```julia
using LinearAlgebra, Plots, Statistics, Random, Polynomials
```

Problem 1

Task 1

At each observation (x_i, y_i), we define a residual, $y_i - f(x_i)$. Another possible formulation for our problem is to minimize:

$$R(\alpha_1, \dots, \alpha_n) = \sum_{i=1}^{m} [y_i - f(x_i)]^2,$$

over all possible choices of parameters $\alpha_1, \dots, \alpha_n$. We can rewrite the problem in the form $R = \mathbf{r}^T \mathbf{r}$, where

$$\mathbf{r} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix} - \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_{m-1}) & \phi_2(x_{m-1}) & \dots & \phi_n(x_{m-1}) \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_n(x_m) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

Recalling that $\mathbf{r}^T \mathbf{r} = \|\mathbf{r}\|^2$, and renaming the variables, we can express our problem as the linear least-squares problem:

$$\arg\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{b} - A\mathbf{x}\|_2$$

with

$$\mathbf{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix} \qquad A = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_{m-1}) & \phi_2(x_{m-1}) & \dots & \phi_n(x_{m-1}) \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_n(x_m) \end{bmatrix}$$

and $\mathbf{x}$ in the form requested.

Task 2

Given the generic expression for $\phi_k(x)$, we can rewrite A as the *Vandermonde* matrix V:

$$V = \begin{bmatrix} 1 & (x_1) & \dots & (x_1)^{n-1} \\ 1 & (x_2) & \dots & (x_2)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (x_{m-1}) & \dots & (x_{m-1})^{n-1} \\ 1 & (x_m) & \dots & (x_m)^{n-1} \end{bmatrix}$$

We wish to prove that rank(V) = n if $x_i \neq x_j$ for $i \neq j$. The rank corresponds to the maximal number of linearly independent columns of V.

Suppose that V is *not* of rank n. Then, the columns should not be linearly indipendent, that is, it would be possible to find some vector $\mathbf{c} = (c_1, \dots, c_n)^T : V\mathbf{c} = \mathbf{0}$.

This expression would give rise to a polynomial $p(x) = c_1 + c_2 * x + \dots + c_n * x^{n-1}$. In order for $V\mathbf{c} = \mathbf{0}$ to be satisfied, this would mean that all the x_i are roots (as they are distinct from each other). But the cardinality of x is $m > n$, so a polynomial of degree n-1 would have more than n-1 roots, which is not possible.

Therefore if $x_i \neq x_j$ for $i \neq j$, the matrix V must be of full rank, rank(V) = n.

Task 3

Let's define the data for our problem as arrays, and compute the Vandermonde matrix:

```
9×3 Matrix{Int64}:
 1    8     64
 1   10    100
 1   12    144
 1   16    256
 1   20    400
 1   30    900
 1   40   1600
 1   60   3600
 1  100  10000
```

```julia
begin
x = [8, 10, 12, 16, 20, 30, 40, 60, 100]
y = [0.88, 1.22, 1.64, 2.72, 3.96, 7.66, 11.96, 21.56, 43.16]
V = [ x[i]^j for i in 1:length(x), j in 0:2 ]
end
```

Here we define the function for computing a solution through the Cholesky decomposition:

First, we define the functions `forwardsub(L,b)` and `backsub(U,b)` to solve the lower triangular linear system with matrix `L` and right-hand side vector `b` and the upper triangular linear system with matrix `U` and right-hand side vector `b` respectively.

```
forwardsub (generic function with 1 method)
```

```
function forwardsub(L,b)
     n = size(L,1)
     x = zeros(n)
    x[1] = b[1]/L[1,1]
    for i in 2:n
        s = sum( L[i,j]*x[j] for j in 1:i-1 )
        x[i] = ( b[i] - s ) / L[i,i]
    end
    return x
end
```

```
backsub (generic function with 1 method)
```

```
function backsub(U,b)
     n = size(U,1)
    x = zeros(n)
    x[n] = b[n]/U[n,n]
    for i in n-1:-1:1
        s = sum( U[i,j]*x[j] for j in i+1:n )
        x[i] = ( b[i] - s ) / U[i,i]
    end
    return x
end
```

Then we define `lsnormal(A,b)` to solve a linear least-squares problem by the normal equations. This function returns the minimizer of ||b-Ax||.

```
lsnormal (generic function with 1 method)
```

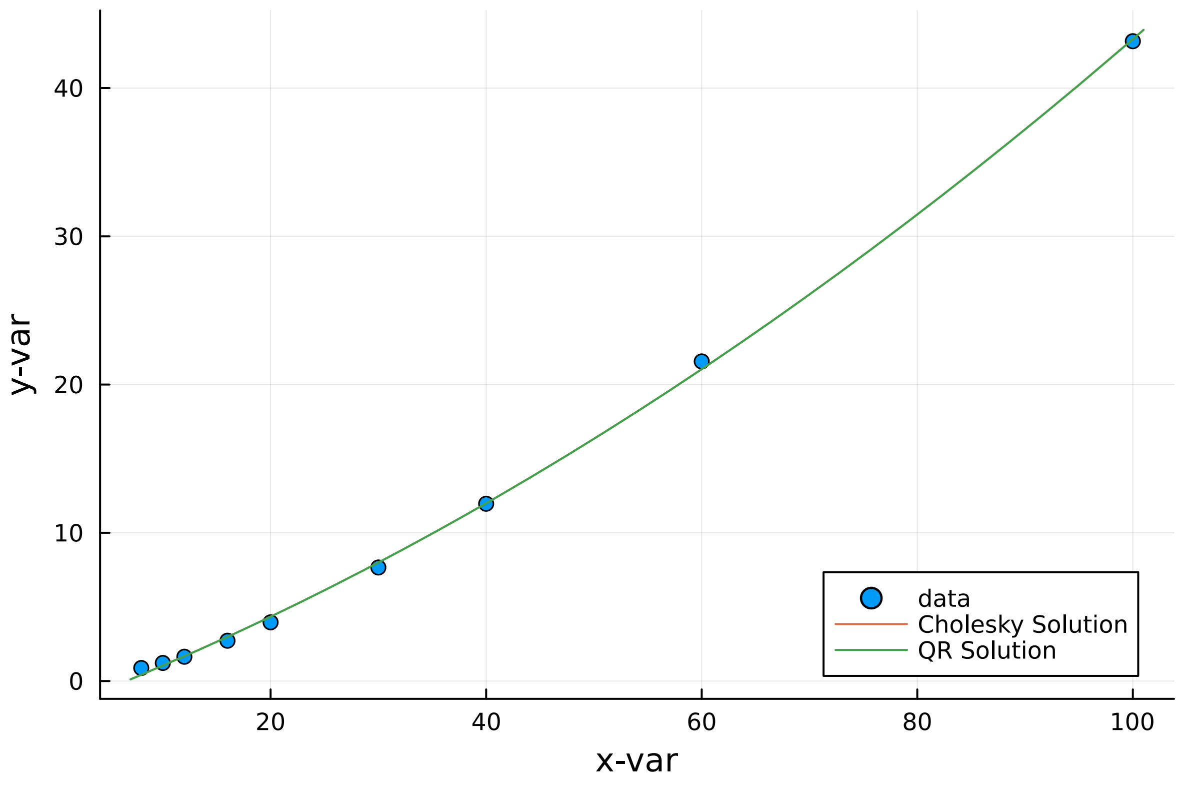
```
function lsnormal(A,b)
    # NOTE: We know that C is square and SPD!
    C = A'*A;  d = A'*b;
    # get cholesky decomposition
    # in upper triangular form
    R = cholesky(C).U
    # solve (R^T)Rx=d
    w = forwardsub(R',d)          # solves R^T w = d
    x = backsub(R,w)              # solves R x = w
    return x
end
```

