HOMEWORK 3 SUBMISSION

Jinen Setpal

October 27, 2025

Checklist

1. Cross-checked independent work with Kunal Kapur.
2. No use of AI tools.
3. Code is included!

Problem 1

We have:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_3 & \boldsymbol{A}_4 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$

Which decomposes into the following two systems:

$$\begin{aligned} \boldsymbol{A}_1\mathbf{x}_1 + \boldsymbol{A}_2\mathbf{x}_2 &= \mathbf{b}_1 \\ \boldsymbol{A}_3\mathbf{x}_1 + \boldsymbol{A}_4\mathbf{x}_2 &= \mathbf{b}_2 \end{aligned}$$

We can assume $\mathbf{x}_2$ given and solve for $\mathbf{x}_1$:

$$\begin{aligned} &\mathbf{x}_1 = \boldsymbol{A}_1^{-1}(\mathbf{b}_1 - \boldsymbol{A}_2\mathbf{x}_2) \\ &\implies \boldsymbol{A}_3\boldsymbol{A}_1^{-1}(\mathbf{b}_1 - \boldsymbol{A}_2\mathbf{x}_2) + \boldsymbol{A}_4\mathbf{x}_2 = \mathbf{b}_2 \\ &\implies \boldsymbol{A}_3\boldsymbol{A}_1^{-1}\mathbf{b}_1 - \boldsymbol{A}_3\boldsymbol{A}_1^{-1}\boldsymbol{A}_2\mathbf{x}_2 + \boldsymbol{A}_4\mathbf{x}_2 = \mathbf{b}_2 \\ &\implies (\boldsymbol{A}_4 - \boldsymbol{A}_3\boldsymbol{A}_1^{-1}\boldsymbol{A}_2)\mathbf{x}_2 = \mathbf{b}_2 - \boldsymbol{A}_3\boldsymbol{A}_1^{-1}\mathbf{b}_1 \end{aligned}$$

Now, we have inverses, but we can just recurse on our solver until we have a trivial system.

Code

```
using SparseArrays, LinearAlgebra, Plots

function solve(A::Matrix, B::Matrix)
        n, k = size(B)
        factor = div(n, 2)

        if n == 1
                return B ./ A
        else
                A_1 = A[1:factor, 1:factor]
                A_2 = A[1:factor, end-factor+1:end]
                A_3 = A[end-factor+1:end, 1:factor]
                A_4 = A[end-factor+1:end, end-factor+1:end]

                x_2 = solve(A_4 - A_3 * solve(A_1, A_2), B[end-factor+1:end,
                    :] - A_3 * solve(A_1, B[1:factor, :]))
                x_1 = solve(A_1, B[1:factor, :] - A_2 * x_2)
                return vcat(x_1, x_2)
        end
end

Z = randn(2^4, 2^4)
```

```
A = Z'Z
b = ones(size(Z)[1], 1)

x_hat = solve(A, b)
println(norm(x_hat - A\b))
```

Output

```
7.166880891681717e-9
```

Problem 2

1. Each step of Cholesky Factorization does two things: a) it preserves the symmetry and positive definiteness of a matrix, and b) it reduces the size of the matrix from $n \times n$ to $n-1 \times n-1$. This reveals a simple application of induction, given it holds trivially for a 1×1 case (just the square root of arbitrary input given positive definiteness), we can assume it works for the $k \times k$ case as our inductive hypothesis, and the inductive step reduces a $(k+1) \times (k+1)$ case to the $k \times k$ case. We therefore have a Cholesky factorization for any symmetric positive definite matrix of arbitrary dimension.

