

Chapter 10

Solving Linear Systems

Suppose we have a (large) linear system of the form

$$\mathbf{Ax} = \mathbf{b}$$

and we want to solve for the unknown vector \mathbf{x} . Note: if we have a system of the form

$$\mathbf{x}\mathbf{C} = \mathbf{d}$$

it can be translated into the “standard” form using the property

$$\mathbf{x}\mathbf{C} = \mathbf{C}^T \mathbf{x}$$

The “by hand” option rapidly becomes intractable as the dimension of \mathbf{x} increases. So, we need some algorithm to do this, so we can use software to obtain a solution. There are a great number of algorithms to solve the above system (especially when \mathbf{A} has special structure). Generally, these can be classified into two categories.

Direct methods modify the matrix in such a way that solving the system becomes easy (or, at least, easier).

Example: Gaussian elimination / LU factorization. Reduces the original matrix to triangular form.

There are many others (various factorizations). We will not discuss these.

Indirect methods do not modify the matrix; also called “iterative methods”. We will focus on these. Why?

- These are typically more efficient than direct methods for large, sparse matrices. That is usually the type of Markov chain we will have.
- Since the original matrix is not modified, we can use strange, compact data structures for the matrix, so long as certain critical operations (e.g., vector–matrix multiply) can be implemented efficiently.

10.1 A quick caveat

The following example is borrowed from *Numerical Linear Algebra and Optimization, Volume 1*, by Gill, Murray, and Wright. Suppose we have the following system, to 3 digits of precision:

$$\mathbf{A} = \begin{bmatrix} 0.550 & 0.423 \\ 0.484 & 0.372 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.127 \\ 0.112 \end{bmatrix}$$

The mathematically exact solution to $\mathbf{Ax} = \mathbf{b}$ is

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

which can be easily verified by matrix–vector multiplication. Mathematically, there is no difficulty here.

However, computers cannot store all real numbers exactly because in reality, reals are represented in base–2 arithmetic with a finite number of bits. Suppose in the above example, the value 0.484 in matrix \mathbf{A} is represented internally, due to floating–point roundoff error, as 0.483. Is this a problem? In other words, what if we use $\hat{\mathbf{A}}$ as

$$\hat{\mathbf{A}} = \begin{bmatrix} 0.550 & 0.423 \\ 0.483 & 0.372 \end{bmatrix}$$

and solve $\hat{\mathbf{A}}\hat{\mathbf{x}} = \mathbf{b}$. Hopefully, $\hat{\mathbf{x}}$ is close to \mathbf{x} . The mathematically exact solution, to 3 digits of accuracy, is:

$$\hat{\mathbf{x}} = \begin{bmatrix} -0.454 \\ 0.890 \end{bmatrix}$$

Lessons:

1. Solving linear systems with floating–point arithmetic can be trickier than you think¹.
2. Do not fall into the trap of trusting numbers simply because “the computer told you so”, especially when floating–point arithmetic is used.

10.2 Iterative methods: theoretical slant

All iterative methods use the same basic idea:

1. Start with an initial “guess” solution, \mathbf{x}_0
2. Compute a sequence

$$\mathbf{x}_{n+1} = \mathbf{B}\mathbf{x}_n + \mathbf{k}$$

of (hopefully increasingly–accurate) approximations to \mathbf{x} . The matrix \mathbf{B} and vector \mathbf{k} depend on the iterative method.

¹Especially for systems like the one above, which are known as “ill conditioned”. The difficulty of this system is that the two equations are very nearly multiples of each other. In effect, we are trying to find the intersection of two nearly parallel lines.

3. It can be proved that \mathbf{x}_n converges (mathematically, using infinite-precision reals) for any \mathbf{x}_0 , provided

$$\lim_{n \rightarrow \infty} \mathbf{B}^n = 0$$

4. \mathbf{x}_N is used as an approximation to \mathbf{x} , for large N .