And now solve!

```julia
begin
    α_C = lsnormal(V, y)
    println(α_C)
    α_QR = V \ y
    println(α_QR)
    # compare the two solutions
    #  This corresponds to requiring equality of about half of the significant
    digits.
    println("The two solutions are approximately equal: ", α_C ≈ α_QR)
    # compute the difference between the two solutions
    println("The difference between the two solutions is: ", norm(α_C - α_QR, 2))
end
```

```
[-1.919149252699108, 0.2782135362917263, 0.0017394008750551443]
[-1.919149252699468, 0.2782135362917223, 0.0017394008750551816]
The two solutions are approximately equal: true
The difference between the two solutions is: 6.1414514523205e-14
```

We plot the results below:



```julia
begin
p_C = Polynomial(α_C)
p_QR = Polynomial(α_QR)

f_C = x -> p_C(x)
f_QR = x -> p_QR(x)
scatter(x,y,label="data",
    xlabel="x-var",ylabel="y-var",leg=:bottomright)
plot!(f_C,7,101,label="Cholesky Solution")
plot!(f_QR,7,101,label="QR Solution")
end
```

Task 4

We compute the residuals for the two solutions. Notice that the resiudal of interest is $\mathbf{r} = \mathbf{d} - C * \alpha$. This residual measures the error in the solution of the system itself, rather than the fit of the model to the observed data (which would be $\mathbf{r} = \mathbf{y} - V\mathbf{x}$ in a regression context).

```
begin
C = V'*V
d = V'*y
r_C = d - C*α_C
r_QR = d - C*α_QR
println("The norm of the residual for the Cholesky solution is: ", norm(r_C, 2))
println("The norm of the residual for the QR solution is: ", norm(r_QR, 2))
end
```

```
The norm of the residual for the Cholesky solution is: 0.0
The norm of the residual for the QR solution is: 1.1641532269429655e-10
```

And now we compute the residuals for the approximate solution $\hat{\alpha}$:

```
begin
α_hat = [-1.919, 0.2782, 0.001739]
@show size(C)
r_hat = d - C*α_hat
println(r_hat)
println("The norm of the residual for the approximate solution is: ",
norm(r_hat, 2))
end
```

```
size(C) = (3, 3)
[0.009503999999999263, 0.716895999999906, 62.090847999905236]
The norm of the residual for the approximate solution is: 62.09498720144942
```

We can notice that the residual for this approximate solution has a much bigger norm compared to those of the solutions computed by Cholseky or QR Factorization. Why is this the case? Clearly, this solution is significantly less accurate than the one obtained via, e.g., QR decomposition. We can also use the following inequality to explain our findings:

$$\kappa^{-1}\frac{\|r\|_2}{\|d\|_2} \leq \frac{\|x-\hat{x}\|_2}{\|x\|_2} \leq \kappa\frac{\|r\|_2}{\|d\|_2}$$

Note that κ, the conditioning number of C, is particularly high. This is due to the ill-conditioned nature of V, which propagates to C as:

$$\kappa(V^TV) = \kappa(V)^2.$$

```
9489.567358824059
```

```julia
begin
κ = cond(C)
κ_V = cond(V)
@show κ
@show κ_V, κ_V^2
println("κ approximately equal κ_V**2 with atol=1e-3: ", isapprox(κ, κ_V^2,
atol=1e-3))

norm_r = norm(r_hat)
norm_d = norm(d)

LB = (norm_r/norm_d)/κ
@show LB
UB = κ*(norm_r/norm_d)
@show UB
end
```

```
κ = 8.221590912906387e7
(κ_V, κ_V ^ 2) = (9067.298888237148, 8.221590912862663e7)
κ approximately equal κ_V**2 with atol=1e-3: true
LB = 1.403895189265603e-12
UB = 9489.567358824059
```

Problem 2

Task 1

Let $A = Q_1 R_1$ be the reduced QR factorization of A.

Then, suppose that R_1 is singular: this would mean that $\exists\ \mathbf{v} \neq \mathbf{0} : R_1\mathbf{v} = \mathbf{0}$. But then $A\mathbf{v} = Q_1 R_1 \mathbf{v} = \mathbf{0}$, and this is not possible as A is of full rank. Thus, R_1 must be non-singular.

The columns of Q_1 are orthonormal, by the property of the QR factorization, and they form a basis for the columns of A . Since A has full rank, the columns of A, and therefore the columns of Q_1, span Ran(A). Being orthonormal and spanning Ran(A), they form an orthonormal basis for Ran(A).

