4 Iterative methods

4.1 What a two year old child can do

Suppose we want to find a number x such that $\cos x = x$ (in radians). This is a nonlinear equation, there is no "explicit" solution. Nevertheless a two-year old child can solve it with a calculator: just push the COS button many times... Suppose that $x_0 = 0$ is the initial number on the display, and $x_1 = \cos x_0$, $x_2 = \cos x_1$ etc. are the successive numbers. This is what you see

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 0.5403 \\ x_3 &= 0.857 \\ x_4 &= 0.654 \\ x_5 &= 0.793 \\ x_6 &= 0.701 \\ &\vdots \\ x_{20} &= 0.738 \\ x_{21} &= 0.739 \\ x_{22} &= 0.739 \end{aligned}$$

and you see that at least the first three digits get stabilized. This will be the approximative solution to $\cos x = x$. We also say that $x \approx 0.739$ is a **fixed point** of the function $f(x) = \cos x$. In general the fixed point of a function f is a value x such that $f(x) = x$.

Of course the method does not always work. Try to find a nonzero root of $x^2 = x$ with this method. Unless you start exactly from $x = 1$, your iteration will either blow up or converge to zero. This means that $x = 0$ is a stable solution, while $x = 1$ is unstable. But in this case you at least got a root...

Try now $\frac{1}{x^2} = x$. Here the situation is worse: the iteration blows up whatever close you started from the true root $x = 1$, unless if you started exactly from the root. This is because $x = 1$ is an unstable root of this equation.

Finally, try $\frac{1}{x} = x$. Here nothing bad happens, just the procedure will never converge, the iteration alternates between two numbers: x_0 and $x_1 = 1/x_0$. In this case the algorithm ended up in an infinite cycle.

The conclusion is that whenever we try to solve an equation of the form

$$f(x) = x$$

(**fixed-point equation**), one can try the two-year-old-child stategy: start from some x_0 and iterate, i.e. generate the sequence $x_1 = f(x_0)$, $x_2 = f(x_1)$ $x_3 = f(x_2)$ etc. If the sequence stabilizes (converges), then you found a solution. This very simple method is extremely useful, if it works...

But we should emphasize, that if the sequence does not converge, then no conclusion can be made! You cannot say that the equation has no solution! Moreover, even if you found a solution, there could be other solutions. Some of them could be accessible by the same method starting from a different initial value x_0, some of them may not be accessible at all. It also could happen that for some initial value the iteration converges, for some other values does not.

Whether the iteration converges or not is basically determined the derivative of f at the fixed point. If $|f'(x_{fix})| < 1$, then the iteration converges if you start it from a point close enough to the solution. The reason is roughly the following: From the iteration we have

$$x^{(n)} = f(x^{(n-1)})$$

and from the fixed point equation

$$x_{fix} = f(x_{fix}).$$

Subtract these equations from each other

$$x^{(n)} - x_{fix} = f(x^{(n-1)}) - f(x_{fix})$$

Now use Taylor expansion around x_{fix}:

$$f(x^{(n-1)}) \approx f(x_{fix}) + f'(x_{fix})\Big(x^{(n-1)} - x_{fix}\Big)$$

if $x^{(n-1)}$ is close to x_{fix}. Hence, from these two last equations:

$$|x^{(n)} - x_{fix}| \approx |f'(x_{fix})||x^{(n-1)} - x_{fix}|$$

Hence, if $|f'(x_{fix})| < 1$, then we see that $x^{(n)}$ is closer to the fixpoint than $x^{(n-1)}$. After n iterations

$$|x^{(n)} - x_{fix}| \approx |f'(x_{fix})|^n |x^{(0)} - x_{fix}|$$

i.e. we see that the procedure *converges exponentially fast* if the number $|f'(x_{fix})|$ is strictly smaller than 1.

Moreover, we can give an estimate on the speed of convergence. We see that after n step the deviation of the approximate solution from the true one is reduced by a factor of $|f'(x_{fix})|^n$. Hence if we want a certain precision ε, then we need at least n steps, where

$$\varepsilon \geq |f'(x_{fix})|^n |x^{(0)} - x_{fix}|$$

i.e.

$$n \geq \frac{\log \varepsilon}{\log |f'(x_{fix})|}$$

if the error of the initial guess is at most of order one, i.e. one can assume that

$$|x^{(0)} - x_{fix}| \leq 1$$

In this formula we can use logarithm of any base.

