1.

( I add some additional comment in the .py, but my code seems to be easy to read because I print out the process)

(a) The value is taken only 3 significant digits whenever a arithmetic operation is made. The order of operation : { * → / → - }.

```python
def get3(double):
    formatted_x = format(double, '.3g')
    return float(formatted_x)


array = [[2.51, 1.48, 4.53, 0.05], [1.48, 0.93, -1.3, 1.03], [2.68, 3.04, -1.48, -0.53]]
for i in range(2):
    for j in range(i+1, 3):
        mul = array[j][i]
        for k in range(4):
            array[j][k] = get3(array[j][k] - get3(get3(mul * array[i][k]) / array[i][i]))
        print (array[0])
        print(array[1])
        print(array[2])
        print("|")
        print("V")


for i in range(2, -1, -1):
    for j in range(i-1, -1, -1):
        mul = array[j][i]
        for k in range(4):
            array[j][k] = get3(array[j][k] - get3(get3(mul * array[i][k]) / array[i][i]))
        print (array[0])
        print(array[1])
        print(array[2])
        print("|")
        print("V")

x = [0, 0, 0]
for i in range(3):
    x[i] = array[i][3] / array[i][i]

print(x)
```

```
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.97, 1.0]
[2.68, 3.04, -1.48, -0.53]
|
V
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.97, 1.0]
[0.0, 1.46, -6.3, -0.583]
|
V
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.97, 1.0]
[0.0, 0.0, 95.7, -26.2]
|
V
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, 0.0, -0.09]
[0.0, 0.0, 95.7, -26.2]
|
V
[2.51, 1.48, -0.01, 1.29]
[0.0, 0.057, 0.0, -0.09]
[0.0, 0.0, 95.7, -26.2]
|
V
[2.51, 0.0, -0.01, 3.62]
[0.0, 0.057, 0.0, -0.09]
[0.0, 0.0, 95.7, -26.2]
|
V
[1.4422310756972114, -1.5789473684210524, -0.27377220480668757]
```

Ans = the last row of the right pic. ([x y z])

(b) I changed the rows directly and inspect the process to make sure that it really did the pivoting

```
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.37, 5.91, 0.546]
[1.48, 0.93, -1.3, 1.03]
|
V
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.37, 5.91, 0.546]
[0.0, -0.75, -0.483, 1.32]
|
V
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.37, 5.91, 0.546]
[0.0, 0.002, -3.71, 1.02]
|
V
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.37, 0.01, 2.18]
[0.0, 0.002, -3.71, 1.02]
|
V
[2.68, 3.04, 0.0, -0.937]
[0.0, -1.37, 0.01, 2.18]
[0.0, 0.002, -3.71, 1.02]
|
V
[2.68, 0.0, 0.0222, 3.9]
[0.0, -1.37, 0.01, 2.18]
[0.0, 0.002, -3.71, 1.02]
|
V
[1.4552238805970148, -1.5912408759124088, -0.2749326145552561]
```

In some places that the value should be zero is not zero because we only choose the three significant digits, but it seems that it won't affact the result.

