# ELIMINATION

*David F. Gleich*

August 21, 2023

## 1 ELIMINATION METHODS FOR LINEAR SYSTEMS

Lecture 11: Finite methods for solving linear systems of equations.

Thus far, we've seen methods that solve $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ via a sequence of vector changes. These methods have worked by updating $\mathbf{x}^{(k)}$ to $\mathbf{x}^{(k+1)}$. At no point did they consider changing the system $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ into another system $\boldsymbol{B}\mathbf{y} = \mathbf{d}$, where $\mathbf{y}$ is somehow easier to find than $\mathbf{x}$ and we can compute $\mathbf{x}$ from $\mathbf{y}$ in a simple fashion.

The next class of methods we will look at will do exactly this! From $\boldsymbol{A}\mathbf{x} = \mathbf{b}$, we will produce a sequence of systems that get progressively smaller by *eliminating* variables. The methods go by a variety of names: elimination, Gaussian elimination, LU decomposition, Cholesky factorization, and even more names including the Schur complement. However, the key idea is almost always the same.

### 1.1 VARIABLE ELIMINATION

Consider the case where we are solving a system $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ then we can write this out and highlight the first row as follows:[1]

$$\boldsymbol{A} = \begin{bmatrix} \alpha & \mathbf{c}^T \\ \mathbf{d} & \boldsymbol{R} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \beta \\ \mathbf{f} \end{bmatrix}.$$

[1] Of course, the curious will wonder what is special about the first row. As is common, there is nothing special about the first row and this could be done for any row. We'll return to this idea later.

Let the solution correspond to the elements

$$\mathbf{x} = \begin{bmatrix} \gamma \\ \mathbf{y} \end{bmatrix}$$

so that

$$\alpha\gamma + \mathbf{c}^T\mathbf{y} = \beta.$$

Then the idea behind all of the elmination methods is that, if we are given $\mathbf{y}$ by some type of oracle, we can compute $\gamma$ from $\mathbf{y}$

$$\gamma(\mathbf{y}) = \frac{1}{\alpha}\left(\beta - \mathbf{c}^T\mathbf{y}\right).$$

This is neat, it says that if you had all by one solution of your system, it's easy to find missing element. [2]

[2] Careful readers will note that we need $\alpha \neq 0$ for this idea to work.

We aren't quite done, however, because this hasn't simplified or changed or system at all. To do that, note that the remaining equations give the expression

$$\gamma\mathbf{d} + \boldsymbol{R}\mathbf{y} = \mathbf{f}.$$

This expression involves $\gamma$ and $\mathbf{y}$. But we have $\gamma$ as a function of $\mathbf{y}$ and so let's just substitute that in. The result is an expression purely in terms of $\mathbf{y}$

$$\gamma(\mathbf{y})\mathbf{d} + \boldsymbol{R}\mathbf{y} = \frac{1}{\alpha}\left(\beta - \mathbf{c}^T\mathbf{y}\right)\mathbf{d} + \boldsymbol{R}\mathbf{y} = \mathbf{f}.$$

By re-arranging the equations, we arrive at the following linear system:

$$(\boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T)\mathbf{y} = \mathbf{f} - \frac{\beta}{\alpha}\mathbf{d}.$$

**A 4x4 example.** Suppose we have:

$$\begin{bmatrix} -2 & -1 & -4 & -1 \\ -1 & -5 & -5 & -2 \\ 4 & 5 & 2 & 0 \\ -2 & -2 & -1 & 0 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 4 \\ 4 \\ -5 \\ 2 \end{bmatrix}$$

then

$$(\boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T)\mathbf{y} = \begin{bmatrix} -4.5 & -3 & -1.5 \\ 3 & -6 & -2 \\ -1 & 3 & 1 \end{bmatrix} \mathbf{y} = \begin{bmatrix} 2 \\ 3 \\ -2 \end{bmatrix}$$

where $\mathbf{y} = \begin{bmatrix} -1 \\ -14/3 \\ 11 \end{bmatrix}$ and $\gamma = 7/3$ so $\mathbf{x} = \begin{bmatrix} 7/3 \\ -1 \\ -14/3 \\ 11 \end{bmatrix}$