2. The positive definiteness condition states that, for a matrix $\boldsymbol{A}$, it holds that:

$$\mathbf{x}^T \boldsymbol{A} \mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0 \qquad \text{and} \qquad \mathbf{x}^T \boldsymbol{A}^T \mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0$$

The second equivalent statement is just the transpose, which holds because the left term is a scalar. We can simplify the statement by adding the two terms to get an equivalent condition:

$$\begin{aligned} & \mathbf{x}^T \boldsymbol{A} \mathbf{x} + \mathbf{x}^T \boldsymbol{A}^T \mathbf{x} > 0 && \forall \mathbf{x} \neq 0 \\ \iff & (\mathbf{x}^T \boldsymbol{A} + \mathbf{x}^T \boldsymbol{A}^T) \mathbf{x} > 0 && \forall \mathbf{x} \neq 0 \\ \iff & (\mathbf{x}^T (\boldsymbol{A} + \boldsymbol{A}^T)) \mathbf{x} > 0 && \forall \mathbf{x} \neq 0 \\ \iff & \mathbf{x}^T (\boldsymbol{A} + \boldsymbol{A}^T) \mathbf{x} > 0 && \forall \mathbf{x} \neq 0 \end{aligned}$$

This is awesome, because $\boldsymbol{B} := \boldsymbol{A} + \boldsymbol{A}^T$ is a symmetric matrix. Proof:

$$\boldsymbol{B}_{ij} = \sum_{k=1}^{n} \boldsymbol{A}_{i,k} + \boldsymbol{A}_{k,j}^T = \sum_{k=1}^{n} \boldsymbol{A}_{k,i}^T + \boldsymbol{A}_{j,k} = \boldsymbol{B}_{ji} \qquad \forall i, j$$

If $\alpha < 0$ in any step of Cholesky factorization of $\boldsymbol{B}$, then our matrix $\boldsymbol{B}$ is not positive definite, which is an equivalent condition for $\boldsymbol{A}$ not being positive definite. This allows us to modify the code from the textbook:

Code

```
using SparseArrays, LinearAlgebra

# important: copied and modified from textbook
function check_definiteness(X::Matrix, check_positive::Bool)  # if
    check_positive = true, check if positive definite, else check if
    negative definite
        A = copy(Float64.(X) + X')
        if !check_positive
                A = -A
        end

        n = size(A,1)
        F = Matrix(1.0I,n,n)
        d = zeros(n)
        for i=1:n-1
                alpha = A[i,i]
                if alpha < 0
                        return false
                end
                d[i] = sqrt(alpha)
                F[i+1:end,i] = A[i+1:end,i]/alpha
                A[i+1:end, i] -= A[i+1:end, i] * A[i, i+1:end]'/alpha
        end
        return true
end

A = randn(10, 10)
```

```julia
PD = A'*A
println("Sanity Check 1 (expect true): ", check_definiteness(PD, true))
println("Sanity Check 2 (expect false): ", check_definiteness(A, true))


function map_index(i::Integer, j::Integer, n::Integer)
        if 1 < i < n+1 && 1 < j < n+1
                return 4n + (i - 2)*(n-1) + j-1
        elseif i == 1
                return j
        elseif i == n+1
                return n + 1 + j
        elseif j == 1
                return 2(n+1) + i - 1
        elseif j == n+1
                return 2(n+1) + n - 2 + i
        end
end

function laplacian(n::Integer, f::Function)
        A = sparse(1I, (n+1)^2, (n+1)^2)
        A[diagind(A)[4n+1:end]] .= -4

        fvec = zeros((n+1)^2)

        global row_index = 4n + 1
        for i in 2:n
                for j in 2:n
                        A[row_index, map_index(i-1, j, n)] = 1
                        A[row_index, map_index(i+1, j, n)] = 1
                        A[row_index, map_index(i, j-1, n)] = 1
                        A[row_index, map_index(i, j+1, n)] = 1
                        fvec[row_index] = f(i, j)

                        global row_index += 1
                end
        end

        return A, fvec/n^2
end

n = 10
A, fv = laplacian(n, (x, y) -> 1)
println()
println("Laplacian with boundary conditions (Positive): ",
   check_definiteness(Matrix(A), true))
println("Laplacian with boundary conditions (Negative): ",
   check_definiteness(Matrix(A), false))

A_smol = A[4n+1:end,4n+1:end]
println()
println("Laplacian without boundary conditions (Positive): ",
   check_definiteness(Matrix(A_smol), true))
println("Laplacian without boundary conditions (Negative): ",
   check_definiteness(Matrix(A_smol), false))
```

