دوره جامع پایتون:

بخش محاسبات عددی

جلسه هجدهم

دکتر ذبیح اله ذبیحی

```
from scipy.interpolate import Rbf
import numpy as np
xs = np.array([0,1,2,3,4,5,6,7,8,9])
ys= np.array([1,2.84,5.90,10.14,16.24,25.04,36.72,50.65,65.98,82.41])

interp_func = Rbf(xs, ys)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

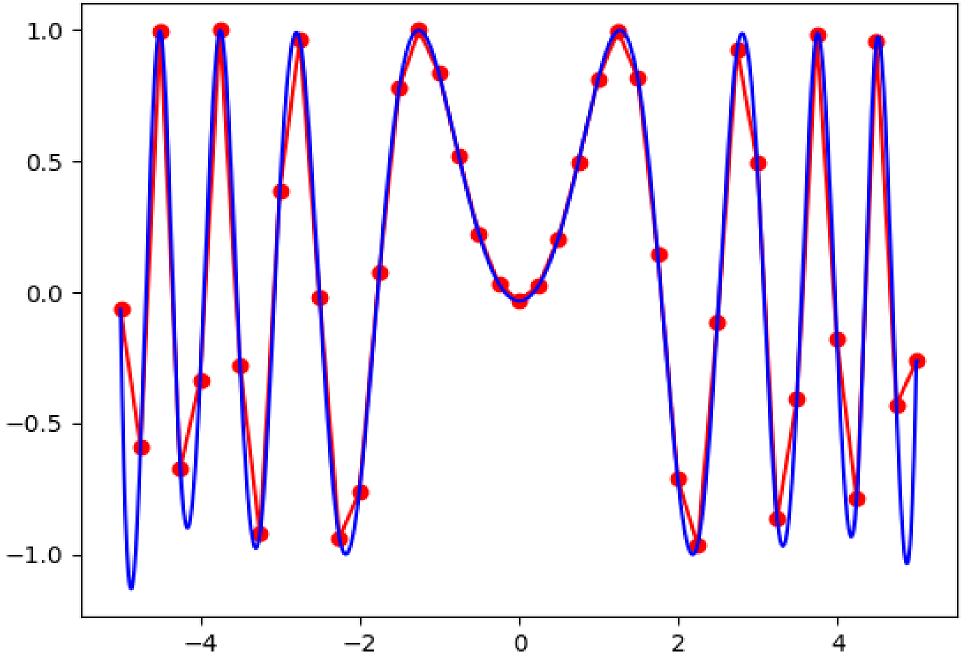
درون یابی دو متغیر

interp2d(x, y, z, kind='linear‘)

kind{‘linear’, ‘cubic’, ‘quintic’}, optional

The kind of spline interpolation to use. Default is ‘linear’.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.arange(-5.01, 5.01, 0.25
y = np.arange(-5.01, 5.01, 0.25)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**2+yy**2)
f = interpolate.interp2d(x, y, z, kind='cubic')
xnew = np.arange(-5.01, 5.01, 1e-2)
ynew = np.arange(-5.01, 5.01, 1e-2)
znew = f(xnew, ynew)
plt.plot(x, z[0, :], 'ro-', xnew, znew[0, :], 'b-')
plt.show()
```

