Scientific Computing assignment 4

Alfred Ajay Aureate R

15 November, 2018

Problem 2.b) The following code is used for implementing the secant method of finding E in the Kepler's equation $M = e \sin E$. The power law exponents converge to a value around $p = 1.69$ as follows:

The values of E after every iteration are: $[0.0, 1.0, 0.3452654139470761, 0.3841670413170331, 0.3902213985381099, 0.39017520076657236, 0.39017524962457845, 0.39017524962497735]$,

where finally, $E_* = 0.39017524962497735$.

The corresponding values of p in the power law convergence behavior, where $|E_{n+1} - E_*| \sim C|E_n - E_*|^p$ are: $[6.27416293, 1.64823281, 1.95197692, 1.68619303, 1.69594167]$. Value of p tending towards $1.69 \approx 1.6 = \alpha$, where α is the golden ratio.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  8 12:14:01 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt


e = 0.5          # eccentricity
M = 0.2          # mean anomaly
E_1 = 0.0        # initial value for E(-1)
E0 = 1.0         # initial value for E(0)
E = [E_1,E0]     # list of E's (values of eccentric anomaly)

def fx(E,e,M):
    return (M - E + (e*np.sin(E)))

def fsecant(E_1,E0,e,M,E):
    if(abs(E0-E_1)>1e-10):
```

```
            E1 = E0 - ((fx(E0,e,M)*(E0 - E_1))/(fx(E0,e,M) - fx(E_1,e,M)))
            E.append(E1)
            return fsecant(E0,E1,e,M,E)

        return E0,E

print("After many iterations, result is:")
print(fsecant(E_1,E0,e,M,E))

print(np.log(abs(np.array(E[2:-1])-E[-1]))/np.log(abs(np.array(E[1:-2])-E[-1])))

plt.loglog(abs(np.array(E[1:-2])-E[-1]),abs(np.array(E[2:-1])-E[-1]))
plt.title('Secant Method')
plt.xlabel('$log|E_{n}-E^{*}|$')
plt.ylabel('$log|E_{n+1}-E^{*}|$')
plt.show()
```

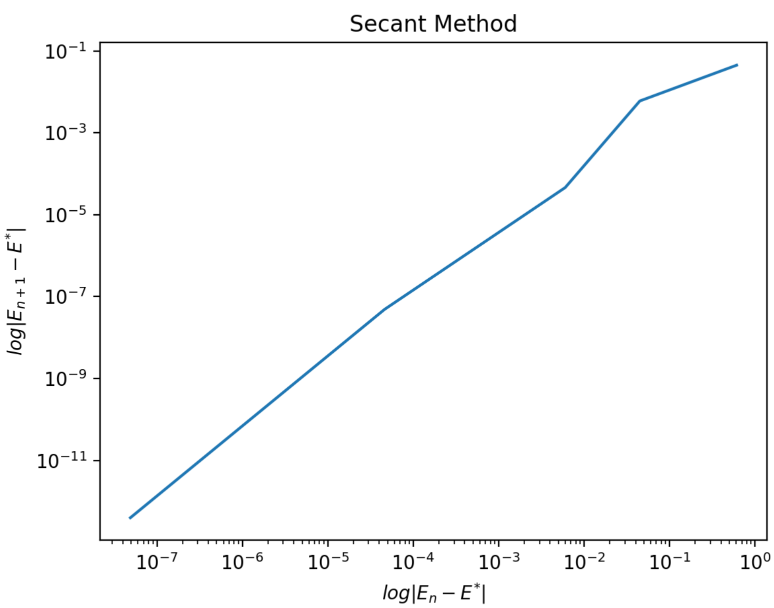

Figure 1: Power law convergence behavior of E_n while using *secant method* for finding E in the Kepler's equation.

Problem 2.c) The following code is used for finding E in the Kepler's equation $M = e\sin E$ by direct iteration.

The values of E after every iteration are: [0.0, 1.0, 0.6207354924039483, 0.49081680237536957, 0.43567321133649267, 0.4110104105096477, 0.39976789643438154, 0.39460227514016233, 0.39222052985163885, 0.391120632152837, 0.39061232950687325, 0.3903773464604816, 0.39026869960365135, 0.3902184620466329, 0.3901952317876737, 0.3901844897628397, 0.39017952245257925, 0.39017722546934636, 0.3901761632969478, 0.3901756721261951, 0.3901754449984842, 0.39017533996982695, 0.39017529140235113, 0.3901752689437209, 0.39017525855837404, 0.3901752537559694, 0.3901752515352357, 0.3901752505083213, 0.3901752500334543, 0.39017524981386575, 0.3901752497123233, 0.390175249665368],

where finally, $E_* = 0.390175249665368$.

The corresponding values of C in the linear power law convergence behavior, where $|E_{n+1} - E_*| \sim C|E_n - E_*|^{p=1}$ are: [0.37807623, 0.4365087, 0.45207929, 0.45793614, 0.46040666, 0.46150198, 0.46199874, 0.4622264, 0.4623312, 0.46237958, 0.46240188, 0.46241209, 0.4624166, 0.46241822, 0.46241798, 0.46241572, 0.46241003, 0.46239736, 0.46236977, 0.46231003, 0.46218076, 0.46190097, 0.46129491, 0.45997961, 0.45711315, 0.45080907, 0.43666278, 0.4034319, 0.31620225]. Value of C seems to be tending towards 0.462, except for some exceptions.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  8 12:14:01 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt

e = 0.5          # eccentricity
M = 0.2          # mean anomaly
E_1 = 0.0        # initial value for E(-1)
E0 = 1.0         # initial value for E(0)
E = [E_1,E0]     # list of E's (values of eccentric anomaly)

def flinear(E_1,E0,e,M,E):
    if(abs(E0-E_1)>1e-10):
        E1 = M + (e*np.sin(E0))
        E.append(E1)
        return flinear(E0,E1,e,M,E)

    return E0,E

print("After many iterations, result is:")
print(flinear(E_1,E0,e,M,E))

print(abs(np.array(E[2:-1])-E[-1])/abs(np.array(E[1:-2])-E[-1]))

plt.loglog(abs(np.array(E[1:-1])-E[-1]),abs(np.array(E[2:])-E[-1]))
```

```
plt.title('Linear Method')
plt.xlabel('$log|E_{n}-E^{*}|$')
plt.ylabel('$log|E_{n+1}-E^{*}|$')
plt.show()
```