Code

```
Sanity Check 1 (expect true): true
Sanity Check 2 (expect false): false

Laplacian with boundary conditions (Positive): false
Laplacian with boundary conditions (Negative): false

Laplacian without boundary conditions (Positive): false
Laplacian without boundary conditions (Negative): true
```

Thus we conclude that Poisson's equation from homework 1 *without boundary conditions* is **negative definite**.

Problem 3

We have:

$$
\boldsymbol{A} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}
$$

1. Since $\boldsymbol{A}_{n-1}$ is tridiagonal, we have that:

$$
(\boldsymbol{A}_{n-1})_{ij} = (\boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T)_{ij} = 0 \qquad \forall i,j : j-1 \leq i \leq j+1
$$

$$
(\boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T)_{ij} = \sum_{k=1}^{n} (\boldsymbol{L}_{n-1})_{ik}(\boldsymbol{L}_{n-1}^T)_{kj} = \sum_{k=1}^{n} (\boldsymbol{L}_{n-1})_{ik}(\boldsymbol{L}_{n-1})_{jk} = \langle \boldsymbol{L}_{i,:}, \boldsymbol{L}_{j,:} \rangle
$$

Let $[n] = \{1, 2, \ldots, n\}$. Since $\boldsymbol{A}_{n-1}$, is positive definite, we have that $(\boldsymbol{A}_{n-1})_{ii} > 0 \ \forall i \in [n-1]$. Therefore,

$$
(\boldsymbol{A}_{n-1})_{ii} = \sum_{k=1}^{n-1} (\boldsymbol{L}_{n-1})_{ik}^2 = \|L_{i,:}\|_2^2 > 0 \qquad \forall i \in [n-1]
$$

For $i = 1$, we have:

$$
(\boldsymbol{A}_{n-1})_{11} = \sum_{k=1}^{n-1} (\boldsymbol{L}_{n-1})_{1k}(\boldsymbol{L}_{n-1})_{k1} = (\boldsymbol{L}_{n-1})_{11}(\boldsymbol{L}_{n-1})_{11} = (\boldsymbol{L}_{n-1})_{11}^2 > 0
$$

Thus $(\boldsymbol{L}_{n-1})_{11} > 0$. We can use this to show that $(\boldsymbol{L}_{n-1})_{i,j} = 0 \ \forall i > j+1$ for $j = 1$:

$$
(\boldsymbol{A}_{n-1})_{i1} = 0 = \langle (\boldsymbol{L}_{n-1})_{i,:}, (\boldsymbol{L}_{n-1})_{1,:} \rangle = (\boldsymbol{L}_{n-1})_{i1}(\boldsymbol{L}_{n-1})_{11}
$$

Since $(\boldsymbol{L}_{n-1})_{11} > 0$, it must hold that $(\boldsymbol{L}_{n-1})_{i1} = 0$. Letting $(\boldsymbol{L}_{n-1})_{21}$ remain arbitrary, we've comprehensively characterized the first column. We can extend this to arbitrary columns next.

Inductive Hypothesis: Given column k, $(\boldsymbol{L}_{n-1})_{j,j} > 0$ and $(\boldsymbol{L}_{n-1})_{i,j} = 0 \ \forall i > j+1 \ \forall j \leq k$.

Inductive Step: T.P.T. For column $k+1$, $(\boldsymbol{L}_{n-1})_{k+1,k+1} > 0$ and $(\boldsymbol{L}_{n-1})_{i,k+1} = 0 \ \forall i > k+2$.

