Лабораторная работа №2 задание 2

Доскоч Роман вариант 5

$$\int_{0}^{22} \left(\frac{x}{2} - \cos(x)\right) dx = \frac{x^2}{4} - \sin(x) = 121.0088513092904$$

¶

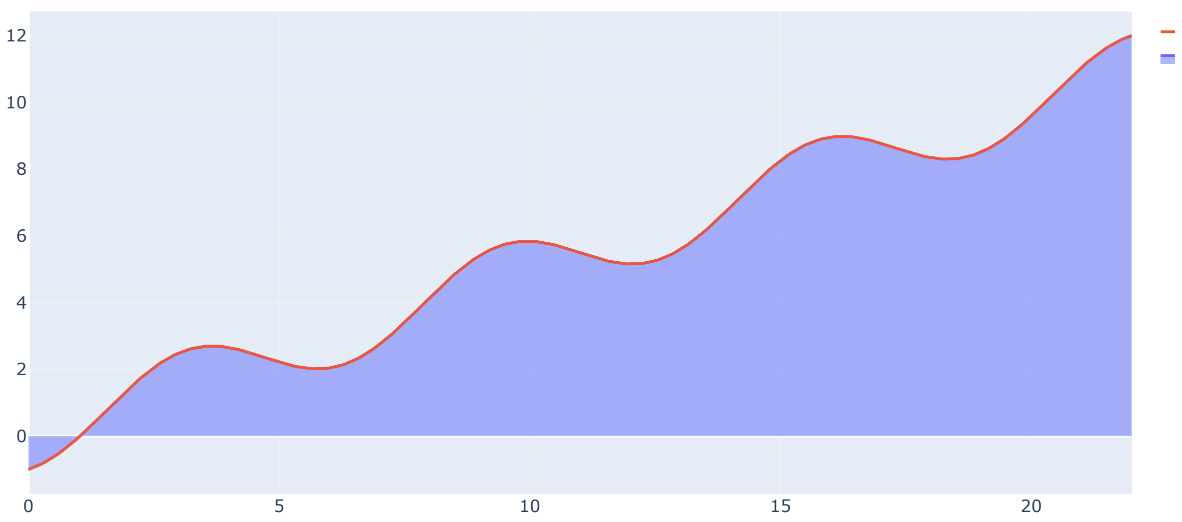
In [1]:
```python
import plotly as pl
import numpy as np
import scipy
import time
import plotly.graph_objs as go
from scipy import integrate
from numpy import sqrt,sin,log,pi,cos,exp
import pandas as pd

# время в миллисекундах
milli_time = lambda: time.time() * 1000.0
```

In [2]:
```python
f=lambda x:x/2-cos(x)
```

In [3]:
```python
a, b = 0, 22
F=lambda x:x**2/4-sin(x)
real_I=F(b)-F(a)
real_I
```

Out[3]: 121.0088513092904

In [4]:
```python
x=np.linspace(a, b, 500)
fig = go.Figure(go.Scatter(x=x, y=f(x), fill='tonexty'))
fig.add_trace(go.Scatter(x=x, y=f(x)))
```



1)Вычислить интеграл методом с пятью равноотстоящими узлами, а также методом Гаусса-3 с шагами равными $\frac{b-a}{1024^i}, i = \overline{0, 2}$

Гаусс-3

$$\int_{-1}^{1} f(t)dt \approx \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right)$$

$$\int_{a}^{b} f(t)dt = \frac{h}{2} \int_{-1}^{1} f\left(\frac{a+b}{2} + \frac{b-a}{2} t\right) dt$$

In [5]:
```
def Gauss3(h):
    start = milli_time()
    p=np.append(np.arange(a,b,h),b)
    A, B = p[:-1], p[1:]
    res=h/(2*9)*sum(5*f((A+B-h*sqrt(.6))/2) + 8*f((A+B)/2) + 5*f((A+B+h*sqrt(.6))/2))
    gaus3_time.append(milli_time()-start)
    return res
```

