Problem 1, part 1:

File: problem1.m

% Problem 1 solution
% Carlos J. Cela, 2012

clear all

% Calculate points for function
step = 0.3;
n = 0:step:2*pi;
v = 5*sin(n*1000);

% Calculate numerical derivative
nd = derivative(v,n);

% Calculate derivative using diff
% diff assumes an increment of 1,
% so we have to divide by our
% increment to scale it properly.
dd = diff(v)./diff(n);

% Verify values
plot(n(1:size(nd,2)),nd,'bs-','linewidth',4,'markersize',20);
hold on
plot(n(1:size(dd,2)),dd,'ro-','linewidth',3,'markersize',10);
hold off

legend('Using combination','Using Matlab diff');
set(gca,'fontsize',14);
grid on
File: derivative.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% derivative V1.0
% % Numerically estimates the first derivative of a vector, using forward difference
% % for first point, backward difference for last point, and central difference for
% % all intermediate points.
% % % Usage:
% % d = derivative(y, x)
% % where
% % y = input vector containing function values
% % x = input vector containing argument increments
% % returns
% % d = Numerical derivative of y. All vectors have the same length.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function d = derivative(y, x)
d(1) = forward(y, x, 1);
for n = 2:size(y,2)-1
    d(n) = central(y, x, n);
end
d(size(y,2)) = backward(y, x, size(y,2));
end

File: forward.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% forward
% % Calculates derivative d of vector y at point x(p) using forward difference
% % approximation.
% % y = vector containing function values
% % x = vector containing increments
% % p = point number where to calculate
% % d = derivative of y at point p
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function d = forward(y, x, p)
d = (y(p+1)-y(p))/(x(p+1)-x(p));
end
File: central.m

% Calculates derivative d of vector y at point x(p) using central difference approximation.
% y = vector containing function values
% x = vector containing increments
% p = point number where to calculate
% d = derivative of y at point p

function d = central(y,x,p)
    d = (y(p+1)-y(p-1))/(x(p+1)-x(p-1));
end

File: backward.m

% Calculates derivative d of vector y at point x(p) using backward difference approximation.
% y = vector containing function values
% x = vector containing increments
% p = point number where to calculate
% d = derivative of y at point p

function d = backward(y,x,p)
    d = (y(p)-y(p-1))/(x(p)-x(p-1));
end
Numerical derivative result for each point n is:

<table>
<thead>
<tr>
<th>n</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-16.66260</td>
</tr>
<tr>
<td>0.3</td>
<td>0.36819</td>
</tr>
<tr>
<td>0.6</td>
<td>16.64633</td>
</tr>
<tr>
<td>0.9</td>
<td>-1.10384</td>
</tr>
<tr>
<td>1.2</td>
<td>-16.59754</td>
</tr>
<tr>
<td>1.5</td>
<td>1.83734</td>
</tr>
<tr>
<td>1.8</td>
<td>16.51635</td>
</tr>
<tr>
<td>2.1</td>
<td>-2.56725</td>
</tr>
<tr>
<td>2.4</td>
<td>-16.40289</td>
</tr>
<tr>
<td>2.7</td>
<td>3.29215</td>
</tr>
<tr>
<td>3.0</td>
<td>16.25740</td>
</tr>
<tr>
<td>3.3</td>
<td>-4.01062</td>
</tr>
<tr>
<td>3.6</td>
<td>-16.08016</td>
</tr>
<tr>
<td>3.9</td>
<td>4.72125</td>
</tr>
<tr>
<td>4.2</td>
<td>15.87151</td>
</tr>
<tr>
<td>4.5</td>
<td>-5.42266</td>
</tr>
<tr>
<td>4.8</td>
<td>-15.63187</td>
</tr>
<tr>
<td>5.1</td>
<td>6.11349</td>
</tr>
<tr>
<td>5.4</td>
<td>15.36169</td>
</tr>
<tr>
<td>5.7</td>
<td>-6.79237</td>
</tr>
<tr>
<td>6.0</td>
<td>-22.34769</td>
</tr>
</tbody>
</table>

**Problem 1, part 2:**
Plotting the results from the numerical code and Matlab diff or using a table to compare the values are both acceptable answers.
Plot:
Values:

<table>
<thead>
<tr>
<th>n</th>
<th>Numerical code (combination)</th>
<th>Matlab (diff)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-16.66260</td>
<td>-16.6626</td>
</tr>
<tr>
<td>0.3</td>
<td>0.36819</td>
<td>17.3990</td>
</tr>
<tr>
<td>0.6</td>
<td>16.64633</td>
<td>15.8937</td>
</tr>
<tr>
<td>0.9</td>
<td>-1.10384</td>
<td>-18.1014</td>
</tr>
<tr>
<td>1.2</td>
<td>-16.59754</td>
<td>-15.0937</td>
</tr>
<tr>
<td>1.5</td>
<td>1.83734</td>
<td>18.7684</td>
</tr>
<tr>
<td>1.8</td>
<td>16.51635</td>
<td>14.2643</td>
</tr>
<tr>
<td>2.1</td>
<td>-2.56725</td>
<td>-19.3988</td>
</tr>
<tr>
<td>2.4</td>
<td>-16.40289</td>
<td>-13.4070</td>
</tr>
<tr>
<td>2.7</td>
<td>3.29215</td>
<td>19.9913</td>
</tr>
<tr>
<td>3.0</td>
<td>16.25740</td>
<td>12.5235</td>
</tr>
<tr>
<td>3.3</td>
<td>-4.01062</td>
<td>-20.5447</td>
</tr>
<tr>
<td>3.6</td>
<td>-16.08016</td>
<td>-11.6156</td>
</tr>
<tr>
<td>3.9</td>
<td>4.72125</td>
<td>21.0581</td>
</tr>
<tr>
<td>4.2</td>
<td>15.87151</td>
<td>10.6849</td>
</tr>
<tr>
<td>4.5</td>
<td>-5.42266</td>
<td>-21.5303</td>
</tr>
<tr>
<td>4.8</td>
<td>-15.63187</td>
<td>-9.7335</td>
</tr>
<tr>
<td>5.1</td>
<td>6.11349</td>
<td>21.9604</td>
</tr>
<tr>
<td>5.4</td>
<td>15.36169</td>
<td>8.7630</td>
</tr>
<tr>
<td>5.7</td>
<td>-6.79237</td>
<td>-22.3477</td>
</tr>
<tr>
<td>6.0</td>
<td>-22.34769</td>
<td>(no value)</td>
</tr>
</tbody>
</table>
Error calculation: To get the average percent error, we first calculate the percent error of each data point, taking as a reference (e.g. 100%) the best value we have. In this case, the best value is the value from our code, which is a better approximation than Matlab diff, which is just a forward difference. Then, we average the error by adding and dividing by the number of points. We do not consider the last point, since Matlab algorithm returns a vector that is shorter.

<table>
<thead>
<tr>
<th>n</th>
<th>Numerical</th>
<th>Matlab</th>
<th>Percent error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>-16.6626</td>
<td>-16.6626</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>0.3682</td>
<td>17.3990</td>
<td>-4625.55</td>
</tr>
<tr>
<td>0.60</td>
<td>16.6463</td>
<td>15.8937</td>
<td>4.52</td>
</tr>
<tr>
<td>0.90</td>
<td>-1.1038</td>
<td>-18.1014</td>
<td>-1539.86</td>
</tr>
<tr>
<td>1.20</td>
<td>-16.5975</td>
<td>-15.0937</td>
<td>9.06</td>
</tr>
<tr>
<td>1.50</td>
<td>1.8373</td>
<td>18.7684</td>
<td>-921.50</td>
</tr>
<tr>
<td>1.80</td>
<td>16.5164</td>
<td>14.2643</td>
<td>13.64</td>
</tr>
<tr>
<td>2.10</td>
<td>-2.5673</td>
<td>-19.3988</td>
<td>-655.63</td>
</tr>
<tr>
<td>2.40</td>
<td>-16.4029</td>
<td>-13.4070</td>
<td>18.26</td>
</tr>
<tr>
<td>2.70</td>
<td>3.2922</td>
<td>19.9913</td>
<td>-507.24</td>
</tr>
<tr>
<td>3.00</td>
<td>16.2574</td>
<td>12.5235</td>
<td>22.97</td>
</tr>
<tr>
<td>3.30</td>
<td>-4.0106</td>
<td>-20.5447</td>
<td>-412.26</td>
</tr>
<tr>
<td>3.60</td>
<td>-16.0802</td>
<td>-11.6156</td>
<td>27.76</td>
</tr>
<tr>
<td>3.90</td>
<td>4.7213</td>
<td>21.0581</td>
<td>-346.03</td>
</tr>
<tr>
<td>4.20</td>
<td>15.8715</td>
<td>10.6849</td>
<td>32.68</td>
</tr>
<tr>
<td>4.50</td>
<td>-5.4227</td>
<td>-21.5303</td>
<td>-297.04</td>
</tr>
<tr>
<td>4.80</td>
<td>-15.6319</td>
<td>-9.7335</td>
<td>37.73</td>
</tr>
<tr>
<td>5.10</td>
<td>6.1135</td>
<td>21.9804</td>
<td>-259.21</td>
</tr>
<tr>
<td>5.40</td>
<td>15.3617</td>
<td>8.7630</td>
<td>42.96</td>
</tr>
<tr>
<td>5.70</td>
<td>-6.7924</td>
<td>-22.3477</td>
<td>-229.01</td>
</tr>
<tr>
<td>6.00</td>
<td>-22.3477</td>
<td>(no value)</td>
<td></td>
</tr>
</tbody>
</table>

