35 Lecture #24: Wednesday, May 6th, 2026

35.1 Iterative methods

We now turn to iterative methods for solving systems of linear equations. Unlike direct methods, which aim to produce the solution in a finite number of steps, iterative methods begin with an initial guess and generate a sequence of approximations which, under suitable assumptions, converges to the exact solution. As we have seen before, a natural way to construct such methods is through a fixed-point formulation, that is, by rewriting the problem in the form $\varphi(\vec{x}) = \vec{x}$.

Let us consider the linear system $A\vec{x} = \vec{b}$. A first formal way to rewrite it as a fixed-point equation is

$$A\vec{x} - \vec{b} + \vec{x} = \vec{x}.$$

If we denote by Id_n the identity matrix, so that $\vec{x} = \mathrm{Id}_n\, \vec{x}$, then this becomes

$$A\vec{x} - \vec{b} + \mathrm{Id}_n\, \vec{x} = \vec{x},$$

or equivalently

$$(A + \mathrm{Id}_n)\vec{x} - \vec{b} = \vec{x}.$$

However, this reformulation is not especially useful from a computational point of view. It should rather be viewed as a source of inspiration: instead of working directly with this expression, we introduce a more flexible fixed-point scheme of the form

$$\vec{x}_{k+1} = B\vec{x}_k + \vec{c} \tag{33}$$

where B is a matrix and $\vec{c}$ is a vector. In other words, we define the iteration map by

$$\varphi(\vec{x}) = B\vec{x} + \vec{c}$$

and then $\vec{x}_{k+1} = \varphi(\vec{x}_k)$ for every $k \in \mathbb{N}$. As in the section on fixed-point methods, the key point is to determine when the map φ is a contraction (see Definition 22.9). We are ready now for specific methods.

35.2 Jacobi iteration

Consider the following 3×3 system. Our goal is to rewrite it in fixed-point form, that is, with x_1, x_2, and x_3 isolated on the left-hand side and suitable expressions involving these variables on the right-hand side.

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 = b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 = b_2, \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 = b_3. \end{cases} \tag{34}$$

We isolate x_1, x_2 and x_3, and obtain

$$\begin{cases} x_1 = \frac{1}{a_{1,1}} \cdot \left[b_1 - (a_{1,2}x_2 + a_{1,3}x_3)\right], \\ x_2 = \frac{1}{a_{2,2}} \cdot \left[b_2 - (a_{2,1}x_1 + a_{2,3}x_3)\right], \\ x_3 = \frac{1}{a_{3,3}} \cdot \left[b_3 - (a_{3,1}x_1 + a_{3,2}x_2)\right]. \end{cases} \tag{35}$$

Now we can see them as equations of the fixed-point type. This system should be understood as an iterative rule. Starting from an initial approximation

$$\vec{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{pmatrix},$$

we compute a new approximation

$$\vec{x}^{(1)} = \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix}$$

by using the old values on the right-hand side. More precisely,

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{1,1}} \cdot \left[b_1 - \left(a_{1,2} x_2^{(k)} + a_{1,3} x_3^{(k)} \right) \right], \\ x_2^{(k+1)} = \dfrac{1}{a_{2,2}} \cdot \left[b_2 - \left(a_{2,1} x_1^{(k)} + a_{2,3} x_3^{(k)} \right) \right], \\ x_3^{(k+1)} = \dfrac{1}{a_{3,3}} \cdot \left[b_3 - \left(a_{3,1} x_1^{(k)} + a_{3,2} x_2^{(k)} \right) \right]. \end{cases}$$

Thus, all quantities on the right-hand side come from the previous step, while all quantities on the left-hand side are the new values. In other words, at each iteration we use the old approximation to compute simultaneously the next one. This is the **Jacobi iteration method**. Let us see an example.

Example 35.1. Consider the system

$$\begin{cases} 2x + y = 1, \\ x + 2y = -1. \end{cases}$$

We rewrite it in iterative form as follows

$$\begin{cases} x = \dfrac{1 - y}{2}, \\ y = \dfrac{-1 - x}{2}. \end{cases}$$

Starting with the initial guess

$$\vec{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

the Jacobi method gives

$$\begin{cases} x_{k+1} = \dfrac{1 - y_k}{2}, \\ y_{k+1} = \dfrac{-1 - x_k}{2}. \end{cases}$$

We now compute a few iterations

$$\vec{x}_1 = \begin{pmatrix} \dfrac{1-0}{2} \\ \dfrac{-1-0}{2} \end{pmatrix} = \begin{pmatrix} 1/2 \\ -1/2 \end{pmatrix}, \qquad \vec{x}_2 = \begin{pmatrix} \dfrac{1-(-1/2)}{2} \\ \dfrac{-1-(1/2)}{2} \end{pmatrix} = \begin{pmatrix} 3/4 \\ -3/4 \end{pmatrix},$$

$$\vec{x}_3 = \begin{pmatrix} \dfrac{1-(-3/4)}{2} \\ \dfrac{-1-(3/4)}{2} \end{pmatrix} = \begin{pmatrix} 7/8 \\ -7/8 \end{pmatrix}.$$

In fact, we can use induction to show that, for every $k \in \mathbb{N}$, we have that

$$x_{k+1} = \frac{2^k - 1}{2^k} \quad \text{and} \quad y_{k+1} = -\frac{2^k - 1}{2^k}.$$

This mens that the iterations converge to the matrix

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

which is indeed the exact solution of the system.