For example if $|f'(x_{fix})| = 0.8$, the error of the initial guess was smaller than 1, and we want 10^{-8} precision, then we need at least

$$n \geq \frac{\lg 10^{-8}}{\lg 0.8} = \frac{-8}{-0.0969} = 82.55$$

To be on the safe side, one usually overestimates this number by one or two, so we need around 83-84 iterations.

The argument above is not quite correct, since we neglected the higher order terms in the Taylor expansion. If you start the iteration from a point close enough to the fixed point, then these higher order terms are really negligible, otherwise they could ruin the convergence. Recall that a similar phenomenon was found at the Newton's iteration.

For a more rigorous discussion and nice pictures generated by a java applet, see

`http://www.math.gatech.edu/ bourbaki/1501/html/pdf/fixedpoints.pdf`

Finally notice that any equation of the form $F(x) = 0$ can be brought into a "fixed-point" equation just by writing it as

$$x - F(x) = x \ .$$

Then the solution to $F(x) = 0$ is equivalent to the fixed point of $f(x) := x - F(x)$.

4.2 Iterative methods for $A\mathbf{x} = \mathbf{b}$

Now we apply the same idea to solve $A\mathbf{x} = \mathbf{b}$. Again, for simplicity, we assume that A is a regular square matrix, hence there is a unique solution.

We have to bring the equation into a fixed-point equation and we do it by splitting the matrix $A = B + (A - B)$ with some cleverly chosen matrix B. The way to think about it is that B is the "main part" which will be chosen a "nice" matrix, and we view A as a "small perturbation" (by $A - B$) of this nice matrix B.

Clearly $A\mathbf{x} = \mathbf{b}$ is equivalent to

$$B\mathbf{x} = (B - A)\mathbf{x} + \mathbf{b}$$

or

$$\mathbf{x} = B^{-1}\Big[(B - A)\mathbf{x} + \mathbf{b}\Big] = (I - B^{-1}A)\mathbf{x} + B^{-1}\mathbf{b}$$

at least if B is invertible (hence we will have to choose B such that B^{-1} be simple).

The algorithm is very simple: start with an initial vector $\mathbf{x}^{(0)}$, and iteratively generate

$$\mathbf{x}^{(n)} := (I - B^{-1}A)\mathbf{x}^{(n-1)} + B^{-1}\mathbf{b}$$

and hope for the best...

How to predict the convergence? At least is there some sufficient condition for convergence?

Let $\delta\mathbf{x}^{(n)} := \mathbf{x} - \mathbf{x}^{(n)}$ be the error after the n-th iteration. From $A\mathbf{x} = \mathbf{b}$ and $B\mathbf{x}^{(n)} = (B - A)\mathbf{x}^{(n-1)} + \mathbf{b}$ we easily see that

$$B(\delta\mathbf{x}^{(n)}) = (B - A)(\delta\mathbf{x}^{(n-1)})$$

i.e.

$$\delta\mathbf{x}^{(n)} = (I - B^{-1}A)(\delta\mathbf{x}^{(n-1)}) = (I - B^{-1}A)^n(\delta\mathbf{x}^{(0)})$$

where $\delta\mathbf{x}^{(0)}$ is the difference of the initial value from the solution. Hence

$$\|\delta\mathbf{x}^{(n)}\| \leq \|(I - B^{-1}A)^n\| \, \|\delta\mathbf{x}^{(0)}\| \tag{4.1}$$

It is clear that if the norm of the matrix $(I - B^{-1}A)^n$ converges to zero, then the iteration converges. Moreover, this norm gives an estimate on the speed of convergence.

We need the following theorem:

Theorem 4.1 *Let H be a square matrix. Then $\|H^n\| \to 0$ if and only if all eigenvalues of H are less than one in absolute value.*

Proof: The theorem intuitively is clear; the eigenvalues are responsible for amplification of vectors. However the rigorous proof is not trivial for general matrices, so we restrict ourselves to the symmetric case.

