last element is less that 100, add the next element of
 the
    % Fibonacci sequence to the vector
 x(end+1)=x(end)+x(end-1);
end
x(end)=[];  % Remove last element
disp(x)
% Calculate Fibonacci sequence vector containing elements less than 50
x = recursiveFib(0:1,50)
n = 50000;
% Call recursive function
x = fact(n);
fact_n = 1;
for m = 1:n
    fact_n = fact_n*m;
end
% Load the data we just collected
rawData = load('sensorData5.mat')
% Timetable containing our collected sensor data
rawData = rawData.Position
% Step 1: convert the Timestamp information into "seconds past start
 time"
% Convert date time vector into numbers representing the corresponding
 minutes
t = minute(rawData.Timestamp) + second(rawData.Timestamp)/60;
t = (t - t(1))*60;      % t is now a vector in which entries are in
 seconds

% Step 2: Convert positional information to relative positions
% Convert both to relative positions
lat_1 = (rawData.latitude - rawData.latitude(1));
long_1 = (rawData.longitude - rawData.longitude(1));

% Both are in units of "degrees" -> convert to units of meters
lat_2 = lat_1*364000*0.3048;
long_2 = long_1*288200*0.3048;

% Convert altitude to relative z position, in meters
alt = rawData.altitude - rawData.altitude(1);

% Step 2: Create data matrix
myData = [t,long_2,lat_2,alt,rawData.speed];
disp(myData)
% myData(:,1): time, seconds
% myData(:,2): longitude (x value), meters relative to start.
% Positive x: move east; negative x: move west
% myData(:,3): latitude (y value), meters relative to start
% Positive y: move north; negative y: move south
% myData(:,4): altitude (z value), meters relative to start
% myData(:,5): speed, meters/second
% Independent variable
t = 0:0.1:10;
% Function (dependent variable) is sin(t)
```

```
f = sin(t);
% Plot the function: adjust the "line specification" to change the
 output
plot(t,f,'k-')
% time
t1 = myData(:,1);
% z position
y1 = myData(:,4);
% Plot the function: adjust the "line specification" to change the
 output
plot(t1,y1,'ko-');
plot([0 20],[-5 5],'r--')
plot(myData(:,1),'ko','MarkerFaceColor','k')
x = -2*pi:pi/20:2*pi;
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'ro-',x,y2,'k-','LineWidth',4,'MarkerSize',10)
x = -2*pi:pi/20:2*pi;
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'r-','LineWidth', 5)
hold on
plot(x,y2,'b-','LineWidth', 5)
hold off
% x position vs. time
plot(myData(:,1), myData(:,2),'ro-');
hold on
% y position vs. time
plot(myData(:,1), myData(:,3),'bo-');
hold on
% z position vs. time
plot(myData(:,1),myData(:,4),'ko-','LineWidth',2)
hold off
% Independent variable
t = 0:.1:10;
% Function (dependent variable) is sin(t)
f = sin(t);
% Plot the function: adjust the "line specification" to change the
 output
plot(t,f,'ko','MarkerFaceColor','y','LineWidth',2)
% x vector
x = -2*pi:pi/20:2*pi;
% Two different y functions
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'Color',[1,37,110]./255,'LineWidth',3);
hold on
plot(x,y2,'Color',[200,0,26]./255,'LineWidth',3);
hold off
% x position vs. y position
plot(myData(:,2), myData(:,3),'ro-')
xlim([-100,100]) % set x limit
ylim([-100,100]) % set y limit
axis square %change shape of axis
```

```
legend('Our ENGR105', 'Location', 'NorthWest');
xlabel('x position relative to start (m)')
ylabel('y position relative to start (m)')
title('x/y movement of our ENGR105 path')
text(myData(1,2), myData(1,3), 'Start')
text(myData(end,2), myData(end,3), 'End')

% Display figure in its own window
set(gcf,'Visible','on');
figure
axes
% Create new plot of f(t) = exp(t)
t = 0:0.1:10;
f = exp(t);
plot(t,f,'k-')
get(gca)
% Change some parameters
set(gca,'YScale','log')
set(gca,'XMinorTic','on')
set(gca,'TickDir','out')
x = 0:pi/20:2*pi;
y = sin(x);
plot(x,y,'LineWidth',3)
% Increase the font size of the figure
set(gca,'FontSize',20)
% Example graphics handle use
x = 0:pi/100:2*pi;
y = sin(x);

myFigure = figure;
plot(x,y)

% Set the background of the figure
% to white
set(myFigure,'Color','w')

% Take the default plot position and
% reduce the width by 1/2
pos = get(myFigure,'Position');
pos(3) = 0.5*pos(3);
set(myFigure,'Position',pos);
% This is the "usual" way we have added legends
t = 0:0.1:5;
f1 = 2*t;
f2 = t.^2;
plot(t,f1,'r-')
legend('f1')
% Create graphics handles
h1 = plot(t,f1,'r-');
hold on
h2 = plot(t,f2,'k-');
hold off
```

```
% Create a legend by inputting. NOTE: this will change handle
 properties in
% h1 and h2!
h3 = legend([h1 h2],{'f1','f2'});
% Modify the the handle associated with each plot
plot(t,f1,'r-', 'DisplayName', 'f1');
hold on
plot(t,f2,'k-','DisplayName','f2');
hold off
legend
x = 0:pi/20:2*pi;
y = sin(x);
h1 = plot(x,y);
get(h1, 'LineWidth')
h2 = plot(x,y,'LineWidth',3);
get(h2, 'LineWidth')
% Create new plot of f(t) = exp(t)
t = 0:0.1:5;
f = exp(t);
plot(t,f,'k-')
% Set y scale to logarithmic
set(gca,'YScale','log')
% x and y values
x = 0:1:25;
y = exp(x);

% Plot: y axis will have logarithmic scale
p = semilogy(x,y);
% Change the line width
set(p,'LineWidth',2)
set(p,'LineWidth',2)
set(gca,'FontSize',12)
% Plot: y axis will have logarithmic scale
p = semilogx(x,y);
x = logspace(0,5,25);
y = x.^5;
p = loglog(x,y);

set(p,'LineWidth',2)
set(gca,'FontSize',20)
set(gca,'XTick',logspace(0,5,6))
% Data values
x = [2,5,10,15];
y = [3,7,15,40];

% Y error bars: positive and negative are same
ye = [1,5,10,10];

% Graphics handle representing error bar plot function
h = errorbar(x,y,ye,'k-');
set(h,'LineWidth',1)
% Graphics handle representing error bar plot function
h = errorbar(x,y,ye,'k.','Marker','none');
hold on
```

```
plot(x,y,'k-','LineWidth',2)
hold off
% Change the parameters *ONLY* of the error bars
set(h,'LineWidth',1)
% Data values
x = [2,5,10,15];
y = [3,7,15,40];

% Positive and negative x error bars
xePos = [1,3,2,3];
xeNeg = [1,3,4,1];

% Positive and negative y error bars
yePos = [1,8,2,20];
yeNeg = [2,6,4,10];

% Create x and y error bar plot
h = errorbar(x,y,yeNeg,yePos,xeNeg,xePos,'k-');
set(h,'LineWidth',1)
x = 0:pi/20:2*pi;
y = sin(x);
scatter(x,y)
theta = 0:0.01:2*pi;
rho = sin(2*theta).*cos(2*theta);
% Plot theta vs. rho
plot(theta,rho,'ko')
% Plot in polar space
polarplot(theta,rho)
data = randi(5,[1,1000]);
h = histogram(data);
% Change bin sizes
set(h,'BinEdges',[0,1.5,4.5,5.5])
histogram(myData(:,5));
axis square
xlabel('Speed (m/s)')
ylabel('Counts')
title('Distribution of movement speed')
t = myData(1:10:end,1);
z = myData(1:10:end,4);

h = bar(t,z);
ylim([-10,10])
set(h,'FaceColor','none')
axis square
x = 1:4;
y1 = randi(3,[1,4]);
y2 = randi(5,[1,4]);
% Note: dimensions of "Y" input
h = bar(x,[y1;y2]');
ylim([0,7])
h = bar(x,[y1;y2]','stacked');
ylim([0,7])
f = figure;
plot(rand(1,20))
```

```
close(f)
% X values
x = 0:pi/20:2*pi;

% Separate Y functions
y1 = sin(x-0);
y2 = sin(x-.5);
y3 = sin(x-1);
y4 = sin(x-1.5);
y5 = sin(x-2);
y6 = sin(x-2.5);

subplot(2,3,1)
plot(x,y1)
axis tight

subplot(2,3,2)
plot(x,y2)
axis tight

subplot(2,3,3)
plot(x,y3)
axis tight

subplot(2,3,4)
plot(x,y4)
axis tight

subplot(2,3,5)
plot(x,y5)
axis tight

subplot(2,3,6)
plot(x,y6)
axis tight
% X values
x = 0:pi/20:2*pi;

% For each loop iteration, fill in one of the subplots
for k = 0:5
    y = sin(x-0.5*k);

    subplot(2,3,k+1)
    plot(x,y)
    axis tight
end
t = 0:pi/100:2*pi;
x = sin(t);
y = cos(t);
z = t;
plot3(x,y,z,'LineWidth',4)
xlabel('x')
ylabel('y')
zlabel('z')
```

```
grid on
% Change the viewpoint
view(45,45)
plot3(myData(:,2),myData(:,3),myData(:,4),'LineWidth',2)
xlabel('x')
ylabel('y')
zlabel('z')
xlim([-100,100])
ylim([-100,100])
zlim([-10,10])
grid on
title('Our ENGR105 adventure in 3D')
text(myData(1,2),myData(1,3),myData(1,4),'Start')
text(myData(end,2),myData(end,3),myData(end,4),'End')
% Subplot 1: Travel in 3D
subplot(2,2,1)
plot3(myData(:,2), myData(:,3), myData(:,4), 'LineWidth', 2)
xlabel('x')
ylabel('y')
zlabel('z')
xlim([-100,100])
ylim([-100,100])
zlim([-10,10])
grid on
title('3D trajectory of movement')
text(myData(1,2), myData(1,3),myData(1,4),'Start')
text(myData(end,2), myData(end,3),myData(end,4),'End')

% Subplot 2: altitude as a function of time
subplot(2,2,2)
plot(myData(:,1), myData(:,4), 'ko-')
zlim([-10,10])
title('Height traversed')
xlabel('time (s)')
ylabel('height (m)')
axis square


% Subplot 3: x vs y position
subplot(2,2,3)
plot(myData(:,2), myData(:,3), 'ro-')
xlabel('x position')
ylabel('y position')
xlim([-100,100])
ylim([-100,100])
axis square
title('Lateral movement')
text(myData(1,2), myData(1,3), 'Start')
text(myData(end,2), myData(end,3), 'End')


% Subplot 4: histogram of speeds
subplot(2,2,4)
histogram(myData(:,5));
```

```
axis square
xlabel('Speed (m/s)')
ylabel('Counts')
title('Distribution of movement speed')

% Add a title to the entire figure
sgtitle('ENGR105 adventure summary')
% specify a range of x
x = 0:0.01:1;

%plot the lengendre polynomial
plot(x,legendreP(x,0),'k-', 'LineWidth', 2)
hold on
plot(x,legendreP(x,1),'r-', 'LineWidth', 1)
hold on
plot(x,legendreP(x,2),'k--', 'LineWidth', 2)
hold on
plot(x,legendreP(x,3),'r--', 'LineWidth', 1)
hold on
plot(x,legendreP(x,4),'b:', 'LineWidth', 2)
hold off
axis square
ylim([-1,1.5])
set(gca,'Color', [0.95, 0.95, 0.95])
xlim([0,1.3])
legend('P_0','P_1','P_2','P_3','P_4')
class(x)
class('LineWidth')
class(legendreP(x,2))
whos x
a = 2;
whos a
% Convert "numbers" to an integer
a = uint8(4)
b = uint16(20)
c = int32(200)
whos a b c
d = uint8(4000)
% Note the range for "signed" integers
f = int8(200)
data = [1.5,3,4,2.2];
output = zeros(1,length(data));

% Goal: element-by-element, we want to square each element of data
for k = 1:length(data)
    output(k) = data(k)^2;
    disp(output)
end
realmax
realmin
a = 'k--';
b = 'LineWidth';
c = 'This is a character array';
disp(c)
```

```
s1 = "This is a string";
disp(s1)

% Index the above character array
c(1)
c(5:10)
c(1:2:end)
% Use vector operations to determine the number of characters
 (including
% spaces) in a piece of text.
length(c)
% Use indexing on string. NOTE: only ONE element of s1
s1(1)
C = {'it is Tuesday', [1 5 9], [2 3 1; 5 9 7], ...
    uint8([1 4]), [.5; .1; .9], {'hello',[1 5]}}
C = {'it is Tuesday', [1 5 9], [2 3 1; 5 9 7], ...
    uint8([1 4]), [.5; .1; .9], {'hello',[1 5]}}
% Reference the cell container
C(1)
class(C(1))
% Reference the contents of a particular cell
C{1}
class(C{1})
for jj = 1:3
    myCell{jj} = rand(randi(5),randi(5));
end
length(myCell)
size(myCell)
myCell{1}
tic % initialize timer

myCell = cell(0); % create empty cell

% Keep filling cell array with random matrices
% until .25 secs have passed
while toc < .01
    myCell{end+1} = rand(randi(5),randi(5));
end
size(myCell)
for jj = 1:100
    myCell(jj) = [];
end
size(myCell)
myCell{1}
for jj = 1:100
    myCell{jj} = [];
end
size(myCell)
myCell{1}
%Specify a range of x
x = 0:0.01:1;
```

```
%Cell array to store line specifications
Str = {'k-', 'r-', 'k--', 'r--', 'b:'};

for k = 0:4
    %Plot legrendre polynomial
    plot(x,legendreP(x,k),Str{k+1},'LineWidth',mod(k+1,2)+1)
    hold on
end
hold off


% Graphical parameters
axis square
ylim([-1,1.5])
set(gca,'Color', [0.95, 0.95, 0.95])
xlim([0,1])
legend('P_0','P_1','P_2','P_3','P_4','Location','eastoutside')

% Step 1: Plot the raw data -> can do so by plotting T_wall vs time
%create a vector for time
t = 0:0.01:50;

Twall = @(h) 50*(1./(t+1) + 5*(t+1).^(-4*h)) + 300;

plot(t,Twall(1), 'k-')
axis square
xlabel('time (minutes)')
ylabel('Temperature (degrees C)')
ylim([350,600])
%Step 2: specify ranges of h
%"relevant" -> determined from our system
h = logspace(-2, 1, 200);

%preallocate the vector
t400 = zeros(1,length(h))

%Step 3: For each h value, solve for the cooling time
for k = 1:length(h)
   %Step 4: Store cooling time in a vector so that we can produce a
 plot
   t400(k) = newtonSolve(@(x) quiz3(x,h(k)),0,10^-4);
end

% Step 5: Produce a plot to visualize how t400 depends on the heat
 transfer
% coefficient
plot(h,t400,'k-')
%set graphical parameters
xlabel('Heat transfer coefficient')
ylabel('Time required to drop below 400 degrees')
axis square
set(gca, 'YScale', 'log')
set(gca, 'XScale', 'log')
set(gca, 'XMinorTick', 'on')
```

```
set(gca, 'TickLength', [0.02, 0.05])
grid on
% Heat transfer coefficient to consider
h1 = 0.07;

% Temperature
T_wall = @(h) 50*(1./(t+1) + 5*(t+1).^(-4*h)) + 300;

% FIRST Subplot: Brute Force approach
subplot(1,2,1)

% Temperature profile corresponding to tested heat transfer
 coefficient
TempH1 = T_wall(h1);

% Simulate "brute force" approach
t_BF = t(TempH1 > 400);
TempH2 = TempH1(TempH1 > 400);

% Plot temperature profile
plot(t,TempH1,'k-')
hold on
% Plot brute force calculations
plot(t_BF(1:100:end),TempH2(1:100:end),'ko')
hold off
axis square
title('Brute Force')


% SECOND Subplot: Newton-Raphson
subplot(1,2,2)

% Simulate Newton-Raphson:
% Determine t values for each iteration
t_NRiterations = newtonSolveReturnXvalues(@(x) quiz3(x,h1),0,10^-4);
% Determine corresponding temperature value
T_NR = 50*(1./(t_NRiterations+1) + 5*(t_NRiterations+1).^(-4*h1)) +
 300;

% Plot temperature profile
plot(t,TempH1,'k-')
hold on
% Plot each NR iteration
plot(t_NRiterations,T_NR,'ro')
hold off
axis square
title('Newton-Raphson')
close all % Close above plots
% Goal: perform some operation on each entry of the vector h.
% Declare vector of heat transfer coefficients
h = 1:0.1:10;
% Use a FOR loop, do some operation on each element of h and store the
% output in the corrsponding entry in some other vector
t_out = [];
```

