ΕΡΓΑΣΙΕΣ ΠΑΡΕΜΒΟΛΗΣ

Παραδοτέες μέχρι 30/11/19

Α.Ένταση μετάδοσης λέιζερ μέσα από διηλεκτρική πλάκα.

Η ένταση I της ακτινοβολίας που μεταδίδεται μέσα από μια πλάκα διηλεκτρικού υλικού με παράλληλες επίπεδες πλευρές όταν μια ακτίνα λέιζερ (μονοχρωματικό φως) προσπίπτει σε μια πλευρά της δίνεται από τη σχέση:

$$I = \frac{1}{1 + g^2 \sin^2 \delta} \qquad g^2 = \frac{4r^2}{(1 - r^2)^2}$$

όπου δ είναι η μετατόπιση της φάσης του κύματος λέιζερ κατά την αλλαγή του μέσου διάδοσης από το κενό στο διηλεκτρικό και r είναι ο συντελεστής ανάκλασης, δηλαδή ο λόγος του πλάτους του ανακλώμενου κύματος προς το πλάτος του προσπίπτοντος κύματος.

1. Σχεδιάστε την I χρησιμοποιώντας τιμές της φασικής μετατόπισης δ=0˚(3˚)360˚ για τρεις τιμές του r=0.7, 0.8, 0.9.

2. Φτιάξτε έναν πίνακα με τιμές της I για τιμές της φασικής μετατόπισης δ=0˚(24˚)360˚ για τις τρεις τιμές του r=0.7, 0.8, 0.9.

3. Χρησιμοποιώντας τις τιμές του πίνακα που φτιάξατε, προσεγγίστε την I στα σημεία δ= 90˚, 180˚, 270˚, που δεν συμπεριλαμβάνονται στον πίνακα, με παρεμβολή Lagrange δεύτερου βαθμού. Συγκρίνετε τα αποτελέσματα με τις ακριβείς τιμές από τον αναλυτικό τύπο της συνάρτησης σε αυτά τα σημεία και σχολιάστε την ακρίβεια της παρεμβολής για τις τρεις τιμές του r.

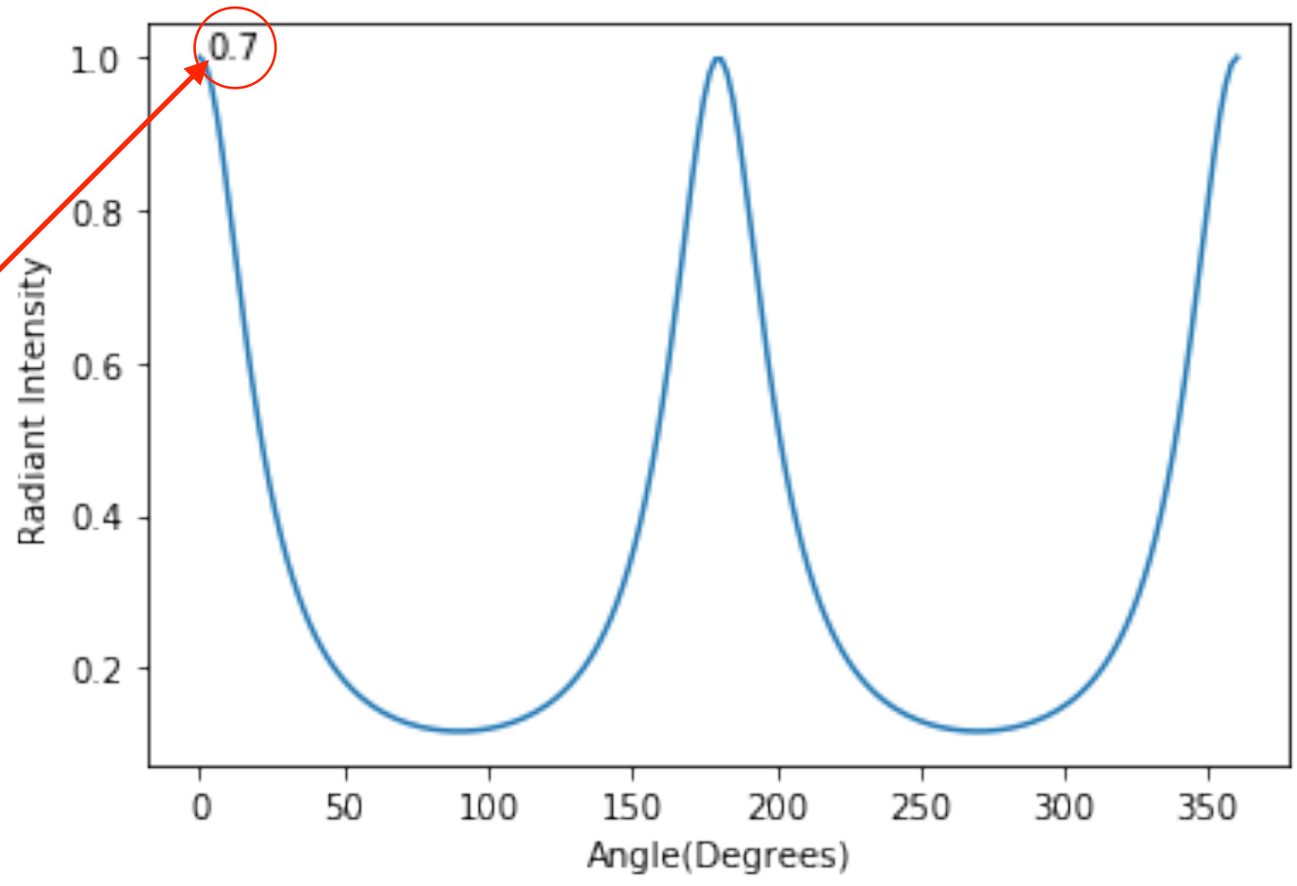
[25]:
```
import matplotlib.pyplot as plt
import numpy as np
import math

#A)

def g(x):
    return 2*x/(1-x**2)

def f(x,g):
    return 1/(1+(g**2)*(np.sin(x))**2)

#a)

r=0.7

for i in range(3):
    x=np.linspace(0,2*math.pi,180)
    y = f(x,g(r))
    plt.plot(x*180/math.pi,y)
    plt.xlabel("Angle(Degrees)")
    plt.ylabel("Radiant Intensity")
    plt.text(2,1,round(r,1))
    plt.show()
    r = r + 0.1

#b)
r=0.7
for i in range(3):
    x=np.linspace(0,2*math.pi,16)
    x_n=[round(x[i]*(180/math.pi)) for i in range(len(x))]
    f_n=[1/(1+(g(r)**2)*(np.sin(x[i]))**2) for i in range(len(x))]
    print("For r = ",round(r,1))
    print()
    print("d\t\t\t\t\tI")
    for x,y in zip(x_n,f_n):
        print(x ,"\t\t\t\t",y)
```

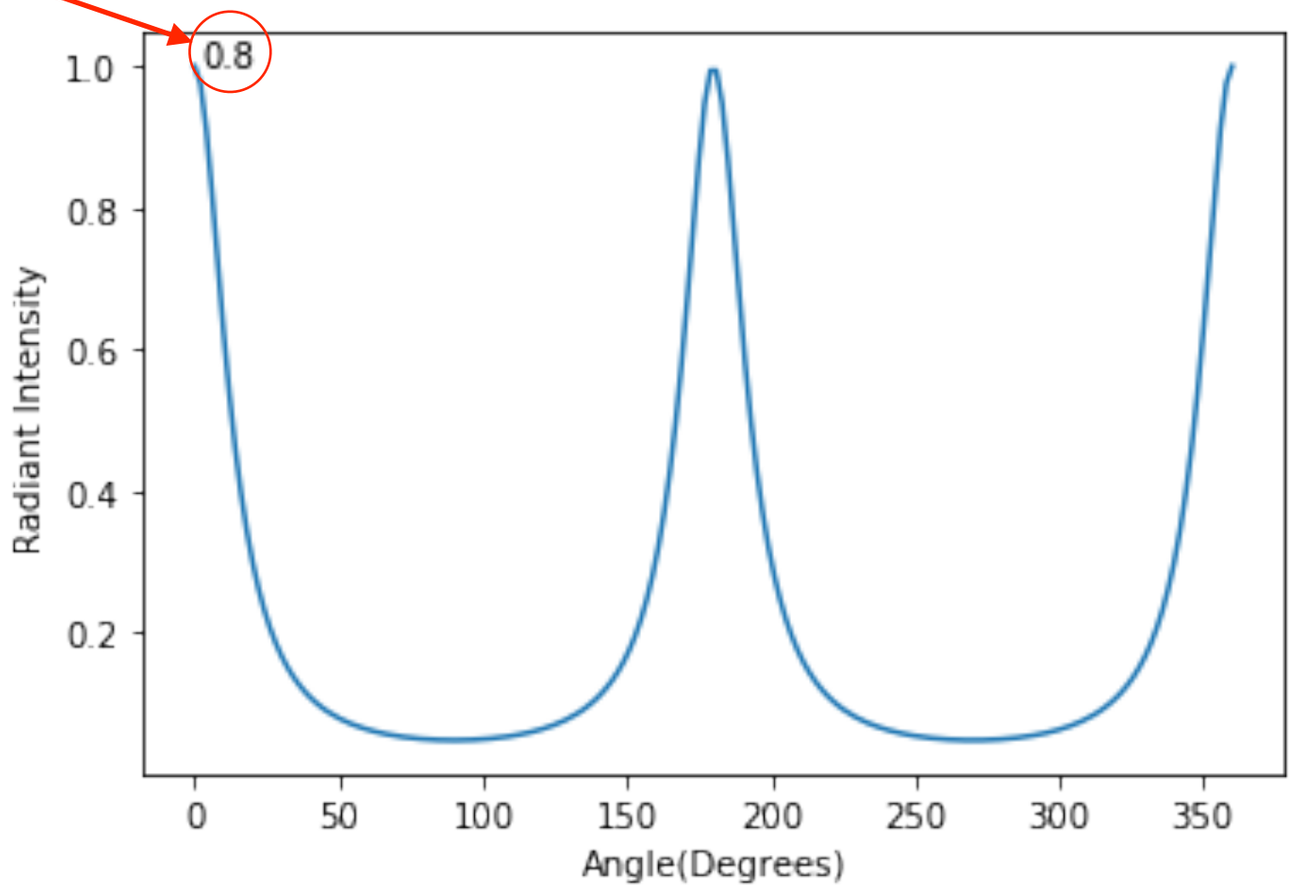
Κώδικας για την πρώτη εργασία παρεμβολής.

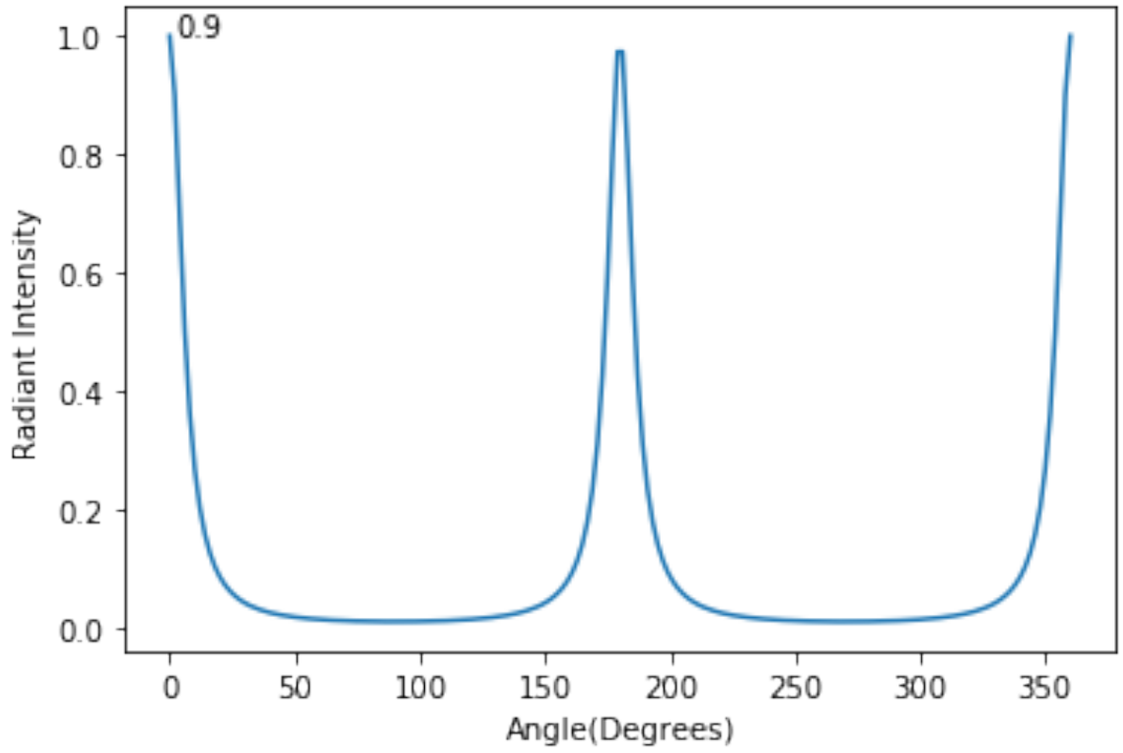
Εφαρμογή της μεθόδου Lagrange β΄ βαθμού.

```
    print()
    r = r + 0.1
#c)
x = np.linspace(0,360,16)
xi=[x[i] for i in range(len(x))]
xp = [(math.pi/180)*90,(math.pi/180)*180,(math.pi/180)*270]
ip = [2,6,10]
r = [0.7,0.8,0.9]
gi = [2*r[i]/(1-r[i]**2) for i in range(len(r))]

for k in range(3):
    f=[]
    x0 = []
    x1 = []
    x2 = []
    L0 = []
    L1 = []
    L2 = []
    p2 = []
    f = []
    error = []
    fi=[1/(1+(gi[k]**2)*(math.sin((math.pi/180)*xi[i]))**2) for i in␣
 ↪range(len(xi))]
    for i in range(3):
        x0.append((math.pi/180)*xi[ip[i]])
        x1.append((math.pi/180)*xi[ip[i] + 1])
        x2.append((math.pi/180)*xi[ip[i] + 2])
        L0.append(((xp[i]-x1[i])*(xp[i]-x2[i]))/((x0[i]-x1[i])*(x0[i]-x2[i])))
        L1.append(((xp[i]-x0[i])*(xp[i]-x2[i]))/((x1[i]-x0[i])*(x1[i]-x2[i])))
        L2.append(((xp[i]-x0[i])*(xp[i]-x1[i]))/((x2[i]-x0[i])*(x2[i]-x1[i])))
        p2.append(fi[ip[i]]*L0[i] + fi[ ip[i] +1]*L1[i] + fi[ip[i] +2]*L2[i])
        f.append([1/(1+(gi[k]**2)*(math.sin((xp[i]))**2))])
        error.append([(abs(p2[i]-f[i])/f[i])*100])
    xp_1 = [90,180,270]
    plt.plot(xp_1,p2,label='2nd degree Lagrange interpolation')
    plt.plot(xp_1,f,label='The function of radiant Intensity')
    plt.xlabel("Angle(Degrees)")
    plt.ylabel("Radiant Intensity")
    plt.text(140,0.8,round(r[k],1))
    plt.legend()
    plt.title("Interpolation in 3 points ")
    plt.show()
    print("For r =",r[k])
    print()
    print("x=",xp_1)
    print("I=",p2)
    print("Itheoritical = ",f)
```

Καθώς αυξάνεται ο συντελεστής ανάκλασης, τα μέγιστα της έντασης στα πολλαπλάσια των 180ο γίνονται οξύτερα, άρα και η προσέγγιση της συνάρτησης σε αυτά δυσκολότερη.







