HW03 QR, SVD, Least Squares and Iterative Methods

Problem 1 (3 points)

(Analytical) Show that the 2-norm condition number of an orthonormal matrix is 1.

Recall that a very nice property an orthonormal matrix Q is that $Q^{-1} = Q^T$. One way to approach doing this problem is to first show that $\|Q\|_2 = 1$. To do this, recall the definition of an induced matrix norm and then recall that the square of the vector 2-norm is the inner product, $\|x\|_2^2 = x^T x$.

Solution

First, we consider the definition of the induced matrix 2-norm: $||A||_2 = \sup_{x \neq 0} \frac{||Ax||_2}{||x||_2}$

Consider squaring both sides of this equations, thus $||A||_2^2 = \sup_{x \neq 0} \frac{||Ax||_2^2}{||x||_2^2}$

The square of the vector 2-norm is the inner product ($||x||_2^2 = x^T x$), thus we have $||A||_2^2 = \sup_{x \neq 0} \frac{(Ax)^T Ax}{x^T x} = \sup_{x \neq 0} \frac{x^T A^T Ax}{x^T x}$

Now consider the square of the 2-norm of an orthonormal matrix Q, we have $||Q||_2^2 = \sup_{x \neq 0} \frac{x^T Q^T Q x}{x^T x}$

Since Q is orthonormal, $Q^T Q = 1$, thus we have $||Q||_2^2 = \sup_{x \neq 0} \frac{x^T x}{x^T x} = 1$

Thus, $||Q||_2 = \pm 1$, however the value of a matrix norm must be greater than or equal to 0, therefore $||Q||_2 = 1$ for orthonormal Q

The 2-norm condition number of Q, $\kappa(Q)$ is $||Q||_2 ||Q^{-1}||_2 = ||Q||_2 ||Q^T||_2$

Consider that since $Q^{-1} = Q^T$, we also have $(Q^T)^{-1} = (Q^T)^T = Q$

Thus, by substituting Q^T into the fourth equation, we have $||Q^T||_2^2 = \sup_{x \neq 0} \frac{x^T (Q^T)^T Q^T x}{x^T x}$

Thus, since $(Q^T)^T = (Q^T)^{-1}$, we have $||Q^T||_2^2 = \sup_{x \neq 0} \frac{x^T x}{x^T x} = 1$