Null(A^T) is the orthogonal complement of Ran(A). Then, the extension of the orthonormal set $(\mathbf{q_1}, \ldots, \mathbf{q_n})$ to a basis of $\mathbb{R}^m$, that is $(\mathbf{q_{n+1}}, \ldots, \mathbf{q_m})$, is an orthonormal basis of Null(A^T).

Task 2

$$A^T = \begin{bmatrix} 1.07 & 1.07 & 1.07 \\ 1.10 & 1.11 & 1.15 \end{bmatrix}$$

and

$$A^T A = \begin{bmatrix} 3.43 & 3.60 \\ 3.60 & 3.76 \end{bmatrix}$$

We first notice that $A^T A$ is square and symmetric. To check if it is positive define, we need to check if the eignevalues are positive. The discriminant is
$|A^T A| = 3.76 * 3.43 - 3.60 * 3.60 = -0.06$, so $A^T A$ cannot be positive definite.

Task 3

We want to decompose:

$$A = \begin{bmatrix} 1.07 & 1.10 \\ 1.07 & 1.11 \\ 1.07 & 1.15 \end{bmatrix}$$

The steps to do it are as follows:

2. **Calculate the first Householder vector z:**

We know that z is the first column of A: $z = \begin{bmatrix} 1.07 \\ 1.07 \\ 1.07 \end{bmatrix}$

3. **Compute the first Householder reflector v:**

The first reflector v is: $v = \frac{z - \|z\| \cdot e_1}{\|z - \|z\| \cdot e_1\|} = \begin{bmatrix} -0.46 \\ 0.628 \\ 0.628 \end{bmatrix}$

4. **Construct the first Householder matrix P_1:**

The first Householder matrix P_1 is: $P_1 = I - 2vv^T = \begin{bmatrix} 0.577 & 0.578 & 0.578 \\ 0.578 & 0.211 & -0.789 \\ 0.578 & -0.789 & 0.211 \end{bmatrix}$

5. **Apply P_1 to A to obtain A_1:**

The result is: $A_1 = P_1 A = \begin{bmatrix} 1.85 & 1.94 \\ -5.02e - 17 & -0.0373 \\ -7.6e - 17 & 0.00266 \end{bmatrix}$

6. **Repeat the process for the submatrix of A_1 to compute the second reflector:**

For the submatrix, we have: $z_1 = \begin{bmatrix} -0.0373 \\ 0.00266 \end{bmatrix}, \quad v_1 = \begin{bmatrix} -0.999 \\ 0.0356 \end{bmatrix}$

7. **Construct the second Householder matrix P_2:**

The second Householder matrix P_2 is: $P_2 = \begin{bmatrix} -1.0 & 0.0711 \\ 0.0711 & 0.997 \end{bmatrix}$

8. **Form the full Q_2 matrix from P_2:**

The full Q_2 matrix is: $Q_2 = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0711 \\ 0.0 & 0.0711 & 0.997 \end{bmatrix}$

9. **Apply Q_2 to A_1 to obtain the upper triangular matrix R:**

The final upper triangular matrix R is: $R = \begin{bmatrix} 1.85 & 1.94 \\ 4.48e-17 & 0.0375 \\ -7.93e-17 & -1.0e-8 \end{bmatrix}$

Remember that $Q = (Q_2 Q_1)^T = Q_1^T Q_2^T$, so:

$$Q = \begin{bmatrix} 0.577 & -0.537 & 0.617 \\ 0.578 & -0.267 & -0.772 \\ 0.578 & 0.804 & 0.154 \end{bmatrix}$$

When we multiply Q by R and round to two decimal places, we get: $QR = \begin{bmatrix} 1.07 & 1.10 \\ 1.07 & 1.11 \\ 1.07 & 1.15 \end{bmatrix}$

Which is precisely the original matrix A!

Task 4

Given a reduced QR decomposition of a $m \times n$ real matrix $A = Q_1 R_1$, with $Q_1 \in \mathbb{R}^{m \times n}$ and $R_1 \in \mathbb{R}^{n \times n}$, we want to find the reduced QR decomposition of the $m \times (n+1)$ matrix

$$A_+ = [A \quad \mathbf{b}]$$

in terms of the reduced QR decomposition of A.

To do so, we start from the full QR decomposition of A

$$A = QR = [Q_1 \quad Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

Here $Q \in \mathbb{R}^{m \times m}$ and $R \in \mathbb{R}^{m \times m}$. The columns of Q_1 form an orthonormal basis of the span of A, while the columns of Q_2 form an orthonormal basis of the kernel of A.

Now, let's consider the full QR decomposition of A_+ by applying consecutive Householder transformations: $H_1, H_2, \dots, H_n, H_{n+1}$. Since the first n columns of A_+ are the same of A, the first n Householder transformations are the same for the two matrices, hence

$$H_n \cdots H_1 A_+ = Q^T A_+ = \begin{bmatrix} R_1 & Q_1^T \mathbf{b} \\ 0 & Q_2^T \mathbf{b} \end{bmatrix} \equiv \begin{bmatrix} R_1 & \mathbf{z} \\ 0 & \mathbf{a} \end{bmatrix}$$

Where $\mathbf{z} \in \mathbb{R}^n$ and $\mathbf{a} \in \mathbb{R}^{(m-n)}$.

The last reflector is then of the form

$$H_{n+1} = \begin{bmatrix} I & 0 \\ 0 & \hat{H}_{n+1} \end{bmatrix}$$

with

$$\hat{H}_{n+1} = I - 2\mathbf{u}_{n+1}\mathbf{u}_{n+1}^T \in \mathbb{R}^{(m-n)\times(m-n)}$$

and $\mathbf{u}_{n+1}$ being the usual vector of unitary norm constructed from $\mathbf{a}$

With these premises, we are able to get the full size QR decomposition of A_+:

$$H_{n+1}\cdots H_1 A_+ = H_{n+1}Q^T A_+ = \begin{bmatrix} Q_1^T \\ \hat{H}_{n+1}Q_2^T \end{bmatrix} A_+ \equiv Q_+^T A_+ \equiv R_+ = \begin{bmatrix} R_1 & \mathbf{z} \\ 0 & \rho \\ 0 & 0 \end{bmatrix}$$

In the reduced form then Q_{1_+} is a $\mathbb{R}^{m\times(n+1)}$ matrix while $R_{1_+} \in \mathbb{R}^{(n+1)\times(n+1)}$ and they can be written as

$$Q_{1_+} = \begin{bmatrix} Q_1^T & \mathbf{q}_{n+1} \end{bmatrix} \qquad R_{1_+} = \begin{bmatrix} R_1 & \mathbf{z} \\ 0 & \rho \end{bmatrix}$$

where $\mathbf{q}_{n+1}$ is the first column of $Q_2\hat{H}_{n+1}^T$ while $\rho = \mathbf{sgn}(a_1)||\mathbf{a}||_2$.