For r = 0.7

d	I
0.0	1.0
24.0	0.44510842340314183
48.0	0.19373754823754305
72.0	0.12794299292398634
96.0	0.11829800155812452
120.0	0.15033813074388766
144.0	0.2775100522657928
168.0	0.7542946695518715
192.0	0.7542946695518727
216.0	0.27751005226579295
240.0	0.15033813074388774
264.0	0.11829800155812452
288.0	0.1279429929239863
312.0	0.19373754823754288
336.0	0.4451084234031409
360.0	1.0

For r = 0.8

d	I
0.0	1.0
24.0	0.23431024270131287

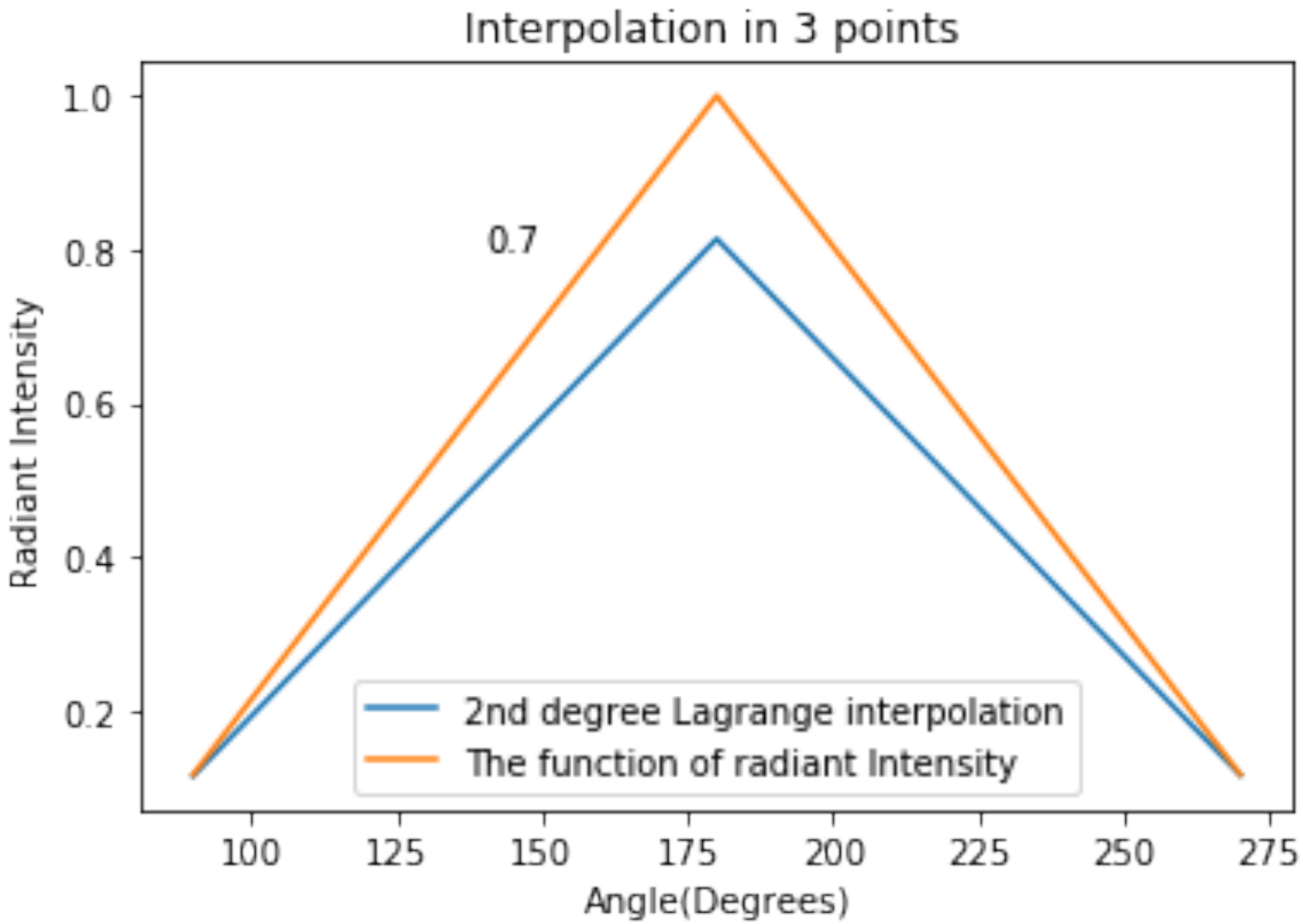
Πίνακας τιμών της έντασης

48.0	0.083970648908103
72.0	0.05300306203171865
96.0	0.04869198654661793
120.0	0.06323185011709605
144.0	0.127803309482311
168.0	0.5394115599355855
192.0	0.5394115599355872
216.0	0.12780330948231108
240.0	0.0632318501170961
264.0	0.04869198654661794
288.0	0.053003062031718645
312.0	0.08397064890810291
336.0	0.23431024270131212
360.0	1.0

For r = 0.9

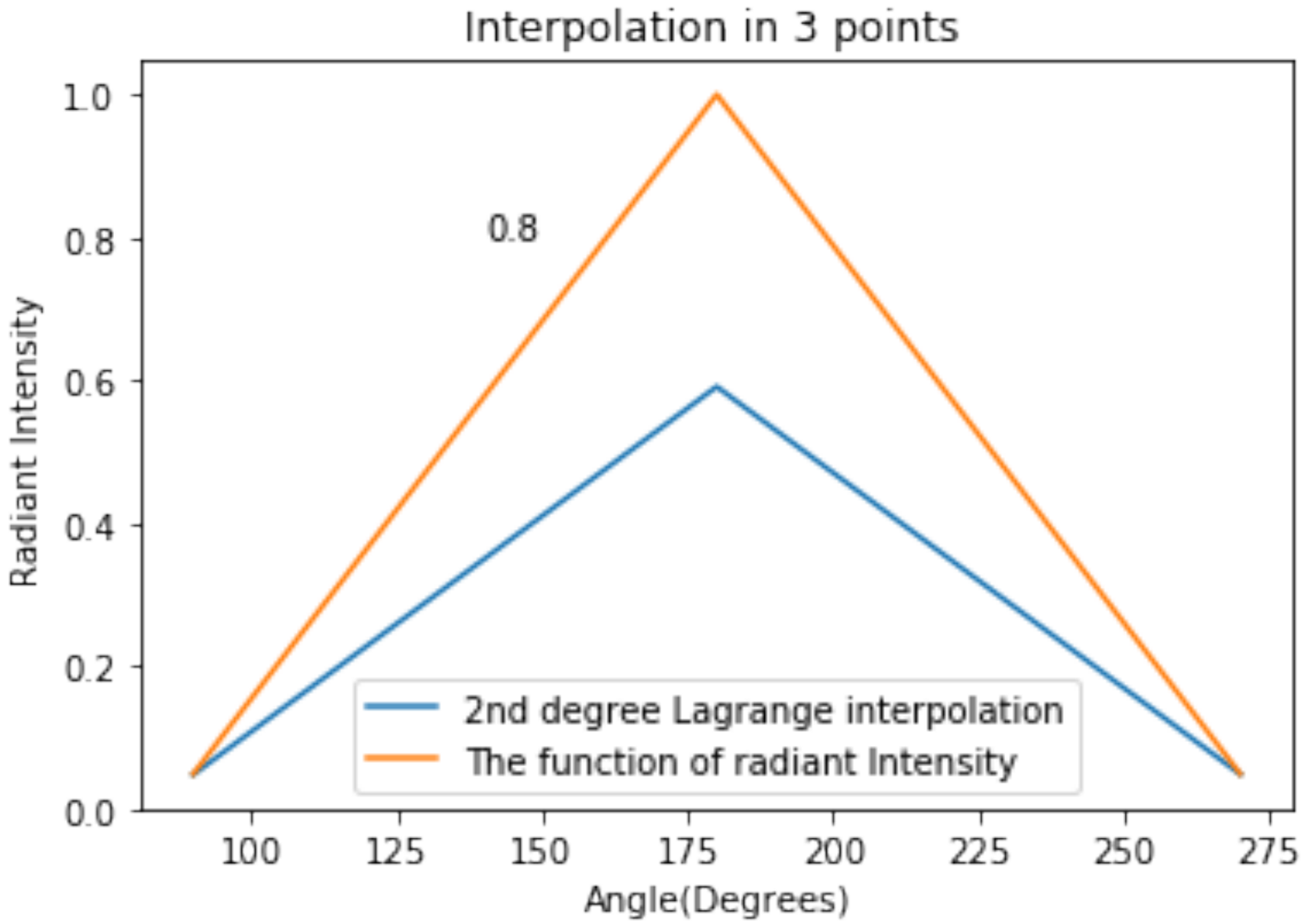
d	I
0.0	1.0
24.0	0.06309992806121455
48.0	0.0197760961298386
72.0	0.01216837171313671
96.0	0.011139571660041772
120.0	0.014638498033332009
144.0	0.031242090247221382
168.0	0.2049315748975127
192.0	0.20493157489751376
216.0	0.03124209024722141
240.0	0.01463849803333202
264.0	0.011139571660041774
288.0	0.012168371713136708
312.0	0.019776096129838574
336.0	0.06309992806121431
360.0	1.0

Η δυσκολία προσέγγισης της συνάρτησης στο μέγιστο των 180ο φαίνεται από το σφάλμα της παρεμβολής: ~20% για r=0.7…



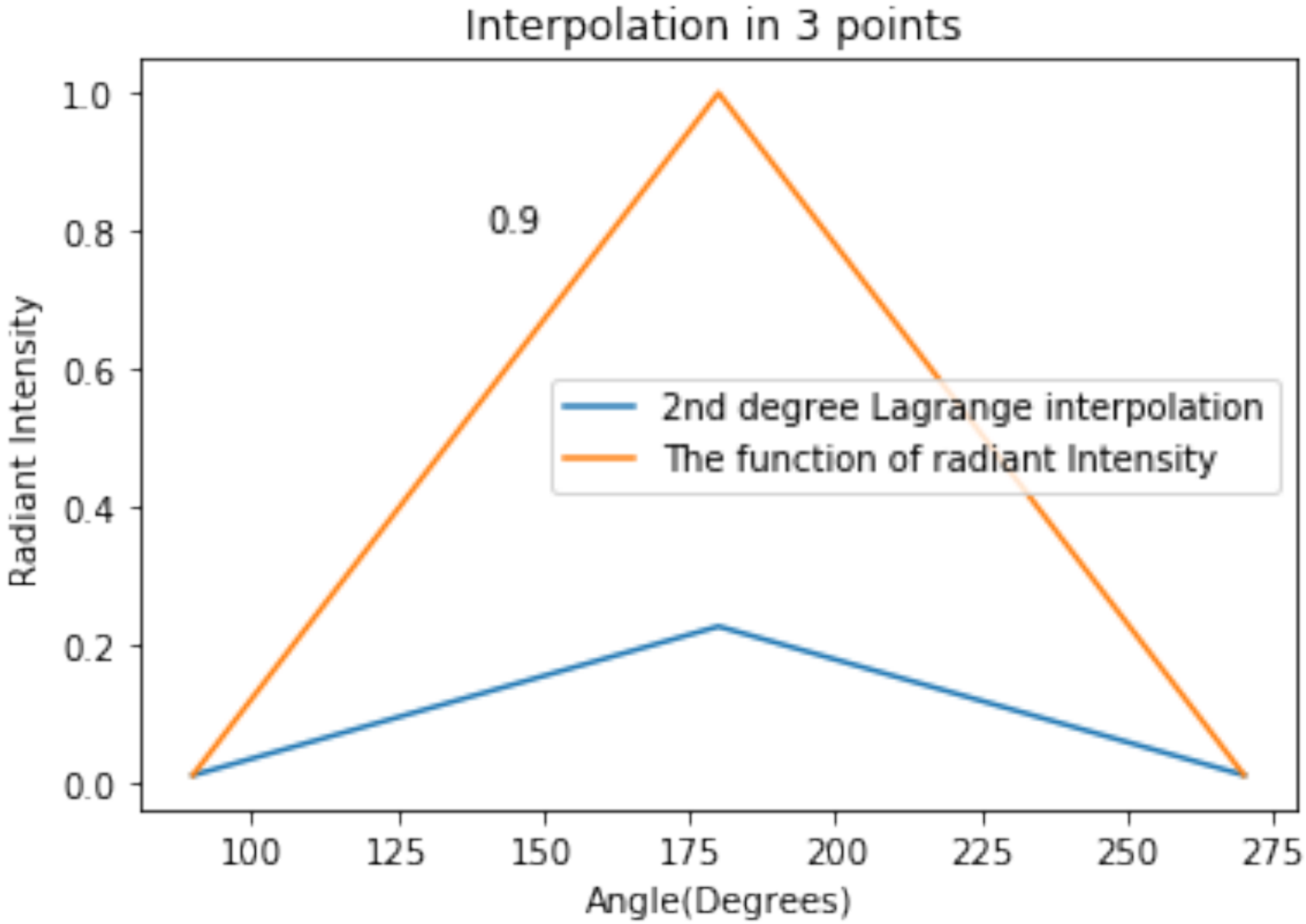

```
For r = 0.7

x= [90, 180, 270]
I= [0.11544522777949359, 0.813892746712632, 0.11680126934787513]
Itheoritical =  [[0.11715688482500793], [1.0], [0.11715688482500793]]
Error= [[array([1.4609957])], [array([18.61072533])], [array([0.30353784])]] %
```

~40% για r=0.8…




```
For r = 0.8

x= [90, 180, 270]
I= [0.047270707474960205, 0.5908625912422454, 0.04800247988143251]
Itheoritical =  [[0.04818560380725754 6], [1.0], [0.04818560380725754 6]]
Error= [[array([1.89869226])], [array([40.91374088])], [array([0.38003867])]] %
```

~80% για r=0.9.




```
For r = 0.9

x= [90, 180, 270]
I= [0.010779997514227329, 0.22664276047879928, 0.0109722973208418 68]
Itheoritical =  [[0.011019199658130087], [1.0], [0.011019199658130087]]
Error= [[array([2.17077602])], [array([77.33572395])], [array([0.42564196])]] %
```

[ ]:

Β.Διαγράμματα Bode για την απόκριση κυκλώματος.

Η απόκριση ενός κυκλώματος ορίζεται από το πηλίκο του σήματος εξόδου προς το σήμα εισόδου (π.χ. την τάση ή το ρεύμα εισόδου και εξόδου) και στην ανάλυση κυκλωμάτων είναι μια μιγαδική συνάρτηση με μέτρο (πλάτος απόκρισης) M και όρισμα (φάση απόκρισης) φ. Για μια κατηγορία κυκλωμάτων, το πλάτος και η φάση απόκρισης δίνονται από τις σχέσεις:

$$M = 10\log\left[(1-\omega^2T^2)^2 + (2\zeta\omega T)^2)\right] \qquad \phi = \tan^{-1}\frac{2\zeta\omega T}{1-\omega^2T^2}$$

όπου log είναι ο δεκαδικός λογάριθμος, ω είναι η κυκλική συχνότητα του σήματος εισόδου, T είναι μια χαρακτηριστική περίοδος ταλάντωσης του κυκλώματος και $\zeta = (R/2)/\sqrt{L/C}$ είναι το κλάσμα απόσβεσης των ταλαντώσεων του κυκλώματος (R, C, L είναι αντίστοιχα η ωμική αντίσταση, η χωρητικότητα και η αυτεπαγωγή του κυκλώματος).

1. Σχεδιάστε τις καμπύλες του πλάτους $M(\omega)$ και της φάσης απόκρισης $\varphi(\omega)$ για T=1, δηλαδή μετρώντας το ω σε μονάδες $1/T$, σαν συνάρτηση του logω στο διάστημα [0.1,3] για τρεις τιμές του ζ=0.1, 0.4, 1.

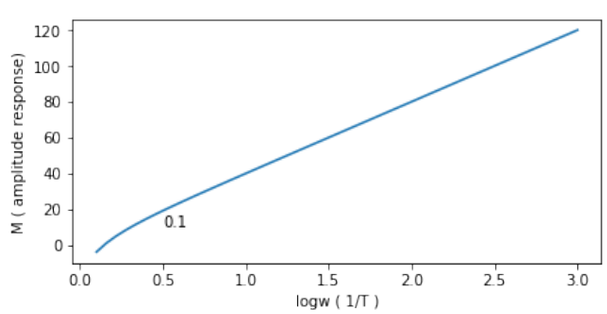
2. Φτιάξτε έναν πίνακα με τιμές του M και της φ για τιμές του logω=0.1(0.1)1 και 1(0.5)3 για τις τρεις τιμές του ζ=0.1, 0.4, 1.

