MATH3511/6111: Scientific Computing

11. Iterative Linear Solvers

Lindon Roberts

Semester 1, 2022

Office: Hanna Neumann Building #145, Room 4.87
Email: lindon.roberts@anu.edu.au
Based on lecture notes written by S. Roberts, L. Stals, Q. Jin, M. Hegland, K. Duru.

Linear Systems

From the last module, there are two basic categories of algorithm for solving linear systems:

- Direct methods: algorithms which produce an exact solution (assuming no rounding errors) in a finite number of steps.
- Iterative methods: algorithms which generate a sequence of vectors converging to the solution, $\lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}$.

Now we will discuss iterative methods for solving linear systems.

Iterative vs. Direct Methods

Direct methods:

- Finishes in a fixed number of steps/flops (possibly large)
- Errors from ill-conditioning and rounding
- Based on matrix factorisations

Iterative methods:

- Returns an approximate solution at every step, but a good solution after a variable number of steps
- Error depends on number of iterations
- Generally less affected by rounding errors, fault tolerant
- Sometimes only requires matrix-vector products (don't need the actual entries of A) — see Fast Fourier Transform, for example.

Both types are useful in different situations.

Iterative Methods: Basic Idea

The basic principle behind many iterative methods for $A\boldsymbol{x} = \boldsymbol{b}$ is: find a matrix M with

- $M \approx A$ (in the sense that $\|M - A\|$ is small in some norm)
- $M\boldsymbol{x} = \boldsymbol{b}$ is easy to solve — what types of matrices have this property?

Broadly speaking, these two properties are conflicting: think about $M = A$ or $M = I$.

We will discuss some reasonable choices for M shortly.

Iterative Methods: Basic Idea

For now, let's say we have found $M \approx A$ which is easy to solve with.

Suppose after k steps we have an approximate solution $\boldsymbol{x}^k \approx \boldsymbol{x}$. Then the residual is

$$\boldsymbol{r}^k = \boldsymbol{b} - A\boldsymbol{x}^k = A(\boldsymbol{x} - \boldsymbol{x}^k).$$

If we use $M \approx A$, we get

$$\boldsymbol{x} \approx \boldsymbol{x}^k + M^{-1}\boldsymbol{r}^k.$$

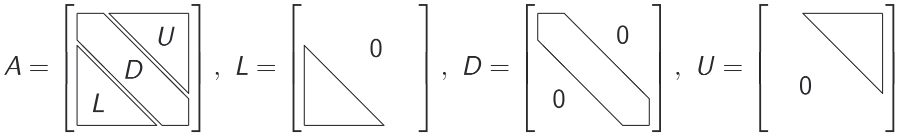
Based on this idea, our iterative method is

$$\boxed{\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + M^{-1}(\boldsymbol{b} - A\boldsymbol{x}^k)} \qquad \text{or, after rearranging,} \qquad \boxed{M\boldsymbol{x}^{k+1} = \boldsymbol{b} - (A - M)\boldsymbol{x}^k}$$

We will write our methods in either of these forms (whatever is more useful at the time).

Splitting Methods

One way to build M is by splitting A into its diagonal and upper/lower triangular parts:



That is, given an $n \times n$ matrix A, we can write A in the form

$$A = L + D + U,$$

where L is the lower triangular part (strictly below the diagonal), D is the diagonal part, and U is the (strictly) upper triangular part.

Splitting Methods

Example

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix}$$

We can write A as

$$A = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}}_{L} + \underbrace{\begin{bmatrix} 6 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 4 \end{bmatrix}}_{D} + \underbrace{\begin{bmatrix} 0 & -2 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{U}$$

Jacobi's Method

We can use the splitting $A = L + D + U$ to give us a choice for $M \approx A$ (easy to invert).