If H is symmetric, then by the spectral theorem it can be written as $H = QDQ^t$ with some orthogonal matrix Q and diagonal matrix D. Clearly $H^n = QD^nQ^t$, and $\|H^n\| = \|QD^nQ^t\| = \|D^n\|$ (PROVE IT(*)). Since D contains the eigenvalues of H in the diagonal, the eigenvalues of D^n are just the n-th power of the eigenvalues of H. The biggest of them clearly goes to zero if and only if the biggest eigenvalue of H is smaller than one in absolute value. □

As a conclusion, we have

Theorem 4.2 *If A, B are both regular square matrices, then the iteration $B\mathbf{x}^{(n)} = (B - A)\mathbf{x}^{(n-1)} + \mathbf{b}$ converges to the the solution if all eigenvalues of $I - B^{-1}A$ are less than one in absolute value. The speed of convergence is exponential with a rate given by the largest (in modulus) eigenvalue of $I - B^{-1}A$.*

REMARK 1.: The condition is not just sufficient, but also essentially necessary. But the important part is the sufficiency; it gives a criterion to decide in advance whether it is worth running iteration or not.

REMARK 2.: The good news is that if the iteration converges, then it usually converges very fast. It means that there is a constant $c < 1$ such that

$$\|\delta\mathbf{x}^{(n)}\| \leq c^n\|\delta\mathbf{x}_0\|$$

If $I - B^{-1}A$ is symmetric, then c clearly can be chosen as its biggest eigenvalue (WHY(*)?) The nonsymmetric case is slightly harder.

It is well known that such an exponential estimate yields fast convergence even if c is very close to 1. For example, if $c = 0.99$, then $c^{1000} \approx e^{-10} \approx 4 \cdot 10^{-5}$. Also remember that it is usually very easy to implement an iterative algorithm. So this is a good algorithm whenever it works.

4.2.1 Jacobi iteration

The art of iterative methods for $A\mathbf{x} = \mathbf{b}$ is the good decomposition. The simplest method is the *Jacobi decomposition* of a square matrix $A = (a_{ij})$ as

$$A = L + D + U$$

where L is a lower triangular matrix, containing all elements of A strictly below the diagonal, U is its upper triangular counterpart and D is a diagonal matrix containing the diagonal elements of A.

Let $B = D$ and $A - B = L + U$ be the decomposition explained in Section 4.2. Since the diagonal matrix is easy to invert, we can easily write up the formulas:

$$\mathbf{x}^{(n)} = -D^{-1}(L + U)\mathbf{x}^{(n-1)} + D^{-1}\mathbf{b} \tag{4.2}$$

or in coordinates

$$x_i^{(n)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j:j\neq i} a_{ij}x_j^{(n-1)}\Big)$$

The summation is taken over all j's except $j = i$. The initial vector $\mathbf{x}^{(0)}$ can be chosen to be $\mathbf{0}$ unless we have a better a-priori guess for the true solution.

Does this work? One could use Theorem 4.2 to check the eigenvalues of $-D^{-1}(L + U)$, but this may not be so easy. The following theorem is not optimal, but it gives a very easily verifiable condition for the convergence of the Jacobi iteration:

Theorem 4.3 *Suppose that A is diagonal dominant, that is*

$$|a_{ii}| > \sum_{j:j\neq i} |a_{ij}| \tag{4.3}$$

for all i. Then the Jacobi iteration converges.

We do not give the rigorous proof of this theorem. We just remark that the condition (4.3) says that in every row the diagonal entry is bigger (in absolute value) than the sum of all the other entries (in absolute value) in the row. In other words, the matrix is "close" to a diagonal matrix in a sense that the matrix $L+U$ containing the offdiagonal terms is "smaller" that the diagonal matrix D.

The speed of convergence of the Jacobi iteration is given by the largest (in modulus) eigenvalue of $D^{-1}(L+U)$. This directly follows Theorem 4.2.