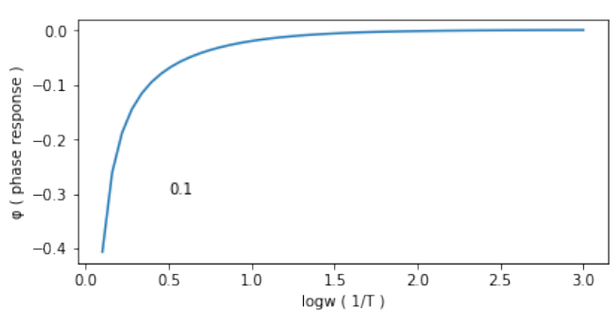
3. Χρησιμοποιώντας τις τιμές του πίνακα που φτιάξατε, προσεγγίστε το M και τη φ στα σημεία logω=0.75 και 1.25, που δεν συμπεριλαμβάνονται στον πίνακα, με παρεμβολή Lagrange πρώτου βαθμού. Συγκρίνετε τα αποτελέσματα με τις ακριβείς τιμές από τους αναλυτικούς τύπους των δύο συναρτήσεων.

[31]:
```
import matplotlib.pyplot as plt
import numpy as np
import math

#a)

#plotting the function for logw=[0.3,1] for z = 0.1 , 0.4 , 1 and period time =␣
↪T = 1

z = [0.1,0.4,1]
for k in range(len(z)):
    logw = np.linspace(0.1,3)
    M = [ 10*math.log((1-pow(10,logw[i])**2)**2+(2*z[k]*pow(10,logw[i]))**2,10)␣
↪for i in range(len(logw))]
    f = np.arctan(2*z[k]*pow(10,logw)/(1-pow(10,logw)**2))

    fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,3))
    ax1.plot(logw,M)
    ax1.set_xlabel("logw ( 1/T )")
    ax1.set_ylabel("M ( amplitude response)")
    ax1.text(0.5,10,z[k])
    ax2.plot(logw,f)
    ax2.set_xlabel("logw ( 1/T )")
    ax2.set_ylabel("  ( phase response )")
    ax2.text(0.5,-0.3,z[k])
    plt.show()

#b)
for k in range(len(z)):
    logw1 = np.arange(0.1,1+0.1,0.1)
    M = [10*math.log((1-pow(10,logw1[i])**2)**2+(2*z[k]*pow(10,logw1[i]))**2,10)␣
↪for i in range(len(logw1))]
    logw2 = np.arange(1,3+0.5,0.5)
    f = np.arctan(2*z[k]*pow(10,logw2)/(1-pow(10,logw2)**2))
    print('z=',z[k])
    print()
    print('logw1\t\t\t\t\tM')
```

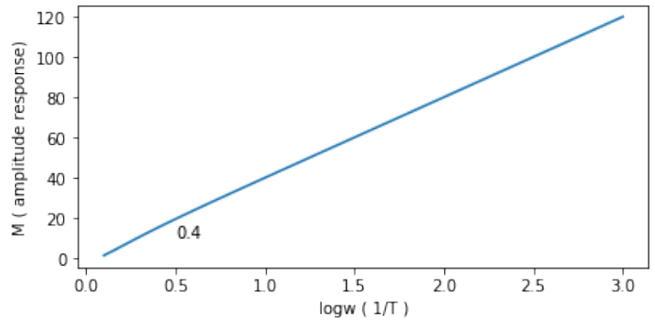
Κώδικας για τη δεύτερη εργασία παρεμβολής.

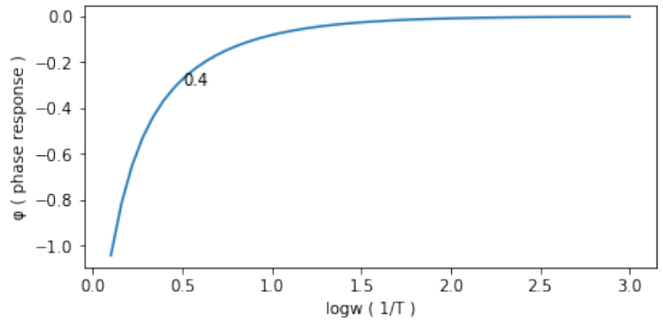
```
    for x,y in zip(logw1,M):
        print(x ,"\t\t\t\t", y)
    print()
    print('logw2\t\t\t\tf')
    for x,y in zip(logw2,f):
        print(x ,"\t\t\t\t", y)
    print()

#c)

for k in range(len(z)):
    print("z=",z[k])
    def M(x):
        return  10*math.log((1-x**2)**2+(2*z[k]*x)**2,10)
    x = pow(10,0.75)
    x0 = pow(10,0.7) #Paremvoli sto x = 0.75 gia tin M
    x1 = pow(10,0.8)
    L0 = (x - x1)/(x0 - x1)
    L1 = (x - x0)/(x1 - x0)
    p1M = M(x0)*L0+M(x1)*L1
    print("M(0.75)=",M(x))
    Error_M = abs(M(x)-p1M)/M(x) *100
    def f(x):
        return math.atan(2*z[k]*x/(1-x**2))
    x = pow(10,1.25)
    x0 = pow(10,1) #Paremvoli sto x = 1.25 gia tin f
    x1 = pow(10,1.5)
    L0 = (x - x1)/(x0 - x1)
    L1 = (x - x0)/(x1 - x0)
    p1f = f(x0)*L0+f(x1)*L1
    Error_f = abs(f(x)-p1f)/f(x) *100
    print("P1_M(0.75)=",p1M)
    print("Error(0.75) = ",Error_M,"%")
    print("f(1.25)=",f(x))
    print("P1_f(1.25)=",p1f)
    print("Error(1.25) = ",Error_f,"%")
    print()

#plotting the function regarding the interpolation
```

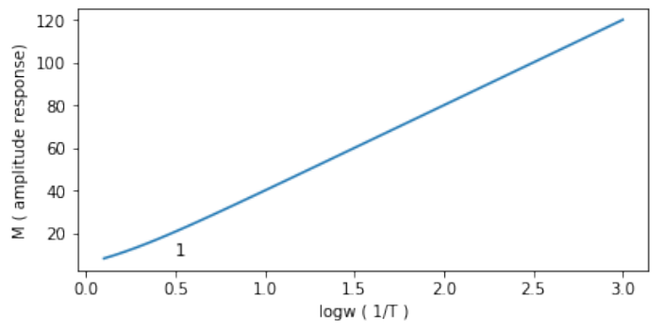


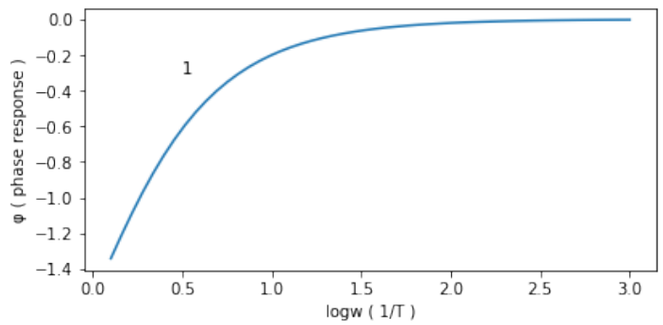


Εδώ φαίνεται ότι το διάστημα [0.1,3.0] σχεδιασμού των συναρτήσεων M και φ που ζητάει η εργασία δεν είναι κατάλληλο.









z= 0.1

logw1	M
0.1	-3.9201366725781286
0.2	3.777206804487874
0.30000000000000004	9.56458090031908
0.4	14.539899539335131
0.5	19.106244048892012
0.6	23.446373285521425
0.7000000000000001	27.654631813280734
0.8	31.78362226494958
0.9	35.864077065770424
1.0	39.91447598003802

logw2	f
1.0	-0.020199272581067927
1.5	-0.006330801627683958
2.0	-0.0020001973525415767
2.5	-0.0006324617723223284
3.0	-0.0002000001973335254

z= 0.4

Πίνακες τιμών των M και φ (f) για διάφορες τιμές του ζ (z).

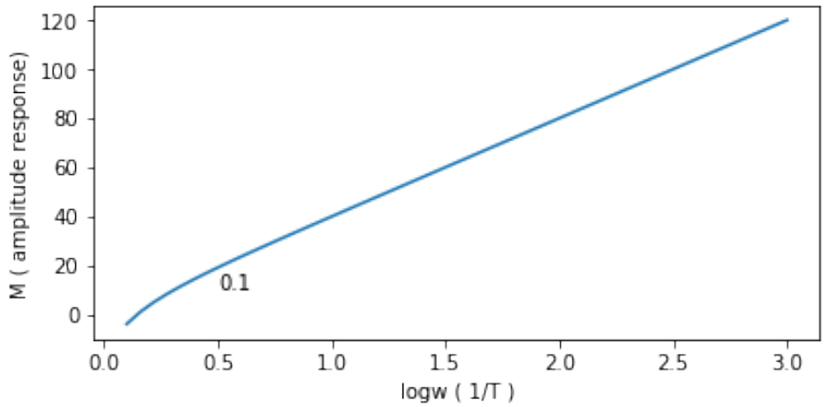
```
logw1                                  M
0.1                                    1.323979274701039
0.2                                    5.903299055287202
0.30000000000000004                                      10.58223802865923
0.4                                    15.082562249093616
0.5                                    19.415114326344032
0.6                                    23.629235115203358
0.7000000000000001                                       27.76552778810995
0.8                                    31.851880640162932
0.9                                    35.90648097872695
1.0                                    39.940970895882096

logw2                                  f
1.0                                    -0.08063287594517382
1.5                                    -0.02531813373087248
2.0                                    -0.008000629368687671
2.5                                    -0.0025298420295139096
3.0                                    -0.0008000006293336869

z= 1

logw1                                  M
0.1                                    8.248852055886793
0.2                                    10.910809262185875
0.30000000000000004                                      13.946455874173909
0.4                                    17.27784068286759
0.5                                    20.8278537031645
0.6                                    24.531447511922046
0.7000000000000001                                       28.33908578559066
0.8                                    32.21548451023914
0.9                                    36.136582566249054
1.0                                    40.086427475652854

logw2                                  f
1.0                                    -0.19933730498232405
1.5                                    -0.06322448399238237
2.0                                    -0.01999933337333048
2.5                                    -0.006324534238612181
3.0                                    -0.001999999333333733

z= 0.1
M(0.75)= 29.726745453283343
P1_M(0.75)= 29.600416235323884
Error(0.75) =  0.424968209715354 %
f(1.25)= -0.011282026218441519
P1_f(1.25)= -0.015207524485727984
Error(1.25) =  -34.79426648441814 %
```

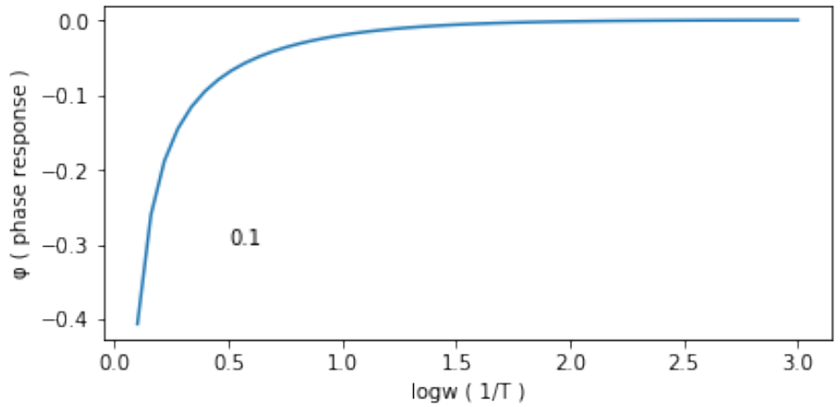
```
z= 0.4
M(0.75)= 29.813623465245072
P1_M(0.75)= 29.691219265377136
Error(0.75) =  0.4105646534733635 %
f(1.25)= -0.04509941800442927
P1_f(1.25)= -0.0607231641953659
Error(1.25) =  -34.642899802836034 %

z= 1
M(0.75)= 30.27041844216076
P1_M(0.75)= 30.165836498709986
Error(0.75) =  0.34549222915634603 %
f(1.25)= -0.11234993750709722
P1_f(1.25)= -0.1503455367324795
Error(1.25) =  -33.81897673328218 %
```

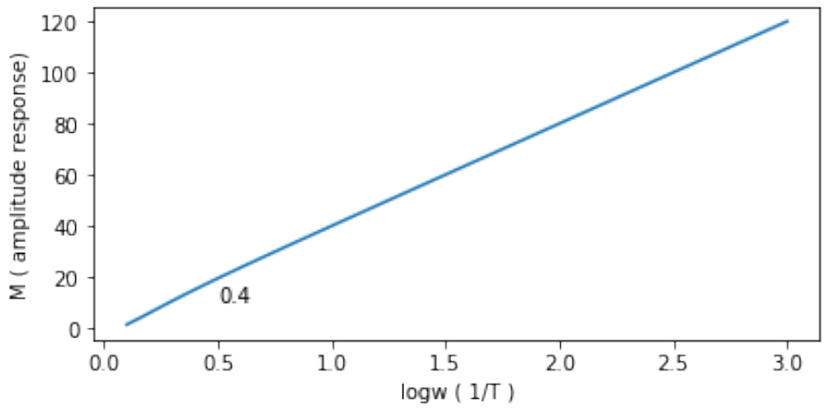
Το σφάλμα στο M είναι μικρό, αλλά στο φ είναι μεγάλο.

