1. **[Rounding errors, 1+2+3pt]** Computers use finite precision to represent real numbers, which leads to rounding. You can see the size of the rounding error for real numbers around 1 using the MATLAB command `eps(1)` or the numpy command `np.spacing(1)`. This number, also called the *machine epsilon*, is $\epsilon = 2.22 \times 10^{-16} \ldots$ for the standard (double precision) representation of numbers in a computer. Try the following experiments in MATLAB (or Python/Octave/Julia).[1]

   (a) Report the analytical/exact result, and the result you get when using your computer for:

   $$a = (1-1) + 10^{-16}, \quad b = 1 - (1 + 10^{-16}).$$

   What do you think is happening?

   (b) The n-th Hilbert matrix $H_n \in \mathbb{R}^{n \times n}$ has the entries $h_{ij} = (i+j-1)^{-1}$ for $i, j = 1, \ldots, n$.[2] It is known that solving systems with the Hilbert matrix increases rounding errors since the matrix is poorly conditioned. Let $\boldsymbol{e}_n$ be the column vector of length n that contains all 1's. Report the exact and the numerically computed values for

   $$\|H_n(H_n^{-1}\boldsymbol{e}_n) - \boldsymbol{e}_n\|, \text{ for } n = 5, 10, 20.$$

   Here, $\|\cdot\|$ is the usual Euclidean norm. Also report the condition numbers of H_n (with respect to either norm) for $n = 5, 10, 20$.

   (c) As you know, for differentiable functions f holds

   $$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

   Thus, to numerically approximate a derivative of a function for which the derivative is hard to derive analytically, one can use, for a small h, the approximation

   $$f'(x_0) \approx f'_{\text{num}}(x_0) := \frac{f(x_0 + h) - f(x_0)}{h}.$$

   In this approximation, one subtracts very similar numbers in the numerator of the fraction from each other, and then multiplies with a large number (namely h^{-1}), which can lead to errors that are much larger than the machine epsilon. Compute approximations of the derivative of the function

   $$f(x) = \frac{\exp(x)}{\cos(x)^3 + \sin(x)^3}$$

   at $x_0 = \pi/4$. In order to do so, use progressively smaller perturbations $h = 10^{-k}$ for $k = 1, \ldots 16$. Present the errors in the resulting approximation in a log-log plot, i.e., use a logarithmic scale to plot the values of h on the x-axis, and a logarithmic scale to plot the errors between the finite difference approximation $f'_{\text{num}}(x_0)$ and the exact value, which is $f'(x_0) = 3.101766393836051$, on the y-axis.[3] What do you observe as h becomes smaller, and for which h do you get the best approximation to the derivative?

[1] You can use the command `format long` to get 15 digits output from your computer. If you need more digits of a number a, you can use `fprintf('%2.20f\n',a)` to see 20 digits.

[2] You can get H_n by using the MATLAB command `hilb(n)`.

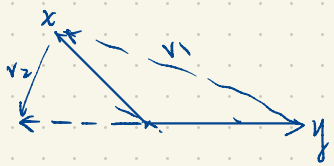
[3] MATLAB offers a `loglog` function to do that.

See Code file.

2. **[Orthongonalization and least squares, 2+3pt]**.

(a) Given any two nonzero vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^n$, construct a Householder matrix H, such that $H\boldsymbol{x}$ is a scalar multiple of $\boldsymbol{y}$. Is the matrix H unique?

(a) Generally, Not Unique.



but $H_v := I - 2\frac{v\,v^T}{v^T\cdot v}$

Generally Speaking.

we can choose, $v_1 = x - \frac{y}{\|y\|}\|x\|$

or $v_2 = x + \frac{y}{\|y\|}\|x\|$.

Then. Both H_{v_1} and H_{v_2} do the job.

They're different as $H_{v_1}(x) = -H_{v_2}(x)$

(b) Use Householder matrices to compute the QR-factorization of the matrix:

$$X = \begin{bmatrix} 9 & -6 \\ 12 & -8 \\ 0 & 20 \end{bmatrix}.$$

Write down both formulations we discussed in class, i.e., $A = \hat{Q}\hat{R}$ with $\hat{Q} \in \mathbb{R}^{m\times n}$, $\hat{R} \in \mathbb{R}^{n\times n}$ as well as $A = QR$ with $Q \in \mathbb{R}^{m\times m}$, $\hat{R} \in \mathbb{R}^{m\times n}$.

