Decompositions and least squares

We now look at several decompositions (or factorisations) of a matrix into products of structured matrices, and their use in solving least squares problems and linear systems. For a square or rectangular matrix $A \in \mathbb{R}^{m\times n}$ with more rows than columns ($m \geq n$), we consider:

1. The *QR decomposition*

$$A = QR = \underbrace{[\,\mathbf{q}_1|\cdots|\mathbf{q}_m\,]}_{m\times m}\underbrace{\begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ & & \times \\ & & 0 \\ & & \vdots \\ & & 0 \end{bmatrix}}_{m\times n}$$

   where Q is orthogonal ($Q^\top Q = I$, $\mathbf{q}_j \in \mathbb{R}^m$) and R is *right triangular*, which means it is only nonzero on or to the right of the diagonal.

2. The *reduced QR decomposition*

$$A = QR = \underbrace{[\,\mathbf{q}_1|\cdots|\mathbf{q}_n\,]}_{m\times n}\underbrace{\begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ & & \times \end{bmatrix}}_{n\times n}$$

   where Q has orthogonal columns ($Q^\top Q = I$, $\mathbf{q}_j \in \mathbb{R}^m$) and R is upper triangular.

For a square matrix we consider the *PLU decomposition*:

$$A = P^\top LU$$

where P is a permutation matrix, L is lower triangular and U is upper triangular.

Finally, for a square, *symmetric positive definite* ($\mathbf{x}^\top A\mathbf{x} > 0$ for all $\mathbf{x} \in \mathbb{R}^n$) matrix we consider the *Cholesky decomposition*:

$$A = LL^\top$$

The importance of these decomposition for square matrices is that their component pieces are easy to invert on a computer:

$$\begin{aligned} A = P^\top LU \Rightarrow &\qquad A^{-1}\mathbf{b} = U^{-1}L^{-1}P\mathbf{b} \\ A = QR \Rightarrow &\qquad A^{-1}\mathbf{b} = R^{-1}Q^\top\mathbf{b} \\ A = LL^\top \Rightarrow &\qquad A^{-1}\mathbf{b} = L^{-\top}L^{-1}\mathbf{b} \end{aligned}$$

and we saw last lecture that triangular and orthogonal matrices are easy to invert when applied to a vector $\mathbf{b}$, e.g., using forward/back-substitution. For rectangular matrices we

will see that they lead to efficient solutions to the *least squares problem*: find $\mathbf{x}$ that minimizes the 2-norm

$$\|A\mathbf{x} - \mathbf{b}\|.$$

In this lecture we discuss the followng:

1. QR and least squares: We discuss the QR decomposition and its usage in solving least squares problems.
2. Reduced QR and Gram–Schmidt: We discuss computation of the Reduced QR decomposition using Gram–Schmidt.
3. Householder reflections and QR: We discuss comuting the QR decomposition using Householder reflections.
4. PLU decomposition: we discuss how the LU decomposition can be computed using Gaussian elimination, and the computation of the PLU decomposition via Gaussian elimination with pivoting.
5. Cholesky decomposition: we introduce symmetric positive definite matrices and show that their LU decomposition can be re-interpreted as a Cholesky decomposition.
6. Timings: we see the relative trade-off in speed between the different decompositions.

In [ ]:
```
using LinearAlgebra, Plots, BenchmarkTools
```

1. QR and least squares

Here we consider rectangular matrices with more rows than columns. A QR decomposition decomposes a matrix into an orthogonal matrix Q times a right triangular matrix R. Note the QR decomposition contains within it the reduced QR decomposition:

$$A = QR = \begin{bmatrix} Q|\mathbf{q}_{n+1}|\cdots|\mathbf{q}_m \end{bmatrix} \begin{bmatrix} R \\ \mathbf{0}_{m-n\times n} \end{bmatrix} = QR.$$

We can use it to solve a least squares problem using the norm-preserving property (see PS3) of orthogonal matrices:

$$\|A\mathbf{x} - \mathbf{b}\| = \|QR\mathbf{x} - \mathbf{b}\| = \|R\mathbf{x} - Q^\top \mathbf{b}\| = \left\| \begin{bmatrix} R \\ \mathbf{0}_{m-n\times n} \end{bmatrix} \mathbf{x} - \begin{bmatrix} Q^\top \\ \mathbf{q}_{n+1}^\top \\ \vdots \\ \mathbf{q}_m^\top \end{bmatrix} \mathbf{b} \right\|$$

Now note that the rows $k > n$ are independent of $\mathbf{x}$ and are a fixed contribution. Thus to minimise this norm it suffices to drop them and minimise:

$$\|R\mathbf{x} - Q^\top \mathbf{b}\|$$

This norm is minimisable if it is attained. Provided the column rank of A is full, R will be invertible (Exercise: why is this?). Thus we have the solution

$$\mathbf{x} = R^{-1} Q^\top \mathbf{b}$$

Example (quadratic fit) Suppose we want to fit noisy data by a quadratic

$$p(x) = p_0 + p_1 x + p_2 x^2$$

That is, we want to choose p_0, p_1, p_2 at data samples $x_1, \ldots, x_m$ so that the following is true:

$$p_0 + p_1 x_k + p_2 x_k^2 \approx f_k$$

where f_k are given by data. We can reinterpret this as a least squares problem: minimise the norm

$$\left\| \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} - \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \right\|$$

We can solve this using the QR decomposition:

In [ ]:
```
m,n = 100,3

x = range(0,1; length=m) # 100 points
f = 2 .+ x .+ 2x.^2 .+ 0.1 .* randn.() # Noisy quadratic

A = x .^ (0:2)'  # 100 x 3 matrix, equivalent to [ones(m) x x.^2]
Q,R̂ = qr(A)
Q̂ = Q[:,1:n] # Q represents full orthogonal matrix so we take first 3 colum

p₀,p₁,p₂ = R̂ \ Q̂'f
```

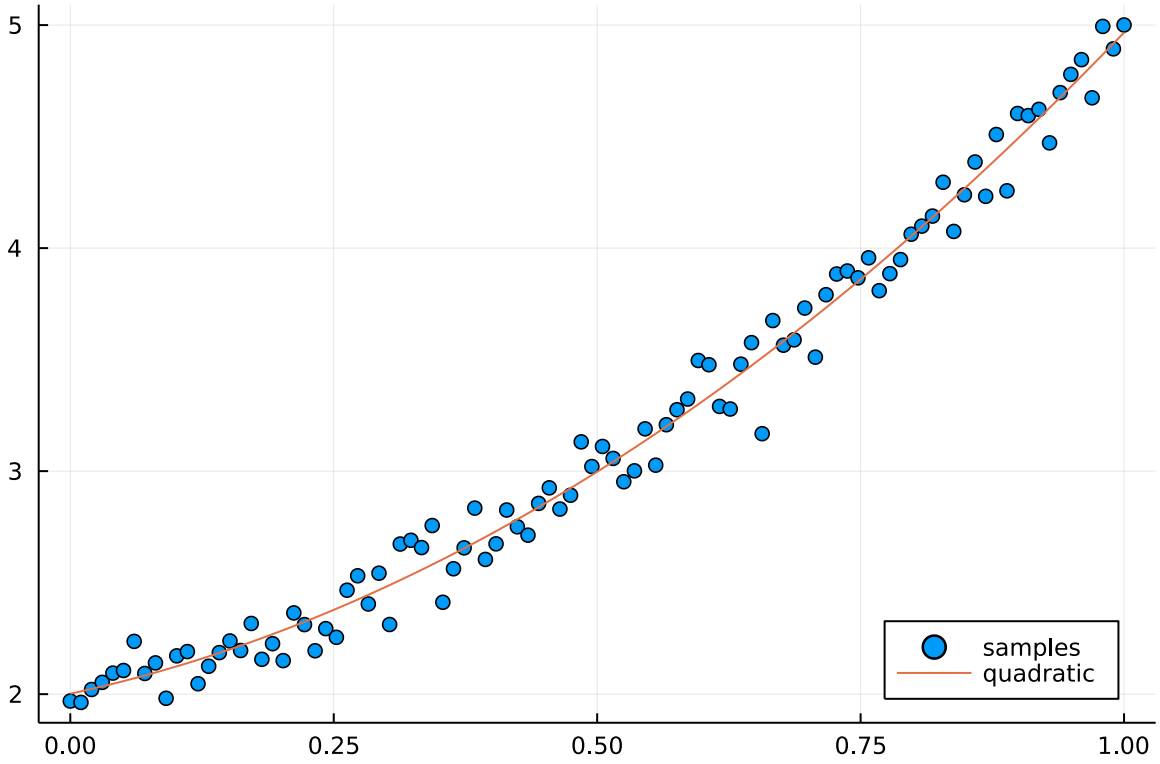
Out[ ]:
```
3-element Vector{Float64}:
 2.0020823730946438
 1.0164297831971307
 1.9480305651560517
```

We can visualise the fit:

In [ ]:
```
p = x -> p₀ + p₁*x + p₂*x^2

scatter(x, f; label="samples", legend=:bottomright)
plot!(x, p.(x); label="quadratic")
```

Out[ ]:



Note that `\` with a rectangular system does least squares by default:

In [ ]:
```
A \ f
```

Out[ ]:
```
3-element Vector{Float64}:
 2.002082373094645
 1.0164297831971287
 1.9480305651560528
```

2. Reduced QR and Gram–Schmidt

How do we compute the QR decomposition? We begin with a method you may have seen before in another guise. Write

$$A = \begin{bmatrix} \mathbf{a}_1 | \dots | \mathbf{a}_n \end{bmatrix}$$

where $\mathbf{a}_k \in \mathbb{R}^m$ and assume they are linearly independent (A has full column rank). Note that the column span of the first j columns of A will be the same as the first j columns of Q, as R must be non-singular:

$$\text{span}(\mathbf{a}_1, \dots, \mathbf{a}_j) = \text{span}(\mathbf{q}_1, \dots, \mathbf{q}_j)$$

In other words: the columns of Q are an orthogonal basis of the column span of A. To see this note that since R̂ is triangular we have

$$\begin{bmatrix} \mathbf{a}_1 | \dots | \mathbf{a}_j \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 | \dots | \mathbf{q}_j \end{bmatrix} R[1:j, 1:j]$$

for all j. That is, if $\mathbf{v} \in \text{span}(\mathbf{a}_1, \dots, \mathbf{a}_j)$ we have for $\mathbf{c} \in \mathbb{R}^j$

$$\mathbf{v} = \begin{bmatrix} \mathbf{a}_1 | \dots | \mathbf{a}_j \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{q}_1 | \dots | \mathbf{q}_j \end{bmatrix} R[1:j, 1:j]\mathbf{c} \in \text{span}(\mathbf{q}_1, \dots, \mathbf{q}_j)$$

while if $\mathbf{w} \in \text{span}(\mathbf{q}_1, \dots, \mathbf{q}_j)$ we have for $\mathbf{d} \in \mathbb{R}^j$

$$\mathbf{w} = \begin{bmatrix} \mathbf{q}_1 | \cdots | \mathbf{q}_j \end{bmatrix} \mathbf{d} = \begin{bmatrix} \mathbf{a}_1 | \cdots | \mathbf{a}_j \end{bmatrix} R[1:j, 1:j]^{-1} \mathbf{d} \in \operatorname{span}(\mathbf{a}_1, \ldots, \mathbf{a}_j).$$

