python™

Основы SciPy

Технологии и языки программирования

Юдинцев В. В.

Кафедра теоретической механики

13 апреля 2019 г.

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Содержание

SciPy

- SciPy – это набор математических алгоритмов и вспомогательных функций, созданных на основе Numpy.
- SciPy в интерактивном режиме iPython предоставляет пользователю функции высокого уровня для вычислений и визуализации данных.
- С SciPy интерактивный сеанс iPython вычислительной средой для обработки данных и системного прототипирования, конкурирующими с такими системами, как MATLAB, Octave, SciLab.

Основные пакеты

- scipy.constants
  физические и математические константы
- scipy.linalg
  задачи линейной алгебры
- scipy.optimize
  поиск корней уравнений, оптимизация
- scipy.integrate
  численное интегрирование и решение обыкновенных дифференциальных уравнений
- scipy.interpolate
  интерполяция и сглаживание

Константы

Физические константы

```
import scipy.constants as CONST

>> CONST.g
9.80665

>> CONST.G
6.67408e-11

>> CONST.c
299792458.0

>> CONST.m_e
9.10938356e-31
```

https://docs.scipy.org/doc/scipy/reference/constants.html#module-scipy.constants

scipy.constants.physical_constants

Словарь значений констант с указанием размерностей величин и погрешностей

```
>> CONST.physical_constants['elementary charge']
(1.6021766208e-19, 'C', 9.8e-28)

>> CONST.physical_constants['molar gas constant']
(8.3144598, 'J mol^-1 K^-1', 4.8e-06)

>> CONST.physical_constants['electron volt']
(1.6021766208e-19, 'J', 9.8e-28)
```

https://docs.scipy.org/doc/scipy/reference/constants.html#module-scipy.constants

Множители

```
>> CONST.giga
1000000000.0

>> CONST.mega
1000000.0

>> CONST.milli
0.001

>> CONST.nano
1e-09

...
```

https://docs.scipy.org/doc/scipy/reference/constants.html#module-scipy.constants

Углы

Градус в радианах

```
>> CONST.degree
0.017453292519943295
```

Минута дуги в радианах

```
>> CONST.arcminute
0.0002908882086657216
```

Секунда дуги в радианах

```
>> CONST.arcsecond
4.84813681109536e-06
```

Пример функции перевода угла, заданного в градусах и минутах дуги в радианы:

```
def deg_min_to_radian(adeg, amin):
    return adeg*CONST.degree + amin*CONST.arcminute
```

Время

CONST.minute	секунд в минуте	60.0
CONST.hour	секунд в часе	3600.0
CONST.day	один день в секундах	86400.0
CONST.week	одна неделя в секундах	604800.0
CONST.year	один год в секундах	31536000.0

Длина

CONST.inch	дюйм в метрах	0.0254
CONST.mile	миля в метрах	1609.3439999999998
CONST.micron	микрон в метрах	1e-06
CONST.light_year	свет. год в метрах	9460730472580800.0
CONST.parsec	парсек в метрах	3.085677...e+16

Давление

CONST.atm	атмосфера в Па	101325.0
CONST.bar	Бар в Па	100000.0
CONST.mmHg	мм. рт. ст. в Па	133.3223684210526З
CONST.psi	фунт-сила на кв. д. в Па	6894.757293168361

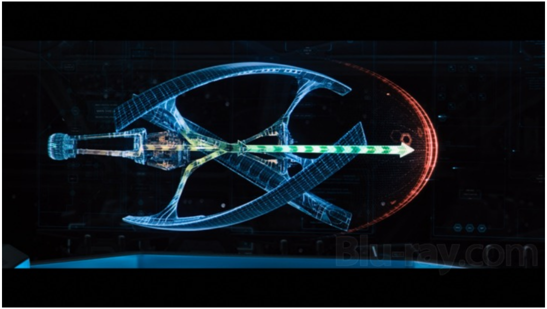
Скорость

CONST.kmh	км/ч в м/с	0.2777777777777778
CONST.mph	миль/ч в м/с	0.44703999999999994
CONST.mach	скорость звука в м/с	340.5
CONST.knot	кнот в м/с	0.5144444444444445