Jacobi's Method (Carl Jacobi, 1845) uses the choice $\boxed{M = D}$:

$$\boxed{\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + D^{-1}(\boldsymbol{b} - A\boldsymbol{x}^k)}$$

or alternatively, since $A - M = (L + D + U) - D = L + U$, we can write

$$D\boldsymbol{x}^{k+1} = \boldsymbol{b} - (A - M)\boldsymbol{x}^k,$$

and so

$$\boxed{\boldsymbol{x}^{k+1} = D^{-1}[\boldsymbol{b} - (L + U)\boldsymbol{x}^k]}$$

(assuming D is invertible — what conditions on D guarantee this?)

Jacobi's Method

$$\boxed{\boldsymbol{x}^{k+1} = D^{-1}[\boldsymbol{b} - (L+U)\boldsymbol{x}^k]}$$

Written out in components, this is

$$x_i^{k+1} = \frac{1}{a_{i,i}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{i,j}x_j^k\right), \qquad \forall i = 1, \ldots, n$$

Notes:

- We assume we have a starting point $\boldsymbol{x}^0$ — this can be anything, but a good guess always helps (e.g. previously computed solution to a similar problem).
- Once $\boldsymbol{x}^k$ is known, each entry of $\boldsymbol{x}^{k+1}$ can be calculated independently of all other entries — useful for parallel processing.

Jacobi's Method: Example

Example

Use Jacobi's Method to solve $A\boldsymbol{x} = \boldsymbol{b}$, where

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{b} = \begin{bmatrix} -1 \\ 8 \\ 8 \end{bmatrix}.$$

Note: the true solution is $\boldsymbol{x} = \begin{bmatrix} -0.5 & 1 & 2 \end{bmatrix}^T$.

The general Jacobi method is

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

Jacobi's Method: Example

Example

Use Jacobi's Method to solve $A\boldsymbol{x} = \boldsymbol{b}$, where

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ 8 \\ 8 \end{bmatrix}.$$

For this problem, the iteration is

$$\begin{aligned} x_1^{k+1} &= \frac{1}{6}\left(-1 + 2x_2^k - 2x_3^k\right), \\ x_2^{k+1} &= \frac{1}{5}\left(8 + 2x_1^k - x_3^k\right), \\ x_3^{k+1} &= \frac{1}{4}\left(8 - 2x_1^k - x_2^k\right). \end{aligned}$$

Jacobi's Method: Example

Starting from $\boldsymbol{x}^0 = \mathbf{0}$, we get

```
 k                          xk
 0 [ 0.000000000000  0.000000000000  0.000000000000]
 1 [-0.166666666667  1.600000000000  2.000000000000]
 2 [-0.300000000000  1.133333333333  1.683333333333]
 3 [-0.350000000000  1.143333333333  1.866666666667]
 4 [-0.407777777778  1.086666666667  1.889166666667]
 5 [-0.434166666667  1.059055555556  1.932222222222]
 6 [-0.457722222222  1.039888888889  1.952319444444]
 7 [-0.470810185185  1.026447222222  1.968888888889]
 8 [-0.480813888889  1.017898148148  1.978793287037]
 9 [-0.486965046296  1.011915787037  1.985932407407]
10 [-0.491338873457  1.008027500000  1.990503576389]
...
30 [-0.499997076705  1.000002694375  1.999996814992]
```

After 30 iterations, we have about 5–6 digits of accuracy.

Gauss-Seidel Method

Another choice of $M \approx A$ easy to invert is $\boxed{M = L + D}$.

This is probably a better approximation than $M = D$, and it is still reasonably easy to solve linear systems with M (lower triangular).

Using this choice of M is called the Gauss-Seidel Method (Carl Friedrich Gauss 1823; Philipp Ludwig von Seidel 1874):

$$\boxed{\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} + (L + D)^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{k})}$$

or alternatively, since $A - M = (L + D + U) - (L + D) = U$, we can write

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

We can rewrite this cleverly...

