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
```

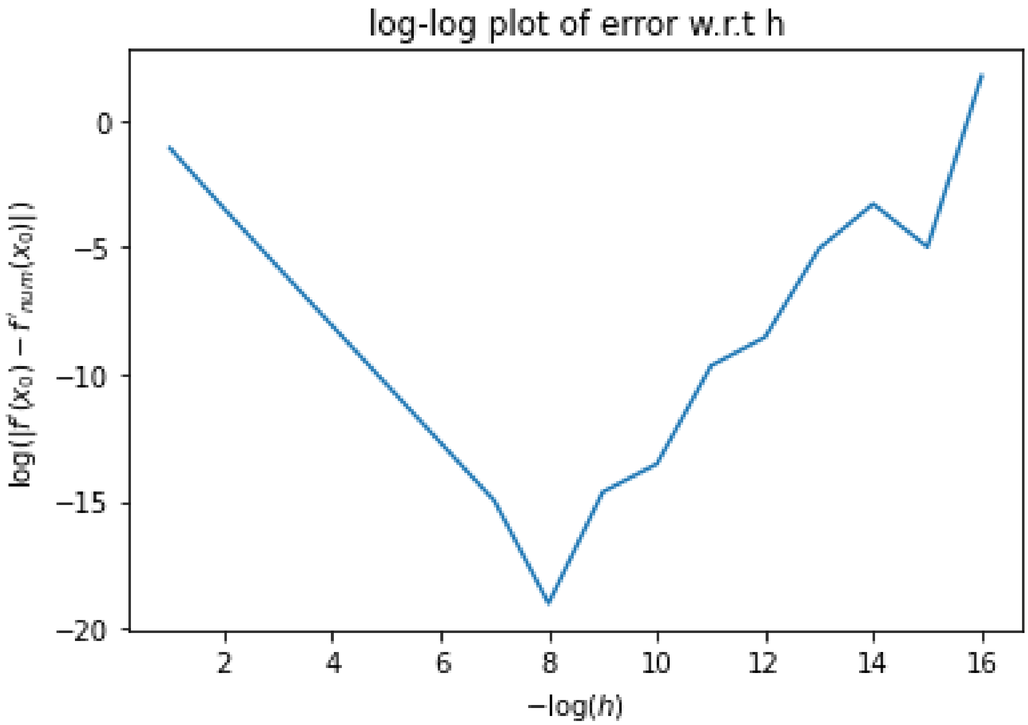
1.4 (c)

[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]))
```

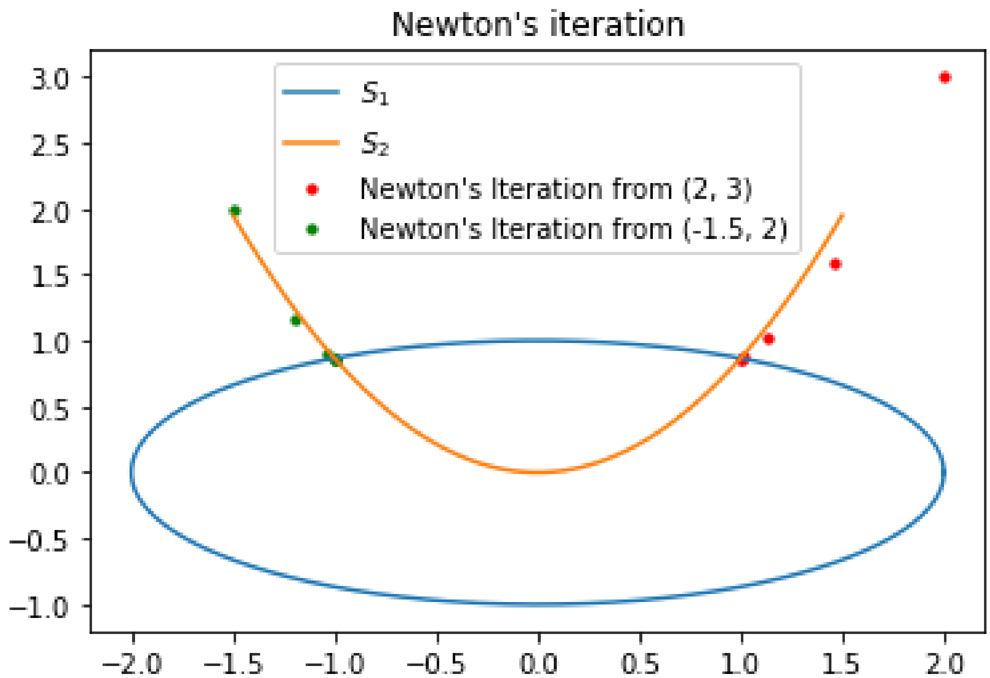
```
Starting from: x = 2, y = 3

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]:
```
# 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