JACOBI AND GAUSS-SEIDEL

David F. Gleich

August 21, 2023

Learning objectives

1. Viewing linear equations as a set of equations where we can solve for any term
2. Realize that Gauss-Seidel is just a "bug" in Jacobi

Now let's see another set of methods that can apply to solving $\boldsymbol{A}\mathbf{x} = \mathbf{b}$.

These, again, follow from different perspectives on what these equations mean. Consider that a system of linear equations represents a simultaneous solution of n individual equations

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \begin{pmatrix} \mathbf{a}_1^T\mathbf{x} = b_1 \\ \mathbf{a}_2^T\mathbf{x} = b_2 \\ \cdots \\ \mathbf{a}_n^T\mathbf{x} = b_n \end{pmatrix}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \cdots \\ \mathbf{a}_n^T \end{bmatrix}$$

is the matrix with a vector for each row. We can use each of these equations to *solve* for any particular variable given the others. For example, given a possible solution $\mathbf{x}$, then

$$\mathbf{a}_1^T\mathbf{x} = b_1 \Leftrightarrow \sum_j A_{1j}x_j = b_1$$

and if $A_{1k} \neq 0$, then we must have

$$x_k = \frac{1}{A_{1k}}(b_1 - \sum_{j\neq k} A_{1j}x_j)$$

at a solution.[1]

[1] There must always exist such a k otherwise the system is singular!

Then one strategy to *improve* an in-accurate solution to $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ is to pick out a set of entries that can all be updated simultaneously and to do so.

1 A SIMPLE TWO VARIABLE EXAMPLE

Let's see an example. Suppose that

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

and we have a current guess $\mathbf{x} = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}$. Then for row 1, we have our choice of index k to update, 1 or 2. For the sake of example, let's use row 1 to update x_2 and then we'll use row 2 to update x_1. The update looks like:

$$x_1^{\text{new}} = \frac{1}{-2}(1 - x_2^{\text{old}})$$
$$x_2^{\text{new}} = \frac{1}{2}(2 - x_1^{\text{old}})$$

For the guess $\mathbf{x} = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}$

$$\mathbf{x}^{\text{new}} = \begin{bmatrix} -1/4 \\ 1/2 \end{bmatrix}.$$

If we repeatedly use this formula, then we converge to the solution $\mathbf{x} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$ after a few iterations.

2 THE GENERAL EXAMPLE AND THE JACOBI ITERATION

If we think about running this on a general system, then we'll need to figure out which equation we use to update which variable. This is tricky, however, because the update to x_2 based on row 1 used an entry A_{12} that was non-zero. In general, this means that we cannot update x_2 from any of the other rows from 2 to n. In an ideal world (like the example above), we'd like to update $x_1, \ldots, x_n$ all at once.[2] That means that we need a distinct row i for each j such that $A_{ij} \neq 0$. We can encode this map in a matrix $\boldsymbol{M}$. Let $\boldsymbol{M}$ be an $n \times n$ matrix where:

$$M(i,j) = \begin{cases} 1 & \text{row } j \text{ is used to update } x_i \\ 0 & \text{otherwise.} \end{cases}$$

[2] There are some really interesting and useful extensions that relax this assumption.

Then note that we must have exactly 1 entry in each row and column of $\boldsymbol{M}$.[3] We can use any matrix $\boldsymbol{M}$ where

$$M_{i,j} \neq 0 \text{ if and only if } A_{i,j} \neq 0.$$

[3] Technically, and looking ahead just a few paragraphs, the matrix $\boldsymbol{M}$ is a permutation matrix, we will use this insight in a moment!

In general these are not-so-easy to find. We'll return to this point in a moment. Let $M_i = j$ for convinience of notation. The iteration is thus:

$$\begin{aligned} x_1^{\text{new}} &= \frac{1}{A_{M_1,1}}(b_{M_1} - \sum_{j \neq 1} A_{M_1,j} x_j) \\ &\vdots \\ x_i^{\text{new}} &= \frac{1}{A_{M_i,i}}(b_{M_i} - \sum_{j \neq i} A_{M_i,j} x_j) \\ &\vdots \\ x_n^{\text{new}} &= \frac{1}{A_{M_n,n}}(b_{M_n} - \sum_{j \neq n} A_{M_i,j} x_j) \end{aligned}$$