EXERCISE: Consider the matrix $A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$. Show that it does not satisfy the condition of Theorem 4.3, but the Jacobi iteration still converges. (Hint: Use Theorem 4.2).

The condition (4.3) is quite restrictive, especially for big matrices. However it could be useful for sparse matrices, especially for matrices with special structure. When partial differential equations are solved by discretization, then the arising big matrix is usually "almost diagonal" in a sense that the only nonzero entries are one or two steps away from the diagonal, e.g., for the one step case $a_{ij} = 0$ if $|i-j| > 1$ (this is also called **tridiagonal matrix**). In this case the condition (4.3) is not so restrictive, since it compares the diagonal entry a_{ii} with only two other nonzero entries $a_{i,i-1}$ and $a_{i,i+1}$.

Problem 4.4 *Find the solution to*

$$\begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$$

first with Gaussian elimination, then with Jacobi iteration up to two digits (10^{-2} precision) starting from the zero vector. Give an estimate on the number of steps to reach 8 digit precision.

SOLUTION: The true solution is $\mathbf{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ either from Gauss, or just "by looking at it".

Then first notice that the matrix $A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$ is diagonally dominant, hence Jacobi algorithm will converge from any initial vector. Let $\mathbf{x}^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and compute the next step according to the Jacobi formula:

$$x_1^{(n)} = \frac{1}{4}(6 - 1 \cdot x_2^{(n-1)})$$

$$x_2^{(n)} = \frac{1}{3}(8 - 2 \cdot x_1^{(n-1)})$$

(BE CAREFUL WITH INDICES)

We compute each number up to three digits to detect the stabilization of the second digit

$$x_1^{(1)} = \frac{1}{4}(6 - 1 \cdot 0) = \frac{3}{2} = 1.5$$

$$x_2^{(1)} = \frac{1}{3}(8 - 2 \cdot 0) = \frac{8}{3} = 2.33$$

The next iteration gives

$$x_1^{(2)} = \frac{1}{4}(6 - 1 \cdot \frac{8}{3}) = \frac{5}{6} = 0.833$$

$$x_2^{(2)} = \frac{1}{3}(8 - 2 \cdot \frac{3}{2}) = \frac{5}{3} = 1.66$$

Next:

$$x_1^{(3)} = \frac{1}{4}(6 - 1 \cdot \frac{5}{3}) = \frac{13}{12} = 1.08$$

$$x_2^{(3)} = \frac{1}{3}(8 - 2 \cdot \frac{5}{6}) = \frac{19}{9} = 2.1$$

Next

$$x_1^{(4)} = \frac{1}{4}(6 - 1 \cdot \frac{19}{9}) = \frac{35}{36} = 0.97$$

$$x_2^{(4)} = \frac{1}{3}(8 - 2 \cdot \frac{13}{12}) = \frac{35}{18} = 1.94$$

Next

$$x_1^{(5)} = \frac{1}{4}(6 - 1 \cdot 1.94) = 1.01$$

$$x_2^{(5)} = \frac{1}{3}(8 - 2 \cdot 0.97) = 2.02$$

Next

$$x_1^{(6)} = \frac{1}{4}(6 - 1 \cdot 2.02) = 0.995$$

$$x_2^{(6)} = \frac{1}{3}(8 - 2 \cdot 1.01) = 1.99$$

and

$$x_1^{(7)} = \frac{1}{4}(6 - 1 \cdot 1.99) = 1.0025$$

$$x_2^{(7)} = \frac{1}{3}(8 - 2 \cdot 0.995) = 2.003$$

Since we see that the error in the successive steps, $\|\mathbf{x}^{(6)} - \mathbf{x}^{(7)}\|$, is less than 1%, we can stop.

There are many different rules to stop the algorithm, they usually differ from each other by a few steps. The important thing is that the true solution is unknown, so the true error, $\|\mathbf{x}^{(n)} - \mathbf{x}_{true}\|$, must be estimated by the error in the successive steps, i.e. $\|\mathbf{x}^{(n)} - \mathbf{x}^{(n+1)}\|$, but usually this is reliable.