(b)

Step 1. Convert $x_1 = \begin{bmatrix} 9 \\ 12 \\ 0 \end{bmatrix}$ to $e_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$\|x_1\| = 15$

but $v_1 = \|x_1\| - 15\,e_1 = \begin{bmatrix} -6 \\ 12 \\ 0 \end{bmatrix}$ $\qquad \begin{bmatrix} -6 \\ 12 \\ 0 \end{bmatrix}\begin{bmatrix} -6 & 12 & 0 \end{bmatrix}$

$$H_{v_1} = I - 2\frac{v\cdot v^T}{v^T v} = I - 2\frac{1}{180}\begin{bmatrix} 36 & -72 & 0 \\ -72 & 144 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= I - 2\cdot\begin{bmatrix} \frac{1}{5} & -\frac{2}{5} & 0 \\ -\frac{2}{5} & \frac{4}{5} & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ \frac{4}{5} & -\frac{3}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow \quad H x_1 = \begin{bmatrix} 15 \\ 0 \\ 0 \end{bmatrix} \qquad H x_2 = \begin{bmatrix} -10 \\ 0 \\ 20 \end{bmatrix}$$

Step 2. Consider the rest of $H x_2$. $\quad x_2' = \begin{bmatrix} 0 \\ 20 \end{bmatrix} \qquad e_2 := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$\|x_2'\| = 20$. Let $v_2 = x_2' - 20\, e_2$.

$$= \begin{bmatrix} -20 \\ 20 \end{bmatrix}$$

$$H'_{v_2} = I - 2\frac{v_2 v_2^T}{v_2^T v_2} = I - 2\,\frac{1}{800}\begin{bmatrix} 400 & -400 \\ -400 & 400 \end{bmatrix}$$

$$= I - \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(This result is natural as casting a permutation matrix can easily transform x_2 to $20 e_2$)

Augment it to 3-dim.

$$H_{v_2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad H_{v_2} H_{v_1} x_1 = \begin{bmatrix} 15 \\ 0 \\ 0 \end{bmatrix}$$

By Step 1 & 2. We have:

$$H_{v_2} H_{v_1} x_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -10 \\ 0 \\ 20 \end{bmatrix} = \begin{bmatrix} -10 \\ 20 \\ 0 \end{bmatrix}$$

As they're symmetric and orthonormal $\rightarrow$

$$H_{v_2} H_{v_1} X = \begin{bmatrix} 15 & -10 \\ 0 & 20 \\ 0 & 0 \end{bmatrix} \Rightarrow X = H_{v_1} H_{v_2} \begin{bmatrix} 15 & -10 \\ 0 & 20 \\ 0 & 0 \end{bmatrix}$$

$$H_{v_1} H_{v_2} = \begin{bmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ \frac{4}{5} & -\frac{3}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & 0 & \frac{4}{5} \\ \frac{4}{5} & 0 & -\frac{3}{5} \\ 0 & 1 & 0 \end{bmatrix}$$

Therefore.

$$\begin{bmatrix} 9 & -6 \\ 12 & -8 \\ 0 & 20 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{3}{5} & 0 & \frac{4}{5} \\ \frac{4}{5} & 0 & -\frac{3}{5} \\ 0 & 1 & 0 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 15 & -10 \\ 0 & 20 \\ 0 & 0 \end{bmatrix}}_{R}$$

or

$$\begin{bmatrix} 9 & -6 \\ 12 & -8 \\ 0 & 20 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{3}{5} & 0 \\ \frac{4}{5} & 0 \\ 0 & 1 \end{bmatrix}}_{\hat{Q}} \underbrace{\begin{bmatrix} 15 & -10 \\ 0 & 20 \end{bmatrix}}_{\hat{R}}$$

(c) Use it to find the least squares solution to the system of linear equations

$$\begin{aligned} 9x - 6y &= 300 \\ 12x - 8y &= 600 \\ 20y &= 900\,. \end{aligned}$$

Plot the three lines above and indicate the location of the least squares solution.