```
for k = h
    t_out(end+1) = k^2;
end
% Use a FOR loop, do some operation on each element of h and store the
% output in the corrsponding entry in some other vector
t_out = zeros(1,length(h));
for k = 1:length(h)
    t_out(k) = h(k)^2;
end
a = 'k--';
b = 'LineWidth';
c = 'This is a character array';
disp(c)
s1 = "This is a string";
disp(s1)
% Index the above character array
c(1)
c(5:10)
c(1:2:end)
% Use vector operations to determine the number of characters
 (including
% spaces) in a piece of text.
length(c)
% Use indexing on string. NOTE: only ONE element of s1
s1(1)
% Create cell array
C = {'it is Thursday', [1 5 9], [2 3 1; 5 9 7], ...
    uint8([1 4]), [.5; .1; .9], {'hello',[1 5]}}
% Reference the cell container
C(1)
% Reference the contents of a particular cell
Str = C{1}
% The contents could then be processed
Str(1:9 )
txt1 = importdata("TextData.txt");
disp(txt1)
txt2 = fileread("TextData.txt")
disp(txt2)
size(txt1)
size(txt2)
M = ['hello' 'hi' ;'phone' 'it';'funny' 'oh']
% F = ['mat' 'labs' ;'fun' 'times';'a' 'b']
V = {'mat', 'labs' ;'fun' , 'times';'a', 'b'}
% Reference the text in the above cell array
c = V{2,1}
% Note the indexing is the same as before. Above, we have a cell array
 in
% the format of a vector.
d = txt1{10}
% Using vector indexing to retrieve sub arrays of characters from the
 "d"
t1 = d(1)
t2 = d(7)
t3 = d(1:2:end-2)
```

```
t4 = fliplr(d)
% Concatenate character arrays
d = [d,', yo']
% Expand array
d(end+1:end+5) = '. Woo'
% Delete entries
d([11:16,18:end-2]) = []
% Remove all spaces
d(d ~= ' ')
% Create new character array
t1 = txt1{29}
% Create subarray
t2  = txt1{29}(1:11)
% Return character array containing all "r"s
t3 = txt1{29}(txt1{29} == 'r')
S = ["position", "velocity";...
    "acceleration","g-force"]
% Return the string in index position 1 (single index notation)
S(1)
S(1,2) = "velocity in x"
S
% Convert one entry into a character sub array
c = char(S(2,1))
c = c(1:3)
c1 = txt1{39}
% Returns logical array indicating whether a particular character is a
% letter or not (true: that character is a letter)
v = isletter(c1)
% Convert all to upper case
sU = upper(c1)
% Convert all to lower case
sL = lower(c1)
% Create a character array of spaces
sB = blanks(7)
c1
% Create index at which word "dollars" appears
ind = strfind(c1, 'dollars')
% Create index at which word "Dollars" appears
ind = strfind(c1,'Dollars')
if ~strcmp('twenty',c1)
    disp('yes')
end
if "owl" ~= "owls"
    disp('yes')
end
% Assume: 10 image files are present in a folder and I want to easily
% open all of them. First: declare base file name
basefile = 'imageFile';
% Second: use a for loop to create text (that could be used to open
 each)
for k = 1:10
    disp([basefile,num2str(k),'.tif'])
end
%Step 1: Create variable with 46th of code from txt1
```

```
%Step 2: Can use spaces to determine where the words are -> have an
%isletter function and an isspace function
%This would determine where spaces are

%Step 3: for space, assign "previous" letters to a cell storing words

%add space to front and end of character array


txt1 = importdata('TextData.txt');
size(txt1)
%read as one long character array
txt2 = fileread('TextData.txt');
size(txt2)
% Read the 39th line as a character array
c = txt1{39}
% Using vector indexing to retrieve sub arrays of characters from the
 "c"
t1 = c(1:5)
% Concatenate array
t2 = [t1,' , ',t1]
% Create a text array without spaces
t2 = t2(t2 ~= ' ')
c = txt1(39)
% Logical array indicating positions that are letters
v1 = isletter(c)
% Logical array indicating positions that are spaces
v2 = isspace(c)
% Convert all to upper case
sU = upper(c)
% Convert all to lower case
sL = lower(c)
% Create index at which word "dollars" appears
ind = strfind(c,'twenty')
ind = strfind(c,'t')
% Step 1: Read the 46th line
c1 = txt1{46}

%Step 2: Remove punctiation
c1 = c1(isletter(c1)|isspace(c1))
%Step 3: Isolate each word
% Step 3a: add a space to front and back of text
c1 = [' ', c1, ' ']
% Step 3b: Determine where spaces are
spaces = strfind(c1, ' ')
% Step 3c: Store each word in cell array. note: if we have n spaces,
 then
% we will have n - 1 words.
myCell = cell(1,length(spaces)-1); % Preallocate cell array

% loop through all spaces to store words
for k = 1:length(myCell)
    myCell{k} = c1(spaces(k)+1:spaces(k+1)-1);
```

```
end
disp(myCell)
numWords = numel(myCell)
numLetters = 0;
%Loop through all words, add number of letters
for k = 1:numWords
    numLetters = numLetters + length(myCell{k})
end
avgLength = numLetters/numWords


% List of codons. Note the use of importdata()
codon_list = importdata('codonList.txt');
disp(codon_list)
% Import genomic information as one long character array.
% NOTE: difference from importdata()
sarsCov2_genome = fileread('sarsCov2_genome.txt');  % Read text file
sarsCov2_genome = sarsCov2_genome(isletter(sarsCov2_genome));   %
 remove spaces and commas
sarsCov2_genome = upper(sarsCov2_genome);   % Convert to upper case
disp(sarsCov2_genome)
% DNA sequence that you know starts the gene
spike_FWprimer = 'ATGTTTGTTTTTCTTGTTTTATTG';
% NOTE: one cell in codon_list (imported above) is a line of
 information
% containing 1) the nucleotides, the name, and abbreviation for
% each amino acid. We want each of these three items in a separate
 column
codon_list = importdata('codonList.txt');
disp(codon_list);
codons = cell(length(codon_list),3);

%process each line and seperate the elements

tempCodon = codon_list{k}
commaInd = strfind(tempCodon, ',')
for k = 1:length(codon_list)
    %First cell codon ATCG
    codons{k,1} = tempCodon(1:commaInd(1)-1);
    %second cell: name of amino acid
    codons{k,2} = tempCodon(1:commaInd(2)-1);
    %third cell: First cell codon ATCG
    codons{k,1} = tempCodon(commaInd(2)+1:end);

end
% Number of bases equals number of elements in the character vector
numBases = length(sarsCov2_genome)

% Step 1: Determine how many of each base are in the genome
base_A = length(sarsCov2_genome(sarsCov2_genome == 'A'))
base_T = length(sarsCov2_genome(sarsCov2_genome == 'T'))
base_C = length(sarsCov2_genome(sarsCov2_genome == 'C'))
base_G = length(sarsCov2_genome(sarsCov2_genome == 'G'))
```

```
% Step 2: Plotting
% Create cell array containing each x label
xval = {'A', 'T', 'C', 'G'}
%Convert cell array to categorical and then reorder to be in the same
 order
xval2 = categorical(xval);
xval2 = reordercats(xval2, xval)';

% Make bar graph
bar(xval2, [base_A, base_T, base_C, base_G]);
title('Bases in the Sars-Cov2 genome')
axis square
ylabel('Number');
xlabel('Base');
% List of codons. Note the use of importdata()
codon_list = importdata('codonList.txt');
disp(codon_list)
% Import genomic information as one long character array.
% NOTE: difference from importdata()
sarsCov2_genome = fileread('sarsCov2_genome.txt');  % Read text file
sarsCov2_genome = sarsCov2_genome(isletter(sarsCov2_genome));   %
 remove spaces and commas
sarsCov2_genome = upper(sarsCov2_genome);   % Convert to upper case
disp(sarsCov2_genome)
% DNA sequence that you know starts the gene
spike_FWprimer = 'ATGTTTGTTTTTCTTGTTTTATTG';
% NOTE: one cell in codon_list (imported above) is a line of
 information
% containing 1) the nucleotides, the name, and abbreviation for
% each amino acid. We want each of these three items in a separate
 column

% Preallocate cell array
codons = cell(length(codon_list),3);

% Loop through each entry in the list of codons
for k = 1:length(codon_list)
    % Read character arrays from the k-th row
    codon_row = codon_list{k};

    % Determine indices where commas occur
    ind = strfind(codon_row,',');

    % First column: bases
    codons{k,1} = codon_row(1:ind(1)-1);
    % Second column: corresponding codon name
    codons{k,2} = codon_row(ind(1)+1:ind(2)-1);
    % Third column: Amino acid
    codons{k,3} = codon_row(ind(2)+1:end);
end
% Number of bases equals number of elements in the character vector
numBases = length(sarsCov2_genome)
% Step 1: Determine how many of each base are in the genome
base_A = length(strfind(sarsCov2_genome, 'A'))
```

```
base_T = length(strfind(sarsCov2_genome, 'T'))
base_C = length(strfind(sarsCov2_genome, 'C'))
base_G = length(strfind(sarsCov2_genome, 'G'))

% Step 2: Plotting
% Create cell array containing each x label
xval = {'A','T','C','G'};
% Convert cell array to categorical and then reorder to be in the same
% order
xval2 = categorical(xval);
xval2 = reordercats(xval2,xval)';

% Make bar graph
bar(xval2,[base_A,base_T,base_C,base_G]);
title('Bases in the Sars-Cov-2 genome')
axis square
ylabel('Number');
xlabel('Base');
% Step 1: Find the index in the genome where the gene starts
DNA_start = strfind(sarsCov2_genome, spike_FWprimer)
% Define the DNA sequence starting at taht position
spike_DNA = sarsCov2_genome(DNA_start:end)
% Step 2: determine length of gene
% Goal: find each instance of a "possible" stop codon. In this case,
 the
% first one that occurs after a "multiple of three bases" will be it.
possibleStops = [];
stopCodons = ['TAA'; 'TAG'; 'TGA']

%Loop through sequence to find potential stop codons

for k = 1:3
    % stop codon to be considered
    tempStop = stopCodons(k,:);
    %find and add potential stop locations
    possibleStops = [possibleStops, strfind(spike_DNA, tempStop)];
end

%find the first stop position that is consisitent
possibleStops = sort(possibleStops, 'ascend')

%find positions in modulus of 3
lastCodon = possibleStops(mod(possibleStops,3) == 1);
lastBase = lastCodon(1) + 2;
% DNA sequence of spike protein
spike_DNA = spike_DNA(1:lastBase)
disp(spike_DNA)
% First, initialize the RNA sequence from the DNA sequence
spike_RNA = spike_DNA;

% The difference between DNA and RNA is that RNA contains uracil bases
 int
% he place of thymine
spike_RNA(spike_DNA == 'T') = 'U';
```

```
length(spike_RNA)
disp(spike_RNA)
% Step 1: Preallocate character array to contain protein sequence.
 Note:
% stop codon is not included in the aminoa cid sequence
spike_protein = blanks(length(spike_DNA)/3-1);


% Step 2: compare each three bases to list of codons to determine
% appropriate amino acod.
% String array: each line corresponds to a codon. To be used for
% identifying codon
codon_bases = string(codons(:,1));

% Note: we do not consider the Stop Codon
for k = 1:length(spike_protein)-3
    % Based on based pairs in codon, identify the index of the codon
 we are
    % interested in
    codon_index = codon_bases == string(spike_DNA((1:3)+(k-1)*3));

    % Retrive amino acid corresponding to codon and store in protein
    % sequence
    spike_Protein(k) = codons{codon_index,3};
end
length(spike_Protein)
disp(spike_Protein)


% To solve this: create a cell array containing "unique" amino acids
 and a
% corresponding vector that specifies if that amino acid has been so
 far

% Create cell array to store unique amino acids
AA_list = {spike_Protein(1)};

%Create a vector to store the corresponding counts
AA_count = 1;

% Loop through each amino acid of protien
for k = 2:length(spike_Protein)
    % Create a logical variable
    % True: amino acid has been observed before in protein sequence
    %False: no match -> new amino acid
    match = false;

    % For each identified "unique" amino acid
    for m = 1:length(AA_list)
        % If the amino acids match, increment the number of times it
        % appears
        if strcmp(AA_list{m}, spike_Protein(k))
            AA_count(m) = AA_count(m)+1;
            match = true;
```

```
        end
    end

    % If the current word did not match, then we have found a new
 amino
    % acid!
    if ~match
        AA_list{end+1} = spike_Protein(k);
        AA_count(end+1) = 1;
    end
end

% Step 2: Plot the distribution
% Convert cell array to categorical
xval_Q6 = categorical(AA_list);

% Make bar graph
h1 = bar(xval_Q6, AA_count);
title('Amino acids in the spike protein')
axis square
ylabel('Number');
xlabel('Amino Acid')
set(h1, 'horizontal', 'on')
```

```
% Write codons to a text file
writecell(codons,'myCodonFile.txt');
% Create and open a new file with specified name
fileID = fopen('S_ProteinSequence.txt','w');
% Write character array to file
fprintf(fileID,spike_Protein);
% Close file so that it can be read elsewhere
fclose(fileID);
% Create two fields for a structure array, a:
% a.b = 2;
% a.c.b = 'hello';
% a
% Access field values
% a.b
% a.c
% a.c.b
```

```
% a.c.c = [1,5,9];
% a.c.d = {'day',3};
% a(2).c.c = 'fun';
close all
x = 0:pi/24:2*pi;
y = sin(x);
h = plot(x,y,'k-');
axis square
get(h)
% Change line color
set(h,'Color','r')
% Change line width
h.LineWidth = 3;
% Change line style
h.LineStyle = ':';
% Import data as a structure array
myData = importdata('L18_TrackingData.txt')
% Numerical data
myData.data
% Text data
myData.textdata
close all % Close previous plots

% Create visualization: note
h1 = rectangle('Position',[0,0,1,1],'Curvature',[1,1]);
h2 = rectangle('Position',[3,3,1,1],'Curvature',[1,1]);

% Specify plotting parameters
axis square
xlim([0,10])
ylim([0,10])
% Step 1: Create cell array of just particle identifiers
particle_ID = myData.textdata(2:end,1);

% Step 2: Create subarrays unique to each particle
A_data = myData.data(strcmp(particle_ID,'A'),:);
B_data = myData.data(strcmp(particle_ID,'B'),:);

% Step 3: Populate structure array with data for both particles
particle(1).ID = 'A';
particle(1).pos = A_data(:,1:2);
particle(1).dia = A_data(:,3);

particle(2).ID = 'B';
particle(2).pos = B_data(:,1:2);
particle(2).dia = B_data(:,3);

particle
close all % Close previous plots

% Goal: loop through all time points, i.e. all rows int he positional
% arrays

% Create axes and specify graphical parameters
```

```
g = axes;
axis square
xlim([0,10])
ylim([0,10])

% Create visualization: note the values of each position
h(1) = rectangle('Position',
[particle(1).pos(1,:),particle(1).dia(1)*[1,1]],'Curvature',[1,1]);
h(2) = rectangle('Position',
[particle(2).pos(1,:),particle(2).dia(1)*[1,1]],'Curvature',[1,1]);

close all % Close previous plots

% Goal: loop through all time points, i.e. all rows in the positional
% arrays.

% Time points
timePts = size(particle(1).pos,1);

% Create axes and specify graphical paramters
axis square
xlim([0,10])
ylim([0,10])

%Create Initial position of particles
h(1) = rectangle('Position',
[particle(1).pos(1,:),particle(1).dia(1)*[1,1]],'Curvature',[1,1]);
h(2) = rectangle('Position',
[particle(2).pos(1,:),particle(2).dia(1)*[1,1]],'Curvature',[1,1]);
% Specify colors
h(1).FaceColor = 'r';
h(2).FaceColor = 'g';

% Loop through all time points
for k = 2:timePts
    % Pause for 1 second
    pause(1)

    % Update position and radius for each particle
    for m = 1:2
        h(m).Position =
 [particle(m).pos(k,:),particle(m).dia(k)*[1,1]];
    end
end

disp('Done')
% Read CSV file: Additional inputs specify "offset" to row/column to
 start
% reading from
data1 = csvread('L18_TrackingData.txt',1,1)
% Read audio file
[y_signal,Fs] = audioread('clip0.mp4');

% Sampling frequency (Hz)
```

```
disp(Fs)
% Size of audio signal
size(y_signal)

% Plot audio signal
plot(y_signal);
axis square
xlabel('index')
ylabel('signal')
% Listen to sound
sound(y_signal,Fs)
% Calculate a time vector using the sampling frequency
time = 1/Fs*(1:size(y_signal,1)); % seconds

% Plot audio signal
plot(time,y_signal)
axis square
xlabel('time (s)')
ylabel('audio signal')
% Depict size of the data array
size(y_signal)
% Listen to the first channel in mono
sound(y_signal(:,1),Fs)
% Listen to only the first channel in stereo: you should only hear it
 in
% one side of your headphones/speakers
sound([y_signal(:,1),zeros(size(y_signal(:,1)))],Fs)
% Listen to only the first channel in stereo: you should only hear it
 in
% the other side of your headphones/speakers
sound([zeros(size(y_signal(:,1))),y_signal(:,1)],Fs)
% Simulate noisy output from instrumentation
t = linspace(0,2*pi,100);
x = sin(t) + 0.3*rand(size(t));
% Plot noisy signal
plot(t,x,'k-')
axis square
xlabel('time (s)')
ylabel('voltage signal')
% Example: rolling filter applied to the data point n = 36 with a
 window
% size of 4. In this plot, the four data points will be averaged
 together,
% and the "filtered output" at the THIS data point will be this
 average.

