Solving Linear Systems on an Industrial Scale

- Iterative methods
  - Jacobi method
  - Gauss-Seidel method
  - Relaxation method
- Pivoting procedures

Solving Linear Systems on an Industrial Scale

- Finding solution to $\mathbf{Ax} = \mathbf{b}$
  - Inverting matrix $\mathbf{A}$, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{x}$ for square non-singular matrices. Time consuming $2n^3$ operations. Equivalent of solving n linear systems with different right hand sides
  - Gaussian elimination ($2n^3/3$ operations). Not that suitable for multiple right hand sides $\mathbf{Ax} = \mathbf{b_i}$, $i = 1, 2, \ldots n, \ldots$
  - **LU** decompositions
- Iterative solutions

Iterative solution of linear systems

- Sometimes an approximate solution of linear systems can be obtained by taking an initial approximation $\mathbf{x}^0$ to the solution, then carrying out an

  Iterative procedure.

- We will investigate the process below.

- We need to recall some properties of norms (length, distance) of vectors as well.

Fixed point iteration for $f(x) = 0$

Recall Newton's method for solving $f(x) = 0$:

Newton's: $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$

In general, fixed point iteration has form:

$x_{k+1} = g(x_k)$ for some suitable function g.

Theorem: [The 1D fixed point theorem] Suppose that the equation $g(x) = x$ has a solution $x = s$ and that there is an interval $I = (s - r, s + r)$ in which $g'(x)$ exists and satisfies the condition $|g'(x)| \leq m < 1$. Then

1. for all $x_0 \in I$, the sequence of iterates,
$$x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots$$
lies in I.
2. $\lim\limits_{k\to\infty} x_k = s$
3. s is the only root of $g(x) = x$.

Fixed point iteration

This result tells us that if we wish to solve a nonlinear equation $f(x) = 0$, it is often possible to do so (approximately) by

1. taking an initial guess x_0,
2. rewriting the equation in the form $x = g(x)$ and
3. iterating, *ie* computing

$$\begin{aligned} x_1 &= g(x_0) \\ x_2 &= g(x_1) \\ &\vdots \\ x_{k+1} &= g(x_k) \\ &\vdots \end{aligned}$$

until the sequence $\{x_k\}$ appears to converge.

Fixed point iteration

Example: Find the smallest root of
$x^2 - 10x + 1 = 0$.
We could do this by using the quadratic formula. Instead let us used fixed point iteration.
We rewrite the equation in the form

$$x = \frac{1+x^2}{10}.$$

Here $g(x) = \frac{1+x^2}{10}$, so $g'(x) = x/5$. Clearly, if $|x| < 5$ then $|g'(x)| < 1$ so the conditions of the theorem are satisfied.
Take $x_0 = 0$. Then $x_1 = \frac{1+x_0^2}{10} = \frac{1}{10}$.
$x_2 = \frac{1+x_1^2}{10} = \frac{1+0.1^2}{10} = 0.101$.
$x_3 = \frac{1+x_2^2}{10} = \frac{1+(0.101)^2}{10} = 0.1010201$.
Application of the quadratic formula gives the exact value for the small root as $x = 5 - \sqrt{25-1} = 0.101020514$.

Iterative solution of linear systems

Can we adapt this strategy to the solution of linear systems? One

of the problems that we will need to address when we solve a linear system iteratively, is

when to terminate the procedure.

To decide when a calculation should be halted, we require a measure of how close an approximate solution is to the true solution.

This requires the notion of a **norm** (distance).

Iterative solution of linear systems

In summary, we need:

- iteration formula
- convergence in $\mathbb{R}^n$ (norms, limits)
- conditions guaranteeing convergence
- practical stopping criterion

Definition of a Norm

A norm on $\mathbb{R}^n$ is a function $\|\cdot\|$ such that

1. $\|\mathbf{x}\| \geq \mathbf{0}$ for all $\mathbf{x} \in \mathbb{R}^n$.
2. $\|\mathbf{x}\| = \mathbf{0}$ if $\mathbf{x} = \mathbf{0}$.
3. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$.
4. $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$, for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

Examples of norms

There are many possible norms on $\mathbb{R}^n$.
Examples: consider a vector $\mathbf{x}$ in $\mathbb{R}^n$ with elements $\mathbf{x} = (x_1, x_2, \dots x_n)$. Then:

1. The Euclidean distance, or l_2 norm:
   $\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
2. The l_∞ or *sup* norm:
   $\|x\|_\infty = \max\limits_{1 \le i \le n} |x_i|$.
3. The l_1 norm:
   $$\|x\|_1 = \sum_{i=1}^n |x_i|$$

Distance between 2 vectors

The *distance* between two vectors $\mathbf{x} = (x_1, x_2, \dots x_n)$ and $\mathbf{y} = (y_1, y_2, \dots y_n)$ in $\mathbb{R}^n$ is the norm of $\mathbf{x} - \mathbf{y}$.

