$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_n
\end{aligned}
$$

Systems of equations such as the one above arise very frequently in all types of applications, and an important problem in numerical linear algebra is to develop methods for solving such systems efficiently. We will often write this system in the form

$$
\begin{pmatrix}
a_{11} & a_{12} & \dots & a_{1n} \\
a_{21} & a_{22} & \dots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \dots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{pmatrix}
=
\begin{pmatrix}
b_1 \\ b_2 \\ \vdots \\ b_n
\end{pmatrix}
$$

using matrix multiplication with the shorthand $Ax = b$. We will explore a few general-purpose methods as well as some specialized methods that arise in particular applications. In general, if $m \neq n$, there will be infinitely many ($m < n$) or zero ($m > n$) solutions to the above system. For our numerical methods, we will want to focus on the particular case where there exists a unique solution so we will require $m = n$. We further require that at least one of the following equivalent conditions holds:

- $A = (a_{ij})$ is invertible.
- $\det(A) \neq 0$
- A has rank n.
- A has only nonzero eigenvalues.
- etc.

1 Direct methods for solving systems

The simplest kind of system is *diagonal*:

$$
\begin{aligned}
a_{11}x_1 &= b_1 \\
&\vdots \\
a_{nn}x_n &= b_n
\end{aligned}
$$

because we can immediately obtain the solution

$$
x_i = \frac{b_i}{a_{ii}}.
$$

The next simplest kind of system is *triangular*. This comes in two forms:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{nn}x_n &= b_n \end{aligned}$$

and

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_n. \end{aligned}$$

In the first case, we can apply *backward-substitution* to find a solution fairly easily

$$x_n = \frac{b_n}{a_{nn}}$$

Once we figure out the value of x_n, we can then substitute it "backwards"

$$\begin{aligned} x_{n-1} &= \frac{1}{a_{n-1,n-1}}\left(b_{n-1} - a_{n-1,n}x_n\right) \\ &\vdots \\ x_1 &= \frac{1}{a_{11}}\left(b_1 - \sum_{j=2}^{n} a_{1j}x_j\right). \end{aligned}$$

Analogously, in the second, we can apply *forward-substitution*. Since these systems are fairly straightforward to solve, one direct method for solving systems of equations is to transform an arbitrary system into a triangular one. To do this, we form the *augmented matrix* $[A|b]$ and then apply *Gaussian elimination* to find a row-echelon form. (Recall that Gaussian elimination uses *elementary row operations* to produce a Row-echelon form. The allowed operations are switching two rows, adding a scalar multiple of one row to another, and multiplying a row by a scalar. I will not describe Gaussian elimination in these notes explicitly – feel free to consult any external resource you prefer to see many, many examples.) Then we extract the unique solution by back-substitution.

1.1 Partial Pivoting

As a part of Gaussian elimination, we may be required to exchange two rows in the matrix A. However, there are situations in which we may want to exchange two rows even if Gaussian elimination does not strictly require it.

Example 1.1. Consider the system of equations

$$\begin{aligned} \varepsilon x_1 + x_2 &= 1 + \varepsilon \\ x_1 + x_2 &= 2 \end{aligned}$$

for ε sufficiently small, e.g. $\varepsilon = 2^{-100}$. We can see by inspection that a solution is $x_1 = x_2 = 1$. However, if we try to solve this using Gaussian elimination on a computer (so using floating point arithmetic), we will instead get

$$\left(\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \to \left(\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & fl(1 - \frac{1}{\varepsilon}) = -\frac{1}{\varepsilon} & fl(2 - \frac{1}{\varepsilon}) = -\frac{1}{\varepsilon} \end{array}\right)$$

which has the solution $x_2 = 1$ and $x_1 = 0$ by back-substitution. In this example, roundoff errors lead us astray. While this example is fairly artificial, there are many "reasonable" systems of equations where a lack of care will lead to disaster.

Question 1.2. What feature of the system in the example above led to the disastrous loss of accuracy?

