6 Iterative methods for linear systems

Iterative methods for solving linear systems

$$A\mathbf{x} = \mathbf{b}$$

are methods that are initialized with an arbitrary guess $\mathbf{x}^{(0)}$ and iteratively improve this guess until the solution of the linear system is achieved up to some predefined accuracy. Such methods are usually applied when direct methods are too expensive or impossible to use. In most cases, the matrices at hand are conveniently available as matrix-vector product. This will be the case when the matrices are too large (and possibly sparse) or are only given as operators (functions) for multiplying them with a vector.

Reminder An iterative method is defined as

$$\mathbf{x}^{(k+1)} = \phi(\mathbf{x}^{(k)}),$$

which simply "looks" only at one previous vector or

$$\mathbf{x}^{(k+1)} = \phi(\mathbf{x}^{(k)}, ..., \mathbf{x}^{(0)}),$$

where ϕ depends on many or even all the previously generated vectors. Denote by $\mathbf{x}^*$ the solution of the linear system, i.e. $A\mathbf{x}^* = \mathbf{b}$. The first requirement from ϕ is that it converges:

$$\lim_{k\to\infty} \{\mathbf{x}^{(k)}\} = \mathbf{x}^*.$$

The second requirement is that the method converges as fast as possible. We define a convergence rate to be

$$\lim_{k\to\infty} \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\|}{\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^p} = C,$$

where $\|\cdot\|$ is some norm, $p \geq 1$ is a scalar denoted as the order of convergence, and $C \neq 0$ is constant called the convergence factor. In all cases here, we will have a linear convergence rate ($p = 1$), and then we will require $|C| < 1$ for convergence. By the norm equivalence theorem, it doesn't matter in which norm we use in the description above to determine the rate of convergence.

Remark 11 (Error and residual)**.** *The vector*

$$\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$$

is called the error vector at iteration k. This vector is the vector that needs to go to zero for the iteration to converge. Closely related is the residual vector

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)},$$

which is also $\mathbf{r}^{(k)} = A\mathbf{e}^{(k)}$. The key difference between the two is that we cannot measure the error without knowing the solution, but we can measure the residual. Note that convergence means

$$\lim_{k\to\infty}\{\mathbf{e}^{(k)}\} = \lim_{k\to\infty}\{\mathbf{r}^{(k)}\} = 0.$$

6.1 Simple iterative methods

A general one-point iterative method splits the matrix A into two: $A = M + N$. Then the linear system is written as

$$M\mathbf{x} + N\mathbf{x} = \mathbf{b}, \quad \textit{or} \quad M\mathbf{x} + (A - M)\mathbf{x} = \mathbf{b}$$

and then the iteration is defined as:

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

The matrix M, which is also called the preconditioner, needs to be inverted at every iteration, meaning that a linear system $M\mathbf{v} = \mathbf{r}$ has to be solved in every iteration. The cost of the solution is naturally comprised from the number of iterations times the work needed to "invert" M. Some methods use a sophisticated M and require a few iterations while others have a simple M and may require many iterations. We usually stop the iterations if one of the following is satisfied for some tolerance ϵ:

$$\frac{\|A\mathbf{x}^{(k)} - \mathbf{b}\|}{\|\mathbf{b}\|} < \epsilon \quad \text{or} \quad \frac{\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|}{\|\mathbf{x}^{(k)}\|} < \epsilon.$$

The left term indicates that the residual is low enough compared to a zero solution, while the other criterion indicates that the *relative* change in the iterations is small enough. The general iterative method can be found in Alg. 7.

Algorithm: General Iterative Method
 # Input: $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $M, N \in \mathbb{R}^{n\times n}$,
maxIter, ϵ, *Convergence criterion*
 # Output: $\mathbf{x}$ *s.t* $A\mathbf{x} \approx \mathbf{b}$
for $k = 1, ..., maxIter$ **do**
  Apply iteration:
  $\mathbf{x}^{(k)} = M^{-1}(\mathbf{b} - N\mathbf{x}^{(k-1)})$ or $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + M^{-1}(\mathbf{b} - A\mathbf{x}^{(k-1)})$,
  if $\frac{\|A\mathbf{x}^{(k)}-\mathbf{b}\|}{\|\mathbf{b}\|} < \epsilon$ *or alternatively* $\frac{\|\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}\|}{\|\mathbf{x}^{(k)}\|} < \epsilon$. **then**
    Convergence is reached, stop the iterations.
  end
end
Return $\mathbf{x}^{(k)}$ as the solution.

Algorithm 7: General Iterative Method

6.1.1 The Jacobi method

Assume that we need to solve $A\mathbf{x} = \mathbf{b}$:

$$\begin{cases} 4x_1 - x_2 + x_3 = 7 \\ 4x_1 - 8x_2 + x_3 = -21 \\ -2x_1 + x_2 + 5x_3 = 15 \end{cases} \quad \text{or} \quad \begin{bmatrix} \mathbf{4} & -1 & 1 \\ 4 & \mathbf{-8} & 1 \\ -2 & 1 & \mathbf{5} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -21 \\ 15 \end{bmatrix} \tag{34}$$

We can see that the matrix A has a dominant diagonal, so it can be approximated well by a diagonal matrix. Let us split the matrix:

$$A = D + L + U = \begin{bmatrix} \mathbf{4} & & \\ & \mathbf{-8} & \\ & & \mathbf{5} \end{bmatrix} + \begin{bmatrix} & & \\ 4 & & \\ -2 & 1 & \end{bmatrix} + \begin{bmatrix} & -1 & 1 \\ & & 1 \\ & & \end{bmatrix},$$

that is D is the diagonal part of A, L is the lower triangular part of A and U is the upper

triangular part. We choose $M = D$. The method then becomes (in matrix form):

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L+U)\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} + D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)}). \tag{35}$$

In scalar form the iteration will be:

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

In our example this will be:

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(7 + x_2^{(k)} - x_3^{(k)}) \\ \frac{1}{8}(-21 + 4x_1^{(k)} + x_3^{(k)}) \\ \frac{1}{5}(15 + 2x_1^{(k)} - x_2^{(k)}) \end{bmatrix}$$

Running the iterations from a guess $x^{(0)} = [1, 2, 2]$ yields:

```
Iter:  0:  [1.0, 2.0, 2.0]
Iter:  1:  [1.75, 3.375, 3.0]
Iter:  2:  [1.84375, 3.875, 3.025]
Iter:  3:  [1.9625, 3.925, 2.9625]
Iter:  4:  [1.99063, 3.97656, 3.0]
Iter:  5:  [1.99414, 3.99531, 3.00094]
Iter:  6:  [1.99859, 3.99719, 2.99859]
Iter:  7:  [1.99965, 3.99912, 3.0]
Iter:  8:  [1.99978, 3.99982, 3.00004]
Iter:  9:  [1.99995, 3.99989, 2.99995]
```

and the convergence history graphs are in Fig. 6.

We can see that the Jacobi iteration looks as follows:

$$\mathbf{x}^{(k+1)} = \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{7}{4} \\ \frac{21}{8} \\ \frac{15}{5} \end{bmatrix} + \begin{bmatrix} 0 & \frac{1}{4} & \frac{-1}{4} \\ \frac{4}{8} & 0 & \frac{1}{8} \\ \frac{2}{5} & \frac{-1}{5} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \mathbf{f} - D^{-1}(L+U)\mathbf{x}^{(k)} = \mathbf{f} - T\mathbf{x}^{(k)}$$

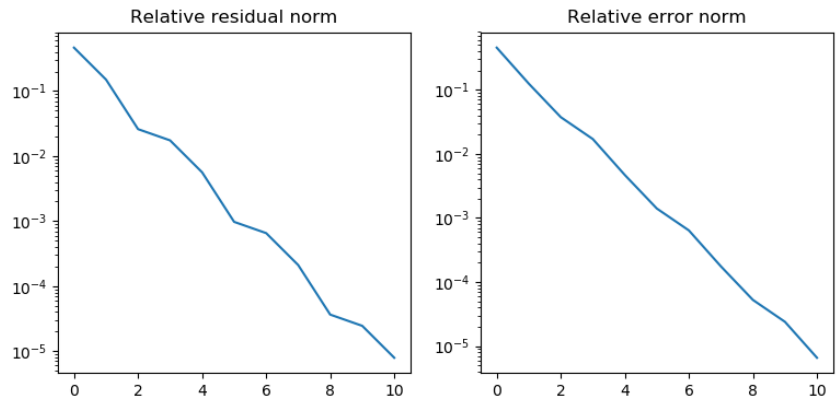

Figure 6: The residual and error history norm for the Jacobi iterations. Note the logarithmic scale of the y axis, when plotting convergence history.