Gauss-Seidel Method

Start with

$$\begin{aligned}(L+D)\boldsymbol{x}^{k+1} &= \boldsymbol{b} - U\boldsymbol{x}^k, \\ D\boldsymbol{x}^{k+1} &= \boldsymbol{b} - L\boldsymbol{x}^{k+1} - U\boldsymbol{x}^k,\end{aligned}$$

and so

$$\boxed{\boldsymbol{x}^{k+1} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{k+1} - U\boldsymbol{x}^k\right)}$$

- Only have to invert D, not $L+D$ (quicker)
- Just Jacobi but with $L\boldsymbol{x}^{k+1}$ instead of $L\boldsymbol{x}^k$.
- But the right-hand side depends on $\boldsymbol{x}^{k+1}$?!

Gauss-Seidel Method

$$\boxed{\boldsymbol{x}^{k+1} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{k+1} - U\boldsymbol{x}^{k}\right)}$$

The right-hand side depends on $\boldsymbol{x}^{k+1}$, but this is ok because L is strictly lower triangular.

In components, we get

$$x_i^{k+1} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{k+1} - \sum_{j=i+1}^{n} a_{i,j}x_j^{k}\right), \qquad i = 1, \ldots, n$$

This time, we need to compute $i = 1$ first, then $i = 2$, etc. (can't parallelise like Jacobi).

This is essentially the same as doing forward substitution with matrix $L + D$.

Gauss-Seidel Method: Example

Example

Use the Gauss-Seidel Method to solve $A\boldsymbol{x} = \boldsymbol{b}$, where

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ 8 \\ 8 \end{bmatrix}.$$

Note: the true solution is $\boldsymbol{x} = \begin{bmatrix} -0.5 & 1 & 2 \end{bmatrix}^T$.

The general Gauss-Seidel method is

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

Gauss-Seidel Method: Example

Example

Use the Gauss-Seidel Method to solve $A\boldsymbol{x} = \boldsymbol{b}$, where

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ 8 \\ 8 \end{bmatrix}.$$

For this problem, the iteration is

$$\begin{aligned} x_1^{k+1} &= \frac{1}{6}\left(-1 + 2x_2^k - 2x_3^k\right), \\ x_2^{k+1} &= \frac{1}{5}\left(8 + 2x_1^{k+1} - x_3^k\right), \\ x_3^{k+1} &= \frac{1}{4}\left(8 - 2x_1^{k+1} - x_2^{k+1}\right). \end{aligned}$$

Gauss-Seidel Method: Example

Starting from $\boldsymbol{x}^0 = \mathbf{0}$, we get

```
 k                         xk
 0 [ 0.000000000000  0.000000000000  0.000000000000]
 1 [-0.166666666667  1.533333333333  1.700000000000]
 2 [-0.222222222222  1.171111111111  1.818333333333]
 3 [-0.382407407407  1.083370370370  1.920361111111]
 4 [-0.445663580247  1.037662345679  1.963416203704]
 5 [-0.475251286008  1.017216244856  1.983321581790]
 6 [-0.488701778978  1.007854972051  1.992387146476]
 7 [-0.494844058142  1.003584947448  1.996525792209]
 8 [-0.497646948254  1.001636062257  1.998414458563]
 9 [-0.498926132102  1.000746655447  1.999276402189]
10 [-0.499509915581  1.000340753330  1.999669769458]
...
30 [-0.499999999925  1.000000000052  1.999999999949]
```

After 30 iterations, we have about 10–11 digits of accuracy (Jacobi had 5–6 digits).