It is possible to find an orthogonal basis using the *Gram–Schmidt algorithm*, We construct it via induction: assume that

$$\operatorname{span}(\mathbf{a}_1, \ldots, \mathbf{a}_{j-1}) = \operatorname{span}(\mathbf{q}_1, \ldots, \mathbf{q}_{j-1})$$

where $\mathbf{q}_1, \ldots, \mathbf{q}_{j-1}$ are orthogonal:

$$\mathbf{q}_k^\top \mathbf{q}_\ell = \delta_{k\ell} = \begin{cases} 1 & k = \ell \\ 0 & \text{otherwise} \end{cases}.$$

for $k, \ell < j$. Define

$$\mathbf{v}_j := \mathbf{a}_j - \sum_{k=1}^{j-1} \underbrace{\mathbf{q}_k^\top \mathbf{a}_j}_{r_{kj}} \mathbf{q}_k$$

so that for $k < j$

$$\mathbf{q}_k^\top \mathbf{v}_j = \mathbf{q}_k^\top \mathbf{a}_j - \sum_{k=1}^{j-1} \underbrace{\mathbf{q}_k^\top \mathbf{a}_j}_{r_{kj}} \mathbf{q}_k^\top \mathbf{q}_k = 0.$$

Then we define

$$\mathbf{q}_j := \frac{\mathbf{v}_j}{\|\mathbf{v}_j\|}.$$

which sastisfies $\mathbf{q}_k^\top \mathbf{q}_j = \delta_{kj}$ for $k \leq j$.

We now reinterpret this construction as a reduced QR decomposition. Define $r_{jj} := \|\mathbf{v}_j\|$ Then rearrange the definition we have

$$\mathbf{a}_j = \begin{bmatrix} \mathbf{q}_1 | \cdots | \mathbf{q}_j \end{bmatrix} \begin{bmatrix} r_{1j} \\ \vdots \\ r_{jj} \end{bmatrix}$$

Thus

$$\begin{bmatrix} \mathbf{a}_1 | \cdots | \mathbf{a}_j \end{bmatrix} \begin{bmatrix} r_{11} & \cdots & r_{1j} \\ & \ddots & \vdots \\ & & r_{jj} \end{bmatrix}$$

That is, we are computing the reduced QR decomposition column-by-column. Running this algorithm to $j = n$ completes the decomposition.

Gram–Schmidt in action

We are going to compute the reduced QR of a random matrix

In [ ]:
```
m,n = 5,4
```

```
A = randn(m,n)
Q,R̂ = qr(A)
Q̂ = Q[:,1:n]
```

Out[ ]:
```
5×4 Matrix{Float64}:
 -0.640643   -0.0823157  -0.22609     0.728861
 -0.0996754  -0.285725    0.575686    0.0805714
  0.490733   -0.793914   -0.166541    0.285984
 -0.580068   -0.530002   -0.0540479  -0.591485
 -0.0484121  -0.0195648   0.766036    0.175009
```

The first column of $\hat{Q}$ is indeed a normalised first column of A:

In [ ]:
```
R = zeros(n,n)
Q = zeros(m,n)
R[1,1] = norm(A[:,1])
Q[:,1] = A[:,1]/R[1,1]
```

Out[ ]:
```
5-element Vector{Float64}:
 -0.6406428105326175
 -0.09967542686075603
  0.49073256021653416
 -0.5800684602848314
 -0.04841212987574009
```

We now determine the next entries as

In [ ]:
```
R[1,2] = Q[:,1]'A[:,2]
v = A[:,2] - Q[:,1]*R[1,2]
R[2,2] = norm(v)
Q[:,2] = v/R[2,2]
```

Out[ ]:
```
5-element Vector{Float64}:
 -0.08231573808619484
 -0.2857251402909879
 -0.7939143184765172
 -0.5300023946604269
 -0.01956476529889328
```

And the third column is then:

In [ ]:
```
R[1,3] = Q[:,1]'A[:,3]
R[2,3] = Q[:,2]'A[:,3]
v = A[:,3] - Q[:,1:2]*R[1:2,3]
R[3,3] = norm(v)
Q[:,3] = v/R[3,3]
```

Out[ ]:
```
5-element Vector{Float64}:
  0.22608968882009056
 -0.575686495796397
  0.16654081939416812
  0.05404794982633636
 -0.7660362170041234
```

(Note the signs may not necessarily match.)

We can clean this up as a simple algorithm:

In [ ]:
```
function gramschmidt(A)
    m,n = size(A)
    m ≥ n || error("Not supported")
    R = zeros(n,n)
    Q = zeros(m,n)
```

```
    for j = 1:n
        for k = 1:j-1
            R[k,j] = Q[:,k]'*A[:,j]
        end
        v = A[:,j] - Q[:,1:j-1]*R[1:j-1,j]
        R[j,j] = norm(v)
        Q[:,j] = v/R[j,j]
    end
    Q,R
end

Q,R = gramschmidt(A)
norm(A - Q*R)
```

Out[ ]:
```
2.4873781269328544e-16
```

Complexity and stability

We see within the `for j = 1:n` loop that we have $O(mj)$ operations. Thus the total complexity is $O(mn^2)$ operations.