6.1.2 Convergence of simple iterative methods

Now, back to the general iteration for solving a linear system:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + M^{-1}(\mathbf{b} - A\mathbf{x}^{(k)}).$$

From this equation we can get an equation for the error:

$$\mathbf{x}^* - \mathbf{x}^{(k+1)} = \mathbf{x}^* - \mathbf{x}^{(k)} - M^{-1}(A\mathbf{x}^* - A\mathbf{x}^{(k)}).$$

where $\mathbf{x}^*$ is the solution of the linear system, i.e. $A\mathbf{x}^* = \mathbf{b}$. Recall the definition of the error vector $\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$, and residual $\mathbf{r}^{(k)} = A\mathbf{e}^{(k)}$, the iteration matrix for the error is given by

$$\mathbf{e}^{(k+1)} = (I - M^{-1}A)\mathbf{e}^{(k)}.$$

Writing $T = (I - M^{-1}A)$ we get

$$\mathbf{e}^{(k+1)} = T\mathbf{e}^{(k)} = T^2\mathbf{e}^{(k-1)} = ... = T^{k+1}\mathbf{e}^{(0)}.$$

To get a condition for the convergence, let us assume that T is diagonalizable, with eigenpairs $(\lambda_i, \mathbf{v}_i)$, that is $T\mathbf{v}_i = \lambda_i\mathbf{v}_i$. We assume that the initial error is a combination of thew

eigenvectors: $\mathbf{e}^{(0)} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i$. Then,

$$\mathbf{e}^{(k+1)} = T^{k+1}\mathbf{e}^{(0)} = T^{k+1}\sum_{i=1}^{n} \alpha_i \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i \lambda_i^{k+1} \mathbf{v}_i \tag{36}$$

The error $\mathbf{e}^{(k+1)}$ will go to 0 (as $k \to \infty$) only if the largest eigenvalue in magnitude is smaller than 1. This conclusion brings us back to the definition of the Spectral radius: The spectral radius of T, denoted by $\rho(T)$ equals to the largest eigenvalue in magnitude

$$\rho(T) = \max_{1 \le i \le n} |\lambda_i|.$$

We thus conclude with the following theorem (formal proof relies on the expression above):

Theorem 16. *Assume that A is invertible. The general iteration*

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

converges for any starting vector $\mathbf{x}^{(0)}$ if and only if

$$\rho(I - M^{-1}A) < 1.$$

This spectral radius is also the convergence factor of the iteration. That is, for every vector norm

$$\lim_{k\to\infty} \frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} = \rho(I - M^{-1}A).$$

Proof. The first part of the theorem is achieved directly from the arguments above. We will now prove the convergence factor claim. Let us assume that $\lambda_{i_{max}}$ is the maximal eigenvalue of the error iteration matrix, and let's assume that it is unique (the theorem still holds if not). Following Eq. (36) we have

$$\frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} = \frac{\|\sum_{i=1}^{n} \alpha_i \lambda_i^{k+1} \mathbf{v}_i\|}{\|\sum_{i=1}^{n} \alpha_i \lambda_i^{k} \mathbf{v}_i\|} = \frac{\frac{1}{\lambda_{i_{max}}^k}\|\sum_{i=1}^{n} \alpha_i \lambda_i^{k+1} \mathbf{v}_i\|}{\frac{1}{\lambda_{i_{max}}^k}\|\sum_{i=1}^{n} \alpha_i \lambda_i^{k} \mathbf{v}_i\|} \tag{37}$$

$$= \frac{\|\sum_{i=1}^{n} \alpha_i (\frac{\lambda_i}{|\lambda_{i_{max}}|})^k \lambda_i \mathbf{v}_i\|}{\|\sum_{i=1}^{n} \alpha_i (\frac{\lambda_i}{|\lambda_{i_{max}}|})^k \mathbf{v}_i\|} \to_{k\to\infty} \frac{\|\alpha_{i_{max}} \lambda_{i_{max}} \mathbf{v}_{i_{max}}\|}{\|\alpha_{i_{max}} \mathbf{v}_{i_{max}}\|} = |\lambda_{i_{max}}| \tag{38}$$

□

Note that a spectral radius is hard to compute. As a result, we may often try to use matrix norms to check convergence, since any matrix norm upper bounds the spectral radius. That is,

$$\|I - M^{-1}A\| < 1 \Rightarrow \rho(I - M^{-1}A) < 1,$$

and if we found a norm for which $\|I - M^{-1}A\| < 1$, then our method converges.

Back to our example, the iteration matrix for the Jacobi method is given by:

$$\begin{bmatrix} 0 & \frac{1}{4} & \frac{-1}{4} \\ \frac{4}{8} & 0 & \frac{1}{8} \\ \frac{2}{5} & \frac{-1}{5} & 0 \end{bmatrix} \Rightarrow \|T\|_\infty = \frac{5}{8} < 1.$$

It means the Jacobi method converges, and its convergence factor is at least $\frac{5}{8}$ (that is an upper bound. The method may converge faster).

6.1.3 The Gauss-Seidel (GS) method

The Gauss Seidel method is in some way very similar to Jacobi, but still very different in concept. In this method, we iterate over all entries i of $\mathbf{x}$, and change each scalar entry x_i so that the i-th equation is satisfied (or $r_i = 0$ for the residual vector $\mathbf{r}$). In scalar form, the method is given by

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

This writing is very similar to the one of Jacobi, having the updated values $x_j^{(k+1)}$ instead of $x_j^{(k)}$ for $j < i$. Indeed, the GS method can be implemented exactly as the Jacobi method with the only difference that in the implementation we actually can iterate on one vector $\mathbf{x}$ instead of two copies for $\mathbf{x}^{(k+1)}$ and $\mathbf{x}^{(k)}$, as in Jacobi. This makes the GS algorithm inheritably sequential and much harder to parallelize efficiently compared to Jacobi.

In vector form, the GS method is achieved by the splitting:

$$(L + D)\mathbf{x} = \mathbf{b} - U\mathbf{x} \Rightarrow (L + D)\mathbf{x}^{(k+1)} = \mathbf{b} - U\mathbf{x}^{(k)},$$

and hence the iteration, with $M = L + D$ reads

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

Back to our example, the Gauss Seidel iteration will read:

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(7 + x_2^{(k)} - x_3^{(k)}) \\ \frac{1}{8}(-21 - 4x_1^{(k+1)} - x_3^{(k)}) \\ \frac{1}{5}(15 + 2x_1^{(k+1)} - x_2^{(k+1)}) \end{bmatrix}$$

This time, the convergence is much faster than the Jacobi method:

```
Iter:  0:  [1.0, 2.0, 2.0]
Iter:  1:  [1.75, 3.75, 2.95]
Iter:  2:  [1.95, 3.96875, 2.98625]
Iter:  3:  [1.99562, 3.99609, 2.99903]
Iter:  4:  [1.99927, 3.99951, 2.9998]
Iter:  5:  [1.99993, 3.99994, 2.99998]
Iter:  6:  [1.99999, 3.99999, 3.0]
Iter:  7:  [2.0, 4.0, 3.0]
```

To state a convergence theorem, recall the definition of a strictly diagonally dominant matrix:

Definition 8 (**Strictly diagonally dominant matrices (SDD)**). *A matrix A is strictly diagonally dominant in rows if for every row i*

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

.

