Problem 1

```
A = [ 2  5 -9  3
      5  6 -4  2
      3 -4  2  7
      11 7  4 -8 ];

b = [ 151
      103
       16
      -32 ];

% Apply LU decomposition to A to obtain the L, U, and P,
[L, U, P] = gaussEliminate(A)

% We can retreive (and if desired override A with) the LU factor is one
% matrix

LU = joinDiags(L, U)
lu(A)

% For any right hand side b, calculate Pb (column vector),
b_ = P * b

% Use forward substitution to solve for y in equation Ly = Pb,
y = forwardSubstitute(L, b_)

% Use back substitution to solve for the unknowns x in equation Ux = y.
x = backSubstitute(U, y)

% Compare to actual solution
A\b

% *% Gauss Elimination*
function [ L, U, P ] = gaussEliminate ( A )
    n = size(A, 1);
    L = eye(n); U = A; P = eye(n);

    for k = 1:n-1 % elimination passes
        % make pos kk be largest from column k in lower diag
        [L, U, P] = maxPivot(L, U, P, k);

        for i = k+1:n % rows
            tmp = U(i,k)/U(k,k);
            for j=1:n
                U(i,j) = U(i,j)-tmp*U(k,j);
            end
            L(i,k) = tmp;
        end
    end
    L = myTril(L);
end

function [ L, U, P ] = maxPivot ( L, U, P, k )
```

```
    % Swap pivot with row with highest magnitude in pivot position
    % Store the transformation in the L and P matrices
    n = size(U, 1);
    k_ = U(k,:);
    k__ = L(k,:);
    k___ = P(k,:);

    [M, I] = max(abs(U(k:n, k)));
    j = k + I - 1;
    j_ = U(j,:);
    j__ = L(j,:);
    j___ = P(j,:);


    U(k,:) = j_; L(k,:) = j__; P(k,:) = j___;
    U(j,:) = k_; L(j,:) = k__; P(j,:) = k___;
end

function [ O ] = joinDiags ( L, U )
    L(eye(size(L))~=0) = 0; % remove ones from L
    O = triu(U) + tril(L);
end

function [ A ] = myTril ( A )
    % Restore the ones in the diagonal of the L matrix
    A = tril(A);
    A(eye(size(A))~=0) = 1;
end

% *Back Substitution*
function [ x ] = backSubstitute ( A, b )
    n = size(A, 1);
    x = zeros(1, n);
    x(n) = b(n)/A(n,n);
    for i = n-1:-1:1
        x(i) = (b(i)-sum(A(i,i+1:n).*x(i+1:n)))/A(i,i);
    end
end

% *Forward Substitution*
function [ x ] = forwardSubstitute ( A, b )
    n = size(A, 1);
    x = zeros(1, n);
    x(1) = b(1)/A(1,1);
    for i = 2:n
        x(i) = (b(i)-sum(A(i,1:i-1).*x(1:i-1)))/A(i,i);
    end
end
```

```
L =

    1.0000         0         0         0
    0.2727    1.0000         0         0
    0.1818   -0.6308    1.0000         0
    0.4545   -0.4769    0.5882    1.0000
```

```
U =

   11.0000    7.0000    4.0000   -8.0000
         0   -5.9091    0.9091    9.1818
         0    0.0000   -9.1538   10.2462
         0   -0.0000    0.0000    3.9882


P =

     0     0     0     1
     0     0     1     0
     1     0     0     0
     0     1     0     0


LU =

   11.0000    7.0000    4.0000   -8.0000
    0.2727   -5.9091    0.9091    9.1818
    0.1818   -0.6308   -9.1538   10.2462
    0.4545   -0.4769    0.5882    3.9882
```

Note that we could have built in the function that merges L and U into a single matrix into the Gauss elimination function and override the input matrix A, however I opted to do it in separate lines to allow for the flexibility of having L, U and P defined all of which are needed for this problem.

```
ans =

   11.0000    7.0000    4.0000   -8.0000
    0.2727   -5.9091    0.9091    9.1818
    0.1818   -0.6308   -9.1538   10.2462
    0.4545   -0.4769    0.5882    3.9882


b_ =

   -32
    16
   151
   103


y =

  -32.0000   24.7273  172.4154   27.9176


x =

    3.0000    5.0000  -11.0000    7.0000


ans =

     3
     5
   -11
     7
```