Сила, энергия и мощность

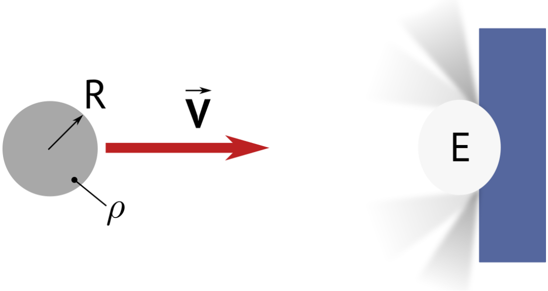
CONST.eV	эВ в Дж	1.6021766208e-19
CONST.calorie	калория в Дж	4.184
CONST.erg	эрг в Дж	1e-07
CONST.ton_TNT	тонна ТНТ в Дж	4184000000.0
CONST.horsepower	л.с. в ваттах	745.6998715822701
CONST.kgf	кгс в ньютонах	9.80665

Пример использования констант



Построить график кинетической энергии частицы, попадающей в защитный экран космического корабля в зависимости от скорости столкновения в долях скорости света. Энергию частицы выразить в тротиловом эквиваленте.

Кинетическая энергия



Кинетическая энергия частицы сферической формы радиуса R с плотностью материала ρ, движущейся со скоростью $V = kc, (k < 1)$:

$$E = \frac{mV^2}{2} = \frac{(4/3\pi R^3)\rho V^2}{2} = \frac{(4/3\pi R^3)\rho c^2}{2} k^2.$$

Пример использования констант

```python
import numpy as np
import scipy.constants as CONST
import matplotlib.pyplot as plt
# Радиус
R   = 0.0005
# Плотность
rho = 1000.0
# Масса
m   = rho*(4.0/3.0)*np.pi*R**3
# Массив долей скорости света: 9 точек от 0 до 0,1
k   = np.linspace(0, 0.1, 9)
# Энергия (массив)
Energy = 0.5*mass*(k*CONST.c)**2
# Энергия в тоннах ТНТ
E_ton_TNT = particle_energy/CONST.ton_TNT
# Энергия в килограммах ТНТ
E_kg_TNT  = 1000.0*E_ton_TNT
```

Построение графика

Массив k

```
array([0.,0.0125,0.025,0.0375,...,0.0875,0.1])
```

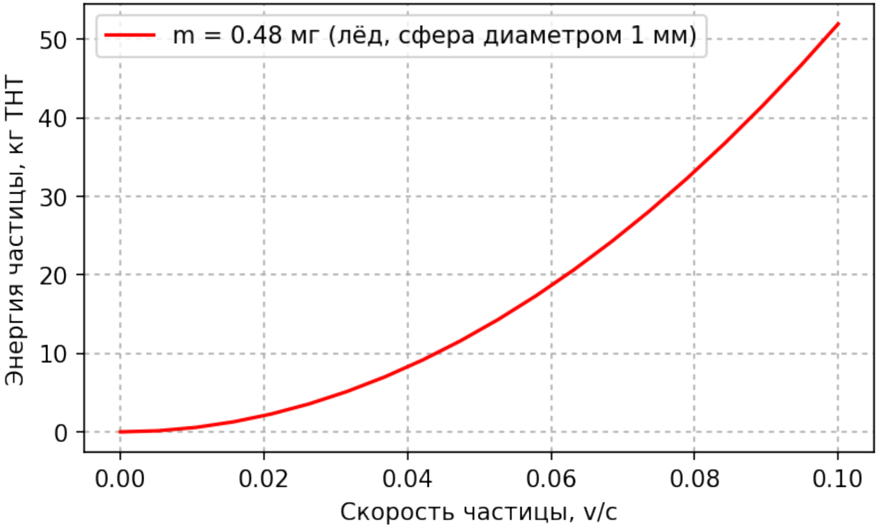
Массив E_kg_TNT

```
array([0.,0.8056,3.224,7.257,...,39.698,51.942])
```

Построение графика

```
plt.figure(figsize=(6, 3.5), dpi=150)
plt.plot(k, E_kg_TNT, 'r-')
plt.xlabel('Скорость частицы, v/c')
plt.ylabel('Энергия частицы, кг ТНТ')
plt.legend(['m = {:3.2f} мг'.format(m*1e6)])
plt.grid(linestyle=':')
```

Зависимость энергии от скорости



Линейная алгебра

scipy.linalg

- Модуль scipy.linalg содержит все функции модуля numpy.linalg
- Функции модуля scipy.linalg могут работать быстрее numpy.linalg, т.к. они всегда используют высокоэффективную библиотеку линейной алгебры BLAS/LAPACK