1. The Euclidean distance, or l_2 norm distance:
   $\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
2. The l_∞ or *sup* norm distance:
   $\|\mathbf{x} - \mathbf{y}\|_\infty = \max\limits_{1 \le i \le n} |x_i - y_i|.$
3. The l_1 norm distance:
   $$\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^n |x_i - y_i|$$

Convergence in $\mathbb{R}^n$

Intuitively a sequence of vectors $\left\{\mathbf{x}^{(k)}\right\}$ converges in $\mathbb{R}^n$ if the terms approach a fixed vector, say $\mathbf{v}$.

Definition: The sequence $\left\{\mathbf{x}^{(k)}\right\} \in \mathbb{R}^n$ converges to a limit $\mathbf{v} \in \mathbb{R}^n$ with respect to the norm $\| \ \|$ if for any $\epsilon > 0$ there is some $N \in \mathbb{N}$ such that for all $k > N$,

$$\left\|\mathbf{x}^{(k)} - \mathbf{v}\right\| < \epsilon.$$

Examples of a norm

Of these the most commonly used is probably the Euclidean norm.

Example: Let $\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ -4 \end{pmatrix}$.

Then

$$\|\mathbf{x}\|_2 = \sqrt{1^2 + 2^2 + 3^2 + (-4)^2} = \sqrt{30},$$

$$\|\mathbf{x}\|_\infty = 4,$$

$$\|\mathbf{x}\|_1 = |1| + |2| + |3| + |-4| = 10.$$

The particular norm which we choose will depend upon the nature of the problem.

Stopping Criteria

Having chosen a norm, we must also choose a stopping criterion for an algorithm. Intuitively, stop when the difference between successive approximations $\mathbf{x}^{(k)}, \mathbf{x}^{(k+1)}$ is sufficiently small.
There are 2 commonly used stopping criteria: we have an iterative procedure for solving $A\mathbf{x} = \mathbf{b}$ generating iterates $\mathbf{x}^{(k)}$, and a norm $\| \cdot \|$.
Let $\epsilon > 0$ be a small positive number.

1. First criterion: stop when the difference between successive iterates satisfies
$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \epsilon.$$
2. Second criterion: stop when the difference between successive iterates satisfies
$$\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k+1)}\|} < \epsilon.$$

(And, just in case, impose an upper bound on the number of iterations ...)

Jacobi iteration method

Jacobi's iteration method is as follows. We consider a linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ \vdots \quad & \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned}$$

This system can be rewritten in the following form

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}\left(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n\right), \\ x_2 &= \frac{1}{a_{22}}\left(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n\right), \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}\left(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}\right). \end{aligned}$$

Jacobi iteration method

If we have an initial approximation $x_i^{(0)}$, $i = 1, 2, ..., n$ for the solution, we generate another approximation $x_i^{(1)}$, $i = 1, 2, ..., n$ by

$$x_i^{(1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(0)} \right].$$

Having obtained $x_i^{(1)}$ we iterate to obtain $x_i^{(2)}$

$$x_i^{(2)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(1)} \right].$$

In this manner, we can generate a sequence, $\left\{ \mathbf{x}^{(k)} \right\} = \left\{ \left(x_i^{(k)} \right) \right\}$ of approximations which, if the system is suitably well behaved, converges to the true solution.
We can use $\mathbf{x}^{(\mathbf{0})} = \mathbf{0}$ as a first approximation, *ie* take $x_i^{(0)} = 0, \; i = 1, ..., n$. This leads to the following algorithm.

Algorithm

Jacobi iteration.
Let $\mathbf{x}^{(0)}$ be an initial approximation to the solution. The Jacobi iterates are given by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)} \right]$$

for $k = 0, 1, 2, 3 \ldots$
We stop when either

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \epsilon,$$

or

$$\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k+1)}\|} < \epsilon.$$

End of Algorithm
The most obvious question to ask is does this actually work? Will the sequence of iterates $\left\{ \left(x_i^{(k)} \right) \right\}$ converge to the true solution?