We have the next theorem regarding the convergence of Jacobi and Gauss Seidel

Theorem 17. *If the matrix A is strictly diagonally dominant in rows, then both Jacobi and Gauss Seidel methods converge.*

Proof. We will not show the proof but it can be achieved by looking at $\|I - M^{-1}A\|_\infty$, and using the fact that any induced matrix norm upper-bounds the spectral radius. □

Another important property for Gauss-Seidel, that we will not prove here:

Theorem 18. *If the matrix A is positive definite, then Gauss-Seidel converges.*

Remark 12 (Weighted Jacobi and Gauss-Seidel)**.** *Sometimes it is helpful to use a weighted Jacobi iteration, which is given by*

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

where usually $0 < \omega \leq 1$ (ω is also called a damping parameter). If $\omega = 1$, then we get the standard Jacobi method. In some cases, only the damped version of Jacobi will converge and the standard Jacobi won't. Similarly to the Jacobi method, the GS method can be used with a damping parameter. However, for GS one can use a parameter $\omega > 1$ which leads to the SOR method.

Remark 13 (The ordering of GS)**.** *In the same way we chose $M = L + D$, we could choose $M = U + D$. The first corresponds to iterating the entries from 1 to n, while the second corresponds to iterating the entries n to 1. In principle, we can even choose a random ordering, which would correspond to a method called randomized Gauss-Seidel.*

6.1.4 More on the convergence rate of simple iterative methods

Assume a symmetric positive definite matrix A. We now examine the convergence rate of a very common method

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha(\mathbf{b} - A\mathbf{x}^{(k)}),$$

where we are free to choose the scalar $\alpha > 0$ to get the best convergence rate. This method is a special case of the "Steepest Descent" method.

We've seen that the equation for the error is given by $\mathbf{e}^{(k+1)} = (I - \alpha A)\mathbf{e}^{(k)}$. It can be shown that for any vector norm we have:

$$\lim_{k\to\infty} \frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} = \rho(I - \alpha A).$$

We now search for the minimal possible value of the convergence factor $\rho(I - \alpha A)$ with respect to α.

$$\rho(I - \alpha A) = \max\{|1 - \alpha\lambda_{max}|, |1 - \alpha\lambda_{min}|\},$$

The two values in the max term are the two extreme eigenvalues. This will be minimal if those two are equal.

$$\alpha_{opt}\lambda_{max} - 1 = 1 - \alpha_{opt}\lambda_{min} \Rightarrow \alpha_{opt} = \frac{2}{\lambda_{max} + \lambda_{min}}.$$

The optimal convergence factor is

$$\rho_{min} = 1 - \alpha_{opt}\lambda_{min} = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \frac{\kappa(A) - 1}{\kappa(A) + 1} = 1 - \frac{2}{\kappa(A) + 1}.$$

Recall that $\kappa_2(A) = \text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}}$ is the condition number of A with respect to the ℓ_2 norm (since the matrix is SPD, then $\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}} = \frac{\lambda_{max}}{\lambda_{min}}$). The property above shows that if the condition number is high, the convergence factor will get close to 1, and it will take a lot of iterations for Steepest Descent to reduce the error (and that's for the optimal value of α in that respect).

6.1.5 Other variants of stationary methods, weighting.

The Richardson method The Richardson iteration is the simplest kind, in which $M = cI$, where $c > 0$. The iteration is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \frac{1}{c}(\mathbf{b} - A\mathbf{x}^{(k)}).$$

The (weighted) Jacobi method The next simplest iteration is the Jacobi method, in which $M_J = D = \text{diag}(A)$—a diagonal matrix whose entries are the diagonal entries of A ($D_{ii} = A_{ii}$). The weighted iteration is

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

where usually $0 < \omega \leq 1$ (ω is also called a damping parameter). If $\omega = 1$, then we get the standard Jacobi method.

Another way to look at the Jacobi method is by the following splitting: $A = D + L + U$, where D is the diagonal part of A, L is the lower triangular part of A and U is the upper

triangular part. Then, the weighted Jacobi method is

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega D^{-1}(\mathbf{b} - (L+U)\mathbf{x}^{(k)}),$$

and is equivalent to the previous writing. Now, however, the iteration does not include the standard matrix-vector multiplication $A\mathbf{x}$, but has a different matrix-vector product. A third way to write the Jacobi method is the scalar one:

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

The Successive Over Relaxation (SOR) method The SOR method is a weighted version of GS (just like Jacobi and its weighted version), only here, we may choose a parameter $\omega > 1$. We start with the scalar form:

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

In vector form we get

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

or, more compactly

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{k} + \omega(D+\omega L)^{-1}\left(\mathbf{b} - A\mathbf{x}^{(k)}\right),$$

Theorem 19. *If the matrix A is positive definite, then SOR converges for $0 < \omega < 2$.*

6.1.6 The variational meaning of Gauss-Seidel

Assume that A is positive definite. Then we can show that the updates of Gauss-Seidel for each x_i are equivalent to **minimizing** the following function

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|_A^2 = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b} + \frac{1}{2}(\mathbf{x}^*)^\top\mathbf{b},$$

with respect to x_i. This function is equivalent to solving a linear system $A\mathbf{x} = b$, and has a minimum of 0, which is obtained at $\mathbf{x}^* = A^{-1}\mathbf{b}$. To show this, consider the necessary

condition for a minimum point: $\nabla f = 0$. In our case it is simply the linear system $A\mathbf{x} = \mathbf{b}$. Note that the term $\frac{1}{2}(\mathbf{x}^*)^\top \mathbf{b}$ is just a constant scalar that does not play part in the minimization.

As noted before, in the Gauss-Seidel method we iterate over all entries i of $\mathbf{x}$, and change each scalar entry x_i so that the i-th equation is satisfied (or $r_i = 0$ for the residual vector $\mathbf{r}$) given the other entries in the current approximation $\mathbf{x}$. By fulfilling the i-th equation in $A\mathbf{x} = \mathbf{b}$, we essentially zeroing the i-th entry of the gradient, which means we set $\frac{\partial f}{\partial x_i} = 0$. It means that we also minimize $f()$ with respect to x_i at each update of x_i. This is the variational property of Gauss-Seidel. From this we can learn that the value of f is monotonically decreasing with each update and since f is bounded from below, the series $\{f(\mathbf{x})\}$ converges. It can be shown that if the matrix A is non-singular, the Gauss-Seidel method converges.

Theorem 20. *If the matrix A is positive definite, then Gauss Seidel converges, and the values $f(\mathbf{x}^{(k)})$ of the function f above are monotonically decreasing with the GS updates.*

Example 22 (Variational property of Gauss Seidel)**.** *Consider the following linear system:*

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

It is easy to show that

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top \mathbf{b} && (39) \\ &= x_1^2 + x_1x_2 + 1.5x_2^2 - 3x_1 - 4x_2. && (40) \end{aligned}$$