To give an estimate on the number of steps to get 10^{-8} precision, we have to compute the largest eigenvalue of

$$M := D^{-1}(L + U) = \begin{pmatrix} 4 & 0 \\ 0 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1/4 \\ 2/3 & 0 \end{pmatrix}$$

The eigenvalues are $\lambda = \pm 1/\sqrt{6}$, hence the speed of convergence is determined by $1/\sqrt{6}$ (you have to choose the one bigger in absolute value, but both eigenvalues have the same absolute value here)

This means that the absolute error gets diminished roughly (not exactly since this is not the norm) by a factor of $1/\sqrt{6}$ after every step:

$$\|\mathbf{x}^{(n)} - \mathbf{x}_{true}\| \leq \left(\frac{1}{\sqrt{6}}\right)^n \|\mathbf{x}^{(0)} - \mathbf{x}_{true}\|$$

Here $\|\mathbf{x}^{(0)} - \mathbf{x}_{true}\|$ is of order one (this is the typical case), you can even forget about its exact value, replace it by 1. So to get a 10^{-8} precision you have to solve

$$10^{-8} \geq \left(\frac{1}{\sqrt{6}}\right)^n$$

i.e. $n \geq -8 \ln 10 / \ln(1/\sqrt{6}) \approx 20.56$. As a rule of thumb you add 2-3 more steps to fix the "order 1" type argument (one can make it more precise), so to get 10^{-8} digit precision one needs approximately 22-23 steps.

4.2.2 Gauss-Seidel iteration

There are several other (a bit more refined) iterative methods, relying on slightly different decomposition of A. *Gauss-Seidel method* uses $B = L + D$, and the n-th step of the iteration is

$$\mathbf{x}^{(n)} = (L + D)^{-1}\left(\mathbf{b} - U\mathbf{x}^{(n-1)}\right)$$

or (CHECK that equivalent)

$$\mathbf{x}^{(n)} = D^{-1}\left(\mathbf{b} - L\mathbf{x}^{(n)} - U\mathbf{x}^{(n-1)}\right)$$

or, in coordinates

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

(CHECK(*) that this is really the correct formula). This is very similar to the Jacobi iteration (see (4.2)), but in many cases it works better. Roughly speaking, Jacobi method considers $A = L+D+U$ as a perturbation of its diagonal D, while Gauss-Seidel considers $A = L+D+U$ as a perturbation of $L + D$. It turns out that Gauss-Seidel is also easier to implement on a computer (WHY?).

From theoretical point of view we have the analogue of Theorem 4.3

Theorem 4.5 *The Gauss-Seidel iteration converges for diagonal dominant matrices, i.e. for matrices satisfying (4.3).*

The speed of convergence of the Gauss-Seidel iteration is given by the largest (in modulus) eigenvalue of $(L+D)^{-1}U$. This directly follows Theorem 4.2.

Problem 4.6 *Consider the same problem as in the previous section, i.e. find the solution to*

$$\begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$$

but now with Gauss-Seidel iteration up to two digits (10^{-2} precision) starting from the zero vector. Give an estimate on the number of steps to reach 8 digit precision.

SOLUTION: We just follow the formula

$$x_1^{(n)} = \frac{1}{4}(6 - 1 \cdot x_2^{(n-1)})$$

$$x_2^{(n)} = \frac{1}{3}(8 - 2 \cdot x_1^{(n)})$$

(Notice the deviation from Jacobi, the in the second line you use $x_1^{(n)}$ instead of $x_1^{(n-1)}$!)

We generate the iteration:

$$x_1^{(1)} = \frac{1}{4}(6 - 1 \cdot 0) = \frac{3}{2} = 1.5$$

$$x_2^{(1)} = \frac{1}{3}(8 - 2 \cdot \frac{3}{2}) = \frac{5}{3} = 1.66$$

Next:

$$x_1^{(2)} = \frac{1}{4}(6 - 1 \cdot \frac{5}{3}) = \frac{13}{12} = 1.08$$

$$x_2^{(2)} = \frac{1}{3}(8 - 2 \cdot \frac{13}{12}) = \frac{35}{18} = 1.94$$

Next:

$$x_1^{(3)} = \frac{1}{4}(6 - 1 \cdot 1.94) = 1.01$$

$$x_2^{(3)} = \frac{1}{3}(8 - 2 \cdot 1.01) = 1.99$$

Next

$$x_1^{(4)} = \frac{1}{4}(6 - 1 \cdot 1.99) = 1.0025$$

$$x_2^{(4)} = \frac{1}{3}(8 - 2 \cdot 1.0025) = 1.998$$

and we can stop. Notice that the iteration was faster (around twice as fast) as Jacobi.