It is equivalent to let $A = \begin{bmatrix} 9 & -6 \\ 12 & -8 \\ 0 & 20 \end{bmatrix}$ $b = \begin{bmatrix} 300 \\ 600 \\ 900 \end{bmatrix}$

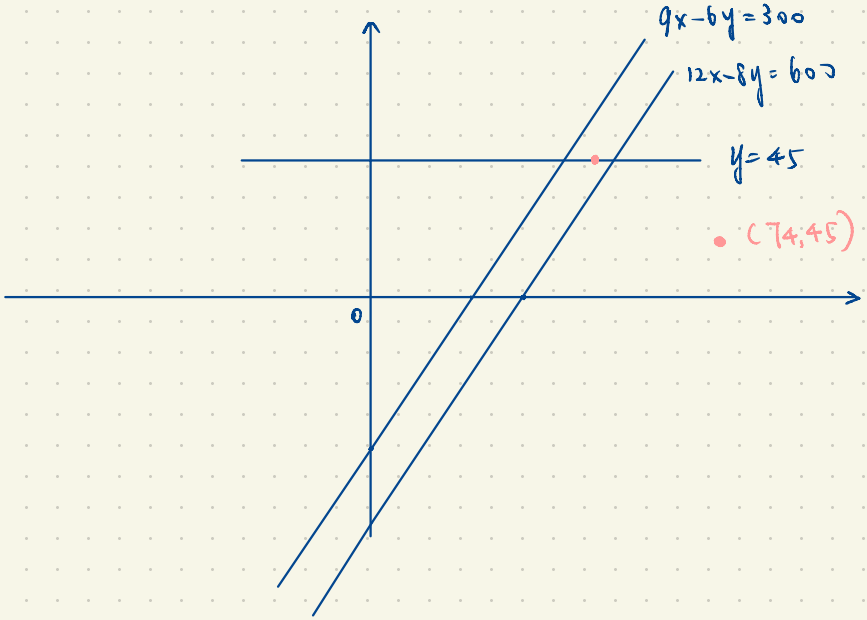
and $x = \begin{bmatrix} x \\ y \end{bmatrix}$

$\min_x \|Ax - b\|_2^2 \iff$ Solving $\hat{R}x = \hat{Q}^T b$

$$\hat{Q}^T b = \begin{bmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 300 \\ 600 \\ 900 \end{bmatrix} = \begin{bmatrix} 660 \\ 900 \end{bmatrix}$$

$$\begin{bmatrix} 15 & -10 \\ 0 & 20 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 660 \\ 900 \end{bmatrix} \iff \begin{cases} x = 74 \\ y = 45 \end{cases}$$

Plot:



3. **[Newton's method for systems, 1+1+2+2pt]** Let $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$ defined by $f(x,y) = (f_1(x,y), f_2(x,y))^T$, where

$$f_1(x,y) = x^2 + 4y^2 - 4, \quad f_2(x,y) = 2y - \sqrt{3}x^2.$$

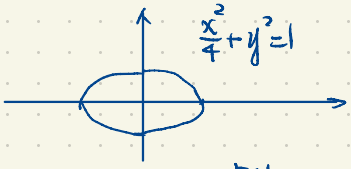
We want to find the roots of f, i.e., all pairs $(x,y) \in \mathbb{R}^2$ such that $f(x,y) = (0,0)^T$.

(a) Sketch or plot the sets $\mathcal{S}_i = \{(x,y) \in \mathbb{R}^2 : f_i(x,y) = 0\}$, $i = 1,2$, i.e., the set of all zeros of f_1 and f_2. What geometrical shapes do these sets have?

(a): $f_1(x,y) = x^2 + 4y^2 - 4 = 0 \iff \frac{x^2}{4} + y^2 = 1$

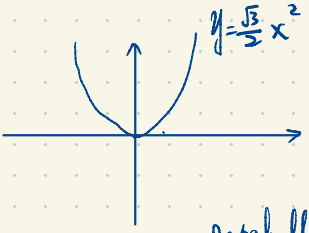
$f_2(x,y) = 2y - \sqrt{3}x^2 = 0 \iff y = \frac{\sqrt{3}}{2}x^2$

$\mathcal{S}_1$:



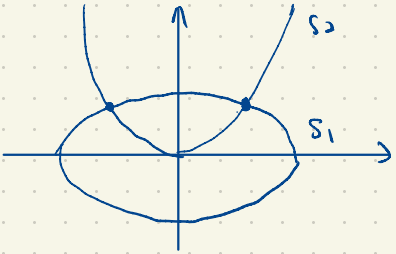
Ellipse.

$\mathcal{S}_2$:



parabolic.