The algorithm is given as follows. Notice that the stopping condition is given at the end of it.

Algorithm 35.2 (JIM: Jacobi iteration method)**.** Given a system $A\vec{x} = \vec{b}$ of linear equations and tolerance ε we have the following steps.

0. Choose an arbitrary initial vector $\vec{x}_0$. Set $k = 0$.

1. Compute
$$(\vec{x}_{k+1})_i = -\frac{1}{a_{i,i}} \left(\sum_{j=1}^{i-1} a_{i,j}(\vec{x}_k)_j + \sum_{j=i+1}^{n} a_{i,j}(\vec{x}_k)_j \right) + \frac{b_i}{a_{i,i}}.$$

   If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geqslant \varepsilon$, increase k by one and go back to step 1.

Remark 35.3. We always can split a matrix A into its diagonal part D and its off-diagonal part $L + U$, where L is the strictly lower triangular part and U is the strictly upper triangular part. In this way, we can always write

$$A = D + L + U.$$

That is, the matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{pmatrix}$$

can be writen as

$$\underbrace{\begin{pmatrix} a_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & a_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & a_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n,n} \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{2,1} & 0 & 0 & \cdots & 0 \\ a_{3,1} & a_{3,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & 0 \end{pmatrix}}_{L} + \underbrace{\begin{pmatrix} 0 & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ 0 & 0 & a_{2,3} & \cdots & a_{2,n} \\ 0 & 0 & 0 & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}}_{U}.$$

We will use the last remark in what follows. Let us go back to the expression in step 1 in Algorithm 35.2 - one can do the same reasoning also with the expression (34) for a 3×3 situation. We have the following chain of equalities

$$(\vec{x})_{k+1} = \begin{pmatrix} \frac{1}{a_{1,1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{2,2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{a_{n,n}} \end{pmatrix} \begin{pmatrix} b_1 - \sum_{j \neq 1} a_{1,j}(\vec{x}_k)_j \\ \vdots \\ b_i - \sum_{j \neq i} a_{i,j}(\vec{x}_k)_j \\ \vdots \\ b_n - \sum_{j \neq n} a_{n,j}(\vec{x}_k)_j \end{pmatrix}$$

$$= \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix}^{-1} \left(\begin{pmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix} - \begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{i,1} & \cdots & 0 & a_{i,n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n-1} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix}_k \right) = D^{-1}\Big(\vec{b} - (L+U)\vec{x}_k\Big).$$

In other words, we arrive at the iteration

$$\vec{x}_{k+1} = -D^{-1}(L+U)\vec{x}_k + D^{-1}\vec{b} \tag{36}$$

which is precisely the formula for the Jacobi iterative method. This should be compared with the fixed-point expression in (33). In this case, the vector $\vec{c}$ is given by $D^{-1}\vec{b}$, whereas the matrix that determines whether the method converges or not is the matrix B, which here is

$$B_{\text{JIM}} := -D^{-1}(L+U).$$

Thus, the convergence of the Jacobi method is governed by the properties of the iteration matrix B_{JIM}.

Remark 35.4. How do we know that, if the sequence $(\vec{x}_k)$ converges, then its limit actually solves the original system? Suppose that $\vec{x}_k \to \vec{x}_f$. Passing to the limit in (36), we obtain

$$\vec{x}_f = -D^{-1}(L+U)\vec{x}_f + D^{-1}\vec{b}.$$

Multiplying by D from the left side, it follows that

$$D\vec{x}_f = -(L+U)\vec{x}_f + \vec{b}.$$

Rearranging the terms, we get

$$(D+L+U)\vec{x}_f = \vec{b}.$$

Since $A = D+L+U$, this shows that $A\vec{x}_f = \vec{b}$ and therefore $\vec{x}_f$ is indeed a solution of the original system. So, if the method works, then it really leads to the solution of the system.

The Jacobi iteration method is computationally much cheaper per step than direct methods such as Gaussian elimination. Indeed, at each iteration one only needs to compute matrix–vector type operations and divide by the diagonal entries, so the cost of one iteration is of order n^2 for a dense $n \times n$ system.

Therefore, if the method converges in N iterations, the total cost is of order $N \cdot n^2$. This is advantageous when M is not too large, especially compared with the $O(n^3)$ cost of the elimination method.

There is another feature that needs to be taken into account. The purpose of the following example is to show that the convergence of an iterative method depends not only on the original system, but also on the way in which the system is rewritten in iterative form. Although the linear system is the same as in Example 35.1, two different fixed-point formulations may lead to completely different behaviors.