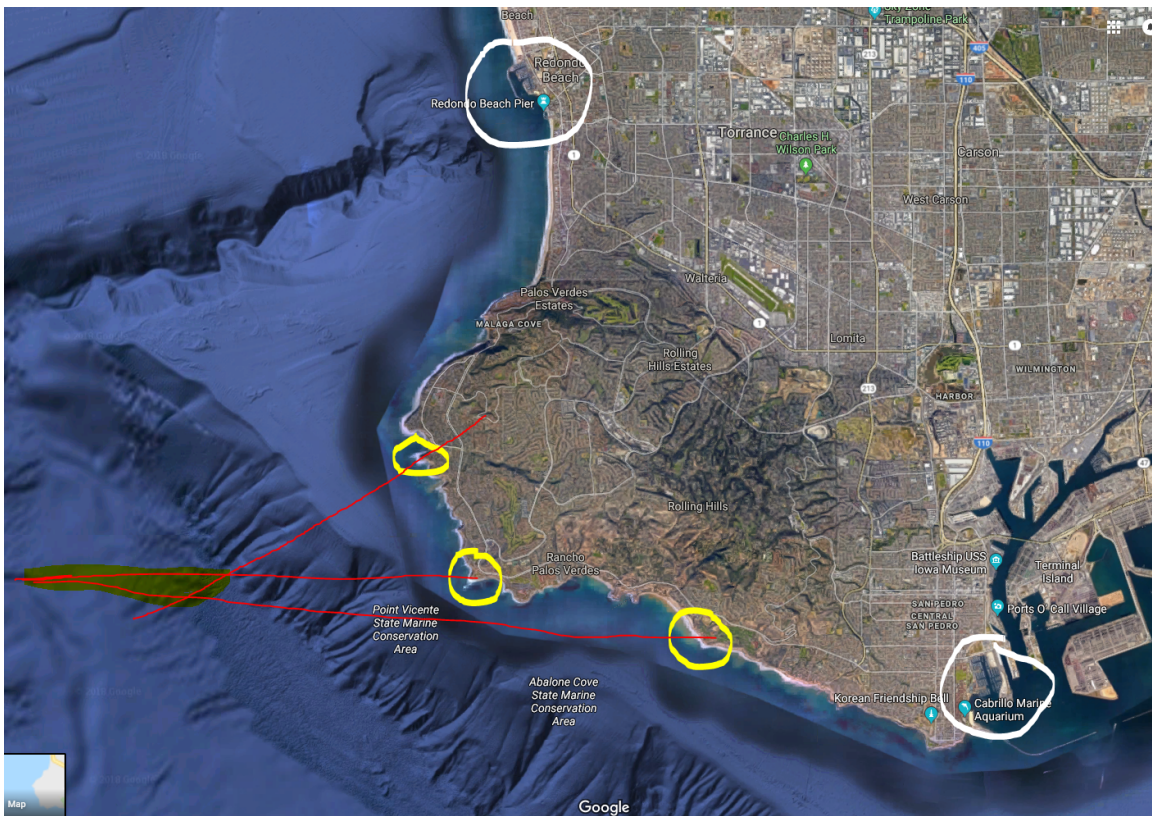
Thus $||Q^T||_2 = 1$

Thus, $\kappa(Q) = ||Q||_2 ||Q^T||_2 = 1 \times 1 = 1$

Problem 2 (2 points)

(In Julia) An otherwise nice day out sailing off the coast of the Palos Verdes Peninsula starts to go poorly when your GPS and radio go out, and your boat starts taking on water from a crack near the keel. You need to get to a port sooner rather than later, since losing your sloop and having to row your rubber dinghy back to shore would be less fun than just getting back in to harbor. Fortunately you have a chart (aka map) and are able to make out the Point Vicente Lighthouse, Resort Point, and a really big golf course on a bluff. You know the two nearest harbors are King's Harbor in Redondo Beach, and Cabrillo Marina in San Pedro.

When you draw the three triangulation lines, they don't all go through the same point since there are errors in how you measure things on a sinking boat using binoculars and a cracked plastic protractor. Instead you get a triangle. You use the center of the triangle to decide to go back to King's Harbor.



(Image from Google Maps 2019. Reproduced here under their terms of use.)

When you get back, you want to check mathematically how much better your choice was, because why wouldn't you? The equations for triangulation lines on a plane have the general form

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

In your case, you measured out the angles on your compass, relative to your boat's supposed position, to get the lines

$$\begin{bmatrix} 0 & 1 \\ 1 & 10 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 8 \end{bmatrix}$$

Use QR decomposition and SVD to get the least-squares solution to this problem.

Solution

QR decomposition:

In [1]:
```
A = [0 1; 1 10; 3 -2]
b = [0;0;8]
```

Out[1]:
```
3-element Vector{Int64}:
 0
 0
 8
```

First, we get the object containing our Q and R matrix

In [2]:
```
using LinearAlgebra
F = qr(A)
```

Out[2]:
```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
  0.0       -0.0983422  -0.995153
 -0.316228  -0.944085    0.0932956
 -0.948683   0.314695   -0.0310985
R factor:
2×2 Matrix{Float64}:
 -3.16228   -1.26491
  0.0      -10.1686
```

Then, we extract the matrices

In [3]:
```
Q = Matrix(F.Q)
```

Out[3]:
```
3×2 Matrix{Float64}:
  0.0       -0.0983422
 -0.316228  -0.944085
 -0.948683   0.314695
```

In [4]:
```
R = F.R
```

Out[4]:
```
2×2 Matrix{Float64}:
 -3.16228   -1.26491
  0.0      -10.1686
```

$||Ax - b||_2$ is minimized when $||Rx - Q^T b|| = 0$, thus $x = R^{-1}Q^T b$

In [5]:
```
x = inv(R) * Q' * b
```

Out[5]:
```
2-element Vector{Float64}:
  2.4990328820116052
 -0.24758220502901365
```

SVD:

First we must prove that A has 2 linearly independent column vectors. We can do this by solving $Ax = 0$. If the only solution is $x = 0$, they are linearly independent

$$\begin{bmatrix} 0 & 1 \\ 1 & 10 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

Thus, we have $0x_1 + x_2 = 0 \Rightarrow x_2 = 0$

Then, we have $x_1 + 10x_2 = 0 \Rightarrow x_1 = 0$

Thus, the only solution is $x = 0$, and A has two linearly independent column vectors. Now we can do SVD