Error Analysis

Let's analyse our general iterative method

$$\boxed{\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + M^{-1}(\boldsymbol{b} - A\boldsymbol{x}^k)} \qquad \text{or} \qquad \boxed{M\boldsymbol{x}^{k+1} = \boldsymbol{b} - (A - M)\boldsymbol{x}^k}$$

Rearranging the first equation and subtracting the true solution $\boldsymbol{x}$, we get

$$\begin{aligned} \boldsymbol{x}^{k+1} - \boldsymbol{x} &= \boldsymbol{x}^k - \boldsymbol{x} + M^{-1}(A\boldsymbol{x} - A\boldsymbol{x}^k), \\ \boldsymbol{x}^{k+1} - \boldsymbol{x} &= (I - M^{-1}A)(\boldsymbol{x}^k - \boldsymbol{x}). \end{aligned}$$

For simplicity, define the error $\boldsymbol{e}^k = \boldsymbol{x}^k - \boldsymbol{x}$ and the matrix $E := I - M^{-1}A$. Then our error formula is

$$\boxed{\boldsymbol{e}^{k+1} = E\boldsymbol{e}^k}$$

So we have $\boldsymbol{e}^k = E\boldsymbol{e}^{k-1} = E^2\boldsymbol{e}^{k-2} = \cdots = E^k\boldsymbol{e}^0$.

Error Analysis

We want to look at the size of the error, so take 2-norms of both sides:

$$\|\boldsymbol{e}^{k+1}\|_2 = \|E\boldsymbol{e}^k\|_2 \leq \|E\|_2 \|\boldsymbol{e}^k\|_2.$$

Inductively we have

$$\boxed{\|\boldsymbol{e}^k\|_2 \leq \|E\|_2^k \|\boldsymbol{e}^0\|_2}$$

Theorem

If $\|E\|_2 < 1$, then for any choice $\boldsymbol{x}^0$ the sequence $\boldsymbol{x}^k$ converges to the true solution $\boldsymbol{x}$, and we have the error bound $\|\boldsymbol{e}^k\|_2 \leq \|E\|_2^k \|\boldsymbol{e}^0\|_2$.

So, we will converge faster if:

- $\|\boldsymbol{e}^0\|$ is small — so $\boldsymbol{x}^0$ is a good guess of the true solution.
- $\|E\|_2 = \|I - M^{-1}A\|_2$ is small — this is a quantitative measure of $M \approx A$.

Error Analysis

Actually, our splitting methods can converge under more general conditions than $\|E\|_2 < 1$. We will need a new concept:

Definition (Spectral Radius)

The spectral radius $\rho(B)$ of a square matrix B is the magnitude of the largest eigenvalue:

$$\rho(B) := \max_j |\lambda_j(B)|,$$

where $\lambda_j(B)$ is an eigenvalue of B (possibly complex).

Some important properties:

- $\rho(B) \leq \|B\|$ for any operator norm $\|\cdot\|$ (e.g. all matrix ℓ_p norms).
- $\rho(B)$ is not a norm: for instance, there exist matrices $B \neq 0$ with $\rho(B) = 0$.

Error Analysis

The main reason the spectral norm is useful is:

Theorem

For any square matrix B, $B^k \to 0$ if and only if $\rho(B) < 1$.

For our splitting methods, $\boldsymbol{e}^k = E^k \boldsymbol{e}^0$, so we get:

Theorem

$\boldsymbol{e}^k \to \mathbf{0}$ for any starting value of $\boldsymbol{e}^0$ if and only if $\rho(E) < 1$.

- If $\|E\|_2 < 1$ then $\rho(E) < 1$, but not the other way around. This convergence result is more general than our previous convergence result.
- If A is singular, then $\rho(E) \geq 1$ for any choice of M, so our method will not converge.
  - Proof: if A is singular, then there exists $\boldsymbol{x} \neq \mathbf{0}$ with $A\boldsymbol{x} = \mathbf{0}$. This gives $E\boldsymbol{x} = \boldsymbol{x}$, so E has an eigenvalue $\lambda = 1$.

Error Analysis

It is often difficult to check if $\rho(E) < 1$ for a particular A and splitting method (Jacobi or Gauss-Seidel). However, there are some classes of matrix where it is easy to prove $\rho(E) < 1$.