Problem 2

```
A = [ 2 -1  0  0
     -1  2 -1  0
      0 -1  2 -1
      0  0 -1  2 ];

f = [
    1
    1
    1
    1
    ];

% Check the solution we should be getting
A\f

% Retreive the elements of the tridiagonal matrix into vectors
[a, b, c] = getVectors(A);

% Perform Gaussian elimination, get new b and f, c stays the same, a in now
% full of 0
[b, f] = gaussEliminate ( a, b, c, f );

% Use back substitution to find the solution to the system
backSubstitute ( b, c, f )

% Retreive the vectors from the tridiagonal matrix
function [ a, b, c ] = getVectors (A)
    B = A; C = A;
    n = size(A, 1);

    A(1,:) = [];
    A(:,n) = [];

    C(:,1) = [];
    C(n,:) = [];

    a = [0, diag(A)']';
    b = diag(B);
    c = [diag(C)', 0]';

end

% Gauss tridiagonal elimination
function [ b, f ] = gaussEliminate ( a, b, c, f )
    n = size(b, 1);
    l = zeros(1, n);
    for j = 2:n
        l(j) = a(j)/b(j-1);
        b(j) = b(j)-l(j)*c(j-1);
        f(j) = f(j)-l(j)*f(j-1);
    end
end
```

```
% Back Substitution
function [ x ] = backSubstitute ( b, c, f )
    n = size(b, 1);
    x = zeros(1, n);
    x(n) = f(n)/b(n);
    for j=n-1:-1:1
        x(j) = (f(j)-c(j)*x(j+1))/b(j);
    end
end
```

```
ans =

    2.0000
    3.0000
    3.0000
    2.0000


ans =

    2.0000    3.0000    3.0000    2.0000
```

② Compute the LU decomposition of the tridiagonal matrix A

$$A = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \begin{matrix} R_2 - (-1/2) R_1 \rightarrow \\ R_3 - (0/2) R_1 \\ R_4 - (0/2) R_1 \end{matrix} \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 3/2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} R_3 - (-1/\tfrac{3}{2}) R_2 \rightarrow$$

$$\rightarrow \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 3/2 & -1 & 0 \\ 0 & 0 & 4/3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} R_4 - ((-1)/\tfrac{4}{3}) R_3 \rightarrow \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 3/2 & -1 & 0 \\ 0 & 0 & 4/3 & -1 \\ 0 & 0 & 0 & 5/4 \end{bmatrix} = U$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1/2 & 1 & 0 & 0 \\ 0 & -2/3 & 1 & 0 \\ 0 & 0 & -3/4 & 1 \end{bmatrix} = L$$

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Solve the system $Ax = b$

1. Use forward substitution to solve y from $Ly = b$
2. Use back substitution to solve x from $Ux = y$

1) $y_1 = b_1 / L_{11} = 1$, $y_2 = \dfrac{b_2 - L_{21} y_1}{L_{22}} = \dfrac{1 - (-1/2)}{1} = 3/2$

$$y_3 = \frac{b_3 - L_{31} y_1 - L_{32} y_2}{L_{33}} = \frac{1 - (-2/3)(3/2)}{1} = 2$$

$$y_4 = \frac{b_4 - L_{41} y_1 - L_{42} y_2 - L_{43} y_3}{L_{44}} = \frac{1 - (-3/4) 2}{1} = 5/2$$

$$y = \begin{bmatrix} 1 \\ 3/2 \\ 2 \\ 5/2 \end{bmatrix}$$

2) $x_4 = y_4 / U_{44} = \frac{5}{2} / \, ^5/_4 = 2$, $x_3 = \dfrac{y_3 - U_{34} x_4}{U_{33}} = \dfrac{2 - (-1)(2)}{4/3} = 3$

$$x_2 = \frac{y_2 - U_{24} x_4 - U_{23} x_3}{U_{22}} = \frac{3/2 - (-1)(3)}{3/2} = \frac{3/2 + 6/2}{3/2} = 3$$

$$x_1 = \frac{y_1 - U_{14} x_4 - U_{13} x_3 - U_{12} x_2}{U_{11}} = \frac{1 - (-1)(3)}{2} = 2$$

$$x = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 2 \end{bmatrix} \quad \square$$

b) Comment on how the number of computations performed by your specialized solver depends on N (nomber of non-zero entries)

- For a general (non - tridiagonal) matrix, the number of operations required for Gaussian elimination is proportional to n^3, as dictated by the forward elimination step
- For the specialized algorithm (for tri-diagonal matrices), the number of operations reauired are:

<u>elimination</u>

$(n-1)$ divissions (n-1 elem, 1 division each)

$2(n-1)$ multiplications (" " " multip ")

$2(n-1)$ additions (" " " additions ")

<u>substitution</u>

n divisions

$(n-1)$ multiplications

$(n-1)$ additions

The complexity of the algorithm is therefore linear in the number of non-zero elements, much faster than the 3rd degree polynomial complexity in the general case.