Unfortunately, the Gram–Schmidt algorithm is *unstable*: the rounding errors when implemented in floating point accumulate in a way that we lose orthogonality:

In [ ]:
```
A = randn(300,300)
Q,R = gramschmidt(A)
norm(Q'Q-I)
```

Out[ ]:
```
3.5531891494246798e-12
```

3. Householder reflections and QR

As an alternative, we will consider using Householder reflections to introduce zeros below the diagonal. Thus, if Gram–Schmidt is a process of *triangular orthogonalisation* (using triangular matrices to orthogonalise), Householder reflections is a process of *orthogonal triangularisation* (using orthogonal matrices to triangularise).

Consider multiplication by the Householder reflection corresponding to the first column, that is, for

$$Q_1 := Q^{\mathrm{H}}_{\mathbf{a}_1},$$

consider

$$Q_1 A = \begin{bmatrix} \times & \times & \cdots & \times \\ & \times & \cdots & \times \\ & \vdots & \ddots & \vdots \\ & \times & \cdots & \times \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & \mathbf{a}^1_2 & \cdots & \mathbf{a}^1_n \end{bmatrix}$$

where

$$r_{1j} := (Q_1 \mathbf{a}_j)[1] \qquad \text{and} \qquad \mathbf{a}^1_j := (Q_1 \mathbf{a}_j)[2:m],$$

noting $r_{11} = -\text{sign}(a_{11})\|\mathbf{a}_1\|$ and all entries of $\mathbf{a}_1^1$ are zero (thus not included). That is, we have made the first column triangular.

But now consider

$$Q_2 := \begin{bmatrix} 1 & \\ & Q^{\mathrm{H}}_{\mathbf{a}_2^1} \end{bmatrix} = Q^H_{\begin{bmatrix} 0 \\ \mathbf{a}_2^1 \end{bmatrix}}$$

so that

$$Q_2 Q_1 A = \begin{bmatrix} \times & \times & \times & \cdots & \times \\ & \times & \times & \cdots & \times \\ & & \vdots & \ddots & \vdots \\ & & \times & \cdots & \times \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ & r_{22} & r_{23} & \cdots & r_{2n} \\ & & \mathbf{a}_3^2 & \cdots & \mathbf{a}_n^2 \end{bmatrix}$$

where

$$r_{2j} := (Q_2 \mathbf{a}_j^1)[1] \qquad \text{and} \qquad \mathbf{a}_j^2 := (Q_2 \mathbf{a}_j^1)[2 : m-1]$$

Thus the first two columns are triangular.

The inductive construction is thus clear. If we define $\mathbf{a}_j^0 := \mathbf{a}_j$ we have the construction

$$\begin{aligned} Q_j &:= \begin{bmatrix} I_{j-1 \times j-1} & \\ & Q^{\pm,\mathrm{H}}_{\mathbf{a}_j^j} \end{bmatrix} \\ \mathbf{a}_j^k &:= (Q_k \mathbf{a}_j^{k-1})[2 : m-k+1] \\ r_{kj} &:= (Q_k \mathbf{a}_j^{k-1})[1] \end{aligned}$$

Then

$$Q_n \cdots Q_1 A = \underbrace{\begin{bmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \\ & & 0 \\ & & \vdots \\ & & 0 \end{bmatrix}}_{R}$$

i.e.

$$A = \underbrace{Q_n \cdots Q_1}_{Q} R.$$

The implementation is cleaner. We do a naive implementation here:

In [ ]:
```
function householderreflection(x)
    y = copy(x)
    # we cannot use sign(x[1]) in case x[1] == 0
    y[1] += (x[1] ≥ 0 ? 1 : -1)*norm(x)
```

```
    w = y/norm(y)
    I - 2*w*w'
end
function householderqr(A)
    m,n = size(A)
    R = copy(A)
    Q = Matrix(1.0I, m, m)
    for j = 1:n
        Qⱼ = householderreflection(R[j:end,j])
        R[j:end,:] = Qⱼ*R[j:end,:]
        Q[:,j:end] = Q[:,j:end]*Qⱼ
    end
    Q,R
end

m,n = 7,5
A = randn(m, n)
Q,R = householderqr(A)
Q*R ≈ A
```

Out[ ]: true

Note because we are forming a full matrix representation of each Householder reflection this is a slow algorithm, taking $O(n^4)$ operations. The problem sheet will consider a better implementation that takes $O(n^3)$ operations.

Example We will now do an example by hand. Consider the 4×3 matrix

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 15 & 18 \\ -2 & -4 & -4 \\ -2 & -4 & -10 \end{bmatrix}$$

For the first column we have

$$Q_1 = I - \frac{1}{12}\begin{bmatrix} 4 \\ 0 \\ -2 \\ -2 \end{bmatrix}\begin{bmatrix} 4 & 0 & -2 & -2 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} -1 & 0 & 2 & 2 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & 2 & -1 \\ 2 & 0 & -1 & 2 \end{bmatrix}$$

so that

$$Q_1 A = \begin{bmatrix} -3 & -6 & -9 \\ & 15 & 18 \\ & 0 & 0 \\ & 0 & -6 \end{bmatrix}$$

In this example the next column is already upper-triangular, but because of our choice of reflection we will end up swapping the sign, that is

$$Q_2 = \begin{bmatrix} 1 & & & \\ & -1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

so that

$$Q_2Q_1A = \begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & 0 & 0 \\ & 0 & -6 \end{bmatrix}$$

The final reflection is

$$Q_3 = \begin{bmatrix} I_{2\times 2} & \\ & I - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{å}1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix}$$

giving us

$$Q_3Q_2Q_1A = \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R}$$

That is,

$$A = Q_1Q_2Q_3R = \underbrace{\frac{1}{3}\begin{bmatrix} -1 & 0 & 2 & 2 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & -1 & 2 \\ 2 & 0 & 2 & -1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R} = \underbrace{\frac{1}{3}\begin{bmatrix} -1 & 0 & 2 \\ 0 & 3 & 0 \\ 2 & 0 & -1 \\ 2 & 0 & 2 \end{bmatrix}}_{Q} \widehat{}$$

2. PLU Decomposition

Just as Gram–Schmidt can be reinterpreted as a reduced QR decomposition, Gaussian elimination with pivoting can be interpreted as a PLU decomposition.

