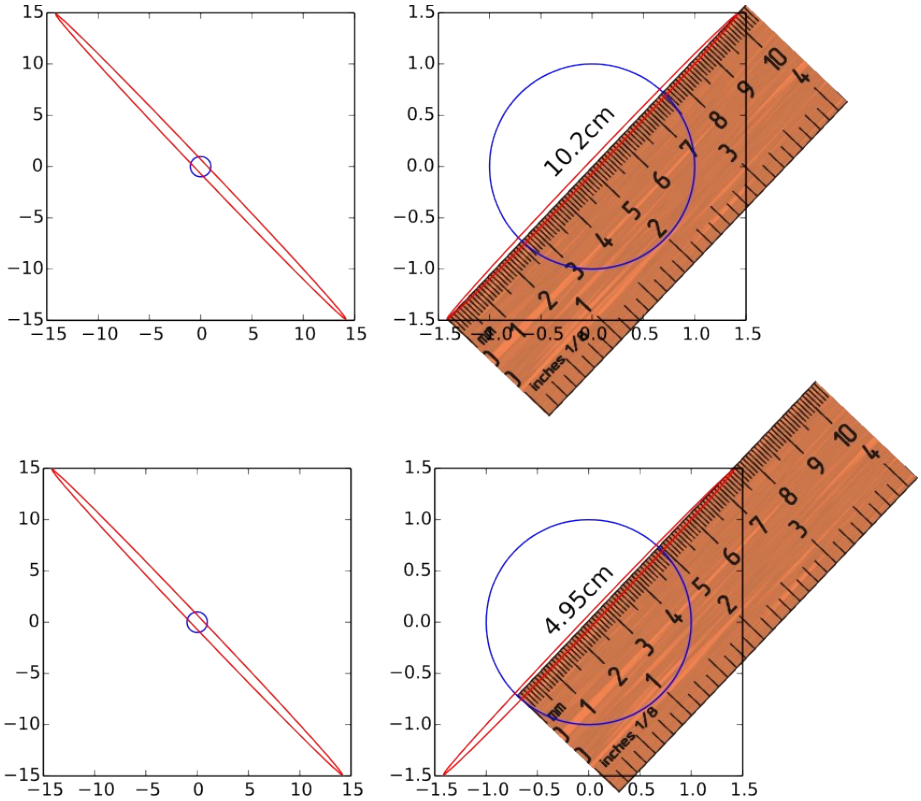
1.

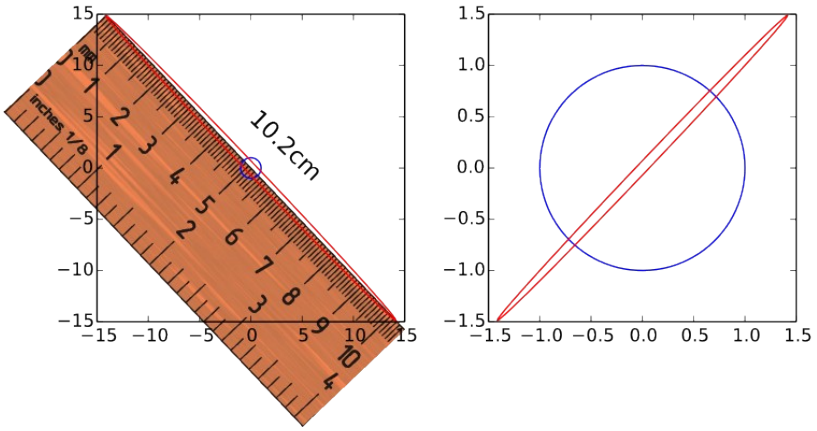
Measure the 2-norm of A and A^{-1} on your plot **using a ruler**, and use these two numbers to estimate cond A.

Make it clear how you obtained your answers.



Using the ruler measurements:

||A|| ~ 10.2cm/4.95cm
= 2.06



||Ainv|| ~ 10.2cm/0.5cm
= 20.4

Then
cond A = ||A|| ||Ainv||

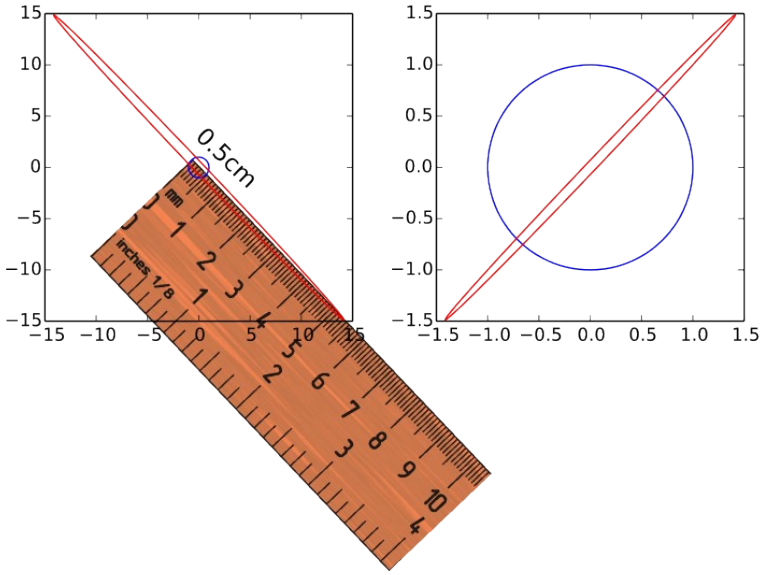
~ (2.06)(20.4)

= 42

BTW, the matrix is

1 1

1.1 1




```
A = np.array([[1,1],[1.1,1]])
Ainv = np.linalg.inv(A)
print( np.linalg.norm(A) )
2.05182845287
print( np.linalg.norm(Ainv) )
20.5182845287
print( np.linalg.cond(A) )
42.0762336142
```

Agreement is excellent.