10.2.1 Stopping the iterations

When can we stop iterating? I.e., which \mathbf{x}_N should we use as the “answer”? Need to decide the *stopping criteria*, which determines when we stop the iterations. Reasonable stopping criteria:

- Convergence of the vector. Specify a desired precision ϵ . This can be “absolute” or “relative” precision.

absolute precision stops when

$$| \mathbf{x}_N[i] - \mathbf{x}_{N-1}[i] | < \epsilon$$

for all elements i .

relative precision stops when

$$\left| \frac{\mathbf{x}_N[i] - \mathbf{x}_{N-1}[i]}{\mathbf{x}_N[i]} \right| < \epsilon$$

for all elements i (taking care if $\mathbf{x}_N[i] = 0$). I.e., relative precision stops when the “percentage change” of each element is below a threshold.

Important: these do NOT guarantee that our solution \mathbf{x}_N is within ϵ (either relative or absolute) of the exact solution \mathbf{x} .

- Residual testing. The idea here is that $\mathbf{A}\mathbf{x}_N$ should be close to \mathbf{b} if \mathbf{x}_N is close to \mathbf{x} . So, we stop once the magnitude of elements of the vector $\mathbf{A}\mathbf{x}_N - \mathbf{b}$ are below some threshold ϵ . Again, this does NOT guarantee that \mathbf{x}_N is within ϵ of the exact solution \mathbf{x} .
- Other tests ...
- Combination of tests. I.e., stop once the relative precision of ϵ_1 is reached, and the residual error is below ϵ_2 .
- Should *always* specify a maximum number of iterations, as a safety net. Even if convergence is guaranteed mathematically, it may not happen in practice due to (tiny) floating-point errors that accumulate over time.

10.3 Method of Jacobi

10.3.1 Intuitive idea

Multiply out $\mathbf{A}\mathbf{x} = \mathbf{b}$ to get the equations. For a 3×3 system we obtain something like:

$$\begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d \\ h \\ l \end{bmatrix}$$

which multiplies out to give 3 equations:

$$\begin{aligned} (1) \quad & ax + by + cz = d \\ (2) \quad & ex + fy + gz = h \\ (3) \quad & ix + jy + kz = l \end{aligned}$$

Solve the first equation for x , the second for y , the third for z , and we obtain

$$\begin{aligned} (1) \quad & x = (by + cz - d) / -a \\ (2) \quad & y = (ex + gz - h) / -f \\ (3) \quad & z = (ix + jy - l) / -k \end{aligned}$$

The method of Jacobi is to plug in the previous guesses for x, y, z into the above equations and compute the new guesses. I.e., we use the old values on the right hand sides to obtain the new values:

$$\begin{aligned} (1) \quad & x_{n+1} = (by_n + cz_n - d) / -a \\ (2) \quad & y_{n+1} = (ex_n + gz_n - h) / -f \\ (3) \quad & z_{n+1} = (ix_n + jy_n - l) / -k \end{aligned}$$

Example 10.1

For the Land of Oz DTMC, compute the steady-state probability vector using the method of Jacobi. The linear system is

$$\boldsymbol{\rho}(\mathbf{P} - \mathbf{I}) = \mathbf{0} \quad \text{or} \quad (\mathbf{P} - \mathbf{I})^\top \boldsymbol{\rho} = \mathbf{0}$$

For the Land of Oz, we have

$$(\mathbf{P} - \mathbf{I})^\top = \begin{bmatrix} -1/2 & 1/2 & 1/4 \\ 1/4 & -1 & 1/4 \\ 1/4 & 1/2 & -1/2 \end{bmatrix} \quad \boldsymbol{\rho} = \begin{bmatrix} r \\ n \\ s \end{bmatrix}$$

Multiplying out and solving gives three equations:

$$\begin{aligned} (1) \quad & r = (1/2 n + 1/4 s - 0) / (1/2) = n + 1/2 s \\ (2) \quad & n = (1/4 r + 1/3 s - 0) / 1 = (r + s)/4 \\ (3) \quad & s = (1/4 n + 1/2 n - 0) / (1/2) = 1/2 r + n \end{aligned}$$

These are the equations to use when updating the solution vector.