% Plot noisy signal
plot(t,x,'k-')
hold on
plot(t(32:35),x(32:35),'ro')
hold off
axis square
xlabel('time (s)')
ylabel('voltage signal')
```

```
xlim([1,3])
ylim([0.7,1.3])
% Define the window
windowSize = 4;
% The numerator of the transfer function
% Note: 1/windowSize is the coefficient. Based on a window size of 4,
 we
% want to analyze x(n) ... x(n - 4 + 1). That is, we want to analyze
 the
% four "most recent" x values. This information is contained within
 the
% ones vector
b = ones(1,windowSize);
% The denominator of the transfer function
a = windowSize;
close all % Close previous plots

% Create the filtered output
y = filter(b,a,x);

% Plot noisy signal and filtered signal on same plot
g1 = axes;
plot(t,x,'k-')
hold on
plot(t,y,'r-','LineWidth',2)
hold off
axis square
xlabel('time (s)')
ylabel('voltage signal')
legend('Raw data','Filtered signal','Location','northeast')
close all % close previous figures

% Create a rolling mean filter
windowSize = 1000;
b = ones(1,windowSize);
a = windowSize;

y_filtered = filter(b,a,y_signal);

% Plot audio signal
plot(time,y_signal,'k-')
hold on
plot(time,y_filtered,'r-','LineWidth',2)
hold off
axis square
xlabel('time (s)')
ylabel('audio signal')
xlim([0,1])
% Create a time interval that is dependent upon the sampling frequency
Fs = 44100;
t = linspace(1/Fs,1,Fs);
% Produce a note corresponding to "middle C"
y_sound = sin(2*pi*262*t);
sound(y_sound,Fs)
```

```matlab
% Read the first image
image1 = imread('L19_image1.jpg');
imshow(image1)
% Dimensions of an image
size(image1)
% Total number of pixels
numel(image1)
% Bit depth of image
class(image1)
close all % close all figures
% Define color
myColor = [0.8,0.2,0.2];
axes
rectangle('Position',[0,0,2,2],'FaceColor',myColor)
% Load a similar color image
image1_color = imread('L19_image1color.jpg');
% Depict dimensions of color image
size(image1_color)
% Show color image
imshow(image1_color)
% Number of images
numIm = 3;
% Character array that is common to all image files
baseName = 'L19_image';

% Define imData as a structure array
imData = struct;
for k = 1:numIm
    imData(k).pic = imread([baseName,num2str(k),'.jpg']);
end
% Show desired image
imshow(imData(3).pic)
% Goal: create a 3-D matrix, where each image is on one *page* of this
% matrix

% Preallocate matrix
imDataMatrix = zeros([size(imData(3).pic),3]);
% Populate each page of the matrix
for k = 1:3
    imDataMatrix(:,:,k) = imData(k).pic;
end
% Ensure imDataMatrix is the correct data type
imDataMatrix = uint8(imDataMatrix);

% Dimensions of this image "matrix"
size(imDataMatrix)
% Find "median" image
image_median = median(imDataMatrix,3);
imshow(image_median)
% Read the first image
image1 = imread('L19_image1color.jpg');
imshow(image1)
% Bit depth of image
class(image1)
```

```
% Create a cropped image: note the array indices
imX = image1(1:500,800:1200,:);
imshow(imX)
% Dimensions of image1
size(imX)
% Display red channel
imX_red = imX;
imX_red(:,:,2:3) = 0;
imshow(imX_red)
% Display blue channel
imX_blue = imX;
imX_blue(:,:,1:2) = 0;
imshow(imX_blue)
% Create grayscale image from green channel
imX_gray = imX(:, :, 2);
imshow(imX_gray)
% Change the "brightness" of a given pixel
imshow(imX_gray + 100)
% Preallocate array to store information about image to be produced
imX2 = uint8(zeros(size(imX_gray)));

% Loop through some values to convert the image into "buckets"
% of only a few gray values
gvThresh = [0,50,120,200,255];
for k = 1:length(gvThresh)-1
    % All pixel between corresponding gray value limits are set to the
    % lower limit
    imX2(imX_gray > gvThresh(k) & imX_gray <= gvThresh(k+1)) =
 gvThresh(k);
end
% Display image
imshow(imX2)
% Create the kernel
B = ones(10);
B = B/sum(B,'all');

% Filter image using filter2
imX_filt = filter2(B, imX_gray)
% Convert filtered image to 8-bit
% NOTE: images must be 8-bit integer values
imX_filt = uint8(imX_filt);

% Create a subplot showing both images
% First image: unfiltered
subplot(1,2,1)
imshow(imX_gray)
title('Unfiltered')
% Second image: mean filtered
subplot(1,2,2)
imshow(imX_filt)
title('Mean filtered')
% Close all previous images
close all
% Create a binary image from imX2
```

```
imX_bw = imX_filt<50;
% Depict properties of the black/white image
whos imX_bw
% Display image
imshow(imX_bw)
% 1) Label all (1) objects in binary image
[L, num] = bwlabel(imX_bw);

% The number of unique binary objects
num
% Inspect different binary objects
imshow(L == 5)
% 2) Fill in dark (0) pixels INSIDE object
% Store specific binary image as a variable
im = L == 5;
im2 = imfill(im,'holes')
imshow(im2)
% Use regionprops to obtain information from the isolated object
pic_info = regionprops(im2, 'all')
% Area in pixels of binary object
a = pic_info.Area
% How "circular" the object is
b = pic_info.Circularity
% First, create a blank set of axes to add to
close all
axes;
xlim([0,10]);
ylim([0,10]);
axis square
% Add two rectangles
h1 = rectangle('Position',[2,1,2,2]);
h2 = rectangle('Position',[0,0,1,1],'Curvature',[1,1]);
h1.FaceColor = 'r';
h1.Position = [5,5,1,1];
text(2,2,'ENGR105')
figure
axis square
xlim([0,10])
ylim([0,10])
h3 = rectangle('Position',[1,1,1,1]);
% Create time and positional vectors
t = 1:10;
x = cos(t/10*2*pi)+1;
y = sin(t/10*2*pi)+1;

% For each time point specified above, move the rectangle to a new
 spot
for k = t
    % Pause MATLAB for 0.5 seconds
    pause(0.5)
    % Update to new position
    h3.Position = [x(k),y(k),1,1];
end
% New plot
```

```
figure;

% Step 1: create vectors for time and corresponding positions
t = 0:.1:2*pi;
x = sin(t);
y = cos(t);
z = t;

% Step 2: Plot nothing and establish axis bounds.
p = plot3(NaN,NaN,NaN,'go-');
xlim([-1,1])
ylim([-1,1])
zlim([0,2*pi])

% Step 3: the plot over time
for jj = 1:length(x)
    p.XData = x(1:jj);
    p.YData = y(1:jj);
    p.ZData = z(1:jj);
    pause(.1)   % Pause for 100 ms
end
close all % Close all previous figures
% Figure to be associated with callback
h1 = figure;
set(h1,'Visible','on')
% Specify the function that will be executed upon a key press event

% Initialize the figure and assign the keyboard callback
% function.
hFig = figure('Position',[200,200,1000,500]);

% Specify plot formatting
text(10,0,{'up arrow = move up','down arrow = move down', ...
    'left arrow = move left','right arrow = move right','q = quit'})
axis square
xlim([-10,10])
ylim([-10,10])

% Create a "ball" and place it at [0,0] initially
hBall = rectangle('Position',[0,0,1,1],'Curvature',[1,1]);

% Assign a keyboard call back
hFig.KeyPressFcn = {@DoWhenKeyIsPressed,hBall};

% Set figure to visible -> have it pop out so we can input the
 keyboard
% call backs
set(hFig,'Visible','on')
clear
%1) Vector representing the positions of the particles

pos = [1,4];

% particle one would be infected
```

```
%particle 2 is healthy

%3)produce visualization of the particles
g1 = axes;
axis image
xlim([0,7])
ylim([-2,2])
set(g1, 'ytick', [])
set(g1, 'Box', 'on')
xlabel('Position')

% Plot both particles initial positions
p1 = rectangle('Position',[pos(1),-0.5,1,1],'Curvature',
[1,1],'FaceColor','r');
p2 = rectangle('Position',[pos(2),-0.5,1,1],'Curvature',
[1,1],'FaceColor','w','LineWidth',2);

close all
%1) Vector representing the positions of the particles

pos = [1,4];

% particle one would be infected
%particle 2 is healthy

%3)produce visualization of the particles
g1 = axes;
axis image
xlim([0,7])
ylim([-2,2])
set(g1, 'ytick', [])
set(g1, 'Box', 'on')
xlabel('Position')

% Plot both particles initial positions
p1 = rectangle('Position',[pos(1),-0.5,1,1],'Curvature',
[1,1],'FaceColor','r');
p2 = rectangle('Position',[pos(2),-0.5,1,1],'Curvature',
[1,1],'FaceColor','w','LineWidth',2);

% Step throught he simulation. In each step, update the particles
 positions
for k = 1:50

    % A) Create vector determining particle step
    d_k = (rand(1,2)-0.5);

    % B) Update particle position
    pos = pos + d_k;

    % C) Ensure particle doesn't move out of bounds
    pos = max(min(pos,6),0);

    % D) Update visualization
```

```
    p1.Position(1) = pos(1);
    p2.Position(1) = pos(2);

    % E) Pause Matlab
    pause(0.2)
end


close all
%1) Vector representing the positions of the particles

pos = [1,4];

% particle one would be infected
%particle 2 is healthy

%3)produce visualization of the particles
g1 = axes;
axis image
xlim([0,7])
ylim([-2,2])
set(g1, 'ytick', [])
set(g1, 'Box', 'on')
xlabel('Position')

% Plot both particles initial positions
p1 = rectangle('Position',[pos(1),-0.5,1,1],'Curvature',
[1,1],'FaceColor','r');
p2 = rectangle('Position',[pos(2),-0.5,1,1],'Curvature',
[1,1],'FaceColor','w','LineWidth',2);

% Step throught he simulation. In each step, update the particles
 positions
for k = 1:50

    % A) Create vector determining particle step
    d_k = (rand(1,2)-0.5);

    % B) Update particle position
    pos = pos + d_k;

    % C) Ensure particle doesn't move out of bounds
    pos = max(min(pos,6),0);

    % Check if the particles interact
    if abs(pos(1) - pos(2)) < 1
        p2.FaceColor = 'r';
    end

    % D) Update visualization
    p1.Position(1) = pos(1);
    p2.Position(1) = pos(2);

    % E) Pause Matlab
```

```
    pause(0.2)
end

close all

% 1) vector representing positions of two particles
pos = [1,4];

% 2) Define particle identities.
% Particle 1 will be infected
% Particle 2 will intiially be healthy

% 3) Produce visualization of each particle
g1 = axes;
axis image
xlim([0,7])
ylim([-2,2])
set(g1, 'ytick', [])
set(g1, 'Box', 'on')
xlabel('Position')


% Plot both particles initial positions
p1 = rectangle('Position',[pos(1),-0.5,1,1],'Curvature',
[1,1],'FaceColor','r');
p2 = rectangle('Position',[pos(2),-0.5,1,1],'Curvature',
[1,1],'FaceColor','w','LineWidth',2);


% Step through the simulation. In each step, update the particles
 positions
while abs(pos(1) - pos(2)) >=1
    % A) Create vector determining particle step
    d_k = (rand(1,2)-0.5);

    % B) Update particle position
    pos = pos + d_k;

    % C) Ensure particle doesn't move out of bounds
    pos = max(min(pos,6),0);

    % Check if the particles interact
    if abs(pos(1) - pos(2)) < 1
        p2.FaceColor = 'r';
    end
    % D) Update visualization
    p1.Position(1) = pos(1);
    p2.Position(1) = pos(2);

    % E) Pause Matlab
    pause(0.2)
end

close all
```

```
% 1) vector representing positions of two particles
pos = [1,4];

% 2) Define particle identities.
% Particle 1 will be infected
% Particle 2 will intiially be healthy

% Number of time steps until infection occurs
numSteps = 0;

% Step through the simulation. In each step, update the particles
 positions

% Step through the simulation. In each step, update the particles
 positions
while abs(pos(1) - pos(2)) >=1
    % A) Create vector determining particle step
    d_k = (rand(1,2)-0.5);

    % B) Update particle position
    pos = pos + d_k;

    % C) Ensure particle doesn't move out of bounds
    pos = max(min(pos,6),0);

    % D) Update Counter
    numSteps = numSteps + 1;
end

disp(numSteps)



% Number of simulations
numSim = 1000;

% Preallocate storage vector
numSteps = zeros(1,numSim);

% Repeat simulation numSim times
for k = 1:numSim
    %1) Vector representing positions of two particles
    pos = [1,4];

    % 2) Define particle identities.
    % Particle 1 will be infected
    % Particle 2 will intiially be healthy

    % Step through the simulation. In each step, update the particles
 positions
    while abs(pos(1) - pos(2)) >=1
        % A) Create vector determining particle step
        d_k = (rand(1,2)-0.5);
```

```
        % B) Update particle position
        pos = pos + d_k;

        % C) Ensure particle doesn't move out of bounds
        pos = max(min(pos,6),0);

        % D) Update Counter
        numSteps(k) = numSteps(k) + 1;
    end

end

% Create visualization
histogram(numSteps)
xlabel('Number of steps')
ylabel('Counts')
xlim([0,700])
ylim([0,400])

% Number of simulations
numSim = 1000;

% Preallocate storage vector
numSteps = zeros(1,numSim);

% Repeat simulation numSim times
for k = 1:numSim
    %1) Vector representing positions of two particles
    pos = [1,4];

    % 2) Define particle identities.
    % Particle 1 will be infected
    % Particle 2 will intiially be healthy

    % Step through the simulation. In each step, update the particles
 positions
    while abs(pos(1) - pos(2)) >=1
        % A) Create vector determining particle step
        d_k = (rand(1,2)-0.5).*[0,1];

        % B) Update particle position
        pos = pos + d_k;

        % C) Ensure particle doesn't move out of bounds
        pos = max(min(pos,6),0);

        % D) Update Counter
        numSteps(k) = numSteps(k) + 1;
    end

end

% Create visualization
```

```
histogram(numSteps)
xlabel('Number of steps')
ylabel('Counts')
xlim([0,700])
ylim([0,400])


% Number of simulations
numSim = 500;

% Preallocate storage vector
numSteps_norm = zeros(1,numSim);
numSteps_quar = zeros(1,numSim);

% Repeat simulation numSim times
for k = 1:numSim
    % Simulate normal
    numSteps_norm(k) = simulateDisease([1,1]);
    % Simulate quarantine
    numSteps_quar(k) = simulateDisease([0,1]);
end

% create visualization
h1 = histogram(numSteps_norm);
hold on
h2 = histogram(numSteps_quar);
hold off


% Visualization
xlabel('Number of steps')
ylabel('Counts')
h1.BinWidth = 50;
h2.BinWidth = 50;
h1.DisplayStyle = 'stairs';
h2.DisplayStyle = 'stairs';
h2.LineWidth = 2;
h1.LineWidth = 1;
h2.EdgeColor = 'r';
h1.EdgeColor = 'k';
xlim([0,700])
ylim([0,400])
axis square
legend('Normal case', 'Quarantine', 'Location', 'Northeast')
currDir = cd;
disp(currDir)
cd(currDir)
dirFiles = dir;
files = {dirFiles.name};
size(files)
files{5}
% Initialize the figure and assign the keyboard callback
% function.
hFig = figure('Position',[200,200,1000,500]);
```

```
% Specify plot formatting
text(10,0,{'up arrow = move up','down arrow = move down', ...
    'left arrow = move left','right arrow = move right','q = quit'})
axis square
xlim([-10,10])
ylim([-10,10])

% Create a "ball" and place it at [0,0] initially
hBall = rectangle('Position',[0,0,1,1],'Curvature',[1,1]);
hBall.FaceColor = 'k';

% Assign a keyboard call back
hFig.KeyPressFcn = {@Topic21DoWhenKeyIsPressed,hBall};

% Set figure to visible -> have it pop out so we can input the
 keyboard
% call backs
set(hFig,'Visible','on')
close all % Close previous windows

% Initialize the figure and assign the keyboard callback
% function.
hFig = figure('Position',[200,200,1000,500]);

% Specify plot formatting
text(10,0,{'up arrow = move up','down arrow = move down', ...
    'left arrow = move left','right arrow = move right','q = quit'})
axis square
xlim([-10,10])
ylim([-10,10])

% Create a "ball" and place it at [0,0] initially
hBall = rectangle('Position',[0,0,1,1],'Curvature',[1,1]);
hBall.FaceColor = 'k';  % Default color is black

% Assign a keyboard call back for when a buttom is pressed
hFig.KeyPressFcn = {@ex3_keyPress,hBall};
% Assign a keyboard call back for when a button is release
hFig.KeyReleaseFcn = {@ex3_keyRealease,hBall};

% Set figure to visible -> have it pop out so we can input the
 keyboard
% call backs
set(hFig,'Visible','on')
close all % close previous figures

% Create a figure to be associated with the mouse callback
hFig1 = figure('Visible','on');
axis equal
xlim([0,10])
ylim([0,10])

% Initialize a text object for displaying coordinates
```

```
g = text(5,5,'x = NaN, y = NaN');

% Assign mouse callback function
hFig1.WindowButtonMotionFcn = {@moveText,g};
close all

% Create a figure and establish the callback associated with a
% mouse button press.
figure('WindowButtonDownFcn',@click,'Visible','on')
close all

% Create a figure with mouse button press callback and plot
% NaNs (nothingness) on the figure.
hFig2 = figure('Visible','on');
h = plot(NaN,NaN,'ro-','MarkerFaceColor','r','MarkerSize',5);

% Assign callback function
hFig2.WindowButtonDownFcn = {@plotPoints1,h}

% Graphical parameters
axis equal
xlim([0,10])
ylim([0,10])
close all % Delete previous

% Create a figure with mouse button press callback
hFig3 = figure('Visible','on');