Example 1.3. Let's try the same example as before, but switching the order of equations first. We'll keep $\varepsilon = 2^{-100}$.

$$\begin{aligned} x_1 + x_2 &= 2 \\ \varepsilon x_1 + x_2 &= 1 + \varepsilon \end{aligned}$$

We can see by inspection that a solution is $x_1 = x_2 = 1$. Now, if we try to solve this using Gaussian elimination on a computer (so using floating point arithmetic), we will get

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ \varepsilon & 1 & 1 \end{array}\right) \to \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & fl(1 - \varepsilon) = 1 & fl(1 - 2\varepsilon) = 1 \end{array}\right)$$

which has the solution $x_2 = 1$ and $x_1 = 1$. This is exactly right, even though we faced some numerical issues along the way. (In general, we might hope that roundoff errors in the input lead to no worse than roundoff errors in the output...)

Question 1.4. What went right in this example compared to before?

Instead of having to scale by a very large number ($\frac{1}{\varepsilon}$), we were instead able to scale by a very small number instead (ε). This is the main idea behind *Gaussian elimination with partial pivoting*. At each stage, before we use row operations to zero out all the entries below a pivot position, we first search for the largest entry below the pivot position and use one row exchange to place that entry in the pivot position. The hope is that by doing this, we will avoid issues where we need to rescale an equation by a very large number, which would destroy accuracy at that step.

It turns out that while this works reasonably well in practice, there are some situations where even partial pivoting is insufficient. In these cases, we can go further with *scaled partial pivoting* and *complete pivoting*. In scaled partial pivoting, we use a preprocessing step where we first rescale each equation so that the largest coefficient a_{ij} which appears has

absolute value equal to 1. We accomplish this by dividing through the i^{th} equation by the largest a_{ij} in absolute value, where $1 \leq j \leq n$.

In complete pivoting, we not only search directly below the pivot position for a larger entry, but we also check the entire submatrix for its largest entry. As a tradeoff, we will then need to use column operations as well. This is quite costly, and is typically only used where the accuracy needed is very strict.

Remark 1.5. We can rephrase Gaussian elimination with (complete, scaled partial, partial) pivoting purely in terms of matrices by recognizing that the operations of scaling a row, adding a scalar multiple of one row to another, and switching two rows can all be accomplished by multiplying on the left by another matrix. Furthermore, the scaling matrix is diagonal and the adding a scalar multiple matrix is lower triangular. This means that if we never need to switch any rows, then Gaussian elimination can be viewed as a factorization $A = LU$ of A into a lower triangular matrix times an upper triangular matrix. Then we can solve $Ax = b$ by solving two triangular systems:

$$Ax = (LU)x = b \qquad \rightarrow \qquad Ly = b \qquad Ux = y.$$

One benefit of this approach is that if we need to solve a bunch of equations all with the same matrix A, then we don't need to perform the whole Gaussian elimination procedure every time since we can just reuse L and U.

In fact, there's always a way to permute the rows of A upfront so that we never need to switch the rows during Gaussian elimination. In matrix form, this means there is some permutation matrix P so that $PA = LU$.

2 Special kinds of matrices

While pivoting was useful to avoid bad numerical issues, it also costs time. For example, partial pivoting adds $O(n^2)$ operations (comparisons + row switches) while complete pivoting adds $O(n^3)$ operations (comparisons + row + column switches) that are required to be performed during the algorithm. While we would like not to have to perform any pivoting at all, this is generally impossible. However, there are a few particular types of matrices which do not require pivoting that arise frequently enough in practice that it is worth studying.

2.1 Diagonally dominant matrices

A square matrix A is *diagonally dominant* if

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

for every $1 \leq i \leq n$. In words, this says that a matrix is diagonally dominant if for the i^{th} row of A, the absolute value of the i^{th} entry of that row is larger than the sum of the absolute values of the other entries in that row. In other words, A is "dominated" by its diagonal entries. If each of the inequalities is strict, then A is *strictly diagonally dominant*.

Theorem 2.1. *If A is strictly diagonally dominant, then*

- *A is invertible*
- *Gaussian elimination can be performed without any pivoting*
- *The resulting solution will be stable with respect to the growth of roundoff errors.*

Intuitively, this result makes sense because the whole point of partial pivoting is to ensure that we never need to rescale by a very large value. If the largest values are already on the diagonal, then we will not run into this issue. Diagonally dominant matrices arise particularly frequently in solving certain kinds of differential equations, which we will discuss if we have time.

2.2 Symmetric, Positive definite matrices

A square matrix A is positive definite if it is symmetric and if $x^T Ax > 0$ for every vector $x \neq 0$. (Sometimes, the condition of symmetry is not required, but we will require it here.) One of the most useful criteria to check whether a matrix is positive definite or not is the following in terms of principal minors.