To estimate the number of steps for large precision, we need to find the eigenvalues of

$$M = (L + D)^{-1}U = \begin{pmatrix} 4 & 0 \\ 2 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1/4 \\ 0 & -1/6 \end{pmatrix}$$

which are 0 and $-1/6$, hence the biggest absolute value is $1/6$. To compute the number of steps we solve

$$10^{-8} \geq (1/6)^n$$

i.e. $n = -8 \ln 10 / \ln(1/6) = 10.28$, hence after 12-13 steps we reach the 10^{-8} precision.

Notice that the eigenvalue of the Gauss-Seidel algorithm was the square of the corresponding eigenvalue of Jacobi algorithm, i.e., the iteration will be twice as fast. This is not true for general matrices, but it is true for tridiagonal ones.

There are several improvements of the Gauss-Seidel method, the most useful one is called *Gauss-Seidel with relaxation.* In general, relaxation is a method to avoid "overshooting". Very intuitively; suppose that you run an iterative method in one dimension, the sequence of iteratively computed numbers $x^{(n)}$ is used to approximate the true solution x. The nicest situation is when you monotonically approach to x, from one direction: $x^{(1)} \leq x^{(2)} \leq x^{(3)} \nearrow x$. Numerically this situation is the most stable. But it could happen that you approach to the

solution in an alternating way; e.g., $x^{(1)} < x$, then $x^{(2)} > x$ and then again $x^{(3)} < x$ etc., and hopefully still $|x - x^{(n)}|$ decreases to zero. Such a scheme carries the potential danger of overshooting; $x^{(2)}$ is "on the other side" of x and it could be farther away from x than $x^{(1)}$. In this case it looks wise to replace $x^{(2)}$, for example, by the average $(x^{(1)} + x^{(2)})/2$ (or some weighted average) since this is clearly closer to x. Such a procedure is called "relaxation" and in particular this idea has been carefully implemented for the Gauss-Seidel iteration.

It is worth to compare Jacobi and Gauss-Seidel iterations; this is the goal of one of the computer projects. Based upon a test on random matrices, it turns out that

(i) Diagonal dominance is a restrictive sufficient condition: these iterative methods converge for much more matrices.

(ii) In general one cannot directly compare the two methods. However, for tridiagonal matrices Jacobi and Gauss-Seidel converge or diverge simultaneously, and when they converge, then Gauss-Seidel is around twice as fast. This can also be proven rigorously.

Here are some results of a program by Nolan Leaky.

Solving 100 random 3 by 3 general system up to precision 10^{-3}

Number of diagonally dominant: 1

Number of matrices for which Jacobi converged: 12

Average number of steps: 62.57

Number of matrices for which Gauss-Seidel converged: 22

Average number of steps: 30.66

Solving 100 random 4 by 4 general system up to precision 10^{-3}

Number of diagonally dominant: 0

Number of matrices for which Jacobi converged: 2

Average number of steps: 124

Number of matrices for which Gauss-Seidel converged: 6

Average number of steps: 45.66

As you can expect, the situation for **tridiagonal** matrices is much better:

Solving 100 random 3 by 3 tridiagonal system up to precision 10^{-3}

Number of diagonally dominant: 5

Number of matrices for which Jacobi converged: 30

Average number of steps: 49.4

Number of matrices for which Gauss-Seidel converged: 30

Average number of steps: 24.4

Solving 100 random 4 by 4 system up to precision 10^{-3}

Number of diagonally dominant: 1

Number of matrices for which Jacobi converged: 14

Average number of steps: 63.13

Number of matrices for which Gauss-Seidel converged: 14

Average number of steps: 33.9