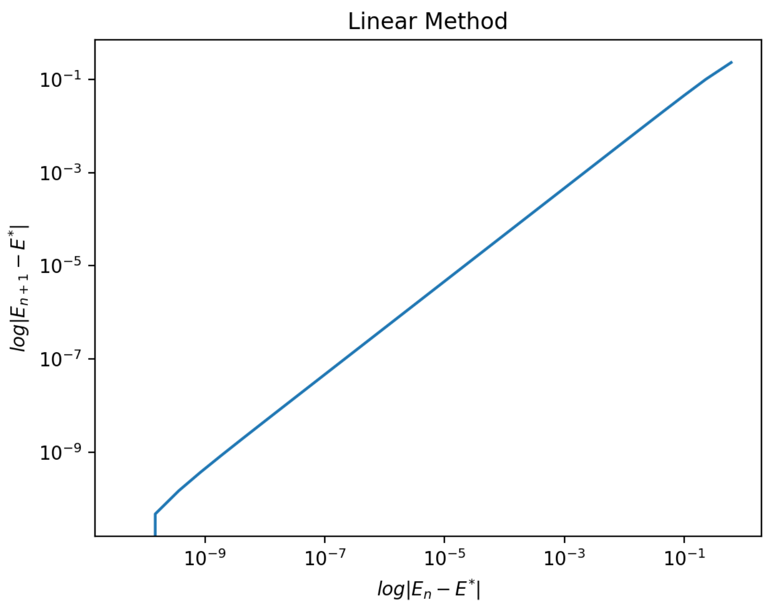

Figure 2: Power law convergence behavior of E_n while using *linear method* for finding E in the Kepler's equation.

Problem 2.d) The following code is used for finding E in the Kepler's equation $M = e \sin E$ using Newton's solver.

The values of E after every iteration are: $[0.0, 1.0, 0.48035198093331269, 0.39176459888254067, 0.3901756973312443, 0.39017524962501277, 0.39017524962497735]$,

where finally, $E_* = 0.39017524962497735$.