Convergence of Jacobi method

The answer is yes, provided the system is 'well behaved'. A sufficient condition is *diagonal dominance*.
Recall: a square matrix $\mathbf{A}$ is diagonally dominant if the following inequality is satisfied for each $i = 1, \ldots, n$:

$$|a_{ii}| > \sum_{j=1, j \neq i}^{n} |a_{ij}|.$$

Theorem: Let $A\mathbf{x} = \mathbf{b}$ be a system of linear equations, and let A be diagonally dominant. Then given any starting values $x_i^{(0)}$, the Jacobi iterates $x_i^{(k)}$ will converge to a solution of the system.

Jacobi iteration method

Example: Solve the following linear system by Jacobi iteration, using the criterion $\|x^{(k+1)} - x^{(k)}\|_\infty < 3 \times 10^{-5}$.

$$\begin{aligned} 3x_1 + x_2 + x_3 &= 5 \\ 2x_1 + 6x_2 + x_3 &= 9 \\ x_1 + x_2 + 4x_3 &= 6 \end{aligned}$$

Solution: It is easy to show that the system is diagonally dominant. We have $3 > 1 + 1, 6 > 2 + 1, 4 > 1 + 1$. To use Jacobi iteration, we rewrite the system in the form

$$\begin{aligned} x_1 &= \frac{1}{3}[5 - x_2 - x_3] \\ x_2 &= \frac{1}{6}[9 - 2x_1 - x_3] \\ x_3 &= \frac{1}{4}[6 - x_1 - x_2] \end{aligned}$$

Jacobi iteration method

$$x_1 = \frac{1}{3}[5 - x_2 - x_3]$$
$$x_2 = \frac{1}{6}[9 - 2x_1 - x_3]$$
$$x_3 = \frac{1}{4}[6 - x_1 - x_2]$$

This leads to the iterative scheme

$$x_1^{(k+1)} = \frac{1}{3}\left[5 - x_2^{(k)} - x_3^{(k)}\right]$$
$$x_2^{(k+1)} = \frac{1}{6}\left[9 - 2x_1^{(k)} - x_3^{(k)}\right]$$
$$x_3^{(k+1)} = \frac{1}{4}\left[6 - x_1^{(k)} - x_2^{(k)}\right]$$

Here $k = 0, 1, 2, \dots$

For the initial values we have $x_1^{(0)} = 0, x_2^{(0)} = 0, x_3^{(0)} = 0$.

So the next values are $x_1^{(1)} = 5/3, x_2^{(1)} = 9/6, x_3^{(1)} = 6/4$.

Jacobi iteration method

For our initial approximation, we take $x_1^0 = x_2^0 = x_3^0 = 0$. This gives

$$\begin{aligned} x_1^{(1)} &= \frac{1}{3}[5 - 0 - 0] = \frac{5}{3} \\ x_2^{(1)} &= \frac{1}{6}[9 - 2 \times 0 - 0] = \frac{9}{6} \\ x_3^{(1)} &= \frac{1}{4}[6 - 0 - 0] = \frac{6}{4} \end{aligned}$$

On the next iteration,

$$\begin{aligned} x_1^{(2)} &= \frac{1}{3}\left[5 - \frac{9}{6} - \frac{6}{4}\right] = 0.666667 \\ x_2^{(2)} &= \frac{1}{6}\left[9 - 2 \times \frac{5}{3} - \frac{6}{4}\right] = 0.694445 \\ x_3^{(2)} &= \frac{1}{4}\left[6 - \frac{5}{3} - \frac{9}{6}\right] = 0.708333 \end{aligned}$$

Jacobi iteration

Continuing this we eventually get

$$x_1^{(20)} = 0.999991, x_2^{(20)} = 0.999992,$$

$$x_3^{(20)} = 0.999992.$$

This satisfies the stopping criterion, so the algorithm terminates after twenty iterations. In fact the true solution is $x_1 = x_2 = x_3 = 1$.

Gauss-Seidel iteration

- Jacobi iteration can be slow. One way is a procedure known as Gauss-Seidel iteration.
- The idea behind Gauss-Seidel iteration is simple.
  Let us assume that we are performing Jacobi iteration. When we carry out the kth iteration, the first step is to calculate a new approximation for the first variable, x_1.
- This new approximation is of course $x_1^{(k+1)}$. Having calculated $x_1^{(k_1)}$, we next calculate $x_2^{(k+1)}$.
- But notice that when we calculate $x_2^{(k+1)}$, we use the "old value" $x_1^{(k)}$ rather than the new value $x_1^{(k+1)}$ which we have just obtained.
- And this is true for every variable x_i. We don't use any of the new values $x_i^{(k+1)}$ until the next iteration.