5 равнотсоящих узла

Для обычных квадартурных форм будем пологать A = 1

$$\int_{a}^{b} f(x)dx \approx \frac{h}{5} \sum_{i=0}^{n} f(x_0) + f(x_1) + f(x_2) + f(x_3) + f(x_4),$$

$$x_j = a + ih + \frac{jh}{4}, j = \overline{0,4}$$

In [6]:
```
def linspaceI(h):
    start = milli_time()

    # добавляю последнюю точку b чтобы закончить промежуток
    p = np.append(np.arange(a,b,h),b)
    X = np.linspace(p[:-1],p[1:],5)
    res = h/5*sum(f(X[0])+f(X[1])+f(X[2])+f(X[3])+f(X[4]))

    linspaceI_time.append(milli_time()-start)
    return res
```

Для интерполяционных квадратурных форм:

$$A_i = \int_{a}^{b} \prod_{i \neq j} \frac{x - x_j}{x_i - x_j} dx$$

$$\int_{a}^{b} f(x)dx \approx \sum_{i=0}^{n} A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3) + A_4 f(x_4),$$

$$x_j = a + ih + \frac{jh}{4}, j = \overline{0,4}$$

Вычисление коэффициентов 5 равностоящих узлов

In [7]:
```
X5=np.linspace(a,b,5)

def A_func(i, xi):
    return lambda x: np.prod([(x-xj)/(xi-xj) for j, xj in enumerate(X5) if i != j])

# интеграл вычислен с помощью библиотеки питона
A=[integrate.quad(A_func(i,xi),a,b)[0] for i, xi in enumerate(X5)]
A
```

Out[7]:
```
[1.7111111111111112,
 7.822222222222223,
 2.9333333333333322,
 7.822222222222223,
 1.7111111111111117]
```

In [8]:
```
def linspaceI_interpolate(h):
    start=milli_time()

    p=np.append(np.arange(a,b,h),b)
    X=np.linspace(p[:-1],p[1:],5)
    res = h/(b-a)*sum(A[0]*f(X[0]) + A[1]*f(X[1]) + A[2]*f(X[2]) + A[3]*f(X[3]) + A[4]*f(X[4]))

    linspaceI_interpolate_time.append(milli_time()-start)
    return res
```

In [9]:
```
gaus3_time, linspaceI_time, linspaceI_interpolate_time=[], [], []
H = [(b-a)/1024**i for i in range(3)]

g3h1, g3h2, g3h3 = [Gauss3(h) for h in H]
print(f'Гаусс3(h1) = {g3h1} погрешность={abs(real_I-g3h1)}')
print(f'Гаусс3(h2) = {g3h2} погрешность={abs(real_I-g3h2)}')
print(f'Гаусс3(h3) = {g3h3} погрешность={abs(real_I-g3h3)}\n')

lh1, lh2, lh3 = [linspaceI(h) for h in H]
print(f'Равностоящие узлы(h1) = {lh1} погрешность={abs(real_I-lh1)}')
print(f'Равностоящие узлы(h2) = {lh2} погрешность={abs(real_I-lh2)}')
print(f'Равностоящие узлы(h3) = {lh3} погрешность={abs(real_I-lh3)}\n')

lph1, lph2, lph3 = [linspaceI_interpolate(h) for h in H]
print(f'Интп. равные узлы(h1) = {lph1} погрешность={abs(real_I-lph1)}')
print(f'Интп. равные узлы(h2) = {lph2} погрешность={abs(real_I-lph2)}')
print(f'Интп. равные узлы(h3) = {lph3} погрешность={abs(real_I-lph3)}\n')

print(f'{real_I}')
```

```
Гаусс3(h1) = 120.99017167314916 погрешность=0.01867963614124335
Гаусс3(h2) = 121.00885130929046 погрешность=5.684341886080802e-14
Гаусс3(h3) = 121.00885130928721 погрешность=3.197442310920451e-12

Равностоящие узлы(h1) = 120.95275461111939 погрешность=0.05609669817101803
Равностоящие узлы(h2) = 121.00885122417385 погрешность=8.511655380516459e-08
Равностоящие узлы(h3) = 121.00885130928944 погрешность=9.663381206337363e-13

Интп. равные узлы(h1) = 120.93788433751992 погрешность=0.07096697177048839
Интп. равные узлы(h2) = 121.00885130929043 погрешность=2.842170943040401e-14
Интп. равные узлы(h3) = 121.00885130929159 погрешность=1.1795009413617663e-12

121.0088513092904
```