(b) Calculate analytically the roots of f, i.e., the intersection of the sets $\mathcal{S}_1$ and $\mathcal{S}_2$.



$$\begin{cases} \frac{x^2}{4} + y^2 = 1 \\ y = \frac{\sqrt{3}}{2}x^2 \end{cases} \Rightarrow \frac{x^2}{4} + \frac{3}{4}x^4 = 1$$

$$3x^4 + x^2 - 4 = 0$$

$$(3x^2 + 4)(x^2 - 1) = 0$$

$$\Rightarrow x = \pm 1$$

$$\Rightarrow \begin{cases} x_1 = 1 & y_1 = \frac{\sqrt{3}}{2} \\ x_2 = -1 & y_2 = -\frac{\sqrt{3}}{2} \end{cases}$$ are roots of f.

(c) Calculate the Jacobian of f, defined by

$$J_f(x,y) = \begin{pmatrix} \partial_x f_1(x,y) & \partial_y f_1(x,y) \\ \partial_x f_2(x,y) & \partial_y f_2(x,y) \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

Here, $\partial_x f_i(x,y)$ and $\partial_y f_i(x,y)$, $i = 1,2$ denote the partial derivatives of f_i with respect to x and y, respectively.

(c) $\partial_x f_1(x,y) = 2x$ $\qquad \partial_y f_1 = 2y$

$\partial_x f_2(x,y) = -2\sqrt{3}x$ $\qquad \partial_y f_2 = 2$

$$\Rightarrow J_f(x,y) = \begin{bmatrix} \partial_x f_1 & \partial_y f_1 \\ \partial_x f_2 & \partial_y f_2 \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ -2\sqrt{3}x & 2 \end{bmatrix}$$

(d) The Newton method in 2D is as follows: Starting from an initial value $(x_0, y_0)^T \in \mathbb{R}^2$, compute the iterates

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - [J_f(x_k, y_k)]^{-1} f(x_k, y_k), \text{ for } k = 0, 1, \ldots,$$

where $[J_f(x_k, y_k)]^{-1}$ is the inverse of the Jacobi matrix of f evaluated at (x_k, y_k). Implement the Newton method in 2D and use it to calculate the first 5 iterates for the starting values $(x_0, y_0) = (2, 3)$ and $(x_0, y_0) = (-1.5, 2)$. Plot these iterates in the xy-plane together with the curves $\mathcal{S}_1$ and $\mathcal{S}_2$. Please also hand in your code.[4]

See Code file.

4. **[Eigenvalue/vector properties, 8pts]** Prove the following statements, using the basic definition of eigenvalues and eigenvectors, or give a counterexample showing the statement is not true. Assume $A \in \mathbb{R}^{n\times n}$, $n \geq 1$.

(a) If λ is an eigenvalue of A and $\alpha \in \mathbb{R}$, then $\lambda + \alpha$ is an eigenvalue of $A + \alpha I$, where I is the identity matrix.

(a) TRUE.

Proof. By def of eigenvalues,

$$\exists\, v \in \mathbb{R}^n \text{ s.t. } \quad Av = \lambda v$$

This implies.

$$(A+\alpha I)v = Av + \alpha I v = (\lambda+\alpha)v$$

It follows $\lambda+\alpha$ is an eigenvalue of $A+\alpha I$. with eigenvector v.

(b) If λ is an eigenvalue of A and $\alpha \in \mathbb{R}$, then $\alpha\lambda$ is an eigenvalue of αA.

(b) TRUE

Proof By def. $\exists\, v \in \mathbb{R}^n$ s.t.

$$Av = \lambda v.$$

Therefore.

$$(\alpha A)v = \alpha A v = \alpha\lambda v.$$

It follows $\alpha\lambda$ is an eigen value of αA. with eigenvector v.

(c) If λ is an eigenvalue of A, then for any positive integer k, λ^k is an eigenvalue of A^k.

(c) TRUE.

Proof. By def. $\exists\, v$ s.t. $Av = \lambda v$.

It implies.

$$A^k v = A^{k-1}(Av) = A^{k-1}(\lambda v) = \lambda(A^{k-1}v) = \lambda A^{k-2}(Av)$$
$$\cdots = \lambda^k v.$$

It follows that λ^k is an eigenvalue of A^k, with eigenvector v.

(d) If B is "similar" to A, which means that there is a nonsingular matrix S such that $B = SAS^{-1}$, then if λ is an eigenvalue of A, it is also an eigenvalue of B. How do the eigenvectors of B relate to the eigenvectors of A?