We can use the above result to show that $(\boldsymbol{L}_{n-1})_{k+1,k+1} > 0$:

$$
\begin{aligned}
(\boldsymbol{A}_{n-1})_{k+1,k+1} &= \langle (\boldsymbol{L}_{n-1})_{k+1,:}, (\boldsymbol{L}_{n-1})_{k+1,:} \rangle > 0 \\
&= \sum_{l=1}^{(k+1)-1} (\boldsymbol{L}_{n-1})_{k+1,l}^2 > 0 \\
&= (\boldsymbol{L}_{n-1})_{k+1,k+1}^2 + \sum_{l=1}^{k-1} (\boldsymbol{L}_{n-1})_{k+1,l}^2 > 0 \\
&= (\boldsymbol{L}_{n-1})_{k+1,k+1}^2 + 0 > 0 \\
(\boldsymbol{A}_{n-1})_{k+1,k+1} &= (\boldsymbol{L}_{n-1})_{k+1,k+1}^2 > 0
\end{aligned}
$$

Finally, we can show that $(\boldsymbol{L}_{n-1})_{i,k+1} = 0 \ \forall i > k+2$:

$$
(\boldsymbol{A}_{n-1})_{i,k+1} = 0 = \langle (\boldsymbol{L}_{n-1})_{i,:}, (\boldsymbol{L}_{n-1})_{k+1,:} \rangle = 0 + 0 + \ldots + 0 + (\boldsymbol{L}_{n-1})_{i,k+1}(\boldsymbol{L}_{n-1})_{k+1,k+1}
$$

Since $(\boldsymbol{L}_{n-1})_{k+1,k+1} > 0$, it holds that $(\boldsymbol{L}_{n-1})_{i,k+1} = 0$.

Therefore, we have proven that $\boldsymbol{L}_{n-1}$ has a non-zero diagonal, and that it is sparse; it is populated only by the diagonal and subdiagonal elements.

2. We can setup this question by unpacking a single step of Cholesky Decomposition:

$$\boldsymbol{A} = \begin{bmatrix} \alpha_1 & \beta_1 \mathbf{e}_1^T \\ \beta_1 \mathbf{e}_1 & \boldsymbol{A}_{n-1} \end{bmatrix} = \boldsymbol{C}\boldsymbol{C}^T = \begin{bmatrix} \gamma & 0 \\ \mathbf{c} & \boldsymbol{C}_1 \end{bmatrix} \begin{bmatrix} \gamma & \mathbf{c}^T \\ 0 & \boldsymbol{C}_1^T \end{bmatrix} = \begin{bmatrix} \gamma^2 & \gamma \mathbf{c}^T \\ \gamma \mathbf{c} & \mathbf{c}\mathbf{c}^T + \boldsymbol{C}_1 \boldsymbol{C}_1^T \end{bmatrix}$$

So, we have $\gamma = \sqrt{\alpha},\ \mathbf{c} = (\beta_1/\gamma)\mathbf{e}_1$. Substituting the given equalities, we have:

$$\boldsymbol{A}_{n-1} = \boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T = \mathbf{c}\mathbf{c}^T + \boldsymbol{C}_1\boldsymbol{C}_1^T = \left(\frac{\beta_1}{\sqrt{\alpha}}\right)^2 \mathbf{e}_{11} + \boldsymbol{C}_1\boldsymbol{C}_1^T$$

$$\implies \boldsymbol{C}_1\boldsymbol{C}_1^T = \boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T - \frac{\beta_1^2}{\alpha_1}\mathbf{e}_{11}$$

We need the Cholesky Factorization for the right hand term. We can integrate element subtraction maintaining the sparse structure of $\boldsymbol{L}_{n-1}$:

$$(\boldsymbol{C}_1)_{11}^2 = \langle (\boldsymbol{C}_1)_{1,:}, (\boldsymbol{C}_1)_{1,:} \rangle = (\boldsymbol{C}_1\boldsymbol{C}_1^T)_{11} = \langle (\boldsymbol{L}_{n-1})_{1,:}, (\boldsymbol{L}_{n-1})_{1,:} \rangle - \frac{\beta_1^2}{\alpha_1} = (\boldsymbol{L}_{n-1})_{11}^2 - \frac{\beta_1^2}{\alpha_1}$$

$$\therefore (\boldsymbol{C}_1)_{11} = \sqrt{(\boldsymbol{L}_{n-1})_{11}^2 - \frac{\beta_1^2}{\alpha_1}}$$

