1 a)

O valor de *h* foi alterado de 0,25 para 0,125 pois calculando a fórmula do erro, para que o mesmo fique menor que 10^-4 teria que fazer no mínimo 28 iterações e com o h=0,25 obtêm-se apenas 24 intervalos, portanto divido o h por 2.

Usando regra de trapézio temos:

Pontos e Área Total:

x	y	y0+2y1+...+2yn-1+yn
-3	0,0001234101	0,0001234101
-2,875	0,0002572088	0,0005144175
-2,75	0,0005195759	0,0010391518
-2,625	0,00101728	0,00203456
-2,5	0,0019304578	0,0038609156
-2,375	0,0035506547	0,0071013093
-2,25	0,0063297252	0,0126594504
-2,125	0,0109367826	0,0218735651
-2	0,0183156612	0,0366313224
-1,875	0,0297292482	0,0594584965
-1,75	0,046770666	0,0935413321
-1,625	0,0713167401	0,1426334802
-1,5	0,1053992968	0,2107985937
-1,375	0,1509775055	0,3019550109
-1,25	0,209611487	0,4192229739
-1,125	0,2820630605	0,564126121
-1	0,3678795533	0,7357591066
-0,875	0,4650432966	0,9300865933
-0,75	0,5697829224	1,1395658448
-0,625	0,6766339267	1,3532678534
-0,5	0,7788008424	1,5576016848
-0,375	0,8688150935	1,737630187
-0,25	0,9394130807	1,8788261614
-0,125	0,9844964417	1,9689928834
0	1	2
0,125	0,9844964417	1,9689928834
0,25	0,9394130807	1,8788261614
0,375	0,8688150935	1,737630187
0,5	0,7788008424	1,5576016848
0,625	0,6766339267	1,3532678534
0,75	0,5697829224	1,1395658448
0,875	0,4650432966	0,9300865933
1	0,3678795533	0,7357591066
1,125	0,2820630605	0,564126121
1,25	0,209611487	0,4192229739
1,375	0,1509775055	0,3019550109
1,5	0,1053992968	0,2107985937
1,625	0,0713167401	0,1426334802
1,75	0,046770666	0,0935413321
1,875	0,0297292482	0,0594584965
2	0,0183156612	0,0366313224
2,125	0,0109367826	0,0218735651
2,25	0,0063297252	0,0126594504
2,375	0,0035506547	0,0071013093
2,5	0,0019304578	0,0038609156
2,625	0,00101728	0,00203456
2,75	0,0005195759	0,0010391518
2,875	0,0002572088	0,0005144175
3	0,0001234101	0,0001234101
		28,3586088505
	h/3 * C27	**1,7724130532**
	Áreal Total	**1,9999539647**

Implementação:

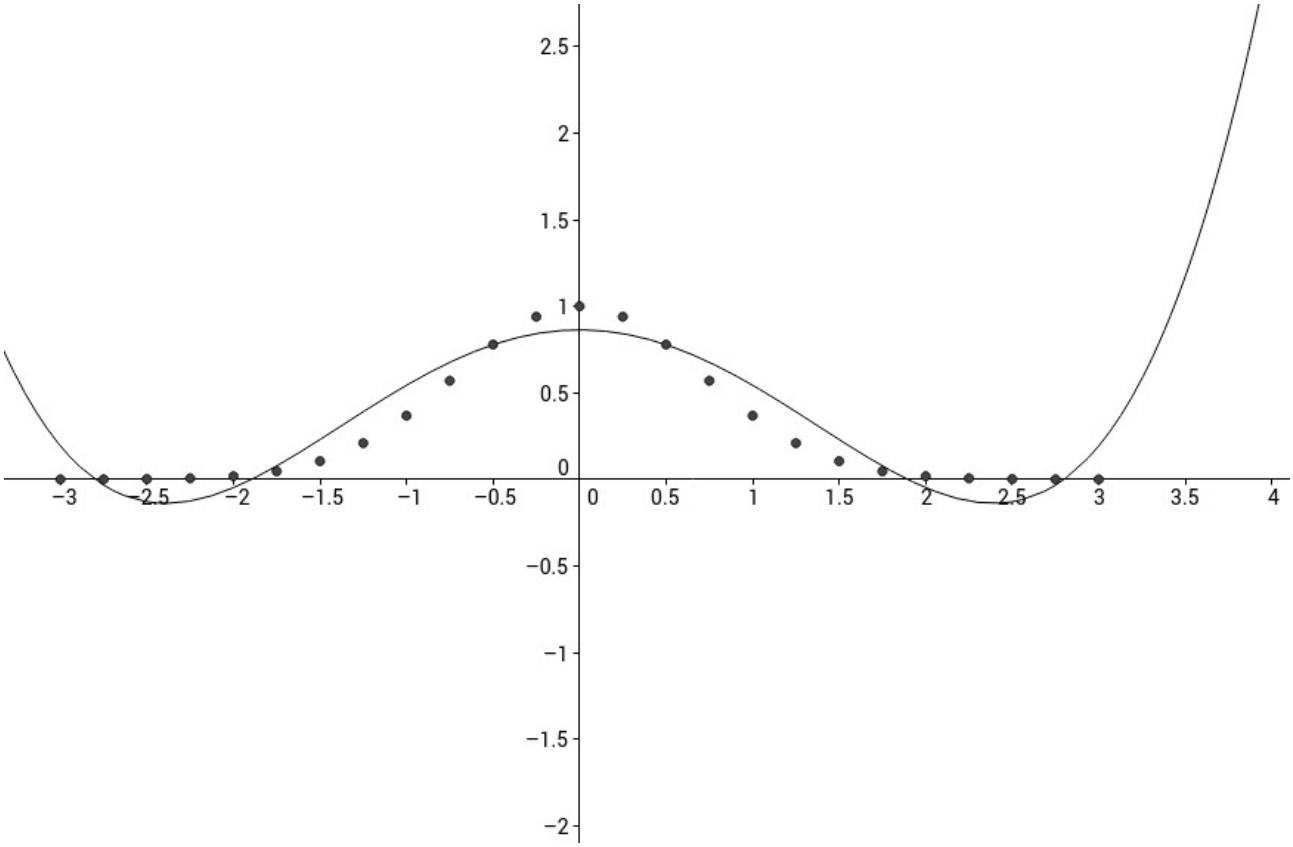
```
# -*- coding:utf-8 -*-

import numpy as np
from math import *

h = 0.125
a = -3.0
b = 3.0
n = int(((b-a)/h)+1)

x = np.linspace(-3.0, 3.0, n)

y = 2.71828182846**-(x**2)

for i in range(1, n-1):
        y[i] = 2*y[i]

AT = (h/2)*sum(y)
AT = AT*(2/sqrt(pi))
print AT
```