(d)

Proof of: If λ is an eigenvalue of A, it is also an eigenvalue of B.

By def. $\exists\, v \in \mathbb{R}^n$. $Av = \lambda v$.

As $B = SAS^{-1} \iff S^{-1}BS = A$

It implies. $Av = (S^{-1}BS)v = \lambda v$

$S^{-1}B(Sv) = \lambda v$

$B(Sv) = \lambda(Sv)$

It implies λ is an eigenvalue of B, with eigenvector Sv.

(e) Every matrix with $n \geq 2$ has at least two distinct eigenvalues, say λ and μ, with $\lambda \neq \mu$.

FALSE

CounterExample:

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ whose only eigenvalue is 1.

(f) Every real matrix has a real eigenvalue.

FALSE.

Conter-EXAMPLE

Think about Rotation Matrix.

$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ $\text{Spec}(A) = \{i, -i\}$

A is real, but No real eigenvalue.

(g) If A is singular, then it has an eigenvalue equal to zero.

(g)

Proof: characteristic function of A:

$\chi_A(\lambda) = \det(A - \lambda I)$

$\chi_A(0) = \det(A - 0I) = \det(A) = 0$

$\Rightarrow$ 0 makes $\chi_A(\lambda)$ 0

$\Rightarrow$ 0 is one of the eigenvalues.

(h) If all the eigenvalues of a matrix A are zero, then $A = 0$.

(h) False.

Counterexample:

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

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} -\lambda & 1 \\ 0 & -\lambda \end{bmatrix}\right) = \lambda^2 = 0$$

all eigenvalues are 0.

but A non zero

HW_4_code

April 1, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import hilbert
```

1 Question 1

1.1 try *eps* in numpy

[2]:
```
np.spacing(1)
```

[2]:
```
2.220446049250313e-16
```

1.2 (a)

[3]:
```
a = (1 - 1) + 1e-16
b = 1 - (1 + 1e-16)
print("a = ", a)
print("b = ", b)
```

```
a =  1e-16
b =  0.0
```

1.2.1 What is happening?

- In the first expression, we do the addition of two relateively close amount ((1-1) and 1e-16). The error is then small.
- However, in the second expression, we are actually doing the substraction between 2 very close amount(1 and 1 + 1e-16). It is very dangerous, as valid digits may be largely canceled out.

1.3 (b)

[4]:
```
error = []
condition_number = []
ns = [5, 10, 20]

for n in ns:
    H_n = hilbert(n)
```

```
    H_n_inv = np.linalg.inv(H_n)
    e_n = np.ones(n).reshape(-1, 1)

    error.append(np.linalg.norm(H_n @ (H_n_inv @ e_n) - e_n, ord=2))
    condition_number.append(np.linalg.cond(H_n, p=2))



for i in range(len(ns)):
    print("The error term when n = {} is {}".format(ns[i], error[i]))
    print("The condition number (under l_2 norm) is {}".
 ↪format(condition_number[i]))
    print()
```

```
The error term when n = 5 is 7.121339907719716e-12
The condition number (under l_2 norm) is 476607.25024259434

The error term when n = 10 is 7.998270949302708e-05
The condition number (under l_2 norm) is 16024416992541.715