Given the fact that an orthogonal matrix does not change the 2-norm of a vector, we can write

$$||A\mathbf{x} - \mathbf{b}||_2^2 = ||Q^T(A\mathbf{x} - \mathbf{b})||_2^2 = ||R\mathbf{x} - \begin{bmatrix} Q_1^T & Q_2^T \end{bmatrix}\mathbf{b}||_2^2$$

Since $R_1\mathbf{x} \in \mathbf{rank}(A)$, $Q_1^T\mathbf{b} \in \mathbf{rank}(A)$ and $Q_2^T\mathbf{b} \in \mathbf{ker}(A)$ we can write

$$||A\mathbf{x} - \mathbf{b}||_2^2 = ||R_1\mathbf{x} - Q_1^T\mathbf{b}||_2^2 + ||Q_2^T\mathbf{b}||_2^2 = ||R_1\mathbf{x} - \mathbf{z}||_2^2 + |\rho|^2$$

Then, the solution of the least square problem $\mathbf{min}\,||A\mathbf{x} - \mathbf{b}||_2$ is $\mathbf{x}^* = R_1^{-1}\mathbf{z}$ and thus

$$||A\mathbf{x}^* - \mathbf{b}||_2 = |\rho|$$

Problem 3

Task 1

If $A \in \mathbb{R}^{m\times n}$ is a matrix of rank n with singular value decomposition $A = U\Sigma V^T$ then Σ can be written as

$$\Sigma = \begin{bmatrix} \Sigma_r \\ 0 \end{bmatrix} \qquad \text{with} \qquad \Sigma_r = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix}$$

with $\sigma_1, \dots, \sigma_n$ singular values. This allows to write the following expressions in terms of the singular factors and vectors:

1. $$(A^T A)^{-1}$$

Since U is orthogonal, $A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T \Sigma V^T$. We can further simplify the expression by rewriting the product $\Sigma^T\Sigma$ as

$$\Sigma^T\Sigma \equiv \Sigma_r^2 = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{bmatrix}$$

Then the singular value decomposition of $(A^T A)^{-1}$ is

$$(A^T A)^{-1} = (V\Sigma_r^2 V^T)^{-1} = V(\Sigma_r^2)^{-1} V^T$$

with

$$(\Sigma_r^2)^{-1} = \begin{bmatrix} 1/\sigma_1^2 & & \\ & \ddots & \\ & & 1/\sigma_n^2 \end{bmatrix} \equiv \Xi$$

The singular values are $1/\sigma_n^2, \dots, 1/\sigma_1^2$

2. $$(A^TA)^{-1}A^T$$

Starting from the previous results, we write

$$(A^TA)^{-1}A^T = V\Xi V^T V\Sigma^T U^T = V\,\Xi\Sigma^T\,U^T$$

The matrix $\Xi\Sigma^T \in \mathbb{R}^{n\times m}$ is

$$\begin{bmatrix} 1/\sigma_1 & & & \\ & \ddots & & 0 \\ & & 1/\sigma_n & \end{bmatrix}$$

Then the singular values are $1/\sigma_n, \dots, 1/\sigma_1$

3. $$A(A^TA)^{-1}$$

Starting from the previous results, we write

$$A(A^TA)^{-1} = U\Sigma V^T V\Xi V^T = U\,\Sigma\Xi\,V^T$$

The matrix $\Sigma\Xi \in \mathbb{R}^{m\times n}$ is

$$\begin{bmatrix} 1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_n \\ & 0 & \end{bmatrix}$$

Then the singular values are $1/\sigma_n, \dots, 1/\sigma_1$

4. $$A(A^TA)^{-1}A^T$$

Starting from the previous results, we write:

$$A(A^TA)^{-1}A^T = U\Sigma V^T V\Xi V^T V\Sigma^T U^T = U\,\Sigma\Xi\Sigma^T\,U^T$$

The matrix $\Sigma\Xi\Sigma^T \in \mathbb{R}^{m\times m}$ is

$$\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$$

with the identity block of size n. The singular values are then all equal to 1.

Task 2

To compute the singular values of the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}$$

we compute the eigenvalues of AA^T:

$$\det(AA^T - \lambda I) = \det \begin{bmatrix} 5-\lambda & 4 \\ 4 & 4-\lambda \end{bmatrix} = \lambda^2 - 9\lambda + 4 = 0$$

The solutions, expressed in base 10 with 4 digits precision, are 8.531 and 0.4689. The singular values of A are the roots of the eigenvalues of AA^T, thus