Выбор адаптивного шага

In [10]:
```
alpha=0.8
def Adaptive_Runge(F, h0, p, e, t):
    x,h,X = a,h0,b
    I = 0
    H = []

    start = milli_time()
    while x != X:
        I1 = F(x, x + 2*h)
        I2 = F(x, x + h) + F(x + h, x + 2*h)
        err = (I2 - I1) / (2**p - 1)
        delta=(e/abs(err))**(1/(p+1))
        h_new=alpha*delta*h
        if(delta < 1):
            h=h_new
            continue

        H.append(x)
        x+=2*h
        I+=I1
        h=min(h_new,(X-x)/2)

    t.append(milli_time()-start)
    return I, np.array(H)
```

Метод Гаусса

In [11]:
```
def Adapt_Gauss3(x1, x2):
    h=x2-x1
    return h/18*(5*f((x1+x2-h*sqrt(.6))/2) +
                 8*f((x1+x2)/2) +
                 5*f((x1+x2+h*sqrt(.6))/2))
```

Интерполяционная квадратурная форма на 5 узлах

In [12]:
```
def Adapt_linspaceI_interpolate(x1, x2):
    h=x2-x1
    X=np.linspace(x1,x2,5)
    return h/(b-a)*sum(A*f(X))
```

Квадратурная форма на 5 равностоящих узлах

In [13]:
```
def Adapt_linspaceI(x1, x2):
    h=x2-x1
    X=np.linspace(x1,x2,5)
    return h/5*sum(f(X))
```

Вычисление

In [14]:
```
adapt_gaus3_time, adapt_linspaceI_time, adapt_linspaceI_interpolate_time=[], [], []
e=1e-12
start_h=10
p=5
g3, g3_path = Adaptive_Runge(Adapt_Gauss3, start_h, p, e, adapt_gaus3_time)
print(f'Гаусс3 = {g3} погрешность = {abs(real_I-g3)} кол. шагов = {len(g3_path)}')

p=6
lp, lp_path = Adaptive_Runge(Adapt_linspaceI_interpolate, start_h, p, e, adapt_linspaceI_interpolate_time)
print(f'Интп. равные узлы = {lp} погрешность = {abs(real_I-lp)} кол. шагов = {len(lp_path)}')

p=4
l, l_path = Adaptive_Runge(Adapt_linspaceI, start_h, p, e, adapt_linspaceI_time)
print(f'Равные узлы = {l} погрешность = {abs(real_I-l)} кол. шагов = {len(l_path)}')
```

```
Гаусс3 = 121.00885130930577 погрешность = 1.5361933947133366e-11 кол. шагов = 99
Интп. равные узлы = 121.00885130929413 погрешность = 3.723243935382925e-12 кол. шагов = 92
Равные узлы = 121.00885130912867 погрешность = 1.617337375137 14e-10 кол. шагов = 26641
```

Взял большой начальный шаг из-за точности питона, при маленьком начальном шаге происходит деление на 0.

Диаграмма выбора шага Гаусс-3

In [15]:
```
fig = go.Figure(go.Scatter(x=x, y=f(x), name='f(x)'))
fig.add_scatter(mode='markers',x=g3_path,y=f(g3_path))
```

