Decompositions and least squares

In this lecture, we look at several factorizations of a matrix. 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 ($Q^\top Q = I$, $\mathbf{q}_j \in \mathbb{R}^m$)

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:

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

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

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

1. QR, Reduced 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:

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

Relationship with least squares

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 find noisy data by a quadratic

$$p(x) = a + bx + cx^2$$

That is, we want to choose a, b, c at data samples $x_1, \ldots, x_m$ so that the following is true:

$$a + bx_k + cx_k^2 \approx f_k$$

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

$$\left\| \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} - \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \right\|$$

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

a,b,c = R̂ \ Q̂'f
```

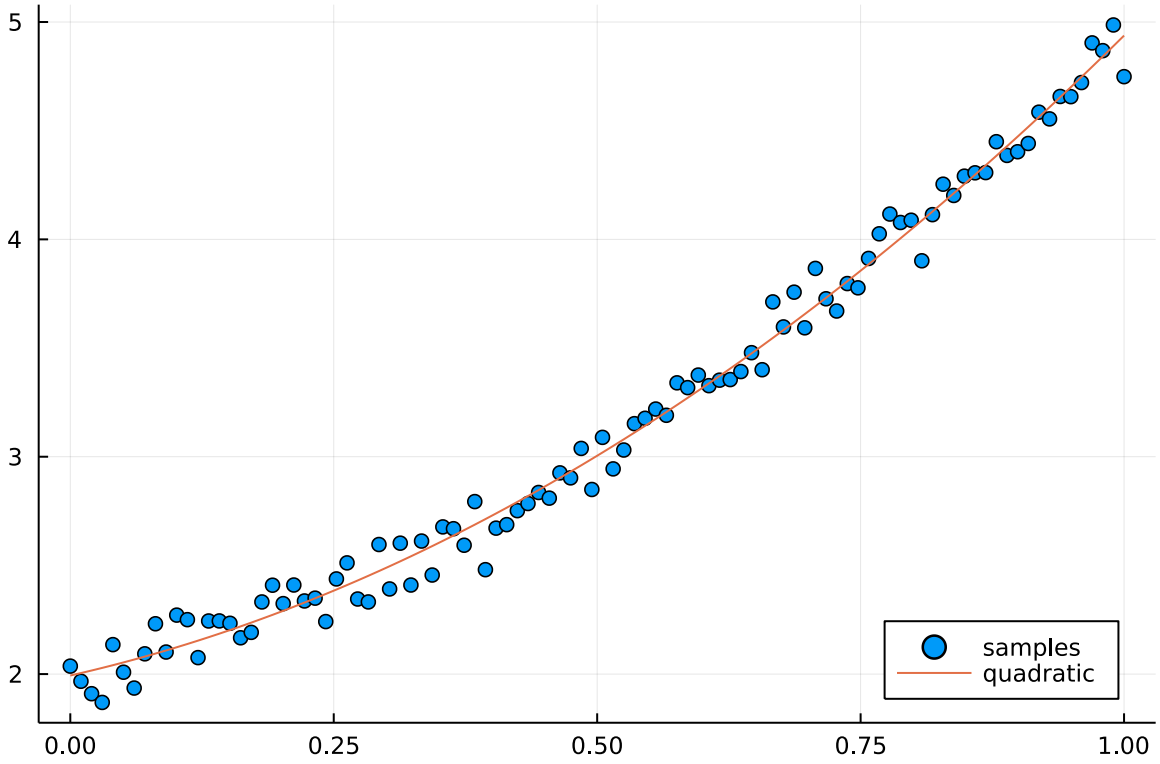
Out[ ]:
```
3-element Vector{Float64}:
 1.993405537560798
 1.1005121973366017
 1.8435561578617732
```

We can visualise the fit:

In [ ]:
```
p = x -> a + b*x + c*x^2

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

Out[ ]:



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

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

Out[ ]:
```
3-element Vector{Float64}:
 1.9934055375607993
 1.100512197336599
 1.8435561578617752
```

Gram–Schmidt and reduced QR

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 $\mathbf{a}_k \in \mathbb{R}^m$. Note that the column span of the first j columns of A will be the same as the first j columns of Q, assuming that R is 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 thism 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.

