In [3]:

```
import numpy as np
import math as mt
```

Лабораторная работа №5

Численное решение системы линейных уравнений, численное интегрирование **Выполнила:** Клюшенкова Полина Анатольевна: 429

Цель работы: научиться считать определенные интеграллы и решать системы линейных уравнений численными методами и с помощью встроенных библиотек.

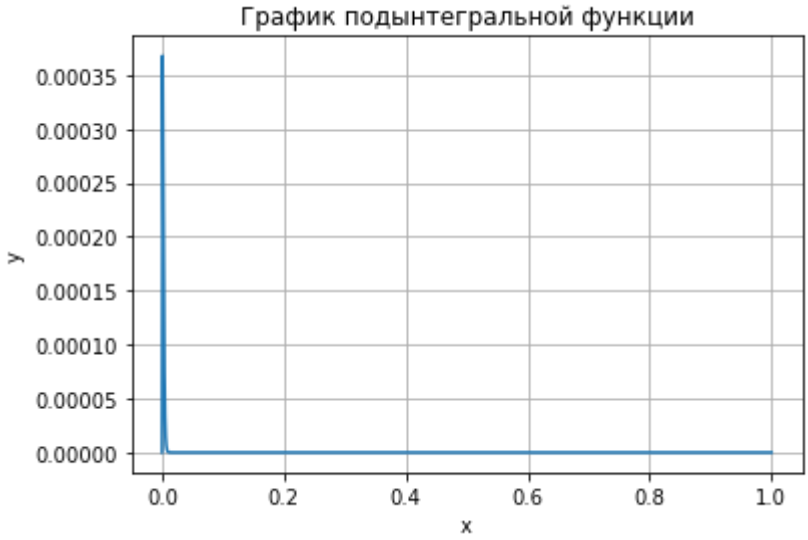
Вычисление интеграла : используем квадратурную формулу 3/8(n=3)

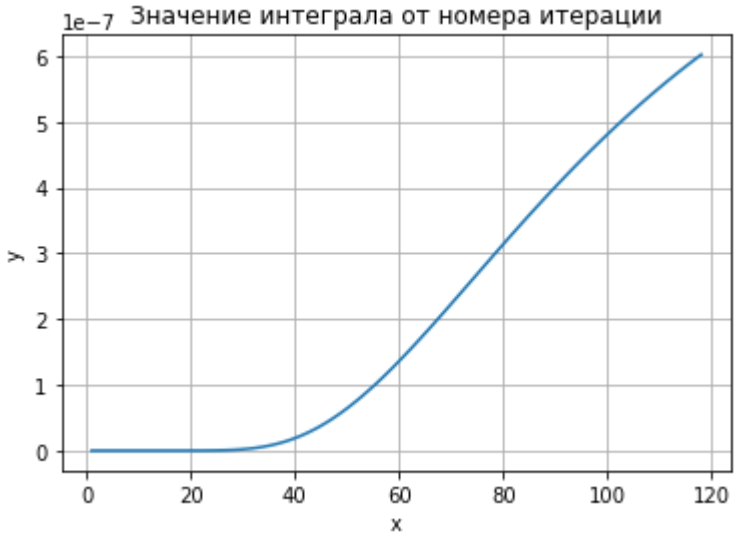
$$\sum_{k=0}^{3} C_k^{(3)} f(x_k^{(3)}) = \frac{(\beta-\alpha)}{8}(f(\alpha) + 3f(\frac{2\alpha+\beta}{3}) + 3f(\frac{\alpha+2\beta}{3} + f(\beta))$$

