“Naive” Gaussian Elimination:

System of Equations:

$$2x_1 + 3x_2 + x_3 = 9$$
$$4x_1 - 2x_2 + 5x_3 = -2$$
$$x_1 + 4x_2 - 6x_3 = 33$$

System in matrix form (Ax = b):

```
>> A = [2 3 1; 4 -2 5; 1 4 -6]
A =
     2     3     1
     4    -2     5
     1     4    -6

>> b = [9; -2; 33]
b =
     9
    -2
    33
```

First step – form augmented matrix by combining *A* and *b*

```
>> C = [A b]
C =
    2    3    1    9
    4   -2    5   -2
    1    4   -6   33
```

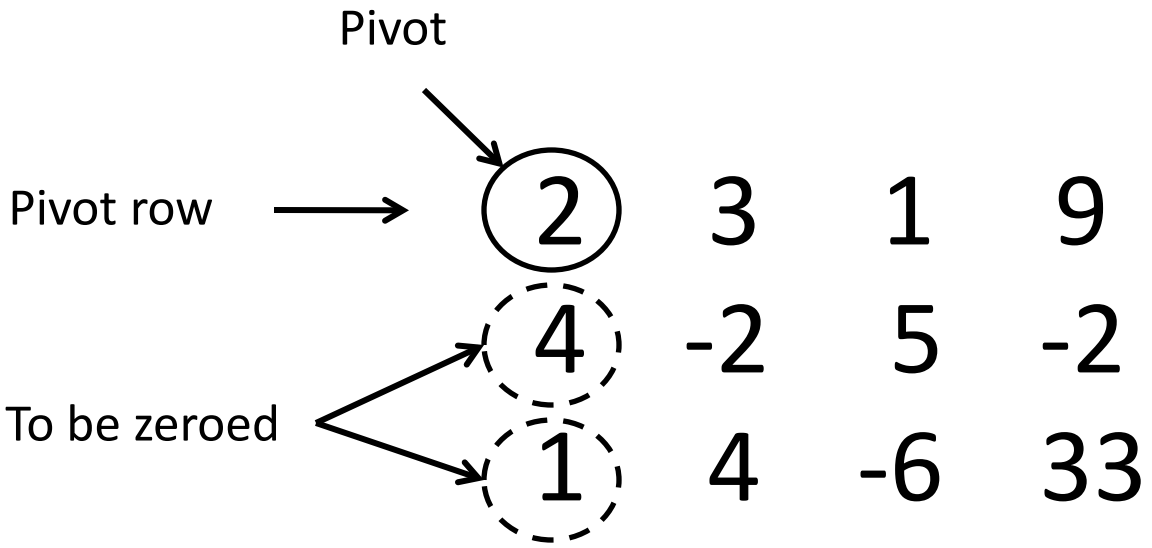
In general:

[M1 M2 M3 ... Mn] % OK as long as all matrices have same number of rows

M1	M2	M3	…	Mn

[M1; M2; M3; ... Mn] % OK as long as all matrices have same number of columns

M1
M2
M3
…
Mn



All elements below the pivot are converted to zero by applying

row = row – (element below pivot/ pivot) * pivot row

In this case

row 2 = row 2 – (4/2) * pivot row
row 3 = row 3 – (1/2) * pivot row

Row 1 is the pivot row, C(1,1) is the pivot:

```
P = C(1,1);
```

Subtract C(2,1)/P times the pivot row from row 2:

```
>> C(2,:) = C(2,:) - (C(2,1)/P) * C(1,:)
C =
     2     3     1     9
     0    -8     3   -20
     1     4    -6    33
```

Subtract C(3,1)/P times the pivot row from row 3:

```
>> C(3,:) = C(3,:) - (C(3,1)/P) * C(1,:)
C =
    2.0000    3.0000    1.0000    9.0000
         0   -8.0000    3.0000  -20.0000
         0    2.5000   -6.5000   28.5000
```

Same idea applied using row 2 as the pivot row and C(2,2) as the pivot:

```
>> P = C(2,2);
>> C(3,:) = C(3,:) - (C(3,2)/P) * C(2,:)

C =

    2.0000    3.0000    1.0000    9.0000
         0   -8.0000    3.0000  -20.0000
         0         0   -5.5625   22.2500
```

For a larger system C(3,3) would be the next pivot and so on.