The condition $\nabla f = 0$ in this case is

$$\frac{\partial f}{\partial x_1} = 2x_1 + x_2 - 3 = 0 \quad (41)$$

$$\frac{\partial f}{\partial x_2} = x_1 + 3x_2 - 4 = 0 \quad (42)$$

$$(43)$$

which is exactly the linear system $A\mathbf{x} = \mathbf{b}$. *We now show that zeroing each of the equations* (41)-(41) *alternately leads to a monotonically decreasing series* f, *and that this proces is equivalent to a general Gauss-Seidel routine. The code in this example plots the graph in Fig. 7.*

```
############## File: GS_Min_Equivalence.jl:
############## Equivalence of GS and Alternating Minimization
# importing black-box Gauss-Seidel code:
import OptimizationMethodsCourseMaterial.NLA.GS
f = (A,x,b)-> 0.5*dot(x,A*x) - dot(x,b);
A = [2.0 1.0 ; 1.0 3.0]; b = [3.0 ; 4.0];
n_iter = 10;epsilon = 1e-12;
(y,nr,Fs) = GS(A,b,zeros(2),epsilon,n_iter);
x = zeros(2);
Fs2 = zeros(2*n_iter + 1); Fs2[1] = 0; nr2 = zeros(n_iter + 1); nr2[1] = norm(b)
for k=1:n_iter
        x[1] = 0.5*(3 - x[2]);
        Fs2[2*k] = f(A,x,b);
        x[2] = (4.0 - x[1])/3.0;
        Fs2[2*k + 1] = f(A,x,b);
        nr2[k+1] = norm(A*x - b);
end
f_min = -3.5;

using PyPlot;
include("GS_Min_Equivalence.jl");
figure()
subplot(1,2,1)
semilogy(0:n_iter, nr, "or");
semilogy(0:n_iter, nr2,"-b");
legend(("Gauss Seidel","Alternating minimization"));
title("Residual norm history");
ylabel("Residual Norm");
xlabel("Iterations");
subplot(1,2,2)
println(Fs)
println(Fs2)

semilogy(0:2:2*n_iter, Fs.-f_min,"or");
semilogy(0:2*n_iter, Fs2.-f_min,"-b");
legend(("Gauss Seidel","Alternating minimization"));
title("Function value minimization history");
ylabel("F(x)-Fmin");
xlabel("xi Updates")
```

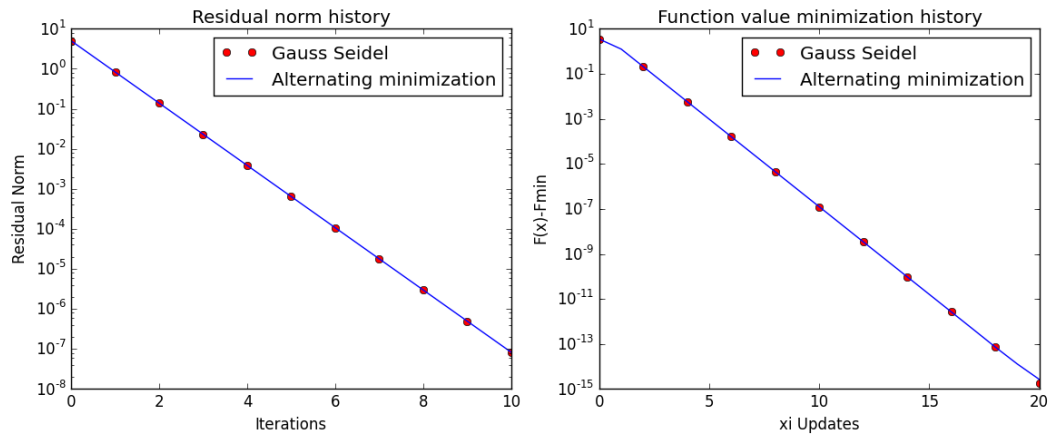

Figure 7: The equivalency of Gauss Seidel and alternating minimization.

6.1.7 One-point non-stationary iterative methods (Steepest Descent)

The methods presented earlier are called "stationary". It means that the iteration function ϕ that defines the method $\mathbf{x}^{(k+1)} = \phi(\mathbf{x}^{(k)})$ do not change with k. To illustrate the meaning of "non-stationary" methods, consider, for example, the weighted Jacobi or Richardson iterations mentioned earlier. These methods need a parameter (c or ω) to be chosen in some optimal way. The next example illustrate an approach that automatically chooses the parameter in each iteration.

Steepest Descent for symmetric positive definite matrices Assume again that $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite and consider the minimization of the following quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}^* - \mathbf{x}\|_A^2 = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b} + \frac{1}{2}(\mathbf{x}^*)^\top\mathbf{b}. \tag{44}$$

Recall that this function has a minimum of 0, obtained at $\mathbf{x} = \mathbf{x}^* = A^{-1}\mathbf{b}$, and the condition $\nabla f = 0$ is the linear system $A\mathbf{x} = \mathbf{b}$. The steepest descent method is the most basic optimization method for minimizing a general function f and is defined by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha\nabla f(\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} + \alpha(\mathbf{b} - A\mathbf{x}^{(k)}),$$

where $\alpha > 0$ is a parameter. The first equality is for a general f, and the second equality is for Eq. (44). This method is exactly the Richardson method with α instead of a constant

$\frac{1}{c}$. The question is: can we choose α is some optimal way? Well, the optimal α is the value such that $\rho(I - \alpha A)$ is minimal, but this is very hard to choose. As an alternative, we can choose α to be optimal for *each iteration* in a greedy sense so we minimize (44). We define the following scalar function $g(\alpha)$:

$$\begin{aligned} g(\alpha) &\triangleq f(\mathbf{x}^{(k)} + \alpha \mathbf{r}^{(k)}) = \tfrac{1}{2}\|\mathbf{x}^* - \mathbf{x}^{(k)} - \alpha \mathbf{r}^{(k)}\|_A^2 = \tfrac{1}{2}\|\mathbf{e}^{(k)} - \alpha \mathbf{r}^{(k)}\|_A^2 \\ &= \tfrac{1}{2}((\mathbf{e}^{(k)})^\top A \mathbf{e}^{(k)}) - \alpha (\mathbf{r}^{(k)})^\top A \mathbf{e}^{(k)} + \tfrac{1}{2}\alpha^2((\mathbf{r}^{(k)})^\top A \mathbf{r}^{(k)}) \end{aligned} \tag{45}$$

And the minimization of g with respect to α is done by:

$$\begin{aligned} g'(\alpha) &= -(\mathbf{r}^{(k)})^\top A \mathbf{e}^{(k)} + \alpha((\mathbf{r}^{(k)})^\top A \mathbf{r}^{(k)}) = 0 \\ &\Rightarrow \alpha_{opt} = \tfrac{(\mathbf{r}^{(k)})^\top A \mathbf{e}^{(k)}}{(\mathbf{r}^{(k)})^\top A \mathbf{r}^{(k)}} = \tfrac{\langle \mathbf{r}^{(k)}, \mathbf{r}^k \rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^k \rangle}. \end{aligned} \tag{46}$$

Which leads to the choice of α_{opt}. This choice of α is a good choice although it is not really optimal for the whole convergence process.

Algorithm: The steepest descent method (for linear systems)
 # Input: $A \in \mathbb{R}^{n\times n}$ *SPD,* $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}^{(0)} \in \mathbb{R}^n$,
maxIter, ϵ*, Convergence criterion*
 # Output: $\mathbf{x}$ *s.t* $A\mathbf{x} \approx \mathbf{b}$
Define the first residual $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
for $k = 1, ..., maxIter$ **do**
  Define a weight: $\alpha = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle}{\langle \mathbf{r}^{(k-1)}, A\mathbf{r}^{(k-1)} \rangle}$
  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha \mathbf{r}^{(k-1)}$,
  $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} = \mathbf{r}^{(k-1)} - \alpha A\mathbf{r}^{(k-1)}$ # $A\mathbf{r}^{(k-1)}$ *is already computed for calculating* α.
  If convergence is reached, break
end
Return $\mathbf{x}^{(k)}$ as the solution.

Algorithm 8: The steepest descent method for linear systems.

We now examine the process. Consider the iteration with α_{opt} using the inner products notation:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^k \rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)} \rangle} \mathbf{r}^{(k)}$$

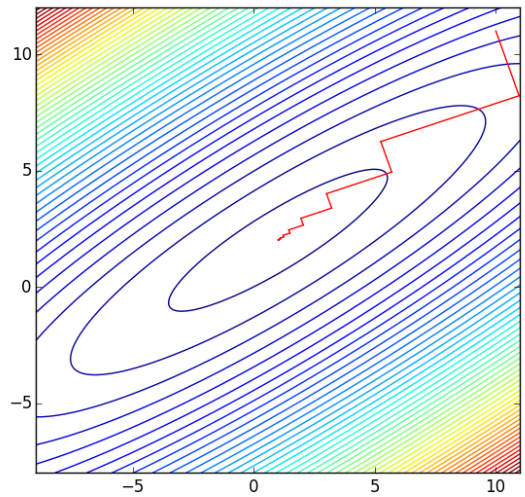

Figure 8: The zigzagging effect of the SD iterations.