% Plot each color line, initially as NaN. The first plot (handle =
 hRed)
% hold the red points and the second plot (handle = hGreen) hold
% the green points.
hRed = plot(NaN,NaN,'ro-','MarkerFaceColor','r','MarkerSize',5);
hold on
hGreen = plot(NaN,NaN,'go-','MarkerFaceColor','g','MarkerSize',5);
hold off

% Assign callback function
hFig3.WindowButtonDownFcn = {@plotPoints2, hRed, hGreen};

axis square
xlim([0,10])
ylim([0,10])
close all % Delete previous

% Create a figure with mouse button press callback
hFig3 = figure('Visible','on');

% Create a data structure to store the order of moves
handles.order = cell(0);
% Update the guidata with that data structure
guidata(hFig3,handles);
```

```
% Plot each color line, initially as NaN. The first plot (handle =
 hRed)
% hold the red points and the second plot (handle = hGreen) hold
% the green points.
hRed = plot(NaN,NaN,'ro-','MarkerFaceColor','r','MarkerSize',5);
hold on
hGreen = plot(NaN,NaN,'go-','MarkerFaceColor','g','MarkerSize',5);
hold off

% Assign callback function
hFig3.WindowButtonDownFcn = {@lotPoints3, hRed, hGreen};

axis square
xlim([0,10])
ylim([0,10])
% Play tic tac toe against AI player who plays randomly
ticTacToe(@playRandom)
% Create figure objects
h1 = figure('Visible','on');
g1 = axes;

% Plot grid lines
plot([-1,-1],[-4,4],'k-')
hold on
plot([1,1],[-4,4],'k-')
plot([-4,4],[1,1],'k-')
plot([-4,4],[-1,-1],'k-')
hold off

% Modify graphical parameters
xlim([-3,3])
ylim([-3,3])
g1.XColor = 'none';
g1.YColor = 'none';
g1.Position(3:4) = 0.75*g1.Position(3:4);
g1.OuterPosition(1) = 0.125;
axis square

% Create two empty text containers. On every turn, we will update the
 the
% position of either the next X or next O marker from NaN to the
% appropriate coordinates
x1 =
 text(NaN(5,1),NaN(5,1),'X','FontName','Arial','FontSize',50,'FontWeight','bold');
x2 =
 text(NaN(4,1),NaN(4,1),'O','FontName','Arial','FontSize',50,'FontWeight','bold');

% Create data structure to store gameboard
handles.gameboard = zeros(3);
% Store data structure
guidata(h1,handles);

% Assign callback function for mouse click
h1.WindowButtonDownFcn = {@placeX,x1};
```

```
% Assign callback function for mouse release
h1.WindowButtonDownFcn = {@place0,x2,@playRandom}

% Create figure objects
h1 = figure('Visible','on');
g1 = axes;

% Plot grid lines
plot([-1,-1],[-4,4],'k-')
hold on
plot([1,1],[-4,4],'k-')
plot([-4,4],[1,1],'k-')
plot([-4,4],[-1,-1],'k-')
hold off

% Modify graphical parameters
xlim([-3,3])
ylim([-3,3])
g1.XColor = 'none';
g1.YColor = 'none';
g1.Position(3:4) = 0.75*g1.Position(3:4);
g1.OuterPosition(1) = 0.125;
axis square

% Create two empty text containers. On every turn, we will update the
 the
% position of either the next X or next O marker from NaN to the
% appropriate coordinates
x1 =
 text(NaN(5,1),NaN(5,1),'X','FontName','Arial','FontSize',50,'FontWeight','bold');
x2 =
 text(NaN(4,1),NaN(4,1),'O','FontName','Arial','FontSize',50,'FontWeight','bold');

% Create data structure to store gameboard
handles.gameboard = zeros(3);
% Store data structure
guidata(h1,handles);

% Assign callback function for mouse click
h1.WindowButtonDownFcn = {@placeX,x1};
% Assign callback function for mouse release
h1.WindowButtonUpFcn = {@placeO,x2,@playRandom};

% Pushbutton to subtract 1 from the current value of the counter
u1 =
 uicontrol('Style','pushbutton','Units','Normalized','Position',...
    [0,0.4,0.2,0.2],'String','New Game');
u1.Callback = {@resetGame,h1,x1,x2,@placeX,@placeO,@playRandom};
u1.FontSize = 14;
% Step 1: Read data from text file. We will use readcell
patientData = readcell('patientData.txt');

% Step 2: Prepare and preallocate necessary vectors
trueResult = zeros(size(patientData,1),1);  % Blood test result
```

```
diagResult = zeros(size(patientData,1),1);  % Diagnostic

% Step 3: Loop through each row (data point / patient) of the
 collected data
for k = 1:size(patientData,1)

    % Evaluate if patient has sickle cell disease based on sequence.
 Will
    % return 1 if they have sickle cell, 0 if they do not
    trueResult(k) = patientData{k,1}(20) == 'U';

    % Evaluate what the diagnostic predicts. Will return 1 if
 diagnostic
    % indicates sickle cell disease, 0 if it does not.
    diagResult(k) = strcmp(patientData{k,2},'sicklecell');

end

% Determine "accuracy" of diagnostic result
% accuracy1: simply finds proportion of predicted true relative to
 absolute
% true
accuracy1 = sum(diagResult(trueResult == 1))/
length(trueResult(trueResult == 1))
% accuracy2 (better): determines the proportion of test results that
 the
% diagnostic accurately classified as having sickle cell or not
accuracy2 = sum(trueResult == diagResult) / length(trueResult)
% Determine patients that have the disease
trueResult = cellfun(@(y) y(20) == 'U', patientData(:,1));

% Result of diagnostic
diagResult = cellfun(@(x) strcmp(x,'sicklecell'),patientData(:,2));

% Determine accuracy, as above
accuracy1 = sum(diagResult(trueResult == 1))/
length(trueResult(trueResult == 1))
accuracy2 = sum(trueResult == diagResult) / length(trueResult)
% Vector
v = [1,5,15,50,100,200]
% Numeric derivative
dv = diff(v)
% Load the data from the excel file
[data,headers] = xlsread('L23_UB_data.xlsx');
% 1) Place the data in time and position vectors
t = data(:,1);
pos = data(:,2);
% 2) Calculate the velocity and acceleration as a
% function of time
v = diff(pos)./diff(t);
a = diff(v)./diff(t(2:end));
close all
% Produce a plot depicting all results
fs = 12;    % Font size
```

```matlab
ms = 16;    % Marker size

% First plot: position vs. time
g1 = axes;
g1.Position = [0.15,0.15,0.7,0.22];
plot(t,pos,'k.-','MarkerSize',ms);
xLim = xlim;
xlabel('Time (s)')
ylabel('Pos. (m)')
g1.FontSize = fs;

% Second plot: velocity vs. time
g2 = axes;
g2.Position = [0.15,0.44,0.7,0.22];
plot(t(2:end),v,'k.-','MarkerSize',ms);
ylabel('Vel. (m/s)')
axis tight
xlim(xLim)
g2.XTick = [];
g2.FontSize = fs;

% Third plot: acceleration vs. time
g3 = axes;
g3.Position = [0.15,0.73,0.7,0.22];
plot(t(3:end),a,'k.-','MarkerSize',ms);
hold on
plot(xLim,[0 0],'k-')
hold off
set(gca,'FontSize',fs)
ylabel('Acc. (m/s^2)')
axis tight
xlim(xLim)
g3.XTick = [];
g3.FontSize = fs;
title('Usain Bolt''s world record 100m sprint')
% Declare 2-D array
A = [4,5,6;3,2,1]
% Difference over rows
diff(A,1,1)
% Difference over columns
diff(A,1,2)
% X values
x = -3:0.5:3;
% Y values
y = -2:0.5:2;
% Create meshgrid
[X,Y] = meshgrid(x,y)
% Specify X component of velocity
VX = Y/max(Y,[],'all');
% Specify Y component of velocity
VY = -X/max(X,[],'all');
close all
% Visualize the velocity profile
q = quiver(X,Y,VX,VY);
```

```
q.Color = 'k';
set(gca,'YAxisLocation','Origin')
set(gca,'XAxisLocation','Origin')
set(gca,'Box','off')
axis square
title('Velocity Field')
close all

% Plot the X component of the velocity
figure
surface(X,Y,VX)
axis square
title('VX')
xlabel('X')
ylabel('Y')
colorbar
colormap('jet')

% Plot the Y component of the velocity
figure
surface(X,Y,VY)
axis square
title('VY')
xlabel('X')
ylabel('Y')
colorbar
colormap('jet')

% Plot the magnitude of the velocity
figure
surface(X,Y,sqrt(VY.^2+VX.^2))
axis square
title('|V|')
xlabel('X')
ylabel('Y')
colorbar;
colormap('jet')
close all
% Compute the curl and angular velocity of the above vector field
[CURLZ, CAV]= curl(X,Y,VX,VY);

% Plot the X component of the velocity
surface(X,Y,CURLZ)
axis square
title('Curl (Z component)')
xlabel('X')
ylabel('Y')
colorbar
colormap('jet')
close all
% Compute the curl and angular velocity of the above vector field
[GX,GY] = gradient(sqrt(VX.^2+VY.^2),0.5,0.5);

% Plot the X component of the velocity
```

```
q = quiver(X,Y,GX,GY);
axis square
xlim([-3,3])
ylim([-3,3])
q.Color = 'k';
set(gca,'YAxisLocation','Origin')
set(gca,'XAxisLocation','Origin')
set(gca,'Box','off')
axis square
title('Gradient of Velocity field')
close all
% New variable C
CX = cos(X);
CY = sin(Y);
quiver(X,Y,CX,CY)

figure
% Calculate the divergence
DIV = divergence(X,Y,CX,CY);

% Plot the X component of the velocity
surface(X,Y,DIV)
axis square
title('Divergence')
xlabel('X')
ylabel('Y')
colorbar
colormap('jet')
close all
% Define X and Y variables
x = 0:0.1:10;
y = -sin(x);
% Calculate definite integral over time
Y = zeros(size(y));
for k = 2:length(y)
    Y(k) = trapz(x(1:k),y(1:k));
end

% Plot the function and its *definite* integral
plot(x,y,'k-')
hold on
plot(x,Y,'r-','LineWidth',2)
hold off
% Create function handle for function of interest
F = @(x) -sin(x)
% Calculate integral
q = integral(F,0,pi)
close all

% Create figure
h1 = figure;

plot([0,20],3.5*[1,1])
hold on
```

```
% Output "sensor" data
g_out = plot(0,0,'ko');
hold on
% Input "sensor" data
g_in = plot(0,0,'ro');
hold off

% Graphical parameters
xlim([0,20]);
ylim([0,5]);
axis square
ylabel('Sensor signals')
xlabel('Time (s)')
legend('System output','System Input','Location','NorthEast')

% Counter to store time
time = 0;
outError = 0;

% Plot in real time for 20 seconds
while time < 20

    % Pause for 100 ms
    pause(0.1);

    % Update time
    time = time + 0.1;

    % The following describes how the system responds to a given input
    % Assume the system drifts in some time dependent fashion
    outError = outError + (abs(mod(time,1)) < 0.1)*(rand - 0.5);
    % Function describing how output responds to input
    y = @(x) sqrt(x) + 2 + 0.5*outError;

    % Update vector to contain input information
    g_in.XData = [g_in.XData,time];
    g_in.YData = [g_in.YData,2];

    % Update vector to contain input information
    g_out.XData = [g_out.XData,time];
    g_out.YData = [g_out.YData,y(g_in.YData(end))];

end
close all

% Create figure
h1 = figure;

% Determine our goal set point
ysp = 3.5;

plot([0,20],ysp*[1,1],'k--')
hold on
% Output "sensor" data
```

```
g_out = plot(0,0,'ko');
% Input "sensor" data
g_in = plot(0,0,'ro');
plot(xlim,ysp,'k-')
hold off

% Graphical parameters
xlim([0,20]);
ylim([0,5]);
axis square
ylabel('Sensor signals')
xlabel('Time (s)')
legend('Set point','System output','System
 Input','Location','NorthEast')

% Counter to store time
time = 0;
outError = 0;

% Declare a variable to represent the error from the set point
x_err = 0;

% Plot in real time for 20 seconds
while time < 20

    % Pause for 100 ms
    pause(0.1);

    % Update time
    time = time + 0.1;

    % The following describes how the system responds to a given input
    % Assume the system drifts in some time dependent fashion
    outError = outError + (abs(mod(time,1)) < 0.1)*(rand - 0.5);
    % Function describing how output responds to input
    y = @(x) sqrt(x) + 2 + 0.5*outError;

    % DEFINE CONTROLLER HERE
    if time > 0.1



    end


    % Update vector to contain input information
    g_in.XData = [g_in.XData,time];
    g_in.YData = [g_in.YData,max(0,x_err)];

    % Update vector to contain input information
    g_out.XData = [g_out.XData,time];
    g_out.YData = [g_out.YData,y(g_in.YData(end))];

end
```

```
% x range of interest
x = 0.5:0.1:2.5;
% Actual function
f = log(x);

% Taylor series approximations
f_ts1 = x-1;
f_ts2 = x-1 - 1/2*(x-1).^2;
f_ts3 = x-1 - 1/2*(x-1).^2 + 1/3*(x-1).^3;


subplot(1,3,1)
plot(x,f,'k-',x,f_ts1,'r-')
axis square
ylim([-1,2])
title('First order')
subplot(1,3,2)
plot(x,log(x),'k-',x,f_ts2,'r-')
axis square
ylim([-1,2])
title('Second order')
subplot(1,3,3)
plot(x,log(x),'k-',x,f_ts3,'r-')
axis square
ylim([-1,2])
title('Third order')
close all
% Time vector
t = 0:0.1:3;

% The function representing the RHS of the differential equation. This
 is
% defined by the problem
f = @(T,X) X-T^2+1;

% Analytical solution
x_an = (1-exp(t))+t.^2+2*t;

% Use Euler method
% Define the time step
dt = 1;

% Preallocate vector to store
t_approx = 0:dt:3;
x_euler = zeros(size(t_approx));

% Calculate each entry in the Euler approximation
for k = 1:length(x_euler)-1
    % Apply the Euler method
    % Calculate the next value of the solution
    x_euler(k+1) = x_euler(k) + f(t_approx(k),x_euler(k))*dt;
end

% Plot the solution
```

```
plot(t,x_an,'k-',t_approx,x_euler,'ro-')
axis square
legend('Analytical solution','Euler
 approximation','location','southwest')
close all
% Define the time step
dt = 0.1;

% Preallocate vectors to store
t_approx = 0:dt:3;                % Time vector for all
x_euler = zeros(size(t_approx)); % Euler
x_pc = zeros(size(t_approx));    % Predictor corrector

% Calculate each entry in the Euler approximation
for m = 1:length(x_euler)-1
    % Use Euler method
    % Calculate the next value of the solution
    x_euler(m+1) = x_euler(m) + f(t_approx(m),x_euler(m))*dt;

    % Use predictor-corrector method
    k1 = f(t_approx(m),x_pc(m));
    k2 = f(t_approx(m)+dt,x_pc(m)+k1*dt);

end

% Plot the solutions
plot(t,x_an,'k-', ...
    t_approx,x_euler,'ro-',...
    t_approx,x_pc,'mo-.')
axis square
legend('Analytical solution','Euler approximation','Predictor-
corrector',...
    'location','southwest')
close all

% Time vector
t = 0:0.1:7;
% Analytical solution
x_an = (1-exp(t))+t.^2+2*t;

% Define the time step
dt = 1;

% Preallocate vectors to store
t_approx = 0:dt:7;                % Time vector for all
x_euler = zeros(size(t_approx)); % Euler
x_pc = zeros(size(t_approx));    % Predictor corrector
x_rk = zeros(size(t_approx));   % Runge-Kutta

% Calculate each entry in the Euler approximation
for m = 1:length(x_euler)-1
    % Use Euler method
    % Calculate the next value of the solution
    x_euler(m+1) = x_euler(m) + f(t_approx(m),x_euler(m))*dt;
```

```
    % Use predictor-corrector method
    k1 = f(t_approx(m),x_pc(m));
    k2 = f(t_approx(m)+dt,x_pc(m)+k1*dt);
    x_pc(m+1) = x_pc(m) + 1/2*(k1+k2)*dt;

    % Use Runge-Kutta method
    K1 = f(t_approx(m),x_pc(m));
    K2 = f(t_approx(m)+dt/2,x_pc(m)+K1*dt/2);
    K3 = f(t_approx(m)+dt/2,x_pc(m)+K2*dt/2);
    K4 = f(t_approx(m)+dt,x_pc(m)+K3*dt);
    x_rk(m+1) = x_rk(m) + 1/6*(K1 + 2*K2 + 2*K3 +K4)*dt;

end

% Plot the solutions
plot(t,x_an,'k-', ...
    t_approx,x_euler,'ro-',...
    t_approx,x_pc,'mo-.')
hold on
plot(t_approx,x_rk,'bo-','MarkerFaceColor','b')
hold off
axis square
legend('Analytical solution','Euler approximation','Predictor-
corrector',...
    'Runge-Kutta','location','southwest')
close all
% Solve ODE using ode45
[ts,xs] = ode45(@myODE,[0,7],[0]);

% Compare to analytical solution
plot(t,x_an,'k-',ts,xs,'ro')
axis square
legend('Analytical solution','ode45','location','southwest')
close all
% Solve ODE using ode45
[t,moles] = ode45(@rxnEqns,[0,10],[1,0,0]);