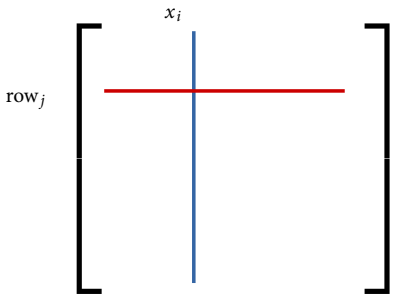

FIGURE 1 – A general setup for the equations described by rows of a system. We need $A_{j,i} \neq 0$ to update x_i based on equation j. We also cannot use equation j to update any other variables, nor can we use other equations to update x_i. The map matrix $\boldsymbol{M}$ encodes the set of updates.

We can state this using a set of matrices with some slight additional notation. Let $\boldsymbol{D}_M$ be the *diagonal* matrix where $[\boldsymbol{D}_M]i, i = A_{M_i,i}$. The idea with $\boldsymbol{D}_M$ is that we can write

$$\mathbf{x}^{\text{new}} = \boldsymbol{D}_M^{-1} \text{some vector}$$

where entries of that vector correspond to $b_{M_i} - \sum_{j \neq i} A_{M_i,j} x_j)$. Consequently, let $\mathbf{b}_M$ be the vector $[\mathbf{b}_M]_i = b_{M_i}$. Also, let $\boldsymbol{N}_M$ be the matrix with entries:

$$[\boldsymbol{N}_M]_{i,j} = \begin{cases} 0 & i = j \\ A_{M_i,j} & i \neq j. \end{cases}$$

$$\mathbf{x}^{\text{new}} = \boldsymbol{D}_M^{-1}(\mathbf{b}_M - \boldsymbol{N}_M \mathbf{x}^{\text{old}}.$$

This iteration is called the *Jacobi* method for solving a linear system of equations.

3 IMPLEMENTATIONS OF JACOBI

```
"""
    jacobi_iteration_map_!(y,A,x,M) sets y to be the next Jacobi iteration from x with map M
"""
function jacobi_iteration(y,A,x,M=1:length(x))
  for i=1:length(x)
    y[i] -= (b[M[i] - A[M[i],:]'*x - A[M[i],i]*x[i]) / A[M[i],i]
  end
end
```

4 A 3X3 EXAMPLE WITH A PERMUTATION INSTEAD OF THE MAP

Most derivations of the Jacobi iteration assume that $\boldsymbol{D}$ is formed from the non-zero diagonal of the linear system of equations, but there is no such restriction in the derivation

of the method. This is simply a notational convenience. All we need is a permutation matrix $\boldsymbol{P}$ such that $\boldsymbol{P} \odot \boldsymbol{A}$ is non-singular to build the matrix $\boldsymbol{M}$ for the above iteration to work.[4]

[4] Here $\odot$ is the element-wise, or Hadamard, products.

Consider the following linear system

$$\begin{bmatrix} 1 & 0 & -3 \\ 0 & -3 & 1 \\ -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{1}$$

Then if we run the Jacobi update with

$$M_1 = 3 \qquad M_2 = 2 \qquad M_3 = 1 \text{ and } \mathbf{x}^{(0)} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$$

We have

$$x_1^{\text{new}} = \frac{1}{A_{3,1} = -3}\left(1 - (A_{3,2} = 0) \cdot x_2 - (A_{3,3} = 1) \cdot x_3\right) = 0$$

$$x_2^{\text{new}} = \frac{1}{(A_{2,2} = -3)}\left(1 - (A_{2,1} = 0) \cdot x_1 - (A_{2,3} = 1) \cdot x_3\right) = 0$$

$$x_3^{\text{new}} = \frac{1}{(A_{1,3} = -3)}\left(1 - (A_{1,1} = 1) \cdot x_1 - (A_{1,2} = 0) \cdot x_2\right) = 0.$$

If we write this as a matrix update, we have:

$$\mathbf{x}^{(k+1)} = \begin{bmatrix} -3 & & \\ & -3 & \\ & & -3 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -3 & & \\ & -3 & \\ & & -3 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \mathbf{x}^{(k)}.$$