Theorem 2.2. *A matrix A is positive definite if and only if each leading principal submatrix*

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{bmatrix}$$

has a positive determinant.

A matrix A is positive definite if and only if all its eigenvalues are positive.

Positive definite matrices also behave very well with respect to Gaussian elimination.

Theorem 2.3. *A symmetric matrix A is positive definite if and only if Gaussian elimination can be performed without any pivoting and all pivot elements positive, and furthermore in this case the resulting solution will be stable with respect to the growth of roundoff errors.*

In fact, it turns out more is true:

Theorem 2.4. *A matrix A is positive definite if and only if it can be factored into the form $A = LDL^T$ where L is lower triangular with 1's along its diagonal and D is a diagonal matrix with positive entries along the diagonal.*

A matrix A is positive definite if and only if it can be factored in the form LL^T where L is lower triangular with nonzero diagonal entries.

These factorizations are closely related, and in particular the factorization $A = LL^T$ is known as the *Cholesky factorization* of A. It is possible and very reasonable to derive algorithms for computing the Cholesky factorization of a given positive definite matrix A

using the format $A = LL^T$ by writing out the entries of L explicitly and then solving for them in a principled manner.

It looks like we haven't gained much, since writing $A = LL^T$ is essentially just another version of $A = LU$, and we'll use it in the same way to solve systems of equations: $Ax = LL^Tx = b$ can be solved using the two steps $Ly = b$ followed by $L^Tx = y$. However, it turns out that Cholesky factorizations behave much better than general LU factorizations, both from a numerical as well as computational time perspective.

Positive definite matrices arise in numerous applications, and you can probably take entire classes devoted to studying positive definite matrices.

2.3 Banded matrices

A matrix A is a *band matrix* if there are $1 < p, q < n$ such that $a_{ij} = 0$ whenever $p \leq j - i$ or $q \leq i - j$. The *band width* is $w = p + q - 1$. Note that p tells you how many diagonals above and including the main diagonal have nonzero entries and q tells you how many diagonals below and including the main diagonal have nonzero entries. Banded matrices arise from certain methods for solving differential equations.

Two of the most common banded matrices are tridiagonal ($p = q = 2$) as well as $p = q = 4$. Recall that usual Gaussian elimination requires $O(n^3)$ operations to perform. However, if we instead assume that we can factor a tridiagonal A into tridiagonal L and U as well, then it turns out that we can achieve LU factorization in $O(n)$ operations instead. (To do this, write out the product LU and figure out a clever way to solve the resulting equations from $A = LU$.) This is a huge win, and the reason why certain methods for differential equations were designed with these results in mind.

3 Iterative methods

As an alternative to the previous "direct" methods for solving linear systems, we also have various iterative methods available. While relatively small linear systems are better solved by the direct methods we discussed above, there are certain situations where these iterative techniqeus work very well.

3.1 Jacobi method

The Jacobi method is based on the following idea. If we have that $a_{ii} \neq 0$ for all $1 \leq i \leq n$, then we can "solve" the i^{th} equation for x_i as follows:

$$x_i = \frac{b_i}{a_{ii}} - \sum_{j \neq i} \frac{a_{ij}}{a_{ii}} x_j$$

which suggests the following procedure.

1. Choose an initial guess x^0.
2. For each $1 \leq j \leq n$, find the j^{th} entry of the next iteration x^1 by plugging in the entries of x_0 into the right hand side for $x_j = \ldots$ above.

3. Repeat this procedure as long as desired.

We can also represent this in matrix form. For any matrix A, we will write $A = L+D+U$ where L is lower triangular, D is diagonal, and U is upper triangular. The way to do this is as follows

$$L = \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & 0 \end{pmatrix}, D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mn} \end{pmatrix}, U = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

Then the above method can also be derived using the following matrix algebra:

$$\begin{gathered} Ax = b \\ (L + D + U)x = b \\ Dx + (L + U)x = b \\ Dx = b - (L + U)x \\ x = D^{-1}b - D^{-1}(L + U)x \end{gathered}$$

In this form, we want to find a fixed point of the equation above, and the iterations can be represented as

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

Note the similarity between this idea and fixed point iteration which we discussed before.