% Plot time-dependent profiles
plot(t,moles(:,1),'k-')
hold on
plot(t,moles(:,2),'r-','LineWidth',2)
plot(t,moles(:,3),'b-')
hold off

% Graphical parameters
axis square
legend('Species A','Species B','Species C','location','northeast')
ylim([0,1])
ylabel('Amount of chemical (moles)')
xlabel('time (s)')
title('Temporal profile of chemical reaction')
% Solve system of linear ODES
[t,x] = ode45(@thirdOrderODE,[0,10],[10,1,2]);
```

```
% Plotting system
p = plot(t,x(:,1),'k-',...
    t,x(:,2),'b-',...
    t,x(:,3),'r-');
set(p,'LineWidth',3)
set(gca,'FontSize',12)
axis square
xlabel('t (s)')
ylabel('pos. (m), vel. (m/s), acc. (m/s^2)')
legend({'Position (y(t))','Velocity (y''(t))','Acceleration
 (y''''(t))'},'location','southoutside')
p = 2;
q = 1.5;
r = 0.1;

% Solve the ODE
[t,x] = ode45(@thirdOrderODE_extraInputs,[0,10],[10,1,2],[],p,q,r);
% Plotting system
p = plot(t,x(:,1),'k-',...
    t,x(:,2),'b-',...
    t,x(:,3),'r-');
set(p,'LineWidth',3)
set(gca,'FontSize',12)
axis square
xlabel('t (s)')
ylabel('pos. (m), vel. (m/s), acc. (m/s^2)')
legend({'Position (y(t))','Velocity (y''(t))','Acceleration
 (y''''(t))'},'location','southoutside')
% Specify angle and velocity of the ball
th = 45;
v = 20;
vx = v*cosd(th);
vy = v*sind(th);
% Solving the set of ODEs. Note the additional input (x velocity)
[~,state] = ode45(@ball_2D,[0,5],[0,vy,0],[],vx);
x_ode = state(:,3); % x values
y_ode = state(:,1); % y values

figure
% Plot the height as a function of time
plot(x_ode,y_ode,'k-')
ylim([0,15])
xlabel('Distance (m)')
ylabel('Height (m)')
axis square
% Phyiscal parameters, as before
th = 60;
v = 15;
vx = v*cosd(th);
vy = v*sind(th);
g = 9.81;
% Estimate time to landing
t_end = 1.5*2*vy/g;
```

```
% Solve system of differential equations. Specify the time points to
 solve
% for
[t,state] = ode45(@ball_2D,[0:0.05:t_end],[2,vy,0],[],vx);
txy = [t,state(:,3),state(:,1)];
% Select time points corresponding to the ball not yet landing
txy = txy(txy(:,3) > 0,:);
close all

% Produce figure. Specify the plots of interest and the graphical
% parameters
h1 = figure('Visible','on');
hTrajectory = plot(txy(1,2),txy(1,3),'r--');
hold on
hBall =
 plot(txy(1,2),txy(1,3),'ro','MarkerFaceColor','r','MarkerSize',8);
hold off
ylim([0,15])
xlim([-5,20])
xlabel('Distance (m)')
ylabel('Height (m)')
axis square

% Loop through all ball positions and update plot
for k = 2:size(txy,1)

    % Update plot of ball and trajectory tracer
    hBall.XData = txy(k,2);
    hBall.YData = txy(k,3);
    hTrajectory.XData(k) = txy(k,2);
    hTrajectory.YData(k) = txy(k,3);

    % Pause for specified amount of time consisted with time points
 solved
    % for
    pause(0.05);

end
close all
% Create a figure
h1 = figure('Visible','on');

% Create and store GUI data structure
handles.h0 = 2;
guidata(h1,handles);

% Define plots for trajectory, ball, and aiming line
hTrajectory = plot(0,handles.h0,'r--');
hold on
hBall = plot(0,handles.h0,'ro','MarkerFaceColor','r','MarkerSize',8);
hAim = plot(0,handles.h0,'k-','LineWidth',2);
hold off
```

```
% Graphical parameters
ylim([-5,15])
xlim([-5,20])
xlabel('Distance (m)')
ylabel('Height (m)')

% Assign callback function for mouse click
h1.WindowButtonDownFcn = {@determineStart,hBall,hTrajectory,hAim};

% Assign callback function for mouse release
h1.WindowButtonUpFcn = {@throwBall,hBall,hTrajectory};
% Using the numerical calculation, as before
sin(pi)
% Using symbolic
sin(sym(pi))
% Declare symbolic variables
syms x y
% Inspect properties of variables
whos x y
% Create symbolic numbers
sym(5)
sym(pi)
sym(sqrt(2))
1+sym(pi)
% Correct: convert pi to symbolic, then operate on it.
sqrt(sym(pi))
% Incorrect: Convert the output from sqrt(pi) to a symbolic
 representation
sym(sqrt(pi))
% Create symbolic variable phi, also known as the "golden ratio"
phi = (1 + sqrt(sym(5)))/2
% Convert symbolic to double
double(phi)
% Operation 1
f1 = phi^2 - phi - 1
% Operation 2
f2 = phi^2 + 1
% Simplify f1
simplify(f1)
% Simplify f2
simplify(f2)
% Create original equation
syms x
g = x^3 + 6*x^2 + 11*x + 6

% Manipulate equation by dividing it by (x+3)
g2 = g/(x+3)
% Simplify equation
g2 = simplify(g2)
% Parenthesis cannot be used to substitute for a variable
g2(1)
% Instead, use subs to evaluate for particular values
subs(g2,x,1)
% Note: the original equation is unchanged
```

```
g2
clear
syms f(x)
% Specify function. NOTE: must define f(x)
f(x) = x^2-1+sqrt(sym(pi));
% the function f(x) will be displayed if either f or f(x) is called
disp(f(x))
disp(f)
% In this case, direct substitution can occur with parenthesis
f(1)
f(2)
clear
% Declare symbolic variables
syms x
% Plot sin(x) over the domain [0,10]
fplot(sin(x), [0,10], 'r--')
% Declare symbolic variable
syms t

% Plot function and specify parameters
h1 = fplot(t^2,[-2,2]);
h1.LineWidth = 2;
h1.Color = 'g';
axis square
% Acquire data vectors that were plotted to create the above plot.
xdata = h1.XData;
ydata = h1.YData;
% Equivalent plot
plot(xdata,ydata,'g-','LineWidth',2)
axis square
clear
% Produce a plot of cos(x) between 0 and 4*pi
fplot(@(x) cos(x),[0,4*pi])
% Declare symbolic variable
syms t
% Produce a 3D plot corresponding to the above
fplot3(t,sin(t),cos(t),[-10,10])
xlabel('x')
ylabel('y')
zlabel('z')

% Add constant rotation to the plot
for k = 1:360
    % Update the view point
    view(k,30)
    % Pause MATLAB for 100 ms
    pause(0.1)
end
syms x y
fimplicit(1==y^2+x,[-2,2])
set(gca,'XAxisLocation','Origin')
set(gca,'YAxisLocation','Origin')
clear
syms x   % Declare symbolic variable
```

```
% Example 1
solve(5*x == 1)
% Example 2: multiple solutions are present
solve(x^2 - 1 == 0)
% Example 3: can obtain complex solutions
solve(x^2 + 1 == 0)
clear
syms x y % declare variables as symbolic
eqn1 = y==2*x-3;    % first equation
eqn2 = y==x;    % second equation
res = solve([eqn1,eqn2],[x,y]);
xSol = res.x % retrieve the x solution
ySol = res.y % retrieve the y solution
tic
syms x1 x2 x3
% Define equations
eqn1 = x1 + 3*x2 + x3 == 1;
eqn2 = 2*x1 + 2*x2 - x3 == 2;
eqn3 = x1 + x2 + x3 == -2;
% Solve set of linear equations
res = solve([eqn1,eqn2,eqn3],[x1,x2,x3]);
% Display the solution for each variable
disp(res.x1)
disp(res.x2)
disp(res.x3)
toc
tic
% Step 1: convert to matrix format
A = [1,3,1;2,2,-1;1,1,1]; %Coefficient matrix defined above
b = [1;2;-2]; % Constant matrix
% Step 2: solve for x
x = A\b
toc
% Declare variables and equations
syms x y
eqn1 = y^2==x;
eqn2 = y==2*x-3;
% Plot equations on the interval for x and y of [-6,6]
fimplicit([eqn1,eqn2],[-6,6,-6,6])
axis square
% Solve system of nonlinear equations
res2 = solve([eqn1,eqn2],[x,y]);
xSol2 = res2.x % retrieve the x solution
ySol2 = res2.y % retrieve the y solution
% Declare variables and equations
syms x y
eqn1 = y^2==x;
eqn2 = y==2*x^2-3;
% Plot equations on the interval for x and y of [-6,6]
fimplicit([eqn1,eqn2],[-6,6,-6,6])
axis square
% Solve system of nonlinear equations
res3 = solve([eqn1,eqn2],[x,y]);
xSol3 = res3.x % retrieve the x solution
```

```
ySol3 = res3.y % retrieve the y solution
% Determine solution 1
sol_1 = fsolve(@eqnSet,[0,0]);
% Determine solution 2
sol_2 = fsolve(@eqnSet,[1,1]);
disp(sol_1)
disp(sol_2)
syms x
diff(sin(x), x)
F = sin(x);
diff(F,x)
diff(F,x,2)
syms x y
diff(sin(x)*cos(y),x)
diff(sin(x)*cos(y),y)
diff(x*sin(x*y), x, y)
syms x % declare x symbolic
% Integrate x^2 + x
res1 = int(x^2+x)
% Expand the polynomial
res1 = expand(res1)
syms x y z
res = expand(int(int(x^2+x+y^3+z^(1/2),y),x))
syms theta
res = int(sin(theta),[pi/4 pi/2])
syms x
int(sin(sinh(x)), x)
clear
% Declare symbolic variables
syms y(t) a
% Define differential equation
eqn = diff(y,t) == a*y;
S = dsolve(eqn)
% Define initial conditions
cond = y(0) == 5;
ySol(t) = dsolve(eqn,cond)
% Define second order ODE. % Note the additional input to diff() to
 create
% a second order derivative.
eqn = diff(y,t,2) == a*y;
% Solve for general solution to differential equation
ySol(t) = dsolve(eqn)
% Specify initial conditions

% Define the first derivative so that it can be used as an initial
% condition
Dy = diff(y,t)
syms b
% Define vector containing initial conditions
cond = [y(0)==b, Dy(0)==1];
% Solve for the specific form of the ODE solution
ySol(t) = dsolve(eqn,cond)
% Define ODEs
syms y(t) z(t)
```

```matlab
% Define vector of ODEs
eqns = [diff(y,t) == z, diff(z,t) == -y];
% Find solution
S = dsolve(eqns);
S.z
S.y
S = dsolve(eqns,[y(0) == 1,z(0) == 0]);
S.z
S.y
%Step 1: Setup
%write down parameters

a = 0.01; % m
eta = 10^-3; % Pa*
rho = 2260;
k1 = 1*10^-5;
k2 = 1*10^-6;


%declare symbols
%x(t): position of ball
% b: coefficient in front of first derivative
% c: coefficient in front of x(t)
syms x(t) b c k

%coeffiecent in front each derivative
b = 6*sym(pi)*a*eta/(rho*4/3*sym(pi)*a/3);
c = k/(rho*4/3*sym(pi)*a^3);

% Define first derivative of x
Dx = diff(x,t);

eq = diff(Dx,t) + b*Dx +c*x == 0
%substitute in one vale
eq1 = subs(eq,k,k1)
sol1 = dsolve(eq1,[x(0) == 0.2, Dx(0) == 0])
% Solve differential equation
% IC: x(0) = 0.02
% IC: dx/dt(0) = 0 (the ball does not have any initial velocity
col = {'k','r'};
kval = [k1,k2];
for m = 1:2

    % Substitute in the variable
    eq1 = subs(eq,k,kval(m));

    % Solve the differential equation
    sol1 = dsolve(eq1, [x(0) == 0.02, Dx(0) == 0]);

    % Plot the solution over time
    fplot(sol1,[0,1000],col{m});
    hold on
end
```

```
hold off
legend('k = 10e-5', 'k = 10e-6')
ylim([-0.02, 0.02])
axis square
ylabel('x position')
xlabel('time')
clear

% STEP 1: Setup
% Write down paramters describing the system

a = 0.01; % m
eta = 10^-3; % Pa*s
rho = 2260;
k1 = 1*10^-5;
k2 = 1*10^-6;

% Coefficient in front of each derivative
b = 6*(pi)*a*eta/(rho*4/3*(pi)*a^3);
c = [k1,k2]/(rho*4/3*(pi)*a^3);

for m = 1:2
    [t,out] = ode45(@springProblem,[0,1000],[0.02,0],[],b,c(m));

    plot(t,out(:,1),'k-')
    hold on
end
hold off
close all

% 1) Vector representing positions of two particles
pos = [1,4];

% 2) Define particle identities.
% Particle 1 will be infected
% Particle 2 will initially be healthy

% 3) Produce visualization of each particle
g1 = axes;
axis image              % Setting this ensures that the x and y scales
 are equal on the resulting image
xlim([0,7])             % X limits
ylim([-2,2])            % Y limits
set(g1,'ytick',[])      % Turn off y tick marks
set(g1,'Box','on')
xlabel('Position')      % X label

% Plot both particles initial position
p1 = rectangle('Position',[pos(1),-0.5,1,1],'Curvature',
[1,1],'FaceColor','r');
p2 = rectangle('Position',[pos(2),-0.5,1,1],'Curvature',
[1,1],'FaceColor','w','LineWidth',2);
```

```
% Step through the simulation. In each step, update the particles
 positions
while abs(pos(1)-pos(2)) >= 1

    % A) Create vector determining particle step
    d_k = (rand(1,2)-0.5);

    % B) Update particle position
    pos = pos + d_k;

    % C) Ensure particle doesn't move out of bounds
    pos = max(min(pos,6),0);

    % If difference between particles is less than one diameter, an
    % infection has occurred
    if abs(pos(1)-pos(2)) < 1
        p2.FaceColor = 'r';
    end

    % D) Update visualization
    p1.Position(1) = pos(1);
    p2.Position(1) = pos(2);

    % E) Pause MATLAB
    pause(0.2)
end
% Initialize the figure/gui window and center
ss = get(0,'ScreenSize');
fig = figure('Visible','on','Position',ss.*.5);
movegui(fig,'center')

% Initialize the counter value
cVal = 0;

% Add an "edit text" box to show the current value of the counter
hedit = uicontrol('Style','edit','Units','Normalized','Position',...
    [0.3 0.05 0.4 0.2],'FontSize',30,'String',cVal);

% Pushbutton to subtract 1 from the current value of the counter
uicontrol('Style','pushbutton','Units','Normalized','Position',...
    [0.2 0.4 0.2 0.2],'String','Subtract 1','Callback',
{@subtractVal,cVal,hedit});

% Pushbutton to add 1 to the current value of the counter
uicontrol('Style','pushbutton','Units','Normalized','Position',...
    [0.6 0.4 0.2 0.2],'String','Add 1','Callback',
{@addVal,cVal,hedit});
close all
fishPic = imread('fish.png');
imshow(fishPic)
% A color image is a 3D matrix
size(fishPic)
close all
```

```
% Create figure and axes that are visible
ss = get(0,'ScreenSize');
h1 = figure('Visible','on','Position',ss.*.5);
g1 = axes;

% Create graphics handle object describing fish
hFish = imshow(fishPic)

% Define scaling factor
sf = 0.25;

% Change scaling of image on axes
hFish.XData = [1,sf*hFish.XData(2)];
hFish.YData = [1,sf*hFish.XData(2)];

% Ensure axes are visible
g1.Visible = 'on';

% Reduce figure window size
h1.Position(3:4) = 0.5*h1.Position(3:4);
% Move the object away from the origin
hFish.XData = hFish.XData + 100;
hFish.YData = hFish.YData + 100;
% Flip the orientation of the object
hFish.XData = fliplr(hFish.XData);
% Change the axes background to blue
g1.Color = [135,206,250]/255;
close all

% Read the fish picture and associated alpha data
[fishPic,~,alphaFish] = imread('fish.png');

subplot(1,2,1)
image(fishPic)
title('color information')

subplot(1,2,2)
image(alphaFish)
title('alpha data')
close all

% Create figure and axes that are visible
h1 = figure('Visible','on');
g1 = axes;

% Create graphics handle object describing fish
hFish(1) = imshow(fishPic);

% Assign alpha data
hFish(1).AlphaData = alphaFish;

% Define scaling factor
sf = 0.25;
```

```
% Change scaling of image on axes
hFish(1).XData = [1,sf*hFish(1).XData(2)]+randi(500);
hFish(1).YData = [1,sf*hFish(1).YData(2)]+randi(500);

% Ensure axes are visible
g1.Visible = 'on';

% Reduce figure window size
h1.Position(3:4) = 0.5*h1.Position(3:4);

% Change the axes background to blue
g1.Color = [135,206,250]/255;
close all
% Create five fish objects
for k = 1:5
    hold on
    % Create graphics handle object describing fish
    hFish(k) = imshow(fishPic);

    % Assign alpha data
    hFish(k).AlphaData = alphaFish;

    % Define scaling factor
    sf = 0.25;