Taking the minus of this equation and adding $\mathbf{x}^*$ we get an equation for the error

$$\mathbf{e}^{(k+1)} = \mathbf{e}^{(k)} - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^k \rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)} \rangle} \mathbf{r}^{(k)}. \tag{47}$$

Notice that such equations appear in the Gram Schmidt orthogonalization process, and here, we're making $\mathbf{e}^{k+1}$ A-orthogonal to the direction $\mathbf{r}^{(k)}$. Although this looks appealing, it has a somewhat unwanted property: it also means that the residuals are orthogonal $\langle \mathbf{r}^{(k+1)}, \mathbf{r}^{(k)} \rangle = 0$, and since $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = \alpha \mathbf{r}^{(k)}$, it also means that the directions are orthogonal to each other

$$\langle \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}, \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \rangle = 0,$$

or that the algorithm is zigzagging as shown in the Fig. 8. The code is for the plot is below.

```
using PyPlot;
A=randn(2,2); A = A'*[1.0 0.0; 0.0 0.1]*A; xs=[1.0; 2.0]; b = A*xs;
m = length(-10:0.1:10);
X = repmat((-10:0.1:10) + xs[1],1,m)';
Y = repmat((-10:0.1:10) + xs[2],1,m);
F = 0.5*(A[1,1]*X.^2+2*A[1,2]*X.*Y+A[2,2]*Y.^2)-b[1]*X-b[2]*Y;
figure(); contour(X,Y,F,50); #hold on; axis image;
x = xs + [9;9];
for k=1:20
    r = b-A*x;
    alpha=dot(r,r)/dot(r,A*r);
    x_prev = copy(x);
    x=x+alpha*r;
    plot([x[1];x_prev[1]],[x[2];x_prev[2]],"r");
end;
```

6.2 Krylov subspace methods

Let us examine two steps of the Steepest descent (SD) method:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}, \quad \text{and} \quad \mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha^{(k-1)}\mathbf{r}^{(k-1)}.$$

Putting the two together leads to

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k-1)} + \alpha^{(k)}\mathbf{r}^{(k)} + \alpha^{(k-1)}\mathbf{r}^{(k-1)}.$$

This way recursively we can show that

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(0)} + \sum_{i=0}^{k} \alpha^{(i)}\mathbf{r}^{(i)}.$$

This means that

$$\mathbf{e}^{(k+1)} \in \mathbf{e}^{(0)} + \text{span}\{\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, ..., \mathbf{r}^{(k)}\} = \mathbf{e}^{(0)} + \text{span}\{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, ..., A^k\mathbf{r}^{(0)}\}. \tag{48}$$

SD chooses the coefficients $\alpha^{(k)}$ in a certain way, but can we do better with this span? It turns out we can!

We will now generalized the SD method to be of the form $\mathbf{x}^{(k+1)} = \phi(\mathbf{x}^{(k)}, \mathbf{x}^{(k-1)}, ..., \mathbf{x}^{(0)})$, instead of being a one-point method that depends only on $\mathbf{x}^{(k)}$. The new family of methods

are defined by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \sum_{i=0}^{k} \alpha_i^{(k)} \mathbf{r}^{(i)}, \tag{49}$$

and this time all the k+1 coefficients $\{\alpha_i^{(k)}\}_{i=0}^k$ are chosen in some optimal way for each step. Note that (48) holds for this as well, and in other words, $\mathbf{e}^{(k+1)}$ is chosen to be out of the span in (48) such that it is optimal in some way. This will introduce a family of very popular solvers called **Krylov Methods**, and the subspace defined in (48) is called a Krylov subspace. By choosing some coefficients for each step, (48) is the same as:

$$\mathbf{e}^{(k+1)} = \mathbf{e}^{(0)} + p_k(A)A\mathbf{e}^{(0)} = (I - p_k(A)A)\mathbf{e}^{(0)},$$

where p_k is some polynomial of degree k that we need to choose in some optimal way.

6.2.1 The Conjugate Gradients (CG) Method

The discussion above fits any matrix A, but for now we keep discussing the case of a symmetric positive definite A. Let us now examine a particular way for choosing $\alpha_i^{(k)}$, such that, similarly to SD, at step k, the functional (44) is minimized over span$\{\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, ..., \mathbf{r}^{(k)}\}$:

$$\{\alpha_i^{(k)}\}_{i=0}^k = \underset{\{\alpha_i\}_{i=0}^k}{\arg\min}\{f(\mathbf{x}^{(k+1)})\} = \underset{\{\alpha_i\}_{i=0}^k}{\arg\min}\left\{ f\left(\mathbf{x}^{(k)} + \sum_{i=0}^{k} \alpha_i \mathbf{r}^{(i)}\right)\right\}. \tag{50}$$

This is similar to what we did in (45)-(46) for a single α in SD. While it is possible to follow the derivation of (45)-(46) for this case as well, we will do that in another way.

We saw earlier that in SD, we get Eq. (47) for the error, where similarly to the Gram Schmidt orthogonalization process, we're making $\mathbf{e}^{k+1}$ A-orthogonal to the direction $\mathbf{r}^{(k)}$. Let us assume that we can build a set of vectors $\{\mathbf{p}^{(i)}\}_{i=0}^k$, such that

$$\text{span}\left(\{\mathbf{p}^{(i)}\}_{i=0}^k\right) = \text{span}\left(\{\mathbf{r}^{(i)}\}_{i=0}^k\right),$$

and such that $\langle \mathbf{p}^{(j)}, A\mathbf{p}^{(i)}\rangle = \delta_{ij}$, i.e., the vectors $\mathbf{p}^{(j)}$ are A-orthogonal. This can be achieved by a Gram Schmidt process, for example.

The naive CG method Now we will see the first two steps of CG:

1. (SD step as initialization) Define $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$. Calc: $\alpha_0^{(0)} = \frac{\langle \mathbf{e}^{(0)}, A\mathbf{p}^0 \rangle}{\langle \mathbf{p}^{(0)}, A\mathbf{p}^{(0)} \rangle}$.
   Now $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0^{(0)}\mathbf{p}^{(0)}$. Note that $\langle \mathbf{e}^{(1)}, A\mathbf{p}^{(0)} \rangle = 0$.
   Calc: $\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha_0^{(0)} A\mathbf{p}^{(0)}$.

2. (CG 2nd step) Define an orthogonal direction by a GS step: $\mathbf{p}^{(1)} = \mathbf{r}^{(1)} - \frac{\langle \mathbf{r}^{(1)}, A\mathbf{p}^{(0)} \rangle}{\langle \mathbf{p}^{(0)}, A\mathbf{p}^{(0)} \rangle}\mathbf{p}^{(0)}$.
   Perform a minimization: $\alpha_0^{(1)}, \alpha_1^{(1)} = \arg\min_{\alpha_0,\alpha_1} g(\alpha_0, \alpha_1)$,
   where $g(\alpha_0, \alpha_1) = f(\mathbf{x}^{(1)} + \alpha_0\mathbf{p}^{(0)} + \alpha_1\mathbf{p}^{(1)}) = \frac{1}{2}\|\mathbf{e}^{(1)} - \alpha_0\mathbf{p}^{(0)} - \alpha_1\mathbf{p}^{(1)}\|_A^2$.
   This is again a quadratic minimization that leads (given the orthogonality of the $\mathbf{p}_i$s) to
   $$\alpha_1^{(1)} = \frac{\langle \mathbf{e}^{(1)}, A\mathbf{p}^{(1)} \rangle}{\langle \mathbf{p}^{(1)}, A\mathbf{p}^{(1)} \rangle}$$
   and surprisingly $\alpha_0^{(1)} = \frac{\langle \mathbf{e}^{(1)}, A\mathbf{p}^{(0)} \rangle}{\langle \mathbf{p}^{(0)}, A\mathbf{p}^{(0)} \rangle} = 0$, because of Step 1. Note that on this step, we made $\mathbf{e}^{(2)}$ A-orthogonal to both $\mathbf{p}^{(1)}$ and $\mathbf{p}^{(0)}$.
   Compute: $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_1^{(1)}\mathbf{p}^{(1)}$, $\mathbf{r}^{(2)} = \mathbf{r}^{(1)} - \alpha_1^{(1)} A\mathbf{p}^{(1)}$.