The corresponding values of p in the power law convergence behavior, where $|E_{n+1} - E_*| \sim C|E_n - E_*|^p$ are: [4.864665092.678501192.268490322.11856743]. Value of p is converging towards 2 as expected for Newton's solver.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  8 12:14:01 2018

@author: alfred_mac
```

```
"""

import numpy as np
import matplotlib.pyplot as plt


e = 0.5          # eccentricity
M = 0.2          # mean anomaly
E_1 = 0.0        # initial value for E(-1)
E0 = 1.0         # initial value for E(0)
E = [E_1,E0]     # list of E's (values of eccentric anomaly)

def fx(E,e,M):
    return (M - E + (e*np.sin(E)))

def fx_deriv(E,e,M):
    return (-1 + (e*np.cos(E)))

def fNewton(E_1,E0,e,M,E):
    if(abs(E0-E_1)>1e-10):
        E1 = E0 - fx(E0,e,M)/fx_deriv(E0,e,M)
#        E1 = E0 - ((fx(E0,e,M)*(E0 - E_1))/(fx(E0,e,M) - fx(E_1,e,M)))
        E.append(E1)
        return fNewton(E0,E1,e,M,E)

    return E0,E

print("After many iterations, result is:")
print(fNewton(E_1,E0,e,M,E))

print(np.log(abs(np.array(E[2:-1])-E[-1]))/np.log(abs(np.array(E[1:-2])-E[-1])))


plt.loglog(abs(np.array(E[1:-1])-E[-1]),abs(np.array(E[2:])-E[-1]))
plt.title('Newton\'s Method')
plt.xlabel('$log|E_{n}-E^{*}|$')
plt.ylabel('$log|E_{n+1}-E^{*}|$')
plt.show()
```

Problem 4) The following code is used for applying Newton's method for optimization in d dimensions without safeguards. It has been implemented for different values of λ and d (no. of dimensions).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

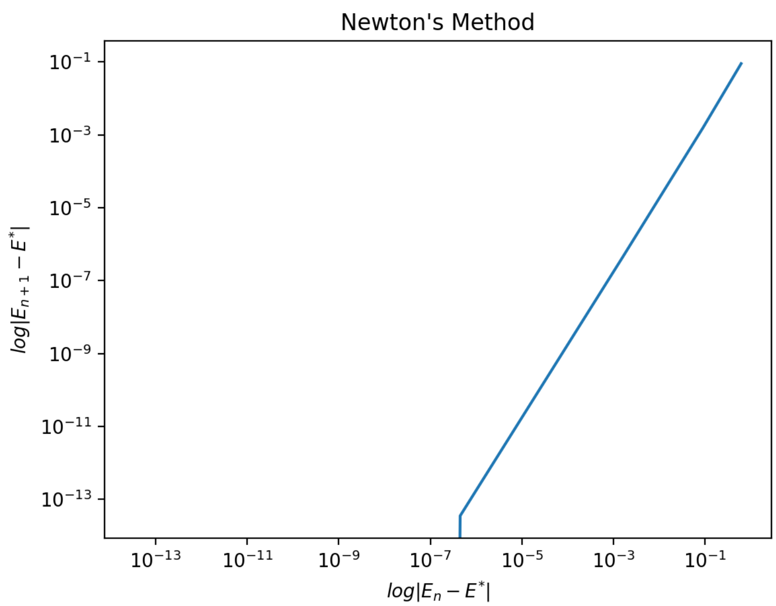

Figure 3: Power law convergence behavior of E_n while using *Newton's solver* for finding E in the Kepler's equation.

```
Created on Thu Nov  8 12:14:01 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la


Lambda = [1e-3,1e-2,1e-1,1,10,100,1000]
D = [5,20,50,200,500]                    # Different no. of dimensions of the s

def H(X,d,lambda_param):
    N = d
    HX = np.zeros([N,N])
    HX[0,0] = 2 + lambda_param*np.exp(X[0])
    HX[0,1] = -1
    HX[N-1,N-1] = 2 + lambda_param*np.exp(X[N-1])
    HX[N-1,N-2] = -1
    for i in range(N-2):
```

```
        HX[i+1,i] = -1
        HX[i+1,i+1] = 2 + lambda_param*np.exp(X[i+1])
        HX[i+1,i+2] = -1

    return HX

def G(X,d,lambda_param):
    N = len(X)
    GX = np.zeros([N,1])
    GX[0,0] = (2*X[0]) - X[1] + lambda_param*np.exp(X[0])
    GX[N-1,0] = (2*X[N-1]) - X[N-2] + lambda_param*np.exp(X[N-1])
    for i in range(N-2):
        GX[i+1,0] = (2*X[i+1]) - X[i] - X[i+2] + lambda_param*np.exp(X[i+1])

    return GX


def Newton_Optimization(X,d,lambda_param):
    Xn = X
    Xn1 = Xn - np.dot(la.inv(H(X,d,lambda_param)),G(X,d,lambda_param))
    if(np.sum((Xn1-Xn)*(Xn1-Xn))>1e-4):
        Xn1 = Newton_Optimization(Xn1,d,lambda_param)

    return Xn1

# Loop for simulating different values of Lambda
#for i in range(len(Lambda)):
#     d = 20
#     lambda_param = Lambda[i]
#     X = np.zeros([d,1])
#     RES = Newton_Optimization(X,d,lambda_param)
#     plt.plot(np.arange(d)/d,RES)