Тип numpy.matrix

Операция умножения для типа np.mat выполняется по правилам матричной алгебры:

```
import numpy as np
A = np.matrix([[1,2], [3,4]])
B = np.matrix([[1,2], [3,4]])
print(A*B)
```

```
[[ 7 10]
 [15 22]]
```

Для типа np.array умножение выполняется поэлементно:

```
A = np.array([[1,2], [3,4]])
B = np.array([[1,2], [3,4]])
print(A*B)
```

```
[[ 1  4]
 [ 9 16]]
```

Произведение матрицы и столбца

Размерности должны быть “совместимы”:

$$\mathbf{A}_{n\times m} \cdot \mathbf{b}_{m\times 1} = \mathbf{R}_{n\times 1}$$

```
import numpy as np

A = np.matrix([[1,2], [3,4]])

b = np.matrix([1,2])

A*b
```

ValueError: shapes (2,2) and (1,2) not aligned: 2 (dim 1) != 1 (dim 0)

Произведение матрицы и столбца

```
A = np.matrix([[1,2], [3,4]])
b = np.matrix([1,2])
# Транспонирование матрицы-строки b перед умножением
A*b.T
```

или

```
A = np.matrix([[1,2], [3,4]])
b = np.matrix( [ [1], [2] ] )
A*b
```

```
[[ 5]
 [11]]
```

Обратная матрица

```
import numpy as np
from scipy import linalg

A  = np.matrix([[1,2],[3,4]])
Ai = linalg.inv(A)
print(iA)
```

```
[[-2. ,  1. ],
 [ 1.5, -0.5]]
```

```
print(A*Ai)
```

```
[[  1.00000000e+00   0.00000000e+00]
 [  8.88178420e-16   1.00000000e+00]]
```

Решение СЛУ

Система уравнений:

$$\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}$$

Решение с использованием обратной матрицы (медленно):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} -232 \\ 129 \\ 19 \end{bmatrix} = \begin{bmatrix} -9.28 \\ 5.16 \\ 0.76 \end{bmatrix}.$$

Решение СЛУ: обратная матрица

```python
import numpy as np
from scipy import linalg

A = np.matrix( [ [1,3,5], [2,5,1], [2,3,8] ] )
b = np.matrix( [ [10], [8], [3] ] )

x = linalg.inv(A).dot(b)

print(x)
```

```
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

Решение СЛУ. Функция solve

```python
import numpy as np
from scipy import linalg

A = np.matrix( [ [1,3,5], [2,5,1], [2,3,8] ] )
b = np.matrix( [ [10], [8], [3] ] )

x = linalg.solve(A,b)

print(x)
```

```
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

Вычисление определителя матрицы

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$$

$$\begin{aligned} |\mathbf{A}| &= 1\begin{vmatrix} 5 & 1 \\ 3 & 8 \end{vmatrix} - 3\begin{vmatrix} 2 & 1 \\ 2 & 8 \end{vmatrix} + 5\begin{vmatrix} 2 & 5 \\ 2 & 3 \end{vmatrix} \\ &= 1\,(5\cdot 8 - 3\cdot 1) - 3\,(2\cdot 8 - 2\cdot 1) + 5\,(2\cdot 3 - 2\cdot 5) = -25. \end{aligned}$$

```python
import numpy as np
from scipy import linalg
A    = np.matrix( [ [1,3,5], [2,5,1], [2,3,8] ] )
detA = linalg.det(A)
print(detA)
```

```
−25.000000000000004
```

Собственные числа и собственные векторы

Собственный вектор $\mathbf{v}$ – вектор, умножение матрицы $\mathbf{A}$ на который даёт коллинеарный вектор – тот же вектор, умноженный на некоторое число λ, называемое собственным числом матрицы $\mathbf{A}$:

$$\mathbf{A} \cdot \mathbf{v} = \lambda \mathbf{v}$$

Собственные числа и собственные векторы

$$\mathbf{A} \cdot \mathbf{v} = \lambda \mathbf{v}$$

```
A = np.array([[1, 2], [3, 4]])
lamb, v = linalg.eig(A)
print(lamb)
```

```
[-0.37228132+0.j   5.37228132+0.j]
```

```
print(v[:,0])
print(v[:,1])
```

```
[-0.82456484   0.56576746]
[-0.41597356  -0.90937671]
```

Интерполирование

Функция одной переменной

```
import numpy as np
from scipy.interpolate import interp1d

x = np.linspace(0, 10, num=11, endpoint=True)
```

[0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]

```
y = np.cos(-x**2/9.0)
```