Definition (Diagonally Dominant)

A matrix A is strictly row diagonally dominant if

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}| < |a_{i,i}|, \qquad \text{for all rows } i = 1, \ldots, n.$$

For example, a strictly row diagonally dominant matrix is

$$\begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix}$$

Error Analysis

For the Jacobi method, we have

$$E_J = I - D^{-1}A = -D^{-1}(L+U), \qquad \text{and so} \qquad e_{i,j} = \begin{cases} -\frac{a_{i,j}}{a_{i,i}}, & i \neq j, \\ 0, & i = j. \end{cases}$$

Remember that the operator ∞-norm is the maximum absolute row sum. So we have

$$\|E_J\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |e_{i,j}| = \max_{i=1,\ldots,n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{|a_{i,j}|}{|a_{i,i}|}$$

Theorem

If A is strictly row diagonally dominant, then the Jacobi error matrix E_J has $\rho(E_J) \leq \|E_J\|_\infty < 1$. Hence the Jacobi method for $A\mathbf{x} = \mathbf{b}$ converges to the true solution $\mathbf{x}$ for any starting point $\mathbf{x}^0$.

Error Analysis

Actually, we get the same result for the Gauss-Seidel method (but this is harder to prove).

Theorem

If A is strictly row diagonally dominant, then the Gauss-Seidel error matrix E_{GS} has $\rho(E_{GS}) \le \|E_{GS}\|_\infty < 1$. Hence the Gauss-Seidel method for $A\mathbf{x} = \mathbf{b}$ converges to the true solution $\mathbf{x}$ for any starting point $\mathbf{x}^0$.

In fact, if A is strictly row diagonally dominant, we have $\|E_{GS}\|_\infty \le \|E_J\|_\infty < 1$.

This might suggest that Gauss-Seidel will converge quicker (like our example before), but this is not guaranteed: the error bound is an inequality ($\|\mathbf{e}^k\| \le \|E\|^k \|\mathbf{e}^0\|$), the true error may be much smaller than the bound.

Fixed Point Iterations

Our error analysis of Jacobi/Gauss-Seidel gives us an example of a fixed-point iteration. In general, we have an iteration

$$\boldsymbol{x}^{k+1} = \boldsymbol{F}(\boldsymbol{x}^k),$$

and we want our iterates $\boldsymbol{x}^k$ to converge to a solution $\boldsymbol{x}^*$ which is a fixed point of $\boldsymbol{F}$ (i.e. $\boldsymbol{x}^* = \boldsymbol{F}(\boldsymbol{x}^*)$).

If $\boldsymbol{F}$ is a contraction: for some $0 < \rho < 1$ we have

$$\|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{y})\| \leq \rho \|\boldsymbol{x} - \boldsymbol{y}\|, \qquad \forall \boldsymbol{x}, \boldsymbol{y},$$

then the sequence $\boldsymbol{x}^k$ is guaranteed to converge to $\boldsymbol{x}^*$ from any starting point $\boldsymbol{x}^0$, with an error bound $\|\boldsymbol{x}^k - \boldsymbol{x}^*\| \leq \rho^k \|\boldsymbol{x}^0 - \boldsymbol{x}^*\|$.

Analysing algorithms by treating them as fixed-point iterations is a common technique (e.g. can be used to analyse Newton's method for rootfinding, power method for finding eigenvalues).

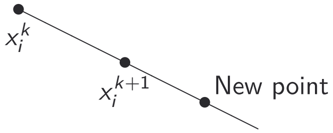
Successive Over-Relaxation

Just like extrapolation for numerical differentiation and quadrature, our iterative methods can be made much faster with very little work.

Here, we look at the Gauss-Seidel method

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

The idea is acceleration: don't just go from x_i^k to x_i^{k+1}, take a point that is a slightly bigger step in the same direction.