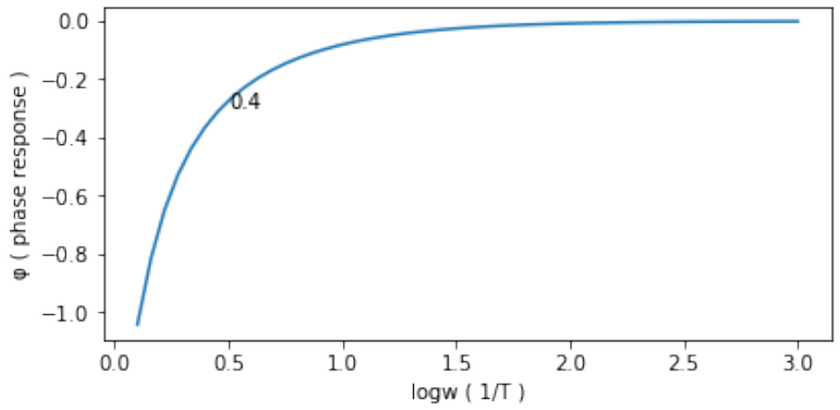
[38]:
```
import matplotlib.pyplot as plt
import numpy as np
import math

#a)

#plotting the function for logw=[0.3,1] for z = 0.1 , 0.4 , 1 and period time =
 ↪T = 1

z = [0.1,0.4,1]
for k in range(len(z)):
    logw = np.linspace(0.1,3)
    M = [ 10*math.log((1-pow(10,logw[i])**2)**2+(2*z[k]*pow(10,logw[i]))**2,10)
 ↪for i in range(len(logw))]
    f = np.arctan(2*z[k]*pow(10,logw)/(1-pow(10,logw)**2))

    fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,3))
    ax1.plot(logw,M)
    ax1.set_xlabel("logw ( 1/T )")
    ax1.set_ylabel("M ( amplitude response)")
    ax1.text(0.5,10,z[k])
    ax2.plot(logw,f)
    ax2.set_xlabel("logw ( 1/T )")
    ax2.set_ylabel("  ( phase response )")
    ax2.text(0.5,-0.3,z[k])
    plt.show()

#b)
for k in range(len(z)):
    logw1 = np.arange(0.1,1+0.1,0.1)
```

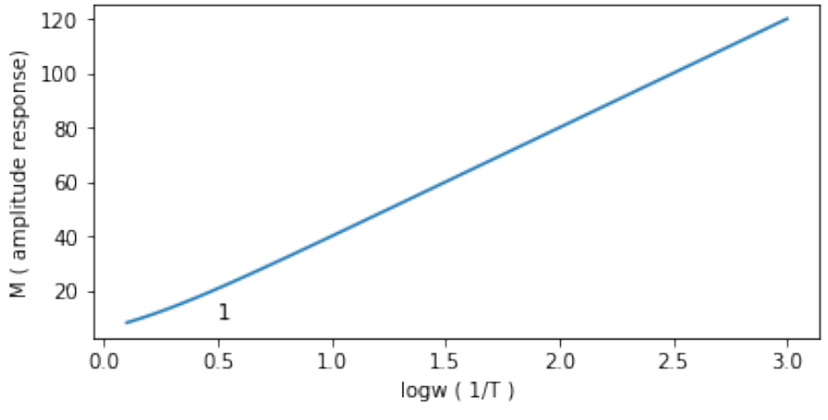
```
    M = [10*math.log((1-pow(10,logw1[i])**2)**2+(2*z[k]*pow(10,logw1[i]))**2,10)
↪for i in range(len(logw1))]
    logw2 = np.arange(1,3+0.5,0.5)
    f = np.arctan(2*z[k]*pow(10,logw2)/(1-pow(10,logw2)**2))
    print('z=',z[k])
    print()
    print('logw1\t\t\t\t\tM')
    for x,y in zip(logw1,M):
        print(x ,"\t\t\t\t", y)
    print()
    print('logw2\t\t\t\t\tf')
    for x,y in zip(logw2,f):
        print(x ,"\t\t\t\t", y)
    print()

#c)

for k in range(len(z)):
    print("z=",z[k])
    def M(x):
        return  10*math.log((1-x**2)**2+(2*z[k]*x)**2,10)
    x = pow(10,0.75)
    x0 = pow(10,0.7) #Paremvoli sto x = 0.75 gia tin M
    x1 = pow(10,0.8)
    L0 = (x - x1)/(x0 - x1)
    L1 = (x - x0)/(x1 - x0)
    p1M = M(x0)*L0+M(x1)*L1
    Error_M = abs(M(x)-p1M)/M(x) *100
    def f(x):
        return math.atan(2*z[k]*x/(1-x**2))
    y = pow(10,1.25)
    x0 = pow(10,1) #Paremvoli sto y = 1.25 gia tin f
    x1 = pow(10,1.5)
    L0 = (y - x1)/(x0 - x1)
    L1 = (y - x0)/(x1 - x0)
    p1f = f(x0)*L0+f(x1)*L1
    Error_f = abs((f(y)-p1f)/f(y)) *100
    print("P1_M(0.75)=",p1M)
    print("M(0.75)=",M(x))
    print("Error(0.75) = ",Error_M,"%")
    print()
    print("P1_f(1.25)=",p1f)
    print("f(1.25)=",f(y))
    print("Error(1.25) = ",Error_f,"%")
    print()
```

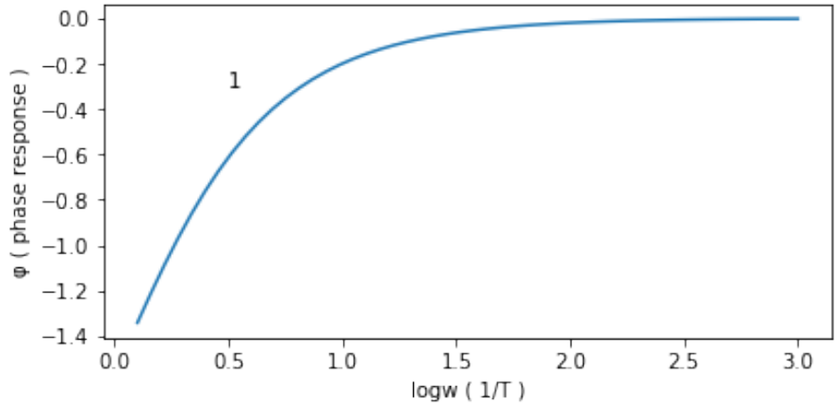

M (amplitude response)
logw (1/T)
0.1


φ (phase response)
logw (1/T)
0.1


M (amplitude response)
logw (1/T)
0.4


φ (phase response)
logw (1/T)
0.4


M (amplitude response)
logw (1/T)
1


φ (phase response)
logw (1/T)
1

1.0	-0.020199272581067927
1.5	-0.006330801627683958
2.0	-0.0020001973525415767
2.5	-0.0006324617723223284
3.0	-0.0002000001973335254

z= 0.4

logw1	M
0.1	1.323979274701039
0.2	5.903299055287202
0.30000000000000004	10.58223802865923
0.4	15.082562249093616
0.5	19.415114326344032
0.6	23.629235115203358
0.7000000000000001	27.76552778810995
0.8	31.851880640162932
0.9	35.90648097872695
1.0	39.940970895882096

logw2	f
1.0	-0.08063287594517382
1.5	-0.02531813373087248
2.0	-0.008000629368687671
2.5	-0.0025298420295139096
3.0	-0.0008000006293336869

z= 1

logw1	M
0.1	8.248852055886793
0.2	10.910809262185875
0.30000000000000004	13.946455874173909
0.4	17.27784068286759
0.5	20.8278537031645
0.6	24.531447511922046
0.7000000000000001	28.33908578559066
0.8	32.21548451023914
0.9	36.136582566249054
1.0	40.086427475652854

logw2	f
1.0	-0.19933730498232405
1.5	-0.06322448399238237
2.0	-0.01999933337333048
2.5	-0.006324534238612181
3.0	-0.001999999333333733

```
z= 0.1
P1_M(0.75)= 29.600416235323884
M(0.75)= 29.726745453283343
Error(0.75) =  0.424968209715354 %

P1_f(1.25)= -0.015207524485727984
f(1.25)= -0.011282026218441519
Error(1.25) =  34.79426648441814 %

z= 0.4
P1_M(0.75)= 29.691219265377136
M(0.75)= 29.813623465245072
Error(0.75) =  0.4105646534733635 %

P1_f(1.25)= -0.0607231641953659
f(1.25)= -0.04509941800442927
Error(1.25) =  34.642899802836034 %

z= 1
P1_M(0.75)= 30.165836498709986
M(0.75)= 30.27041844216076
Error(0.75) =  0.34549222915634603 %

P1_f(1.25)= -0.1503455367324795
f(1.25)= -0.11234993750709722
Error(1.25) =  33.81897673328218 %
```

[ ]:

[0]:

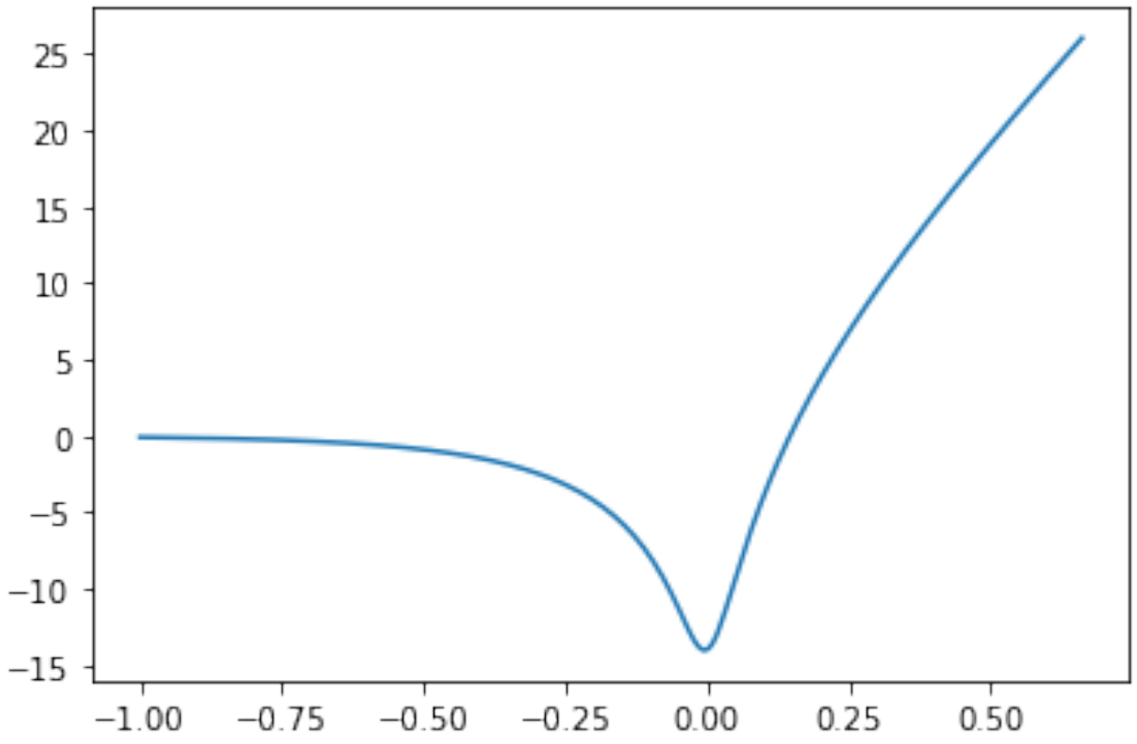
[0]:
```
#B.Δ   μμ   Bode                μ   .
#E     1
import math
import numpy as np
import matplotlib.pyplot as plt

z=[0.1,0.4,1.]

for t in range (3):
 print("Γ             :",z[t])
 logw=[0.0]*300
 w=[0.0]*300
 #M:
 M=[0.0]*300
 #F:
 F=[0.0]*300
 logw[0]=0.
 w[0]=0.
 M[0]=0.
 for i in range (300):
    w[i]=0.1 + i*0.015
    logw[i]=math.log10(w[i])
    M[i]=10*math.log10((1-w[i]*w[i])*(1-w[i]*w[i])+4*z[t]*z[t]*w[i]*w[i])
    F[i]=math.atan((2*z[t]*w[i]/(1-w[i]*w[i])))
   # print("",i,")",logw[i],"Π   :",M[i],"Φ  :",F[i])


plt.plot(logw,M)
plt.show()
plt.plot(logw,F)
plt.show()
```

Εδώ οι συναρτήσεις M και φ ξανασχεδιάζονται στο πιο κατάλληλο διάστημα [-1.00,0.75].

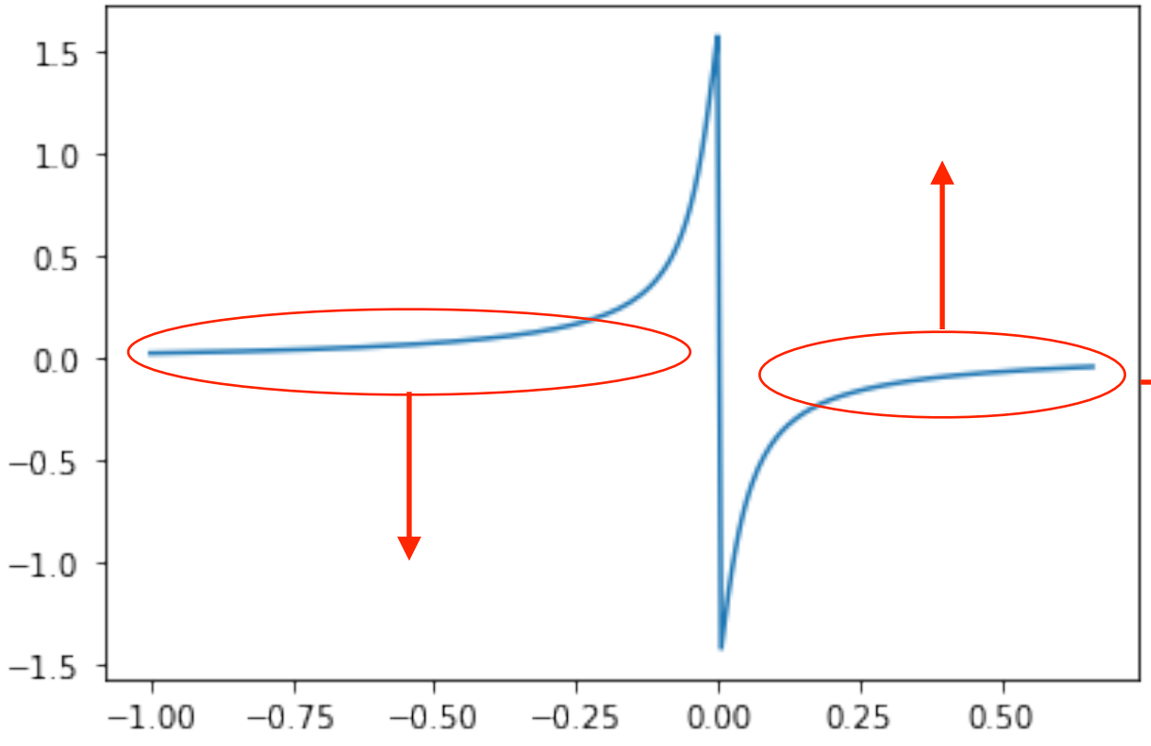
Εδώ έπρεπε να χρησιμοποιηθεί η συνάρτηση math.atan2(2*z[t]*w[i],1-w[i]*w[i]).

```
Γ             : 0.1
```

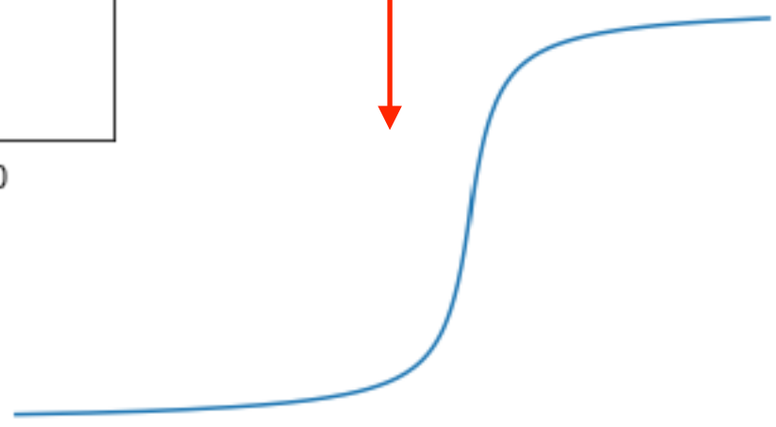
Το μέτρο M της απόκρισης του κυκλώματος παρουσιάζει ελάχιστο στη συχνότητα 0.



Εδώ φαίνεται η ανάγκη χρήσης της atan2(y,x) που επιστρέφει το όρισμα $\phi = \arctan(y/x)$ στο διάστημα (−π,0) όταν y<0 και στο [0,π] όταν y≥0.