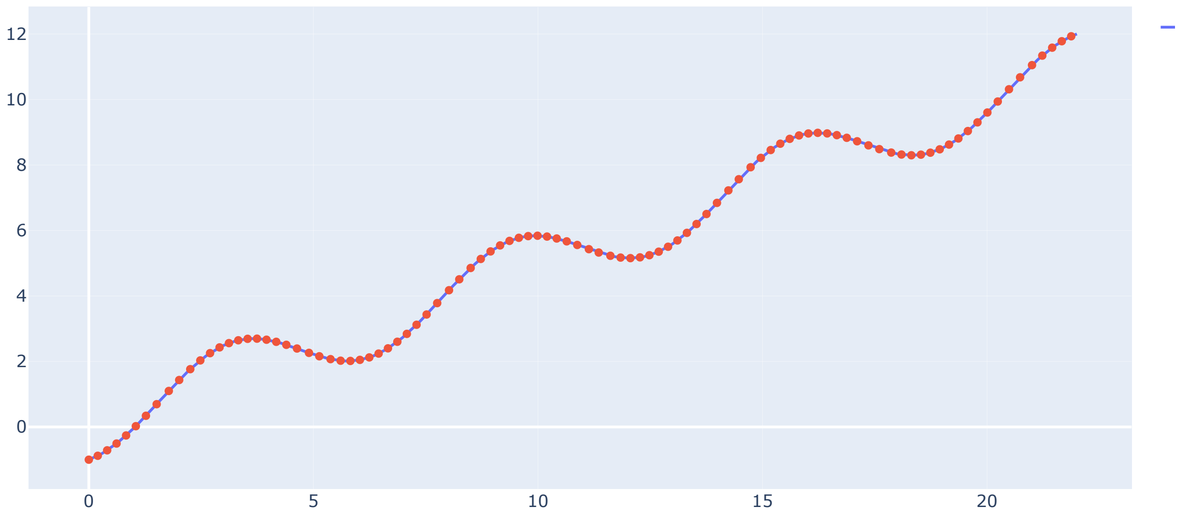


Диаграмма выбора шага 5 равностоящий интерполяционных узлов

In [16]:

```
fig = go.Figure(go.Scatter(x=x, y=f(x), name='f(x)'))
fig.add_scatter(mode='markers',x=lp_path,y=f(lp_path))
```

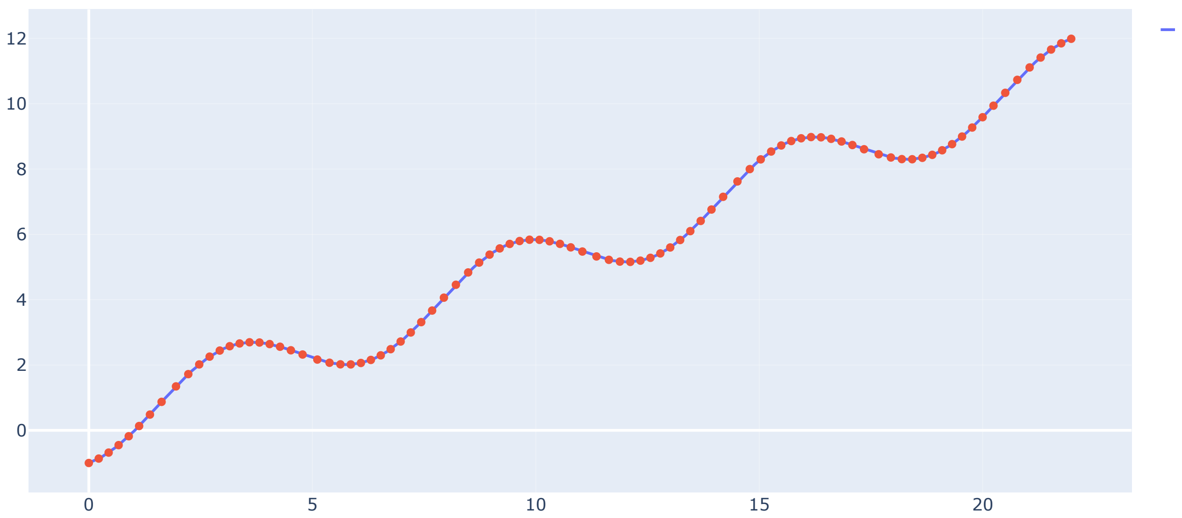
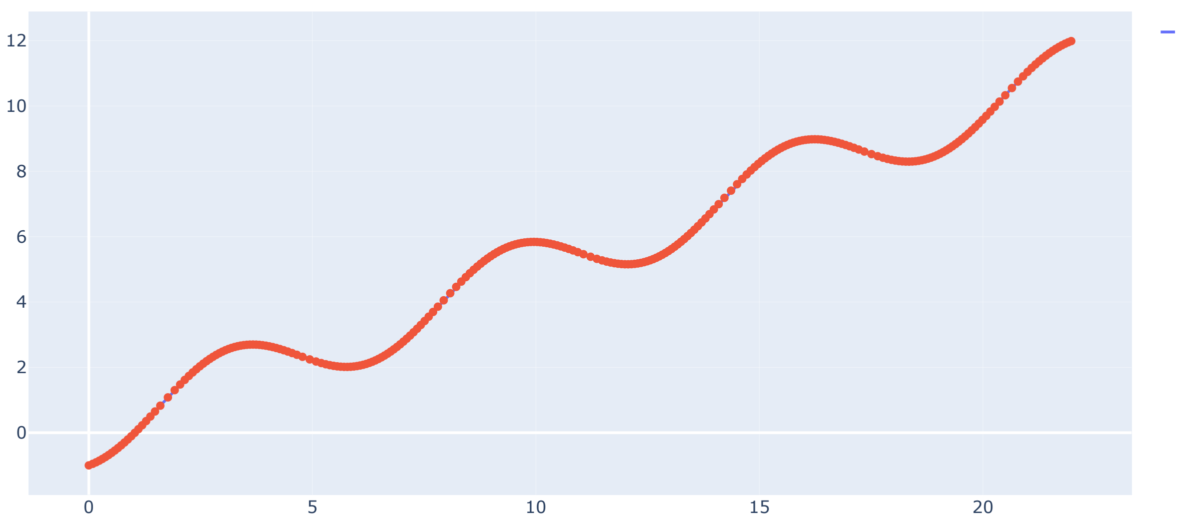