If $\boldsymbol{A}$ was $n \times n$, then this method takes the system

$$(\boldsymbol{A}, \mathbf{b}) \quad \text{to} \quad (\boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T, \mathbf{f} - \frac{\beta}{\alpha}\mathbf{d})$$

where this new system is $(n-1) \times (n-1)$. Hence, we arrive at an *easier* or smaller system to solve! To solve it, we can apply the same idea again until we get down to a $1 \times 1$ system. This algorithm is easy to implement on a computer that supports recursion.

```
function elimination_solve(A::Matrix, b::Vector)
  m,n = size(A)
  @assert(m==n, "the system is not square")
  @assert(n==length(b), "vector b has the wrong length")
  if n==1
    return [b[1]/A[1]]
  else
    R = A[2:end,2:end]
    c = A[1,2:end]
    d = A[2:end,1]
    alpha = A[1,1]
    y = elimination_solve(R-d*c'/alpha, b[2:end]-b[1]/alpha*d)
    gamma = (b[1] - c'*y)/alpha
    return pushfirst!(y,gamma)
  end
end
```

This idea is called *variable elimination*. We elminate the variable $\gamma$ from the system of equation $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ by solving for its expression and them substituting that solution into the rest of the equations.

Note that if $\boldsymbol{A}$ is symmetric, then $\mathbf{d} = \mathbf{c}$ and hence $\boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T = \boldsymbol{R} - \frac{1}{\alpha}\mathbf{c}\mathbf{c}^T$ is symmetric as well.

In fact, if $\boldsymbol{A}$ is symmetric positive definite, then $\mathbf{z}^T\boldsymbol{A}\mathbf{z} > 0$ for any $\mathbf{z}$. We can show that $\boldsymbol{D} - \frac{1}{\alpha}\mathbf{c}\mathbf{c}^T$ is *also* symmetric positive definite too! To do so, we will show that $\mathbf{g}^T\boldsymbol{R}\mathbf{g} - \frac{1}{\alpha}(vc^T\mathbf{g})^2 > 0$ for any $\mathbf{g}$. We consider using a specially chosen vector $\mathbf{z}$ applied to the equation for $\boldsymbol{A}$

$$\mathbf{z}^T\boldsymbol{A}\mathbf{z} = \begin{bmatrix} \rho \\ \mathbf{g} \end{bmatrix}^T \begin{bmatrix} \alpha & \mathbf{c}^T \\ \mathbf{c} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \rho \\ \mathbf{g} \end{bmatrix} = \alpha\rho^2 + 2\rho\mathbf{c}^T\mathbf{g} + \mathbf{g}^T\boldsymbol{D}\mathbf{g}.$$

At the moment, $\rho$ is still arbitrary. We can choose it to be anything. However, our goal is to pick $\rho$ such that we learn about $\mathbf{g}^T\boldsymbol{R}\mathbf{g} - \frac{1}{\alpha}(\mathbf{c}^T\mathbf{g})^2$. To do so, let $\rho = -(\mathbf{c}^T\mathbf{g})/\alpha$. Then

$$\alpha\rho^2 + 2\rho\mathbf{c}^T\mathbf{g} + \mathbf{g}^T\boldsymbol{D}\mathbf{g} = (\mathbf{c}^T\mathbf{g})^2/\alpha - 2(\mathbf{c}^T\mathbf{g})^2/\alpha + \mathbf{g}^T\boldsymbol{D}\mathbf{g} = \mathbf{g}^T\boldsymbol{D}\mathbf{g} - (\mathbf{c}^T\mathbf{g})^2/\alpha > 0$$

as required.

This means that if we *eliminate* a variable on a symmetric positive definite system. The remaining system is still symmeric positive definite.