Iteration 0 : use $\boldsymbol{\rho}_0 = [1/3, 1/3, 1/3]$

Iteration 1 :

$$\begin{aligned} r_1 &= n_0 + 1/2 s_0 = 1/3 + 1/2 \cdot 1/3 = 1/2 \\ n_1 &= (r_0 + s_0)/4 = (1/3 + 1/3)/4 = 1/6 \\ s_1 &= 1/2 r_0 + n_0 = 1/2 \cdot 1/3 + 1/3 = 1/2 \end{aligned}$$

Note: these do not sum to one! Fix: periodically “normalize” the solution vector, by dividing each element by the sum of elements. At the end of iteration 1:

$$\begin{aligned} r_1 &= 1/2 \cdot 6/7 = 3/7 \approx 0.42857 \\ n_1 &= 1/6 \cdot 6/7 = 1/7 \approx 0.14286 \\ s_1 &= 1/2 \cdot 6/7 = 3/7 \approx 0.42857 \end{aligned}$$

Iteration 2:

$$\begin{aligned}
r_2 &= 1/7 + 1/2 \cdot 3/7 = 5/14 \quad (\textit{normalize}) = 5/13 \approx 0.38461 \\
n_2 &= (3/7 + 3/7)/4 = 3/14 \quad (\textit{normalize}) = 3/13 \approx 0.23076 \\
s_2 &= 1/2 \cdot 3/7 + 1/7 = 5/14 \quad (\textit{normalize}) = 5/13 \approx 0.38461
\end{aligned}$$

Iteration 3:

$$\begin{aligned}
r_3 &= 3/13 + 1/2 \cdot 5/13 = 11/26 \quad (\textit{normalize}) = 11/27 \approx 0.40740 \\
n_3 &= (5/13 + 5/13)/4 = 5/26 \quad (\textit{normalize}) = 5/27 \approx 0.18518 \\
s_3 &= 1/2 \cdot 5/13 + 3/13 = 11/26 \quad (\textit{normalize}) = 11/27 \approx 0.40740
\end{aligned}$$

This is slowly converging to the exact solution $\boldsymbol{\rho} = [2/5, 1/5, 2/5]$.

10.3.2 Implementation

Looking at the equations we obtained in the above examples, we want to store the diagonals of matrix \mathbf{A} separately. Let \mathbf{D} be a matrix, just containing the diagonals, so that the system we want to solve is

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

This can be rewritten as:

$$\begin{aligned}
(\mathbf{A} + \mathbf{D})\mathbf{x} &= \mathbf{b} \\
\mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{x} &= \mathbf{b} \\
\mathbf{A}\mathbf{x} - \mathbf{b} &= -\mathbf{D}\mathbf{x} \\
(-\mathbf{D})^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b}) &= \mathbf{x}
\end{aligned}$$

Note that

1. Since $-\mathbf{D}$ is a diagonal matrix, it is trivial to compute its inverse: simply invert each diagonal element!
2. This matches exactly the set of equations we used above.

So the method of Jacobi uses the sequence

$$\mathbf{x}_{n+1} = (-\mathbf{D})^{-1}(\mathbf{A}\mathbf{x}_n - \mathbf{b})$$

To implement the Jacobi iteration, we need two solution vectors, one to hold \mathbf{x}_{n+1} while it is being computed, and the other to hold \mathbf{x}_n .

10.4 Method of Gauss–Seidel**10.4.1 Intuitive idea**

During the Jacobi iteration, we always use the “old guesses” when computing the “new guesses”. I.e., for a 3×3 system, the Jacobi iteration is:

$$\begin{aligned}
(1) \quad x_{n+1} &= (by_n + cz_n - d) / -a \\
(2) \quad y_{n+1} &= (ex_n + gz_n - h) / -f \\
(3) \quad z_{n+1} &= (ix_n + jy_n - l) / -k
\end{aligned}$$

Idea: why not use the *newest* value of each variable? That's what Gauss–Seidel does. So, with Gauss–Seidel, the iteration for a 3×3 system is:

$$\begin{aligned} (1) \quad x_{n+1} &= (by_n + cz_n - d) / -a \\ (2) \quad y_{n+1} &= (ex_{n+1} + gz_n - h) / -f \\ (3) \quad z_{n+1} &= (ix_{n+1} + jy_{n+1} - l) / -k \end{aligned}$$

Note, this assumes that the equations are evaluated in the specified order.

