Numerical linear algebra

Arne Morten Kvarving

Department of Mathematical Sciences
Norwegian University of Science and Technology

October 29 2007

Problem and solution strategy

- We want to solve the system

$$\underline{A}\,\underline{x} = \underline{b}, \quad \underline{b}, \underline{x} \in \mathbb{R}^N, \underline{A} \in \mathbb{R}^{N \times N}.$$

- From linear algebra we know that the solution is given by

$$\underline{x} = \underline{A}^{-1}\underline{b}$$

- *Very* common problem in numerics, as a problem of its own but even more often as a substep in another algorithm.
- Naming scheme:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \ddots & & \\ a_{N1} & \cdots & & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \ddots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \ddots \\ b_N \end{bmatrix}$$

Solution strategy

- The solution strategy we choose depends on the properties and the structure of the matrix $\underline{A}$.
- Stupidly simply example: $\underline{A}$ is an orthogonal matrix. We have that

$$\underline{A}^{-1} = \underline{A}^{T}$$

so we solve the system as

$$\underline{x} = \underline{A}^{T}\underline{b}.$$

Transposing a matrix is very cheap compared to finding its inverse.

- SPD matrices.
- $\underline{A} = \mathrm{diag}\,(a_{11}, a_{22}, \cdots, a_{NN})$
  Solve by

$$x_i = \frac{b_i}{a_{ii}}, \quad \forall\ i \in [1, N].$$

- Sparse matrix structure

General matrices

- For general matrices we have learned the solution strategy in Mathematics 3 (i.e. your basic linear algebra course).
- Cramer's rule - based on finding Nth order determinants. Has a computational complexity which scales as $N!$ and very prone to round off errors. Totally unapplicable in real life - just a theoretic tool.
- Gaussian elimination.

Gaussian elimination

- Gaussian elimination is a systematic elemination process which lets us put the matrix in a triangular form

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix}.$$

- With the matrix on triangular form we can easily solve the system using backward substitution.

$$\begin{aligned} 3x_1 + 5x_2 + \quad 2x_3 &= 8 \\ 8x_2 + \quad 2x_3 &= -7 \\ 6x_3 &= 3 \end{aligned}$$

Backward substitution

- The last equation only involves one unknown:

$$x_3 = \frac{3}{6} = \frac{1}{2}$$

- Since we now know the value of x_3 we can stick this into the second equation, and we again have an equation involving only one unknown:

$$x_2 = \frac{-7 - 2x_3}{8} = \frac{-7 - 1}{8} = -1.$$

- The same procedure for the first equations yields

$$x_1 = \frac{8 - 2x_3 - 5x_2}{3} = \frac{8 - 1 + 5}{3} = 4.$$

- This is a systematic process, suited for implementation on a computer.

Obtaining the triangular form

- As before our system is of the form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 \quad &+ \cdots + a_{1N}x_N \quad = b_1 \\ a_{21}x_1 + a_{22}x_2 \quad &+ \cdots + a_{2N}x_N \quad = b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad &= \quad \vdots \end{aligned}$$

- We want to get rid of the term $a_{21}x_1$. We do that by applying the operation
  (row 2) = (row 2)-$\frac{a_{21}}{a_{11}}$(row 1).
- This gives the system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 \qquad\qquad &+ \cdots + a_{1N}x_N \qquad\qquad = b_1 \\ 0 + \left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 \quad &+ \cdots + a_{2N}x_N \quad = b_2 - \frac{a_{21}}{a_{11}}b_1 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad &= \quad \vdots \end{aligned}$$

Obtaining the triangular form

- We repeat this
  - for all rows
  - for all columns under the diagonal
- Requirement: We need $a_{kk} \neq 0$. If not, we have to reorder the rows - this is known as *pivoting*.
- Additionally, we would like $a_{kk} >> 0$ to avoid cancellation errors. We minimize this problem by always choosing the row with the largest a_{kk} - this is known as *partial pivoting*.