In [17]:

```
import scipy as sp
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
import math
import pylab
from matplotlib import mlab

def f(t,p):
    n=np.sin(t)*np.exp(- p *t)
    return n

def trv(a,b,p):
    fla=((b-a)/8)*(f(a, p) + 3*f((2*a+b)/3, p) + 3*f((a+2*b)/3, p) +f(b, p))
    return (fla)


def rims(i):
    mas=[]
    it1=trv(0,1,i)
    mas.append(it1)
    it2=trv(0,0.5,i)+trv(0.5,1,i)
    mas.append(it2)
    j=2
    while (abs(it1/it2-1)>0.01):
        j+=1
        it1=it2
        it2=0
        for k in range (j):
            it2+=trv(k/j,(k+1)/j,i)
        mas.append(it2)
    #print (j)
    return (mas)

    #return (it2)

mas=rims(1000)
# print(mas)
# print(len(mas))
dlina=len(mas)

int=[]
int.append(0)
i=1
while i<1000:
    mas=rims(i)
    int.append(mas[len(mas)-1])
    i+=1


tlist = mlab.frange (0, 1, 0.001)
ylist = [f (t,1000) for t in tlist]
plt.xlabel('x')
plt.ylabel('y')
plt.title('График подынтегральной функции')
pylab.plot (tlist, ylist)
plt.grid(True)
pylab.show()
```

```
xlist = mlab.frange (1,dlina, 1)
ylist = mas
#plt.axis([-10, 10, -10,10])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Значение интеграла от номера итерации')
pylab.plot (xlist, ylist)
plt.grid(True)
pylab.show()
```





Задание4и4+

Решить для n=1000 и n=10000 методом наискорейшего спуска систему:

$$n^2 x_i = \sum_{j=1}^{n} \frac{1}{1+i^2} * j * sin(j) * x_j + int_0^{\infty} \sin(t) exp(-i * t) dt, i = \overline{1,n}$$

Метод наискорейшего спуска

Итерационными называются приближенные методы, в которых решение системы получается как предел последовательности векторов

$$(x^k)_{k=1}^{\infty}$$

, каждый последующий элемент которой вычисляется по некоторому единому правилу. Начальный элемент

$$x^1$$

выбирается произвольно. Условие сходимости

$$\| \frac{x^{k+1} - x^k}{x^k} \| < \epsilon$$

Как правило, для итерационного метода решения системы существует такая последовательность невырожденных матиц

$$H_k$$

, что правило мостроения элементов итерационной последовательности записывается в виде

$$x^{k+1} = x^k - H_k(Ax^k - b)$$

или

$$x^{k+1} = T_k x^k + H_k b,$$

где

$$T_k = E - H_k E,$$

E - единичная матрица nxn

В методе наискорейшего спуска

$$H_k = \tau E$$
$$T_k = E - \tau A$$
$$\tau = \frac{(r^k, r^k)}{(Ar^k, r^k)}$$

Тогда итерационный процесс имеет вид

$$x^{k+1} = (E - \tau_k A)x^k + \tau_k Eb$$

In [8]:

```
n =1000
M=np.zeros((n,n))
for i in range(n):
    for j in range(n):
        M[i,j] = -j*mt.sin(j)/(1+i**2)
    M[i,i] +=n**2
np.set_printoptions(suppress=True, precision =4, linewidth=100)
# print(M)
```

Задание5и5+

Построить график решения X(i), а также график зависимости нормы решения от номера итерации предлагаемого метода численного решения системы.

Решить систему с помощью функции numpy.linalg.solve(…). Результат графически сравнить с решением, полученным в пункте 3. Разницу объяснит.

In [16]:

```
import copy
import matplotlib.pyplot as plt
rk=np.zeros(n)
tauk = np.zeros(n)
eps = 0.001
b = list()
for i in range(n):
    b.append(1)
xk = np.zeros(n)
for i in range(n):
    xk[i]= 1
xk1 = np.zeros(n)
xx = list()
for i in range(n):
    xx.append(i)
norma = 1
normax = list()
mx = list()
m = 0
while(norma>eps):
    N = 0
    for i in range(n):
        sum = 0
        for j in range(n):
            sum+=M[i,j]*xk[j]
        rk[i] = int[i] - sum
        N+=rk[i]**2
    norma = mt.sqrt(N)
    for i in range(n):
        sum =0
        for j in range(n):
            sum+=M[i,j]*rk[j]
        tauk[i] = rk[i]**2/(sum*rk[i])
    for i in range(n):
        xk1[i]=xk[i]+tauk[i]*rk[i]
    xk = copy.copy(xk1)
    xk1 = np.zeros(n)
    m+=1
    mx.append(m)
    sumx =0
    for i in range(n):
        sumx+=xk[i]**2
    normax.append(mt.sqrt(sumx))
plt.plot(xx, xk)
plt.title('График решения системы')
plt.grid()
plt.show()
plt.plot(mx, normax)
plt.title('График нормы решения в зависимости от номера итерации')
plt.grid()
plt.show()
```