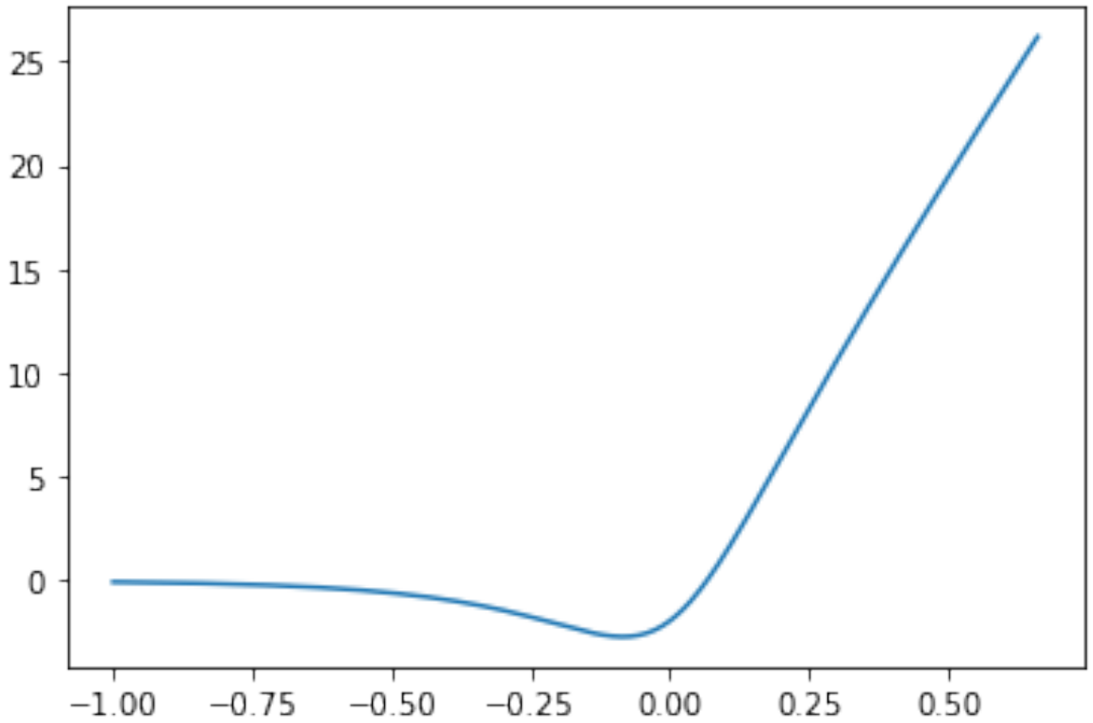


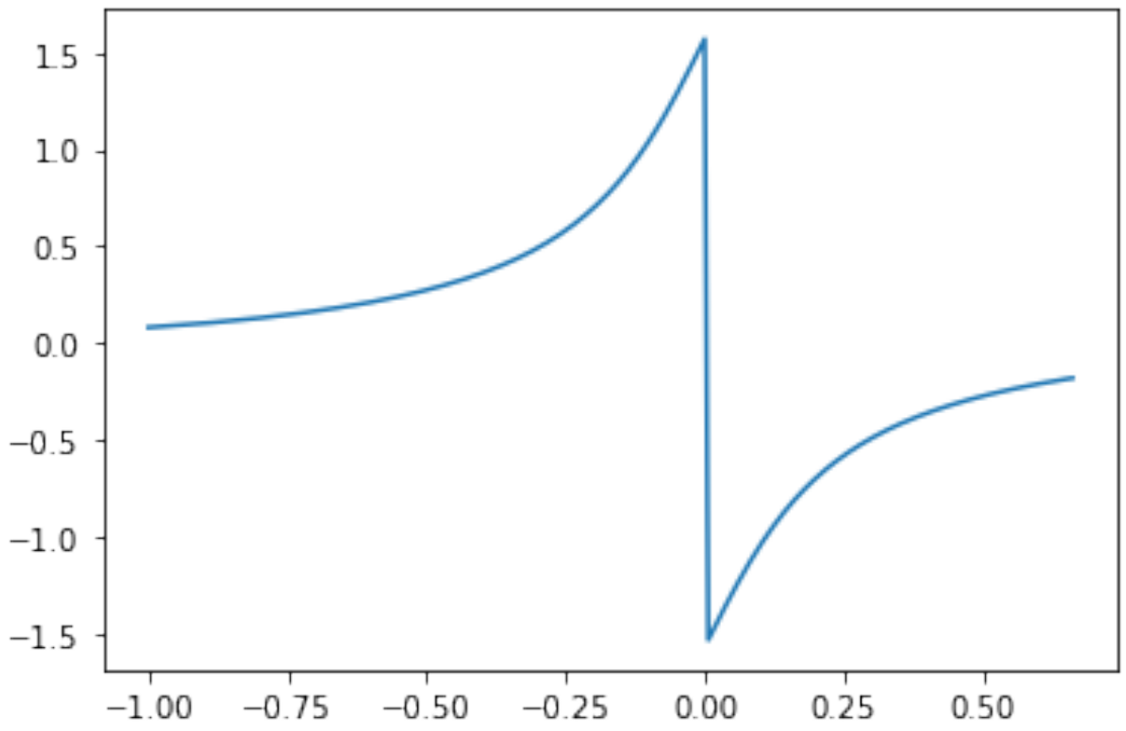
Το σχήμα της φάσης φ θα έπρεπε να είναι έτσι:



Γ : 0.4

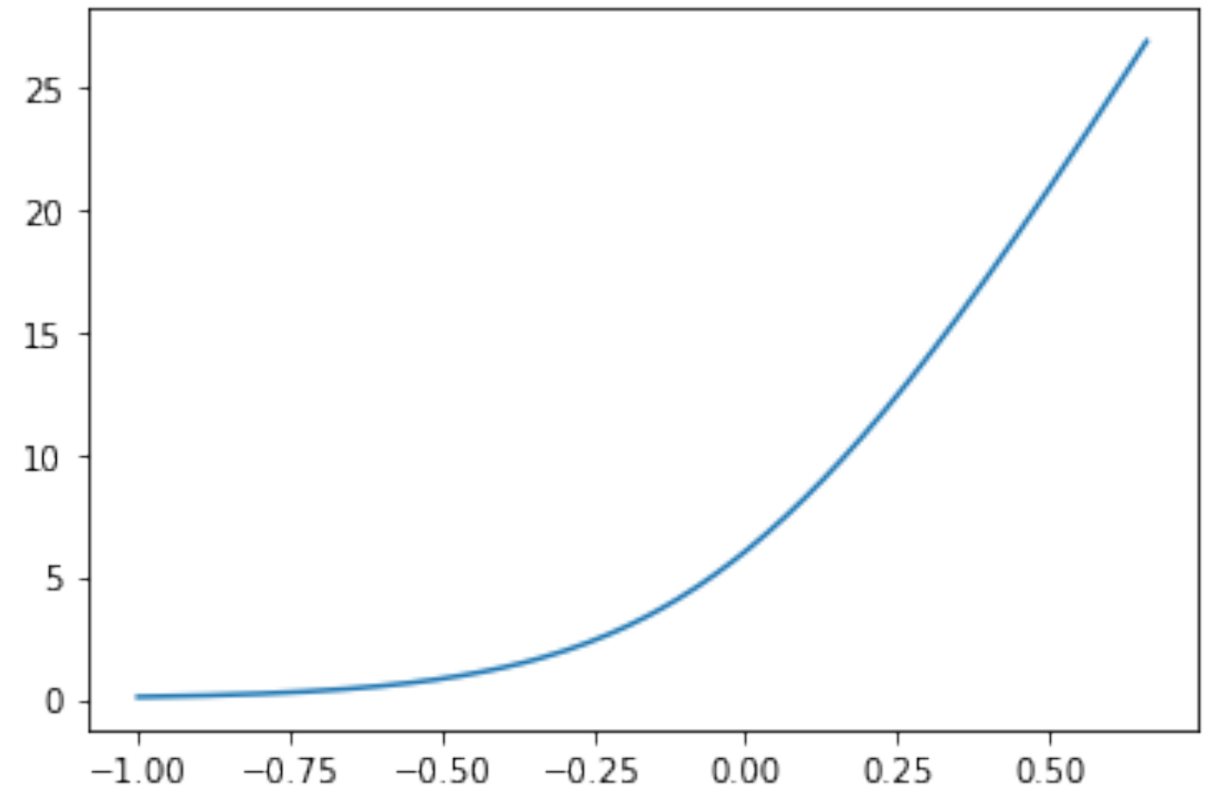
Το ελάχιστο του M ρηχαίνει και η μεταβολή του φ επιβραδύνεται όταν ο συντελεστής ανάκλασης r αυξάνεται.

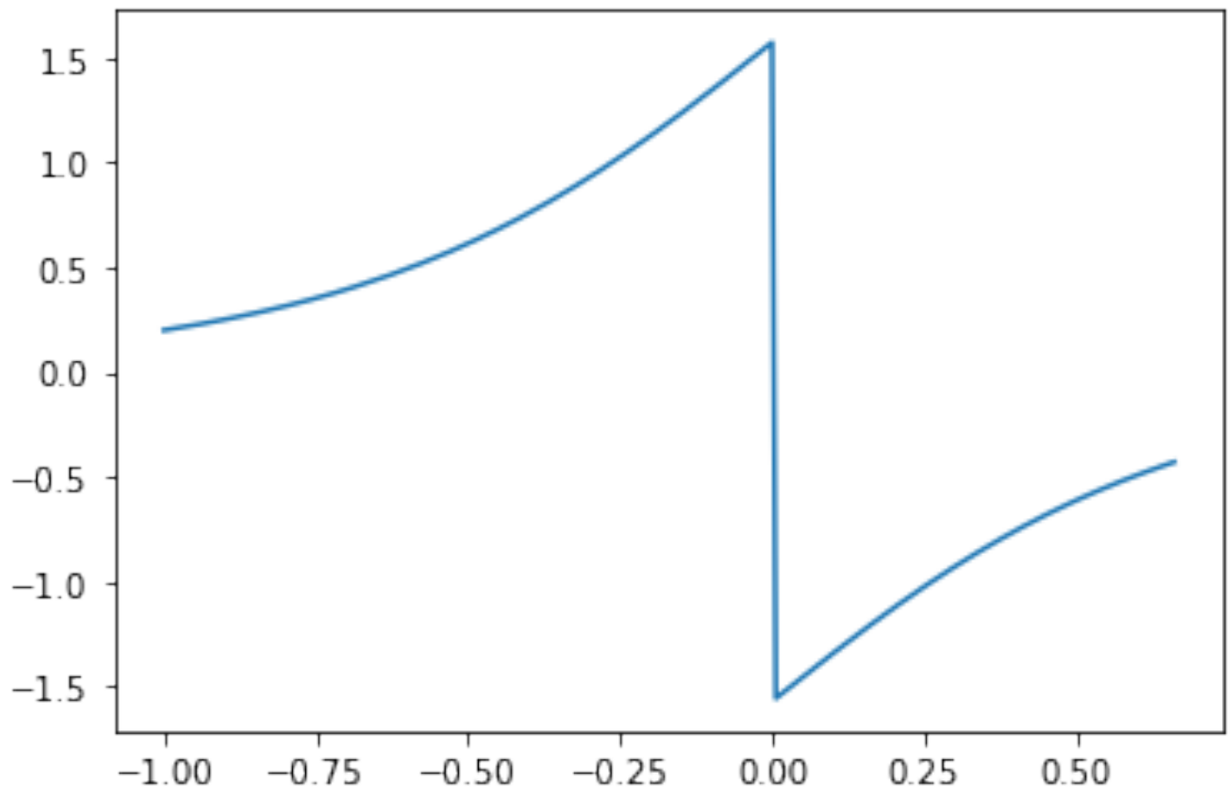




Γ : 1.0

Για r=1 το ελάχιστο του M εξαφανίζεται.





Κινητική ενέργεια, με το τετράγωνο της γραμμικής ταχύτητας εκφρασμένο σαν παράγωγος της ακτίνας ως προς την πολική γωνία (ο χρόνος έχει απαλειφθεί). Το τετράγωνο της ακτίνας είναι ο όρος στροφορμής.

Δυναμική ενέργεια βαρυτικής έλξης. Συνολικά E < 0, για να παραμένει η Γη σε κλειστή τροχιά.

ΕΡΓΑΣΙΕΣ ΠΑΡΑΓΩΓΙΣΗΣ

Παραδοτέες μέχρι 10/12/19

Α.Ενέργεια περιστροφής της Γης γύρω από τον Ήλιο.

Η συνολική ενέργεια ενός πλανήτη μάζας m σε τροχιά γύρω από ένα άστρο μάζας M είναι:

$$E = \frac{GMma(1-\varepsilon^2)}{2r^4}\left[\left(\frac{dr}{d\theta}\right)^2 + r^2\right] - \frac{GMm}{r} \qquad G = 6.67\times10^{-11}\ \mathrm{Nm^2/kg^2}$$

και η εξίσωση που περιγράφει την τροχιά του σε πολικές συντεταγμένες είναι:

$$r = \frac{a(1-\varepsilon^2)}{1-\varepsilon\cos\theta}$$

Εφόσον E < 0, είναι και R < 0. Μάλιστα, R ≃ −2 AU.

η οποία αντιστοιχεί σε έλλειψη με εκκεντρότητα $0<\varepsilon<1$ και μεγάλο ημιάξονα a. Για την τροχιά της Γης γύρω από τον Ήλιο η εκκεντρότητα είναι $\varepsilon=0.0167$ και ο μεγάλος ημιάξονας $a = 1.495\times10^{11}$ m ορίζει την αστρονομική μονάδα μήκους AU. Από τη διατήρηση της μηχανικής ενέργειας, η ποσότητα $1/R = E/(GMm)$, με διάσταση αντίστροφου μήκους, πρέπει να είναι σταθερή. Δουλεύοντας σε ένα σύστημα με μονάδα μήκους AU, υπολογίστε την παράγωγο της ακτινικής απόστασης της Γης r από τον Ήλιο σε δύο γωνίες, $\theta=0$ και $\theta=\pi/2$, με τη μέθοδο Newton-Gregory πρώτου βαθμού και με τη μέθοδο των προσδιοριστέων συντελεστών, επιλέγοντας ένα κατάλληλο βήμα h στην κάθε περίπτωση. Ελέγξτε την ακρίβεια της παραγώγισης σε κάθε περίπτωση συγκρίνοντας τις τιμές της $1/R$ στις δύο γωνίες $\theta=0$ και $\theta=\pi/2$, οι οποίες θα πρέπει να είναι ίσες.

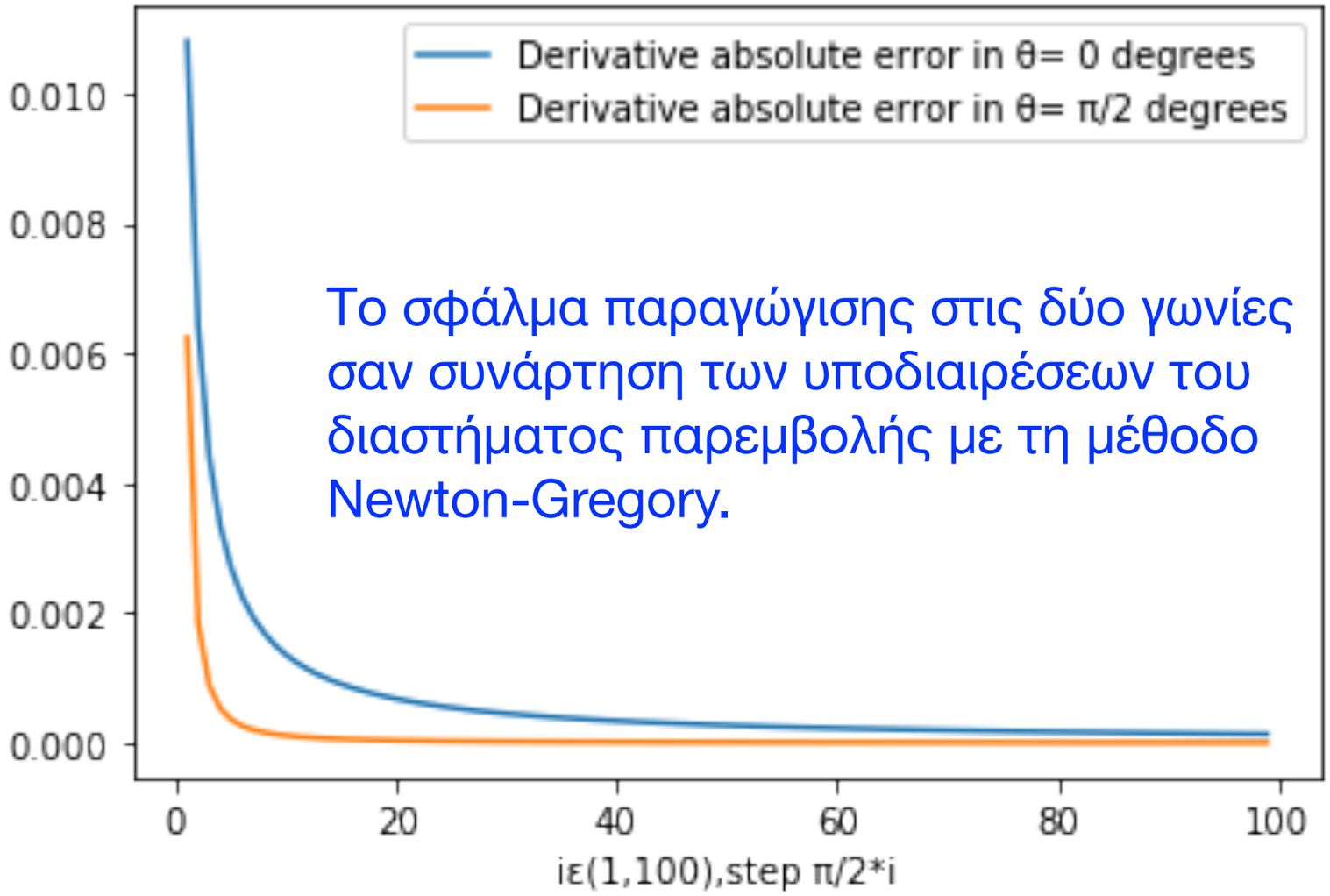
[21]:
```
import matplotlib.pyplot as plt
import numpy as np
import math


def f(x):
    return a*(1-e**2)/(1-e*np.cos(x))

def df(x):
    return -(a*e*(1-e**2)*np.sin(x))/(1-e*np.cos(x))**2

a = 1
e = 0.0167

# Newton-Gregory Method

#testing the step


E0 = []
E1 = []
D0 = []
D1 = []

I = []
DE = []
for i in range(1,100): #gia vima  /2, /4, /6, /8,...
    x = np.arange(-math.pi,math.pi+math.pi/(2*i),math.pi/(2*i))
    f = [a*(1-e**2)/(1-e*math.cos(x[k])) for k in range(len(x)) ]
    df = [-(a*e*(1-e**2)*math.sin(x[k]))/(1-e*math.cos(x[k]))**2 for k in␣
 ↪range(len(x)) ]
    p1_1=(f[2*i+i+1]-f[2*i+i])*2*i/math.pi #paragwgos sto  /2
    p1_0=(f[2*i+1]-f[2*i])*2*i/math.pi #paragwgos sto 0
    D0.append(abs(df[2*i] -p1_0)) #diafora paragwgou metaksi methodou kai tupou␣
 ↪paragwgow sto   = 0
    D1.append(abs(df[2*i+i] - p1_1)) #diafora paragwgou metaksi methodou kai␣
 ↪tupou paragwgow sto   = /2
```