Special "one-column" lower triangular matrices

Consider the following set of $n \times n$ lower triangular matrices which equals identity apart from one-column:

$$\mathcal{L}_j := \left\{ I + \begin{bmatrix} \mathbf{0}_j \\ \mathbf{l}_j \end{bmatrix} \mathbf{e}_j^\top : \mathbf{l}_j \in \mathbb{R}^{n-j} \right\}$$

where $\mathbf{0}_j$ denotes the zero vector of length j. That is, if $L_j \in \mathcal{L}_j$ then it is equal to the identity matrix apart from in the j th column:

$$L_j = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \ell_{j+1,j} & 1 & \\ & & \vdots & & \ddots \\ & & \ell_{n,j} & & 1 \end{bmatrix} =$$

These satisify the following special properties:

Proposition (one-column lower triangular inverse) If $L_j \in \mathcal{L}_j$ then

$$L_j^{-1} = I - \begin{bmatrix} \mathbf{0}_j \\ \mathbf{l}_j \end{bmatrix} \mathbf{e}_j^\top = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{j+1,j} & 1 & \\ & & \vdots & & \ddots \\ & & -\ell_{n,j} & & 1 \end{bmatrix} \in \mathcal{L}_j.$$

Proposition (one-column lower triangular multiplication) If $L_j \in \mathcal{L}_j$ and $L_k \in \mathcal{L}_k$ for $k \geq j$ then

$$L_j L_k = I + \begin{bmatrix} \mathbf{0}_j \\ \mathbf{l}_j \end{bmatrix} \mathbf{e}_j^\top + \begin{bmatrix} \mathbf{0}_k \\ \mathbf{l}_k \end{bmatrix} \mathbf{e}_k^\top$$

Lemma (one-column lower triangular with pivoting) If σ is a permutation that leaves the first j rows fixed (that is, $\sigma_\ell = \ell$ for $\ell \leq j$) and $L_j \in \mathcal{L}_j$ then

$$P_\sigma L_j = \tilde{L}_j P_\sigma$$

where $\tilde{L}_j \in \mathcal{L}_j$.

Proof Write

$$P_\sigma = \begin{bmatrix} I_j & \\ & P_\tau \end{bmatrix}$$

where τ is the permutation with Cauchy notation

$$\begin{pmatrix} 1 & \cdots & n-j \\ \sigma_{j+1} - j & \cdots & \sigma_n - j \end{pmatrix}$$

Then we have

$$P_\sigma L_j = P_\sigma + \begin{bmatrix} \mathbf{0}_j \\ P_\tau \mathbf{l}_j \end{bmatrix} \mathbf{e}_j^\top = \underbrace{(I + \begin{bmatrix} \mathbf{0}_j \\ P_\tau \mathbf{l}_j \end{bmatrix} \mathbf{e}_j^\top)}_{\tilde{L}_j} P_\sigma$$

noting that $\mathbf{e}_j^\top P_\sigma = \mathbf{e}_j^\top$ (as $\sigma_j = j$). ∎

LU Decomposition

Before discussing pivoting, consider standard Gaussian elimation where one row-reduces to introduce zeros column-by-column. We will mimick the computation of the QR decomposition to view this as a *triangular triangularisation*.

Consider the matrix

$$L_1 = \begin{bmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & & 1 \end{bmatrix} = I - \begin{bmatrix} 0 \\ \frac{\mathbf{a}_1[2:n]}{\mathbf{a}_1[1]} \end{bmatrix} \mathbf{e}_1^\top.$$

We then have

$$L_1 A = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & \mathbf{a}_2^1 & \cdots & \mathbf{a}_n^1 \end{bmatrix}$$

where $\mathbf{a}_j^1 := (L_1\mathbf{a}_j)[2:n]$ and $u_{1j} = a_{1j}$. But now consider

$$L_2 := I - \begin{bmatrix} 0 \\ \frac{\mathbf{a}_2^1[2:n-1]}{\mathbf{a}_2^1[1]} \end{bmatrix} \mathbf{e}_1^\top.$$

Then

$$L_2 L_1 A = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & \mathbf{a}_3^2 & \cdots & \mathbf{a}_n^2 \end{bmatrix}$$

where

$$r_{2j} := (\mathbf{a}_j^1)[1] \qquad \text{and} \qquad \mathbf{a}_j^2 := (L_2\mathbf{a}_j^1)[2:n-1]$$

Thus the first two columns are triangular.