Example 10.2

For the Land of Oz DTMC, compute the steady-state probability vector using the method of Gauss–Seidel. Use the equations we obtained for Jacobi, but use the newest variable values:

$$\begin{aligned} (1) \quad r &= n + 1/2 s \\ (2) \quad n &= (r + s)/4 \\ (3) \quad s &= 1/2 r + n \end{aligned}$$

Iteration 0 : use $\rho_0 = [1/3, 1/3, 1/3]$

Iteration 1 :

$$\begin{aligned} r_1 &= n_0 + 1/2 s_0 = 1/3 + 1/2 \cdot 1/3 = 1/2 \\ n_1 &= (r_1 + s_0)/4 = (1/2 + 1/3)/4 = 5/24 \\ s_1 &= 1/2 r_1 + n_1 = 1/2 \cdot 1/2 + 5/24 = 11/24 \end{aligned}$$

Again, we need to normalize the vector so that the entries sum to one:

$$\begin{aligned} r_1 &= 1/2 \cdot 24/28 = 12/28 \approx 0.42857 \\ n_1 &= 5/24 \cdot 24/28 = 5/28 \approx 0.17857 \\ s_1 &= 11/24 \cdot 24/28 = 11/28 \approx 0.39285 \end{aligned}$$

Iteration 2:

$$\begin{aligned} r_2 &= 5/28 + 1/2 \cdot 11/28 = 21/56 \quad (\textit{normalize}) = 84/212 \approx 0.39623 \\ n_2 &= (21/56 + 11/28)/4 = 43/224 \quad (\textit{normalize}) = 43/212 \approx 0.20283 \\ s_2 &= 1/2 \cdot 21/56 + 43/224 = 85/224 \quad (\textit{normalize}) = 85/212 \approx 0.40094 \end{aligned}$$

Iteration 3:

$$\begin{aligned} r_3 &= 43/212 + 1/2 \cdot 85/212 = 171/424 \quad (\textit{normalize}) = 684/1708 \approx 0.40046 \\ n_3 &= (171/424 + 85/212)/4 = 341/1696 \quad (\textit{normalize}) = 341/1708 \approx 0.19964 \\ s_3 &= 1/2 \cdot 171/424 + 341/1696 = 683/1696 \quad (\textit{normalize}) = 683/1708 \approx 0.39988 \end{aligned}$$

Compare this convergence rate with that of Jacobi.

10.4.2 Theory

To see how Gauss–Seidel fits into the theory discussed earlier, split \mathbf{A} into $\mathbf{D} + \mathbf{L} + \mathbf{U}$, where \mathbf{D} are the diagonal entries, \mathbf{L} is lower-triangular, and \mathbf{U} is upper-triangular. Split the system we are solving into

$$\begin{aligned} (\mathbf{D} + \mathbf{L} + \mathbf{U})\mathbf{x} &= \mathbf{b} \\ (\mathbf{D} + \mathbf{L})\mathbf{x} + \mathbf{U}\mathbf{x} &= \mathbf{b} \end{aligned}$$

The Gauss–Seidel iteration is given by

$$\begin{aligned}(\mathbf{D} + \mathbf{L})\mathbf{x}_{n+1} + \mathbf{U}\mathbf{x}_n &= \mathbf{b} \\ \mathbf{x}_{n+1} &= (\mathbf{D} + \mathbf{L})^{-1}(-\mathbf{U}\mathbf{x}_n + \mathbf{b})\end{aligned}$$

10.4.3 Implementation

The implementation of Gauss–Seidel, in practice, can be done almost exactly the same as the implementation of Jacobi. The difference is, we use a *single* vector instead of two vectors. The new values will overwrite the old values in the vector. This automatically ensures that the newest value of each variable is used in the equations.