### 1.2 VARIABLE ELIMINATION AS A MATRIX EXPRESSION

The really interesting part about variable elimination is that we can express it as a matrix expression! The following expression seems like magic. Essentially, by examining the above equation long enough, we can deduce an expression like the following. It allows us to express the elimination operation as a matrix itself. Again, let $\boldsymbol{A} = \begin{bmatrix} \alpha & \mathbf{c}^T \\ \mathbf{d} & \boldsymbol{R} \end{bmatrix}$. Then

$$\underbrace{\begin{bmatrix} 1 & 0 \\ -\mathbf{d}/\alpha & \boldsymbol{I} \end{bmatrix}} = \boldsymbol{L}_1 \underbrace{\begin{bmatrix} \alpha & \mathbf{c}^T \\ \mathbf{d} & \boldsymbol{R} \end{bmatrix}}_{=\boldsymbol{A}} \underbrace{\begin{bmatrix} 1 & -\mathbf{c}^T/\alpha \\ 0 & \boldsymbol{I} \end{bmatrix}}_{=\boldsymbol{U}_1} = \begin{bmatrix} \alpha & 0 \\ 0 & \boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T \end{bmatrix}.$$

Note that $\boldsymbol{L}_1$ is a non-singular matrix of the form:

$$\boldsymbol{L}_1 = \boldsymbol{I} - \mathbf{v}\mathbf{e}_1 \text{ where } v_1 = 0.$$

This means that $\boldsymbol{L}_1^{-1} = \boldsymbol{I} + \mathbf{v}\mathbf{e}_1$, which can be verified because $\boldsymbol{L}_1\boldsymbol{L}_1^{-1} = \boldsymbol{I}$. Likewise, $\boldsymbol{U}_1 = \boldsymbol{I} - \mathbf{e}_1\mathbf{u}^T$ where $\mathbf{u}_1 = 0$. It's inverse is $\boldsymbol{I} + \mathbf{e}_1\mathbf{u}^T$ as well. Using these matrices, we can transform:

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \rightarrow \boldsymbol{L}_1\boldsymbol{A}\boldsymbol{U}_1\boldsymbol{U}_{-1}\mathbf{x} = \boldsymbol{L}_1\mathbf{b}.$$

If we expand this block-wise, then we get:

$$\begin{bmatrix} \alpha & 0 \\ 0 & \boldsymbol{R} - \frac{1}{\alpha}\mathbf{d}\mathbf{c}^T \end{bmatrix} \begin{bmatrix} \gamma + \frac{1}{\alpha}\mathbf{c}^T\mathbf{y} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\mathbf{d}/\alpha & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \beta \\ \mathbf{f} \end{bmatrix},$$

which is exactly our reduced system.

Consequently, we can express our entire sequence of reductions as follows:

$$\boldsymbol{L}_{n-1}\boldsymbol{L}_{n-2}\cdots\boldsymbol{L}_1\boldsymbol{A}\boldsymbol{U}_1\boldsymbol{U}_2\cdots\boldsymbol{U}_{n-1} = \begin{bmatrix} \alpha_1 & & & \\ 0 & \alpha_2 & & \\ 0 & 0 & \dots & \\ & & & \alpha_n \end{bmatrix} = \boldsymbol{D}$$

or

$$\boldsymbol{A} = \boldsymbol{L}_1^{-1}\boldsymbol{L}_2^{-1}\cdots\boldsymbol{L}_{n-1}^{-1}\boldsymbol{D}\boldsymbol{U}_{n-1}^{-1}\boldsymbol{U}_{n-2}\cdots\boldsymbol{U}_1^{-1}.$$

It turns out that these elimination matrix $\boldsymbol{L}_i^{-1}$ and $\boldsymbol{U}_i^{-1}$ have some rather special properties that allow us to realize this form in an exceedingly simple way. For all $i$ we have $\boldsymbol{L}_i^{-1} = (\boldsymbol{I} + \mathbf{v}\mathbf{e}_i)$ with $v_1 = v_2 = \ldots = v_i = 0$ and so

$$\boldsymbol{L}_i^{-1}\boldsymbol{L}_j^{-2} = (\boldsymbol{I} + \mathbf{v}_i\mathbf{e}_i^T)(\boldsymbol{I} + \mathbf{v}_j\mathbf{e}_j^T) = \boldsymbol{I} + \mathbf{v}_i\mathbf{e}_i^T + \mathbf{v}_j\mathbf{e}_j^T + \mathbf{v}_i\mathbf{e}_i^T\mathbf{v}_j\mathbf{e}_j^T = \boldsymbol{I} + \mathbf{v}_i\mathbf{e}_i^T + \mathbf{v}_j\mathbf{e}_j^T \text{ when } i < j.$$