Here, we only pick the positive solution since we have shown in the previous question that the diagonal elements are each > 0. The other element affected by this change is $(\boldsymbol{C}_1\boldsymbol{C}_1^T)_{21}$:

$$(\boldsymbol{C}_1)_{11}(\boldsymbol{C}_1)_{21} = \langle (\boldsymbol{C}_1)_{1,:}, (\boldsymbol{C}_1)_{2,:} \rangle = (\boldsymbol{C}_1\boldsymbol{C}_1^T)_{21} = \langle (\boldsymbol{L}_{n-1})_{1,:}, (\boldsymbol{L}_{n-1})_{2,:} \rangle = (\boldsymbol{L}_{n-1})_{11}(\boldsymbol{L}_{n-1})_{21}$$

$$\therefore (\boldsymbol{C}_1)_{21} = \frac{(\boldsymbol{L}_{n-1})_{11}(\boldsymbol{L}_{n-1})_{21}}{(\boldsymbol{C}_1)_{11}} = \frac{(\boldsymbol{L}_{n-1})_{11}(\boldsymbol{L}_{n-1})_{21}}{\sqrt{(\boldsymbol{L}_{n-1})_{11}^2 - \frac{\beta_1^2}{\alpha_1}}}$$

The rest of the matrix is unaffected by the single-element change (because we have that $\boldsymbol{C}$ is only populated by the diagonal and sub-diagonal, $(\boldsymbol{C}_1)_{j1} = 0\ \forall j > 2$ and so the dot product breakdown used earlier can ignore $(\boldsymbol{C}_1)_{11}$) and is identical to $\boldsymbol{L}_{n-1}$. Thus, we have obtained $\gamma, \mathbf{c}, \boldsymbol{C}_1$ which together constitutes the Cholesky factorization of $\boldsymbol{A}$ following the first equation.

3. We can apply the algorithm mathematically described above using a constant set of operations:

Code

```
function tridiag_cholesky(A::Matrix)
        L = spzeros(size(A))
        n, m = size(A)

        if n == 1
                return sqrt(A)
        else
                L[1, 1] = sqrt(A[1, 1])  # set gamma
                L[2, 1] = A[2, 1] / L[1, 1]  # rest of column vector is
                    already zero

                # recurse till n = 1, and make relevant corrections
                L_1 = tridiag_cholesky(A[2:end, 2:end])
                L[2:end, 2:end] = L_1
                L[2, 2] = sqrt(L_1[1, 1]^2 - L[2, 1]^2)
                if n > 2
                        L[3, 2] = L_1[1, 1] * L_1[2, 1] / L[2, 2]
                end
        end

        return L
end
```

```
A = Tridiagonal(ones(9), Vector(1:10), ones(9))
F, d = oracle(Matrix(A))
println("Decomposition Error: ", norm(A - F*F'))
```

Output

```
Decomposition Error: 2.886579864025407e-15
```

This question is very cool!!

Problem 4

We have an undetermined linear system $(\boldsymbol{A} - \lambda \boldsymbol{I})\mathbf{x} = 0$. Running our elimination solver on this system results in the final 2x2 system of equations being linearly dependent. This propogates into a divide-by-zero error. For a full-rank system, our solver returns the *correct but undesired* trivial solution $\mathbf{x} = 0$.

Output

```
3-element Vector{Float64}:
 NaN
 NaN
 NaN
```

By fixing a single variable from the solution $\mathbf{x}_n := 1$, we can simply solve the linear system as normal. $\mathbf{x}_{:n-1}$ 'reacts' to $\mathbf{x}_n$ and the overall result is an eigenvector.