$$\sigma_1 = 2.921 \qquad \sigma_2 = 0.6847$$

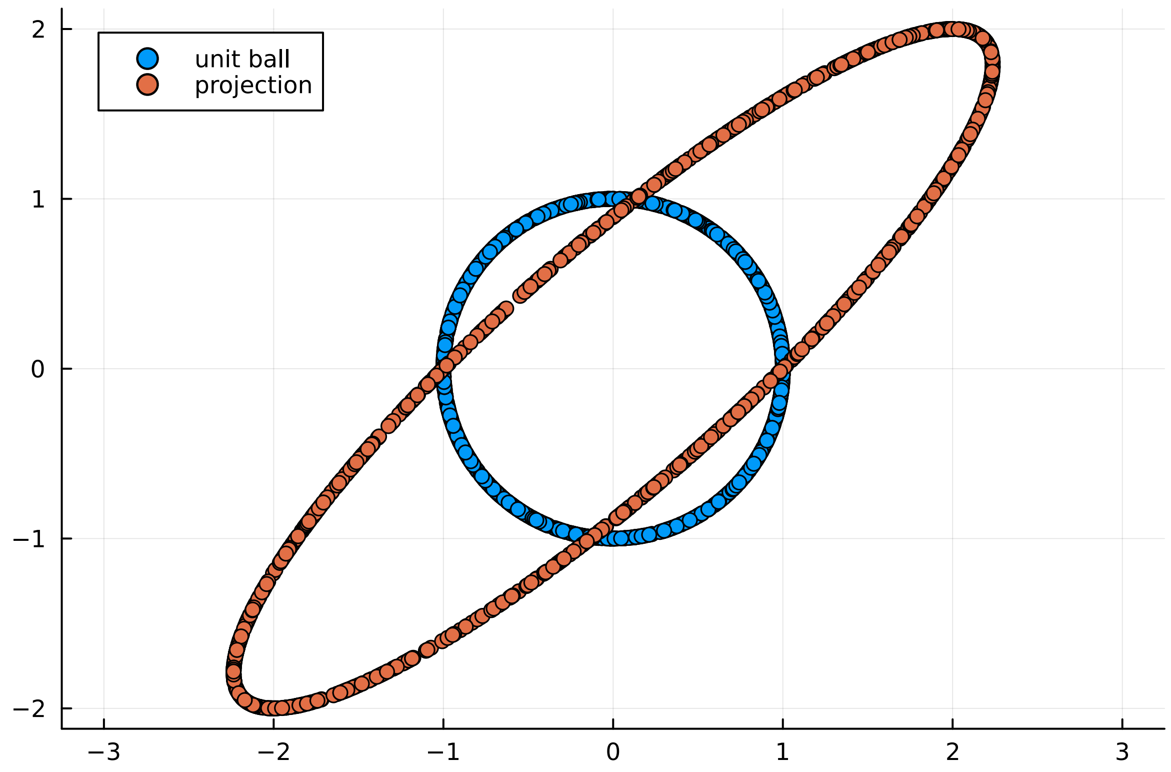
then the condition number is $\sigma_1/\sigma_2 = 4.266$. We can cross check our results using the functions of the Julia standard library:

```
▸ [2.92081, 0.684742]
```
```julia
svdvals([1 2; 0 2])
```

```
4.265564437074638
```
```julia
cond([1 2; 0 2])
```

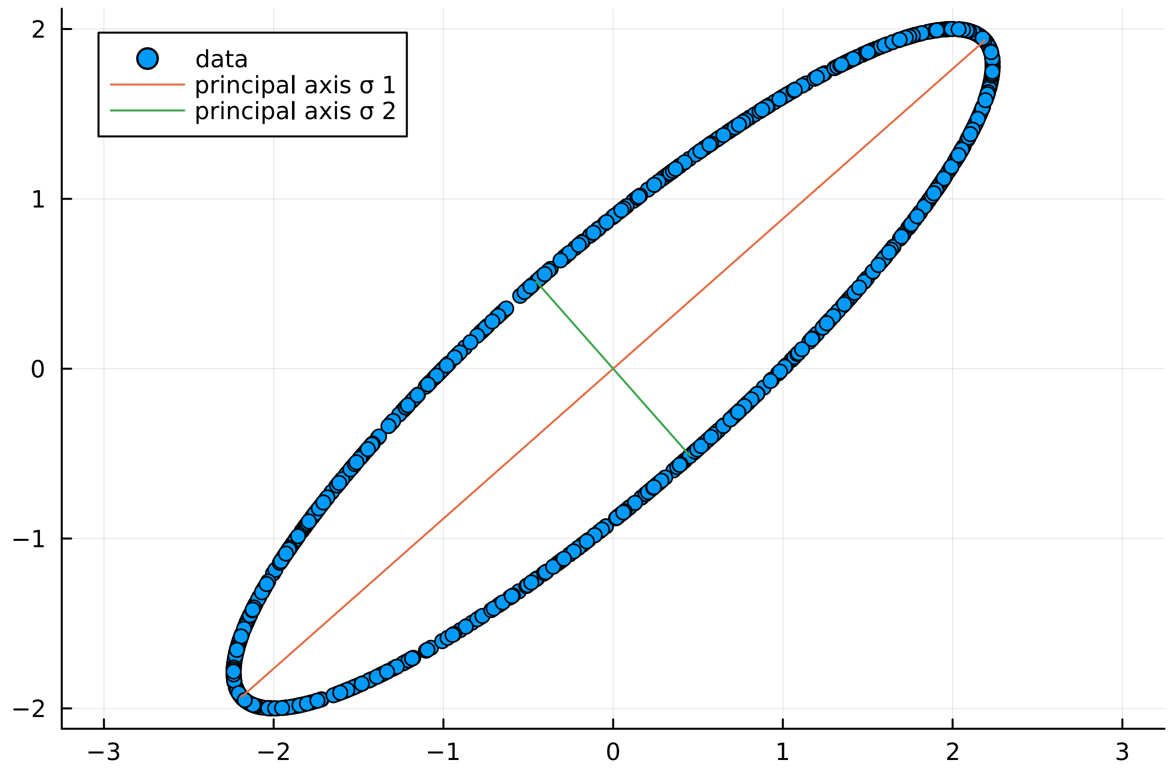
Here we plot in the 2d plane the effect of the matrix A applied to the unit ball. The projection is an ellypse where the principal axes are approximately 2.9 and 0.68 times the diameter of the unit ball.