LU factorization

- This is Gaussian elimination (almost) like Matlab does it, and in general this is the way it is implemented in most software used today.
- We want to find two matrices $\underline{L}$ and $\underline{U}$ such that

$$\underline{A} = \underline{L}\,\underline{U}\,.$$

  where $\underline{L}$ is lower triangular and $\underline{U}$ is upper triangular.
- Why? Because

$$\begin{aligned} \underline{A}\,\underline{x} &= \underline{b} \\ \underline{L}\,\underline{U}\,\underline{x} &= \underline{b} \Rightarrow \\ \underline{L}\,\underline{v} &= \underline{b} \\ \underline{U}\,\underline{x} &= \underline{v} \end{aligned}$$

  where we can solve the two last equations using forward substitution for the first and then backward substitution for the second.

Doolittle's method

Doolittle: Choose ones on the diagonal of $\underline{L}$:

- For an example system of dimension 2;

$$\underline{A} = \begin{bmatrix} 2 & 3 \\ 8 & 5 \end{bmatrix} = \begin{bmatrix} 1 & \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ & u_{22} \end{bmatrix} = \underline{L}\,\underline{U}$$

This yields

$$\begin{aligned} u_{11} = 2, u_{12} = 3 \\ 2l_{21} = 8 \Rightarrow l_{21} = 4 \\ 4 \cdot 3 + u_{22} = 5 \Rightarrow u_{22} = -7, \end{aligned}$$

that is

$$\underline{L} = \begin{bmatrix} 1 & \\ 4 & 1 \end{bmatrix}, \underline{U} = \begin{bmatrix} 2 & 3 \\ & -7 \end{bmatrix}$$

Doolittles metode - Algorithm

- We apply the following steps

$$u_{1k} = a_{1k} \qquad k = 1, \cdots, N$$

$$l_{j1} = \frac{a_{j1}}{u_{11}} \qquad j = 2, \cdots, N$$

$$u_{jk} = a_{jk} - \sum_{s=1}^{j-1} l_{js} u_{sk} \qquad k = j, \cdots, N, j \geq 2$$

$$l_{jk} = \frac{1}{u_{kk}} \left(a_{jk} - \sum_{s=1}^{k-1} l_{js} u_{sk} \right) \qquad j = k+1, \cdots, N, k \geq 2$$

Crout's method

Crout: Choose ones on the diagonal of $\underline{U}$:

- For an example system of dimension 2 this reads

$$\underline{A} = \begin{bmatrix} 2 & 3 \\ 8 & 5 \end{bmatrix} = \begin{bmatrix} l_{11} & \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} 1 & u_{12} \\ & 1 \end{bmatrix} = \underline{L}\,\underline{U}$$

This yields

$$\begin{aligned} l_{11} = 2, l_{21} = 8 \\ 2u_{12} = 3 \Rightarrow u_{12} = \frac{3}{2} \\ 8 \cdot \frac{3}{2} + l_{22} = 5 \Rightarrow l_{22} = -7, \end{aligned}$$

that is

$$\underline{L} = \begin{bmatrix} 2 & \\ 8 & -7 \end{bmatrix}, \underline{U} = \begin{bmatrix} 1 & \frac{3}{2} \\ & 1 \end{bmatrix}$$

LU factorization

- Both of these factorizations are *unique*.
- Some matrices cannot be LU factorized. In that case we need to apply row exchange operations.

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- The obvious benefit of doing Gaussian elimination in this way is that we can calculate $\underline{L}$ and $\underline{U}$ once, and then use them to solve the problem for many different $\underline{b}$. In its usual form, Gaussian elimination requires us to modify the vector $\underline{b}$ prior to finding the solution.

Cholesky's method

- We now have a symmetric, positive definite matrix (SPD) matrix $\underline{A}$, that is
  - $\underline{x}^T \underline{A} \underline{x} > 0 \quad \forall\, \underline{x} \neq \underline{0}$.
  - $\underline{A}$ is symmetric, i.e. $\underline{A} = \underline{A}^T$.
  - This is the same as saying that all the eigenvalues of $\underline{A}$ are real and positive.
  - We can then choose $\underline{U} = \underline{L}^T$! With a general diagonal.