(c) now add the function chop

( the non-zero digit in b would not affact the result, but not in c, so I add some code to set the zeros)

```python
import math
def get3(double):
    formatted_x = format(double, '.3g')
    return float(formatted_x)

def chop(num, n=3):
    if num == 0:
        return 0
    sign = -1 if num < 0 else 1
    num = abs(num)
    exponent = int(math.floor(math.log10(num)))
    if exponent < n - 1:
        return sign * math.floor(num * 10**(n - exponent - 1)) / 10**(n - exponent - 1)
    else:
        return sign * math.floor(num / 10**(exponent - n + 1)) * 10**(exponent - n + 1)

array = [ [2.68, 3.04, -1.48, -0.53],[2.51, 1.48, 4.53, 0.05],[1.48, 0.93, -1.3, 1.03] ]
for i in range(2):
    for j in range(i+1, 3):
        mul = array[j][i]
        for k in range(4):
            array[j][k] = chop(array[j][k] - chop(chop(mul * array[i][k]) / array[i][i]))
        print (array[0])
        print(array[1])
        print(array[2])
        print("|")
        print("V")
array[1][0] = 0
array[2][0] = 0
array[2][1] = 0
print (array[0])
print(array[1])
print(array[2])
print("|")
print("V")

for i in range(2, -1, -1):
    for j in range(i-1, -1, -1):
        mul = array[j][i]
        for k in range(4):
            array[j][k] = chop(array[j][k] - chop(chop(mul * array[i][k]) / array[i][i]))
        print (array[0])
        print(array[1])
        print(array[2])
        print("|")
        print("V")

x = [0, 0, 0]
for i in range(3):
    x[i] = array[i][3] / array[i][i]
print("x = ",end="")
print(x)
```

```
[2.68, 3.04, -1.48, -0.53]
[0.00999, -1.36, 5.91, 0.546]
[1.48, 0.93, -1.3, 1.03]
|
V
[2.68, 3.04, -1.48, -0.53]
[0.00999, -1.36, 5.91, 0.546]
[0.01, -0.739, -0.483, 1.32]
|
V
[2.68, 3.04, -1.48, -0.53]
[0.00999, -1.36, 5.91, 0.546]
[0.00458, -0.004, -3.68, 1.02]
|
V
[2.68, 3.04, -1.48, -0.53]
[0, -1.36, 5.91, 0.546]
[0, 0, -3.68, 1.02]
|
V
[2.68, 3.04, -1.48, -0.53]
[0, -1.36, 0.02, 2.17]
[0, 0, -3.68, 1.02]
|
V
[2.68, 3.04, -0.01, -0.937]
[0, -1.36, 0.02, 2.17]
[0, 0, -3.68, 1.02]
|
V
[2.68, 0.01, 0.0346, 3.9]
[0, -1.36, 0.02, 2.17]
[0, 0, -3.68, 1.02]
|
V
x = [1.4552238805970148, -1.5955882352941175, -0.27717391304347827]
```

(d)

Add the following code to compute the error

```python
y = [0,0,0]
array = [[2.51, 1.48, 4.53, 0.05], [1.48, 0.93, -1.3, 1.03],[2.68, 3.04, -1.48, -0.53] ]
for i in range(3):
    for j in range(3):
        y[i] += array[i][j]*x[j]

    y[i] = y[i] - array[i][3]
print("y = ",end="")
print(y)

acr = (y[0]**2+y[1]**2+y[2]**2)**(1/2)

print("accuracy : ",end="")
print (acr)
```

for(a)

```
x = [1.4422310756972114, -1.5789473684210524, -0.27377220480668757]
y = [-0.007030193037451851, -0.008015194351012278, 0.0003621459824245665]
accuracy : 0.010667619431762735
```

for(b)

```
x = [1.4552238805970148, -1.5912408759124088, -0.2749326145552561]
y = [0.0021307000128319292, 0.0012897276068746244, -0.00047199323194413445]
accuracy : 0.0025349669535553315
```

for (c)

```
x = [1.4552238805970148, -1.5955882352941175, -0.27717391304347827]
y = [-0.014456474023743796, 0.00016037141657432308, -0.01037084398976984]
accuracy : 0.017792407516988205
```

(b) (round & partial pivoting) is the one with less error.

The accuracy formula : ||Ax'-b||2, x' is the computed result

2.