This makes sense when we consider the singular value decomposition of $A = U\Sigma V^T$. When applied to a $\mathbb{R}^2$ vector, the orthogonal matrix V^T is a rotation, then the diagonal matrix Σ scales the rotated vector and U applies another rotation.



```julia
begin
    θ = 2 * π * rand(1000)
    ball = [cos.(θ) sin.(θ)]'
    proj = [1 2; 0 2] * ball
    projplot = scatter(ball[1,:], ball[2,:], label = "unit ball")
    scatter!(projplot, proj[1,:], proj[2,:], label = "projection")
    plot(projplot, aspect_ratio=:equal)
end
```

Given the fact that V^T has no graphical effect since it is applied to a circle, the tilt of the ellypse is given only by U. We can use this information to plot the principal axes on the plot and verify that our intuitive explanation is in fact correct.



```julia
begin
    U, Σ, _ = svd([1 2; 0 2])
    pa1 = Σ[1] * U * [1 -1; 0 0]
    pa2 = Σ[2] * U * [0 0; 1 -1]
    paplot = scatter(proj[1,:], proj[2,:], label="data")
    plot!(paplot, pa1[1,:], pa1[2,:], label="principal axis σ 1")
    plot!(paplot, pa2[1,:], pa2[2,:], label="principal axis σ 2")
    plot(paplot, aspect_ratio=:equal)
end
```

Task 3

The minimum norm solution $\hat{\mathbf{x}}$ for the problem $||\mathbf{b} - A\mathbf{x}||_2 = \mathbf{min}$, where

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

is given by

$$\hat{\mathbf{x}} = A^{+}\mathbf{b}$$

where A^{+} is the Moore–Penrose pseudoinverse. Given the SVD of $A = U\Sigma V^T$, the Moore–Penrose pseudoinverse is given by $V\Sigma^{+}U^T$, where Σ^{+} is given by replacing every nonzero diagonal element of Σ by its reciprocal.

Thus we can use the Julia standard library functions to find the minimum norm solution of our problem:

```
▸ [2.13974e-16, 1.0, 1.0]
```

```julia
begin
    # FROM THE svd() FUNCTION DOCUMENTATION:
    #
    # U, S, V and Vt can be obtained from the factorization
    # F with F.U, F.S, F.V and F.Vt, such that A = U * Diagonal(S) * Vt.
    #
    # If full = false (default), a "thin" SVD is returned.
    # For an M \times N matrix A, in the full factorization
    # U is M \times M and V is N \times N, while in
    # the thin factorization U is M \times K and V
    # is N \times K, where K = \min(M,N) is the number of singular values.
    D = svd([1 1 0; 0 1 1])
    x̂ =  D.V * Diagonal(1 ./ D.S) * D.U' * [1, 2]
end
```

Task 4

Using the features of the Julia `LinearAlgebra` standard library, we compute the SVD decomposition, the condition number, the rank and the Moore–Penrose pseudoinverse of the following matrix:

```
B = 3×4 Matrix{Int64}:
       -2    -4    -2    -4
        2    -2     2     1
     -800   200  -800  -401
```

The SVD decomposition is:

```
SVD{Float64, Float64, Matrix{Float64}, Vector{Float64}}
U factor:
3×3 Matrix{Float64}:
  0.00270396   -0.97275      -0.231839
 -0.00270189   -0.231847      0.972749
  0.999993      0.00200387    0.00325517
singular values:
3-element Vector{Float64}:
 1216.8905608474079
    5.549627759643728
    0.751366894446353
Vt factor:
4×4 Matrix{Float64}:
 -0.657417     0.164348     -0.657417    -0.329537
 -0.0218556    0.856899     -0.0218556    0.514557
 -0.25947     -0.488584     -0.25947      0.791603
 -0.707107    -2.41474e-14   0.707107     3.66374e-14
```

```julia
begin
    F = svd(B)
    svd(B, full=true)
end
```

There are three nonzero singular values, hence the matrix is of rank 3, as we can cross-check:

```
true
```

```julia
rank(B) == 3
```

The Moore–Penrose pseudoinverse:

```
B⁺ = 4×3 Matrix{Float64}:
       0.0838907    -0.335006  -0.00167224
       0.000557414  -0.668339  -0.00167224
       0.0838907    -0.335006  -0.00167224
      -0.334448      1.00334    0.00334448
```

```
B⁺ = F.V * Diagonal(1 ./ F.S) * F.U'
```

As we can cross-check:

```
true
```

```
pinv(B) ≈ B⁺
```

The condition number, defined as the ratio of the greatest and smallest singular value, is:

```
κ_B = 1619.5690412259612
```

```
κ_B = F.S[1] / F.S[end]
```

As we can cross check:

```
true
```

```
cond(B) ≈ κ_B
```

Task 5

We want to find the best rank 1 and rank 2 approximation of `B`. To do so, we use the singular value decomposition. In fact, we know that if $U\Sigma V^T$ is the SVD decomposition of a matrix B, then $U\Sigma_k V^T$ (with Σ_k being the rectangular matrix were only the first k singular values are taken) is the best rank k approximation of B.

Thus, the best rank 1 approximation is:

```
B_1 = 3×4 Matrix{Float64}:
         -2.16318     0.540774    -2.16318    -1.08432
          2.16152    -0.540359     2.16152     1.08349
       -799.999     199.992     -799.999    -401.008
```

```
B_1 = F.U * Diagonal([F.S[1], 0, 0]) * F.Vt
```

The best rank 2 approximation is:

```
B_2 = 3×4 Matrix{Float64}:
         -2.0452    -4.08511    -2.0452     -3.86211
          2.18964   -1.6429      2.18964     0.421424
       -799.999     200.001    -799.999    -401.002
```

```
B_2 = F.U * Diagonal([F.S[1], F.S[2],0]) * F.Vt
```

With condition number

```
219.27426731149427
```

```julia
F.S[1]/F.S[2]
```

As expected, the rank 2 approximation has a better condition number than the full rank decomposition, which is one of the reasons to approximate the matrix in the first place.