To avoid the details with $\boldsymbol{M}$, let us simply *permute* the three rows of $\boldsymbol{A}$ so that row 3 comes first, then row 2, then row 1. Note that we simply reordered the rows, not the variables.

$$\begin{bmatrix} -3 & 0 & 1 \\ 0 & -3 & 1 \\ 1 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

In this case, we can simply split the matrix into it's two pieces:

$$\boldsymbol{D} = \text{ diagonal } = \begin{bmatrix} -3 & & \\ & -3 & \\ & & -3 \end{bmatrix} \qquad \boldsymbol{N} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

With this setup, we can easily write the iteration

$$\mathbf{x}^{(k+1)} = \boldsymbol{D}^{-1}\mathbf{b} - \boldsymbol{N}\mathbf{x}^{(k)}.$$

5 JACOBI WITH A PERMUTATION MATRIX

Let us derive the same update as above, using the permutation to show how it generalizes the previous algorithm. That's because the matrix $\boldsymbol{M}$ really is a permutation! A permutation matrix is just a way to reorder the rows or columns of a matrix. It reorders the rows if we multiply on the left.

For a permutation matrix $\boldsymbol{P}$, we have $P_{i,j} = 1$ if $\mathbf{y} = \boldsymbol{P}\mathbf{x}$ has $y_i = x_j$. That is, $P_{i,j}$ if i in the output was j in the input. So we can write:

$$\mathbf{b}_M = \boldsymbol{P}\mathbf{b} \text{ where } P_{i,j} = \begin{cases} 1 & j = M_i \\ 0 & \text{otherwise.} \end{cases}$$

We can also apply the same permutation to $\boldsymbol{A}$ as in

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \text{ then } \boldsymbol{PA} = \begin{bmatrix} \mathbf{a}_{M_1}^T \\ \mathbf{a}_{M_2}^T \\ \vdots \\ \mathbf{a}_{M_n}^T \end{bmatrix}.$$

This corresponds with solving a linear system

$$\boldsymbol{PA}\mathbf{x} = \boldsymbol{P}\mathbf{x}$$

or really, just re-ordering the equations we were given. Now note that

$$\boldsymbol{D}_M = \text{diagonal elements of } \boldsymbol{PA} \text{in a diagonal matrix.}$$

Then also,

$$\boldsymbol{N}_M = \boldsymbol{PA} - \boldsymbol{D}_M.$$

This is why most textbooks do not describe the setup with using an equation to solve for an unknown, it's entirely equivalent to the following setup that is much closer to a typical description of Jacobi

1. Pick a permutation matrix $\boldsymbol{P}$ such that $\boldsymbol{PA}$ has non-zero diagonal elements.
2. Let $\boldsymbol{D}$ be the diagonal elements of $\boldsymbol{PA}$.
3. Let $\boldsymbol{N}$ be the matrix $\boldsymbol{PA} - \boldsymbol{D}$

Then the Jacobi method implements the iteration:

$$\mathbf{x}^{(k+1)} = \boldsymbol{D}^{-1}(\boldsymbol{P}\mathbf{b} - \boldsymbol{N}\mathbf{x}^{(k)}).$$

This makes it much simplier to do the analysis below when we are interested in issues of convergence: permute the matrix, then look at the diagonal entries and t

EXAMPLE 1 *Show that the linear system* (1) *will not converge using Jacobi with the* standard *permutation* $\boldsymbol{P} = \boldsymbol{I}$.

EXAMPLE 2 *Note that the 3x3 least squares problem for our quadratic fitting example will not converge with Jacobi for* any *iteration.*

5.1 INTERESTING TANGENT

Of course, this begs the question, why do we need a *single* permutation matrix $\boldsymbol{P}$? Why can't we get away with using a sequence of iterations where we just ensure that each element is updated every so-often. I'm almost sure this has been studied, but don't know the reference off the top of my head. Or even a random pair at each step.