Example 35.5. Consider the system

$$\begin{cases} 2x + y = 1, \\ x + 2y = -1. \end{cases}$$

If we switch the two equations, we obtain

$$\begin{cases} x + 2y = -1, \\ 2x + y = 1. \end{cases}$$

Now we isolate the variables in the order suggested by this new system:

$$\begin{cases} x_{k+1} = -1 - 2y_k, \\ y_{k+1} = 1 - 2x_k. \end{cases}$$

Starting from the initial guess

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

we compute

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} -1 - 2 \cdot 0 \\ 1 - 2 \cdot 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} -1 - 2 \cdot 1 \\ 1 - 2 \cdot (-1) \end{pmatrix} = \begin{pmatrix} -3 \\ 3 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} -1 - 2 \cdot 3 \\ 1 - 2 \cdot (-3) \end{pmatrix} = \begin{pmatrix} -7 \\ 7 \end{pmatrix}.$$

By induction, we can see that, for every $k \in \mathbb{N}$, we have that $x_k = -(2^k - 1)$ and $y_k = 2^k - 1$ and we see that the iterates do not converge. On the contrary, their absolute values grow rapidly, so this iterative form is not suitable for solving the system. This shows that the way in which the system is rewritten as a fixed-point problem is crucial for the convergence of the method.

Example 35.6. In the Maple experiment in Figure 139, we apply the Jacobi method to the linear system $A\vec{x} = \vec{b}$, where

$$A = \begin{pmatrix} 4 & 2 & -1 \\ 0 & 2 & 1 \\ -1 & 1 & 3 \end{pmatrix} \quad \text{and} \quad \vec{b} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}.$$

The initial approximation is chosen as

$$\vec{x_0} = \begin{pmatrix} 13 \\ 31 \\ -14 \end{pmatrix}$$

which is very far from the actual solution. Even so, the iterates produced by the Jacobi method gradually approach the vector

$$\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}.$$

The output displayed by Maple shows this convergence step by step. After a few iterations, the approximations are already much closer to the solution, and after 28 iterations the method reaches

$$\begin{pmatrix} 0.999995482 \\ -0.9999853615 \\ 0.9999899513 \end{pmatrix},$$

which is a very accurate approximation of 1, - 1 and 1. This experiment illustrates an important feature of the Jacobi method: when the method converges, it may still do so even if the initial guess is quite poor. The convergence is progressive, and the successive iterates move steadily toward the exact solution.

```
> A:=<<4,2,-1>|<0,2,1>|<-1,1,3>>:b:=<3,1,1>:
> MatrixIterate(A,b,xinit=<13.,31.,-14.>,method=jacobi,tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 31.0000000000,-14.0000000000]
k=01  x=[ -2.7500000000, -5.5000000000, -5.6666666670]
k=02  x=[ -0.6666666668,  6.0833333350,  1.2500000000]
k=03  x=[  1.0625000000,  0.5416666670, -1.9166666670]
k=04  x=[  0.2708333332,  0.3958333335,  0.5069444443]
k=05  x=[  0.8767361110, -0.0243055554,  0.2916666665]
k=06  x=[  0.8229166665, -0.5225694440,  0.6336805553]
k=07  x=[  0.9084201388, -0.6397569440,  0.7818287033]
k=08  x=[  0.9454571758, -0.7993344905,  0.8493923610]
k=09  x=[  0.9623480902, -0.8701533565,  0.9149305553]
k=10  x=[  0.9787326388, -0.9198133675,  0.9441671487]
k=11  x=[  0.9860417872, -0.9508162135,  0.9661820020]
k=12  x=[  0.9915455005, -0.9691327880,  0.9789526667]
k=13  x=[  0.9947381668, -0.9810218340,  0.9868927627]
k=14  x=[  0.9967231908, -0.9881845485,  0.9919200003]
k=15  x=[  0.9979800000, -0.9926831910,  0.9949692467]
k=16  x=[  0.9987423118, -0.9954646235,  0.9968877303]
k=17  x=[  0.9992219325, -0.9971861770,  0.9980689787]
k=18  x=[  0.9995172448, -0.9982564220,  0.9988027030]
k=19  x=[  0.9997006758, -0.9989185965,  0.9992578890]
k=20  x=[  0.9998144722, -0.9993296205,  0.9995397573]
k=21  x=[  0.9998849392, -0.9995843505,  0.9997146973]
k=22  x=[  0.9999286742, -0.9997422875,  0.9998230967]
k=23  x=[  0.9999557742, -0.9998402225,  0.9998903207]
k=24  x=[  0.9999725802, -0.9999009345,  0.9999319987]
k=25  x=[  0.9999829998, -0.9999385795,  0.9999578380]
k=26  x=[  0.9999894595, -0.9999619190,  0.9999738600]
k=27  x=[  0.9999934650, -0.9999763895,  0.9999837930]
k=28  x=[  0.9999959482, -0.9999853615,  0.9999899513]

                                        [ 0.9999959482 ]
                                        [-0.9999853615 ]
                                        [ 0.9999899513 ]
```

Figure 139: An experiment with the Jacobi iterative method.