Gauss-Seidel iteration

- By contrast, in Gauss-Seidel iteration, we use the most recent estimate $x_i^{(k+1)}$ as soon as we have obtained it.
- Let us reconsider the previous problem
- **Example**: The Jacobi iterative scheme was

$$x_1^{(k+1)} = \frac{1}{3}\left[5 - x_2^{(k)} - x_3^{(k)}\right]$$
$$x_2^{(k+1)} = \frac{1}{6}\left[9 - 2x_1^{(k)} - x_3^{(k)}\right]$$
$$x_3^{(k+1)} = \frac{1}{4}\left[6 - x_1^{(k)} - x_2^{(k)}\right]$$

For Gauss-Seidel iteration, we adjust this to

$$x_1^{(k+1)} = \frac{1}{3}\left[5 - x_2^{(k)} - x_3^{(k)}\right]$$
$$x_2^{(k+1)} = \frac{1}{6}\left[9 - 2x_1^{(k+1)} - x_3^{(k)}\right]$$
$$x_3^{(k+1)} = \frac{1}{4}\left[6 - x_1^{(k+1)} - x_2^{(k+1)}\right]$$

Gauss-Seidel iteration

$$\begin{aligned}
x_1^{(k+1)} &= \frac{1}{3}\left[5 - x_2^{(k)} - x_3^{(k)}\right] \\
x_2^{(k+1)} &= \frac{1}{6}\left[9 - 2x_1^{(k+1)} - x_3^{(k)}\right] \\
x_3^{(k+1)} &= \frac{1}{4}\left[6 - x_1^{(k+1)} - x_2^{(k+1)}\right]
\end{aligned}$$

So we obtain $x_1^{(k+1)}$ from the first equation, we then use this new value in the second equation to obtain $x_2^{(k+1)}$. Then both $x_1^{(k+1)}$ and $x_2^{(k+1)}$ are used to find $x_3^{(k+1)}$.

$$\begin{aligned}
x_1^{(1)} &= \frac{1}{3}[5 - 0 - 0] = \frac{5}{3} \\
x_2^{(1)} &= \frac{1}{6}\left[9 - 2 \times \frac{5}{3} - 0\right] = 0.944444 \\
x_3^{(1)} &= \frac{1}{4}\left[6 - \frac{5}{3} - 0.944444\right] = 0.847222
\end{aligned}$$

Gauss-Seidel iteration

- After 8 iterations we get $x_1^{(8)} = 1.00000, x_2^{(8)} = 1.00000, x_3^{(8)} = 1.00000$. This certainly satisfies the stopping criterion.
- This is much faster convergence than with ordinary Jacobi iteration. In fact with this example, on the fifth iteration we already have $x_1^{(5)} = 0.999953, x_2^{(5)} = 1.00003, x_3^{(5)} = 1.00000$.
- Note that the convergence rates will differ for each variable.
- **Algorithm** Gauss-Seidel Iteration.
  Let $\mathbf{x}^0$ be an initial approximation vector to the solution of the linear system. Then the Gauss-Seidel iterative process can be expressed as follows

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_i^{(k)} \right]$$

End of algorithm

Gauss-Seidel iteration

- Gauss-Seidel iteration can be shown to converge under similar conditions to Jacobi iteration.
  Given any initial approximation $\mathbf{x}^0$, the Gauss-Seidel iteration process for solving $A\mathbf{x} = \mathbf{b}$ will converge to the true solution if A is diagonally dominant.
- **Note**. The condition that the system $A\mathbf{x} = \mathbf{b}$ be diagonally dominant is a *sufficient* condition for convergence of the Gauss-Seidel and Jacobi algorithms.
  It is not a **necessary** condition.
- There are linear systems which are not diagonally dominant, but Gauss-Seidel iteration will still converge to the true solution $\mathbf{x}^*$, provided the starting approximation is sufficiently close to $\mathbf{x}^*$.

Relaxation method

- The idea behind relaxation methods is that instead of directly substituting the latest values obtained by an iterative method, we substitute the values according to some **weighting**.
- There are two main relaxation techniques.
  Successive over relaxation **SOR** and
  Successive under relaxation **SUR** .
- Successive over relaxation is a technique which is used frequently in schemes for the numerical solution of partial differential equations.