[1.0 0.9938 0.9028 0.5403 ... 0.6684 0.6764 -0.9111 0.1152]

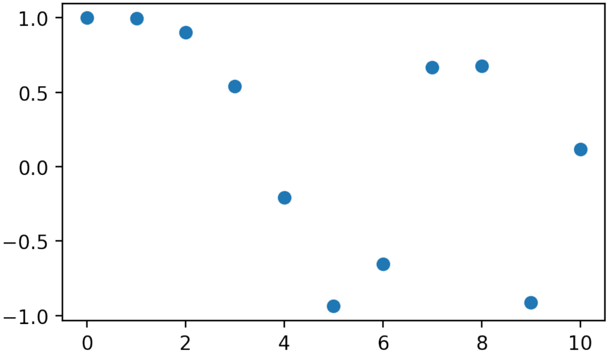
```
1
```

Табличная функция

```
0.0          1.000
1.0          0.994
2.0          0.903
3.0          0.540
4.0         −0.206
5.0         −0.935
6.0         −0.654
7.0          0.668
8.0          0.676
9.0         −0.911
10.          0.115
```

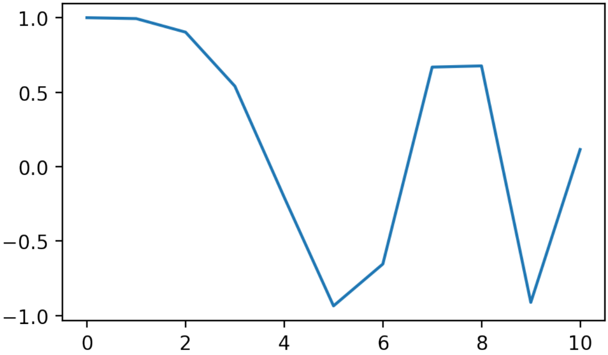
Табличная функция

```python
import matplotlib.pyplot as plt

plt.plot(x, y, 'o')
```



Табличная функция

```python
import matplotlib.pyplot as plt

plt.plot(x, y, '-')
```



Линейная интерполяция

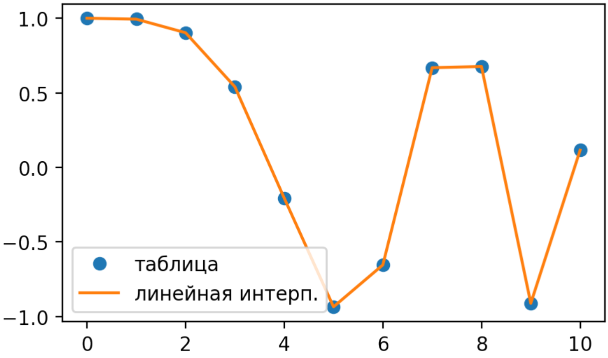
Известные точки соединяются прямыми линиями:

```
from scipy.interpolate import interp1d

f = interp1d(x, y)

print(f(2.5))
```

0.7215759876135193

Линейная интерполяция

```python
xnew = np.linspace(0, 10, num=41, endpoint=True)

plt.plot(x, y, 'o', xnew, f(xnew), '-')
plt.legend(['таблица', 'линейная интерп.'],
           loc='best')
```



Кубическая сплайн-интерполяция

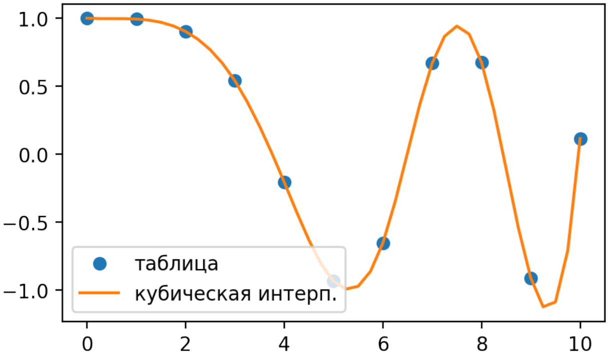
Точки гладко соединяются кубическими полиномами:

```
from scipy.interpolate import interp1d

f = interp1d(x, y, kind='cubic')

print(f(2.5))
```