10.4.4 Comparison with Jacobi

- The order of equations can affect the convergence rate of Gauss–Seidel. Not so for Jacobi.
- Gauss–Seidel requires only one solution vector, while Jacobi requires two.
- The required basic operation for Gauss–Seidel is “dot product of a given matrix row, and a given vector”. This implies that we require efficient access to a given row of the matrix. Conversely, Jacobi requires only vector–matrix multiplication. We thus have more flexibility in how we store the matrix when Jacobi is used.

Theory says:

- Jacobi and Gauss–Seidel will either both converge or both fail to converge.
- Gauss–Seidel has a faster “expected rate” of convergence.

The above applies to “infinite precision” reals.

10.5 Tricks for both methods

10.5.1 Storage of diagonals

Both Jacobi and Gauss–Seidel require direct access to the diagonals of our matrix (denoted by \mathbf{D} in the theoretical discussions). In practice, we can store the diagonal entries in a separate vector \mathbf{d} , and use a diagonal–free matrix \mathbf{A} . On systems where floating–point multiplication is less expensive than floating–point division, we can instead use a vector \mathbf{h} with $\mathbf{h}[i] = -1/\mathbf{d}[i]$, so that our Jacobi or Gauss–Seidel iteration performs “multiplication by $\mathbf{h}[i]$ ” instead of “division by $-\mathbf{d}[i]$ ”.

10.5.2 Relaxation

The idea behind *relaxation* is the following. At each iteration, we are computing

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \boldsymbol{\delta}_n$$

for an appropriate difference vector $\boldsymbol{\delta}_n$. Think of the solution method (Jacobi or Gauss–Seidel) as a way of computing $\boldsymbol{\delta}_n$. We should, therefore, be able to change the rate of convergence using instead the iteration

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \omega\boldsymbol{\delta}_n$$

for a real-valued scalar ω , which is called the *relaxation parameter*. How do various values of ω affect convergence?

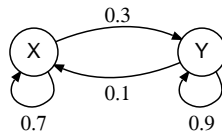
- $\omega > 2$ Won't work
- $\omega > 1$ We try to “speed up” convergence; may become unstable. Called *over relaxation*.
- $\omega = 1$ Ordinary Jacobi or Gauss–Seidel
- $\omega < 1$ We try to “slow down” convergence; useful to stabilize. Called *under relaxation*.
- $\omega \leq 0$ Clearly, won't work.

Some notes on selecting the relaxation parameter ω :

- For each linear system, there is an optimal value of ω , which produces the fewest possible number of iterations. Call this ω^* .
- The computation required to obtain the value of ω^* is equivalent to solving the linear system.
- The number of iterations required may increase dramatically as w is increased past ω^* .
- The optimal value ω^* may be less than one.

Example 10.3

Consider the following 2-state DTMC:



To solve for the steady-state probability vector $\boldsymbol{\rho} = [x, y]$ we solve the linear system

$$\boldsymbol{\rho} \cdot (\mathbf{P} - \mathbf{I}) = \mathbf{0}$$

$$[x, y] \cdot \begin{bmatrix} -0.3 & 0.3 \\ 0.1 & -0.1 \end{bmatrix} = \mathbf{0}$$

Multiplying this out and simplifying gives the following equations for Jacobi:

$$\begin{aligned} (1) \quad x_{n+1} &= 1/3 y_n = x_n + (1/3 y_n - x_n) \\ (2) \quad y_{n+1} &= 3 x_n = y_n + (3 x_n - y_n) \end{aligned}$$

Adapting the above for use with relaxation gives us

$$\begin{aligned} (1) \quad x_{n+1} &= x_n + \omega(1/3 y_n - x_n) = x_n(1 - \omega) + \omega(1/3 y_n) \\ (2) \quad y_{n+1} &= y_n + \omega(3 x_n - y_n) = y_n(1 - \omega) + \omega(3 x_n) \end{aligned}$$