Task 6

First, we write the function to generate the upper triangular matrix $R = (r_{ij})$ with $r_{ii} = 1$ and $r_{ij} = -1$ for $j > i$:

```
generate_R (generic function with 1 method)
```

```julia
function generate_R(n)
    R = [j > i ? -1 : 0 for i in 1:n, j in 1:n]
    for i in 1:n
        R[i,i] = 1
    end
    return R
end
```

For example, the matrix R of size 8 is:

```
8×8 Matrix{Int64}:
 1  -1  -1  -1  -1  -1  -1  -1
 0   1  -1  -1  -1  -1  -1  -1
 0   0   1  -1  -1  -1  -1  -1
 0   0   0   1  -1  -1  -1  -1
 0   0   0   0   1  -1  -1  -1
 0   0   0   0   0   1  -1  -1
 0   0   0   0   0   0   1  -1
 0   0   0   0   0   0   0   1
```

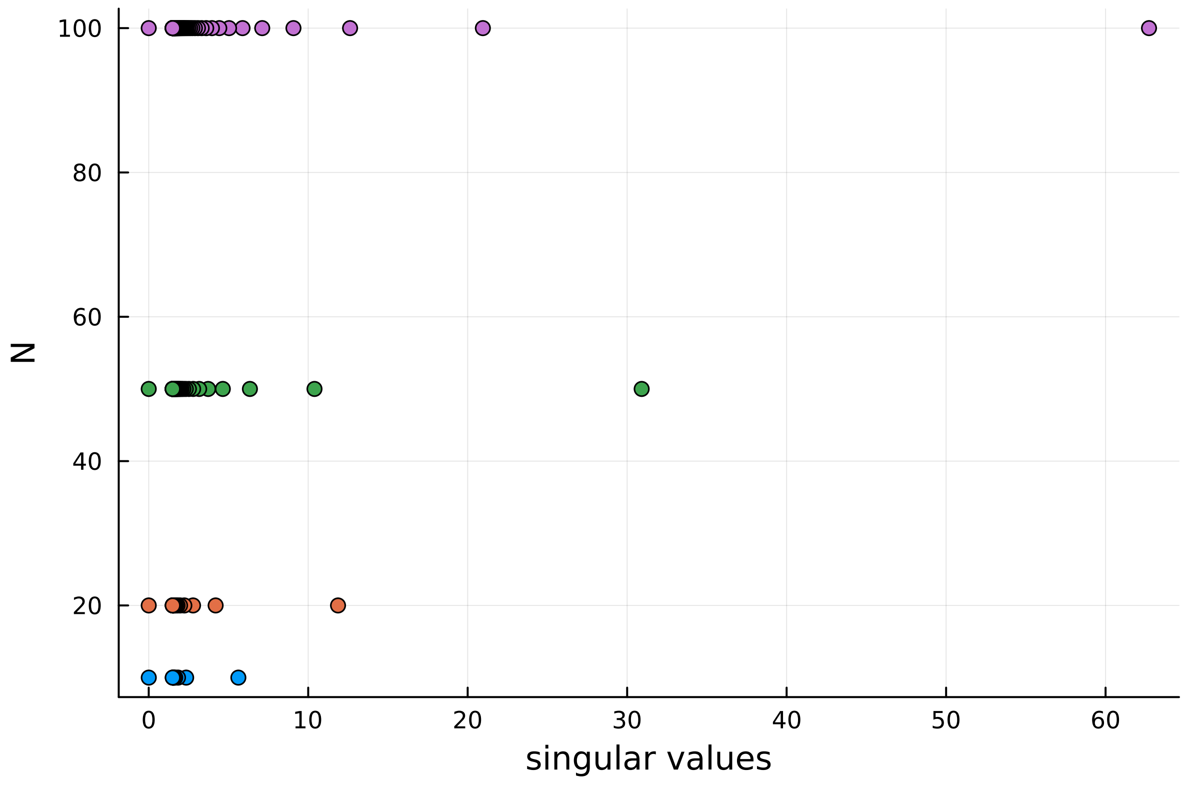
```julia
generate_R(8)
```

The columns of R are linear independent, hence the matrix has full rank.

Next, we evaluate the singular values of the matrices R of size 10, 20, 50 and 100:

```julia
begin
    Ns = [10,20,50,100]
    Ss = []
    for n in Ns
        push!(Ss, svdvals(generate_R(n)))
    end
end
```

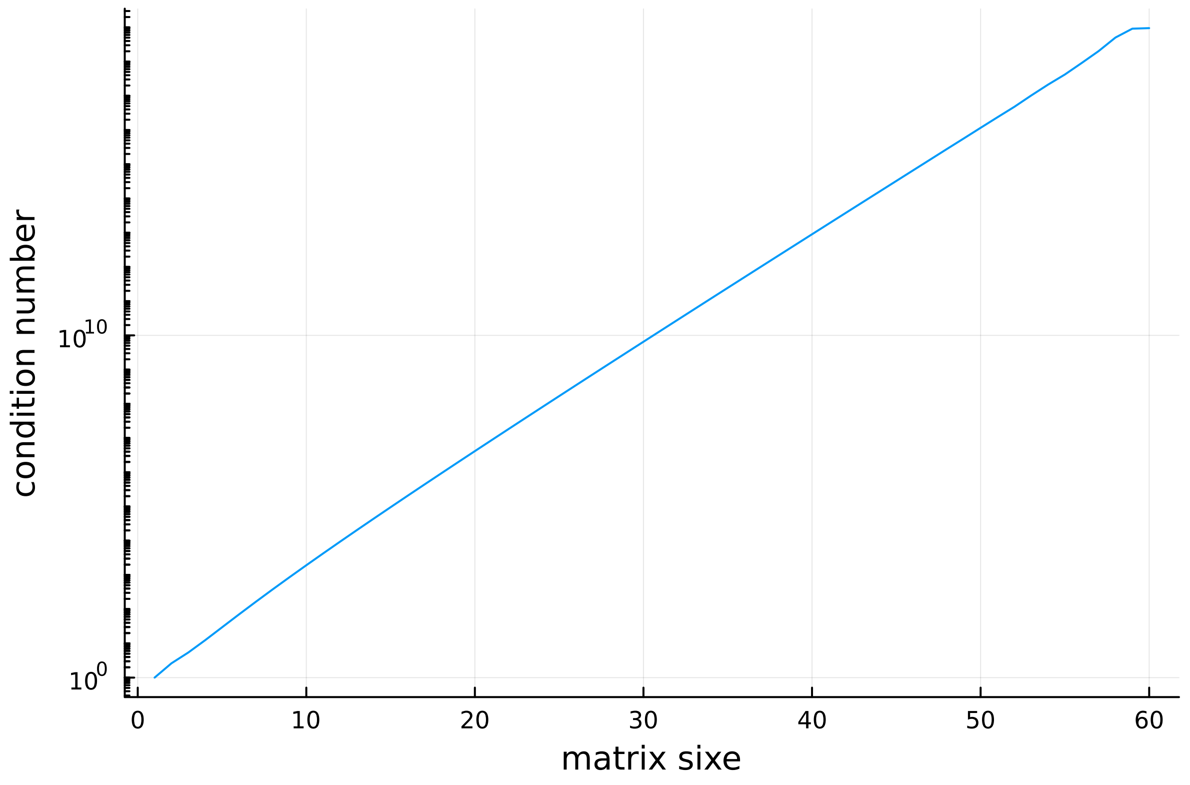
Here we plot the results:



```
begin
    scp = scatter(Ss[1], [Ns[1] for _ in Ss[1]])
    for (s,n) in zip(Ss[2:end], Ns[2:end])
        scatter!(scp, s, [n for _ in s])
    end
    plot(scp, legend=false, xlabel="singular values", ylabel="N")
end