Successive Over-Relaxation

So, given the Gauss-Seidel points x_i^k and x_i^{k+1}, we look at

$$x_i^{k+1}(\omega) = x_i^k + \omega(x_i^{k+1} - x_i^k),$$

for some $\omega > 0$.

This gives us the Successive Over-Relaxation (SOR) method, written as:

$$x_i^{k+1} = (1-\omega)x_i^k + \frac{\omega}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{k+1} - \sum_{j=i+1}^{n} a_{i,j}x_j^k\right).$$

If $\omega = 1$, we just get x_i^{k+1} (regular Gauss-Seidel), and if $\omega > 1$ we are doing acceleration.

Note: if $\omega > 1$ we talk about "over-relaxation" and if $\omega < 1$ we call it "under-relaxation".

Successive Over-Relaxation

To write SOR as a splitting method, we try to solve the linear system $\omega A\boldsymbol{x} = \omega\boldsymbol{b}$ with the splitting

$$\omega A = [\omega L + D] + [(\omega - 1)D + \omega U].$$

We then take $M = \omega L + D$, and so the iteration is

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} + (\omega L + D)^{-1}(\omega\boldsymbol{b} - \omega A\boldsymbol{x}^{k})$$

or

$$(\omega L + D)\boldsymbol{x}^{k+1} = \omega\boldsymbol{b} - [(\omega - 1)D + \omega U]\boldsymbol{x}^{k}.$$

Successive Over-Relaxation: Example

Example

Use SOR to solve $A\boldsymbol{x} = \boldsymbol{b}$, where

$$A = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 5 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ 8 \\ 8 \end{bmatrix}.$$

For this problem, the iteration is

$$\begin{aligned} x_1^{k+1} &= (1-\omega)x_1^k + \frac{\omega}{6}\left(-1 + 2x_2^k - 2x_3^k\right), \\ x_2^{k+1} &= (1-\omega)x_2^k + \frac{\omega}{5}\left(8 + 2x_1^{k+1} - x_3^k\right), \\ x_3^{k+1} &= (1-\omega)x_3^k + \frac{\omega}{4}\left(8 - 2x_1^{k+1} - x_2^{k+1}\right). \end{aligned}$$

Successive Over-Relaxation: Example

Starting from $\boldsymbol{x}^0 = \mathbf{0}$ and with $\omega = 1.2$, we get

```
 k                         xk
 0 [ 0.000000000000  0.000000000000  0.000000000000]
 1 [-0.200000000000  1.824000000000  1.972800000000]
 2 [-0.219520000000  0.976358400000  1.844244480000]
 3 [-0.503250432000  1.040549437440  2.020936531968]
 4 [-0.491504751411  0.990943064162  1.993432625204]
 5 [-0.502694874135  1.002094017534  2.002302194180]
 6 [-0.499544295831  0.999247407891  1.999491916296]
 7 [-0.500188944196  1.000181765297  2.000160453669]
 8 [-0.499953686510  0.999947368535  1.999955910611]
 9 [-0.500012679528  1.000015021573  2.000011919123]
10 [-0.499996223115  0.999995948001  1.999996565644]
...
30 [-0.500000000000  1.000000000000  2.000000000000]
```

After 30 iterations, we have accuracy to machine precision (Gauss-Seidel had 10–11 digits).

Successive Over-Relaxation: Analysis

There are some theoretical results for SOR. The first tells us that we must always choose $0 < \omega < 2$.

Theorem

If $\omega \leq 0$ or $\omega \geq 2$, then SOR diverges for every starting point $\mathbf{x}^0$.

With this restriction, we are guaranteed to converge if A is symmetric positive definite.

Theorem

If A is symmetric positive definite and $0 < \omega < 2$, then SOR converges to the true solution for any starting point $\mathbf{x}^0$.

Since we can take $\omega = 1$, this also means Gauss-Seidel converges if A is symmetric positive definite.