This means that at iteration k we should not minimize f over previous directions because of the orthogonalization. It turns out that the k-th iteration of CG can be performed using one variable $\alpha^{(k)}$, because the rest are just zero. Assume that we have all the previous directions $\mathbf{p}^0, ..., \mathbf{p}^{k-1}$ A-orthogonal, and assume that we have $\mathbf{x}^{(k)}$, and $\mathbf{r}^{(k)}$.

- First make $\mathbf{p}^{(k)}$ orthogonal to all previous directions by GS:
$$\mathbf{p}^{(k)} = \mathbf{r}^{(k)} - \sum_{i=0}^{k-1} \frac{\langle \mathbf{r}^{(k)}, A\mathbf{p}^{(i)} \rangle}{\langle \mathbf{p}^{(i)}, A\mathbf{p}^{(i)} \rangle}\mathbf{p}^{(i)}. \tag{51}$$
- Choose $\alpha^{(k)}$ such that $f(\mathbf{x}^{k+1})$ is minimized: $\alpha^{(k)} = \frac{\langle \mathbf{e}^{(k)}, A\mathbf{p}^{(k)} \rangle}{\langle \mathbf{p}^{(k)}, A\mathbf{p}^{(k)} \rangle}$, or equivalently such that $\mathbf{e}^{(k+1)}$ is A-orthogonal to $\mathbf{p}^{(k)}$ (in addition to $\mathbf{p}^{(i)}$, for $i = k-1, ..., 0$).
- Compute: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$, $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)} A\mathbf{p}^{(k)}$.

The true CG method In the naive CG method we saw a nice process, which performs quite a lot of iterations in Eq. (51). For a large k, this may be very expensive. In fact, this is not so different than minimizing (50) directly (it is about the same cost for $k << n$).

We will see now the true advantage of CG, by showing that (51) can be significantly reduced. For $i < k-1$ we have that in (51)

$$\langle \mathbf{r}^{(k)}, A\mathbf{p}^{(i)} \rangle = \langle \mathbf{r}^{(k)}, \frac{1}{\alpha^{(i)}}(\mathbf{r}^{(i+1)} - \mathbf{r}^{(i)}) \rangle = \langle \frac{1}{\alpha^{(i)}}(\mathbf{r}^{(i+1)} - \mathbf{r}^{(i)}), A\mathbf{e}^{(k)} \rangle = 0,$$

because at the conclusion of each CG step we have that $\mathbf{e}^{(k)}$ is A-orthogonal to all the $\mathbf{p}_i$'s and the $\mathbf{r}_i$'s for $i < k$. This means that Eq. (51) has only one term in the sum, that we will denote as β. Algorithm

Algorithm: The Conjugate Gradient method
 # Input: $A \in \mathbb{R}^{n\times n}$ *SPD,* $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}^{(0)} \in \mathbb{R}^n$,
maxIter, ϵ, *Convergence criterion*
 # Output: $\mathbf{x}$ *s.t* $A\mathbf{x} \approx \mathbf{b}$
Define the first residual $\mathbf{p}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
for $k = 1, ..., maxIter$ **do**
  Define a weight: $\alpha = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{p}^{(k-1)} \rangle}{\langle \mathbf{p}^{(k-1)}, A\mathbf{p}^{(k-1)} \rangle} = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle}{\langle \mathbf{p}^{(k-1)}, A\mathbf{p}^{(k-1)} \rangle}$
  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha\mathbf{p}^{(k-1)}$,
  $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} = \mathbf{r}^{(k-1)} - \alpha A\mathbf{p}^{(k-1)}$
  If convergence is reached, break
  $\beta = -\frac{\langle \mathbf{r}^{(k)}, A\mathbf{p}^{(k-1)} \rangle}{\langle \mathbf{p}^{(k-1)}, A\mathbf{p}^{(k-1)} \rangle} = \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle}$
  $\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta\mathbf{p}^{(k-1)}$
end
Return $\mathbf{x}^{(k)}$ as the solution.

Algorithm 9: The Conjugate Gradients method.

Theorem 21. *If $A \in \mathbb{R}^{n\times n}$, SPD and full rank, then the CG method converges to the solution of $A\mathbf{x} = \mathbf{b}$, at at most n iterations for any initial guess.*

Proof. The proof follows immediately from (50): at the n-th iteration we will be forming a minimization over the full space $\mathbb{R}^n$. □

Theorem 22. *If $A \in \mathbb{R}^{n\times n}$, SPD and full rank, then the CG method produces iterates $\{\mathbf{x}^{(k)}\}$ which satisfy*

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\|_A \leq 2\|\mathbf{x}^* - \mathbf{x}^{(0)}\|_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k$$

Proof. See Luenberger (1973,p.187). □

While the steepest descent method converges according to the condition number of A, the CG method converges according to the square root of this condition number.

6.2.2 Other Krylov Methods

The Conjugate Gradient method fits only for symmetric positive definite systems. For other cases, other Krylov methods should be used. We will not cover this in detail in this course. For a symmetric but indefinite systems, the MINRES method should be used. For a non-symmetric case, the GCR or GMRES methods should be used.

6.3 Preconditioning

Krylov subspace methods are effective in solving linear systems as the Krylov subspace gets larger. However, for highly ill-conditioned problems, also the Krylov methods require many iterations and struggle with conditioning in their inner computations. To help Krylov methods, we may apply "preconditioning". Suppose we need to solve the system

$$A\mathbf{x} = \mathbf{b}.$$

Given a matrix M, s.t $M^{-1}A \approx I$ in some sense, we can solve the system

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b},$$

and get the same solution $\mathbf{x}$. That is, instead of having $\mathbf{b}$, we'll just have $M^{-1}\mathbf{b}$ as a right-hand-side, and every time we usually multiply A with a vector, we'll also multiply the answer by M^{-1}. Now, the condition number for the solver of choice will be that of $M^{-1}A$, instead of A. The procedure is called "left preconditioning". A similar variant is the right preconditioning:

$$AM^{-1}M\mathbf{x} = \mathbf{b} \Rightarrow AM^{-1}\mathbf{y} = \mathbf{b} \Rightarrow \mathbf{x} = M^{-1}\mathbf{y},$$

where here, we will use the original right hand side $\mathbf{b}$, solve the system $AM^{-1}\mathbf{y} = \mathbf{b}$, and at the end compute $\mathbf{x} = M^{-1}\mathbf{y}$ to get the answer to the original system. The two preconditioning forms may be combined (i.e, left and right).

We've seen, however, that the CG method is suitable only for SPD matrices, which are in particular symmetric. Since either of the preconditioned matrices above is symmetric even if M is symmetric, this will break the orthogonality of Conjugate Gradients. For this reason, there's also symmetric preconditioning, which takes a convenient in the CG method. The symmetric preconditioning will read:

$$L^T A L L^{-1}\mathbf{x} = L^T\mathbf{b}$$

where $M^{-1} = LL^T$, is a symmetric decomposition of the (assumed to be symmetric) matrix M^{-1}. It may be for example the Cholesky factorization of M^{-1}.