The error term when n = 20 is 7.047686992816417
The condition number (under l_2 norm) is 1.3553657908688225e+18
```

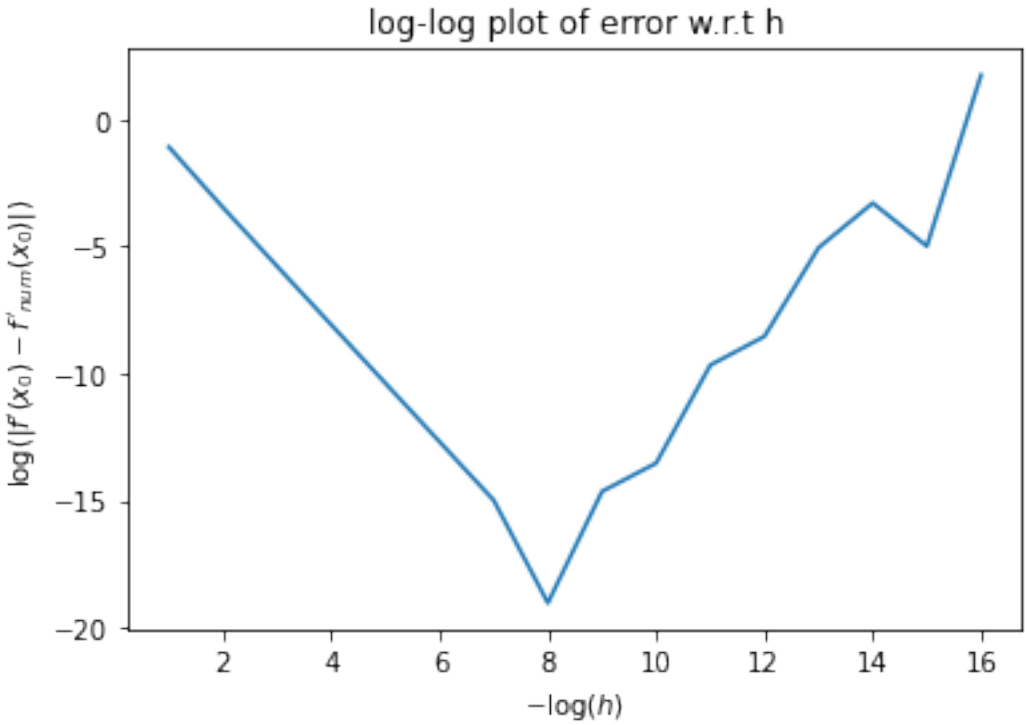
1.4 (c)

[5]:
```
f = lambda x: np.exp(x)/(np.cos(x)**3 + np.sin(x)**3)

# the following parameter stores the analytical value
# of f'(x_0)
# denoted f_p_gt (ground truth)
f_p_gt = (2**(1/2)) * np.exp(np.pi/4)
x_0 = np.pi / 4
```

[6]:
```
# the following function
def f_p_num(x=x_0, h=1e-1):
    return (f(x + h) - f(x)) / h
```

[7]:
```
error = []
ks = list(range(1, 17))

for k in ks:
    f_p = f_p_num(h=10**(-k))
    error.append(abs(f_p - f_p_gt))

log_error = np.log(error)
```

```
plt.plot(ks, log_error)
plt.xlabel(r"$-\log(h)$")
plt.ylabel(r"$\log(\|f'(x_0) - f'_{num}(x_0)\|)$")
plt.title("log-log plot of error w.r.t h")
plt.show()
```



1.4.1 Observations

- when h beacomes smaller, the error become smaller at first and then become larger.
- The best approximation: $h = 1 \times 10^{-8}$

2 Question 4

[8]:
```
f = lambda x,y: np.array([x**2 + 4*y**2 - 4, 2*y - np.sqrt(3)*x**2])

J_f = lambda x,y: np.array([
    [2*x, 8*y],
    [-2*np.sqrt(3)*x, 2]
])

J_f_inv = lambda x,y: np.linalg.inv(J_f(x, y))
```

```
# here we use x to denote [x, y]^{T}
```

[9]:
```
x_0 = np.array([2, 3])
iter = 5
history_1 = np.empty((iter+1, 2))
x = x_0 # set the initial value
for i in range(iter + 1):
    history_1[i] = x
    # do newton's iteration once
    # Here we didn't convert it to column vector
    # but it does the job, though some abuse of notation
    x = x - J_f_inv(x[0], x[1]) @ f(x[0], x[1])

print(f"Starting from: x = {x_0[0]}, y = {x_0[1]}\n")