In [6]:
```
U, S, V = svd(A)
```

Out[6]:
```
SVD{Float64, Float64, Matrix{Float64}, Vector{Float64}}
U factor:
3×2 Matrix{Float64}:
 -0.097426  -0.0133927
 -0.978354   0.184712
  0.182567   0.982701
singular values:
2-element Vector{Float64}:
 10.255151072551799
  3.1355823189863234
Vt factor:
2×2 Matrix{Float64}:
 -0.0419938  -0.999118
  0.999118   -0.0419938
```

In [7]:
```
y = V * inv(diagm(S)) * U' * b
```

Out[7]:
```
2-element Vector{Float64}:
  2.4990328820116057
 -0.24758220502901374
```

Just to check, let's make sure our QR and SVD results are similar enough.

In [8]:
```
x - y
```

Out[8]:
```
2-element Vector{Float64}:
 -4.440892098500626e-16
  8.326672684688674e-17
```

Problem 3 (3 points)

(By hand) Without using electronic implements, take two steps of the Jacobi iteration for the problem

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

beginning with $x = \vec{0}$. Typeset your results.

Solution

We have $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Our first Jacobi iteration then looks like:

$$\begin{bmatrix} \frac{1}{2}(1-0-0) \\ \frac{1}{2}(1-0-0) \\ \frac{1}{2}(1-0-0) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

Then the second Jacobi iteration then looks like:

$$\begin{bmatrix} \frac{1}{2}(1-(-1)\frac{1}{2}-0) \\ \frac{1}{2}(1-(-1)\frac{1}{2}-(-1)(\frac{1}{2}) \\ \frac{1}{2}(1-0-(-1)(\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} \frac{3}{4} \\ 1 \\ \frac{3}{4} \end{bmatrix}$$

Problem 4 (2 points)

(In Julia) Use Gauss-Seidel iteration from the `IterativeSolvers.jl` package to solve $Ax = b$, $A \in \mathbb{R}^n$ for

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

and

$$b = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Take $n = 100$. Think about what you can use in Julia to construct the matrix more easily.

Solution

In [9]:
```
l = repeat([-1], 99)
m = repeat([2], 100)
n = repeat([-1], 99)
b = repeat([1], 100)
```

Out[9]:
```
100-element Vector{Int64}:
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 ⋮
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
```

In [10]:
```
using LinearAlgebra
A = Tridiagonal(l,m,n)
```

Out[10]:
```
100×100 Tridiagonal{Int64, Vector{Int64}}:
  2  -1   ·   ·   ·   ·   ·   ·   ·  …   ·   ·   ·   ·   ·   ·   ·   ·   ·
 -1   2  -1   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·  -1   2  -1   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·  -1   2  -1   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·  -1   2  -1   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·  -1   2  -1   ·   ·  …   ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·  -1   2  -1   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·  -1   2  -1      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·  -1   2      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·  -1      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·  …   ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ⋮                   ⋮               ⋱                   ⋮
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·  …  -1   ·   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      2  -1   ·   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·     -1   2  -1   ·   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·  -1   2  -1   ·   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·  -1   2  -1   ·   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·  …   ·   ·   ·  -1   2  -1   ·   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·  -1   2  -1   ·   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·  -1   2  -1   ·
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·  -1   2  -1
  ·   ·   ·   ·   ·   ·   ·   ·   ·      ·   ·   ·   ·   ·   ·   ·  -1   2
```

In [11]:
```
import Pkg; Pkg.add("IterativeSolvers")
using IterativeSolvers
x = gauss_seidel(A, b)
```

```
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.9/Project.toml`
 No Changes to `~/.julia/environments/v1.9/Manifest.toml`
```

```
Out[11]: 100-element Vector{Float64}:
           2.7001380920410156
           4.736652374267578
           6.249699592590332
           7.357873916625977
           8.158591985702515
           8.729787111282349
           9.13234469294548
           9.412824153900146
           9.606145061552525
           9.738037623465061
           9.82715624012053
           9.886825483292341
           9.926433119457215
           ⋮
          10.0
          10.0
          10.0
           9.998046875
           9.984375
           9.93212890625
           9.78759765625
           9.4635009765625
           8.837646484375
           7.757293701171875
           6.047882080078125
           3.5239410400390625
```