1 b)

$r^2 = 0.93019728591$

Curva ajustada: y = 0,86366 – 0,35061x^2 + 0,03068x^4

Gráfico:

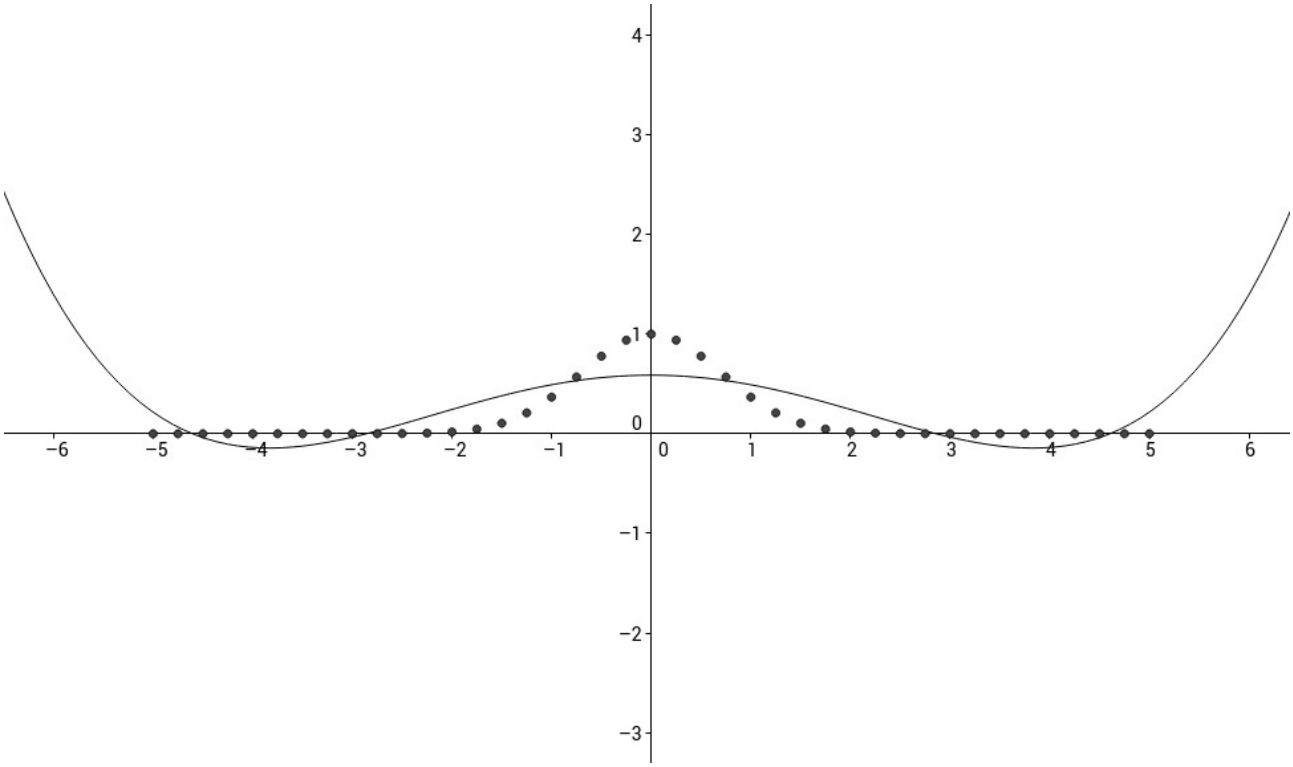


1 d)

$r^2 = 0.727134571136$

Curva Ajustada: y = 0,588241 -0,0996784x^2 + 0,00339294x^4

Gráfico:



1 b) Implementação do Ajuste de Curva(polinômio de grau 5):

```
import numpy as np
from math import *

linhas = 6
colunas = 7
soma = 0.0

h = 0.125
a = -3.0
b = 3.0
n = int(((b-a)/h)+1)

x = np.linspace(a, b, n)
x2 =  x**2
x3 =  x**3
x4 =  x**4
x5 =  x**5
x6 =  x**6
x7 =  x**7
x8 =  x**8
x9 =  x**9
x10 = x**10

somx = sum(x)
somx2 = sum(x2)
somx3 = sum(x3)
somx4 = sum(x4)
somx5 = sum(x5)
somx6 = sum(x6)
somx7 = sum(x7)
somx8 = sum(x8)
somx9 = sum(x9)
somx10 = sum(x10)

y = 2.71828182846**(-(x**2))

y2 = y**2
somy2 = sum(y2)
xy = x*y

x2y = x2*y
x3y = x3*y
x4y = x4*y
x5y = x5*y

somy = sum(y)
somxy = sum(xy)
somx2y = sum(x2y)
somx3y = sum(x3y)
somx4y = sum(x4y)
somx5y = sum(x5y)
```

```
matriz = [
[    n-1,        somx,          somx2,          somx3,      somx4,       somx5,    somy],
[   somx,       somx2,          somx3,          somx4,      somx5,       somx6,   somxy],
[  somx2,       somx3,          somx4,          somx5,      somx6,       somx7,  somx2y],
[  somx3,       somx4,          somx5,          somx6,      somx7,       somx8,  somx3y],
[  somx4,       somx5,          somx6,          somx7,      somx8,       somx9,  somx4y],
[  somx5,       somx6,          somx7,          somx8,      somx9,      somx10,  somx5y],
]

# Gauss
def zera(l, ll, x, y):
        m = l[x]/ll[y]
        for x in range(0, len(l)):
                l[x] = l[x] - (m*ll[x])

for j in range(0, colunas-2):
        for i in range(j, linhas):
                if i != j:
                        zera(matriz[i], matriz[j], j, j)

b = []
for i in range(linhas-1, -1, -1):
        b.append(matriz[i][colunas-1])

b[linhas-1] = matriz[linhas-1][colunas-1] / matriz[linhas-1][colunas-2]
soma = 0.0
for i in range(linhas-2, -1, -1):
        #print i
        for j in range(colunas-2, i, -1):
                #print j
                soma = soma + matriz[i][j]*b[j]

        soma = matriz[i][colunas-1] - soma
        b[i] = soma / matriz[i][i]
        soma = 0.0

for i in range(0, linhas):
        print b[i]

print ''
print 'y =',round(b[0],12), '+', round(b[1], 12),'x','+', round(b[2], 12),'x^2','+', round(b[3], 12),'x^3','+', round(b[4],
12),'x^4','+', round(b[5], 12),'x^5'

soma = 0.0
ybar = []
for i in range(0,len(x)):
   ybar.append((b[0] + (b[1] * x[i]) + (b[2] * (x[i] ** 2)) + (b[3] * (x[i] ** 3)) + (b[4] * (x[i] ** 4)) + (b[5] * (x[i]
** 5))))
   soma = soma + ((y[i] - (b[0] + b[1] * x[i] + (b[2] * (x[i] ** 2)) + (b[3] * (x[i] ** 3)) + (b[4] * (x[i] ** 4)) +
(b[5] * (x[i] ** 5)))) ** 2)

baixo = somy2 - (somy2 / n)
r2 = 1 - (soma / baixo)

print ''
print 'r2 =',r2
```