    % Change scaling of image on axes
    hFish(k).XData = [1,sf*hFish(k).XData(2)]+randi(500);
    hFish(k).YData = [1,sf*hFish(k).YData(2)]+randi(500);

end
hold off
close all

% Read the fish picture and associated alpha data
[rockPic,~,alphaRock] = imread('Rock.png');

subplot(1,2,1)
image(rockPic)
title('color information')

subplot(1,2,2)
image(alphaRock)
title('alpha data')
close all

% Step 1: set up figure

% Create figure and axes that are visible
h1 = figure('Visible','on');
g1 = axes;

% Create three fish objects
for k = 1:3
    hold on
```

```
    % Create graphics handle object describing fish
    hFish(k) = imshow(fishPic);

    % Assign alpha data
    hFish(k).AlphaData = alphaFish;

    % Define scaling factor
    sf = 0.25;

    % Change scaling of image on axes
    hFish(k).XData = [1,sf*hFish(k).XData(2)]+randi(500);
    hFish(k).YData = [1,sf*hFish(k).YData(2)]+randi(500);

end
hold off

% Ensure axes are visible
g1.Visible = 'on';

% Reduce figure window size
h1.Position(3:4) = 0.5*h1.Position(3:4);

% Change the axes background to blue
g1.Color = [135,206,250]/255;
xlim([-200,1500])
ylim([-200,1000])

% Step 2: loop through 50 simulation time steps
for m = 1:25
    % For each fish, update position
    for k = 1:3
        % Change position of image on axes
        hFish(k).XData = hFish(k).XData + randi(100) - 50;
        hFish(k).YData = hFish(k).YData + randi(100) - 50;
    end
    pause(0.1)
end
close all

% Create figure
% Create figure and axes that are visible
h1 = figure('Visible','on');
g1 = axes;

% Create data structure to store info about interface
handles = struct;
% Create field to store the current xy direction of fish
handles.moveDir = "left";
% Store the data structure as GUI data
guidata(h1,handles);

% Create fish object
% Create graphics handle object describing fish
hFish = imshow(fishPic);
```

```
% Assign alpha data
hFish.AlphaData = alphaFish;
% Define scaling factor
sf = 0.25;
% Change scaling of image on axes
hFish.XData = [1,sf*hFish.XData(2)]+randi(500);
hFish.YData = [1,sf*hFish.YData(2)]+randi(500);

% Specify additional graphical parameters
% Ensure axes are visible
g1.Visible = 'on';

% Reduce figure window size
h1.Position(3:4) = 0.5*h1.Position(3:4);

% Change the axes background to blue
g1.Color = [135,206,250]/255;
xlim([-200,1500])
ylim([-200,1000])
% Assign key press function
h1.KeyPressFcn = {@Topic25DoWhenKeyIsPressed,hFish};
% Set up the timer object t.
t = timer('TimerFcn', @sayIt, 'StartFcn', @startItUp, 'StopFcn',
 @stopIt, 'ExecutionMode', 'fixedRate', 'StartDelay', 2, 'Period',
 3, 'TasksToExecute', 5);



% start timer
start(t)
close all

% Create figure
% Create figure and axes that are visible
h2 = figure('Visible','on');
g2 = axes;

% Create rock object using the previously loaded rock
% Create graphics handle object describing rock
hRock = imshow(rockPic);
hRock.Visible = 'off';
% Assign alpha daya
hRock.AlphaData = alphaRock;
% Store XY dimmensions of rock
hRockSize = [hRock.XData(2), hRock.YData(2)];

% Ensure axes are visible
g2.Visible = 'on';

% Change the axes background to blue
g2.Color = [135,206,250]/255;
xlim([-200,1500])
ylim([-200,1000])
h2.Position = [672,238,941,698];
```

```
close all

% Create figure
% Create figure and axes that are visible
h2 = figure('Visible','on');
g2 = axes;

% Physical parameters
rho_g = 2670;
rho_w = 1000;
eta = 10^-3;
a = 150;
g = 9.81;

% Constants
b = 6*eta/(rho_g*4/3*a^2);
c = (rho_g - rho_w)*g/rho_g;
% Estimate time to stop ODE calculation
t_end = 5*sqrt(1000/c);

%Solve ODE
[t,out] = ode45(@rockODE, 0:0.05:t_end, [0,0], [],b,c);
out = -out;
y = out(out(:,1)<600);
t = t(out(:,1)<600);

% Create rock object using the previously loaded rock
% Create graphics handle object describing rock
hRock = imshow(rockPic);
hRock.Visible = 'off';

% Assign alpha data
hRock.AlphaData = alphaRock;
% Store XY dimmensions of rock
hRockSize = [hRock.XData(2),hRock.YData(2)];

% Ensure axes are visible
g2.Visible = 'on';

% Change the axes background to blue
g2.Color = [135, 206, 250]/255;
xlim([-200,1500])
ylim([-200,500])
h2.Position = [672,238,941,698];
close all

% Establish the figure and the button down/up and motion
% callbacks.
h1 = figure('Visible','on');

% The diameter of the markers.
D = 1;
```

```
% 'ind' will be used to track the index of the marker that is to
% be moved when a mouse movement is detected. An empty set
% indicates that no index has been selected.
handles.ind = [];

% Store guidata
guidata(h1,handles);

% Initialize the center locations of 10 markers.
x = (10-D)*rand(1,10)+D/2;
y = (10-D)*rand(1,10)+D/2;

% Plot the markers.
p = plot(x,y,'ko');

% Adjust the axis scaling so that the radius of the markers is
% scaled to the size of the axes.
axis equal
axis([0 10 0 10])
ax = gca;
ax.Units = 'points';
p.MarkerSize = D/diff(xlim)*ax.Position(3);
% Declare callback functions
h1.WindowButtonDownFcn = {@pick,D,p};
h1.WindowButtonMotionFcn = {@move,p};
h1.WindowButtonUpFcn = @place;
function c = Pythagorean(a,b)
    % Compute c for the relation
    % a^2+b^2=c^2.
    % a, b, and c are scalars.

    % Compute c.
    c = sqrt(a^2+b^2);
end
function [p,s] = myFunction(a,b,c)
    % p and s are computed within the function
    p = a+b;
    s = b*c;
end
function out = mySquareRoot(a)
    % Returns the square root of a scalar,
    % vector, or multidimensional array
    % input, a, to out
    % M. Siedlik
    % 9/6/2020

    % Calculates the square root
    % of the inputout = sqrt(a);
    out = sqrt(a);
end
function out = returnOddElements(v)
    % Select the odd indices of the input vector
    out = v(1:2:end);
```

```
end
function F = eqnSet(x)
    % System of two non linear equations
    % F(1) and F(2): "equation" 1 and 2, respectively
    % x(1) and x(2): variable 1 and 2, respectively
    F(1) = x(2)-2*x(1)^2+3;
    F(2) = x(2)^2-x(1);
end
function [n, diceVal] = diceRollSimulator(max_n, n_sides)
%H1 help line, running a simulation until any number comes up max_n
 times
%in a row

%Simpler: roll dice until 3 1's land in a row

%roll one dice
n = 0;
numInARow = 0;
%previous value not 1-6
preDiceVal  = 0;

%evaluate dice face value
% If dice face = 1 it is good unless restart
%continue to iterate until true
    while numInARow < max_n
        %have a variable to hold the pervious dice value

        diceVal = randi(6);
        if diceVal == preDiceVal
            numInARow = numInARow + 1;
        else
            numInARow = 1;
            preDiceVal = diceVal;
        end
        %counter
        n = n+1;
    end
end



function out = checkEndGame(gameboard,player)
    %check to see if any player has one
    out = 1; %check if game is still going
    %checks the first column
    if prod(gameboard(1:3) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard(4:6) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard(7:9) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
```

```
    elseif prod(gameboard([1,4,7]) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard([2,5,8]) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard([3,6,9]) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard([1,5,9]) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    elseif prod(gameboard([3,5,7]) == player)
        disp(strcat(num2str(player), "wins"))
        out = 0;
    end

end

function res1 = square_add1(a,b)
    % Example function. In this case there are two inputs and one
    % output.
    res1 = a^2 + b^2 + 1;
end
function res2 = operateOnKnownValues(fxn)
    % Example function: input 1 is another function
    res2 = fxn(1,2);

end
function res3 = exampleA(fxn,A)
    % Example function: input 1 is a function, input 2 is an array
    res3 = fxn(A) + 1;

end
function res4 = exampleB(fxn,A)
    % Example function: input 1 is a function, input 2 is an array
    res4 = fxn(A) - 1;

end
function force = springForce1(delta)
    % Calculate the force required to cause a deformation of delta
    % Spring constant = 0.5
    force = 0.5*delta;
end
function force = springForce2(delta)
    % Calculate the force required to cause a deformation of delta
    % in a nonlinear spring
    force = .2*delta^2 + .5*delta;
end
function force = parallelSprings(spring1,spring2,delta)
    % Force for two springs loaded in parallel config.
    % spring1 = handle to force/deflection eqn for spring 1
    % spring2 = handle to force/deflection eqn for spring 2
    % delta = displacement of springs
```

```
    force = spring1(delta) + spring2(delta);
end

function fxn1(f,x)
    % Example function
    output = f(x);
    disp(strcat('The output of this function is...
 ',num2str(output)));
end
% % Script for using the Newton-Raphson method to solve for f(x)
% function xn = newtonSolve(F, xn, tol)
% % Step 1: specify initial guess
%     xn = 5; % initial guess
%     xprev = inf; % Variable to store previous xn value -> ensure it
 is
%                 % not xn to enter while loop
%
%     % Step 2: iterate until xn -> xn+1
%     while abs(xn - xprev) > 10^-5
%
%         % Step 3: evaluate function at xn
%         f = F(xn);
%         % Step 3a: evaluate derivative at xn
%         f_prime_n = 4*xn^3 + 4*(xn-2);
%
%         % Instead of calculating the "next" x value, store the
 previous xn
%         % value
%         xprev = xn;
%
%         % Step 3b: update xn to next value for the iteration
%         xn = xn - f(1)/f(2);
%     end
%     disp(xn)
%
% end
function F = originalEqn(x)
    %function
    F(1) = x^4 + 2*(x-2)^2 - 20;
    %derivative of function
    F(2) = 4*x^3 + 4*(x-2);
end

function G = newEqn(x)
    % Function of interest
    G(1) = x + exp(-x) - 4;
    % Derivative of function of interest
    G(2) = 1 - exp(-x);
end

function out = recursiveAdd(in)
    % Adds a number to the input up to
    % a value of 5
```

```
    if in<5
        in = in + 1;
        out = recursiveAdd(in);
    else
        out = in;
    end
end
function x = recursiveFib(x,maxVal)
    % Function using recursion to create a Fibonacci sequence vector

    % If the last element is less that maxVal, add the next element of
 the
    % Fibonacci sequence to the vector
    if  x(end-1) + x(end) < maxVal
        % Calculate next entry
        x(end+1) = x(end)+x(end-1);
        % Use recursion to repeat the process
        x = recursiveFib(x,maxVal);
    end
end
function y = fact(n)
    % Recursive function for determining the factorial

    % Specify the number we are interested in solving the factorial
 for
    y = n;

    % Specify the "lower limit" in the factorial calculation
    if n == 0
        y = 1;
    else
        % We multiply by all the integers before the input,
        % one at a time...
        y = y*fact(n-1);
    end
end
function pn = legendreP(x,n)
    %calculate the n-th vector legendre polynomial
    %vector 1: vector of positions
    %vector 2: order of legendre polynomial

    %n = 0 -> pn should be n for all x
    if n == 0
        %make a vector full of ones based on the size of x
        pn = ones(size(x));

    % if n = 1 -> pn = x
    elseif n == 1
        pn = x;

    %pn depends on previous values
    else
```

```
        pn = 1/n*((2*n-1)*x.*legendreP(x,n-1) -
 (n-1)*legendreP(x,n-2));
    end

end
function xn = newtonSolve(F,xn,tol)
    % The purpose of this function is to apply the Newton-Raphson
 method to
    % solve any arbitrary function, specified in an input F.
    % Inputs:
    % F: function and derivative
    % x: initial guess
    % tol: specified tolerance

    % Step 1: specify initial guess
    % Input xn -> initial guess
    xprev = inf; % Variable to store previous xn value -> ensure it is
               % not xn to enter while loop

    % Step 2: iterate until xn -> xn+1
    while abs(xn - xprev) > tol
        % Apply input function to algorithm. NOTE: first output is
 function
        % evaluated at xn, the second output is the derivative of the
        % function evaluated at xn
        f = F(xn);

        % Instead of calculating the "next" x value, store the
 previous xn
        % value
        xprev = xn;

        % Step 3b: update xn to next value for the iteration
        xn = xn - f(1)/f(2);
    end
end

function xn = newtonSolveReturnXvalues(F,xn,tol)
    % The purpose of this function is to apply the Newton-Raphson
 method to
    % solve any arbitrary function, specified in an input F.
    % Inputs:
    % F: function and derivative
    % x: initial guess
    % tol: specified tolerance

    % Step 1: specify initial guess
    % Input xn -> initial guess
    xprev = inf; % Variable to store previous xn value -> ensure it is
               % not xn to enter while loop

    % Step 2: iterate until xn -> xn+1
    while abs(xn(end) - xprev) > tol
```

```
        % Apply input function to algorithm. NOTE: first output is
 function
        % evaluated at xn, the second output is the derivative of the
        % function evaluated at xn
        f = F(xn(end));

        % Instead of calculating the "next" x value, store the
 previous xn
        % value
        xprev = xn(end);

        % Step 3b: update xn to next value for the iteration
        xn(end+1) = xn(end) - f(1)/f(2);
    end

end
function G = quiz3(t,h)
    % These two equations represent a function and its derivative that
 can
    % be solved by
    % G(1): function
    % G(2): derivative of function

    G(1) = 50*(1./(t+1) + 5*(t+1).^(-4*h)) + 300 - 400;
    G(2) = 50*(-1*(t+1).^-2 + 5*(-4*h)*(t+1).^(-4*h-1));
end

function FctnToCall(~,eventdata)
    if strcmp(eventdata.Key,'w')
        disp('Pressed a w!')
    end
end

function DoWhenKeyIsPressed(~,eventdata,hBall)
    % Utilizes keypresses to move the x/y coordinates of the
    % ball or closes the figure.

    % Movement increment
    inc = 0.3;

    % Move up
    if strcmp(eventdata.Key,'uparrow')
        hBall.Position(2) = hBall.Position(2)+inc;
    % Move down
    elseif strcmp(eventdata.Key,'downarrow')
        hBall.Position(2) = hBall.Position(2)-inc;
    % Move left
    elseif strcmp(eventdata.Key,'leftarrow')
        hBall.Position(1) = hBall.Position(1)-inc;
    % Move right
    elseif strcmp(eventdata.Key,'rightarrow')
        hBall.Position(1) = hBall.Position(1)+inc;
    % Close figure
    elseif strcmp(eventdata.Key,'q')
```

```
        close()
    % Display invalid keystroke
    else
        disp('Key not mapped')
    end
end
function numSteps = simulateDisease(moveModifier)
    % Inputs:
    % [0,1]: particle 1 is in quarantine
    % [1,1]: particles can move freely
    % [1,0]: particle 2 is in quarantine

    % 1) vector representing posititions of two particles
    pos = [1,4];

    % 2) Define particle identities
    % Particle 1 will be infected
    % Particle 2 will initially be healthy

    % Define counter for steps
    numSteps = 0;

    % Step through the simulation. in each step update the particles
    % positions
    while abs(pos(1) - pos(2)) >=1

        % A) Create vector determining particle step
        d_k = (rand(1,2)-0.5).*moveModifier;

        % B) Update particle position
        pos = pos + d_k;

        % C) Ensure particle doesn't move out of bounds
        pos = max(min(pos,6),0);

        % D) Update counter
        numSteps = numSteps + 1;
    end
end

function Topic21DoWhenKeyIsPressed(~,eventdata,hBall)
    % Utilizes keypresses to move the x/y coordinates of the
    % ball or closes the figure.

    % Movement increment
    inc = 0.3;

    % Move up
    if strcmp(eventdata.Key,'uparrow')
        hBall.Position(2) = min(hBall.Position(2)+inc,9);

    % Move down
    elseif strcmp(eventdata.Key,'downarrow')
        hBall.Position(2) = max(hBall.Position(2)-inc,-10);
```

```
    % Move left
    elseif strcmp(eventdata.Key,'leftarrow')
        hBall.Position(1) = max(hBall.Position(1)-inc,-10);
    % Move right
    elseif strcmp(eventdata.Key,'rightarrow')
        hBall.Position(1) = min(hBall.Position(1)+inc,9);
    % Close figure
    elseif strcmp(eventdata.Key,'q')
        close()
    % Display invalid keystroke
    else
        disp('Key not mapped')
    end
end
function ex3_keyPress(~,eventdata,hBall)
    % Utilizes keypresses to move the x/y coordinates of the
    % ball or closes the figure.

    % Movement increment
    inc = 0.3;

    % Reset FaceColor to be black
    hBall.FaceColor = 'k';

    % Move up
    if strcmp(eventdata.Key,'uparrow')
        hBall.Position(2) = min(hBall.Position(2)+inc,9);

    % Move down
    elseif strcmp(eventdata.Key,'downarrow')
        hBall.Position(2) = max(hBall.Position(2)-inc,-10);
    % Move left
    elseif strcmp(eventdata.Key,'leftarrow')
        hBall.Position(1) = max(hBall.Position(1)-inc,-10);
    % Move right
    elseif strcmp(eventdata.Key,'rightarrow')
        hBall.Position(1) = min(hBall.Position(1)+inc,9);
    % Close figure
    elseif strcmp(eventdata.Key,'q')
        close()
    % Display invalid keystroke
    else
        disp('Key not mapped')
    end
end
function ex3_keyRelease(~,~,hBall)
    % Function to be executed when key is released. Takes a graphics
 object
    % corresponding to the ball as an input

    % Increase the whiteness of the ball until it becomes fully white
    while hBall.FaceColor(1) < 0.9

        % Make FaceColor whiter
```

```
        hBall.FaceColor = min(hBall.FaceColor + 0.1,1);

        % Pause for 100 ms
        pause(0.1)
    end

    % If ~1 second passes without moving, game over
    t1 = text(-10,0,'GAME OVER');
    t1.FontSize = 50;
    t1.Color = 'r';

end
function moveText(~,~,g)
    % The mouse button motion callback.