Relaxation method

- From the example of Gauss-Seidel iteration we saw that the speed at which the iteration converges is not uniform for every variable.
- At any given iteration, some of the values $x_i^{(k)}$ will be more accurate than the others.
- This suggests that we ought to find some method which allows us to "update some variables more than other variables."
- Relaxation methods do exactly this.
  The idea behind relaxation methods is to introduce a parameter, called the relaxation parameter, which acts as a kind of weighting on the variables.
- The algorithm is a generalisation of the Gauss-Seidel method.

Relaxation method

Algorithm The relaxation algorithm. For a linear system, $A\mathbf{x} = \mathbf{b}$ we start with an initial approximation $\mathbf{x}^0$ to the solution. We then update the approximation according to the iterative scheme

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right]$$

End of algorithm

1. One important distinction between Gauss-Seidel and the relaxation method is, that the equation for $x_i^{(k+1)}$ depends upon $x_i^{(k)}$. Compare with, where $x_i^{(k+1)}$ depends only on the variables $x_j^{(k)}$ with $j \neq i$.
2. The variable ω is known as the relaxation parameter. The method is called successive under-relaxation (SUR) if $0 < \omega < 1$ and successive over-relaxation (SOR) for $1 < \omega < 2$. If $\omega = 1$ the algorithm reduces to Gauss-Seidel iteration.

Relaxation method

- It is known that the algorithm diverges for
  $\omega > 2$.
  It is also known that there is an optimal value for ω
  That is, a value for which the algorithm converges to the true solution most rapidly.
- There are in fact ways of calculating this optimal value, but they are extremely laborious and it is generally not worth the effort.
- In practice the parameter ω is chosen through trial and error. Note that this is, unfortunately, not uncommon in numerical analysis.
- There exist numerous numerical schemes which have to be tweaked in some way, in order to optimize their performance.

Relaxation method

Example

Let us solve the following linear system by relaxation with $\omega = 0.7$ and $\omega = 1.1$

$$\begin{aligned} -5x_1 - x_2 + 2x_3 &= 1 \\ 2x_1 + 6x_2 - 3x_3 &= 2 \\ 2x_1 + x_2 + 7x_3 &= 32 \end{aligned}$$

Solution: As usual we take $x_i^{(0)} = 0, \; i = 1, 2, 3$
The relaxation scheme can be written as

$$\begin{aligned} x_1^{(k+1)} &= x_1^{(k)} - \frac{\omega}{5}\left(1 + 5x_1^{(k)} + x_2^{(k)} - 2x_3^{(k)}\right) \\ x_2^{(k+1)} &= x_2^{(k)} + \frac{\omega}{6}\left(2 - 2x_1^{(k+1)} - 6x_2^{(k)} + 3x_3^{(k)}\right) \\ x_3^{(k+1)} &= x_3^{(k)} + \frac{\omega}{7}\left(32 - 2x_1^{(k+1)} - x_2^{(k+1)} - 7x_3^{(k)}\right) \end{aligned} \tag{1}$$

Relaxation method

With $\omega = 0.7$ we get

$$x_1^{(1)} = 0 - \frac{0.7}{5}(1 + 0 + 0 - 0) = -0.14$$

$$x_2^{(1)} = 0 + \frac{0.7}{6} \times (2 - 2(-0.14) - 6 \times 0 + 3 \times 0) = 0.266$$

$$x_3^{(1)} = 0 + \frac{0.7}{7}(32 - 2(-0.14) - 0.266 - 7 \times 0) = 3.2014$$

$$x_1^{(2)} = -0.14 - \frac{0.7}{5} \times$$
$$(1 + 5(-0.14) + .266 - 2(3.2014)) = 0.677152$$

$$x_2^{(2)} = 0.266 + \frac{0.7}{6} \times$$
$$(2 - 2(-0.14) - 6(0.266) + 3(3.2014)) = 1.27562$$

$$x_3^{(2)} = 3.2014 + \frac{0.7}{7} \times$$
$$(32 - 2(-0.14) - 0.266 - 7(3.2014)) = 3.89743$$

Relaxation method

- The process converges after ten iterations.
  We get
  $$x_1^{(10)} = 1.00029, x_2^{(10)} = 1.99961, x_3^{(10)} = 3.99993.$$
- The exact solution is $x_1 = 1, x_2 = 2, x_3 = 4$.
  If instead we had taken $\omega = 1.1$ then the process converges after 12 iterations.
- We have for this value of ω
  $x_1^{(12)} = 1.00012, x_2^{(12)} = 2.00005, x_3^{(12)} = 3.99994$.
- The optimal value for ω is $\omega \approx 0.88$.
- For this value we get convergence after seven iterations, with $x_1^{(7)} = 1.00016, x_2^{(7)} = 1.99982, x_3^{(7)} = 3.99997$.
- For $\omega > 1.4$ the relaxation method does not converge.