3.2 Gauss-Seidel

The Gauss-Seidel method is a slight modification of the Jacobi method. The idea is that we compute x_j^{i+1} sequentially for each $1 \leq j \leq n$ and x_j^{i+1} is supposed to be more accurate than x_j^i, at least if the method works. Then for the next j, we might as well go ahead and use the more accurate estimate x_j^{i+1} that we already computed.

1. Choose an initial guess x^0.
2. For each $1 \leq j \leq n$, find the j^{th} entry of the next iteration x^1 by plugging in the entries x_k^0 for $k > j$ and x_k^1 for $k < j$ into the right hand side for $x_j = \dots$ above.
3. Repeat this procedure as long as desired.

As before, we can write this in matrix form as well.

$$\begin{gathered} (L + D + U)x = b \\ (L + D)x + Ux = b \\ (L + D)x = b - Ux \\ x = (L + D)^{-1}b - (L + D)^{-1}Ux \end{gathered}$$

In this form, the iterations can be represented as

$$x^{i+1} = (L + D)^{-1}b - (L + D)^{-1}Ux.$$

3.3 General iterative methods

In general, an iterative method for solving a linear system can be expressed in the format

$$x^{i+1} = c + Tx^i$$

for some vector c and some matrix T. Jacobi and Gauss-Seidel as presented above as two examples of this, but there are many more possibilities as well. In order to solve the system $Ax = b$ using the above format, we are looking for a fixed point

$$x = c + Tx$$

such that the equation can be rearranged to

$$\begin{aligned} x &= c + Tx \\ x - Tx &= c \\ (I - T)x &= c \end{aligned}$$

such that there is some invertible matrix M satisfying $A = M(I - T)$ and $b = Mc$.

3.4 Error analysis for iterative methods

Suppose that we transform the system $Ax = b$ into the iterative format

$$x^{i+1} = c + Tx^i.$$

Since we are looking for a fixed point

$$x = c + Tx$$

we can subtract the two equations to find our errors. Let $e^i = x - x^i$ represent the error of the i^{th} iteration. Then we get the equation

$$e^{i+1} = Te^i.$$

Thus, we see that the error at one step is exactly the iteration function T applied to the previous error. In general, we would like the error to go to 0, but this can be quite difficult to analyze. However, but there is one situation in which we can guarantee that the errors do converge to 0.

Definition 3.1. For any (square) matrix A, the *spectral radius* $\rho(A)$ is defined to be the maximum absolute value of an eigenvalue of A.

Theorem 3.2. *If the fixed point formulation $x = c + Tx$ of a linear system $Ax = b$ has a unique fixed point (so $Ax = b$ has a unique solution), then the iterations will converge if $\rho(T) < 1$.*

The idea is fairly straightforward here. If $\rho(T) < 1$, then expressing e^0 in terms of an eigenbasis of T gives

$$e^0 = \sum_{k=1}^{n} d_i v_i \implies Te^0 = \sum_{k=1}^{n} d_k T(v_k) = \sum_{k=1}^{n} d_k \lambda_k v_k.$$

Repeating this idea, we obtain

$$e^i = T^i e^0 = \sum_{k=1}^{n} d_k \lambda_k^i v_k.$$

Since $|\lambda_k| \leq \rho(T) < 1$, then $|\lambda_k^i| \xrightarrow{i\to\infty} 0$, so $e^i \xrightarrow{i\to\infty} 0$. As a result, we can also see in this case that the rate of convergence is essentially determined by the spectral radius.

$$e^i = T^i e^0 = \sum_{k=1}^{n} d_k \lambda_k^i v_k \leq \rho(T)^i \sum_{k=1}^{n} d_k v_k = \rho(T)^i e^0.$$

Thus the best choice of iteration matrix is the one which minimizes the spectral radius, though can be difficult to determine in practice.