0.7679872529415279

Кубическая сплайн-интерполяция

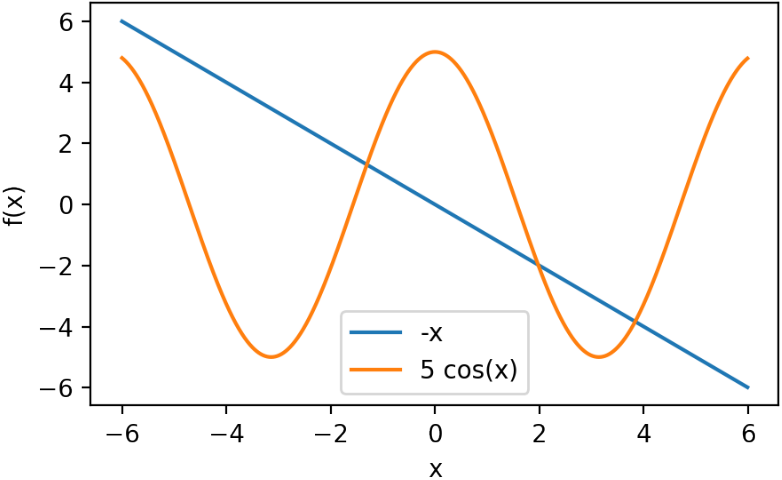
```
xnew = np.linspace(0, 10, num=41, endpoint=True)

plt.plot(x, y, 'o', xnew, f(xnew), '-')
plt.legend(['таблица',
            'кубическая интерп.'], loc='best')
```



Решение уравнений

Нахождение корней уравнения

$$\boxed{x + 5\cos(x) = 0} \quad \text{или} \quad 5\cos(x) = -x \tag{1}$$



Нахождение корней уравнения

$$\boxed{x + 5\cos(x) = 0} \quad \text{или} \quad 5\cos(x) = -x \qquad (2)$$

Поиск решения уравнения (2) с начальным приближением $x_0 = -1.0$:

```
import numpy as np
from scipy.optimize import root

def func(x):
    return x + 5 * np.cos(x)

sol = root(func, -1)

print(sol.x, sol.success)
```

[-1.30644001] True

Решение системы нелинейных уравнений

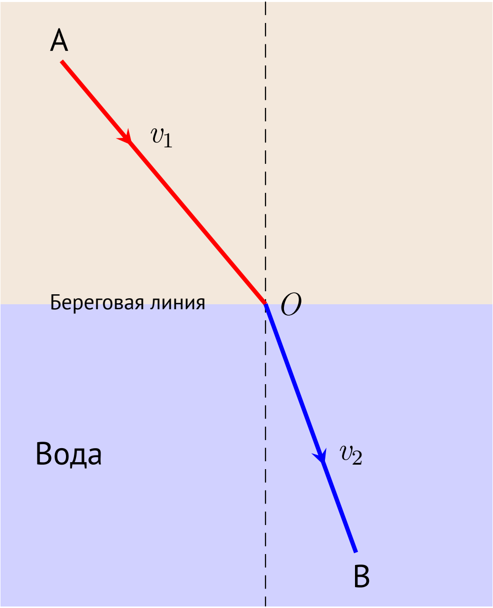
$$\begin{cases} x_0 \cos(x_1) = & 4, \\ x_0 x_1 - x_1 = & 5. \end{cases}$$

```
def func2(x):
    f = [ x[0] * np.cos(x[1]) - 4,
          x[0]*x[1] - x[1] - 5 ]
    return f

sol = root(func2, [1, 1])

print(sol.x, sol.success)
```

[6.50409711 0.90841421] True

Оптимизация

Поиск минимума функции



Скорость движения в воде меньше, чем в воздухе:

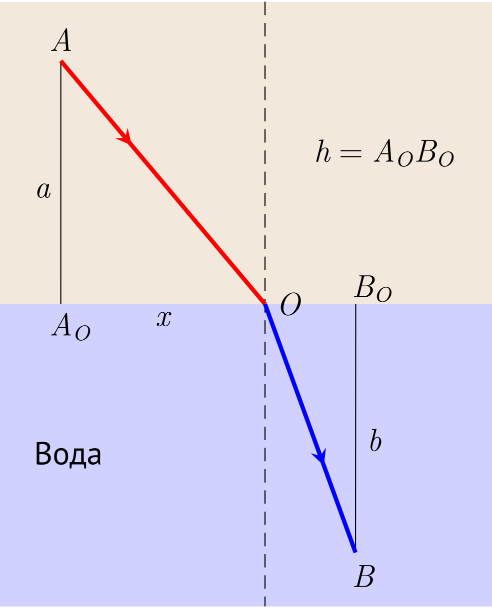
$$v_2 < v_1, \quad k = v_1/v_2$$

Найти оптимальный по времени путь из точки A в точку B

$$t_{AB} = AO/v_1 + OB/v_2$$

$$t_{AB} \to min$$

Поиск минимума функции



Суммарное время движения

$$t_{AB}(x) = \frac{\sqrt{x^2 + a^2}}{v_1} + \frac{\sqrt{(h-x)^2 + b^2}}{v_2} = \frac{\sqrt{x^2 + a^2}}{v_1} + k\frac{\sqrt{(h-x)^2 + b^2}}{v_1}$$