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

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

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

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

for $k, \ell < 1$. 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 the same properties as the assumption.

We now reinterpret this construction as a reduced QR decomposition. Define

$$r_{jj} := \|\mathbf{v}_j\|, \qquad r_{kj} := r_{kj}.$$

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.

We now see this 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}:
 -0.369696    0.756191    0.0790538  -0.462543
  0.0909861  -0.457487   -0.0727869  -0.878095
 -0.297037    0.0910467  -0.87819     0.0267829
  0.0464023   0.249774    0.374698   -0.0775168
 -0.87445    -0.384973    0.277195    0.090975
```

In [ ]:
```
R̂
```

Out[ ]:
```
4×4 Matrix{Float64}:
 1.64104  -0.396842  -1.02978   0.461145
 0.0      -1.86871   -0.945953  0.850716
 0.0       0.0       -2.31435   0.590741
 0.0       0.0        0.0       2.00013
```

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}:
 -0.36969570259539347
  0.09098614771064727
 -0.29703665012362696
  0.046402266072449544
 -0.8744499222905507
```

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}:
 -0.7561912188542081
  0.4574866564648364
 -0.09104665086586611
 -0.2497739161085708
  0.3849731131940379
```

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}:
 -0.07905381801563263
  0.07278690174258332
  0.8781895117748354
 -0.3746983472903358
 -0.2771946802130837
```

(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[ ]: 2.8313748239171495e-16

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[ ]: 1.0802568964667228e-10

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 $r_{11} = -\mathrm{sign}(a_{11})\|\mathbf{a}_1\|$ 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

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

so that

$$Q_2Q_1A = \begin{bmatrix} \times & \times & \times & \cdots & \times \\ & \times & \times & \cdots & \times \\ & & \vdots & \ddots & \vdots \\ & & \times & \cdots & \times \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ & r_{22} & r_{23} & \cdots & r_{2n} \\ & & \mathbf{a}_3^2 & \cdots & \mathbf{a}_n^2 \end{bmatrix}$$

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

$$Q_n\cdots Q_1A = \underbrace{\begin{bmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \\ & & 0 \\ & & \vdots \\ & & 0 \end{bmatrix}}_{R}$$

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

$$Q_2 Q_1 A = \begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & 0 & 0 \\ & 0 & -6 \end{bmatrix}$$

The final reflection is

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

giving us

$$Q_3 Q_2 Q_1 A = \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R}$$

That is,

$$A = Q_1 Q_2 Q_3 R = \underbrace{\frac{1}{3}\begin{bmatrix} -1 & 0 & 2 & 2 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & -1 & 2 \\ 2 & 0 & 2 & -1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R} = \underbrace{\frac{1}{3}\begin{bmatrix} -1 & 0 & 2 \\ 0 & 3 & 0 \\ 2 & 0 & -1 \\ 2 & 0 & 2 \end{bmatrix}}_{\hat{Q}} \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \end{bmatrix}}_{\hat{R}}$$

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.

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

$$L_{\mathbf{v}} = \begin{bmatrix} 1 & & & \\ -\frac{v_2}{v_1} & 1 & & \\ \vdots & & \ddots & \\ -\frac{v_n}{v_1} & & & 1 \end{bmatrix} = I - \left[\frac{\mathbf{v}}{v_1} - \mathbf{e}_1\right]\mathbf{e}_1^\top$$

so that

$$L_{\mathbf{v}}\mathbf{v} = \mathbf{v} - (\mathbf{v} - v_1\mathbf{e}_1) = v_1\mathbf{e}_1.$$