This enables us to quickly compute these as follows:

```
function myreduce_all(A::Matrix)
  A = copy(A) # save a copy
  n = size(A,1)
  L = Matrix(1.0I,n,n)
  U = Matrix(1.0I,n,n)
  d = zeros(n)
  for i=1:n-1
    alpha = A[i,i]
    d[i] = alpha
    U[i,i+1:end] = A[i,i+1:end]/alpha
    L[i+1:end,i] = A[i+1:end,i]/alpha
    A[i+1:end,i+1:end] -= A[i+1:end,i]*A[i,i+1:end]'/alpha
  end
  d[n] = A[n,n]
  return L,U,d
end
L,U,d = myreduce_all(A)
L*Diagonal(d)*U - A
```

This is what is most commonly called the *LU* decomposition of a matrix.

Suppose we start with the system

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}.$$

This has the solution $x = 1$, $y = 5$.

Then if we try the variable elimination approach, our first step is

$$0x + y = 5$$

$$x = \frac{5 - y}{0}$$

break!

This scenario involves division by zero because $x$ really is not really a component of that system!

The solution is easy, if we wish to eliminate $x$ from this equation, we need to use an equation that includes $x$.[3] In this case, we can simply swap rows

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}.$$

After which we have

$$x + y = 6 \rightarrow x = 6 - y$$

$$y = 5$$

which we can quickly solve.

Computers need more structure in order to realize these same things. Pivoting is the idea they use to reorder the equations.

[3] This is identical to how in Jacobi and Gauss-Seidel, if we wished to update the value for a variable $x_i$, we needed to use an equation that used the variable $x_i$.

DEFINITION 1 (Pivoting) *Pivoting reorders the equations (rows) of $\boldsymbol{A}$ in a linear system so that we can compute an LU-decomposition for any non-singular system.*

We can always use pivoting to find a variable to eliminate.

THEOREM 2 *If $\boldsymbol{A}$ is non-singular, then at each step of an LU factorization, there must be a variable and equation pair that we can eliminate.*

Proof Assume by way of contradiction that we *cannot* find an equation (row) to eliminate a variable (column) in the $k$th step of an LU factorization. After $k - 1$ steps of an LU factorization we have

$$\boldsymbol{L}_{k-1}\boldsymbol{L}_{k-2}\cdots\boldsymbol{L}_1\boldsymbol{A}\boldsymbol{U}_1\boldsymbol{U}_2\cdots\boldsymbol{U}_{k-1} = \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & \ddots & & \\ & & & d_{k-1} & \\ & & & & \boldsymbol{R} \end{bmatrix}$$

By our assumption, we are in the scenario where there is no equation (row) of $\boldsymbol{R}$ to eliminate the $k$th variable. The $k$th variable is involved in the first column of $\boldsymbol{R}$. This means that the first column of $\boldsymbol{R}$ is entirely zero. This implies that $\boldsymbol{R}$ is singular because it has a column that is entirely zero.

Now, $\boldsymbol{A}$ is non-singular, as are the products

$$\boldsymbol{L}_{k-1}\boldsymbol{L}_{k-2}\cdots\boldsymbol{L}_1 \text{ and } \boldsymbol{U}_1\boldsymbol{U}_2\cdots\boldsymbol{U}_{k-1}.$$

Consequently, the left-hand side is non-singular, which means the right hand side must be as well. However, our assumption implied that $\boldsymbol{R}$ was singular, which is how we arise at the contradiction. ∎

This gives rise to the following algorithm for solving a system of linear equations.

```
function solve1_pivot!(A::Matrix, b::Vector)
  m,n = size(A)
  @assert(m==n, "the system is not square")
  @assert(n==length(b), "vector b has the wrong length")
  if n==1
    return [b[1]/A[1]]
  else
    # let's make sure we have an equation
    # that we can eliminate!
    alpha = A[1,1]
    newrow = 1
    if alpha == 0
      for j=2:n
        if A[j,1] != 0
          newrow = j
          break
        end
      end
      if newrow == 1
        error("the system is singular")
      end
    end
    # swap rows 1, and newrow
    if newrow != 1
      tmp = A[1,:]
      A[1,:] .= A[newrow,:]
      A[newrow,:] .= tmp
      b[1], b[newrow] = b[newrow], b[1]
    end
    D = A[2:end,2:end]
    c = A[1,2:end]
    d = A[2:end,1]
    alpha = A[1,1]
    y = solve1_pivot!(D-d*c'/alpha, b[2:end]-b[1]/alpha*d)
    gamma = (b[1] - c'*y)/alpha
    return pushfirst!(y,gamma)
  end
end
```