35.3 Gauss-Seidel iteration

We now consider a slightly different way of using this system (34). Once again we isolate x_1, x_2 and x_3 are in (35). Instead of computing all new values only from the previous iteration, we update the components one after another, using each newly computed value as soon as it becomes available. Thus, after computing the new value of x_1, we immediately use it in the formula for x_2; then, after computing the new values of x_1 and x_2, we use both of them in the formula for x_3. In this way, the iteration takes

the form

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{1,1}} \cdot \left[b_1 + a_{1,2} x_2^{(k)} + a_{1,3} x_3^{(k)} \right], \\ x_2^{(k+1)} = \dfrac{1}{a_{2,2}} \cdot \left[b_2 + a_{2,1} x_1^{(k+1)} + a_{2,3} x_3^{(k)} \right], \\ x_3^{(k+1)} = \dfrac{1}{a_{3,3}} \cdot \left[b_3 + a_{3,1} x_1^{(k+1)} + a_{3,2} x_2^{(k+1)} \right]. \end{cases}$$

In other words, the method reuses the most recent information at each step, rather than waiting until the next iteration. This gives a new promising method.

Algorithm 35.7 (GSM: Gauss–Seidel iteration)**.** Given a system $A\vec{x} = \vec{b}$ of linear equations and tolerance ε, we have the following steps.

0. Choose an arbitrary initial vector $\vec{x}_0$. Set $k = 0$.

1. Compute

$$(\vec{x}_{k+1})_i = -\frac{1}{a_{i,i}} \left(\sum_{j=1}^{i-1} a_{i,j} (\vec{x}_{k+1})_j + \sum_{j=i+1}^{n} a_{i,j} (\vec{x}_k)_j \right) + \frac{b_i}{a_{i,i}}.$$

   If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geqslant \varepsilon$, increase k by one and go back to step 1.

We now write the Gauss–Seidel iteration in matrix form, in the same spirit as we did before for the Jacobi method. In this notation, the Gauss–Seidel iteration is equivalent to

$$\begin{aligned} (\vec{x})_{k+1} \quad = \quad & \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix}^{-1} \left[\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \right. \\ & \left. - \left(\begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{2,1} & 0 & \cdots & 0 \\ a_{3,1} & a_{3,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}_{k+1} + \begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ 0 & 0 & \ddots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n} \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}_{k} \right) \right]. \end{aligned}$$

In compact form, this can be written as

$$\vec{x}_{k+1} = D^{-1} \left[\vec{b} - L\vec{x}_{k+1} - U\vec{x}_k \right].$$

At this point, we must solve the equation for $\vec{x}_{k+1}$. Rearranging the terms, we obtain

$$\vec{x}_{k+1} + D^{-1} L \vec{x}_{k+1} = -D^{-1} U \vec{x}_k + D^{-1} \vec{b}.$$

Multiplying on the left by D, it follows that

$$(D + L)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b}.$$

Finally, multiplying by $(D + L)^{-1}$, we arrive at

$$\vec{x}_{k+1} = -(D + L)^{-1} U \vec{x}_k + (D + L)^{-1} \vec{b}.$$

Thus, in the fixed-point notation, the iteration matrix is

$$B_{\mathrm{GSM}} = -(D+L)^{-1}U,$$

while the constant vector is $\vec{c} = (D+L)^{-1}\vec{b}$.

Remark 35.8. Let us show that, if the sequence $(\vec{x}_k)$ converges to some vector $\vec{x}_f$, then this limit must solve the original system. Assume that $\vec{x}_k \to \vec{x}_f$. Since also $\vec{x}_{k+1} \to \vec{x}_f$, passing to the limit in the relation

$$(D+L)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b}$$

gives

$$(D+L)\vec{x}_f = -U\vec{x}_f + \vec{b}.$$

Rearranging the terms, we obtain

$$(D+L+U)\vec{x}_f = \vec{b}.$$

Since $A = D+L+U$, it follows that $A\vec{x}_f = \vec{b}$. Therefore, whenever the Gauss-Seidel iteration converges, its limit is a solution of the original system.

The Gauss–Seidel method has essentially the same computational complexity per iteration as the Jacobi method. For a dense $n \times n$ system, each step requires on the order of n^2 operations, since every new component is obtained by combining the entries of one row with the available approximations. Hence, if the method converges in N iterations, the total cost is of order $N \cdot n^2$. In practice, however, Gauss–Seidel often converges in fewer iterations than Jacobi, because it makes immediate use of the newest computed values. For this reason, even though the cost per iteration is comparable, Gauss-Seidel is frequently more efficient overall.