for i in range(iter+1):
    print("The #{} iteration: x = {:.4f}, y = {:.4f}".format(i,␣
 ↪history_1[i][0], history_1[i][1]))
```

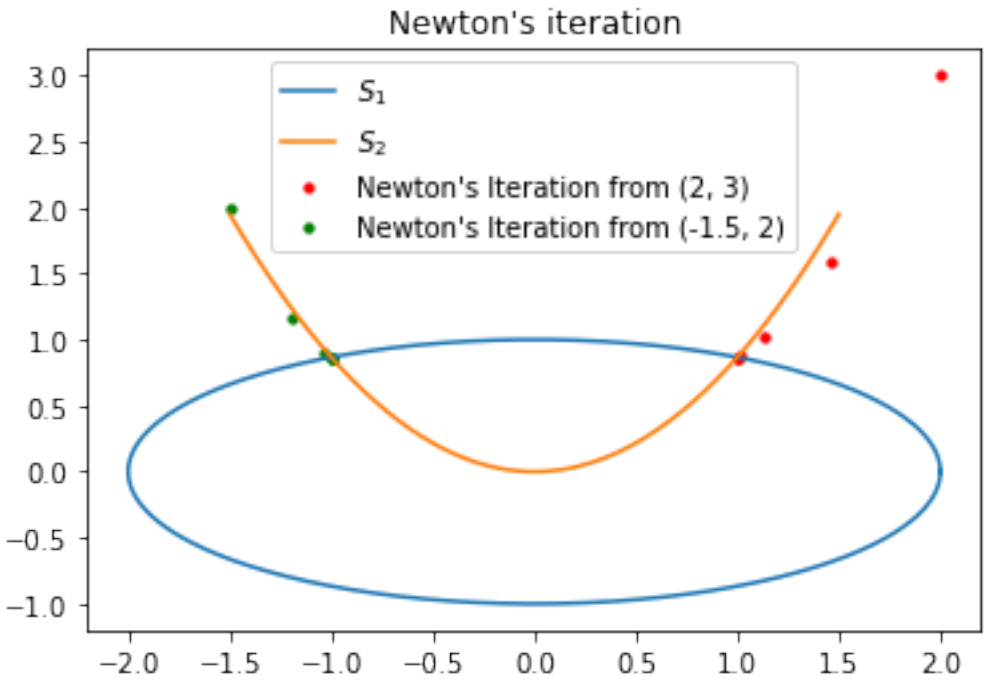
```
Starting from: x = 2, y = 3

The #0 iteration: x = 2.0000, y = 3.0000
The #1 iteration: x = 1.4590, y = 1.5902
The #2 iteration: x = 1.1320, y = 1.0172
The #3 iteration: x = 1.0127, y = 0.8759
The #4 iteration: x = 1.0001, y = 0.8661
The #5 iteration: x = 1.0000, y = 0.8660
```

[10]:
```
x_0 = np.array([-1.5, 2])
iter = 5
history_2 = np.empty((iter+1, 2))
x = x_0 # set the initial value
for i in range(iter + 1):
    history_2[i] = x
    # do newton's iteration once
    # Here we didn't convert it to column vector
    # but it does the job, though some abuse of notation
    x = x - J_f_inv(x[0], x[1]) @ f(x[0], x[1])

print(f"Starting from: x = {x_0[0]}, y = {x_0[1]}\n")

for i in range(iter+1):
    print("The #{} iteration: x = {:.4f}, y = {:.4f}".format(i,␣
 ↪history_2[i][0], history_2[i][1]))
```

```
Starting from: x = -1.5, y = 2.0

The #0 iteration: x = -1.5000, y = 2.0000
```

```
The #1 iteration: x = -1.1987, y = 1.1659
The #2 iteration: x = -1.0330, y = 0.9003
The #3 iteration: x = -1.0008, y = 0.8666
The #4 iteration: x = -1.0000, y = 0.8660
The #5 iteration: x = -1.0000, y = 0.8660
```

[11]:
```python
# visualization

# for the ellipse we use trig-parametrization
t_space = np.linspace(0, 2*np.pi, 10000)
plt.plot([2*np.cos(t) for t in t_space], [np.sin(t) for t in t_space],␣
 ↪label=r"$S_1$")

x_space = np.linspace(-1.5, 1.5, 10000)
plt.plot(x_space, .5*np.sqrt(3)*x_space**2, label=r"$S_2$")

# plt.scatter([1, -1], [.5*np.sqrt(3)]*2, label="Analytic sols")

plt.scatter(history_1[:, 0], history_1[:, 1], s=10.5, c="r", label="Newton's␣
 ↪Iteration from (2, 3)")

# for i in range(history_1.shape[0]):
#     txt = (r"($x_{}, y_{}$)").format(i, i)
#     plt.annotate(txt, (history_1[i][0], history_1[i][1]))

plt.scatter(history_2[:, 0], history_2[:, 1], s=10.5, c="g", label="Newton's␣
 ↪Iteration from (-1.5, 2)")

# for i in range(history_2.shape[0]):
#     txt = (r"($x_{}, y_{}$)").format(i, i)
#     plt.annotate(txt, (history_2[i][0], history_2[i][1]))

plt.title("Newton's iteration")

plt.legend()

plt.show()
```


Newton's iteration
S_1
S_2
Newton's Iteration from (2, 3)
Newton's Iteration from (-1.5, 2)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0