Cholesky's method - example

- We use the matrix
$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$
The matrix is obviously symmetric and its eigenvalues are given by $\lambda_1 = 1, \lambda_2 = 3$. So this matrix is definitely SPD.
- We want to find
$$\underline{A} = \underline{L}\,\underline{L}^T = \begin{bmatrix} a & \\ b & c \end{bmatrix} \begin{bmatrix} a & b \\ & c \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Cholesky's methode - example

- We decide the values for a, b, c by

$$a^2 = 2 \Rightarrow a = \sqrt{2}$$

$$ab = 1 \Rightarrow b = \frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}$$

$$b^2 + c^2 = 2 \Rightarrow c = \sqrt{2 - \frac{1}{2}} = \frac{\sqrt{6}}{2}$$

Cholesky's method - algorithm

- We do the following steps

$$l_{11} = \sqrt{a_{11}} \tag{1}$$

$$l_{j1} = \frac{a_{j1}}{l_{11}} \tag{2}$$

$$l_{jj} = \sqrt{a_{jj} - \sum_{s=1}^{j-1} l_{js}^2} \quad j = 2, \cdots, N \tag{3}$$

$$l_{pj} = \frac{1}{l_{jj}} \left(a_{pj} - \sum_{s=1}^{j-1} l_{js} l_{ps} \right) \quad p = j+1, \cdots, N, j \geq 2 \tag{4}$$

Cholesky's method - another example

To show how to apply these operations in actual compuations we look at the problem

$$\underline{A} = \begin{bmatrix} 4 & 2 & 14 \\ 2 & 17 & -5 \\ 14 & -5 & 83 \end{bmatrix} = \begin{bmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ & l_{22} & l_{32} \\ & & l_{33} \end{bmatrix}.$$

(1)

$$l_{11} = \sqrt{a_{11}} = 2$$

(2)

$$l_{21} = \frac{a_{21}}{l_{11}} = 1, l_{31} = \frac{a_{31}}{l_{11}} = 7$$

(3)

$$l_{22} = \sqrt{a_{22} - l_{21}^2} = \sqrt{17 - 1} = 4$$

Cholesky's method - another example

$$\underline{A} = \begin{bmatrix} 4 & 2 & 14 \\ 2 & 17 & -5 \\ 14 & -5 & 83 \end{bmatrix} = \begin{bmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ & l_{22} & l_{32} \\ & & l_{33} \end{bmatrix}.$$

(4)

$$l_{32} = \frac{1}{l_{22}} (a_{32} - l_{21} l_{31}) = \frac{1}{4} (-5 - 7 \cdot (-1)) = -3$$

(3)

$$l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = \sqrt{83 - 7^2 - (-3)^2} = 5$$

Cholesky's method

Theorem

Cholesky factorization is numerically stable.

$$a_{jj} = l_{j1}^2 + \cdots + l_{jj}^2 \quad \text{(take the square of (3))}$$

Hence any element in this sum satisifies

$$l_{jk}^2 \leq l_{j1}^2 + \cdots + l_{jj}^2 = a_{jj}$$

This means that there will be no problems with numbers growing beyond control $\Rightarrow$ small risk of cancellation errors as long as the matrix $\underline{A}$ is "kind" initially.

How to construct the inverse of a matrix

- In general, explicit construction of the inverse of a matrix is something you should *never* do, atleast not unless it is absolutely necessary.
- If we really need to do it, one way to proceed is as follows: Solve

$$\underline{A}\,\underline{x} = \underline{b} \text{ for } \underline{b} \in \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \cdots$$

- Put the obtained $\underline{x}$-vectors as columns in a matrix. This resulting matrix is exactly the inverse of $\underline{A}$.
- This is the same as solving the matrix-matrix equation

$$\underline{A}\,\underline{X} = \underline{I}$$

Evaluating a numerical method

- When we want to evaluate the efficiency of a numerical method, we need to take several aspects into consideration. These include
  - storage requirements
  - computational time requirements
  - stability
  - for parallel NUMA-computers; data locality.
- As mentioned earlier, the computational time required can be measured in (atleast) two ways:
  - Actual user time / wall clock time. This will depend strongly on the hardware involved.
  - How the amount of operations required scales with the problem size - in general this is much more interesting.