The inductive construction is again clear. If we define $\mathbf{a}_j^0 := \mathbf{a}_j$ we have the construction

$$\begin{aligned} L_j &:= I + \begin{bmatrix} \mathbf{0}_j \\ \frac{\mathbf{a}_{j+1}^j[2:n-j]}{\mathbf{a}_{j+1}^j[1]} \end{bmatrix} \mathbf{e}_j^\top \\ \mathbf{a}_j^k &:= (L_k\mathbf{a}_j^{k-1})[2:n-k+1] \\ u_{kj} &:= (L_k\mathbf{a}_j^{k-1})[1] \end{aligned}$$

Then

$$L_{n-1}\cdots L_1 A = \underbrace{\begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ & & u_{nn} \end{bmatrix}}_{U}$$

i.e.

$$A = \underbrace{L_{n-1}^{-1} \cdots L_1^{-1}}_{L} U.$$

Writing

$$L_j = I + \begin{bmatrix} \mathbf{0}_j \\ \ell_{j+1,j} \\ \vdots \\ \ell_{n,j} \end{bmatrix} \mathbf{e}_j^\top$$

and using the properties of inversion and multiplication we therefore deduce

$$L = \begin{bmatrix} 1 & & & & \\ -\ell_{21} & 1 & & & \\ -\ell_{31} & -\ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ -\ell_{n1} & -\ell_{n2} & \cdots & -\ell_{n,n-1} & 1 \end{bmatrix}$$

Example (computer) We will do a numerical example (by-hand is equivalent to Gaussian elimination). The first lower triangular matrix is:

In [ ]:
```
n = 4
A = randn(n,n)
L₁ = I -[0; A[2:end,1]/A[1,1]] * [1 zeros(1,n-1)]
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.0        -0.0  -0.0  -0.0
  1.15995     1.0  -0.0  -0.0
 -0.0724294  -0.0   1.0  -0.0
  0.400929   -0.0  -0.0   1.0
```

Which indeed introduces zeros in the first column:

In [ ]:
```
A₁ = L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.16946      -1.77156     1.27168  -0.519889
  0.0          -0.528188    1.18052  -0.312403
  0.0          -0.00625178  1.14613  -0.195066
 -5.55112e-17  -0.57422     1.64806  -0.917347
```

Now we make the next lower triangular operator:

In [ ]:
```
L₂ = I - [0; 0; A₁[3:end,2]/A₁[2,2]] * [0 1 zeros(1,n-2)]
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.0  -0.0         -0.0  -0.0
 -0.0   1.0         -0.0  -0.0
 -0.0  -0.0118363    1.0  -0.0
 -0.0  -1.08715     -0.0   1.0
```

So that

In [ ]:

```
A₂ = L₂*L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.16946      -1.77156      1.27168   -0.519889
  0.0          -0.528188     1.18052   -0.312403
  0.0           0.0          1.13216   -0.191368
 -5.55112e-17  -5.55112e-17  0.364656  -0.577719
```

The last one is:

In [ ]:
```
L₃ = I - [0; 0; 0; A₂[4:end,3]/A₂[3,3]] * [0 0 1 zeros(1,n-3)]
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.0   -0.0   -0.0        -0.0
 -0.0    1.0   -0.0        -0.0
 -0.0   -0.0    1.0        -0.0
 -0.0   -0.0   -0.322089    1.0
```

Giving us

In [ ]:
```
U = L₃*L₂*L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.16946      -1.77156      1.27168      -0.519889
  0.0          -0.528188     1.18052      -0.312403
  0.0           0.0          1.13216      -0.191368
 -5.55112e-17  -1.11022e-16  2.22045e-16  -0.516081
```

and

In [ ]:
```
L = inv(L₁)*inv(L₂)*inv(L₃)
```

Out[ ]:
```
4×4 Matrix{Float64}:
  1.0        0.0        0.0       0.0
 -1.15995    1.0        0.0       0.0
  0.0724294  0.0118363  1.0       0.0
 -0.400929   1.08715    0.322089  1.0
```

Note how the entries of L are indeed identical to the negative lower entries of L_1, L_2 and L_3.

Example (by-hand)

Consider the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 1 & 4 & 9 \end{bmatrix}$$

We have

$$L_2 L_1 A = L_2 \begin{bmatrix} 1 & & \\ -2 & 1 & \\ -1 & & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 1 & 4 & 9 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & -\frac{3}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ & 2 & 6 \\ & 3 & 8 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ & 2 & 6 \\ & & -1 \end{bmatrix}}_{U}$$

We then deduce L by taking the negative of the lower entries of L_1, L_2:

$$L = \begin{bmatrix} 1 & & \\ 2 & 1 & \\ 1 & \frac{3}{2} & 1 \end{bmatrix}.$$

PLU Decomposition

We learned in first year linear algebra that if a diagonal entry is zero when doing Gaussian elimnation one has to *row pivot*. For stability, in implementation one *always* pivots: swap the largest in magnitude entry for the entry on the diagonal. In terms of a decomposition, this leads to

$$L_{n-1}P_{n-1}\cdots P_2L_1P_1A = U$$

where P_j is a permutation that leaves rows 1 through $j-1$ fixed, and swaps row j with a row $k \geq j$ whose entry is maximal in absolute value.

Thus we can deduce that

$$L_{n-1}P_{n-1}\cdots P_2L_1P_1 = \underbrace{L_{n-1}\tilde{L}_{n-2}\cdots\tilde{L}_1}_{L^{-1}}\underbrace{P_{n-1}\cdots P_2P_1}_{P}.$$

where the tilde denotes the combined actions of swapping the permutation and lower-triangular matrices, that is,

$$P_{n-1}\cdots P_{j+1}L_j = \tilde{L}_jP_{n-1}\cdots P_{j+1}.$$

where $\tilde{L}_j \in \mathcal{L}_j$. The entries of L are then again the negative of the entries below the diagonal of $L_{n-1}, \tilde{L}_{n-2}, \ldots, \tilde{L}_1$.