Pivoting procedures

- All numerical calculations are performed with round off error by a computer. Lets assume our computer can keep only 3 significant figures

$$0.435 + 0.00132 = 0.436$$

- Because of this rounding error we are loosing last digits
- Let's assume that we are row reducing 300×300 linear system. This would require $2n^3/3 = 18 \times 10^6$ (twenty million operations! How this round off errors can affect the final solution? This is not a simple question.
- Lets consider an example

$$\begin{cases} 0.0001x_1 & + & x_2 & = & 1, \\ x_1 & + & x_2 & = & 2. \end{cases}$$

- The augmented matrix is

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Pivoting procedures

- The augmented matrix is

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

- Using the row operation $r_2 \to r_2 - 10^4 r_1$

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 0 & -9999 & -9998 \end{bmatrix}$$

- The last equation is $-9999x_2 = -9998$. Because of the round off error we will have $-10000x_2 = -10000$; so $x_2 = 1$. The accurate value for $x_2 = 9998/9999 = 0.999899989998\ldots$.
- The difference between the two is on the fourth digit.
- So "computer" produced $x_2 = 1$. Using the back substitution $0.0001x_1 + x_2 = 1$ we deduce $x_1 = 0$, which is wrong. The numbers $x_1 = 0$ and $x_2 = 1$ do not satisfy the initial system.

Pivoting procedures

- Now lets solve the same linear system

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

- But first lets swap the rows $r_1 \leftrightarrow r_2$

$$\begin{bmatrix} 1 & 1 & 2 \\ 0.0001 & 1 & 1 \end{bmatrix}$$

- Using the row operation $r_2 \to 10^4 r_2 - r_1$

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 9999 & 9998 \end{bmatrix}$$

  So the computer will produce $x_2 = 9998/9999 \approx 1$.
- The backward substitution $x_1 + x_2 = 2$ will lead to $x_1 = 1$ which is not that off from the exact solution $x_1 = 10000/9999 \approx 1.00010001\ldots$.
- The difference between the two is on the fourth digit.

Pivoting procedures

- So, by swapping the rows we were able to produce a "reasonably" accurate result

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

  while the initial approach produced an erroneous result

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

- Note that the exact solution is

$$\mathbf{x} = \begin{bmatrix} \frac{10000}{9999} \\ \\ \frac{9998}{9999} \end{bmatrix}$$

- This process of swapping rows is called **partial pivoting**.

Pivoting procedures

- In each step of Gaussian elimination the computer compares each pivot in a given column and swaps the rows if necessary to put the row with the largest pivot first (**partial pivoting**).
- In numerical linear algebra this procedure is called **elimination with partial pivoting**. One of the fundamental algorithms of numerical linear algebra.
- In contrast to the **partial pivoting** strategy, in a **full pivoting** or **complete pivoting** strategy the computer looks not only for the largest value in the given column but for all columns and swaps the columns as well during the elimination process.
- In the process of **full pivoting** both rows and columns are swapped.
- These leads to the sapping of the order of unknowns.
- **Full pivoting** is slower then **partial pivoting** and the partial pivoting is quite adequate for many applications.

Sensitive matrices

- Lets consider two linear systems the augmented matrices of which are given by

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1.0001 & 2 \end{bmatrix}; \qquad \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1.0001 & 2.0001 \end{bmatrix}$$

- The solution of the first is $x_1 = 2$ and $x_2 = 0$, while the solution of the second is $x_1 = 1$ and $x_2 = 1$.
- The slight change of the right hand side produced substantial change to the solution.
- Some matrices are extremely sensitive to small changes. These matrices are called **ill-conditioned matrices**.
- The determinants of such matrices are very small or very large. In this example $\det = 10^{-4}$.
- Special "balancing" numerical techniques needs to be applied to avoid numerical instabilities.
- Even balanced matrices can be sensitive to numerical methods (see partial pivoting example considered earlier $\det = 0.9999$).