For $L_1 := L_{\mathbf{a}_1}$ we 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 $\mathbf{a}_j^1 := (L_1\mathbf{a}_j)[2:m]$ and $u_{1j} = a_{1j}$. But now consider

$$L_2 := \begin{bmatrix} 1 & \\ & L_{\mathbf{a}_2^1} \end{bmatrix}.$$

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

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

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

$$\begin{aligned} L_j &:= \begin{bmatrix} I_{j-1 \times j-1} & \\ & L_{\mathbf{a}_j^j} \end{bmatrix} \\ \mathbf{a}_j^k &:= (L_k \mathbf{a}_j^{k-1})[2 : m-k+1] \\ u_{kj} &:= (L_k \mathbf{a}_j^{k-1})[1] \end{aligned}$$

Then

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

i.e.

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

Finally, note that the L_k are immediately invertible:

$$L_{\mathbf{v}}^{-1} = \begin{bmatrix} 1 & & & \\ \frac{v_2}{v_1} & 1 & & \\ \vdots & & \ddots & \\ \frac{v_n}{v_1} & & & 1 \end{bmatrix} = I + \left[\frac{\mathbf{v}}{v_1} - \mathbf{e}_1 \right] \mathbf{e}_1^\top$$

Moreover, as L_k we note that L simplifies:

Example 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
 -0.157922   1.0   -0.0   -0.0
 -0.977475  -0.0    1.0   -0.0
  1.0501    -0.0   -0.0    1.0
```

Which indeed introduces zeros in the first column:

In [ ]:
```
A₁ = L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
 0.89756   0.367246   1.63822    0.863699
 0.0      -0.167934   0.124238  -0.719144
 0.0       0.411062  -2.00338   -0.477911
 0.0       0.696679   0.435702  -0.495634
```

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   2.44776   1.0  -0.0
 -0.0   4.14854  -0.0   1.0
```

So that

In [ ]:
```
A₂ = L₂*L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
 0.89756       0.367246   1.63822    0.863699
 0.0          -0.167934   0.124238  -0.719144
 0.0           0.0       -1.69927   -2.23821
 1.11022e-16   0.0        0.951109  -3.47903
```

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   0.559716   1.0
```

Giving us

In [ ]:
```
U = L₃*L₂*L₁*A
```

Out[ ]:
```
4×4 Matrix{Float64}:
 0.89756   0.367246      1.63822        0.863699
 0.0      -0.167934      0.124238      -0.719144
 0.0       0.0          -1.69927       -2.23821
 0.0       1.11022e-16  -2.77556e-16   -4.73179
```

and

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

Out[ ]:
```
4×4 Matrix{Float64}:
  1.0        0.0       0.0       0.0
  0.157922   1.0       0.0       0.0
  0.977475  -2.44776   1.0       0.0
 -1.0501    -4.14854  -0.559716  1.0
```

Note how the entries of `L` are indeed identical to the negative lower entries of `L1`, `L2` and `L3`.

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: move the largest in magnitude entry to the diagonal. In terms of a decomposition, this leads to

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

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

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 $\mathbf{k} \in \{1, \dots, n\}^n$ where any integer appears only once then $A[\mathbf{k}, \mathbf{k}]$ is also positive definite.

We leave the proofs to the problem sheets.

Write a symmetric pos. def. matrix as

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

Note that A_1 is positive definite (because $A_1 = L_1^{-1} A L_1^{-\top}$) and therefore $K - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha}$ is also positive definite (as it is a subslice of A_1). Thus we can repeat.

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 = 1000
A = Symmetric(rand(n,n)) + 100I # shift by 10 ensures positivity
@btime cholesky(A);
@btime lu(A);
@btime qr(A);
```

```
  4.221 ms (5 allocations: 7.63 MiB)
  7.622 ms (4 allocations: 7.64 MiB)
  24.841 ms (7 allocations: 8.18 MiB)
```

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

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.

For this reason,