Κώδικας για την πρώτη εργασία παραγώγισης.

```
    I.append(i)

print("Newton Gregory method")
print()
plt.plot(I,D0, label="Derivative absolute error in  = 0 degrees")
plt.plot(I,D1, label="Derivative absolute error in  =  /2 degrees")
plt.xlabel("i (1,100),step  /2*i")
plt.legend()
plt.show()

#we choose i = 40

i = 40

x = np.arange(-math.pi,math.pi+math.pi/(2*i),math.pi/(2*i))
f = [a*(1-e**2)/(1-e*math.cos(x[k])) for k in range(len(x))]
df = [-(a*e*(1-e**2)*math.sin(x[k]))/(1-e*math.cos(x[k]))**2 for k in␣
↪range(len(x)) ]
p1_1=(f[2*i+i+1]-f[2*i+i])*2*i/math.pi #paragwgos sto  /2
p1_0=(f[2*i+1]-f[2*i])*2*i/math.pi #paragwgos sto 0
E0 = a*(1-e**2)/2*f[2*i]**4 *((p1_0)**2 + f[2*i]**2 ) - 1/f[2*i]
E1 = a*(1-e**2)/2*f[2*i+i]**4 *((p1_1)**2 + f[2*i+i]**2 ) - 1/f[2*i+i]

print(f"p1[0 degrees ] = {p1_0} , p1[ /2 degrees] = {p1_1}")
print(f"df/d [0 degrees]={df[2*i]} , df/d [ /2 degrees]={df[2*i+i]}")
print("1/R for   = 0 :",E0,"1/[AU]")
print("1/R for   =  /2 :",E1,"1/[AU]")
print("Error 1/R : ",abs(E0-E1)*100,"%")
print("Absolute error for  =0 degrees",abs(p1_0-df[2*i]))
print("Absolute error for  = /2 degrees",abs(p1_1-df[2*i+i]))
print()


# Method of undetermined coefficients in two neighboar points

d0 = []
d1 = []
Ii = []

for i in range(1,100): #gia vima  /2, /4, /6, /8,...
    y = np.arange(-math.pi,math.pi+math.pi/(2*i),math.pi/(2*i))
    F = [a*(1-e**2)/(1-e*math.cos(y[k])) for k in range(len(y)) ]
    dF = [-(a*e*(1-e**2)*math.sin(y[k]))/(1-e*math.cos(y[k]))**2 for k in␣
↪range(len(y)) ]
    p1=(F[2*i+i]-F[2*i+i-1])*2*i/math.pi #paragwgos sto  /2
    p0=(F[2*i]-F[2*i-1])*2*i/math.pi #paragwgos sto 0
```

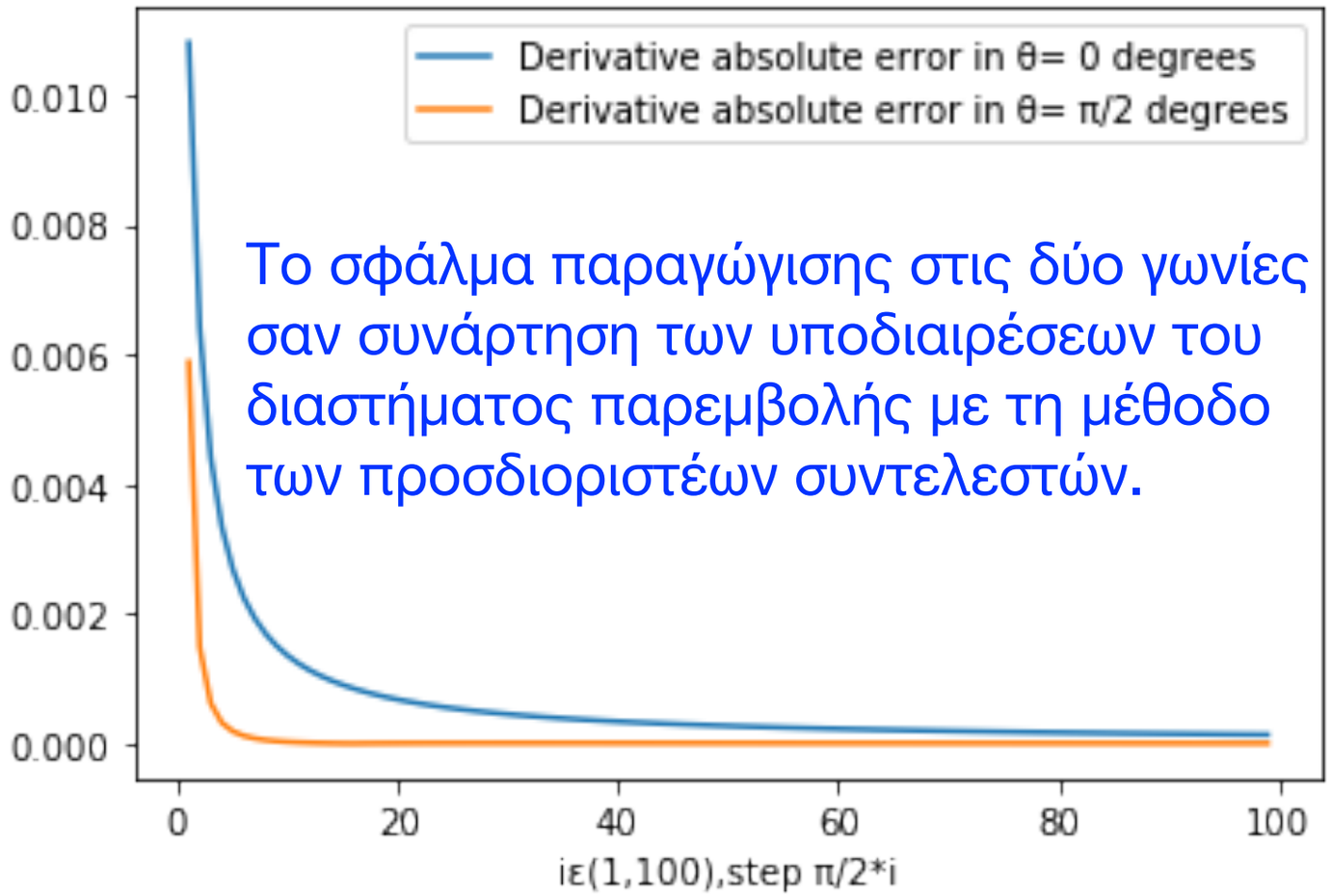
```
    d0.append(abs(dF[2*i] -p0)) #diafora paragwgou metaksi methodou kai tupou␣
 ↪paragwgow sto   = 0
    d1.append(abs(dF[2*i+i] - p1)) #diafora paragwgou metaksi methodou kai tupou␣
 ↪paragwgow sto   = /2
    Ii.append(i)

print("Method of undetermined coefficients in two neighboar points")
print()
plt.plot(Ii,d0, label="Derivative absolute error in  = 0 degrees")
plt.plot(Ii,d1, label="Derivative absolute error in  =  /2 degrees")
plt.xlabel("i (1,100),step  /2*i")
plt.legend()
plt.show()

#we choose i = 40

i = 40

y = np.arange(-math.pi,math.pi+math.pi/(2*i),math.pi/(2*i))
F = [a*(1-e**2)/(1-e*math.cos(y[k])) for k in range(len(y))]
dF = [-(a*e*(1-e**2)*math.sin(y[k]))/(1-e*math.cos(y[k]))**2 for k in␣
 ↪range(len(y)) ]
p1=(F[2*i+i]-F[2*i+i-1])*2*i/math.pi #paragwgos sto  /2
p0=(F[2*i]-F[2*i-1])*2*i/math.pi #paragwgos sto 0
E0 = a*(1-e**2)/2*F[2*i]**4 *((p1_0)**2 + F[2*i]**2 ) - 1/F[2*i]
E1 = a*(1-e**2)/2*F[2*i+i]**4 *((p1_1)**2 + F[2*i+i]**2 ) - 1/F[2*i+i]


print(f"p1[0 degrees ] = {p0} , p1[ /2 degrees] = {p1}")
print(f"df/d [0 degrees]={dF[2*i]} , df/d [ /2 degrees]={dF[2*i+i]}")
print("1/R for   = 0 :",E0,"1/[AU]")
print("1/R for   =  /2 :",E1,"1/[AU]")
print("Error in 1/R : ",abs(E0-E1)*100,"%")
print("Absolute error for  =0 degrees",abs(p0-dF[2*i]))
print("Absolute error for  = /2 degrees",abs(p1-dF[2*i+i]))
print()
print("Newton Gregory method")
print("Absolute error for  =0 degrees",abs(p1_0-df[2*i]))
print("Absolute error for  = /2 degrees",abs(p1_1-df[2*i+i]))
print()
print("Method of undetermined coefficients in two neighboar points")
print("Absolute error for  =0 degrees",abs(p0-dF[2*i]))
print("Absolute error for  = /2 degrees",abs(p1-dF[2*i+i]))
```

```
Newton Gregory method
```



Το σφάλμα παραγώγισης στις δύο γωνίες σαν συνάρτηση των υποδιαιρέσεων του διαστήματος παρεμβολής με τη μέθοδο Newton-Gregory.

```
p1[0 degrees ] = -0.0003389937154476285 , p1[ /2 degrees] =
-0.016680115682004806
df/d [0 degrees]=-3.067280319020087e-16 , df/d [ /2
degrees]=-0.016695342536999985
1/R for   = 0 : -0.431489424666732 1/[AU]
1/R for   =  /2 : -0.501115347243248 1/[AU]
Error 1/R :  6.9625922576516 %
Absolute error for  =0 degrees 0.0003389937154473218
Absolute error for  = /2 degrees 1.522685499517895e-05

Method of undetermined coefficients in two neighboar points
```



Το σφάλμα παραγώγισης στις δύο γωνίες σαν συνάρτηση των υποδιαιρέσεων του διαστήματος παρεμβολής με τη μέθοδο των προσδιοριστέων συντελεστών.

```
p1[0 degrees ] = 0.00033899371544762 85 , p1[ /2 degrees] = -0.016702002297892 94
df/d [0 degrees]=-3.0672803190200874e-16 , df/d [ /2
degrees]=-0.016695342536999985
1/R for   = 0 : -0.431489424666732 1/[AU]
1/R for   =  /2 : -0.501115347243248 1/[AU]
Error in 1/R :  6.9625922576516 %
Absolute error for  =0 degrees 0.00033899371544793 52
Absolute error for  = /2 degrees 6.659760892955419e-06

Newton Gregory method
Absolute error for  =0 degrees 0.00033899371544732 18
Absolute error for  = /2 degrees 1.5226854995178951e-05

Method of undetermined coefficients in two neighboar points
Absolute error for  =0 degrees 0.00033899371544793 52
Absolute error for  = /2 degrees 6.659760892955419e-06
```

Β.Επαγόμενο φορτίο σε επίπεδο αγωγό από σημειακό φορτίο κοντά στον αγωγό.

Όταν πλησιάσουμε ένα σημειακό φορτίο q σε απόσταση d από έναν επίπεδο αγωγό, το πεδίο του φορτίου επάγει μια κατανομή φορτίου πάνω στην επιφάνεια του αγωγού με επιφανειακή πυκνότητα σ. Μπορούμε να υπολογίσουμε αυτή την πυκνότητα με τη λεγόμενη "μέθοδο των εικόνων", δηλαδή υποθέτοντας ένα αντίθετο σημειακό φορτίο ίσου μέτρου q με το αρχικό σε απόσταση d από την άλλη μεριά της επιφάνειας του αγωγού και παραλείποντας τελείως τον αγωγό. Παίρνοντας τον άξονα z των συντεταγμένων κάθετο στην επιφάνεια του αγωγού με $z>0$ από την πλευρά του αρχικού φορτίου, το δυναμικό από την υπόθεση δύο φορτίων (του αρχικού και του "φορτίου-εικόνας") και η επαγόμενη επιφανειακή πυκνότητα φορτίου από αυτό το δυναμικό στον πραγματικό αγωγό δίνονται από τις σχέσεις:

$$\varepsilon_0 V(x,y,z) = \frac{1}{4\pi}\left[\frac{q}{\sqrt{x^2+y^2+(z-d)^2}} - \frac{q}{\sqrt{x^2+y^2+(z+d)^2}}\right] \qquad \sigma = -\varepsilon_0 \frac{\partial V}{\partial z}(z=0)$$