Example 35.9. Consider the system

$$\begin{cases} 2x + y = 1, \\ x + 2y = -1. \end{cases}$$

We isolate the variables as

$$\begin{cases} x = 1/2(1-y), \\ y = 1/2(-1-x). \end{cases}$$

Using the Gauss–Seidel iteration, we compute first the new value of x and then immediately use it to compute the new value of y. Thus, the scheme is

$$\begin{cases} x_{k+1} = 1/2(1-y_k), \\ y_{k+1} = 1/2(-1-x_{k+1}). \end{cases}$$

Starting from

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

we obtain

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1/2 \\ -3/4 \end{pmatrix}, \qquad \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 7/8 \\ -15/16 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} 31/32 \\ -63/64 \end{pmatrix}.$$

We see that the iterates converge to

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

which is the exact solution of the system. If we switch the equations, we obtain

$$\begin{cases} x + 2y = -1, \\ 2x + y = 1. \end{cases}$$

We then isolate the variables as

$$\begin{cases} x_{k+1} = -1 - 2y_k, \\ y_{k+1} = 1 - 2x_{k+1}. \end{cases}$$

Starting from

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

we obtain

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix}, \qquad \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} -7 \\ 15 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} -31 \\ 63 \end{pmatrix}.$$

We see that the iterates do not converge. On the contrary, their absolute values grow very quickly, so this iterative form is not suitable for solving the system.

Remark 35.10. In iterative methods, it is often convenient to arrange the equations so that the diagonal coefficients are as large as possible in absolute value. The intuition is that, after isolating each variable, the remaining coefficients are divided by the diagonal entry, so larger diagonal terms usually lead to smaller iteration coefficients and therefore improve the chances of convergence.

Example 35.11. Let us try Gauss-Seidel on Example 35.6 in Figure 140. In this Maple experiment, we apply the Gauss-Seidel iteration to the same linear system as before, starting from the initial approximation

$$\vec{x}_0 = \begin{pmatrix} 13 \\ 31 \\ -14 \end{pmatrix}.$$

The output shows the successive iterates produced by the method. We observe that the sequence converges rapidly toward

$$\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix},$$

which is the exact solution of the system. More precisely, after only 15 iterations (while with Jacobi's 28 iterations), Maple obtains the approximation

$$\begin{pmatrix} 0.9999992160 \\ -0.9999976480 \\ 0.9999989547 \end{pmatrix},$$

which is already extremely close to the exact solution. Compared with the Jacobi method, the convergence is clearly faster, since the Gauss–Seidel iteration makes immediate use of the newest computed values at each step.

```
> MatrixIterate(A,b,xinit=<13.,31.,-14.>,method=gaussseidel,tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 31.0000000000,-14.0000000000]
k=01  x=[ -2.7500000000, 10.2500000000, -4.0000000000]
k=02  x=[ -0.2500000000,  2.7500000000, -0.6666666667]
k=03  x=[  0.5833333332,  0.2500000004,  0.4444444443]
k=04  x=[  0.8611111110, -0.5833333330,  0.8148148147]
k=05  x=[  0.9537037038, -0.8611111115,  0.9382716053]
k=06  x=[  0.9845679012, -0.9537037035,  0.9794238680]
k=07  x=[  0.9948559670, -0.9845679010,  0.9931412893]
k=08  x=[  0.9982853222, -0.9948559665,  0.9977137627]
k=09  x=[  0.9994284408, -0.9982853225,  0.9992379213]
k=10  x=[  0.9998094802, -0.9994284405,  0.9997459733]
k=11  x=[  0.9999364932, -0.9998094795,  0.9999153240]
k=12  x=[  0.9999788310, -0.9999364930,  0.9999717747]
k=13  x=[  0.9999929438, -0.9999788315,  0.9999905920]
k=14  x=[  0.9999976480, -0.9999929440,  0.9999968640]
k=15  x=[  0.9999992160, -0.9999976480,  0.9999989547]
```

$$\begin{bmatrix} 0.9999992160 \\ -0.9999976480 \\ 0.9999989547 \end{bmatrix}$$

Figure 140: An experiment with the Gauss-Seidel iterative method.

In general, one should not expect a universal comparison between the Jacobi and Gauss-Seidel methods: depending on the system, it may happen that the Jacobi iteration converges while the Gauss-Seidel iteration does not, and conversely. Indeed, as shown in Figure 141, the Jacobi method produces a reasonably good approximation for the system, whereas the Gauss-Seidel iteration stops because the numbers become too large, which is a clear indication of divergence. Thus, in this example, the Jacobi method converges while the Gauss-Seidel method does not.