Поиск минимума функции

Необходимо найти значение x, при котором функция:

$$t_{AB}(x) = \frac{1}{v_1}\left(\sqrt{x^2 + a^2} + k\sqrt{(h - x)^2 + b^2}\right)$$

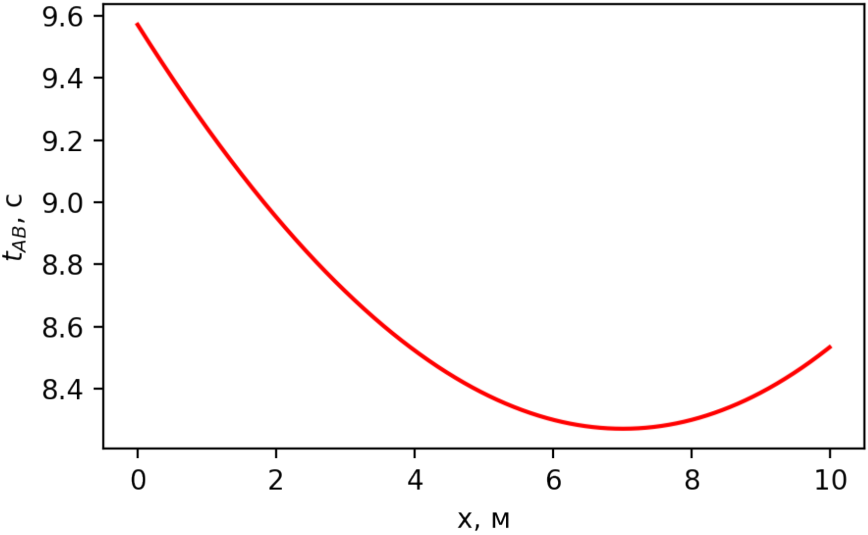
достигает минимума.

```python
import numpy as np
from scipy.optimize import minimize

def time_AB(x,   a, b, h, v, k):
    La = np.sqrt(x**2 + a**2)
    Lb = np.sqrt((h-x)**2 + b**2)
    res = La/v + k*Lb/v
    return res
```

График функции t_{AB}

```
import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.01)
# a=10, b=10, h=10, v=4, k=2
y = run_time(x,10,10,10,4,2)

plt.plot(x,, 'r-')

plt.xlabel('x, м')
plt.ylabel('$t_{AB}$, c')
```

График функции t_{AB}



Поиск минимума функции

Для поиска минимума модуль scipy.optimize содержит функцию minimize

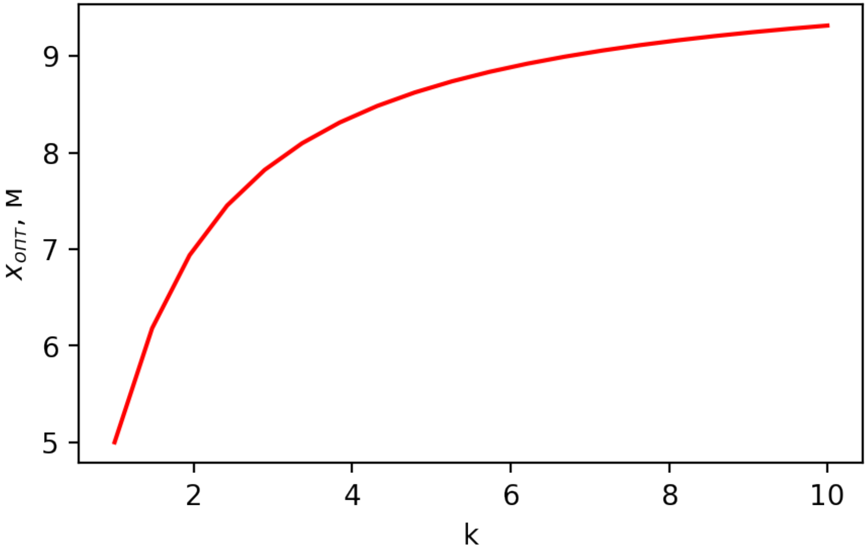
```
import numpy as np
from scipy.optimize import minimize

def time_AB(x,   a, b, h, v, k):
    La = np.sqrt(x**2 + a**2)
    Lb = np.sqrt((h-x)**2 + b**2)
    res = La/v + k*Lb/v
    return res

res = minimize(time_AB, 2.5,
               args = (10.0, 10.0, 10.0, 4.0, 2.0) )

print(res.x, res.fun, res.success)
```