Writing

$$\tilde{L}_j = I + \begin{bmatrix} \mathbf{0}_j \\ \tilde{\ell}_{j+1,j} \\ \vdots \\ \tilde{\ell}_{n,j} \end{bmatrix} \mathbf{e}_j^\top$$

and using the properties of inversion and multiplication we therefore deduce

$$L = \begin{bmatrix} 1 & & & & & \\ -\tilde{\ell}_{21} & 1 & & & & \\ -\tilde{\ell}_{31} & -\tilde{\ell}_{32} & 1 & & & \\ \vdots & \vdots & \ddots & \ddots & & \\ -\tilde{\ell}_{n-1,1} & -\tilde{\ell}_{n-1,2} & \cdots & -\tilde{\ell}_{n-1,n-2} & 1 & \\ -\tilde{\ell}_{n1} & -\tilde{\ell}_{n2} & \cdots & -\tilde{\ell}_{n,n-2} & -\ell_{n,n-1} & 1 \end{bmatrix}$$

Example

Again we consider the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 1 & 4 & 9 \end{bmatrix}$$

Even though $a_{11} = 1 \neq 0$, we still pivot: placing the maximum entry on the diagonal to mitigate numerical errors. That is, we first pivot and upper triangularise the first column:

$$L_1 P_1 A = L_1 \begin{bmatrix} 0 & 1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 1 & 4 & 9 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{1}{2} & & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 8 \\ 1 & 1 & 1 \\ 1 & 4 & 9 \end{bmatrix}$$

We now pivot and upper triangularise the second column:

$$L_2 P_2 L_1 P_1 A = L_2 \begin{bmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & 8 \\ 0 & -1 & -3 \\ 0 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 8 \\ 0 & 2 & 5 \\ 0 & -1 & -3 \end{bmatrix} = \underbrace{\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}}$$

We now move P_2 to the right:

$$L_2 P_2 L_1 P_1 = \underbrace{\begin{bmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}}_{L_2 \tilde{L}_1} \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}_{P}$$

That is

$$L = \begin{bmatrix} 1 & & \\ \frac{1}{2} & 1 & \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$$

We see how this example is done on a computer:

In [ ]:
```
A = [1 1 1;
     2 4 8;
     1 4 9]
L,U,σ = lu(A) # σ is a vector encoding the permutation
```

Out[ ]:
```
LU{Float64, Matrix{Float64}}
L factor:
3×3 Matrix{Float64}:
 1.0   0.0  0.0
 0.5   1.0  0.0
 0.5  -0.5  1.0
U factor:
3×3 Matrix{Float64}:
 2.0  4.0   8.0
 0.0  2.0   5.0
 0.0  0.0  -0.5
```

The permutation is

In [ ]:
```
σ
```

Out[ ]:
```
3-element Vector{Int64}:
 2
 3
 1
```

Thus to invert a system we can do:

In [ ]:
```
b = randn(3)
U\(L\b[σ]) == A\b
```

Out[ ]:
```
true
```

Note the entries match exactly because this precisely what `\` is using.

3. Cholesky Decomposition

Cholesky Decomposition is a form of Gaussian elimination (without pivoting) that exploits symmetry in the problem, resulting in a substantial speedup. It is only relevant for *symmetric positive definite* matrices.

Definition (positive definite) A square matrix $A \in \mathbb{R}^{n\times n}$ is *positive definite* if for all $\mathbf{x} \in \mathbb{R}^n$ we have

$$\mathbf{x}^\top A\mathbf{x} > 0$$

First we establish some basic properties of positive definite matrices:

Proposition (conj. pos. def.) If $A \in \mathbb{R}^{n\times n}$ is positive definite and $V \in \mathbb{R}^{n\times n}$ is non-singular then

$$V^\top AV$$

is positive definite.

Proposition (diag positivity) If $A \in \mathbb{R}^{n\times n}$ is positive definite then its diagonal entries are positive: $a_{kk} > 0$.

Theorem (subslice pos. def.) If $A \in \mathbb{R}^{n\times n}$ is positive definite and $\mathbf{k} \in \{1, \ldots, n\}^n$ where any integer appears only once then $A[\mathbf{k}, \mathbf{k}]$ is also positive definite.

We leave the proofs to the problem sheets. Here is the key result:

Theorem (Cholesky and sym. pos. def.) A matrix A is symmetric positive definite if and only if it has a Cholesky decomposition

$$A = LL^\top$$

where the diagonals of L are positive.

Proof If A has a Cholesky decomposition it is symmetric ($A^\top = (LL^\top)^\top = A$) and for $\mathbf{x} \neq 0$ we have

$$\mathbf{x}^\top A\mathbf{x} = (L\mathbf{x})^\top L\mathbf{x} = \|L\mathbf{x}\|^2 > 0$$

where we use the fact that L is non-singular.