### 1.3 ON SYMMETRIC MATRICES

Suppose we are able to run the $LU$ decomposition of a matrix with no pivoting. In other words, suppose that $\alpha$ = `A[i,i]` is non-zero at each step. In this case, we produce:

$$\boldsymbol{A} = \boldsymbol{LDU}.$$

Now, because $\boldsymbol{A}$ is symmetric, we have

$$\boldsymbol{A} = \boldsymbol{A}^T = \left(\boldsymbol{LDU}\right)^T = \boldsymbol{U}^T\boldsymbol{D}\boldsymbol{L}^T.$$

This strongly hints that $\boldsymbol{L} = \boldsymbol{U}^T$. This result is indeed correct, as can be verified by looking at the each step of the algorithm on a symmetric $\boldsymbol{A}$ and noting that we preserve symmetry at each step as shown above. However, in general, we cannot assume that any symmetric matrix can be decomposed like this without pivoting.

[4] here is a more general $\boldsymbol{LDL}^T$ factorization that can be computed in such cases. A simple counter example is: $\boldsymbol{A} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right]$.

[4] T

### 1.4 THE CHOLESKY DECOMPOSITION

The Cholesky decomposition is the LU decomposition, without pivoting, applied to symmetric positive definite matrix. For a symmetric positive definite matrix, we can show that pivoting is not required. This is actually a corollary of one of the definitions of what it means to be a positive definite matrix.

Consequently, for a symmetric positive definite matrix we always have

$$\boldsymbol{A} = \boldsymbol{LDL}^T.$$

Moreover, because $\boldsymbol{A}$ is positive definite, we have $D_{i,i} > 0$. This happens because after each step of the reduction, the reduced matrix is *also* positive definite. The diagonal entries of a positive definite matrix are always positive, and these determine the entries of the diagonal $\boldsymbol{D}$. As shown above, after each elimination step, the matrix remains positive definite as well.

Because $\boldsymbol{D}$ is strictly positive, we can take the square root of each entry and compute

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}^{1/2}(\boldsymbol{D}^{1/2}\boldsymbol{L})^T = \boldsymbol{F}\boldsymbol{F}^T \text{ or } \boldsymbol{F}^T\boldsymbol{F}.$$

This gives the Cholesky routine

```
""" Compute the Cholesky factorization A = FF' and return F """
function (A::Matrix)
  A = copy(A) # save a copy
  n = size(A,1)
  F = Matrix(1.0I,n,n)
  d = zeros(n)
  for i=1:n-1
    alpha = A[i,i]
    d[i] = sqrt(alpha)
    F[i+1:end,i] = A[i+1:end,i]/alpha
    A[i+1:end,i+1:end] -= A[i+1:end,i]*A[i,i+1:end]'/alpha
  end
  d[n] = sqrt(A[n,n])
  return F*Diagonal(d), d
end
PD = A'*A
F,d = myreduce_all_cholesky(PD)
F*F' - A
```

Compare with the previous LU code to see the subtle differences.

What happens if your matrix is not symmetric positive definite? Then at some point in the decomposition, you will have $\alpha < 0$. This is actually one of the fastest ways to test if a matrix is symmetric positive definite as it avoid all eigenvalue computations.

### 1.5 A MORE GENERAL PERSPECTIVE ON THIS IDEA.

— TODO – Finish

There is no restriction that the regions have no overlap when we split into two pieces. Consider:

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

where $\mathbf{x} = \boldsymbol{C}\mathbf{y} + \boldsymbol{D}\mathbf{z}$ for a $n \times n - 2$ full-rank matrix $\boldsymbol{C}$ and a $n \times n - 1$ full-rank matrix $\boldsymbol{D}$. We require the matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ be able to express any vector $\mathbf{x}$. [5]

[5] This can be formalized, but the exact property escapes me as I'm writing these notes.