In general (n equations) the pivot runs from C(1,1) to C(n-1, n-1)

Once the last pivot has been used all elements below the diagonal of matrix *A* are zero (*A* is *upper triangular*).

The system of equations represented by C is entirely equivalent to the original system (all operations have been mathematically legitimate) and can easily be solved using *back substitution*.

New system of equations:

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &= 9 \\ -8x_2 + 3x_3 &= -20 \\ -5.5625x_3 &= 22.25 \end{aligned}$$

```
>> x(3) = C(3,4)/C(3,3)
x =

    0         0   -4.0000

>> x(2) = (C(2,4) - x(3) * C(2,3))/C(2,2)
x =

    0    1.0000   -4.0000

>> x(1) = (C(1,4) - x(2) * C(1,2) - x(3) * C(1,3))/C(1,1)
x =

     5     1    -4
```

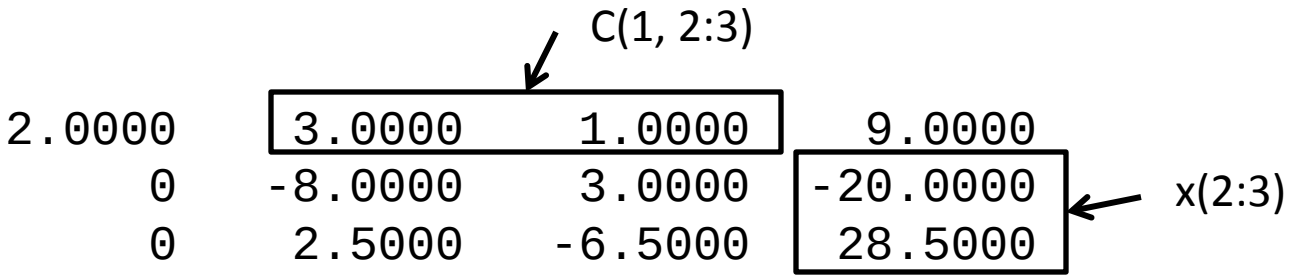
The basic elimination process is easily implemented in Matlab.

Elements to the left of the pivot can be ignored (these elements will be zero and will stay that way).

```
function [ x ] = naiveGauss( A, b )
%NAIEVEGAUSS naive Gaussian elimination
% solves Ax = b
% A must be n x n and b must be n x 1
% the result is n x 1

[n, nn] = size(A); [m, mm] = size(b);
if n ~= nn || n ~= m || mm ~= 1, error 'bad dimensions', end

np = n + 1; % a useful value

C = [A b];
for i = 1 : n - 1  % for all pivots
   for k = i + 1 : n % for all rows after pivot row
      C(k, i : np) = C(k,i : np) - (C(k, i) / C(i, i)) * C(i, i : np);
   end
end
```

Back substitution requires slightly more complex programming.

```
% back substitution
x = ones(n, 1); % preallocate (column vector)
x(n) = C(n, np) / C(n, n);
for i = n - 1: -1: 1  % work backwards to first equation
   x(i) = (C(i, np) - C(i, i + 1: n) * x (i + 1: n))/C(i, i);
end

end
```



x(1) = (C(1,4) – (C(1,2) * x(2) + C(1,3) * x(3))) / C(1,1)

= (C(1,4) – (C(1,2:3) * x(2:3))) / C(1,1)

Cost of Gaussian elimination:

Computational time for elimination process is O(n^3) ("order *n* cubed").

Detailed discussion is in the text (p222).

Follows from the fact that the process involves three nested loops and that in each case the number of iterations depends upon *n*.

The third loop is implicit in the vectorized code. It can be explicit by rewriting the elimination code as shown below:

```
for i = 1 : n - 1  % for all pivots
   for k = i + 1 : n % for all rows after pivot row
      multiplier = C(k, i) / C(i, i);
      for m = i : np
         C(k, m) = C(k, m) - multiplier * C(i, m);
      end
   end
end
```

Computational time for back substitution is $O(n^2)$ ("order n squared").

Follows from the fact that the process involves two nested loops and that in each case the number of iterations depends upon n. Again the last loop is implicit.

Computational time for the complete process is $O(n^3)$ (as n increases n^3 soon swamps n^2).

In crude terms every doubling of n increased the time required by a factor of $2^3 = 8$.

If solving a system of 100 equations takes 1 second solving a system of 800 equations (three doublings) will take approximately 512 seconds = 8.533 minutes.