Code

```
using LinearAlgebra

function solve1_pivot2_fixed(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
                @show(b)
                display(A)
                return [1.]   #  <-- this is the only functional change!
        else
                # let's make sure we have an equation
                # that we can eliminate!
                # let's try that again, where we pick the
                # largest magnitude entry!
                maxval = abs(A[1,1])
                newrow = 1
                for j=2:n
                        if abs(A[j,1]) > maxval
                                newrow = j
                                maxval = abs(A[j,1])
                        end
                end
                if maxval < eps(1.0)
                        error("the system is singular")
                end
                @show newrow
                # 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]
                display(D)
                c = A[1,2:end]
                d = A[2:end,1]
                a = A[1,1]
                y = solve1_pivot2_fixed(D-d*c'/a, b[2:end]-b[1]/a*d)
```

```
                z = (b[1] - c'*y)/a
                return pushfirst!(y,z)
        end
end

A = [1 2 2; 0 2 1; -1 2 2]
lambda = 1

Y = A - lambda * I
b = zeros(size(Y)[1])

println("Valid problem? ", Bool(rank(A) - rank(Y)))
x_hat = solve1_pivot2_fixed(Y, b)[1:end]
println("Correct Solution? ", all((A * x_hat) ./ x_hat .== lambda))
```

Output

```
Valid problem? true
newrow = 3
2x2 Matrix{Int64}:
 1  1
 2  2
newrow = 2
1x1 Matrix{Float64}:
 1.0
b = [0.0]
1x1 Matrix{Float64}:
 0.0
Correct Solution? true
```

Problem 5

We have two approaches to producing orthogonalizing $\mathbf{a}$:

$$\boldsymbol{H}\mathbf{a} = \pm\|\mathbf{a}\|\mathbf{e}_1 = \boldsymbol{G}_{n-1}\boldsymbol{G}_{n-2}\cdots\boldsymbol{G}_1\mathbf{a}$$

Note: I got confused about how to interpret a 'length n vector', because rotations preserve vector length by default, and therefore both parts of this question have an identical answer. From reviewing the lecture, I instead assumed length as number of elements in the vector; this formulation is similar to the class discussion.

1. For a 2-dimensional vector:

$$\boldsymbol{G}\mathbf{a} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} ca_1 + sa_2 \\ -sa_1 + ca_2 \end{bmatrix} = \begin{bmatrix} \|\mathbf{a}\| \\ 0 \end{bmatrix}$$
$$\therefore c = \frac{a_1}{\|\mathbf{a}\|}, \quad s = \frac{a_2}{\|\mathbf{a}\|}$$

The orthogonal matrix from the Given's rotation is $\boldsymbol{G}^T$:

$$\mathbf{a} = \underbrace{\boldsymbol{G}^T}_{\boldsymbol{Q}}\underbrace{\|\mathbf{a}\|\mathbf{e}_1}_{\boldsymbol{R}} = \begin{bmatrix} \frac{a_1}{\|\mathbf{a}\|} & -\frac{a_2}{\|\mathbf{a}\|} \\ \frac{a_2}{\|\mathbf{a}\|} & \frac{a_1}{\|\mathbf{a}\|} \end{bmatrix}\begin{bmatrix} \|\mathbf{a}\| \\ 0 \end{bmatrix} = \frac{1}{\|\mathbf{a}\|}\begin{bmatrix} a_1 & -a_2 \\ a_2 & a_1 \end{bmatrix}\begin{bmatrix} \|\mathbf{a}\| \\ 0 \end{bmatrix}$$