$$A := \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$b := \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \tag{2}$$

```
> MatrixIterate(A,b,xinit=<0.,0.,0.>,method=jacobi,tolerance=0.00001,output=value);
  MatrixIterate(A,b,xinit=<0.,0.,0.>,method=gaussseidel,tolerance=0.00001,output=value);
k=00 k=01 k=02 k=03 k=04 k=05 k=06 k=07 k=08 k=09 k=10 k=11 k=12 k=13 k=14 k=15 k=16 k=17 k=18 k=19 k=20 k=21 k=
22 k=23 k=24 k=25 k=26 k=27 k=28 k=29 k=30 k=31 k=32 k=33 k=34 k=35
```

$$\begin{bmatrix} -4.000034333 \\ 9.666717528 \\ -0.666709902 \end{bmatrix}$$

```
k=00 k=01 k=02 k=03 k=04 k=05 k=06 k=07 k=08 k=09 k=10 k=11 k=12 k=13 k=14 k=15 k=16 k=17 k=18 k=19 k=20 Numbers
in iteration too large.
```

$$\begin{bmatrix} -3.999950000 \\ 1.188395800 \times 10^{6} \\ -1.188386800 \times 10^{6} \end{bmatrix} \tag{3}$$

Figure 141: An experiment with the Gauss-Seidel iterative method.

35.4 Relaxation

As we saw in Section 22.4, we now apply the idea of relaxation to the iterative methods introduced in this section. Let us consider once again the basic iteration

$$\vec{x}_{k+1} = B\vec{x}_k + \vec{c}.$$

Suppose that the convergence of this method is not satisfactory, or that we would like to modify its behavior. We then introduce a parameter λ and define the relaxed iteration by

$$\vec{x}_{k+1} = \lambda \cdot (B\vec{x}_k + \vec{c}) + (1 - \lambda) \cdot \vec{x}_k.$$

In other words, instead of taking the full new iterate $B\vec{x}_k + \vec{c}$, we combine it with the previous value $\vec{x}_k$ through the parameter λ. Notice that this expression can be rewritten as

$$\vec{x}_{k+1} = (\lambda B + (1 - \lambda)\,\mathrm{Id}_n)\vec{x}_k + \lambda\vec{c}.$$

This shows that the relaxed method is again an iteration of the same form, but with a new iteration matrix. We denote this matrix by

$$B_\lambda = \lambda B + (1 - \lambda)\,\mathrm{Id}_n\,.$$

The convergence of the relaxed method will therefore depend on the properties of B_λ. Let us do some experiments in Maple.

We apply the Gauss-Seidel method in Figure 142 to the system $A\vec{x} = \vec{b}$, where

$$A = \begin{pmatrix} -3 & 3 & -1 \\ -4 & 5 & -1 \\ -5 & -2 & 1 \end{pmatrix} \quad \text{and} \quad \vec{b} = \begin{pmatrix} -4 \\ -7 \\ -5 \end{pmatrix},$$

starting from the initial approximation

$$\vec{x}_0 = \begin{pmatrix} 13 \\ 14 \\ 23 \end{pmatrix}.$$

The successive iterates grow very rapidly in magnitude and alternate wildly, which shows that the method is not converging. In fact, after a number of steps, Maple stops the computation and reports that the numbers in the iteration have become too large. This is a clear indication of divergence. In Figure 144, on othe other hand, we apply relaxation to the previous iterative method with parameter $\lambda = 0.79$ (we encourage the reader to try different λ's to see what happens). Starting from the same initial approximation as before, we observe that the behavior changes completely: while the original method diverged, the relaxed iteration now converges steadily to the solution after 39 iterations. In Figure 143, we apply the Gauss-Seidel method to a new linear system, again starting from the initial approximation $(13, 14, 23)$. In contrast with the previous example, the method now converges without the need for relaxation after 24 iterations. In Figure 145, we apply relaxation with parameter $\lambda = 1.2$ to the same iterative process the method converges with only 12 iterations.

```
> A:=<<-3,-4,-5>|<3,5,-2>|<-1,-1,1>>:
  b:=<-4,-7,-5>:
> MatrixIterate(A,b,xinit=<13.,14.,23.>,method=gaussseidel,tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 14.0000000000, 23.0000000000]
k=01  x=[  7.6666666670,  9.3333333340, 52.0000000100]
k=02  x=[ -6.6666666700,  3.6666666660,-31.0000000200]
k=03  x=[ 15.3333333400,  4.6666666680, 81.0000000400]
k=04  x=[-21.0000000100, -2.0000000000,-114.0000000000]
k=05  x=[ 37.3333333300,  5.6666666600, 192.9999999000]
k=06  x=[-57.3333333000, -8.6666666600,-308.9999998000]
k=07  x=[ 95.6666666000, 13.3333333200, 499.9999996000]
k=08  x=[-151.9999999000,-23.0000000000,-810.9999995000]
k=09  x=[ 248.6666665000, 35.3333333000, 1308.9999990000]
k=10  x=[-399.6666663000,-59.3333332000,-2121.9999980000]
k=11  x=[ 649.3333327000, 93.6666666000, 3428.9999970000]
k=12  x=[-1047.9999990000,-153.9999998000,-5552.9999950000]
k=13  x=[ 1698.3333320000, 246.6666666000, 8979.9999930000]
k=14  x=[-2745.3333310000,-401.6666654000,-14534.9999900000]
k=15  x=[ 4444.6666630000, 647.3333320000, 23512.9999800000]
k=16  x=[-7188.9999930000,-1049.9999980000,-38049.9999600000]
k=17  x=[ 11634.6666600000, 1696.3333360000, 61560.9999700000]
k=18  x=[-18822.6666500000,-2747.3333260000,-99612.9999000000]
k=19  x=[ 30458.3333100000, 4442.6666600000, 161171.9999000000]
k=20  x=[-49279.9999700000,-7191.0000000000,-260786.9998000000]
k=21  x=[ 79739.3332700000, 11632.6666600000, 421956.9997000000]
k=22  x=[ 129018.3332000000, 18824.6666200000, 682745.9992000000]
k=23 Numbers in iteration too large.
```