3.5 Successive over-relaxation (SOR)

One generalization of the Gauss-Seidel involves scaling the equation $Ax = b$ by a constant $\omega \in (0, 2)$ in an attempt to minimize the spectral radius of the iteration matrix:

$$\begin{gathered}\omega(L + D + U)x = \omega b \\ (\omega L + D)x + ((\omega - 1)D + \omega U)x = \omega b \\ x = (\omega L + D)^{-1}\omega b - (\omega L + D)^{-1}((\omega - 1)D + \omega U)x\end{gathered}$$

In this case, the iteration matrix is

$$(\omega L + D)^{-1}((\omega - 1)D + \omega U)$$

and in general it's quite difficult to select an appropriate ω. However, one case where SOR is very useful is when A is positive definite and tridiagonal. (This situation arises very frequently when studying certain partial differential equations.) In this case, the optimal choice of ω is

$$\omega = \frac{2}{1 + \sqrt{1 - \rho(T_j)^2}}$$

where T_j is the Jacobi iteration matrix $-D^{-1}(L + U)$. In this case, we will have that $\rho((\omega L + D)^{-1}((\omega - 1)D + \omega U)) = \omega - 1$.

4 Condition number

In our error analysis, we showed that if the spectral radius is less than 1, then the errors converge to zero. In particular, we saw that the error at the i^{th} step is dependent upon the initial error $e^i = T^i e^0 \leq \rho(T)^i e^0$ so if the initial guess is very bad, then we may require extra iterations to achieve a desired tolerance. However we generally do not know what the exact solution x is so it's difficult to use this to understand the error at a particular step.

One thing we can compute is the residual $b - Ax^i$ to see how far off the output is from the desired output. However, there is a subtle difference between a small residual and a small error, as the following example shows.

Example 4.1. Consider the linear system

$$x_1 + 2x_2 = 3$$
$$1.0001 + 2x_2 = 3.0001$$

which has solution $(x_1, x_2) = (1, 1)$. However, consider the approximation $(3, 0)$. We have the residual $b - A\tilde{x} = (0, -.0002)$ which is very close to zero. However, the difference $(1, 1) - (3, 0) = (-2, 1)$ is not very close to zero, so this approximation is not very good even though the residual was very small.

This example is somewhat artificial, but serves as a good illustration of what can go wrong. The issue here is that the two lines are almost parallel, so there are many "near" solutions which can be quite far from the actual solution.

Let $\tilde{x}$ be an approximation to the unique solution x of $Ax = b$ and $b - A\tilde{x} = r$ the residual of this approximation. Then

$$A(x - \tilde{x}) = b - A\tilde{x} = r \implies x - \tilde{x} = A^{-1}r.$$

Now we can take the 2-norms of both sides (square root of the sum of squares of the entries):

$$||x - \tilde{x}||_2 = ||A^{-1}r||_2 \leq ||A^{-1}||_2 ||r||_2.$$

The 2-norm of a matrix is the largest absolute value of any of its *singular values*. In general, we care more about the relative error

$$\frac{||x - \tilde{x}||_2}{||x||_2}.$$

To bound the relative error, we use the estimate

$$||A||_2 ||x||_2 \geq ||Ax||_2 = ||b||_2 \implies \frac{1}{||x||_2} \leq \frac{||A||_2}{||b||_2}$$

to obtain

$$\frac{||x - \tilde{x}||_2}{||x||_2} \leq ||A^{-1}||_2 ||A||_2 \frac{||r||_2}{||b||_2}.$$

This gives an indication for how closely the relative error of the approximation is estimated by the relative error of the residual $b - A\tilde{x}$. The constant of proportionality here is $||A^{-1}||_2 ||A||_2$. Since the singular values of A^{-1} are the inverses of the singular values of A, the constant of proportionality here is the ratio of the largest singular value of A to the smallest singular value of A.

Definition 4.2. The *condition number* of a matrix A with respect to the matrix 2-norm is the ratio of its largest and smallest singular values. The condition number can be defined more generally with respect to any (natural) matrix norm as $||A^{-1}||\ ||A||$ though the interpretation may not be so straightforward as it is for the matrix 2-norm.

If the condition number of A is reasonably small (the smallest it could possibly be is 1 if all the singular values are identical) then you can expect that solving linear systems involving A will go roughly "as expected." However, if the condition number is large, then that is always cause for suspicion – if something goes wrong, investigating how the method depends on your matrix is a good place to look for errors.

Example 4.3. The matrix A in the example above is

$$\begin{pmatrix} 1 & 2 \\ 1.0001 & 2 \end{pmatrix}.$$

Its singular values are approximately 3.16231 and .0000632449, so their ratio is ≈ 50000, which is rather large. Therefore, we should be suspicious of using the residuals directly to track the performance of an iterative method.