2.

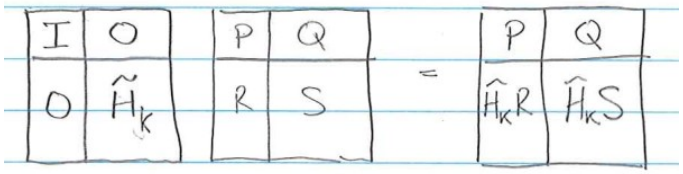
In solving a linear system Ax=b with finite-precision arithmetic, we have a (Wilkinson) bound on the norm of error relative to the solution: ||error|| / ||x||. For example, in Homework 4, Question 4, I found that the relative error bound with ϵmach = 2^{−52} for systems with Hilbert matrix H3 is about 2 × 10^{−11}.

This does not mean that each component of the solution has this relative accuracy.

Provide an example of a pair of 3-vectors x and an approximation $\tilde{x}$, for which the norm of the difference relative to the norm of x is 1% while there is a 100% error in one of the components.

$$
\begin{gathered}
x = [1, .01, 1] \\
\tilde{x} = [1.01, .02, 1.005] \\
\frac{||\tilde{x} - x||_\infty}{||x||_\infty} = \frac{.01}{1} = 1\% \\
\frac{|\tilde{x}_2 - x_2|}{|x_2|} = \frac{.01}{.01} = 100\%
\end{gathered}
$$

3 a.



b. Arithmetic operation count for more economical Householder QR

Householder QR factorization - more economical version

```
import numpy as np
norm = np.linalg.norm  # by default, 2-norm

# np.random.seed(123) # get same random matrix each time for t

def HQRstep(A):
    # modify A i.e. replace it by HA, and return v
    m = A.shape[0] # #rows of A
    z = A[:,0]
    w = np.zeros(m) # start with all zeros for w
    w[0] = norm(z) if z[0]<0 else -norm(z)
    v = w - z
    vTv = np.dot(v,v)
    A[:,:] -= np.outer((2/vTv)*v,np.dot(v,A))# overwrites A wi
    return v

def HQR(A):  # the whole process (all columns)
    # modify A and return Q
    m,n = A.shape
    H = np.eye(m)
    for j in range(n):  # for each column in turn
        v = HQRstep(A[j:,j:])
        vTv = np.dot(v,v)
        leftvector = (2/vTv)*v
        H[j:,:j] -= np.outer(leftvector,np.dot(v,H[j:,:j]))
        H[j:,j:] -= np.outer(leftvector,np.dot(v,H[j:,j:]))
    Q = H.T # transpose of H
    return Q

# Try it out

m,n = 5,3
A = np.random.randn(m,n)
Acopy = A.copy()
Q = HQR(A)

def round(x): return np.round(x,4)

print('Q')
print(round(Q))
print('R')
print(round(A))  # note A has now been overwritten with R
```

Arithmetic operation count

```
import sympy as sp

def nops_norm(M): return 2*M + 1
def nops_dot(M,N): # dot of M-vector with MxN matrix
    return 2*M*N
def nops_outer(M,N): return M*N

def nops_HQRstep(M,N):
    nops = 0
    nops += nops_norm(M)    # line 11
    nops += M               # line 12
    nops += nops_dot(M,1)   # line 13
    nops += M+1             # line 14 left argument
    nops += nops_dot(M,N)   # line 14 right argument
    nops += nops_outer(M,N) # line 14
    nops += M*N             # line 14 -=
    return nops

M,N = sp.symbols('M,N')
print('operations for HQRstep(M,N):')
display( nops_HQRstep(M,N) )
```

operations for HQRstep(M,N):

$$4MN + 6M + 2$$

```
def nops_HQR(m,n):
    def nops_loop(j):
        M,N = m-j,n-j
        nops = 0
        nops += nops_HQRstep(M,N)  # line 22
        nops += nops_dot(M,1) # line 23
        nops += 1 + M  # line 24
        nops += nops_dot(M,j) # line 25 right argument
        nops += nops_outer(M,j) # line 25
        nops += M*j  # line 25 -=
        nops += nops_dot(M,N) # line 26 right argument
        nops += nops_outer(M,N) # line 26
        nops += M*N  # line 26 -=
        nops = sp.expand(nops)
        return nops
    j = sp.symbols('j')
    nops_loop_j = nops_loop(j) # create an expression
    return sp.expand( sp.Sum( nops_loop_j, (j,0,n-1)).doit() )

m,n = sp.symbols('m,n')
print('operations for HQR(m,n)')
nops_HQR(m,n)
```

operations for HQR(m,n)

answer:

$$6mn^2 + 11mn - \frac{8n^3}{3} - \frac{5n^2}{2} + \frac{49n}{6}$$

```
nops_HQR(m,m) # what if applied to a square mxm matrix?
```

$$\frac{10m^3}{3} + \frac{17m^2}{2} + \frac{49m}{6}$$