(a)

compact the system in to matrix A
if bandwith > 3
    B = A
(use B to store value and A to compute)
for i = 0~n    (Gauss elimination)
    eliminate the ith column with row operation and store the changed value in B
    the elimination process take the value in B if the index (j,k) j<i and k<i
    else use the value in A
    if bandwith == 3
        eliminate with only matrix A (since all the thing is eliminated, no need to store new value
repeat for elimination

get matrix X with $X[i] = \frac{A[i][-1]}{A[i][i]}$

(b)

```
array = [4 -1 100;4 -1 200;4 -1 200;4 -1 200;4 -1 200;4 -1 100];

arr = compute(array);
disp(arr)
function arr = compute(array)
    for i = 2:size(array,1)
        % only 1 row has to be reduced for each i
        array(i,1) = array(i,1)-array(i,2)*array(i-1,2)/array(i-1,1);
        array(i,3) = array(i,3)-array(i,2)*array(i-1,3)/array(i-1,1);
    end
    for i = size(array,1)-1:-1:1
        % don't need to compute (i,2) because it wouldn't affact the result
        array(i,3) = array(i,3)-array(i+1,3)*array(i,2)/array(i+1,1);
    end
    arr = zeros(size(array,1),1);
    for i = 1:size(array,1)
        arr(i,1) = array(i,3)/array(i,1);
    end
end
```

```
Q2
46.3415
85.3659
95.1220
95.1220
85.3659
46.3415
```

(c)

```
array = [4 -1 100;4 -1 200;4 -1 200;4 -1 200;4 -1 200;4 -1 100];

arr = compute(array);
disp(arr)
function arr = compute(array)
    for i = 2:size(array,1)
        array(i,1) = array(i,1)-array(i,2)*array(i-1,2)/array(i-1,1);  } 6×(n-1)
        array(i,3) = array(i,3)-array(i,1)*array(i-1,3)/array(i-1,1);  }
    end
    for i = size(array,1)-1:-1:1
        array(i,3) = array(i,3)-array(i+1,3)*array(i,2)/array(i+1,1);  3×(n-1)
    end
    arr = zeros(size(array,1),1);
    for i = 1:size(array,1)
        arr(i,1) = array(i,3)/array(i,1);  1×n
    end
end
```

$6\times(n-1)+3(n-1)+n = 10n-9$ arithmetic operations

3.

(a)

$$L+U = \begin{bmatrix} 0 & -1.21 & 3.22 \\ -3.07 & 0 & 2.11 \\ 1.26 & 3.11 & 0 \end{bmatrix}$$

$$D^{-1} = \begin{bmatrix} \frac{1}{4.63} & 0 & 0 \\ 0 & \frac{1}{5.48} & 0 \\ 0 & 0 & \frac{1}{4.57} \end{bmatrix}$$

```
LU = [0 -1.21 3.22;-3.07 0 2.11;1.26 3.11 0];
D = [4.63 0 0;0 5.48 0;0 0 4.57];
D = inv(D);
b = [2.22;-3.17;5.11];
x = [0;0;0];
for i = 1:500
    x = -D*LU*x + D*b;
end
disp(x)
```

```
>> Q3a
   -8.9893
   -9.4845
   10.0510
```

(b)

```
LD = [4.63 0 0;-3.07 5.48 0;1.26 3.11 4.57];
U = [0 -1.21 3.22;0 0 2.11;0 0 0];

b = [2.22;-3.17;5.11];
x = [0;0;0];
for i = 1:500
    x = -(LD\U)*x+LD\b;
end
disp(x)
```

```
Q3b
-8.9893
-9.4845
10.0510
```

4.

(a)(b)

```
A = [4.62 -1.21 3.22; -3.07 5.48 2.11;1.26 3.11 4.57];
b = [2.22;-3.17;5.11];
x=[0 0 0]';
maxerror = 0.00001;
n = size(x,1);
for w = 1:0.1:2                              % choose w = 1,1.1,1.2,1.3...2
    error = inf;
    itr = 0;
    x=[0 0 0]';
    while error>maxerror
        x_old = x;
        for i=1:n
            sum = 0;
            for j=1:n
                sum = sum+A(i,j)*x(j);       % the sum
            end
            x(i) = x(i)+(w/A(i,i))*(b(i)-sum);   % update x(i)
        end
        itr = itr+1;
        error = norm(x_old-x);
    end
    fprintf("w : %-2f\nIteration : %d\n",w,itr)
end
```

```
>> Q4a
w : 1.000000
Iteration : 114
w : 1.100000
Iteration : 91
w : 1.200000
Iteration : 71
w : 1.300000
Iteration : 54
w : 1.400000
Iteration : 35
w : 1.500000
Iteration : 29
w : 1.600000
Iteration : 44
w : 1.700000
Iteration : 90
w : 1.800000
Iteration : 123039
w : 1.900000
Iteration : 5191
w : 2.000000
Iteration : 2791
```