```

We see that the greatest singular value depends on the dimension of the matrix. Here we plot the condition number (defined as the ratio of the greatest and smallest non-zero singular value) of R_n with n from 2 to 60, to check its dependency on the matrix size as well:



```julia
begin
    κs = []
    for n in 1:60
        S = svdvals(generate_R(n))
        S = S[S.>0]
        append!(κs, S[1]/S[end])
    end
    plot(1:60, κs, yscale=:log10, legend=false, xlabel="matrix sixe",
    ylabel="condition number", yminorticks=true)
end
```

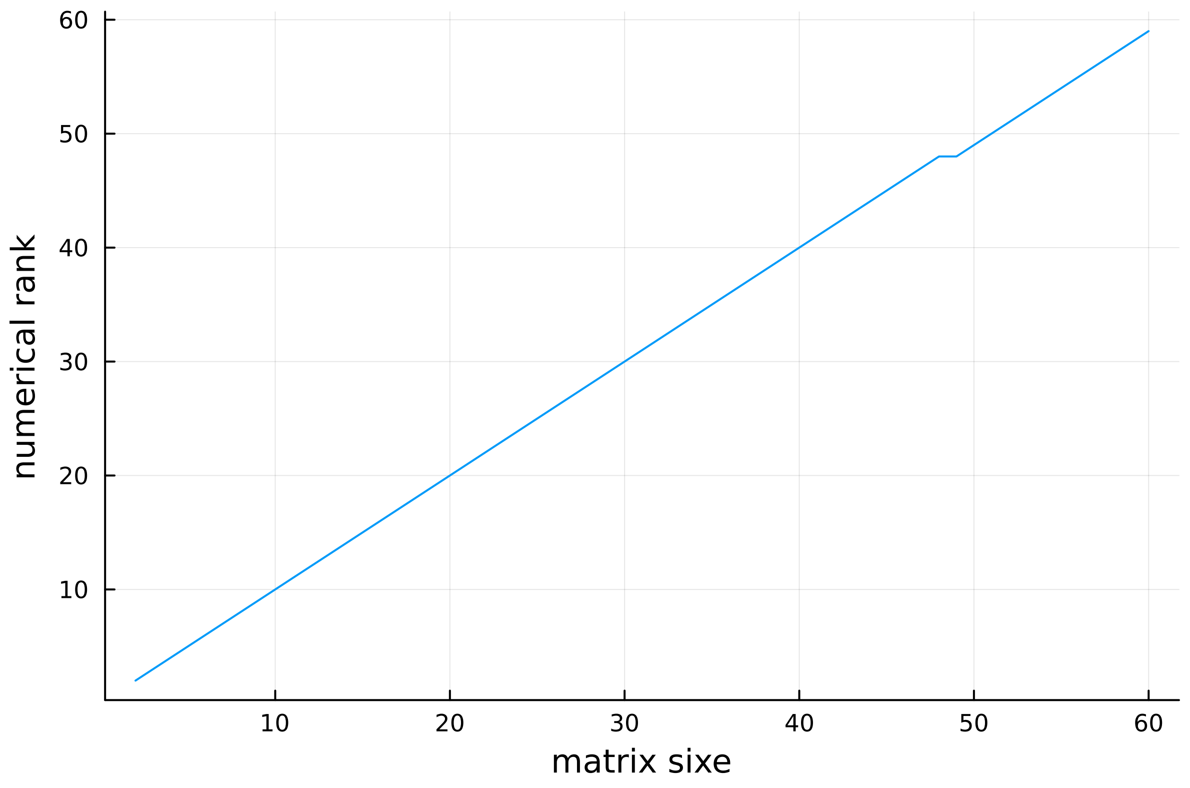
We see that dependency is roughly exponential. This means that, above a certain dimension, the condition number will be greater than the inverse of the machine precision, hence the matrix will be not numerically full rank.

Here we define a funcion `ϵ_rank()` to compute the numerical rank of a matrix by counting how many singular values are equal or greater than the machine epsilon times the greatest singular value:

```
ϵ_rank (generic function with 1 method)
```

```julia
function ϵ_rank(A)
    S = svdvals(A)
    T = typeof(S[1])
    ϵ = eps(T)
    rank_threshold = ϵ * S[1]
    return count(x -> x >= rank_threshold, S)
end
```

And then we can verify our statements on the rumerical rank of R_n



```julia
begin
    num_ranks = []
    matrices = 2:60
    for n in matrices
        num_rank = ϵ_rank(generate_R(n))
        if num_rank < n
            @show n, num_rank
        end
        append!(num_ranks, num_rank)
    end
    plot(matrices, num_ranks, legend=false, xlabel="matrix sixe",
    ylabel="numerical rank")
end
```

```
(n, num_rank) = (49, 48)
(n, num_rank) = (50, 49)
(n, num_rank) = (51, 50)
(n, num_rank) = (52, 51)
(n, num_rank) = (53, 52)
(n, num_rank) = (54, 53)
(n, num_rank) = (55, 54)
(n, num_rank) = (56, 55)
(n, num_rank) = (57, 56)
(n, num_rank) = (58, 57)
(n, num_rank) = (59, 58)
(n, num_rank) = (60, 59)
```