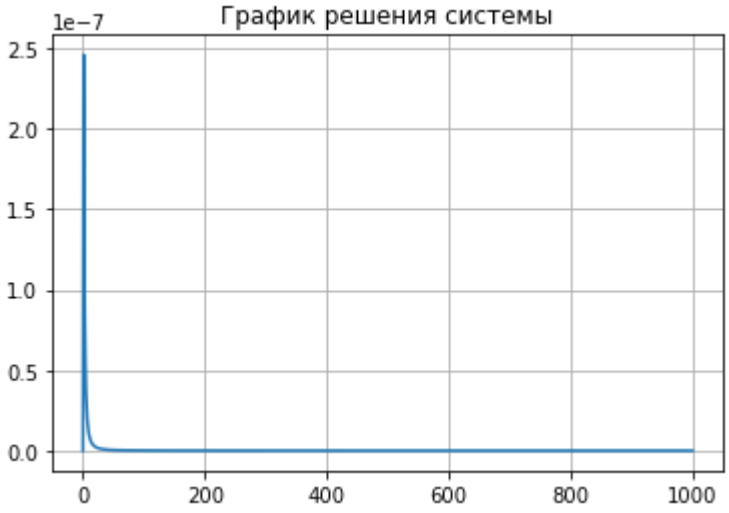


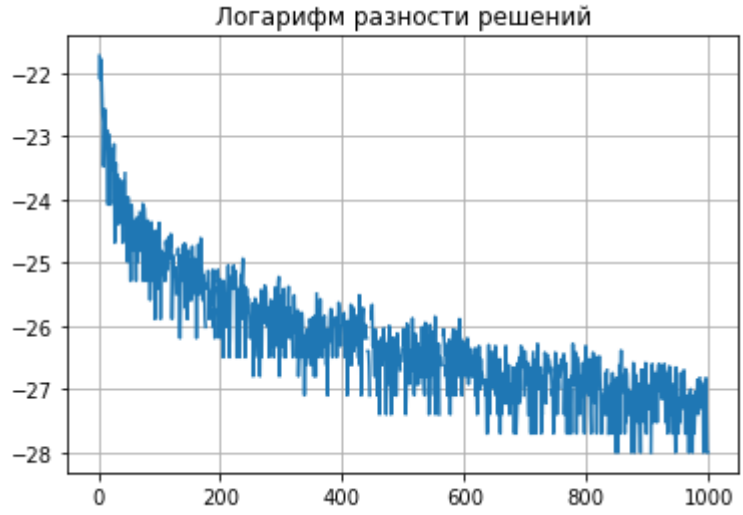
График решения системы
1e−7
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000


График нормы решения в зависимости от номера итерации
0.0000009
0.0000008
0.0000007
0.0000006
0.0000005
0.0000004
0.0000003
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

In [15]:

```
import numpy as np
import matplotlib.pyplot as plt
# b = np.zeros(n)
# for i in range(n):
#     b[i]=1
prov = np.linalg.solve(M, int)
# print(prov)
plt.plot(xx, np.log10(abs((xk-prov))))
plt.title('Логарифм разности решений')
# plt.plot(xx, prov)
plt.grid()
plt.show()
```

```
D:\Anaconda3\lib\site-packages\ipykernel\__main__.py:8: RuntimeWarning: divi
de by zero encountered in log10
```



Вывод: в ходе лабораторной работы был исследован метод 3/8 решения интеграла, а так же метод решения системы линейных уравнений, результат решения системы, псле проверки, оказался достаточно точным