  From this point of view a better computer does not solve a problem faster, it can just solve larger problems.

Evaluating a numerical method

- It is very tempting to think that todays computers are so powerful that it is not imperative to solve things in the most efficient way possible. However, I will be as bold as to say it is even more important than before! The reason is that more powerful computers makes us able to solve more complex problems. However algorithms seldom scale linearly so we will be penalized severly, more severe than before, simply because our problems are much larger.
- Example: Gaussian elimination needs $\mathcal{O}\left(N^3\right)$ operations - OUCH!

N	T
10	1000s = 17min
100	$1 \cdot 10^6$s = 11 days
1000	$1 \cdot 10^9$s = 31 years

It is worth noting that $N = 1000$ is far from a large problem - we often solve problems with millions of unknowns.

How to construct the inverse in a smarter way

- As an example of how this works, we will find the inverse of the matrix

$$\left[\begin{array}{ccc|ccc} 3 & 5 & 2 & 1 & & \\ & 8 & 2 & & 1 & \\ & & 6 & & & 1 \end{array}\right]$$

- We scale each row in order to obtain 1 on the diagonal:

$$\left[\begin{array}{ccc|ccc} 1 & \frac{5}{3} & \frac{2}{3} & \frac{1}{3} & & \\ & 1 & \frac{1}{4} & & \frac{1}{8} & \\ & & 1 & & & \frac{1}{6} \end{array}\right]$$

How to construct the inverse in a smarter way

- We do the operation (row 1) = (row 1)-$\frac{5}{3}$(row 2)

$$\left[\begin{array}{ccc|ccc} 1 & & \frac{1}{4} & \frac{1}{3} & \frac{-5}{24} & \\ & 1 & \frac{1}{4} & & \frac{1}{8} & \\ & & 1 & & & \frac{1}{6} \end{array}\right]$$

- We do the operation (row 1) = (row 1)-$\frac{1}{4}$(row 3)

$$\left[\begin{array}{ccc|ccc} 1 & & & \frac{1}{3} & \frac{-5}{24} & \frac{-1}{24} \\ & 1 & \frac{1}{4} & & \frac{1}{8} & \\ & & 1 & & & \frac{1}{6} \end{array}\right]$$

How to construct the inverse in a smarter way

- We do the operation (row 2) = (row 2)-$\frac{1}{4}$(row 3)

$$\left[\begin{array}{ccc|ccc} 1 & & & \frac{1}{3} & \frac{-5}{24} & \frac{-1}{24} \\ & 1 & \frac{1}{4} & & \frac{1}{8} & \\ & & 1 & & & \frac{1}{6} \end{array}\right]$$

- The matrix on the right hand side is now $\underline{A}^{-1}$!
- The way we obtained it is systematic - suited for implementation on a computer.
- The method will be considerably faster than the previous proposed one.

Iterative equation solvers

- Gaussian elimination is a direct method - we obtain the "correct" solution using a fixed number of operations. The number of required operations are often too high. Another problem with (general) Gaussian elimination is that it does not exploit the structure of the matrix, such as sparseness.
- Often we do no need to solve the system of equations exactly. The reason for this can be that it is only a substep in some other algorithm where we already have several sources of errors.
- We then resort to using iterative equation solvers - indirect methods.

Fixed point iterations

- It is tempting to try the iterative schemes we have looked at previously.
- Fixed point iterations:

$$\underline{f}(\underline{x}) = \underline{0} = \underline{g}(\underline{x}) - \underline{x}$$
$$\underline{A}\,\underline{x} - \underline{b} = \underline{0} = \underline{g}(\underline{x}) - \underline{x} \Rightarrow$$
$$(\underline{A} + \underline{I})\underline{x} - \underline{b} = \underline{x} \Rightarrow \underline{x}^{k+1} = (\underline{A} + \underline{I})\underline{x}^{k} - \underline{b}$$

This will *seldom* work as the requirement of $\underline{g}$ being a constraction is the same as demanding that

$$\max_i |\lambda_i(\underline{I} + \underline{A})| < 1$$

something which is an extremely strict condition and hence very seldom fulfilled.