$$\begin{bmatrix} 208758.6664 \\ 30456.33328 \\ 1.104700999 \times 10^{6} \end{bmatrix}$$

Figure 142: An experiment with a divergent iteration using Gauss-Seidel.

```
> A:=<<3,-3,2>|<-1,5,1>|<-2,0,-5>>:
  b:=<0,8,9>:
> MatrixIterate(A,b,xinit=<13.,14.,23.>,method=gaussseidel,tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 14.0000000000, 23.0000000000]
k=01  x=[ 20.0000000000, 13.6000000000,  8.9200000000]
k=02  x=[ 10.4800000000,  7.8880000000,  3.9696000000]
k=03  x=[  5.2757333330,  4.7654400000,  1.2633813340]
k=04  x=[  2.4307342230,  3.0584405340, -0.2160182040]
k=05  x=[  0.8754680420,  2.1252808260, -1.0247566180]
k=06  x=[  0.0252558633,  1.6151535180, -1.4668669510]
k=07  x=[ -0.4395267947,  1.3362839230, -1.7085539330]
k=08  x=[ -0.6936079810,  1.1838352110, -1.8406761500]
k=09  x=[ -0.8325056963,  1.1004965820, -1.9129029620]
k=10  x=[ -0.9084364473,  1.0549381320, -1.9523869540]
k=11  x=[ -0.9499452587,  1.0300328450, -1.9739715350]
k=12  x=[ -0.9726367417,  1.0164179550, -1.9857711050]
k=13  x=[ -0.9850414183,  1.0089751490, -1.9922215380]
k=14  x=[ -0.9918226423,  1.0049064150, -1.9957477730]
k=15  x=[ -0.9955297103,  1.0026821740, -1.9976754490]
k=16  x=[ -0.9975562413,  1.0014662550, -1.9987292450]
k=17  x=[ -0.9986640783,  1.0008015530, -1.9993053210]
k=18  x=[ -0.9992696963,  1.0004381820, -1.9996202420]
k=19  x=[ -0.9996007673,  1.0002395400, -1.9997924000]
k=20  x=[ -0.9997817533,  1.0001309480, -1.9998865120]
k=21  x=[ -0.9998806920,  1.0000715850, -1.9999379590]
k=22  x=[ -0.9999347777,  1.0000391330, -1.9999660850]
k=23  x=[ -0.9999643457,  1.0000213930, -1.9999814590]
k=24  x=[ -0.9999805083,  1.0000116950, -1.9999898650]
```