Average error: -479.19 percent

Note the following:

1) The use of a forward approximation for derivatives tends to shift the derivative curve, so while both curves look similar in shape, error at each point is large when using forward or backward approximation only.

2) When we are trying to average an error, we have positive and negative values, which compensate each other. This is undesirable, because it masks the magnitude of the numerical error. Because of this, root mean square (RMS) errors are often use instead of simple averaging.

Problem 1, part 3:
The lengths of the vectors are different. This is so because using the Matlab diff command is equivalent to use the forward approximation, and the last data point cannot be calculated by this method.
Problem 2, part 1:

Gauss-Seidel iterative solver V1.0
Carlos J. Cela, Jan 2012

Solves a linear system of the form A x = b

Usage:

x = GSsolve(A, b, tolerance)

where

A = square coefficient matrix
b = constant term vector
tolerance = small number indicating the target tolerance (error to achieve convergence)

returns

x = unknown vector

Example:

GSsolve(A,b,0.001)

function x = GSsolve(A, b, tolerance)

n = size(A,1); % Get matrix size
x = zeros(n,1); % Result vector for iteration n
xn= zeros(n,1); % Result vector for iteration n+1
done = false; % Flag to exit iteration loop
ni = 0; % Number of iterations

while done==false

% Count the number of iterations
ni = ni+1;
% Do one iteration for each element of the unknown vector
for i = 1:n
    t1 = 0;
t2 = 0;

    % First summation term
    for j = 1:i-1
        if j>0
            t1 = t1 + A(i,j)*xn(j);
        end
    end

    % Second summation term
    for j = i+1:n
        t2 = t2 + A(i,j)*x(j);
    end

    % Assemble the iteration equation and calculate unknowns
    xn(i) = 1/A(i,i)*(b(i)-t1-t2);
end

% Calculate relative difference between results of this and last iteration
d = sum(abs(xn-x))/sum(abs(xn));
% Report iteration number and error
disp([num2str(ni) '  ' num2str(d)]);

% Update unknown vector with the recently-calculated values
x = xn;

% Check for convergence
if(d<tolerance)
    done = true;
end
end
end

Code verification:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Linear solver code verification
% Carlos J. Cela, 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

% Setup system
A = [
    10 -7 0; ...
    -3  6 1; ...
    2 -1 5; ...
];

b =[ 7; 4; 6];

% Solve using matlab
A\b

% Solve using GSsolve()
GSsolve(A, b, 0.001)

Output:

Matlab:
 1.64921
 1.35602
 0.81152

GSsolve:
 1.64852
 1.35560
 0.81171

Result values are consistent within the error requested.
Problem 2, part 2:

The routine GSsolve was modified to return a vector with the error for each iteration (trivial). A plot was made showing the convergence error vs iteration number.

It can be observed that the error decays in an exponential form, fast at the beginning, and slow later on.