Now let us assume that we solve the system

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$$

with the standard CG algorithm, where $\hat{A} = L^T A L$, $\hat{\mathbf{x}} = L^{-1}\mathbf{x}$, and $\hat{\mathbf{b}} = L^T\mathbf{b}$. We will go over all the ingredients used in Alg. 9 and see how they change. For starts, the residual

$$\hat{\mathbf{r}}^{(k)} = \hat{\mathbf{b}} - \hat{A}\hat{\mathbf{x}^{(k)}} = L^T\mathbf{b} - L^T A L L^{-1}\mathbf{x}^{(k)} = L^T\mathbf{b} - L^T A\mathbf{x}^{(k)} = L^T\mathbf{r}^{(k)}.$$

Let us also define the following quantities:

$$\hat{\mathbf{p}}^{(k)} = L^{-1}\mathbf{p}^{(k)}, \quad \mathbf{z}^{(k)} = M^{-1}\mathbf{r}^{(k)}.$$

Given an initial guess $\hat{\mathbf{x}}^{(0)} = L^{-1}\mathbf{x}^{(0)}$ (which are often both 0), and residuals $\hat{\mathbf{r}}^{(0)} = L^T\mathbf{r}^{(0)}$ the CG algorithm first applies:

$$\hat{\mathbf{p}}^{(0)} = \hat{\mathbf{r}}^{(0)} \Rightarrow L^{-1}\mathbf{p}^{(0)} = L^T\mathbf{r}^{(0)} \Rightarrow \mathbf{p}^{(0)} = M^{-1}\mathbf{r}^{(0)},$$

where we used the decomposition of M^{-1}.

Below $\mathbf{z}^{(k)} = \tilde{r}_k$

The step length $\hat{\alpha}$ transforms as follows:

$$\begin{aligned}
\hat{\alpha}_n &= \left(\hat{\boldsymbol{r}}_{n-1}^T \hat{\boldsymbol{r}}_{n-1}\right) / \left(\hat{\boldsymbol{p}}_{n-1}^T \hat{A} \hat{\boldsymbol{p}}_{n-1}\right) \\
&= \left(\left(L^T \boldsymbol{r}_{n-1}\right)^T L^T \boldsymbol{r}_{n-1}\right) / \left(\left(L^{-1} \boldsymbol{p}_{n-1}\right)^T \left(L^T A L\right) \left(L^{-1} \boldsymbol{p}_{n-1}\right)\right) \\
&= \left(\boldsymbol{r}_{n-1}^T L L^T \boldsymbol{r}_{n-1}\right) / \left(\boldsymbol{p}_{n-1}^T L^{-T} L^T A L L^{-1} \boldsymbol{p}_{n-1}\right) \\
&= \left(\boldsymbol{r}_{n-1}^T \underbrace{L L^T}_{=M^{-1}} \boldsymbol{r}_{n-1}\right) / \left(\boldsymbol{p}_{n-1}^T A \boldsymbol{p}_{n-1}\right) \\
&= \left(\boldsymbol{r}_{n-1}^T \tilde{\boldsymbol{r}}_{n-1}\right) / \left(\boldsymbol{p}_{n-1}^T A \boldsymbol{p}_{n-1}\right).
\end{aligned}$$

The approximate solution is updated according to

$$\begin{aligned}
\hat{\boldsymbol{x}}_n = \hat{\boldsymbol{x}}_{n-1} + \hat{\alpha}_n \hat{\boldsymbol{p}}_{n-1} \quad &\Longleftrightarrow \quad L^{-1} \boldsymbol{x}_n = L^{-1} \boldsymbol{x}_{n-1} + \hat{\alpha}_n L^{-1} \boldsymbol{p}_{n-1} \\
&\Longleftrightarrow \quad \boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \hat{\alpha}_n \boldsymbol{p}_{n-1}.
\end{aligned}$$

The residuals are updated as

$$\begin{aligned}
\hat{\boldsymbol{r}}_n = \hat{\boldsymbol{r}}_{n-1} - \hat{\alpha}_n \hat{A} \hat{\boldsymbol{p}}_{n-1} \quad &\Longleftrightarrow \quad L^T \boldsymbol{r}_n = L^T \boldsymbol{r}_{n-1} - \hat{\alpha}_n (L^T A L) L^{-1} \boldsymbol{p}_{n-1} \\
&\Longleftrightarrow \quad \boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \hat{\alpha}_n A \boldsymbol{p}_{n-1}.
\end{aligned}$$

The gradient correction factor β transforms as follows:

$$\begin{aligned}
\hat{\beta}_n &= \left(\hat{\boldsymbol{r}}_n^T \hat{\boldsymbol{r}}_n\right) / \left(\hat{\boldsymbol{r}}_{n-1}^T \hat{\boldsymbol{r}}_{n-1}\right) \\
&= \left(\left(L^T \boldsymbol{r}_n\right)^T \left(L^T \boldsymbol{r}_n\right)\right) / \left(\left(L^T \boldsymbol{r}_{n-1}\right)^T \left(L^T \boldsymbol{r}_{n-1}\right)\right) \\
&= \left(\boldsymbol{r}_n^T \underbrace{L L^T}_{=M^{-1}} \boldsymbol{r}_n\right) / \left(\boldsymbol{r}_{n-1}^T \underbrace{L L^T}_{=M^{-1}} \boldsymbol{r}_{n-1}\right) \\
&= \left(\boldsymbol{r}_n^T \tilde{\boldsymbol{r}}_n\right) / \left(\boldsymbol{r}_{n-1}^T \tilde{\boldsymbol{r}}_{n-1}\right).
\end{aligned}$$

Finally, for the new search direction we have

$$\begin{aligned}
\hat{\boldsymbol{p}}_n = \hat{\boldsymbol{r}}_n + \hat{\beta}_n \hat{\boldsymbol{p}}_{n-1} \quad &\Longleftrightarrow \quad L^{-1} \boldsymbol{p}_n = L^T \boldsymbol{r}_n + \hat{\beta}_n L^{-1} \boldsymbol{p}_{n-1} \\
&\Longleftrightarrow \quad \boldsymbol{p}_n = M^{-1} \boldsymbol{r}_n + \hat{\beta}_n \boldsymbol{p}_{n-1} \\
&\Longleftrightarrow \quad \boldsymbol{p}_n = \tilde{\boldsymbol{r}}_n + \hat{\beta}_n \boldsymbol{p}_{n-1},
\end{aligned}$$

where we have multiplied by L and used the definition of M in the penultimate step.

6.3.1 GMRES: a Krylov method for non-Hermitian systems.

In this section we will learn about the Generalized Minimal Residual (GMRES) method, which is a quite simple yet very popular and effective Krylov method, which, unlike PCG, is suitable for non-symmetric problems.

Algorithm: PCG – Preconditioned Conjugate Gradients
 # Input: $A \in \mathbb{R}^{n\times n}$ *SPD, An SPD preconditioner* M^{-1}, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}^{(0)} \in \mathbb{R}^n$, *maxIter*, ϵ, *Convergence criterion*
 # Output: $\mathbf{x}$ *s.t* $A\mathbf{x} \approx \mathbf{b}$
Define the first residual and directions:
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$, $\mathbf{z}^{(0)} = M^{-1}\mathbf{r}^{(0)}$, $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$
for $k = 1, ..., maxIter$ **do**
  Define a weight: $\alpha = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{z}^{(k-1)} \rangle}{\langle \mathbf{p}^{(k-1)}, A\mathbf{p}^{(k-1)} \rangle}$
  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha \mathbf{p}^{(k-1)}$,
  $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} = \mathbf{r}^{(k-1)} - \alpha A\mathbf{p}^{(k-1)}$
  If convergence is reached, break
  $\mathbf{z}^{(k)} = M^{-1}\mathbf{r}^{(k)}$
  $\beta = \frac{\langle \mathbf{r}^{(k)}, \mathbf{z}^{(k)} \rangle}{\langle \mathbf{r}^{(k-1)}, \mathbf{z}^{(k-1)} \rangle}$
  $\mathbf{p}^{(k)} = \mathbf{z}^{(k)} + \beta \mathbf{p}^{(k-1)}$
end
Return $\mathbf{x}^{(k)}$ as the solution.