# Loop for simulating different values of d
for i in range(len(D)):
    lambda_param = 1
    d = D[i]
    X = np.zeros([d,1])
    RES = Newton_Optimization(X,d,lambda_param)
    plt.plot(np.arange(d)/d,RES)

#plt.title('Varying $\lambda$')
#plt.legend(['$\lambda=0.0001$','$\lambda=0.001$','$\lambda=0.01$','$\lambda=0.1
plt.title('Varying $d$')
plt.legend(['$d=5$','$d=20$','$d=50$','$d=200$','$d=500$'],loc='lower left')
plt.xlabel('$i$')
```

```
plt.ylabel('$X_{i}$')
plt.show()
```

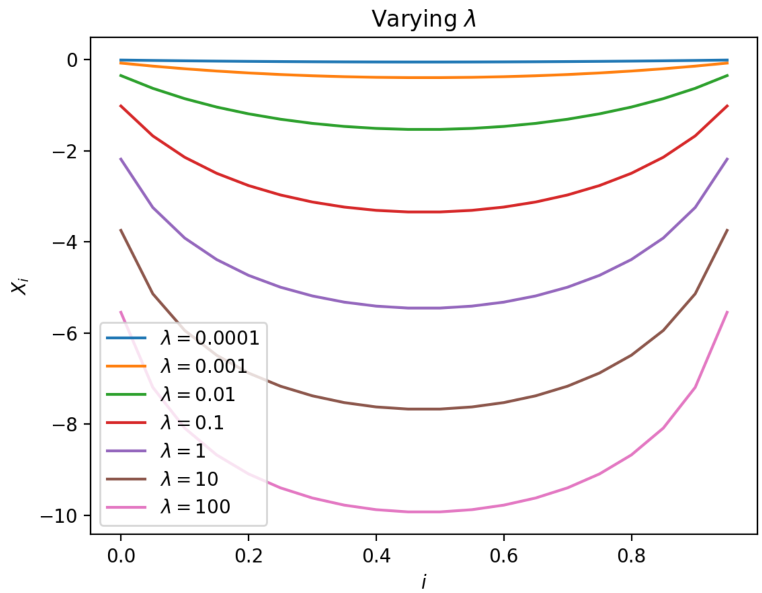

Figure 4: Optimized value of X for different values of λ

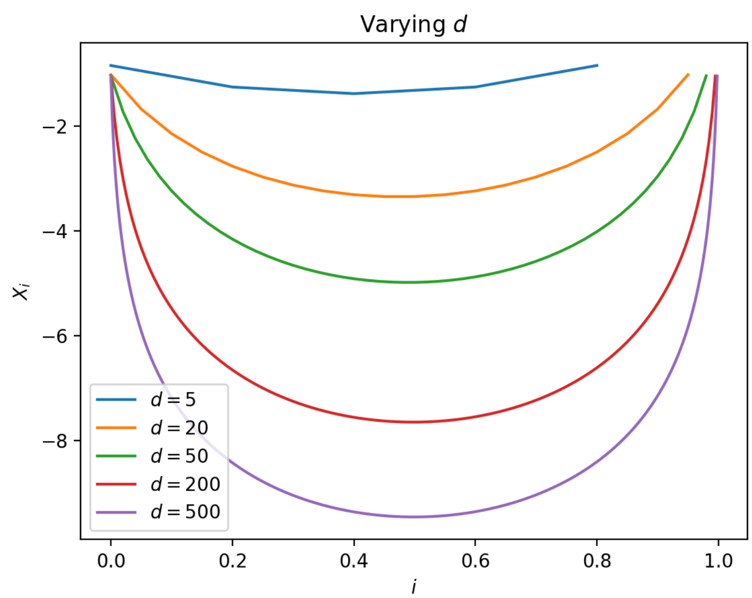

Figure 5: Optimized value of X for different values of d (no. of dimensions)