1 c)

Ao momento em que aumenta o intervalo a função obtida pelo ajuste vai ficando cada vez menos precisa, se o intervalo for muito grande já não fica muito parecida com a função erf(x). Resumindo para problemas no intervalo [-3, 3] acredito ser eficiente o ajuste, mas para outro intervalo já não recomendo.

2 a)

Para i = 1 o valor já é dado, nesse caso é 1, ou seja y(1) = 1, o próximo valor obtêm-se da seguinte formula:

$$y(i) = y(i-1)+h*f(x(i), y(i))$$

Aplicando essa formula de i=2 até i=n, obtêm-se o seguinte resultado:

h=0,25:

h	xi	y'i	y(xi)	Erro yi – y(xi)
0,25	1	1	1	0
0,25	1,25	1	1,0219569066	0,0219569066
0,25	1,5	1,04	1,0672623542	0,0272623542
0,25	1,75	1,0931555556	1,1220712265	0,0289156709
0,25	2	1,1517705069	1,1812322183	0,0294617114
0,25	2,25	1,2128308639	1,2424553856	0,0296245216
0,25	2,5	1,2749499128	1,3046037109	0,0296537981
0,25	2,75	1,3374250129	1,3670703687	0,0296453558
0,25	3	1,3998783392	1,4295160741	0,029637735

h=0,05:

h	xi	y'i	y(xi)	Erro yi – y(xi)
0,05	1	1	1	0
0,05	1,05	1	1,0011535538	0,0011535538
0,05	1,1	1,0022675737	1,0042817279	0,0020141542
0,05	1,15	1,0063152614	1,0089826281	0,0026673667
0,05	1,2	1,0117818831	1,014952314	0,0031704309
0,05	1,25	1,0183942331	1,0219569066	0,0035626734
0,05	1,3	1,0259419444	1,029813689	0,0038717445
0,05	1,35	1,0342605141	1,038377995	0,0041174809
0,05	1,4	1,0432195507	1,0475339193	0,0043143686
0,05	1,45	1,0527144573	1,0571876113	0,004473154
0,05	1,5	1,0626604319	1,0672623542	0,0046019222
0,05	1,55	1,0729880643	1,0776948972	0,004706833
0,05	1,6	1,0836400541	1,0884326869	0,0047926328
0,05	1,65	1,0945687323	1,0994317492	0,004863017
0,05	1,7	1,1057341629	1,1106550521	0,0049208892
0,05	1,75	1,1171026757	1,1220712265	0,0049685508
0,05	1,8	1,1286457172	1,1336535567	0,0050078395
0,05	1,85	1,1403389446	1,1453791782	0,0050402336
0,05	1,9	1,1521615033	1,1572284331	0,0050669298
0,05	1,95	1,1640954468	1,1691843495	0,0050889026
0,05	2	1,1761252682	1,1812322183	0,0051069501
0,05	2,05	1,1882375168	1,1933592458	0,005121729
0,05	2,1	1,2004204854	1,2055542677	0,0051337823
0,05	2,15	1,2126639512	1,2178075118	0,0051435606
0,05	2,2	1,2249589607	1,2301104006	0,0051514398
0,05	2,25	1,2372976514	1,2424553856	0,0051577341
0,05	2,3	1,2496731007	1,2548358079	0,0051627072
0,05	2,35	1,2620791997	1,2672457806	0,0051665809
0,05	2,4	1,2745105461	1,2796800886	0,0051695425
0,05	2,45	1,2869623532	1,2921341036	0,0051717504
0,05	2,5	1,2994303721	1,3046037109	0,0051733388
0,05	2,55	1,3119108252	1,3170852469	0,0051744217
0,05	2,6	1,3244003492	1,3295754451	0,005175096
0,05	2,65	1,3368959455	1,3420713897	0,0051754442
0,05	2,7	1,3493949379	1,3545704745	0,0051755366
0,05	2,75	1,3618949354	1,3670703687	0,0051754333
0,05	2,8	1,3743937999	1,3795689853	0,0051751854
0,05	2,85	1,3868896183	1,392064455	0,0051748367
0,05	2,9	1,3993806777	1,4045551021	0,0051744244
0,05	2,95	1,4118654439	1,4170394243	0,0051739803
0,05	3	1,4243425426	1,4295160741	0,0051735315

h=0,1:

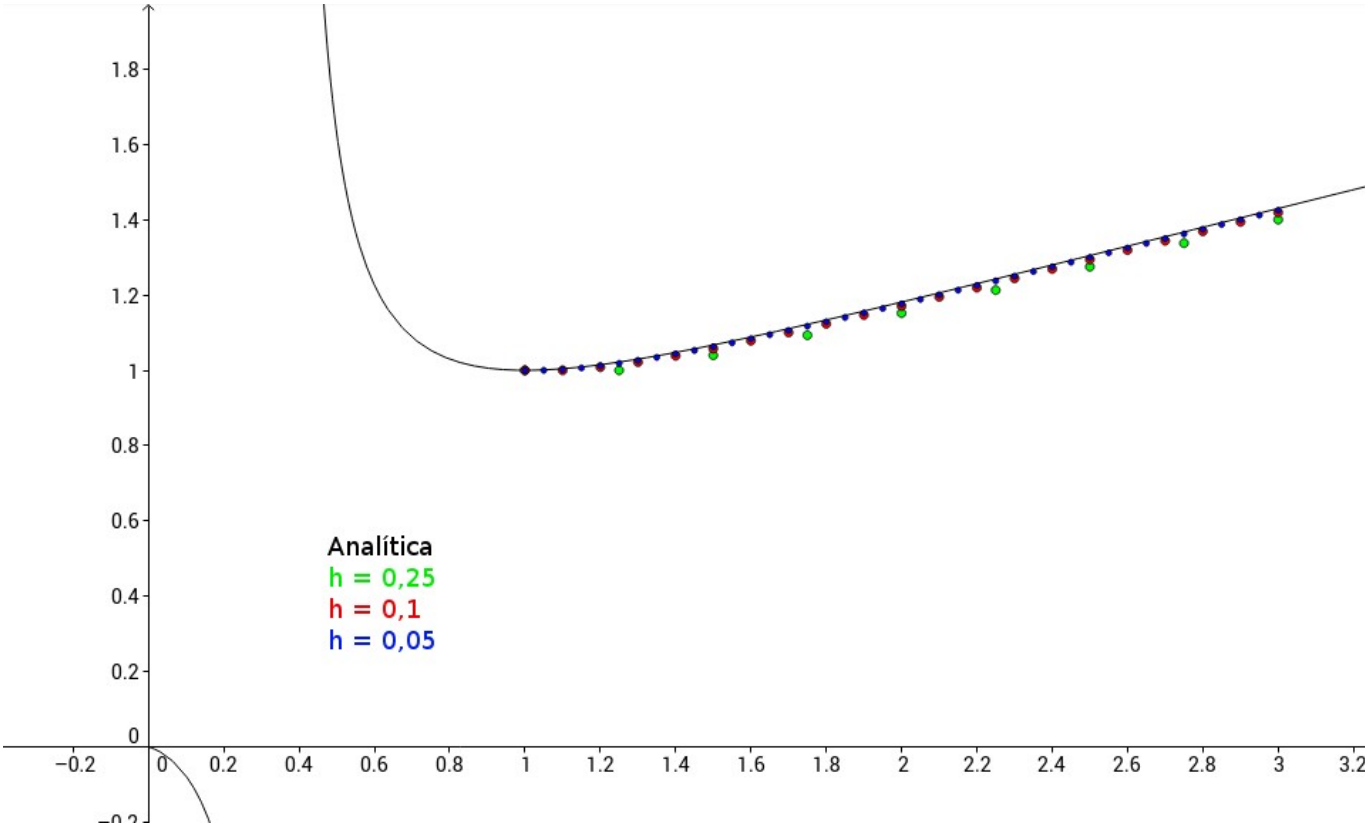
h	xi	y'i	y(xi)	Erro: |yi – y(xi)|
0,1	1	1	1	0
0,1	1,1	1	1,0042817279	0,0042817279
0,1	1,2	1,0082644628	1,014952314	0,0066878512
0,1	1,3	1,0216894717	1,029813689	0,0081242172
0,1	1,4	1,0385147342	1,0475339193	0,009019185
0,1	1,5	1,0576681921	1,0672623542	0,009594162
0,1	1,6	1,0784610936	1,0884326869	0,0099715933
0,1	1,7	1,1004321647	1,1106550521	0,0102228874
0,1	1,8	1,1232620516	1,1336535567	0,0103915052
0,1	1,9	1,1467235965	1,1572284331	0,0105048365
0,1	2	1,1706515696	1,1812322183	0,0105806487
0,1	2,1	1,1949235206	1,2055542677	0,0106307471
0,1	2,2	1,219447266	1,2301104006	0,0106631346
0,1	2,3	1,2441524799	1,2548358079	0,010683328
0,1	2,4	1,2689849054	1,2796800886	0,0106951832
0,1	2,5	1,2939022853	1,3046037109	0,0107014256
0,1	2,6	1,3188714468	1,3295754451	0,0107039984
0,1	2,7	1,3438661785	1,3545704745	0,010704296
0,1	2,8	1,3688656624	1,3795689853	0,0107033229
0,1	2,9	1,3938532983	1,4045551021	0,0107018038
0,1	3	1,4188158138	1,4295160741	0,0107002604

Implementação:

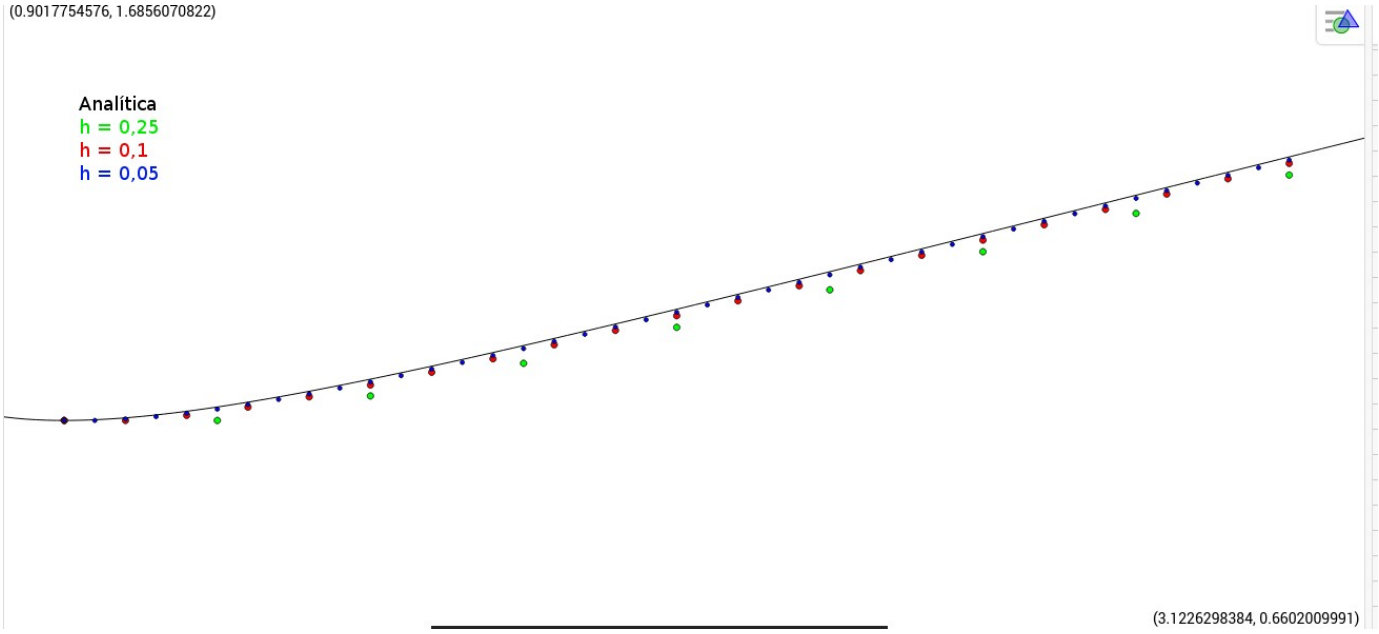
```
# -*- coding:utf-8 -*-

import numpy as np
from math import *

h = 0.25
a = 1.0
b = 3.0
n = int(((b-a)/h)+1)

x = list(np.linspace(a, b, n))
y = x[:]
yx = x[:]

for i in range(1, n):
	y[i] = y[i-1]+( h*((y[i-1]/x[i-1]) - (((y[i-1]/x[i-1])**2))))

for i in xrange(n):
	yx[i] = x[i]/(1+log(x[i]))

print 'xi\t', "\ty'i", '\ty(xi)', '\tErro'
for i in xrange(n):
	print x[i], '\t', y[i], '\t', yx[i], '\t', y[i]-yx[i]
```

2 b)

Visão Geral:

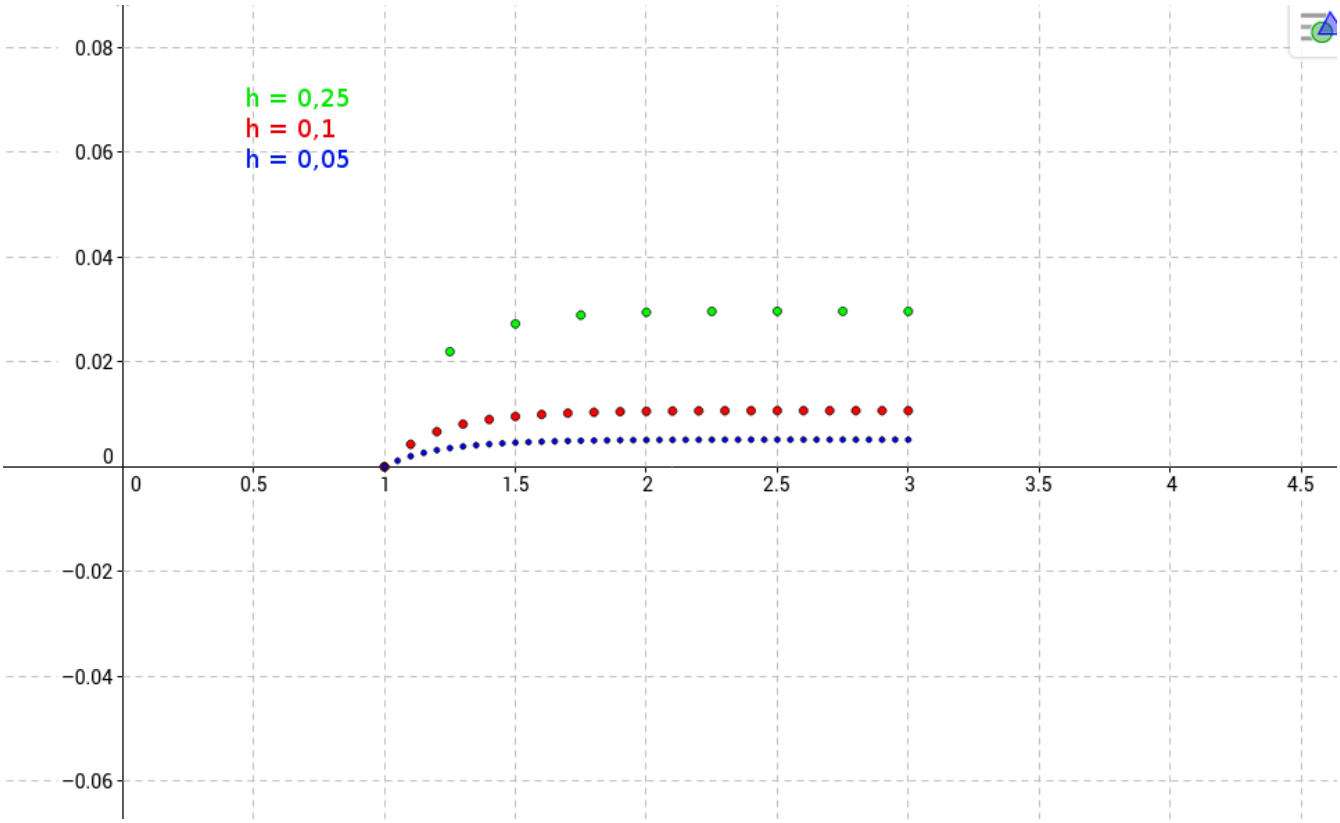


Zoom:



2 c)

Erros:



3 a)
Usando o método de Runge-Kutta de 4ª ordem:

h=0,05:

xi	y'i	y(xi)	Erro yi – y(xi)
1	1	1	0,00E+00
1,05	1,0011535458	1,0011535538	8,00E-09
1,1	1,004281715	1,0042817279	1,29E-08
1,15	1,008982612	1,0089826281	1,61E-08
1,2	1,014952296	1,014952314	1,81E-08
1,25	1,0219568872	1,0219569066	1,94E-08
1,3	1,0298136687	1,029813689	2,02E-08
1,35	1,0383779742	1,038377995	2,08E-08
1,4	1,0475338981	1,0475339193	2,11E-08
1,45	1,05718759	1,0571876113	2,14E-08
1,5	1,0672623327	1,0672623542	2,15E-08
1,55	1,0776948757	1,0776948972	2,16E-08
1,6	1,0884326654	1,0884326869	2,16E-08
1,65	1,0994317276	1,0994317492	2,16E-08
1,7	1,1106550306	1,1106550521	2,16E-08
1,75	1,1220712049	1,1220712265	2,15E-08
1,8	1,1336535352	1,1336535567	2,15E-08
1,85	1,1453791568	1,1453791782	2,15E-08
1,9	1,1572284116	1,1572284331	2,14E-08
1,95	1,1691843281	1,1691843495	2,14E-08
2	1,181232197	1,1812322183	2,13E-08
2,05	1,1933592245	1,1933592458	2,13E-08
2,1	1,2055542464	1,2055542677	2,13E-08
2,15	1,2178074906	1,2178075118	2,13E-08
2,2	1,2301103794	1,2301104006	2,12E-08
2,25	1,2424553644	1,2424553856	2,12E-08
2,3	1,2548357867	1,2548358079	2,12E-08
2,35	1,2672457594	1,2672457806	2,12E-08
2,4	1,2796800675	1,2796800886	2,11E-08
2,45	1,2921340825	1,2921341036	2,11E-08
2,5	1,3046036898	1,3046037109	2,11E-08
2,55	1,3170852258	1,3170852469	2,11E-08
2,6	1,329575424	1,3295754451	2,11E-08
2,65	1,3420713685	1,3420713897	2,11E-08
2,7	1,3545704534	1,3545704745	2,11E-08
2,75	1,3670703476	1,3670703687	2,11E-08
2,8	1,3795689642	1,3795689853	2,12E-08
2,85	1,3920644338	1,392064455	2,12E-08
2,9	1,4045550809	1,4045551021	2,12E-08
2,95	1,4170394031	1,4170394243	2,12E-08
3	1,4295160529	1,4295160741	2,12E-08

h=0,25:

xi	y'i	y(xi)	Erro yi – y(xi)
1	1,0000000000	1	0,00E+00
1,25	1,0219421812	1,0219569066	1,47E-05
1,5	1,0672463025	1,0672623542	1,61E-05
1,75	1,1220552439	1,1220712265	1,60E-05
2	1,1812164297	1,1812322183	1,58E-05
2,25	1,2424397315	1,2424553856	1,57E-05
2,5	1,3045881155	1,3046037109	1,56E-05
2,75	1,3670547681	1,3670703687	1,56E-05
3	1,4295004191	1,4295160741	1,57E-05

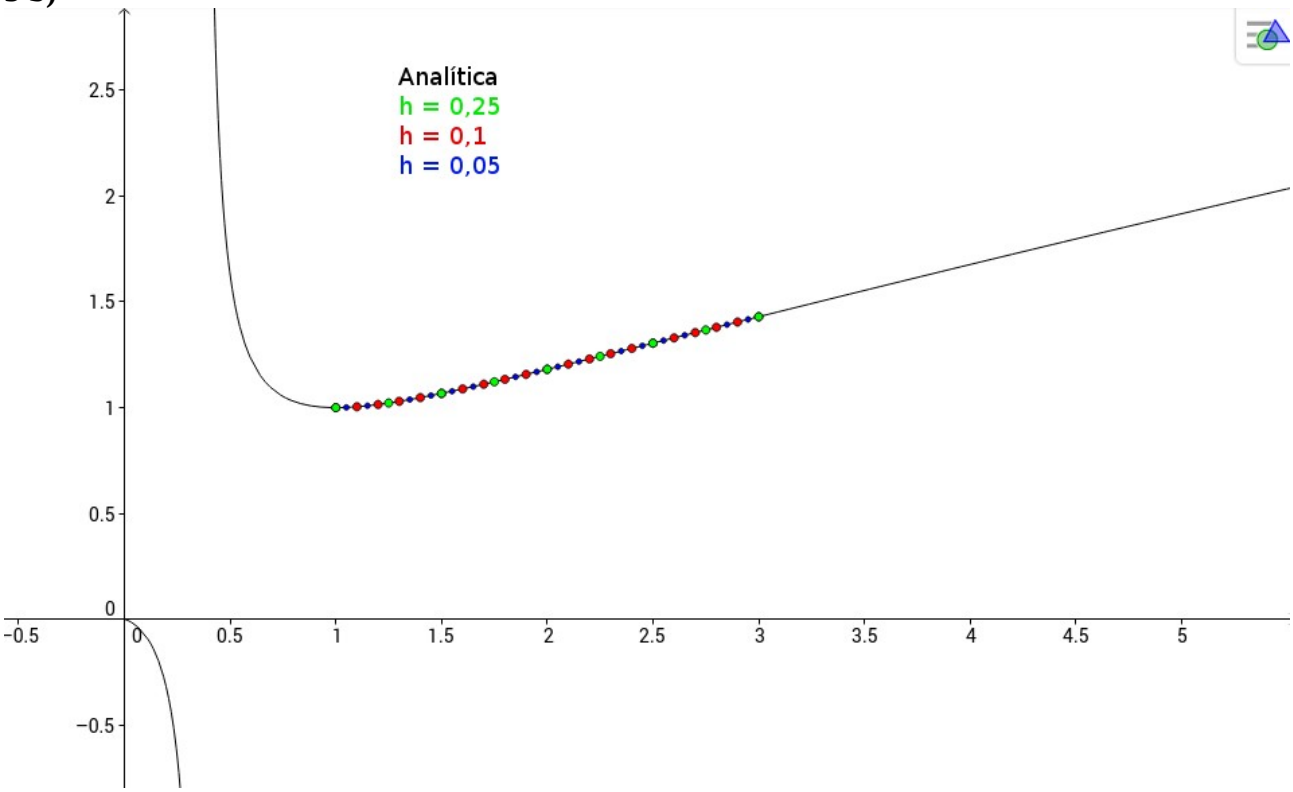
h=0,1:

xi	**y'i**	**y(xi)**	**Erro: |yi – y(xi)|**
1	1	1	0,00E+00
1,1	1,0042815038	1,0042817279	2,24E-07
1,2	1,0149520033	1,014952314	3,11E-07
1,3	1,0298133426	1,029813689	3,46E-07
1,4	1,0475335583	1,0475339193	3,61E-07
1,5	1,0672619878	1,0672623542	3,66E-07
1,6	1,0884323193	1,0884326869	3,68E-07
1,7	1,1106546852	1,1106550521	3,67E-07
1,8	1,1336531911	1,1336535567	3,66E-07
1,9	1,157228069	1,1572284331	3,64E-07
2	1,1812318557	1,1812322183	3,63E-07
2,1	1,2055539064	1,2055542677	3,61E-07
2,2	1,2301100404	1,2301104006	3,60E-07
2,3	1,2548354484	1,2548358079	3,59E-07
2,4	1,2796797297	1,2796800886	3,59E-07
2,5	1,3046033523	1,3046037109	3,59E-07
2,6	1,3295750866	1,3295754451	3,59E-07
2,7	1,3545701158	1,3545704745	3,59E-07
2,8	1,3795686264	1,3795689853	3,59E-07
2,9	1,4045547426	1,4045551021	3,59E-07
3	1,429515714	1,4295160741	3,60E-07

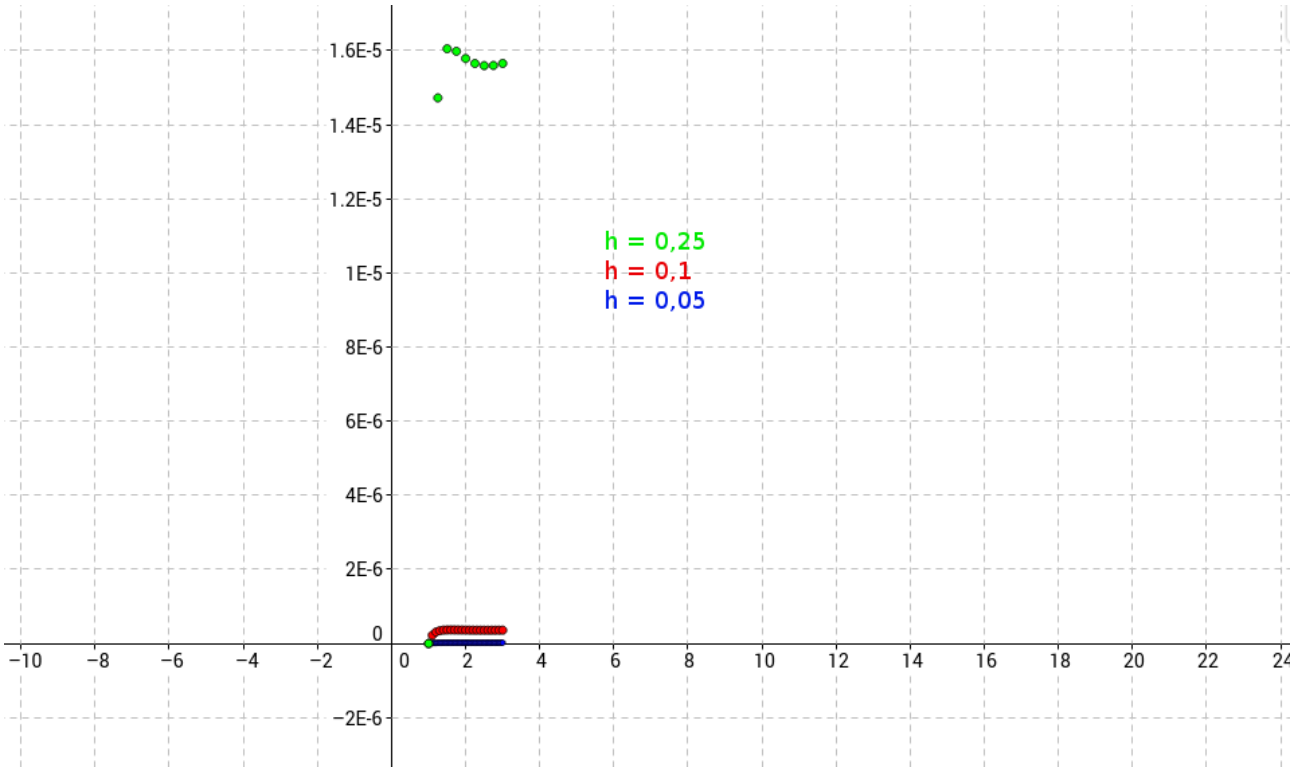
Implementação de Runge-Kutta 4ª ordem:

```
import numpy as np
from math import *

def fxy(x, y):
        return (y/x) - ((y/x)**2)

h = 0.25
a = 1.0
b = 3.0
n = int(((b-a)/h)+1)

x = list(np.linspace(a, b, n))
y = list(x[:])
yx = list(x[:])
k1 = x[:]
k2 = x[:]
k3 = x[:]
k4 = x[:]

for i in range(0, n-1):
        k1 = h*fxy(x[i],y[i])
        k2 = h*fxy((x[i]+h/2), y[i]+k1/2)
        k3 = h*fxy((x[i]+h/2), y[i]+k2/2)
        k4 = h*fxy(x[i]+h,y[i]+k3)

        y[i+1] = y[i]+((1.0/6.0)*(k1+ 2*k2 + 2*k3 + k4))

for i in xrange(n):
        yx[i] = x[i]/(1+log(x[i]))

for i in xrange(n):
        print y[i]
```

3 b)



3 c)


1.6E-5
1.4E-5
1.2E-5
1E-5
8E-6
6E-6
4E-6
2E-6
0
−2E-6
−10
−8
−6
−4
−2
0
2
4
6
8
10
12
14
16
18
20
22
h = 0,25
h = 0,1
h = 0,05

4)
Discretização:

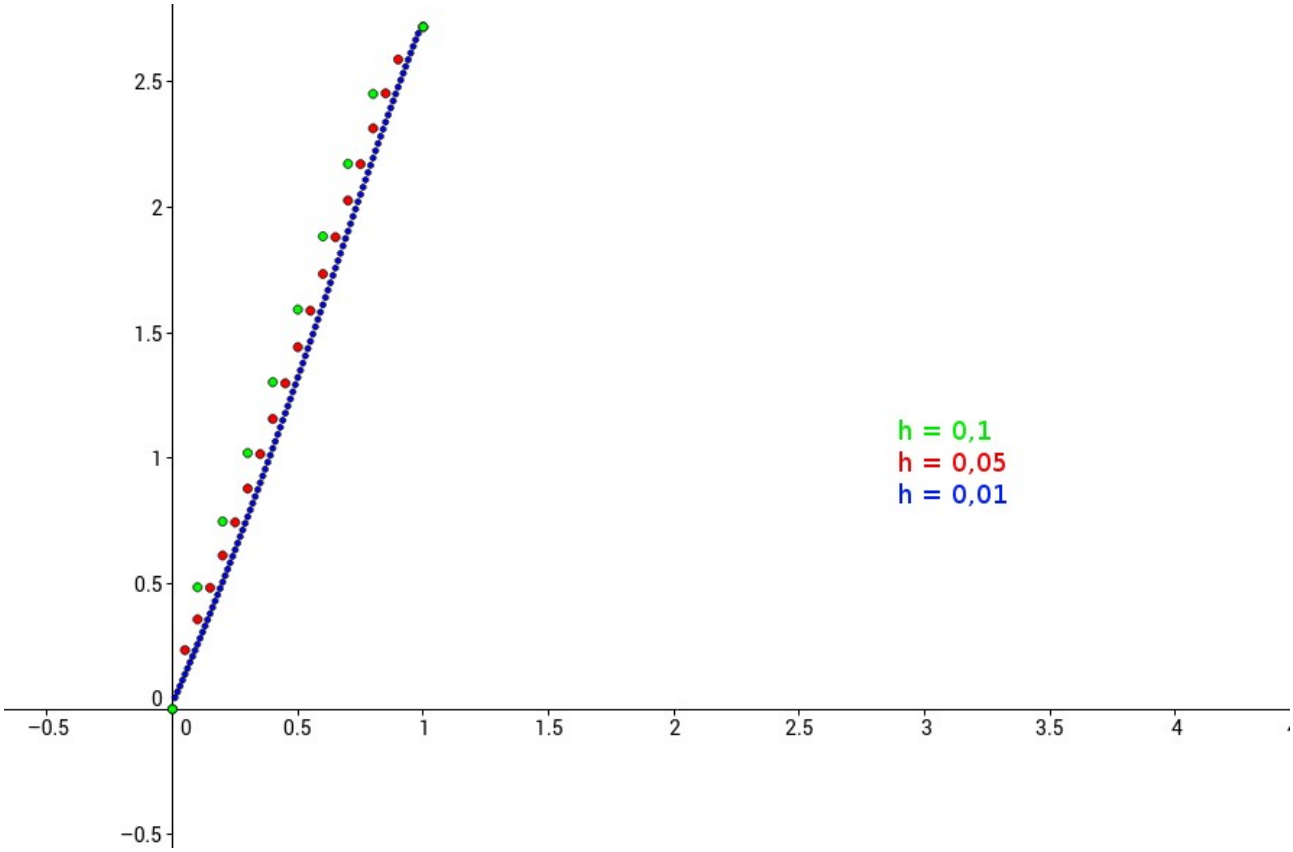
$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2}=\frac{y_{i+1}-y_{i-1}}{2h}-xy-e^x(x^2+1)$$
$$\frac{2h^2(y_{i+1}-2y_i+y_{i-1})}{h^2}=\frac{2h^2(y_{i+1}-y_{i-1})}{2h}-2h^2(yih)$$
$$2(y_{i+1}-2y+y_{i-1})=h(y_{i+1}-y_{i-1})-2ih^3y_i-2h^2e^{ih}(ih^2+1)$$
$$2y_{i+1}-4y_i+2y_{i-1}=hy_{i+1}-hy_{i-1}-2ih^3y_i-2h^2e^{ih}(ih^2+1)$$
$$2y_{i+1}-hy_{i+1}+2ih^3y_i-4y_i+2y_{i-1}+hy_{i-1}=2h^2e^{ih}(ih^2+1)$$
$$y_{i+1}(2-h)+y_i(2ih^3-4)+y_{i-1}(2+h)=2h^2e^{ih}(ih^2+1)$$

$$A=(2-h)$$
$$B=(2ih^3-4)$$
$$L=(2+h)$$

$$Ay_{i+1}+By_i+Ly_{i-1}=2h^2e^{ih}(ih^2+1)$$

Implementação:

```
import numpy as np

e = 2.71828182846
h = 0.1
a = 0.0
b = 1.0
n = int(((b-a)/h))
x = np.linspace(a, b, n+1)
linhas = n-1
colunas = n

matriz = np.zeros(shape=(n-1,n))

h2 = h**2
for i in range(0, n-1):
	ih = i*h
	beta = (2.0 *(i*(h**3))) - 4.0
	alfa = 2.0 - h
	lamp = 2.0 + h
	res = (-2.0 * h2 * e**ih)*((ih**2)+1.0)

	matriz[i-1][n-1] = res
	matriz[n-2][n-1] = res
	matriz[n-2][n-1] = matriz[n-2][n-1] - alfa*e
	for j in range(0, n-1):
		if i==j:
			matriz[i][j] = beta
		if (i+1)==j:
			matriz[i][j] = alfa
		if (i-1)==j:
			matriz[i][j] = lamp
# Gauss
def zera(l, ll, x, y):
	m = l[x]/ll[y]
	for x in range(0, len(l)):
		l[x] = l[x] - (m*ll[x])

for j in range(0, colunas-2):
	for i in range(j, linhas):
		if i != j:
			zera(matriz[i], matriz[j], j, j)

b = []
for i in range(linhas-1, -1, -1):
	b.append(matriz[i][colunas-1])

b[linhas-1] = matriz[linhas-1][colunas-1] / matriz[linhas-1][colunas-2]
soma = 0.0
for i in range(linhas-2, -1, -1):
	#print i
	for j in range(colunas-2, i, -1):
		#print j
		soma = soma + matriz[i][j]*b[j]

	soma = matriz[i][colunas-1] - soma
	b[i] = soma / matriz[i][i]
	soma = 0.0

print 'X', '\t', '    Y'
print x[0], '\t', 0.0
for i in range(1, n-1):
	print x[i], '\t', b[i]
print x[n], '\t', e
```



João Miguel Gehlen da Silva
Ciência da Computação
Calculo numérico