Newton iterations

- Newton iterations:

$$\underline{x}^{k+1} = \underline{x}^{k} - \underline{A}^{-1}\left(\underline{A}\,\underline{x}^{k} - \underline{b}\right) = \underline{A}^{-1}\underline{b}\,.$$

  But this is exactly what we try to solve in the first place!

New strategy

- We need new iterative methods that works better for linear systems of equations.
- General strategy: Split the matrix $\underline{A}$.

$$\underline{A} = \underline{D} + \underline{L} + \underline{U}$$

$$\begin{bmatrix} \ddots & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \ddots \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \vdots & 0 & 0 \\ \vdots & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 & \cdots & \vdots \\ 0 & 0 & \vdots \\ 0 & 0 & 0 \end{bmatrix}$$

Important: $\underline{L}$ and $\underline{U}$ are *NOT* the same matrices that are involved in $\underline{L}\,\underline{U}$-factorization.

Gauss-Jakobi and Gauss-Seidel iterations

- We now consider

$$\begin{aligned} \underline{A}\,\underline{x} &= \underline{b} \\ (\underline{D} + \underline{L} + \underline{U})\underline{x} &= \underline{b} \\ \underline{D}\,\underline{x} &= \underline{b} - \underline{L}\,\underline{x} - \underline{U}\,\underline{x} \Rightarrow \\ \underline{x}^{k+1} &= \underline{D}^{-1}\left(\underline{b} - \underline{L}\,\underline{x}^{k} - \underline{U}\,\underline{x}^{k}\right) \end{aligned}$$

  This is Gauss-Jakobi iterations.

- We can order the terms differently;

$$\begin{aligned} \underline{A}\,\underline{x} &= \underline{b} \\ (\underline{D} + \underline{L} + \underline{U})\underline{x} &= \underline{b} \\ (\underline{D} + \underline{L})\underline{x} &= \underline{b} - \underline{U}\,\underline{x} \Rightarrow \\ \underline{x}^{k+1} &= (\underline{D} + \underline{L})^{-1}\left(\underline{b} - \underline{U}\,\underline{x}^{k}\right) \end{aligned}$$

  This is Gauss-Seidel iterations.

The iterations stated in component form

- Gauss-Jakobi:

$$x_j^{k+1} = \frac{1}{a_{jj}} \left(b_j - \sum_{i=1, i \neq j}^{n} a_{ji} x_i^k \right)$$

- Gauss-Seidel:

$$x_j^{k+1} = \frac{1}{a_{jj}} \left(b_j - \sum_{j=i}^{j-1} a_{ji} x_i^{k+1} - \sum_{i=j+1}^{n} a_{ji} x_i^k \right)$$

- Question: What is the actual difference between these two methods?

Gauss-Jakobi and Gauss-Seidel - an example

- We consider the problem

$$\begin{bmatrix} 7 & -6 \\ -8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

- Gauss-Jakobi:

$$x_1^{k+1} = \frac{1}{7}\left(3 - a_{12}x_2^k\right) = \frac{6}{7}x_2^k + \frac{3}{7}$$
$$x_2^{k+1} = \frac{1}{9}\left(-4 - a_{21}x_1^k\right) = \frac{8}{9}x_1^k - \frac{4}{9}$$

- Gauss-Seidel:

$$x_1^{k+1} = \frac{1}{7}\left(3 - a_{12}x_2^k\right) = \frac{6}{7}x_2^k + \frac{3}{7}$$
$$x_2^{k+1} = \frac{1}{9}\left(-4 - a_{21}x_1^{k+1}\right) = \frac{8}{9}x_1^{k+1} - \frac{4}{9}$$