Диаграмма выбора шага 5 равностоящий интерполяционных узлов

In [17]:

```
_, l_path = Adaptive_Runge(Adapt_linspaceI, start_h, p, 1e-6, [])
fig = go.Figure(go.Scatter(x=x, y=f(x), name='f(x)'))
fig.add_scatter(mode='markers',x=l_path,y=f(l_path))
```



Для наглядности графика с равностоящими узлами использовал точность в 10^{-6}

Видно что послений граик неэффективный. Это обьясняется слишком большой погрешностью изза каторой при выборе шага идет постоянное его уменьшение и больше шансов зациклиться и топтаться на одном месте.

Можно заметить что на ровных промежутках графика алгоритм выбирал шаги длиннее, соответственно на изогнутых короче. Это обьясняется самим алгоритмом. Мы уменьшаем шаг если точность не было достигунта на текущем шаге на 2

Время (миллисекунды)

Время (миллисекунды)

In [18]:

```
pd.DataFrame([['Равностоящие узлы',*linspaceI_time],
              ['Интер. равностоящие узлы',*linspaceI_interpolate_time],
              ['Гаусс-3',*gaus3_time]]
                 , columns=['Колличество разбиений', '(b-a)/1024^0','(b-a)/1024^1','(b-a)/1024^2'])
```

Out[18]:

	Колличество разбиений	(b-a)/1024^0	(b-a)/1024^1	(b-a)/1024^2
0	Равностоящие узлы	0.0	0.0	268.619385
1	Интер. равностоящие узлы	0.0	0.0	308.293945
2	Гаусс-3	0.0	0.0	213.403564

In [19]:

```
pd.DataFrame([['Интер. равностоящие узлы с выбром шага',*adapt_linspaceI_interpolate_time],
              ['Гаусс-3 с выбром шага',*adapt_gaus3_time],
              ['Равностоящие узлы с выбром шага',*adapt_linspaceI_time]]
                 , columns=['Колличество разбиений', '1e-12'])
```

Out[19]:

	Колличество разбиений	1e-12
0	Интер. равностоящие узлы с выбром шага	22.218262
1	Гаусс-3 с выбром шага	0.000000
2	Равностоящие узлы с выбром шага	3580.722656

Вывод

Судя по вренени адаптивный шаг на практике показал себя лучше, тем более в нем можно настраивать нужную нам точнось
Так же можно заметить что колличество шагов потраченное с выбором шага значительно меньше (около 100 и 1024^2)
На практике доказано что у Гаусса-3(НАСТ) лучшая практическая применимость (по определению у него наивысшая АСТ)