HW 3 Solution key
Carlos J. Cela, 2012

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% central
%
% 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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% backward
%
% 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
```

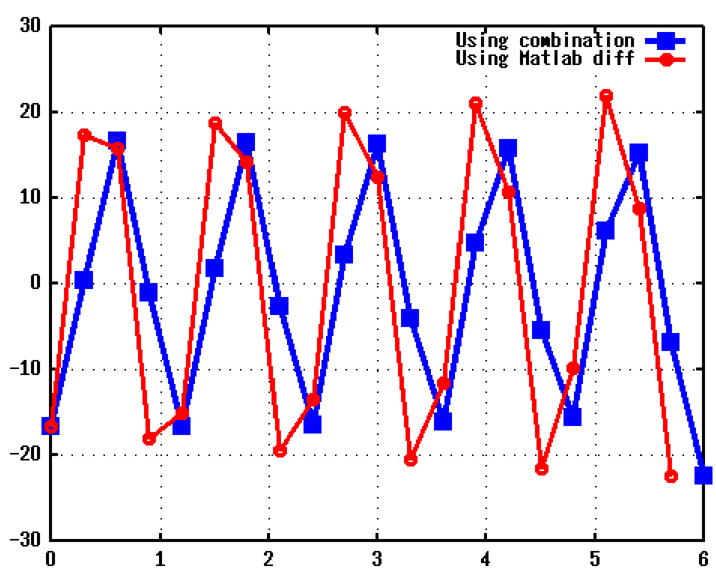
Numerical derivative result for each point n is:

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

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:

n	**Numerical code (combination)**	**Matlab (diff)**
0	-16.66260	-16.6626
0.3	0.36819	17.3990
0.6	16.64633	15.8937
0.9	-1.10384	-18.1014
1.2	-16.59754	-15.0937
1.5	1.83734	18.7684
1.8	16.51635	14.2643
2.1	-2.56725	-19.3988
2.4	-16.40289	-13.4070
2.7	3.29215	19.9913
3.0	16.25740	12.5235
3.3	-4.01062	-20.5447
3.6	-16.08016	-11.6156
3.9	4.72125	21.0581
4.2	15.87151	10.6849
4.5	-5.42266	-21.5303
4.8	-15.63187	-9.7335
5.1	6.11349	21.9604
5.4	15.36169	8.7630
5.7	-6.79237	-22.3477
6.0	-22.34769	(no value)

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.

n	Numerical	Matlab	Percent error
0.00	-16.6626	-16.6626	0.00
0.30	0.3682	17.3990	-4625.55
0.60	16.6463	15.8937	4.52
0.90	-1.1038	-18.1014	-1539.86
1.20	-16.5975	-15.0937	9.06
1.50	1.8373	18.7684	-921.50
1.80	16.5164	14.2643	13.64
2.10	-2.5673	-19.3988	-655.63
2.40	-16.4029	-13.4070	18.26
2.70	3.2922	19.9913	-507.24
3.00	16.2574	12.5235	22.97
3.30	-4.0106	-20.5447	-412.26
3.60	-16.0802	-11.6156	27.76
3.90	4.7213	21.0581	-346.03
4.20	15.8715	10.6849	32.68
4.50	-5.4227	-21.5303	-297.04
4.80	-15.6319	-9.7335	37.73
5.10	6.1135	21.9604	-259.21
5.40	15.3617	8.7630	42.96
5.70	-6.7924	-22.3477	-229.01
6.00	-22.3477	(no value)	
		Average error:	**-479.19 percent**

Note the following:

1) The use of a forward aproximation 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 convengence)
%
% 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
```

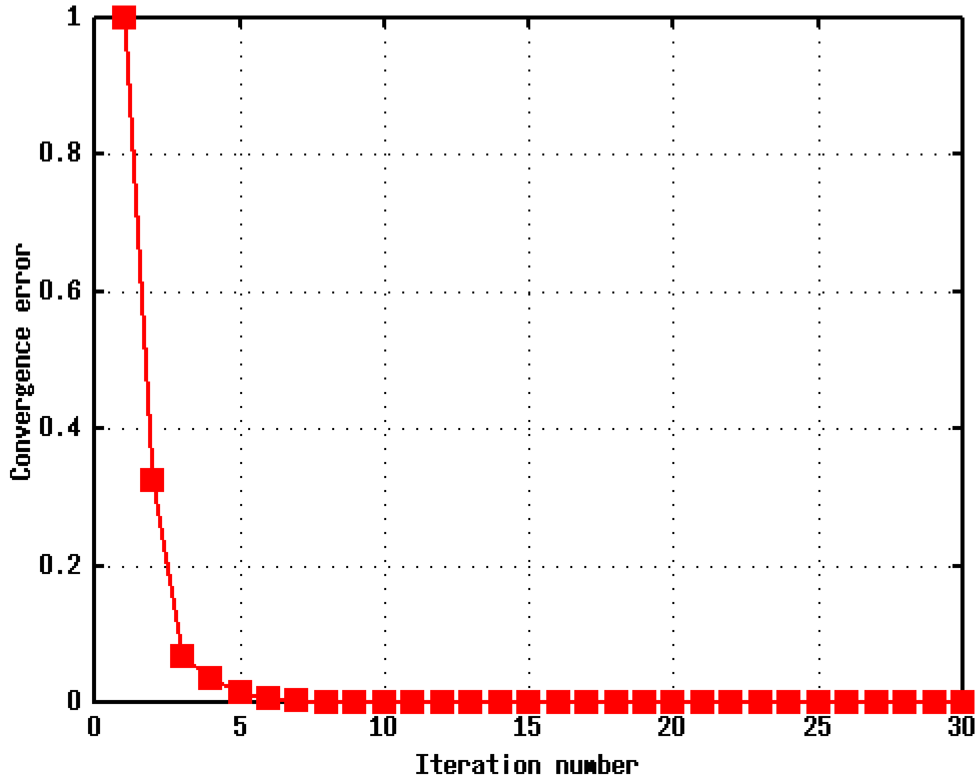
GSsolve:

```
   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.