Some observations

- The actual difference between the methods is the fact that Gauss-Seidel iterations use new iteration values as soon as they are available.
- So why even consider Gauss-Jakobi iterations? - Well, in parallel implementations we want to avoid data exchange during the iterations.
- Both of these methods (as well as many more) can be stated in the form

$$\underline{M}\,\underline{x}^{k+1} = \underline{N}\,\underline{x}^{k} + b \Rightarrow$$
$$\underline{x}^{k+1} = \underline{M}^{-1}\underline{N}\,\underline{x}^{k} + \underline{M}^{-1}\underline{b}$$

- Gauss-Jakobi has $\underline{M} = \underline{D}\,, \underline{N} = -(\underline{U} + \underline{L}\,)$.
- Gauss-Seidel has $\underline{M} = \underline{D} + \underline{L}\,, \underline{N} = -\underline{U}\,$.

Convergence of the methods

- First : By convergence we here mean that the sequence

$$\underline{x}^0, \underline{x}^1, \cdots, \underline{x}^k$$

converges to the exact solution of $\underline{A}\,\underline{x} = \underline{b}$.

- A very common way to measure the error is using the *residual* error.

$$\underline{r} = \underline{b} - \underline{A}\,\underline{x}^k$$

- Here we use another definition, namely the *displacement* error

$$\underline{d}^k = \underline{x}^k - \underline{x}\,.$$

The problem of convergence can now be stated as: What is the value of $\underline{d}^{k+1}$ given $\underline{d}^k$?

Convergence of the methods

- We insert our iterative scheme into the definition of the displacement error and get

$$\begin{aligned}\underline{d}^{\,k+1} = \underline{x}^{\,k+1} - \underline{x} &= \underline{M}^{-1}\underline{N}\,\underline{x}^{\,k} + \underline{M}^{-1}\underline{b} - \left(\underline{M}^{-1}\underline{N}\,\underline{x} + \underline{M}^{-1}\underline{b}\right) \\ &= \underline{M}^{-1}\underline{N}\left(\underline{x}^{\,k} - \underline{x}\right) = \underline{M}^{-1}\underline{N}\,\underline{d}^{\,k}\end{aligned}$$

- We now apply this recursively to obtain

$$\underline{d}^{\,k} = \left(\underline{M}^{-1}\underline{N}\right)^{k}\underline{d}^{\,0}.$$

- For convergence we need "$\underline{M}^{-1}\underline{N} < 1$" in some sense. We need a measure for the size of a matrix - this measure is known as a *norm*.

Matrix norms

- There are several ways to measure the size of a matrix.
- They have a rigorous definition which we skip here.
- The Frobenius norm:

$$\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$$

- The one-norm - largest column sum

$$\|A\|_1 = \max_i \sum_j |a_{ij}|$$

- The two-norm - largest row sum

$$\|A\|_2 = \max_j \sum_i |a_{ij}|$$

Convergence of the methods

- Two results for norms which we use here are
  - Over $\mathbb{R}^{N \times N}$ all norms are equivalent. That is, for any pair of norms, $\|\cdot\|_a, \|\cdot\|_b$ we can find constants c_1 and c_2 such that

$$c_1 \|\cdot\|_a \leq \|\cdot\|_b \leq c_2 \|\cdot\|_a.$$

  - The spectral radius of a matrix $\underline{A}$ is defined as

$$\rho(\underline{A}) = \max_i |\lambda_i|$$

    where λ_i is the i'th eigenvalue of the matrix $\underline{A}$. We then have the result that

$$\rho(\underline{A}) = \inf_{\|\cdot\|} \|A\|,$$

    that is the spectral radius is the smallest measure of the size of a matrix we can find.

Convergence of the methods

- Since all norms are equivalent, convergence in one norm implies convergence in all others.
- Hence it is sufficient to consider the norm with the smallest value - we can consider the spectral radius of the matrix.
- Our condition for convergence hence reads

$$\lim_{k \to \infty} \underline{d}^{\,k} = \underline{0} \Leftrightarrow \rho(\underline{M}^{-1}\underline{N}) < 1$$