    % Get the current mouse location.
    pos = get(gca, 'CurrentPoint');
    x = pos(1,1); y = pos(1,2);

    % Update the text object readout to the current location
    % and place its location at the cursor tip.
    g.String = ['x = ',num2str(x),', y = ',num2str(y)];
    g.Position = [x, y, 0]; % position 0 in 3rd dimension
end
function click(h,~)
    % The callback for a mouse button press. Print the mouse
    % button identifier to the command window.
    h.SelectionType
end

function plotPoints1(~,~,h)
    % Call back function for updating a plot

    % Get the current mouse location
    pos = get(gca,'CurrentPoint');

    % Update the plot with the new position
    h.XData(end+1) = pos(1,1);
    h.YData(end+1) = pos(1,2);
    drawnow
end
function plotPoints2(h,~,hRed,hGreen)
    % Callback function for updating a plot with multiple colored
 points

    % Get the current mouse location.
    pos = get(gca, 'CurrentPoint');

    % If a left click: add to the red plot
    if strcmp(h.SelectionType,'normal')

        % Add to the data in the red plot
        hRed.XData(end+1) = pos(1,1);
        hRed.YData(end+1) = pos(1,2);
```

```
        drawnow

    % If a right click: add to the green plot
    elseif strcmp(h.SelectionType,'alt')

        % Add to the data in the red plot
        hGreen.XData(end+1) = pos(1,1);
        hGreen.YData(end+1) = pos(1,2);
        drawnow

    end
end
function plotPoints3(h,~,hRed,hGreen)
    % Callback function for updating a plot with multiple colored
 points

    % Get the current mouse location.
    pos = get(gca, 'CurrentPoint');

    % Load GUI information
    handles = guidata(h);

    % If a left click: add to the red plot
    if strcmp(h.SelectionType,'normal')

        % Add to the data in the red plot
        hRed.XData(end+1) = pos(1,1);
        hRed.YData(end+1) = pos(1,2);
        drawnow

        % Specify that the most recent point was red
        handles.order{end+1} = 'r';

    % If a right click: add to the green plot
    elseif strcmp(h.SelectionType,'alt')

        % Add to the data in the red plot
        hGreen.XData(end+1) = pos(1,1);
        hGreen.YData(end+1) = pos(1,2);
        drawnow

        % Specify that the most recent point was red
        handles.order{end+1} = 'g';

    % If a middle click (and note empty: delete the most recent point
    elseif strcmp(h.SelectionType,'extend') && ~isempty(handles.order)

        % If last point was red
        if strcmp(handles.order{end},'r')
            % Remove last data poitn
            hRed.XData(end) = [];
            hRed.YData(end) = [];
            drawnow
```

```
            % Delete last entry
            handles.order(end) = [];

        % If last point was green
        else
            % Remove last data poitn
            hGreen.XData(end) = [];
            hGreen.YData(end) = [];
            drawnow

            % Delete last entry
            handles.order(end) = [];

        end
    end

    % Update the guidata with the data structure
    guidata(h,handles);
end
function ticTacToe(compStrategy)
    % Define initial variables and arrays
    gameboard = zeros(3,3); % gameboard that we will populate

    % Repeat: alternate for players
    player = 1; % player one

    % Logical variable to determine if game should stil be playing
    gameStillGoing = 1;

    % Iterate until someone has won (or the game is a draw)
    while gameStillGoing

        % Player 1: human player. If it is player 1's turn, specify a
        % position to be played
        if player == 1

            % Logical variable corresponding if the move is valid
            invalidMove = 1;

            % Make player pick a valid position
            while invalidMove

                % Position to play, in single index notation
                playPos = input('Select a position to play...');

                if ~isempty(playPos)
                    % Determine if move is allowable
                    invalidMove = all((gameboard(:) == 0)'.*(1:9) ~=
 playPos);
                end
            end

        % Player 2: computer player. If it is player 2's turn,
 determine the
```

```
        % position played based on the algorithm contained within
 compStrategy().
        else
            % Call a function that will output a position based upon
 the
            % strategy defined for the computer
            playPos = compStrategy(gameboard);
        end

        % Update the gameboard
        gameboard(playPos) = player;

        % Step 3: display game board
        disp(gameboard)

        % Step 4: evaluate stopping condition -> did someone win?
        gameStillGoing = checkEndGame2(gameboard, player);

        % Step 5: change player
        player = mod(player,2)+1;
    end
end
function out = checkEndGame2(gameboard,pl)
    % The purpose of this function is to determine if anyone has won
 the
    % game or if a draw has occured.
    % Input 1, gameboard: matrix containing moves
    % Input 2, which player's turn is it

    % Default condition: game needs to keep going
    out = 1;

    % Each row represents a different way in which the player could
 win
    testMatrix =
 [gameboard;gameboard';diag(gameboard)';diag(rot90(gameboard))'];

    if any(all(testMatrix==pl,2))
        disp(strcat(num2str(pl),' wins!'))
        out = 0;
    elseif sum(gameboard(:)>0) == 9
        disp('Draw!')
        out = 0;
    end
end
function posPlayed = playRandom(gameboard)
    % This function contains code describing the computer player's
 strategy
    % Note: Students should optimize this and ensure it is well
 commented

    % Determine available positions to play
    availablePos = (gameboard(:) == 0)'.*(1:9);
    positions = availablePos(availablePos > 0);
```

```
    % Return a random (available) position. This code thus has the
 computer
    % play randomly based upon the available positions
    posPlayed = positions(randi(length(positions)));
end
function placeX(h1,~,x1)
    % The purpose of this function is assign an X once a player has
 clicked
    % an appropriate location

    % Retrieve current X and Y coordinates
    pos = get(gca,'CurrentPoint');
    pos = pos(1,1:2);

    % Convert the mouse click location into one of 9 possible game
 play
    % positions. NOTE: this is based on the visual gameboard created
 in the
    % above script

    % If click corresponds to index position 3
    if pos(1)<-1 && pos(2)<-1
        playPos = 3;
        % Update position vector. This is what will be assigned to the
        % text that will be added to the figure panel
        pos = [-2.5,-2];

    % If click corresponds to index position 2
    elseif pos(1)<-1 && pos(2)<1
        playPos = 2;
        pos = [-2.5,0];

    % If click corresponds to index position 1
    elseif pos(1)<-1
        playPos = 1;
        pos = [-2.5,2];

    % If click corresponds to index position 6
    elseif pos(1)<1 && pos(2)<-1
        playPos = 6;
        pos = [-0.5,-2];

    % If click corresponds to index position 5
    elseif pos(1)<1 && pos(2)<1
        playPos = 5;
        pos = [-0.5,0];

    % If click corresponds to index position 4
    elseif pos(1)<1
        playPos = 4;
        pos = [-0.5,2];

    % If click corresponds to index position 9
```

```
    elseif pos(2)<-1
        playPos = 9;
        pos = [1.5,-2];

    % If click corresponds to index position 8
    elseif pos(2)<1
        playPos = 8;
        pos = [1.5,0];

    % If click corresponds to index position 7
    else
        playPos = 7;
        pos = [1.5,2];

    end


    % Load gameboard information
    handles = guidata(h1);

    % Update internal gameboard. Note: the click corresponds to player
 1
    handles.gameboard(playPos) = 1;

    % Evaluate stopping condition -> did someone win?
    % NOTE: no output is necessary
    checkEndGame3(handles.gameboard,1,h1);

    % Create a matrix containing positional information for all text
 that
    % has been played
    textPos = reshape([x1.Position],3,5)';

    % Use the above matrix to determine the turn number from the non
 NaN values
    numTurn = 6-sum(isnan(textPos(:,1)));

    % Update the plot with the new position
    x1(numTurn).Position(1:2) = pos;

    % Save data
    guidata(h1,handles);

end

function placeO(h1,~,x2,compStrategy)
    % The purpose of this function is assign an O once a player has
 clicked
    % an appropriate location

    % Load gameboard information
    handles = guidata(h1);

    % Determine position to play based on input strategy
```

```
    playPos = compStrategy(handles.gameboard);

    % Update internal gameboard
    handles.gameboard(playPos) = 2;

    % Determine where to place text
    % Note: this is determined entirely by figure that was created
    if playPos == 1
        pos = [-2.5,2];
    elseif playPos == 2
        pos = [-2.5,0];
    elseif playPos == 3
        pos = [-2.5,-2];
    elseif playPos == 4
        pos = [-0.5,2];
    elseif playPos == 5
        pos = [-0.5,0];
    elseif playPos == 6
        pos = [-0.5,-2];
    elseif playPos == 7
        pos = [1.5,2];
    elseif playPos == 8
        pos = [1.5,0];
    else
        pos = [1.5,-2];
    end

    % Create a matrix containing positional information for all text
 that
    % has been played
    textPos = reshape([x2.Position],3,4)';

    % Use the above matrix to determine the turn number from the non
 NaN values
    numTurn = 5-sum(isnan(textPos(:,1)));

    % Update the plot with the new position
    x2(numTurn).Position(1:2) = pos;

    % Step 4: evaluate stopping condition -> did someone win?
    checkEndGame3(handles.gameboard,2,h1);

    % Save data
    guidata(h1,handles);
end
function checkEndGame3(gameboard,pl,h1)
    % The purpose of this function is to determine if anyone has won
 the
    % game or if a draw has occured.
    % Input 1, gameboard: matrix containing moves
    % Input 2, which player's turn is it
    % Input 3, graphics object handle corresponding to figure window
```

```
    % Each row represents a different way in which the player could
 win
    testMatrix =
[gameboard;gameboard';diag(gameboard)';diag(rot90(gameboard))'];

    % If any three in a rows have happened
    if any(all(testMatrix==pl,2))
        % Remove click press and release function handles
        h1.WindowButtonDownFcn = '';
        h1.WindowButtonUpFcn = '';
        % Add text saying that the player won
        x3 = text(-5,5,['Player ',num2str(pl),' wins!']);
        x3.FontSize = 50;
        x3.Color = 'r';


    % If nine turns have occured, then the game is a draw
    elseif sum(gameboard(:)>0) == 9
        % Remove click press and release function handles
        h1.WindowButtonDownFcn = '';
        h1.WindowButtonUpFcn = '';
        % Add text saying that the player won
        x3 = text(-2,5,['Draw!']);
        x3.FontSize = 50;
        x3.Color = 'r';
    end
end
function resetGame(~,~,h,x1,x2,f1,f2,f3)
    % To be assign to some pushbutton for resetting the game

    % Create data structure to store gameboard
    handles.gameboard = zeros(3);
    % Store data structure
    guidata(h,handles);

    % Create two empty text containers. On every turn, we will update
 the the
    % position of either the next X or next O marker from NaN to the
    % appropriate coordinates
    for k = 1:4
        x1(k).Position(1:2) = nan(1,2);
        x2(k).Position(1:2) = nan(1,2);
    end
    x1(5).Position(1:2) = nan(1,2);

    % Assign callback function for mouse click
    h.WindowButtonDownFcn = {f1,x1};
    % Assign callback function for mouse release
    h.WindowButtonUpFcn = {f2,x2,f3};


end
function dxdt = myODE(t,x)
    dxdt = x(1) - t^2 + 1;
```

```
end
function dxdt = rxnEqns(t,x)
    % Here, x(1) : A
    % x(2): B
    % x(3): C
    dxdt = zeros(3,1);
    dxdt(1) = -x(1) + 2*x(2);
    dxdt(2) = x(1) - 4*x(2) + 0.5*x(3);
    dxdt(3) = 2*x(2) - 0.5*x(3);

end
function dxdt = thirdOrderODE(t,x)
    % Parameters
    p = 2;
    q = 1.5;
    r = 0.1;

    % Note:
    % x(1) corresponds to y
    % x(2) corresponds to y'
    % x(3) corresponds to y''

    % Preallocating the output as a column vector
    dxdt = zeros(3,1);

    dxdt(1) = x(2);
    dxdt(2) = x(3);
    dxdt(3) = -p*x(3)-q*x(2)-r*x(1);
end
function dxdt = thirdOrderODE_extraInputs(t,x,p,q,r)
    dxdt = zeros(3,1);
    dxdt(1) = x(2);
    dxdt(2) = x(3);
    dxdt(3) = -p*x(3)-q*x(2)-r*x(1);
end
function dz = ball_2D(t,z,vx)
    % Compute the x and y position of a ball subject to gravity and
    % in a vacuum. This function is called upon by MATLAB's ODE
    % solvers. t = time (s), z = [y-position, y-velocity, x-position]
    % in units of (m, m/s, m), and dz = [y-velocity; y-acceleration;
    % x-position] in units of (m/s, m/s^2, m).

    dz(1,1) = z(2); % z(1): y-position
    dz(2,1) = -9.81;% z(2): y-velocity
    dz(3,1) = vx;   % z(3): x-position
end
function determineStart(hObj,~,hBall,hTrajectory,hAim)
    % Mouse click: determine kinematics parameters

    % Load information associated with a data structure
    handles = guidata(hObj);

    % Get the current mouse location
    pos = get(gca,'CurrentPoint');
```

```
    pos = pos(1,1:2);

    % Specify the inital x and y positions
    hBall.XData = 0;
    hBall.YData = handles.h0;
    hTrajectory.XData = 0;
    hTrajectory.YData = handles.h0;

    % Calculate difference in x and y values
    dy = abs(pos(2) - handles.h0);
    dx = abs(pos(1) - 0);
    % Calculate angle to throw
    handles.th = atan2d(dy,dx);

    % Define initial velocity to be proportional to distance away
 clicked
    handles.v = 3*sqrt(dy^2 + dx^2);

    % Create a line showing aim
    hAim.XData(2) = pos(1);
    hAim.YData(2) = pos(2);

    % Store object data
    guidata(hObj,handles);

end
function throwBall(hObj,~,hBall,hTrajectory)
    % Mouse release: calculate and visualize trajectory

    % Load GUI data
    handles = guidata(hObj);
    th = handles.th;
    h0 = handles.h0;
    v = handles.v;

    % Calculate parameters
    vx = v*cosd(th);
    vy = v*sind(th);
    g = 9.81;
    t_end = 2*2*vy/g;

    % Solve system of differential equations. Specify the time points
 to solve
    % for
    [t,state] = ode45(@ball_2D,[0:0.05:t_end],[2,vy,0],[],vx);
    txy = [t,state(:,3),state(:,1)];
    % Select time points corresponding to the ball not yet landing
    txy = txy(txy(:,3) > 0,:);


    % Loop through all ball positions and update plot
    for k = 2:size(txy,1)

        % Update plot of ball and trajectory tracer
```

```
        hBall.XData = txy(k,2);
        hBall.YData = txy(k,3);
        hTrajectory.XData(k) = txy(k,2);
        hTrajectory.YData(k) = txy(k,3);

        % Pause for specified amount of time consisted with time
 points solved
        % for
        pause(0.05);

    end

end
% function F = eqnSet(x)
%     % System of two non linear equations
%     % F(1) and F(2): "equation" 1 and 2, respectively
%     % x(1) and x(2): variable 1 and 2, respectively
%     F(1) = x(2)-2*x(1)^2+3;
%     F(2) = x(2)^2-x(1);
% end
function dZdt = springProblem(t,z,b,c)

    dZdt = zeros(2,1);
    dZdt(1) = z(2); % z(1) = x
    dZdt(2) = -b*z(2) - c*z(1); % z(2) = dx/dt

end

function subtractVal(hObj,~,hedit,h1)
    % Subtract 1 from the counter value
    handles = guidata(h1);
    cVal = handles.cVal - 1;

    set(hedit,'String',cVal)
    guidata(handles,h1);
end
function addVal(hObj,~,hedit,h1)
    % Add 1 to the counter value
    handles = guidata(h1);
    cVal = handles.cVal + 1;

    set(hedit,'String',cVal)
    guidata(handles,h1);
end
function Topic25DoWhenKeyIsPressed(h1,eventdata,hFish)
    % Utilizes keypresses to move the x/y coordinates of the
    % ball or closes the figure.