We can do something similar with Householder transformations:

$$\boldsymbol{H} = \boldsymbol{I} - \frac{2\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|_2^2} \quad \text{where} \quad \mathbf{u} = \mathbf{a} - \|\mathbf{a}\|\mathbf{e}_1 = \begin{bmatrix} a_1 - \|\mathbf{a}\| \\ a_2 \end{bmatrix}$$

$$\begin{aligned}
\therefore \boldsymbol{H} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{2}{\mathbf{u}^T\mathbf{u}}\begin{bmatrix} a_1 - \|\mathbf{a}\| \\ a_2 \end{bmatrix}\begin{bmatrix} a_1 - \|\mathbf{a}\| & a_2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{2}{(a_1 - \sqrt{a_1^2 + a_2^2})^2 + a_2^2}\begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{2}{2(a_1^2 + +a_2^2 - \sqrt{a_1^2 + a_2^2})}\begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{\|\mathbf{a}\|^2 - \|\mathbf{a}\|}\begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{\|\mathbf{a}\|(1 - \|\mathbf{a}\|)}\begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 \end{bmatrix}
\end{aligned}$$

The orthogonal matrix from the Householder's rotation is $\boldsymbol{H}^T$:

$$\mathbf{a} = \underbrace{\boldsymbol{H}^T}_{\boldsymbol{Q}}\underbrace{\|\mathbf{a}\|\mathbf{e}_1}_{\boldsymbol{R}} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{\|\mathbf{a}\|(1 - \|\mathbf{a}\|)}\begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 \end{bmatrix}\right)\|\mathbf{a}\|\mathbf{e}_1$$

$\boldsymbol{Q}$ in case of Householder is symmetric, whereas Givens' produces a skew-symmetric matrix.

2. For length 3 vectors, we have:

$$\boldsymbol{G} = \boldsymbol{G}_2\boldsymbol{G}_1 = \begin{pmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{pmatrix} = \begin{pmatrix} c_2c_1 & s_2 & c_2s_1 \\ -c_1s_2 & c & -s_2s_1 \\ -s_1 & 0 & c_1 \end{pmatrix}$$
$$\mathbf{a} = \boldsymbol{G}^T\|\mathbf{a}\|\mathbf{e}_1 = \begin{bmatrix} c_2c_1 & -c_1s_2 & -s_1 \\ s_2 & c & 0 \\ c_2s_1 & -s_2s_1 & c_1 \end{bmatrix}\|\mathbf{a}\|\mathbf{e}_1$$

The orthogonal matrix in this case is slightly sparse, but we can't really take advantage of this because $\boldsymbol{R}$ is itself sparse and has a zero in the third index. For the case of Householder, we have:

$$\boldsymbol{H} = \boldsymbol{I} - \frac{2\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|_2^2} \quad \text{where} \quad \mathbf{u} = \mathbf{a} - \|\mathbf{a}\|\mathbf{e}_1 = \begin{bmatrix} a_1 - \|\mathbf{a}\| \\ a_2 \\ a_3 \end{bmatrix}$$

$$\begin{aligned}
\therefore \boldsymbol{H} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{2}{\mathbf{u}^T\mathbf{u}} \begin{bmatrix} a_1 - \|\mathbf{a}\| \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} a_1 - \|\mathbf{a}\| & a_2 & a_3 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{2}{\mathbf{u}^T\mathbf{u}} \begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 & (a_1 - \|\mathbf{a}\|)a_3 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 & a_2a_3 \\ (a_1 - \|\mathbf{a}\|)a_3 & a_3a_2 & a_3^2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{\|\mathbf{a}\|(1 - \|\mathbf{a}\|)} \begin{bmatrix} (a_1 - \|\mathbf{a}\|)^2 & (a_1 - \|\mathbf{a}\|)a_2 & (a_1 - \|\mathbf{a}\|)a_3 \\ (a_1 - \|\mathbf{a}\|)a_2 & a_2^2 & a_2a_3 \\ (a_1 - \|\mathbf{a}\|)a_3 & a_3a_2 & a_3^2 \end{bmatrix}
\end{aligned}$$

This allows us to make two observations: appending additional dimensions to Householder can be cheap, because we need to only add $2n - 1$ elements to a Householder matrix of one fewer dimensions. Further, since norms increase monotonically as the the overall magnitude of the orthogonal matrix is also more likelk y to reduce. If it doesn't, then it is likely that $\mathbf{a}$ is inherently sparse to begin with, and Given's rotations may produce an orthogonal matrix more cheaply.