6 THE CONVERGENCE OF THE JACOBI METHOD

It's easy to determine the convergence of the Jacobi matrix with our knowledge of the spectral-radius of a matrix. Let's look at the error in the kth-step of the method:

$$\mathbf{x}^{(k+1)} - \mathbf{x} = \boldsymbol{D}_M^{-1}(\mathbf{b} - \boldsymbol{N}_M\mathbf{x}^{(k)}) - \mathbf{x}.$$

But note that we designed this so that $\mathbf{x}$ is a fixed point of the update, so $\mathbf{x} = \boldsymbol{D}_M^{-1}(\mathbf{b} - \boldsymbol{N}_M\mathbf{x})$ as well. This means that

$$\mathbf{x}^{(k+1)} - \mathbf{x} = \boldsymbol{D}_M^{-1}(\mathbf{b} - \boldsymbol{N}_M\mathbf{x}^{(k)} - \mathbf{b} + \boldsymbol{N}_M\mathbf{x}) = -\boldsymbol{D}_M^{-1}\boldsymbol{N}_M(\mathbf{x}^{(k)} - \mathbf{x}).$$

Consequently,

$$\mathbf{x}^{(k+1)} = (-\boldsymbol{D}_M^{-1}\boldsymbol{N}_M)^{k+1}(\mathbf{x}^{(0)} - \mathbf{x}).$$

This converges, for all starting points $\mathbf{x}^{(0)}$ if and only if $\rho(-\boldsymbol{D}_M^{-1}\boldsymbol{N}_M) < 1$.

7 A MISTAKE IN AN IMPLEMENTATION OF THE JACOBI METHOD

If we are solving large linear systems of equations, then we may have vectors with *billions* of entries! This means that storing another vector $\mathbf{x}^{\text{new}}$ may be expensive itself. In this case, I hope you can agree that it may occur to someone to try and save memory as follows:

just update the solution to $\mathbf{x}$ with only a single set of memory!

For instance, consider the following implementation of the Jacobi iteration

```
"""
    jacobi_iteration!(y,A,x,M) updates x "like" Jacobi, but in-place, with map M
"""
function jacobi_iteration!(A,b,x,M=1:length(x))
  for i=1:length(x)
    x[i] -= (b[M[i]] - A[M[i],:]'*x - A[M[i],i]*x[i]) / A[M[i],i]
  end
end
```

This is wrong if the objective is to implement the Jacobi iteration. However, it turns out that this idea gives rise to a method called the Gauss-Seidel method.

$$x_1 = \frac{1}{A_{M_1,1}}(b_{M_1} - \sum_{j \neq M_1} A_{M_1,j} x_j)$$

$$\dots$$

$$x_i^{\text{new}} = \frac{1}{A_{M_i,1}}(b_{M_i} - \sum_{j \neq M_i} A_{M_i,j} x_j)$$

$$\dots$$

8 ANALYZING GAUSS-SEIDEL

9 THE GAUSS-SEIDEL AND STEEPEST DESCENT METHOD

Note, you can show that Gauss-Seidel converges on any symmetric positive definite matrix using the matrix $\boldsymbol{M} = \boldsymbol{I}$.

We can derive the Gauss-Seidel method as a mistake in Jacobi. Let's now consider what happens on a symmetric matrix $\boldsymbol{A}$ with unit diagonals. We have:

$$x_1^{\text{new}} = (b_1 - \sum_{j>1} A_{1,j} x_j^{\text{old}})$$

$$\dots$$

$$x_i^{\text{new}} = (b_i - \sum_{j<i} x_j^{\text{new}} - \sum_{j>i} A_{1,j} x_j^{\text{old}})$$

Recall the iteration for coordinate descent:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma_k \mathbf{e}_i \text{ where } \gamma_k = -[\boldsymbol{A}\mathbf{x} - \mathbf{b}]_i / A_{i,i}$$

Written in terms of our problem

$$x_i^{\text{new}} = x_i - \sum_{j=1} A_{ij} \mathbf{x}^{\text{old}} + b_i = b_i - \sum_{j \neq i} A_{i,j} x_j^{\text{old}}.$$

This shows that if we update the ith variable, then we are doing a closely related update to Gauss-Seidel. To see that they are the same, remember how we arrived at Gauss-Seidel, we simply did the Jacobi update but *forgot* to allocate new memory. This means that, in the program, we have:

$$x_i^{\text{new}} = (b_i - \sum_{j<i} x_j^{\text{cur}} - \sum_{j>i} A_{1,j} x_j^{\text{cur}})$$

And now we can see that, expressed this way, Gauss-Seidel update is exactly the same as steepest descent.