With a uniform initial guess and “ordinary” Jacobi ($\omega = 1$) we obtain the following iterations:

Iteration 0 :

$$\begin{aligned} x_0 &= 1/2 \\ y_0 &= 1/2 \end{aligned}$$

Iteration 1 :

$$\begin{aligned}x_1 &= 1/3 \cdot 1/2 = 1/6 \quad (\text{normalize}) = 1/10 \\y_1 &= 3 \cdot 1/2 = 3/2 \quad (\text{normalize}) = 9/10\end{aligned}$$

Iteration 2 :

$$\begin{aligned}x_2 &= 1/3 \cdot 9/10 = 3/10 \quad (\text{normalize}) = 1/2 \\y_2 &= 3 \cdot 1/10 = 3/10 \quad (\text{normalize}) = 1/2\end{aligned}$$

This will repeat forever!

With a uniform initial guess and Jacobi with $\omega = 0.9$, we instead obtain the following iterations:

Iteration 0 :

$$\begin{aligned}x_0 &= 1/2 \\y_0 &= 1/2\end{aligned}$$

Iteration 1 :

$$\begin{aligned}x_1 &= 1/10 \cdot 1/2 + 3/10 \cdot 1/2 = 2/10 \quad (\text{normalize}) = 1/8 = 0.12500 \\y_1 &= 1/10 \cdot 1/2 + 27/10 \cdot 1/2 = 14/10 \quad (\text{normalize}) = 7/8 = 0.87500\end{aligned}$$

Iteration 2 :

$$\begin{aligned}x_2 &= 1/10 \cdot 1/8 + 3/10 \cdot 7/8 = 11/40 \quad (\text{normalize}) = 11/28 \approx 0.39285 \\y_2 &= 1/10 \cdot 7/8 + 27/10 \cdot 1/8 = 17/40 \quad (\text{normalize}) = 17/28 \approx 0.60714\end{aligned}$$

Iteration 3 :

$$\begin{aligned}x_2 &= 1/10 \cdot 11/28 + 3/10 \cdot 17/28 = 52/280 \quad (\text{normalize}) = 52/366 \approx 0.14207 \\y_2 &= 1/10 \cdot 17/28 + 27/10 \cdot 11/28 = 314/280 \quad (\text{normalize}) = 314/366 \approx 0.85792\end{aligned}$$

This will slowly converge to the exact solution of $\rho = [1/4 , 3/4]$.

Example 10.4

For the “Land of Oz” DTMC, the table below shows the value of ρ_5 , i.e., the estimate of the steady-state distribution after five iterations, for Jacobi and Gauss-Seidel using various values for ω :

ω	Jacobi	Gauss-Seidel
0.4	[0.399385 , 0.201230 , 0.399385]	[0.399967 , 0.202279 , 0.397754]
0.5	[0.399941 , 0.200117 , 0.399941]	[0.400309 , 0.200432 , 0.399259]
0.6	[0.399999 , 0.200001 , 0.399999]	[0.400159 , 0.200016 , 0.399825]
0.7	[0.400000 , 0.200000 , 0.400000]	[0.400033 , 0.199987 , 0.399980]
0.8	[0.400019 , 0.199962 , 0.400019]	[0.400000 , 0.199999 , 0.400000]
0.9	[0.400315 , 0.199370 , 0.400315]	[0.400000 , 0.200000 , 0.400000]
1.0	[0.401869 , 0.196262 , 0.401869]	[0.400007 , 0.199995 , 0.399998]
1.1	[0.406882 , 0.186236 , 0.406882]	[0.400104 , 0.199906 , 0.399991]
1.2	[0.419037 , 0.161926 , 0.419037]	[0.400488 , 0.199491 , 0.400021]
1.3	[0.443092 , 0.113815 , 0.443092]	[0.401459 , 0.198297 , 0.400243]
1.4	[0.483227 , 0.033546 , 0.483227]	[0.403383 , 0.195652 , 0.400965]

For this linear system, the optimal relaxation parameter for Jacobi appears to be around 0.7, while for Gauss-Seidel the value appears to be around 0.9.