[7.00534301] 8.27179134231555 True

Зависимость x от отношения скоростей k

```
def x_opt(ki):
    return minimize(run_time,2.5,
                    args=(10.0,10.0,10.0,4.0,ki)).x[0]

k = np.linspace(1,10,20)

xopt = np.fromiter( (x_opt(ki) for ki in k),
                    np.float,
                    count=len(k))

plt.figure(figsize=(5, 3), dpi=200)
plt.plot(k,xopt, 'r-')
plt.xlabel('k')
plt.ylabel('$x_{опт}$, м')
```

Зависимость x от отношения скоростей k



Интегрирование

Определённые интегралы

Модуль scipy.integrate

$$I = \int_0^1 x\sin(x)\,dx.$$

```
import scipy.integrate as integrate

def fun(x):
    return x*np.sin(x)

result = integrate.quad(fun, 0.0, 1.0)

print(result)
```

(0.30116867893975668, 3.3436440165485948e-15)

Первое значение в кортеже – значение интеграла, второе – верхняя оценка погрешности вычисленного значения интеграла

Определённые интегралы

Модуль scipy.integrate

$$I = \int_0^1 x \sin(x)\, dx.$$

С использованием лямбда-функции:

```
import scipy.integrate as integrate

result=integrate.quad(lambda x: x*np.sin(x),0.0,1.0)

print(result)
```

(0.30116867893975678, 3.3436440165485948e-15)

Параметры подинтегральной функции

$$I = \int_0^1 x^n \sin(x)\, dx.$$

```
import scipy.integrate as integrate

def integrand(x, n):
    return (x**n)*np.sin(x)

result = integrate.quad(integrand, 0.0, 1.0,
                        args = (3,))

print(result)
```

(0.17709857491700906, 1.9661891550116667e-15)

Двойной интеграл

$$I = \int_{y=0}^{1/2} \int_{x=0}^{1-2y} xy \, dx \, dy = \frac{1}{96}$$

```
import scipy.integrate as integrate

def integrand(x, y):
    return x*y
result = integrate.dblquad(integrand, 0.0, 0.5,
                           lambda x: 0,
                           lambda x: 1-2*x)

print(result)
```

(0.010416666666666668, 4.101620128472366e-16)

Интегрирование табличной функции

Для интегрирования можно использовать

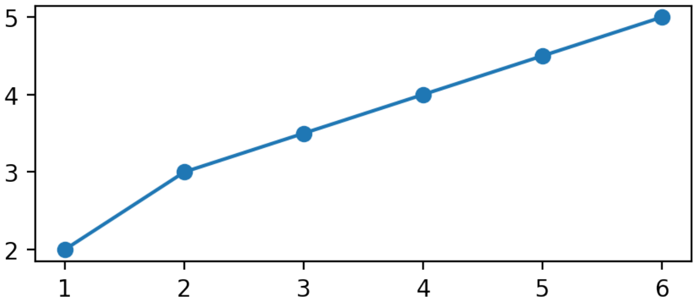
- метод трапеций;
  `scipy.integrate.trapz`
- метод Симпсона;
  `scipy.integrate.simps`
- ...

Функции интегрирования табличных функций определены в модуле `scipy.integrate`.