Επιλέγοντας ένα κατάλληλο βήμα h, παραγωγίστε με τη μέθοδο Newton-Gregory δεύτερου βαθμού το δυναμικό $\varepsilon_0 V$ ως προς z στα σημεία $(x,y)=\{(0,0), (1,0), (1,1), (0,2), (-2,2), (-3,0), (-3,3)\}$ cm της επιφάνειας του αγωγού και συγκρίνετε τα αποτελέσματα με τις τιμές από την αναλυτική έκφραση της επιφανειακής πυκνότητας φορτίου στα ίδια σημεία, η οποία δίνεται από τη σχέση:

$$\sigma(x,y) = \frac{-qd}{2\pi(x^2+y^2+d^2)^{3/2}},$$

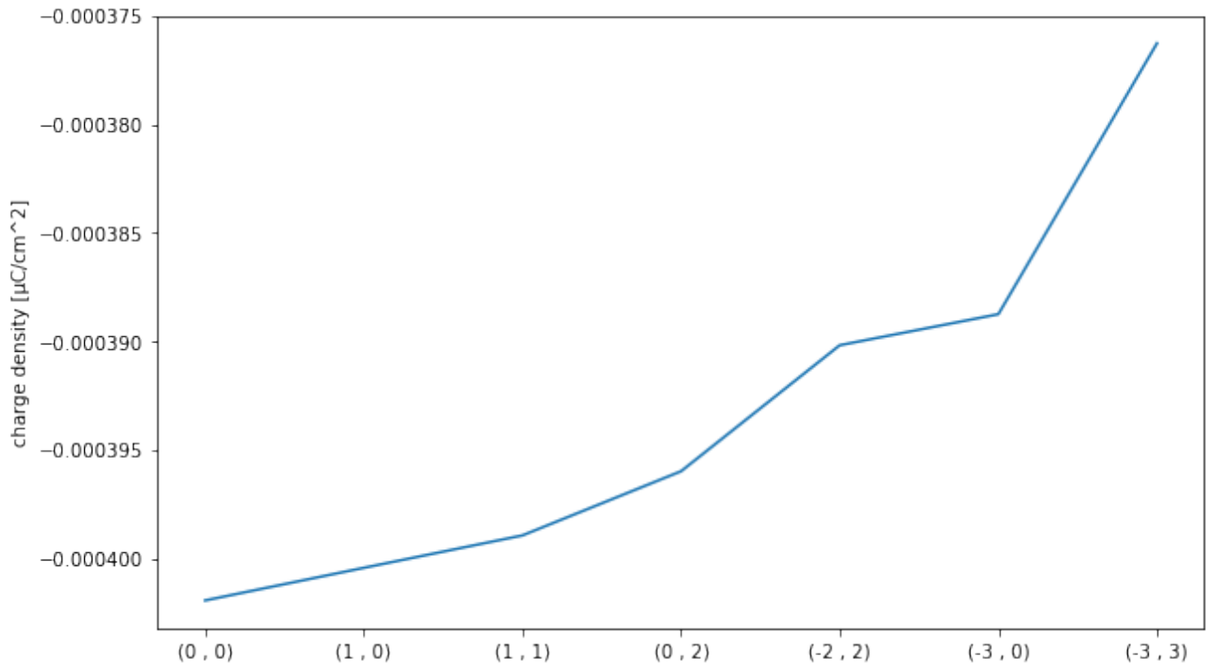
για q=1 μC και d=10 cm. Παρατηρήστε ότι η πυκνότητα φορτίου είναι μέγιστη στο σημείο $(x,y)=(0,0)$ (ακριβώς απέναντι από το σημειακό φορτίο). Πόση ακρίβεια έχει ο υπολογισμός σας;

[130]:

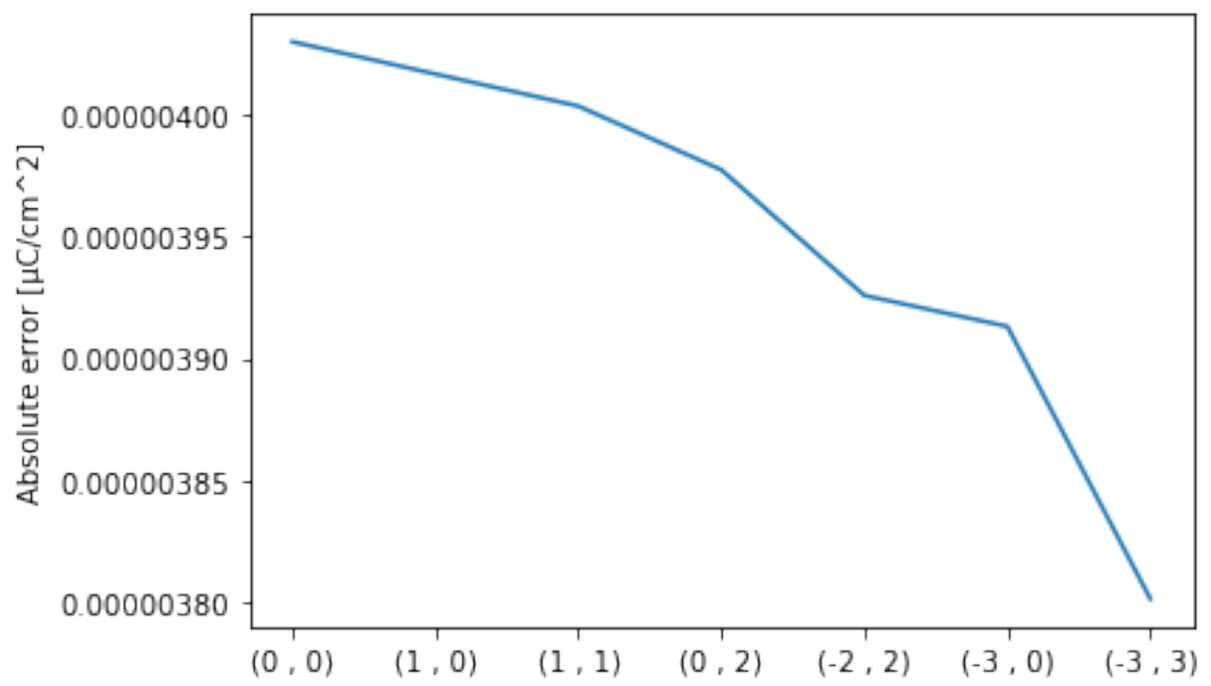
Κώδικας για τη δεύτερη εργασία παραγώγισης.

```
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.arange(-3,4)
ipx = [3,4,4,3,1,0,0]
ipy = [3,3,4,5,5,3,6]
h  =1
q = 1
d = 20
s= []
dV = []
diff = []
I = []
h = 0.32
for i in range(7):
    f2 = 1/4*math.pi * (q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + (2*h - d)**2) - q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + (2*h + d)**2))
    f1 = 1/4*math.pi * (q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + (h - d)**2) - q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + (h + d)**2))
    f0 = 1/4*math.pi * (q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + d**2) - q/math.sqrt( x[ipx[i]]**2 + x[ipy[i]]**2 + d**2))
    df = - ( f2 - 4*f1 + 3*f0 )/2*h
    s.append(-df) ## density Newton Gregory
    dV.append(- q*d/(2*math.pi*(x[ipx[i]]**2+x[ipy[i]]**2+d**2)**(3/2))) # density function
    diff.append(abs(s[i]-dV[i]))
    I.append(f'({x[ipx[i]]} , {x[ipy[i]]})')
plt.figure(figsize=(10,6))
plt.plot(I,s)
plt.ylabel("charge density [ C/cm^2]")
plt.show()
print("max(|charge density|) at (0,0)")
print()
plt.plot(I,diff)
plt.ylabel("Absolute error [ C/cm^2]")
plt.show()
```

```
print("10^(-6) [ C/cm^2] precision")
```




```
max(|charge density|) at (0,0)
```




```
10^(-6) [ C/cm^2] precision
```

Εδώ φαίνεται να μειώνεται το απόλυτο σφάλμα δσ καθώς μεγαλώνει η απόσταση από την προβολή του φορτίου στο αγώγιμο επίπεδο, αλλά ταυτόχρονα μεγαλώνει και το |σ| στο επάνω σχήμα. Άρα, το σχετικό σφάλμα δσ/|σ| θα ήταν πιο χρήσιμο για την εκτίμηση της ακρίβειας.

ΕΡΓΑΣΙΕΣ ΟΛΟΚΛΗΡΩΣΗΣ

Παραδοτέες μέχρι 20/12/19

Α.Μήκος τροχιάς βλήματος.

Βλήμα βάλλεται με αρχική ταχύτητα $v_0 = 800$ m/s υπό γωνία $\theta = 30^\circ$ ως προς το έδαφος. Το μήκος της τροχιάς του δίνεται από τη σχέση:

$$s = \int_0^{(2/g)v_0 \sin\theta} \sqrt{v_0^2 \cos^2\theta + (v_0 \sin\theta - gt)^2}\, dt$$

όπου t είναι ο χρόνος πτήσης του βλήματος και g=9.8 m/s^2 είναι η ένταση της βαρύτητας στην επιφάνεια της Γης. Ολοκληρώστε αριθμητικά αυτή την έκφραση με τον κανόνα του τραπεζίου. Προσαρμόστε το βήμα h της ολοκλήρωσης έτσι ώστε να επιτύχετε ακρίβεια της τάξης του 10^{-4} σε σύγκριση με την ακριβή τιμή από την έκφραση:

$$s = \frac{v_0^2}{g}\left[\sin\theta + \frac{1}{2}\cos^2\theta \ln\left(\frac{1+\sin\theta}{1-\sin\theta}\right)\right].$$

In [2]:
```
# vivliothikes
import numpy as np
import matplotlib.pyplot as plt
```

A. Mikos Troxias Vlimatos

Κώδικας για την πρώτη εργασία ολοκλήρωσης.

In [3]:
```
# i akrivis ekfrasi
def s_exact():
    # python - trigonometrikes se aktinia, log is ln
    sin_th = np.sin(np.deg2rad(th))
    cos_th = np.cos(np.deg2rad(th))
    return (v0**2/g) * (sin_th + (1/2) * cos_th**2 * np.log((1 + sin_th)/(1 - sin_th)))
```

In [4]:
```
# ypologismos tis synartisis
def s(t):
    sin_th = np.sin(np.deg2rad(th))
    cos_th = np.cos(np.deg2rad(th))
    return (v0**2 * cos_th**2 + (v0 * sin_th - (g * t))**2)**(1/2)
```

In [5]:
```
# statheres
v0 = 800
g = 9.8
th = 30
```

In [30]:
```
s_ex = s_exact()
print(f"S Exact: {s_ex}")
```

```
S Exact: 59557.851967382274
```

In [37]:
```
a = 0.
```

```
b = (2. / g) * v0 * np.sin(np.deg2rad(th))

n = 50000000
sum = 0

# synthiki gia tin eyresi ton diastimaton
# se sxolio giati den fainetai na ginetai pote true
# while np.abs(sum - s_ex) > 1e-4:

h = (b - a) / n

t0 = a
sum = 0.

# print("Trapeziou")
# print("  x \t    s(t)")

for i in range(n):
    t = t0 + i * h

    s_t = s(t);

    if i == 0 or i == n-1:
        sum += s_t
    else:
        sum += 2. * s_t

    # print(f"{t:4.1f} \t {s_t:8.5f}")

sum *= h/2

print(f"\n  I = {sum:.6f}")
# n +=1 # ayxisi tis epanalipsis
```

```
  I = 59557.850661
```

In [38]:
```
print(f"h = {h:.10f}, n = {n-1}, Difference = {np.abs(sum - s_ex):.10f}")
```

```
h = 0.0000016327, n = 50000000, Difference = 0.0013060948
```

me 50000000 epanalipsis exoume akriveia ton 2 dekadikon psifion (yparxei kapoio lathos stin ylopoiisi;;;)

B. Elefteri energeia stin aktinovolia melanoy somatos

Kleisti Morfi

In [22]:
```
# kleisti morfi
I_fourier = (np.pi ** 4)/15
```

Seira

In [23]:
```
# ypologismos seiras me prosthiki oron mexri tin akriveia
sum = 0
n = 1
while np.abs((6 * sum) - I_fourier) >= 1e-4:
    sum = sum + (1/n**4)
    n = n + 1
```

Έχει παρερμηνευτεί η έννοια της ζητούμενης ακρίβειας 10^{-4}: αυτή αναφέρεται στα σημαντικά ψηφία και όχι στα δεκαδικά., δηλ. στο σχετικό σφάλμα και όχι στο απόλυτο. Εδώ, με 50 εκατομμύρια επαναλήψεις, έχει επιτευχθεί ακρίβεια τάξης 10^{-8}.

Κώδικας για τη δεύτερη εργασία ολοκλήρωσης: υπολογισμός με σειρά.

Β.Ελεύθερη ενέργεια στην ακτινοβολία μελανού σώματος.

Η ελεύθερη ενέργεια Helmholtz στην ακτινοβολία του μελανού σώματος συναρτήσει του όγκου V και της θερμοκρασίας T του σώματος δίνεται από την έκφραση:

$$F(T,V)=\frac{V(kT)^4}{\pi^2(\hbar c)^3}\int_0^\infty x^2\ln(1-e^{-x})dx \qquad\qquad x\equiv\frac{\hbar\omega}{kT}$$

όπου ω είναι η κυκλική συχνότητα των εκπεμπόμενων φωτονίων και k είναι η σταθερή του Boltzmann. Η έκφραση αυτή περιλαμβάνει ένα ολοκλήρωμα με άπειρο επάνω όριο. Με παραγοντική ολοκλήρωση, αυτό το ολοκλήρωμα γράφεται:

$$\frac{1}{3}\int_0^\infty \ln(1-e^{-x})d(x^3)=\left[x^3\ln(1-e^{-x}\right]_0^\infty-\frac{1}{3}\int_0^\infty\frac{x^3dx}{e^x-1}$$

Ο ολοκληρωμένος όρος μηδενίζεται στα δύο όρια και δεν συνεισφέρει. Μας ενδιαφέρει λοιπόν το δεύτερο ολοκλήρωμα, το οποίο αναπτύσσεται σε σειρά:

$$I\equiv\int_0^\infty\frac{x^3dx}{e^x-1}=6\sum_{n=1}^\infty\frac{1}{n^4}$$

αλλά υπολογίζεται και σε κλειστή μορφή με ανάλυση Fourier:

$$I=\frac{\pi^4}{15}.$$

1. Μελετήστε τη σύγκλιση της σειράς προσθέτοντας όρους μέχρι το άθροισμα να διαφέρει από την αναλυτική τιμή το πολύ κατά $O(10^{-4})$.

2. Επιλέξτε μια μέγιστη τιμή αποκοπής $x_{\max}$ για το επάνω όριο και υπολογίστε το ολοκλήρωμα με τον κανόνα του Simpson. Συγκρίνετε το αποτέλεσμα με την αναλυτική τιμή για βήμα $h=x_{\max}/10$ και $x_{\max}/50$. Προσαρμόστε το βήμα και την τιμή αποκοπής ώστε να επιτύχετε ακρίβεια $O(10^{-4})$. Ποια είναι η σχέση ακρίβειας-κόστους σε υπολογισμούς για την αριθμητική ολοκλήρωση και την προσέγγιση του ολοκληρώματος με σειρά;