Problem 3

```
A = [
    0.1036 0.2122
    0.2081 0.4247
    ]

b = [
    0.7381
    0.9327
    ]

A\b
```

```
A =

    0.1036    0.2122
    0.2081    0.4247


b =

    0.7381
    0.9327


ans =

 -722.6525
  356.2907
```

Published with MATLAB® R2017a

③ Solve with only 4 significant figures

$$0.1036\, x_1 + 0.2122\, x_2 = 0.7381$$

$$0.2081\, x_1 + 0.4247\, x_2 = 0.9327$$

$$\left[\begin{array}{cc|c} .1036 & .2122 & .7381 \\ .2081 & .4247 & .9327 \end{array}\right] \quad R_2 - \frac{.2081}{.1036} R_1$$

$$\left[\begin{array}{cc|c} .1036 & .2122 & .7381 \\ 0 & -.1543E\text{-}2 & -.5499 \end{array}\right]$$

Use back substitution to find x_1, x_2

$x_2 = -.5499 / -.1543E\text{-}2 = 0.3563\ E3$ <----- x_2

$$x_1 = \frac{.7381 - (.2122)(.3563E3)}{.1036} = \frac{.7381 - .7561E2}{.1036} = \frac{-.7487}{.1036}$$

$x_1 = -0.7227\ E3$ <----- x_1

→ The results obtained by matlab A\b are the following

{ -722,6525 , 356, 2907 } which differ from the computed solution by the 4rth digit in both cases. This agrees to the expected error produced by the rounding mechanism with $\beta = 10$, $p = 4$ which should not exceed this boundary.

$$E_{rel} \leq \frac{1}{2}\beta^{1-P} = \frac{1}{2} 10^{-3} = 0{,}005$$

④ Proove the bounds on the absolute and relative errors associated with rounding

→ with base β, p digits of precission, any number x can be represented as $fl_r(x)$ as follows

$$x = \pm (0.d_1 d_2 \dots d_p \dots d_n)\beta^e$$

$$fl_r(x) = \begin{cases} \pm (0.d_1 d_2 \dots d_p)\beta^e & \text{when } d_{p+1} < \beta/2 \\ \pm (0.d_1 d_2 \dots d_{p+1})\beta^e & \text{when } d_{p+1} \geq \beta/2 \end{cases}$$

So there are two cases. In case 1, rounding behaves exactly like chopping, so we are interested in case 2. and hope to find a lower bound than β^{1-p} (for chopping)

→ Consider $|fl(x) - x|$ because both numbers have the same sign, this diff becomes greater when x smaller, so having $d_{p+1} \geq \beta/2 \Rightarrow d_{p+1} = \beta/2$, $d_{p+2} = \dots = d_n = 0$

Sibstracting $|fl(x) - x|$ gives

$$|(0, 0, 0_2 \dots 0_p\ \beta/2\ 0_{p+2} \dots 0_n)\beta^e|$$

which with the same reasoning as for chopping is bonded by

$\frac{1}{2}\beta^{e-p}$ (since β^{e-p} is given when $d_{p+1} = d_{p+2} = \dots d_n = \beta-1$ and this is $\frac{1}{2}$ of that)

→ As for chopping $|x| \geq \beta^{e-1}$ (equal when $d_1 = d_2 = \dots = d_p = \beta - 1$)

$$\therefore \frac{|fl(x) - x|}{|x|} \leq \frac{\frac{1}{2}\beta^{e-p}}{\beta^{e-1}} = \frac{1}{2}\beta^{1-p} \qquad \square$$

Problem 5

```
syms('OK','A','B','N','H','XI0','XI1','XI2','NN','I','X','XI','s','x');

F = @cos;

OK = 1;
res = ones(7,1);
for i=1:7

    A = 0;
    B = pi/2;

    N = 2^i;

    H = (B-A)/N;
    XI0 = F(A) + F(B);
    XI1 = 0;
    XI2 = 0;
    NN = N - 1;

    for I = 1:NN
        X = A + I * H;
        if rem(I,2) == 0
            XI2 = XI2 + F(X);
        else
            XI1 = XI1 + F(X);
        end
    end

    XI = (XI0 + 2.0 * XI2 + 4.0 * XI1) * H / 3.0;
    res(i) = XI;

end

format long
err = (res - 1);

rat = ones(6,1);

for i=1:6
    rat(i) = err(i) / err(i + 1);
end

rat
```

```
rat =

  16.940059660203076
  16.223806321752807
  16.055292262322119
  16.013782490272465
  16.003442132276163
```

```
16.000868785233823
```

Published with MATLAB® R2017a

We observe that the ratio of subsequent aproximations gets closer and closer to 16. This is the behaviour we expected because, if the error depends on the 4th power of h, then when h is reduced by half, the error should be reduced by (1/2)^4 = 1/16. This is endeed what we observe.