Интегрирование табличной функции

```
from scipy.integrate import simps

x = np.array([1., 2., 3.0, 4., 5.0, 6.])
y = np.array([2., 3., 3.5, 4., 4.5, 5.])
I1 = simps(y, x)
print(I1)
```

18.5416666667



Решение ОДУ

Обыкновенное ДУ

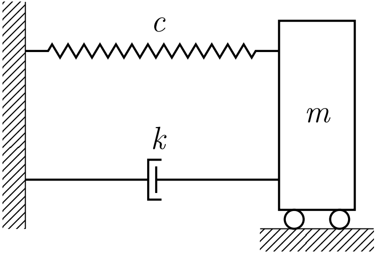
$$\frac{dy}{dx} = f(y, x), \quad y(x_0) = y_0$$

Уравнение маятника
($k > 0,\, c > 0$):

$$m\frac{d^2y}{dt^2} + k\frac{dy}{dt} + cy = 0$$

Линейное дифференциальное уравнения второго порядка.
Начальные условия:

$$y(0) = 1, \quad y'(0) = 0$$



Преобразование к системе ДУ 1-го порядка

Уравнение второго порядка можно привести к системе двух уравнений первого порядка введя новую переменную, обозначающую скорость

$$m\frac{d^2y}{dt^2} + k\frac{dy}{dt} + cy = 0, \quad k > 0, c > 0$$

Скорость маятника

$$\frac{dy}{dt} = v \tag{3}$$

Уравнение после замены второй производной

$$m\frac{dv}{dt} + kv + cy = 0 \tag{4}$$

Форма Коши

$$\begin{cases} \dfrac{dy}{dt} = v \\ \dfrac{dv}{dt} = -(kv + cy)/m \end{cases} \tag{5}$$

Функция правых частей, зависящая от независимой переменной t (время) и интегрируемых переменных y и v:

```python
def right_side(q, t):
    m = 1.0
    c = 10.0
    k = 0.5

    dqdt = [ q[1], -(k*q[1] + c*q[0])/m ]

    return dqdt
```

Получение численного решения (таблицы)

```
from scipy.integrate import odeint
```

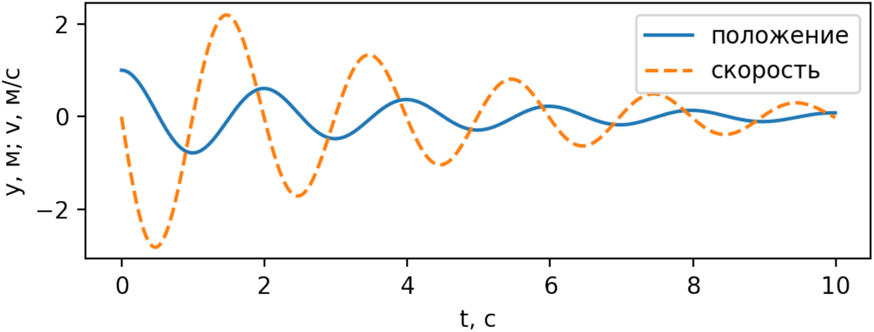
Получить решени (таблицу) на интервале от 0 до 10 секунд с шагом 0.01 с:

```
t  = np.arange(0, 10.0, 0.01)
y0 = [1.0, 0]
q  = odeint(right_side, y0, t)
```

Первая колонка матрицы q содержит значения y для соответствующих значений t, вторая колонка – значения скорости.

Построение графика решения

```
plt.plot(t, q[:,0], '-', t, q[:,1], '--')
plt.xlabel('t, c')
plt.ylabel('y, м; v, м/c')
plt.legend(['положение', 'скорость'], loc='best')
```



Задания

Задание 1

Построить таблицу, показывающую зависимость ускорения свободного падения от высоты над поверхности Земли, для высот от 0 до 1000 км с шагом 100 км. Ускорение свободного падения определяется при помощи формулы:

$$g = G\frac{M}{(R_e + h)^2}$$

где G – гравитационная постоянная, h – высота, $R_e = 6371$ км – радиус Земли, $M = 5.972 \cdot 10^{24}$ кг – масса Земли.

Задание 2

Найти решение системы линейных уравнений:

$$\begin{cases} 2x + 3y + z = 2 \\ 3x + y + 2z = 7 \\ x + 2y + 3z = 3 \end{cases}$$

Задание 3-5

Используя метод трапеций, найти значение интеграла:

$$F = \int_{-3}^{3} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Найти решения уравнения

$$x^3 + \cos x = 0$$

Найти минимум функции двух переменных:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Справка

1. SciPy tutorial

   https://docs.scipy.org/doc/scipy/reference/tutorial/index.html
2. Scipy Reference Guide

   https://docs.scipy.org/doc/scipy/reference/
3. Scipy Lecture Notes

   http://www.scipy-lectures.org/
4. Numpy User Guide

   https://docs.scipy.org/doc/numpy/user/
5. Введение в научный Python

   http://geometry.karazin.ua/resources/documents/20161211134615_988a1d6a.pdf