    % Load GUI data
    % Create field to store the current xy direction of fish
    handles = guidata(h1);

    % Movement increment
    inc = 10;
```

```
    % Move up
    if strcmp(eventdata.Key,'uparrow')
        hFish.YData = hFish.YData - inc;

    % Move down
    elseif strcmp(eventdata.Key,'downarrow')
        hFish.YData = hFish.YData + inc;

    % Move left
    elseif strcmp(eventdata.Key,'leftarrow')
        % If the current direction is set to move left
        if handles.moveDir == "left"
            hFish.XData = hFish.XData - inc;
        % Otherwise, flip the orientation of the fish
        else
            hFish.XData = fliplr(hFish.XData) - inc;
            handles.moveDir = "left";
        end

    % Move right
    elseif strcmp(eventdata.Key,'rightarrow')
        % If the current direction is set to move left
        if handles.moveDir == "right"
            hFish.XData = hFish.XData + inc;
        % Otherwise, flip the orientation of the fish
        else
            hFish.XData = fliplr(hFish.XData) + inc;
            handles.moveDir = "right";
        end

    % Close figure
    elseif strcmp(eventdata.Key,'q')
        close()
    % Display invalid keystroke
    else
        disp('Key not mapped')
    end

    % Store the data structure as GUI data
    guidata(h1,handles);
end
function startItUp(~,~)
    % Timer StartFcn callback.
    disp('A timer has been started.')
    disp('It will wait 2 seconds to fire the first time...')
    disp('and then it will fire every 3 seconds for 5 rounds!')
end
function sayIt(~,~)
    % Timer fxn callback.
    disp('Hello person!')
end
function stopIt(t,~)
    % Timer StopFcn callback.
```

```
    disp('The timer ''StopFcn'' just executed.')
    disp(['It utilizes the first input of the callback, '...
        'the handle of the timer, to stop the timer.'])
    disp('Always delete timers after you are done using them.')

    delete(t)
end
function dropRock(~,~,hRock,hRockSize)
    % Timer function that is called everytime the rock should be
 dropped

    % Reset the rock's initial position to be above the water line
    hRock.XData = [1,hRockSize(1)] + randi(1000);
    hRock.YData = [-hRockSize(2),-1];

    % Ensure rock is visible
    hRock.Visible = 'on';

    % Loop through time steps and move the rock down by some known
 amount
    % of time
    for k = 1:50
        hRock.YData = hRock.YData + 100;
        pause(0.1)
    end
end
function dYdt = rockODE(t,y,b,c)

    dYdt = zeros(2,1);
    dYdt(1) = y(2); % y(1) = x
    dYdt(2) = -b*y(2) - c; % z(2) = dy/dt

end
function dropRock_2(~,~,hRock,hRockSize,y_data)
    % Timer function that is called everytime the rock should be
 dropped

    % Reset the rock's initial position to be above the water line
    hRock.XData = [1,hRockSize(1)] + randi(1000);
    hRock.YData = [-hRockSize(2),-1];

    initialPos = hRock.YData;

    % Ensure rock is visible
    hRock.Visible = 'on';

    % Loop through time steps and move the rock down by some known
 amount
    % of time
    for k = 1:length(y_data)
        hRock.YData = initialPos + y_data(k);
        pause(0.05)
    end
end
```

```matlab
function pick(h1,~,D,p)
    % Mouse button down callback. Determines if the mouse was
    % clicked within the limits of a marker and, if so, takes
    % control of the marker.

    handles = guidata(h1);

    % Get the location of the mouse.
    pos = get(gca,'CurrentPoint');
    xMouse = pos(1,1);
    yMouse = pos(1,2);

    % Determine which marker centerpoint the mouse is closest
    % to and record the vector index number.
    [distance,minInd] = min(sqrt((xMouse-p.XData).^2+(yMouse-
p.YData).^2));

    % If the mouse is within a radius of the marker, then
    % take control of the x/y marker centerpoint index and
    % calculate the difference between the centerpoint of the
    % marker and where the user actually clicked.
    if distance <= D/2
        handles.ind = minInd;
    end

    guidata(h1,handles);

end
function move(h1,~,p)
    % Mouse motion callback. Moves the marker centerpoint
    % held in x/y index 'ind' if 'ind' is not an empty set.
    % Updates visuals accordingly.

    handles = guidata(h1);
    ind = handles.ind;

    % Get the location of the mouse.
    pos = get(gca,'CurrentPoint');
    xMouse = pos(1,1);
    yMouse = pos(1,2);

    % Update the x/y centerpoint corresponding to 'ind'.
    p.XData(ind) = xMouse;
    p.YData(ind) = yMouse;
    drawnow
end
function place(h1,~)
    % Mouse up callback. Release control of 'ind'.
    handles = guidata(h1);
    handles.ind = [];
    guidata(h1,handles);
end
```

hello world

```
y =

     5


x =

     7


y =

    10


y =

    16


ans =

    1.4142


result =

    1.4142

 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also ASIN, SIND, SINPI.

    Documentation for sin
       doc sin

    Other functions named sin

       sym/sin


a =

  -2.4493e-16


b =

    12


a =
```

```
     1     2     3     4     5

b =

          10         100        1000       10000      100000

c =

     []

d =

     1     1     1     1     1

f =

     0     0     0     0     0

y =

     5

ans =

     5     9     4

ans =

     3     4     8

ans =

     8

ans =

     4

ans =

     4     3     5
```

```
x =

     1     2     3

x =

     5     2     3

x =

     8     9     3

y =

     8     9     3

y =

     5     9     1

y =

     5     6     8

z =

     2     6     8

z =

     2     6     8     4

z =

     2     6     8     4     5     7

z =

     1     6     1     4     1     7

z =

     1     4     1     4     1     4
```

```
z =

     1     4     1     4     1     4     0     7

x =

     5     2     1     8

x =

     5     3     9     4     1     6

x =

     3     9     4     1     6

x =

     9     1

x =

     9     5     8     2

x =

     5     8     2

x =

     8

x =

     1     3     5     7

ans =

     1     7

x =

     1     3     5
```

```
x1 =

     1     3     5     1     3     5


x2 =

  Columns 1 through 7

    1.0000    3.0000    5.0000    1.0000    1.4000    1.8000    2.2000

  Columns 8 through 9

    2.6000    3.0000


x =

  Columns 1 through 13

     1     4    25     6    16    16     2    25     4     9     1
 4     5

  Columns 14 through 15

    36     3


x =

     5
     4
     6
     9


y2 =

     5     4     6     9


x =

     5
     2
     1
     8


x =

     9
```

```
     2
     1
     8


x =

     4
     3
     1
     8


x =

     4
     3
     9
     8


x =

     3
     8


x =

     5     2     1     4     9     1


L =

     6


ans =

     5     2     1


x =

    50    50    50     4     9     1


ans =

     1     2     3


ans =
```

```
    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000


ans =

     1     2     3


ans =

     1     2     3


ans =

    2.5000


ans =

    2.2500


ans =

    2.5000

     1

     1     5     9

     1
     5
     9

     1     5     9
     9     5     1


(:,:,1) =

     1     5     9
     9     5     1


(:,:,2) =

     2     3     4
     4     3     2


ans =

     3
```

```
ans =

     3


ans =

     3


ans =

     2     3


ans =

     2     3     2


ans =

     6


A =

     2     5     9
     6     8     3


A1 =

     1     2     3
     4     5     6


A2 =

     1     2     3     4     5     6


B1 =

     1
     2
     3
     4
     5
     6
```

```
a =

     1     2     3
     4     5     6


b =

     7     8     9
    10    11    12


C1 =

     1     2     3
     4     5     6
     7     8     9
    10    11    12


C2 =

     1     2     3     7     8     9
     4     5     6    10    11    12


A =

     1     4     7
     2     5     8
     3     6     9


ans =

     3


ans =

     7


B =

     1     2     3
     4     5     6


ans =

     1     4
     2     5
     3     6
```

```
A =

     1     4     7
     2     5     8
     3     6     9


ans =

     1
     2
     3


ans =

     2     5     8


ans =

     1     4
     2     5


ans =

     1     4     7
     2     5     8


ans =

     5


ans =

     1     2     3     4     5


ans =

     1
     2
     3
     4
     5
     6
     7
     8
     9
```

```
ans =

     1
     5
     9


A =

     1     4     7
     2     5     8
     3     6     9


A =

     1     4     7
     2     5     8
    50     6     9


A =

     0     0     7
     0     0     8
    50     6     9


A =

     0     0     7     0
     0     0     8     0
    50     6     9     0
     0     0     0    33


C(:,:,1) =

     0     0     0
     0     0     0
     0     0     0


C(:,:,2) =

     0     0     0
     0     0     0
     0     0     0


C(:,:,3) =
```

```
     0     0     0
     0     0     0
     0     0     1


A =

     1     5
     2     3


B =

     2     3
     1     5


A =

     1     5
     2     3


A =

     1     5
     2     3


B =

     2     3
     1     5


A =

     1     4
     2     3


B =

     1     2
     3     4


ans =

    13    18
    11    16


ans =
```

```
     5    10
    11    24


B =

     2     1
     5     3
     4     6


A =

     1     2     3


ans =

    24    25


A =

     1     2
     4     8


B =

    16     8
     4     2


A =

     2     4
     2     6


B =

     1     2
     1     3


A =

     2     4
     2     6


b =
```

```
     5
     4


ans =

    3.5000
   -0.5000


ans =

    3.5000
   -0.5000


A =

     2     4
     2     6


b =

     5     4


ans =

    5.5000   -3.0000


ans =

    5.5000   -3.0000


A =

     1     5
     2     3


B =

     2     3
     1     5


A =

     1     5
     2     3
```

```
ans =

     5    25
    10    15


A =

     1     5
     2     3


B =

     2     3
     1     5


ans =

     2    15
     2    15


ans =

    13    18
    11    16


ans =

     5    10
    11    24


B =

     2     1
     5     3
     4     6


A =

     1     2     3


ans =

    24    25
```

```
A =

     1     2
     4     8


B =

    16     8
     4     2


ans =

     2     2
     2     2


ans =

    0.5000    0.5000
    0.5000    0.5000


A =

     2     4
     2     6


b =

     5
     4


ans =

    3.5000
   -0.5000


ans =

    3.5000
   -0.5000


A =

     2     4
     2     6
```

```
b =

     5     4


ans =

    5.5000   -3.0000


ans =

    5.5000   -3.0000


A =

     1    -1
     1    -2


b =

     0
    -3


x =

   -1.5000
    1.5000
   -2.0000

    1.0000         0         0         0         0         0         0
         0         0    1.0000         0         0         0         0
         0         0         0    1.0000         0         0         0
    1.0000   -1.0000         0         0  -79.5775         0         0
         0    1.0000   -1.0000         0         0 -159.1549         0
         0    1.0000         0   -1.0000         0         0 -318.3099
         0         0         0         0    1.0000   -1.0000   -1.0000

   110
   100
   100
     0
     0
     0
     0

  110.0000
  105.7143
  100.0000
  100.0000
    0.0539
```

```
    0.0359
    0.0180


g1 =

     5


h1 =

    12


g2 =

     6


h2 =

     6


u =

  Columns 1 through 7

    1.0000    2.2857    3.5714    4.8571    6.1429    7.4286    8.7143

  Column 8

   10.0000


v_odd =

    1.0000    3.5714    6.1429    8.7143


Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.


sol_1 =

    1.0000   -1.0000


Equation solved.
```

```
fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.


sol_2 =

    1.4498    1.2041


a =

     5     3     2     6     2     1


b =

     1     3     5     7     9


c =

    1.0000    2.7500    4.5000    6.2500    8.0000

     0
     1
     1
     2
     3
     5
     8
    13
    21
    34


x =

     0     1     1     2     3     5     8    13    21    34

     1

     4

     9

    16

    25

    36

    49
```

```
    64

    81

   100

v =

     2

v =

     2     4

v =

     2     4     6

v =

     2     4     6     8

v =

     2     4     6     8    10

     2     4     6     8    10

v1 =

    10

v1 =

    10    -4

v1 =

    10    -4     8

v1 =

    10    -4     8     2

     5    -2     4     1
```

```
    10    -4     8     2


ans =

     4


v =

     1     2     3     4     1


v =

  Columns 1 through 7

    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000    3.2000

  Columns 8 through 11

    3.4000    3.6000    3.8000    4.0000


v =

  Columns 1 through 7

    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000    3.2000

  Columns 8 through 11

    3.4000    3.6000    3.8000    4.0000


v =

  Columns 1 through 7

    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000    3.2000

  Columns 8 through 11

    3.4000    3.6000    3.8000    4.0000


jj =

     1
     2
     3
     4
     5
```

```
jj =

     1


jj =

     2


jj =

     3


jj =

     4


jj =

     5


M =

     5     3
     1     9


jj =

     5
     1


jj =

     3
     9


jj =

     5


jj =

     1
```

```
jj =

     3

jj =

     9

     1     1     1     2     2     2     3     3     3

     1     2     3     1     2     3     1     2     3

     0     1     1     2     3     5     8    13    21    34

ans =

  logical

   1

ans =

  logical

   0

ans =

  logical

   1

s =

  logical

   1

b =

     2

c =

    10
```

```
ind =

     2


ind =

     3


ind =

     4


ind =

     5


N =

  Columns 1 through 6

           1           3           7          18          48
 127

  Columns 7 through 12

         336         886        2336        6158       16238
 42813

  Columns 13 through 18

      112884      297635      784760     2069138     5455595
 14384499

  Columns 19 through 20

    37926902   100000000


N =

  Columns 1 through 6

           1           3           7          18          48
 127

  Columns 7 through 12

         336         886        2336        6158       16238
 42813
```

```
  Columns 13 through 18

      112884       297635       784760      2069138      5455595
 14384499

  Columns 19 through 20

    37926902   100000000

     0     1     1     2     3     5     8    13    21    34

     0     1     1     2     3     5     8    13    21    34    55
 89


N =

  Columns 1 through 6

           1            3            7           18           48
 127

  Columns 7 through 12

         336          886         2336         6158        16238
 42813

  Columns 13 through 18

      112884       297635       784760      2069138      5455595
 14384499

  Columns 19 through 20

    37926902   100000000


N =

  Columns 1 through 6

           1            3            7           18           48
 127

  Columns 7 through 12

         336          886         2336         6158        16238
 42813

  Columns 13 through 18

      112884       297635       784760      2069138      5455595
 14384499
```

```
  Columns 19 through 20

    37926902   100000000

    25

     5

    24

res =

    90

    90

   0

   0

    95      0

    95      0

res1 =

  logical

   0

Alpha =

  logical

   1

var_2 =

  logical

   1

ind =

  logical

   1
```

```
ip =

  logical

   0


a =

     1     0     0     1


b =

     1     1     0     0


ans =

  1×4 logical array

   1   0   0   0


out =

  logical

   0

    90


res =

    90

    90

   1

    15    17
     5     2


ans =

     3


ans =

    5.0000
```

```
   13.1333

     1     2     3     4     5

    1.0000    1.7500    2.5000    3.2500    4.0000

     4     2     1
     5     4     2
     2     4     5


ans =

     1     2     5


ans =

  Columns 1 through 7

    3.2500    1.7500    1.7500    2.5000    4.0000    2.5000    1.0000

  Columns 8 through 10

    1.7500    3.2500    3.2500


ans =

     4     1
     2     5


ans =

    2.0000    3.7500    5.5000    7.2500    9.0000


ans =

    1.0000    3.5000    7.5000   13.0000   20.0000


ans =

    1.0000    1.7500    2.5000    3.2500    4.0000
    2.0000    3.5000    5.0000    6.5000    8.0000
    3.0000    5.2500    7.5000    9.7500   12.0000
    4.0000    7.0000   10.0000   13.0000   16.0000
    5.0000    8.7500   12.5000   16.2500   20.0000


ans =
```

```
     5     4     4

  Columns 1 through 7

    1.0000    1.4142    1.4142    2.0000    1.7321    1.7321    1.7321

  Columns 8 through 10

    2.4495    1.0000    1.7321

  Columns 1 through 13

    13    26    39    52    65    78    91   104   117   130   143
 156   169

  Columns 14 through 23

   182   195   208   221   234   247   260   273   286   299

     5     5


ans =

    15


Vt =

    0.0409


x_end =

   93.7013

    2.1141

    2.1141


v =

     5     3     4     6     8


p =

     5     8     6     3


q =
```

```
     5     3


r =

     4     6     8


s =

     8     6     4     3


t =

     5     4     8


M =

     5     2     4
     3     7     5


B =

     5     2
     3     7


C =

     5
     3


D =

     2     4
     7     5


F =

     2     4


G =

     5     7     3


ans =
```

```
     7     6
    10    12


ans =

    10     8
    21    35


ans =

    31    24    30
    36    55    47


A =

    0.7854    2.3562
    3.9270    5.4978


ans =

    1.0000   -1.0000
    1.0000   -1.0000


a =

     1     2     3


ans =

     6


A =

     1     2     3
     4     5     6


ans =

     5     7     9


ans =

     6
    15
```

```
ans =

    21


ans =

    21


a =

     1     2     3     4


ans =

     1     3     6    10


A =

     1     2     3
     4     5     6


ans =

     1     3     6
     4     9    15


a =

     1     2     3     4


ans =

    24


A =

     1     2     3
     4     5     6


ans =

     6
   120
```

```
a =

     1     2     3     4


ans =

     1     2     6    24


A =

     1     2     3
     4     5     6


ans =

     1     2     6
     4    20   120


ans =

     0     0
     0     0
     0     0


ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1


ans =

     8     1     6
     3     5     7
     4     9     2


ans =

     1     0     0
     0     1     0
     0     0     1


ans =
```

```
     2     3     1
     3     1     2


an_1 =

  logical

   0


c =

  logical

   0


animal =

  logical

   0


M =

     1     5     8
     9     4     3


A =

     7     2     6
     3     0     5


V =

  2×3 logical array

   0   1   0
   0   0   0


r3 =

  2×3 logical array

   1   0   0
   0   0   0


out =
```

```
  2×3 logical array

   0   1   1
   1   1   0


v =

     6     4     0     1


g2 =

  1×4 logical array

   1   1   0   1


availablePos =

     1     2     3     4     5     6     7     8     9


Error using input
Cannot call INPUT from EVALC.

Error in FinalParticipation (line 1304)
        playPos = input("Select a position to play...")
```

Published with MATLAB® R2020b