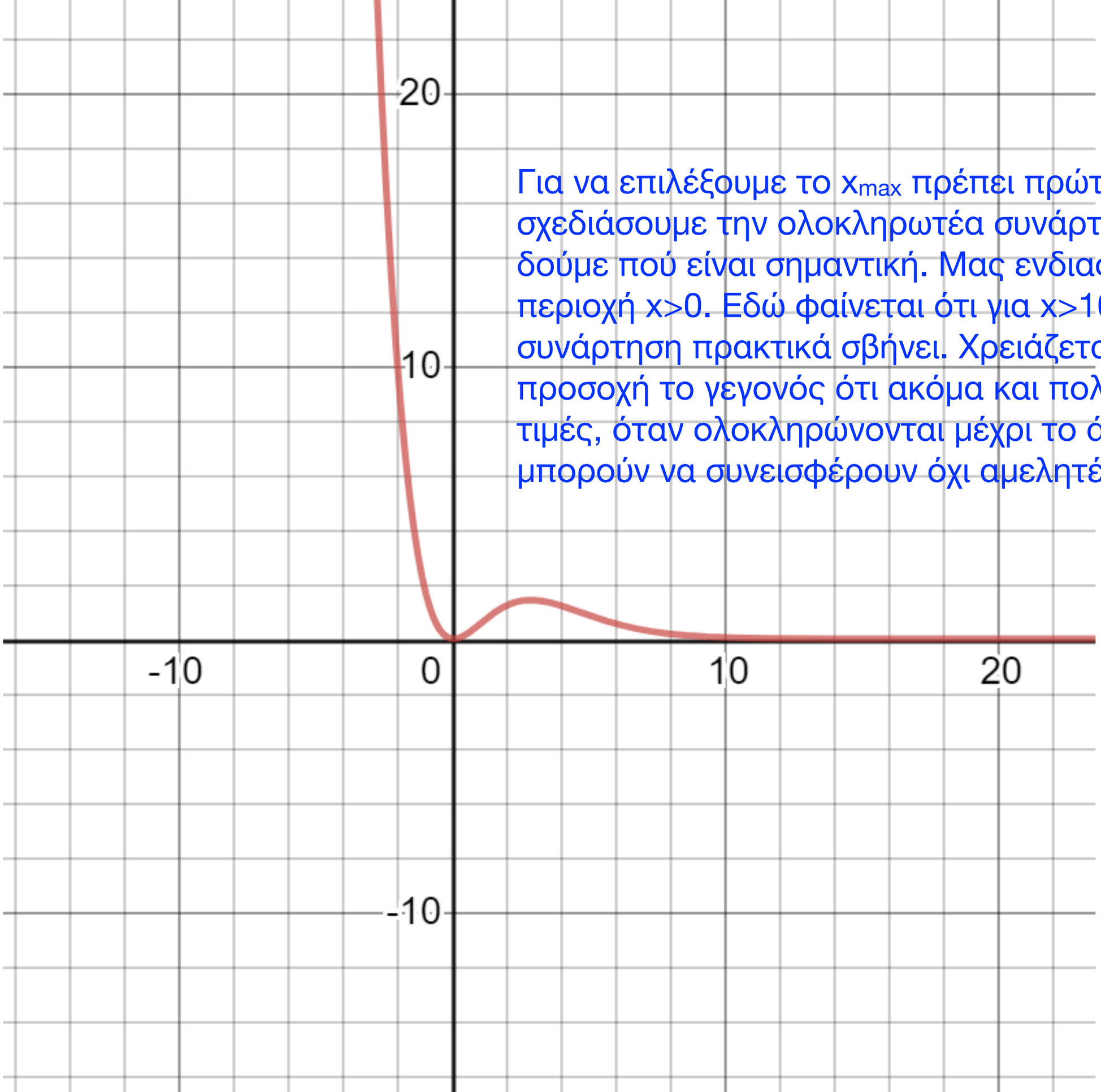
In [24]:
```
# apotelesmata seiras
print(f"Terms n = {n-1}")
print(f"I Series = {6 * sum}")
print(f"I Fourier = {I_fourier}")
print(f"I Series - I Fourier = {np.abs((6 * sum) - I_fourier):.6f}")
```

```
Terms n = 27
I Series = 6.493843297481234
I Fourier = 6.493939402266828
I Series - I Fourier = 0.000096
```

Oloklíroma - Simpson

In [25]:
```
# h synartisi toy oloklíromatos
def f(x):
    return x**3/(np.exp(x) - 1)
```

Κώδικας για τη δεύτερη εργασία ολοκλήρωσης: υπολογισμός με ολοκλήρωση Simpson.



Για να επιλέξουμε το x_{max} πρέπει πρώτα να σχεδιάσουμε την ολοκληρωτέα συνάρτηση και δούμε πού είναι σημαντική. Μας ενδιαφέρει η περιοχή $x>0$. Εδώ φαίνεται ότι για $x>10$ η συνάρτηση πρακτικά σβήνει. Χρειάζεται όμως προσοχή το γεγονός ότι ακόμα και πολύ μικρές τιμές, όταν ολοκληρώνονται μέχρι το άπειρο, μπορούν να συνεισφέρουν όχι αμελητέα.

In [26]:
```
# oloklirosi me Simpson

def simpson(x_max, n):
    # x_max - megisti timi apokopis
    # n - vimata


    # x0
    # arixiki timi 1e-15 (i synartisi den orizetai sto 0)
    # to apotelesma den allazei akoma kai an gia kathe alli epanalipsi
    # meta thn 1h kanoyme tin timi toy x0 = 0
    x0 = 1e-15

    h = x_max/n;

    sum = 0.

    for i in range(n):

        x = x0 + i * h

        f_x = f(x)

        if i == 0 or i == (n-1):
            sum += f_x
        elif i%2 == 0:
            sum += 2. * f_x
        else:
            sum += 4. * f_x

    sum *= h / 3.

    return sum
```

Μέθοδος Simpson.

In [27]:
```
# dialegoume os timi apokopis to 10 poy tha mas dosei
# h = 1 stin proti periptosi kai sti synexeia 0.02

x_max = 10
```

In [28]:
```
# 1h periptosi

n = 10
I_simpson = simpson(x_max, n)

print(f"I Simpson = {I_simpson:.15f}")
print(f"I Fourier = {I_fourier:.15f}")
print(f"I Simpson - I Fourier = {np.abs(I_simpson- I_fourier):.6f}")
```

```
I Simpson = 6.343408520095359
I Fourier = 6.493939402266828
I Simpson - I Fourier = 0.150531
```

In [29]:
```
# 2h periptosi

n = 50
I_simpson = simpson(x_max, n)

print(f"I Simpson = {I_simpson:.15f}")
print(f"I Fourier = {I_fourier:.15f}")
print(f"I Series - I Fourier = {np.abs(I_simpson- I_fourier):.6f}")
```

```
I Simpson = 6.418482954267916
I Fourier = 6.493939402266828
I Series - I Fourier = 0.075456
```

In [30]:
```
# dokimazei syndiasmoys x_max kai n gia na kathorisei ton kalytero syndiasmo
def adjust():
    for x_max in np.arange(10,20,0.5):
        for n in range(10, 54, 2):  # to n prepei na einai artion stin Simpson
            I_simpson = simpson(x_max, n)
            if np.abs(I_simpson - I_fourier) < 1e-4:
                return (x_max, int(n))
```

In [31]:
```
x_max, n = adjust()
print(f"x_max = {x_max}, n = {n}")
```

```
x_max = 15.0, n = 24
```

In [32]:
```
I_simpson = simpson(x_max, n)

print(f"I Simpson = {I_simpson:.15f}")
print(f"I Fourier = {I_fourier:.15f}")
print(f"I Simpson - I Fourier = {np.abs(I_simpson - I_fourier):.6f}")
```

```
I Simpson = 6.493937753504380
I Fourier = 6.493939402266828
I Simpson - I Fourier = 0.000002
```

=> timi apokopis x_max = 15.0 kai bimata n = 24

Sygrisi

Stin periptorisi tis seiras theloume 27 vimata ara kai epanalipseis gia na petyxoyme tin epithimiti akriveia Stin periptosi tis Simpson theloyme 24 epanalipseis (afoy broume tin timi apokopis)

Στην ακόλουθη εναλλακτική λύση της δεύτερης εργασίας ολοκλήρωσης έχει εφαρμοστεί μια μέθοδος εύρεσης της x_{max} με βελτιστοποίηση της ακρίβειας, χρησιμοποιώντας συναρτήσεις από μια ιδιωτική βιβλιοθήκη C του χρήστη με το όνομα numcalc.h. Το x_{max} προσδιορίζεται με τη συνθήκη η ακρίβεια προσέγγισης του ολοκληρώματος με σειρά (το σχετικό σφάλμα) να είναι 10^{-5}.

Ερώτηση Β: Για το πρώτο κομμάτι της ερώτησης βρήκαμε μια τιμή του n για την οποία το άθροισμα θεωρείται ακριβές. Στην συνέχεια, για να βρούμε το σημείο αποκοπής του ολοκληρώματος, χρησιμοποιήσαμε μια συνάρτηση αριθμητικής εύρεσης ρίζας που βρίσκεται στην βιβλιοθήκη "numcalc.h". Θέσαμε την ακρίβειά της ως:

$$10^{-5}$$

Εδώ θα έπρεπε να χρησιμοποιηθεί όχι η ολοκληρωτέα συνάρτηση dI, αλλά το ολοκλήρωμα Simpson της dI (με 10 ή 50 βήματα αντίστοιχα) σαν συνάρτηση του x_{max}.

```c
include <stdio.h>
#include <math.h>
#include "overlay.h"
#include "numcalc.h"
#include "myphys.h"

double dI (double x);

int main (void)
{
    double const pi=ph_const("pi");
    double sum=0., I_exact=(pow(pi,4))/(15.);
    int n=1;

    do{
        sum += 6/(pow(n,4)+0.);
        ++n;
    }while (fabs(sum-I_exact) > pow(10,-4));
    printf("\n sum = %lf, n = %d (last term: %e)\n\n", sum, n, 6/(pow(n,4)+0.));

    analyze_level(dI, 0, 100, 6/(pow(n,4)+0.), 1, pow(10,-5));
    /*seeing where we want the cutoff point to be*/
    printf("\n");
    return 0;
}
double dI (double x)
{
    double e=ph_const("e");
    return (pow(x,3))/(pow(e,x)-1.);
}
```

Η ολοκληρωτέα συνάρτηση.

code block 1

Η dI γίνεται 10^{-5} για x=20.5. Αυτό όμως δεν εξασφαλίζει ότι το ολοκλήρωμα που αποκόπτεται θέτοντας x_{max}=20.5 είναι < 10^{-4}.

Με τα εξής αποτελέσματα:

```
sum = 6.493843, n = 28 (last term: 9.761558e-006)


the function intersects with y = 0.000010 in 1 point:
20.500000 (f(20.50) = 0.000011)
```

results for question B1

Ο πρώτος όρος της σειράς που παραλείπεται είναι ο 28ος < 10^{-5}. Άρα, χρειάζονται 27 όροι για να έχει η προσέγγιση με σειρά την επιθυμητή ακρίβεια.

Για το δεύτερο ερώτημα, ο αλγόριθμος ήταν ως εξής:

```c
#include <stdio.h>
#include <math.h>
#include "overlay.h"
#include "myphys.h"

double dI (double x);

int main (void)
{
    double pi=ph_const("pi"), I=0., I_exact=(pow(pi,4))/(15.);
    int points = 10;
    double a=0.1, b=20.5, h=(b-a)/(points+0.);
    /*dI is undefined for a=0, so we'll use a=0.1*/

    for(int i=0; i<=points; i+=2){/*integration using Simpson's rule*/
        I += dI(a+h*i) + 4*dI(a+h*(i+1)) + dI(a+h*(i+2));
    }
    I *= h/(3+0.);
    printf("\n step: %lf\n numerical: %lf\n analytic: %lf\n diff: %e\n\n"
    , h, I, I_exact, (I-I_exact));
    return 0;
}
double dI (double x)
{
    double e=ph_const("e");
    return (pow(x,3))/(pow(e,x)-1.);
}
```

code block 2

Αρχικά για 10 σημεία και στη συνέχεια για 50, τα αποτελέσματα ήταν τα εξής:

```
step: 2.040000
numerical: 6.723491
analytic: 6.493939
diff: 2.295520e-001

step: 0.408000
numerical: 6.494060
analytic: 6.493939
diff: 1.210525e-004
```

results for question B2

Η δεύτερη περίπτωση κρίνεται ικανοποιητικά ακριβής, ενώ τελικά μπορούμε να παρατηρήσουμε πως η σειρά χρειάστηκε μόλις 28 επαναλήψεις για να φτάσει σε ακρίβεια καλύτερη από αυτήν που πετύχαμε με 50 επαναλήψεις χρησιμοποιώντας αριθμητική ολοκλήρωση.

Η μέθοδος με την σειρά κρίνεται καλύτερη σε αυτό το πρόβλημα. Βέβαια, δεν μπορούμε να έχουμε μια τέτοια λύση για κάθε πρόβλημα ή μάλλον για τα περισσότερα, ενώ η αριθμητική ολοκλήρωση εφαρμόζεται εύκολα σε πολλά.

Απόσπασμα της βιβλιοθήκης C numcalc.h.

Η μέθοδος διχοτόμησης.

```
double bisection (double (*f)(), double a, double b, double accuracy)
/*uses bisection logic to find a single root of f(x)=0 in the given
 interval*/
{
    double c;
    do{
        c = (a+b)/2.0;
        if ((*f)(a)*(*f)(c) < 0)
        b = c;
        if ((*f)(c)*(*f)(b) < 0)
        a = c;
    }
    while (fabs((*f)(c)) > accuracy);
    return c;
}
```

Εύρεση αριθμού ριζών (σημείων μηδενισμού μιας συνάρτησης) με τη μέθοδο της διχοτόμησης.

```
int sort_roots (double (*f)(), double a, double b, double
 distinctive_ability,
double accuracy, double *roots)
/*standard function that stores roots in an array*/
{
    int cuts, i, rN=0;
    cuts = ceil((b-a)/(distinctive_ability + 0.));/*using our desirable
     distinctive
    ability to determine the width of each interval*/
    double approx[cuts], h, bolzano;
    h = (b-a)/(cuts + 0.);/*note that the new width might be slighly
     smaller than the
    desirable distinctive ability, but we don't mind! It just means it's
     more accurate*/

    approx[0] = (*f)(a);
    for (i=0; i<cuts; ++i){
        approx[i+1] = (*f)(a + h*(i+1));
        bolzano = approx[i+1]*approx[i];

        if (bolzano <= 0.){/*we will now use one of our single root
         functions*/
            roots[rN] = bisection(f, a + h*i, a + h*(i+1), accuracy);
            ++rN;
        }
    }
    return rN;/*returns the number of roots*/
}
```

Προσέγγιση των ριζών με τη μέθοδο της διχοτόμησης.

```
void analyze_roots (double (*f)(), double a, double b, double
 distinctive_ability, double accuracy)
/*announces the roots of the function on the console*/
{
    int cuts, i, rN=0;
    cuts = ceil((b-a)/(distinctive_ability + 0.));/*using our desirable
     distinctive
    ability to determine the width of each interval*/
```

```
double approx[cuts], h, bolzano, roots[cuts];

h = (b-a)/(cuts + 0.);/*note that the new width might be slighly
 smaller than
the desirable distinctive ability, but we don't mind! It just means
 it's more
accurate*/

approx[0] = (*f)(a);
for (i=0; i<cuts; ++i)
{
    approx[i+1] = (*f)(a + h*(i+1));
    bolzano = approx[i+1]*approx[i];

    if (bolzano <= 0.)/*we will now use one of our single root
     functions*/
    {
        roots[rN] = bisection (f, a + h*i, a + h*(i+1), accuracy);
        ++rN;
    }
}
/*here we will announce the results.*/
if (rN == 0)
    printf("\n the function has no roots.");
else
{
    if (rN == 1){
        printf("\n the function has 1 root:");
    }else{
        printf("\n the function has %d roots:", rN);
    }
    for (i=0; i<rN; ++i){
        printf("\n %lf (f(%.2f) = %.1e)", roots[i], roots[i],
         (*f)(roots[i]));
    }
}
}
```

Η μέθοδος που εφαρμόζεται στη δεύτερη εργασία ολοκλήρωσης.

```
void analyze_level (double (*f)(), double a, double b, double level,
double distinctive_ability, double accuracy)
/*like analyze_roots, but gives you the solutions of f(x)=level*/
{
    double g (double x){
    /*the roots of this function will be the roots of f(x)=level*/
        return (*f)(x) - level;
    }
    int cuts, i, rN=0;
    cuts = ceil((b-a)/(distinctive_ability + 0.));/*using our desirable
     distinctive
    ability to determine the width of each interval*/
    double approx[cuts], h, bolzano, roots[cuts];

    h = (b-a)/(cuts + 0.);/*note that the new width might be slighly smaller
    than the desirable distinctive ability, but that just means
```

```
it's more accurate*/

approx[0] = g(a);
for (i=0; i<cuts; ++i){
    approx[i+1] = g(a + h*(i+1));
    bolzano = approx[i+1]*approx[i];

    if (bolzano <= 0.){/*we will now use one of our single root
     functions*/
        roots[rN] = bisection (g, a + h*i, a + h*(i+1), accuracy);
        ++rN;
    }
}
//here we will announce the results.
if (rN == 0){
    printf("\n the function does not intersect with y = %lf.", level);
}else{
    if (rN == 1){
        printf("\n the function intersects with y = %lf in 1 point:",
         level);
    }else{
        printf("\n the function intersects with y = %lf in %d points:",
         level, rN);
        }
        for (i=0; i<rN; ++i){
            printf("\n %lf (f(%.2f) = %lf)", roots[i], roots[i],
             (*f)(roots[i]));
        }
    }
}
```