Pivoting:

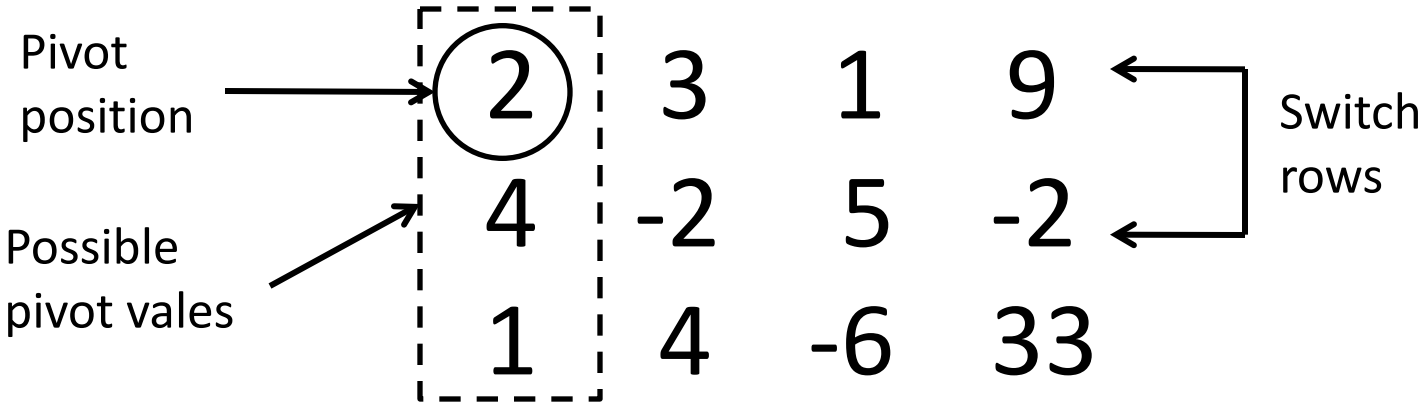
Naive Gaussian elimination fails if a pivot element is zero.

This problem is addressed by *partial pivoting* (aka *row pivoting*).

If any of the elements below the pivot position have a greater magnitude than the pivot element rows are switched so as to make the magnitude of the pivot element as large as possible.

Switching rows does not affect the system of equations (it merely alters the order of the equations).

If all possible pivots are zero the system of equations cannot be solved.



Function *max* can be used to locate the maximum value in a vector:

```
[maxVal, positionOfMaxVal ] = max(vector);
```

```
C = [A b];
for i = 1 : n - 1  % for all pivots

   [ pivot, k ] = max(abs(C(i:n, i))); % find best pivot value
   if k ~= 1 % row interchange required - swap row i with row i + (k - 1)
      temp = C(i, :);
      C(i, :) = C(i + (k - 1), :);
      C(i + (k - 1), :) = temp;
   end

   if C(i, i) == 0, error 'no unique solution', end

   for k = i + 1 : n % for all rows after pivot row
      C(k, i : np) = C(k, i : np) - (C(k, i) / C(i, i)) * C(i, i : np);
   end

end
```

Notes on pivoting:

Apart from dealing with pivot elements that are zero, pivoting also minimizes the effects of round off errors.

In the iterative techniques previously discussed (bisection search, etc.), round off errors do not accumulate. From the point of view of round off errors, each iteration is a fresh start.

This is not true of Gaussian elimination. Errors accumulate. Final element values are the cumulative result of many arithmetic operations.

Errors can lead to poor solutions or, in extreme cases, to solutions that are of no use at all.

It is also possible to switch columns as well as rows (*complete pivoting*) but the advantages of doing this do not outweigh the additional complexity.

Illustration of the advantages of pivoting:

System of equations: $0.0001\,x_1 + x_2 = 3$

$x_1 + 2x_2 = 5$

Assuming a machine with 7 significant digits, no pivoting:

final C = 0.0001 1 3 $x_1 = -1.002004$

0 -998 -2995 $x_2 = 3.001002$

Assuming a machine with only 2 significant digits, no pivoting:

final C = 0.0001 1 3 $x_1 = 0$ (very bad)

0 -1000 -3000 $x_2 = 3$

Assuming a machine with only 2 significant digits, partial pivoting:

initial C = 1 2 5

0.0001 1 3

final C = 1 2 5 $x_1 = -1$

0 1 3 $x_2 = 3$