Algorithm 10: The Preconditioned Conjugate Gradient (PCG) method.

The vanilla GMRES To start the process, we will just apply the algorithm as it is, although this will result in a rather unstable algorithm. Let us now examine another way for choosing $\alpha_i^{(k)}$, such that at step k, we simply minimize the residual over span$\{\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, ..., \mathbf{r}^{(k)}\}$:

$$\{\alpha_i^{(k)}\}_{i=0}^k = \underset{\{\alpha_i\}_{i=0}^k}{\arg\min}\{\|(A\mathbf{x}^{(k+1)} - \mathbf{b})\|_2^2\} \tag{52}$$

$$= \underset{\{\alpha_i\}_{i=0}^k}{\arg\min}\left\|A\left(\mathbf{x}^{(0)} + \sum_{i=0}^k \alpha_i \mathbf{r}^{(i)}\right) - \mathbf{b}\right\|_2^2 \tag{53}$$

$$= \underset{\alpha\in\mathbb{R}^{k+1}}{\arg\min}\left\|AR^{(k)}\alpha - \mathbf{r}^{(0)}\right\|_2^2, \tag{54}$$

where $R^{(k)} = [\mathbf{r}^{(0)}|\mathbf{r}^{(0)}|\cdots|\mathbf{r}^{(k)}]$ is the matrix of all residuals as columns. The solution to this system is obtained via the normal equations:

$$\alpha^{(k)} = (AR^{(k)}AR^{(k)})^{-1}(AR^{(k)})^T\mathbf{b}.$$

This algorithm will accumulate all the residuals in the matrix R, which can be prohibitively expensive. Therefore, it is common to limit the subspace, either looking at the last m iterations or restarting the process every m iterations.

Algorithm: Vanilla restarted GMRES(m)
 # Input: $A \in \mathbb{R}^{n\times n}$ *SPD,* $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}^{(0)} \in \mathbb{R}^n$,
  maxIter, ϵ, *Convergence criterion*
 # Output: $\mathbf{x}$ *s.t* $A\mathbf{x} \approx \mathbf{b}$
for $t = 1, ..., maxIter$ **do**
  Define the first residual and directions:
  $\mathbf{w}^{(0)} = \mathbf{r}^{(t-1)} = \mathbf{b} - A\mathbf{x}^{(t-1)}$
  $R^{(0)} =$ [empty array]
  $AR^{(0)} =$ [empty array]
  for $k = 1, ..., m$ **do**
    $R^{(k)} \leftarrow [R^{(k-1)}|\mathbf{w}^{(k-1)}]$
    $\mathbf{w}^{(k)} = A\mathbf{w}^{(k-1)}$
    $AR^{(k)} \leftarrow [AR^{(k-1)}|\mathbf{w}^{(k)}]$
    $\alpha^{(k)} = (AR^{(k)}AR^{(k)})^{-1}(AR^{(k)})^T\mathbf{r}^{(t-1)}$.
    If convergence is reached (compactly check $\|AR^{(k)}\alpha^{(k)} - \mathbf{r}^{(t-1)}\|_2^2$), break
  end
  $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)} + R^{(k)}\alpha^{(k)}$,
end
Return $\mathbf{x}^{(t)}$ as the solution.

Algorithm 11: The restarted vanilla GMRES method (not stable for large subspaces). Remark: in practice we do not recompute all $(AR^{(k)}AR^{(k)})$. We only compute the addition w.r.t previous iteration. There are other small tricks to save repeated computations.

The problem with the algorithm above is that the matrices AR tends to be ill-conditioned very fast (columns tend to be linearly dependent). The solution is unstable. The method GCR(m) aims to perform an orthogonalization of AR as part of the iterations through a (modified) Gram Schmidt process. The real method GMRES does a similar orthogonalization process for the matrix R which is less prone to be linearly dependent than AR. The process that they used is also known as the Arnoldi method. Therefore, GMRES is more stable than the GCR method and the vanilla orthmin method showed above. In terms of exact arithmetics, all the methods are identical.

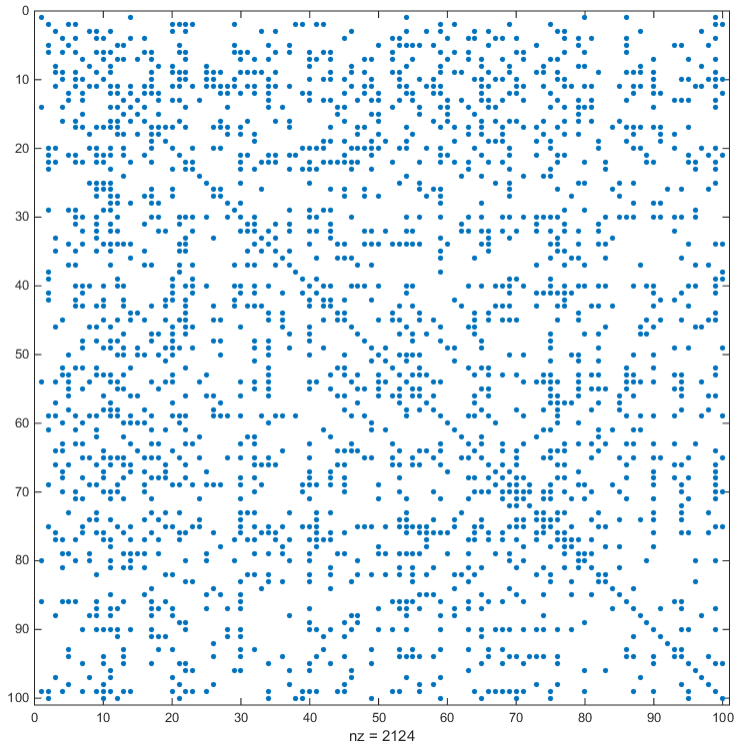

Figure 9: The non-zero structure of a random sparse matrix. The figure is achieve by `A = sprand(100,100,0.05);spy(A'*A);` in MATLAB.

6.4 Sparse Matrices

Iterative methods are useful when the multiplication of a matrix A with a vector can be computed efficiently. One of the most common instances of such a case is sparse matrices. A matrix $A \in \mathbb{R}^{m \times n}$ is called sparse when the number of non-zeros in A are much less than mn. Figure 9 demonstrates a non-zero structure of a sparse matrix, where the blue dots correspond to non-zero entries and the white space corresponds to zero entries.

Common data structures for storing sparse matrices The main advantage of sparse matrices is their compact storage. Assume the following matrix:

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 1 & -2 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

1. **Coordinate list (COO)**: COO stores a list of (row, column, value) tuples. Ideally, the entries are sorted (by row index, then column index) to improve random access times.
   `rowIdx = [1, 2, 4, 2, 4]`
   `colIdx = [1, 1, 1, 2, 3]`
   `values = [2, 1, 1,-2, 1]`.

2. **Compressed sparse row (CSR)** In this storage format the non-zero values are sorted by rows, and we have a pointer list `rowPtr` to the beginning of each row. The list is of size $m+1$, and row i starts at index `rowPtr[i]` and ends at `rowPtr[i+1]-1`. The matrix above is stored by:
   `rowPtr = [1, 2, 4, 4, 6]`
   `colIdx = [1, 1, 2, 1, 3]`
   `values = [2, 1,-2, 1, 1]`.
   This approach is efficient for multiplying $A\mathbf{x}$.

3. **Compressed sparse column (CSC)** The storage format is similar to CSR, only now the matrix is sorted by columns and not rows. That is, now we have a pointer list `colPtr` to the beginning of each column, of size $n+1$. The matrix above is stored by:
   `colPtr = [1, 4, 5, 6]`
   `rowIdx = [1, 2, 4, 2, 4]`
   `values = [2, 1, 1,-2, 1]`. This approach is efficient for multiplying $A^\top\mathbf{x}$. This is the storage type for sparse matrices in Julia and MATLAB.