For the other direction we will prove it by induction, with the 1×1 case being trivial. Write

$$A = \begin{bmatrix} \alpha & \mathbf{v}^\top \\ \mathbf{v} & K \end{bmatrix} = \underbrace{\begin{bmatrix} \sqrt{\alpha} & \\ \frac{\mathbf{v}}{\sqrt{\alpha}} & I \end{bmatrix}}_{L_1} \underbrace{\begin{bmatrix} 1 & \\ & K - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha} \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} \sqrt{\alpha} & \frac{\mathbf{v}^\top}{\sqrt{\alpha}} \\ & I \end{bmatrix}}_{L_1^\top}.$$

Note that $K - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha}$ is a subslice of $A_1 = L_1^{-1} A L_1^{-\top}$, hence by the previous propositions is itself symmetric positive definite. Thus we can write

$$K - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha} = \tilde{L}\tilde{L}^\top$$

and hence $A = LL^\top$ for

$$L = L_1 \begin{bmatrix} 1 & \\ & \tilde{L} \end{bmatrix}.$$

satisfies $A = LL^\top$. ∎

Note hidden in this proof is a simple algorithm form computing the Cholesky decomposition. We define

$$\begin{aligned} A_1 &:= A \\ \mathbf{v}_k &:= A_k[2 : n-k+1, 1] \\ \alpha_k &:= A_k[1,1] \\ A_{k+1} &:= A_k[2 : n-k+1, 2 : n-k+1] - \frac{\mathbf{v}_k \mathbf{v}_k^\top}{\alpha_k}. \end{aligned}$$

Then

$$L = \begin{bmatrix} \sqrt{\alpha_1} & & & & \\ \frac{\mathbf{v}_1[1]}{\sqrt{\alpha_1}} & \sqrt{\alpha_2} & & & \\ \frac{\mathbf{v}_1[2]}{\sqrt{\alpha_1}} & \frac{\mathbf{v}_2[1]}{\sqrt{\alpha_2}} & \sqrt{\alpha_2} & & \\ \vdots & \vdots & \ddots & \ddots & \\ \frac{\mathbf{v}_1[n-1]}{\sqrt{\alpha_1}} & \frac{\mathbf{v}_2[n-2]}{\sqrt{\alpha_2}} & \cdots & \frac{\mathbf{v}_{n-1}[1]}{\sqrt{\alpha_{n-1}}} & \sqrt{\alpha_n} \end{bmatrix}$$

This algorithm succeeds if and only if A is symmetric positive definite.

Example Consider the matrix

$$A_0 = A = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

Then

$$A_1 = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix}$$

Continuing, we have

$$A_2 = \frac{1}{2}\left(\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix}\right) = \frac{1}{3}\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}$$

Finally

$$A_3 = \frac{5}{4}.$$

Thus we get

$$L = L_1L_2L_3 = \begin{bmatrix} \sqrt{2} & & & \\ \frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2} & & \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{3}} & \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{12}} & \frac{\sqrt{5}}{2} \end{bmatrix}$$

4. Timings

The different decompositions have trade-offs between speed and stability. First we compare the speed of the different decompositions on a symmetric positive definite matrix, from fastest to slowest:

In [ ]:
```
n = 100
A = Symmetric(rand(n,n)) + 100I # shift by 10 ensures positivity
@btime cholesky(A);
@btime lu(A);
@btime qr(A);
```

```
  28.524 μs (5 allocations: 78.25 KiB)
  55.173 μs (4 allocations: 79.08 KiB)
  221.079 μs (7 allocations: 134.55 KiB)
```

On my machine, `cholesky` is ~1.5x faster than `lu`, which is ~2x faster than QR.

In terms of stability, QR computed with Householder reflections (and Cholesky for positive definite matrices) are stable, whereas LU is usually unstable (unless the matrix is diagonally dominant). PLU is a very complicated story: in theory it is unstable, but the set of matrices for which it is unstable is extremely small, so small one does not normally run into them.

Here is an example matrix that is in this set.

In [ ]:
```
function badmatrix(n)
    A = Matrix(1I, n, n)
    A[:,end] .= 1
```

```
    for j = 1:n-1
        A[j+1:end,j] .= -1
    end
    A
end
A = badmatrix(5)
```

Out[ ]:
```
5×5 Matrix{Int64}:
  1   0   0   0  1
 -1   1   0   0  1
 -1  -1   1   0  1
 -1  -1  -1   1  1
 -1  -1  -1  -1  1
```

Note that pivoting will not occur (we do not pivot as the entries below the diagonal are the same magnitude as the diagonal), thus the PLU Decomposition is equivalent to an LU decomposition:

In [ ]:
```
L,U = lu(A)
```

Out[ ]:
```
LU{Float64, Matrix{Float64}}
L factor:
5×5 Matrix{Float64}:
  1.0   0.0   0.0   0.0  0.0
 -1.0   1.0   0.0   0.0  0.0
 -1.0  -1.0   1.0   0.0  0.0
 -1.0  -1.0  -1.0   1.0  0.0
 -1.0  -1.0  -1.0  -1.0  1.0
U factor:
5×5 Matrix{Float64}:
 1.0  0.0  0.0  0.0   1.0
 0.0  1.0  0.0  0.0   2.0
 0.0  0.0  1.0  0.0   4.0
 0.0  0.0  0.0  1.0   8.0
 0.0  0.0  0.0  0.0  16.0
```

But here we see an issue: the last column of `U` is growing exponentially fast! Thus when `n` is large we get very large errors:

In [ ]:
```
n = 100
b = randn(n)
A = badmatrix(n)
norm(A\b - qr(A)\b) # A \ b still uses lu
```

Out[ ]:
```
1182.3234752106287
```

Note `qr` is completely fine:

In [ ]:
```
norm(qr(A)\b - qr(big.(A)) \b) # roughly machine precision
```

Out[ ]:
```
1.5641814394355523778414968638287123123287546522479929403433698861913406425077706e-14
```

Amazingly, PLU is fine if applied to a small perturbation of `A`:

In [ ]:
```
ε = 0.000001
Aε = A .+ ε .* randn.()
norm(Aε \ b - qr(Aε) \ b) # Now it matches!
```

Out[ ]:
```
8.78477899875885e-15
```

The big *open problem* in numerical linear algebra is to prove that the set of matrices for which PLU fails has extremely small measure.