The stopping condition is norm(x_old – x) < 1e-5

The original convergence times of iteration is same as the w=1, which is 114. To achieve a speed up of 3, we should find a w which has iteration count less than 38, and we can see that w = 1.5 satisfies the condition.

5.

(a)

$$A = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{-10} \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{10} \end{bmatrix}$$

$$\|A\| \cdot \|A^{-1}\| = 10^{10} \times 10^{10} = 10^{20}$$

(b)

$$B = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix} = \begin{bmatrix} 10^{10} \times 1 & 0 \\ 0 & 10^{10} \times 1 \end{bmatrix} = 10^{10} I$$

since $\mathrm{cond}(I) = 1$ and $\mathrm{cond}(I) = \mathrm{cond}(kI)$

$\mathrm{cond}(B) = 1$

(c)

same as (b), $\mathrm{cond}(C) = \mathrm{cond}(10^{-10} I) = 1$

(d)

$\det\left(\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}\right) = 0$, the matrix is singular

$\mathrm{cond}\left(\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}\right) = \infty$

(a)(d) are ill-conditioned, (b)(c) are well-conditioned

6.

```
array = [4 -1 0 0 0 0 100;-1 4 -1 0 0 0 200;0 -1 4 -1 0 0 200;0 0 -1 4 -1 0 200;0 0 0 -1 4 -1 200;0 0 0 0 -1 4 100];

w = 3;
amount = (w-1)/2+1;
%  you can input the array and w(band width) on your own

for i = 1:size(array,1)-1
    for j = i+1:i+amount-1
        for k = i+1:i+amount-1
            array(j,k) = array(j,k)-array(j,i)*array(i,k)/array(i,i);
        end
        array(j,size(array,2)) = array(j,size(array,2)) - array(j,i)*array(i,size(array,2))/array(i,i);
        array(j,i) = 0;
    end
end
% display now result
disp(array)
for i = size(array,1):-1:2
    for j = i-amount+1:i-1
        for k = i-amount+1:i-1
            array(j,k) = array(j,k)-array(j,i)*array(i,k)/array(i,i);
        end
        array(j,size(array,2)) = array(j,size(array,2)) - array(j,i)*array(i,size(array,2))/array(i,i);
        array(j,i) = 0;
    end
end
% display now result
disp(array)
x = zeros(size(array,1),1);
for i = 1:size(array,1)
    x(i,1) = array(i,size(array,2))/array(i,i);
end
% display answer
disp(x);
```

Use the matrix in 2 for example:

The 2 matrix printed is the process of computing

```
Q6
4.0000   -1.0000         0         0         0         0  100.0000
     0    3.7500   -1.0000         0         0         0  225.0000
     0         0    3.7333   -1.0000         0         0  260.0000
     0         0         0    3.7321   -1.0000         0  269.6429
     0         0         0         0    3.7321   -1.0000  272.2488
     0         0         0         0         0    3.7321  172.9487

4.0000         0         0         0         0         0  185.3659
     0    3.7500         0         0         0         0  320.1220
     0         0    3.7333         0         0         0  355.1220
     0         0         0    3.7321         0         0  355.0087
     0         0         0         0    3.7321         0  318.5903
     0         0         0         0         0    3.7321  172.9487

46.3415
85.3659
95.1220
95.1220
85.3659
46.3415
```