$$\begin{bmatrix} -0.9999805083 \\ 1.000011695 \\ -1.999989865 \end{bmatrix}$$

Figure 143: An experiment with a convergent iteration using Gauss-Seidel.

```
> MatrixIterate(A,b,xinit=<13.,14.,23.>,method=[sor,0.79],tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 14.0000000000, 23.0000000000]
k=01  x=[  8.7866666670, 11.0211733300, 53.0007872000]
k=02  x=[ -2.3516137000,  8.0963509190, 10.6835256500]
k=03  x=[  4.1422832630,  4.9001537670, 22.3978022300]
k=04  x=[ -0.1037536265,  3.3963127510,  5.7098057900]
k=05  x=[  2.2110288860,  1.9067578890,  8.9953175800]
k=06  x=[  0.6552211690,  1.1297791130,  2.3121913080]
k=07  x=[  1.4745782340,  0.4285132843,  3.0371951890]
k=08  x=[  0.9017255241,  0.0337551607,  0.3029599635]
k=09  x=[  1.1895828130, -0.2992274040,  0.3396944053]
k=10  x=[  0.9773032150, -0.4975104070, -0.8043829187]
k=11  x=[  1.0773546220, -0.6566815655, -0.9009265295]
k=12  x=[  0.9980433536, -0.7554861212, -1.3905913950]
k=13  x=[  1.0322774690, -0.8319661655, -1.4790347320]
k=14  x=[  1.0023374770, -0.8809230970, -1.6932227530]
k=15  x=[  1.0137769480, -0.9178160142, -1.7513071360]
k=16  x=[  1.0023293870, -0.9419757180, -1.8468950550]
k=17  x=[  1.0060107190, -0.9598255450, -1.8806299830]
k=18  x=[  1.0015659660, -0.9717132112, -1.9240536040]
k=19  x=[  1.0026761980, -0.9803688866, -1.9424631160]
k=20  x=[  1.0009192020, -0.9862057028, -1.9624914170]
k=21  x=[  1.0012132670, -0.9904100568, -1.9721786830]
k=22  x=[  1.0005045610, -0.9932714613, -1.9815334160]
k=23  x=[  1.0005586360, -0.9953162287, -1.9865150460]
k=24  x=[  1.0002664550, -0.9967173858, -1.9909291330]
k=25  x=[  1.0002605590, -0.9977127808, -1.9934521040]
k=26  x=[  1.0001373410, -0.9983983170, -1.9955517860]
k=27  x=[  1.0001228080, -0.9988832142, -1.9968162620]
k=28  x=[  1.0000696660, -0.9992184154, -1.9978213310]
k=29  x=[  1.0000583660, -0.9994547502, -1.9984504380]
k=30  x=[  1.0000349530, -0.9996185765, -1.9989338790]
k=31  x=[  1.0000279190, -0.9997338091, -1.9992452530]
k=32  x=[  1.0000174040, -0.9998138505, -1.9994786410]
k=33  x=[  1.0000134220, -0.9998700512, -1.9996321790]
k=34  x=[  1.0000086190, -0.9999091478, -1.9997451670]
k=35  x=[  1.0000064770, -0.9999365640, -1.9998206720]
k=36  x=[  1.0000042520, -0.9999556574, -1.9998754840]
k=37  x=[  1.0000031340, -0.9999690339, -1.9999125460]
k=38  x=[  1.0000020920, -0.9999783573, -1.9999391760]
k=39  x=[  1.0000015200, -0.9999848842, -1.9999573400]
```

$$\begin{bmatrix} 1.000001520 \\ -0.9999848842 \\ -1.999957340 \end{bmatrix}$$

Figure 144: Relaxation with $\lambda = 0.79$. We get convergence.

```
> MatrixIterate(A,b,xinit=<13.,14.,23.>,method=[sor,1.2],tolerance=0.00001,output=iterates);
k=00  x=[ 13.0000000000, 14.0000000000, 23.0000000000]
k=01  x=[ 21.4000000000, 14.5280000000,  6.9987200000]
k=02  x=[  7.1301760000,  4.1481267200,  0.8582908920]
k=03  x=[  0.9198482020,  1.7526653620, -1.4694913540]
k=04  x=[ -0.6584965787,  1.0953493910, -1.9192962330]
k=05  x=[ -0.9655979143,  1.0056996240, -1.9982598430]
k=06  x=[ -1.0032084420,  0.9965499972, -2.0027160830]
k=07  x=[ -1.0029111800,  0.9985939506, -2.0011916010]
k=08  x=[ -1.0009334640,  0.9996091159, -2.0003035540]
k=09  x=[ -1.0002125040,  0.9999251738, -2.0000592510]
k=10  x=[ -1.0000348300,  0.9999898872, -2.0000072940]
k=11  x=[ -1.0000029140,  0.9999999246, -1.9999999590]
k=12  x=[ -0.9999994142,  1.0000004370, -1.9999996220]
```

$$\begin{bmatrix} -0.9999994142 \\ 1.000000437 \\ -1.999999622 \end{bmatrix}$$

Figure 145: Relaxation with $\lambda = 1.2$. We get a much faster convergence.

We have the following algorithm.

Algorithm 35.12 (Relaxation (SOR, Successive OverRelaxation method))**.** Given a system $A\vec{x} = \vec{b}$ of linear equations, tolerance ε, and parameter of relaxation ω, we have the following steps.

0. Choose an arbitrary initial vector $\vec{x}_0$. Set $k = 0$.

1. Compute

$$(\vec{x}_{k+1})_i = (1-\omega)(\vec{x}_k)_i - \frac{\omega}{a_{i,i}} \left(\sum_{j=1}^{i-1} a_{i,j}(\vec{x}_{k+1})_j + \sum_{j=i+1}^{n} a_{i,j}(\vec{x}_k)_j \right) + \frac{\omega b_i}{a_{i,i}}.$$

   If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geqslant \varepsilon$, increase k by one and go back to step 1.

This algorithm describes the relaxed version of the Gauss-Seidel iteration, usually called the SOR method. At each step, the new approximation is obtained by combining two ingredients: the previous iterate and the new Gauss–Seidel correction. The parameter ω controls this balance. When $\omega = 1$, the method reduces to the usual Gauss–Seidel iteration; when $\omega < 1$, one speaks of under-relaxation; and when $\omega > 1$, of over-relaxation. The role of the parameter ω is to modify the speed and stability of convergence. A suitable choice of ω may significantly improve the behavior of the method as we have seen in the previous examples.