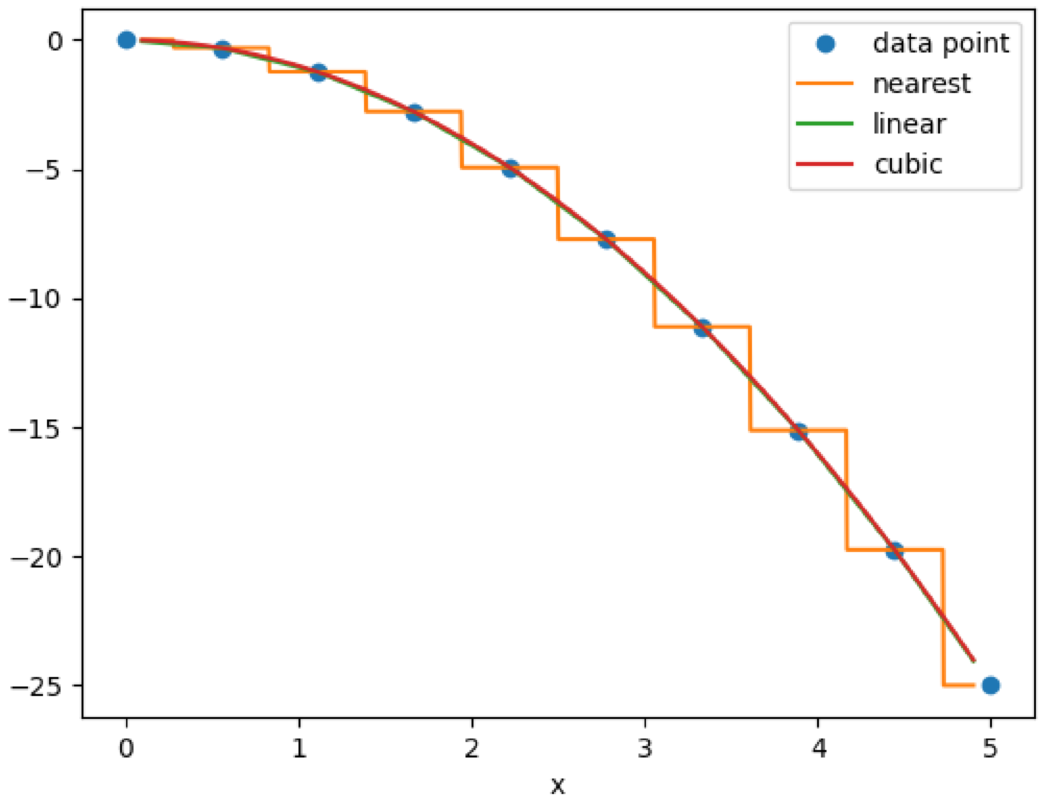


درون یابی n متغیره

- interpn(points, values, xi, method='linear‘)
- The method of interpolation to perform. Supported are “linear” and “nearest”, and “splinef2d”. “splinef2d” is only supported for 2-dimensional data.

```
import numpy as np
from scipy.interpolate import interpn
def value_func_3d(x, y, z):
    return 2 * x + 3 * y - z
x = np.linspace(0, 5)
y = np.linspace(0, 5)
z = np.linspace(0, 5)
points = (x, y, z)
values = value_func_3d(*np.meshgrid(*points))
point = np.array([2.21, 3.12, 1.15])
print(interpn(points, values, point))
```

```
import numpy as np
import scipy.interpolate
import matplotlib.pyplot as plt
def create_data(n):
    xmax = 5.
    x = np.linspace(0, xmax, n)
    y = - x**2
    return x, y
n = 10
x, y = create_data(n)
#use finer and regular mesh for plot
xfine = np.linspace(0.1, 4.9, n * 100)
#interpolate with piecewise constant function (p=0)
y0 = scipy.interpolate.interp1d(x, y, kind='nearest')
#interpolate with piecewise linear func (p=1)
y1 = scipy.interpolate.interp1d(x, y, kind='linear')
#interpolate with piecewise constant func (p=2)
y2 = scipy.interpolate.interp1d(x, y, kind='quadratic')
plt.plot(x, y, 'o', label='data point')
plt.plot(xfine, y0(xfine), label='nearest')
plt.plot(xfine, y1(xfine), label='linear')
plt.plot(xfine, y2(xfine), label='cubic')
plt.legend()
plt.xlabel('x')
plt.show()
```



فیت کردن تابع روی دیتا

scipy.optimize.curve_fit(f, xdata, ydata)

روش کمترین مربعات غیر خطی

popt: Optimal values for the parameters

pcov:The estimated covariance of popt

مثال

y= a*exp(b*x)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
x = np.linspace(0, 1, num = 40)
y = 3.45 * np.exp(1.334 * x) + np.random.normal(size = 40)
def f(x, a, b):
    return a*np.exp(b*x)
params, param_cov = optimize.curve_fit(f, x, y)
print(params)
plt.scatter(x, y, label='Data')
plt.plot(x, f(x, params[0], params[1]),label='Fitted function')
plt.legend()
plt.show()
```

انتگرال

آنتگرال یگانه با تابع quad()

quad(f,a,b)

$$I = \int_a^b f(x)dx = \int_0^1 x\,dx$$

```
def f(x):
    return x
I=integrate.quad(f,0,1)
print(I)
```

$$I = \int_{2}^{5} e^{x}\, dx$$

```
from scipy import integrate
import numpy as np
def f(x):
    return np.exp(x)
I=integrate.quad(f,2,5)
print(I)
```

$$I = \int_0^1 (ax^2 + bx)$$

```
from scipy.integrate import quad
def f(x, a, b):
    return a*x**2 + b
a = 2
b = 1
I = quad(f, 0, 1, args=(a,b))
print(I)
```

dblquad () انتگرال دو گانه

dblquad(f,a,b,c,d)

$$I = \int_{c}^{d} \int_{a}^{b} f(x, y)\, dx\, dy$$

$$I = \int_{0}^{1} \int_{2}^{3} xy\, dx\, dy = \int_{0}^{1} y\, dy \int_{2}^{3} x\, dx$$

```
from scipy import integrate
def f(x,y):
   return x*y
I=integrate.dblquad(f,2,3,0,1)
print(I)
```

انتگرال سه گانه tplquad()

tplquad(f,a,b,c,d,e,f)

$$I=\int_{e}^{f}\int_{c}^{d}\int_{a}^{b} f(x,y,z)\,dx\,dy\,dz$$

$$I=\int_{1}^{2}\int_{0}^{1}\int_{2}^{3} xyz\,dx\,dy\,dz=\int_{1}^{2} zdz\int_{0}^{1} y\,dy\int_{2}^{3} x\,dx$$

```
from scipy import integrate
def f(x,y,z):
    return x*y*z
I=integrate.tplquad(f,2,3,0,1,1,2)
print(I)
```

انتگرال گیری به روش رومبرگ

romberg(f,a,b)

```
from scipy import integrate
import numpy as np
def f(x):
    return x
I=integrate.romberg(f,0,1)
print(I)
```

انتگرال گیری به روش ذوزنقه

- trapezoid(y, x, dx=1)
- --------------------------------

```
from scipy import integrate
import numpy as np
x=np.array([2,4,6])
y=np.array([2,4,6])
I=integrate.trapz(y,x)
print(I)
```

```
from scipy import integrate
import numpy as np
y=np.array([2,4,6])
I=integrate.trapz(y)
print(I)
```

```
from scipy import integrate
import numpy as np
y=np.array([2,4,6])
I=integrate.trapz(y,dx=1)
print(I)
```

```
from scipy import integrate
import numpy as np

y=np.array([2,4,6])
I=integrate.trapz(y,dx=1)
print(I)
```

```
from scipy import integrate
import numpy as np
x = np.linspace(-2, 2, num=20)
y = x
y_int = integrate.cumulative_trapezoid(y, x, initial=0)
print(y_int)
```

انتگرال به روش سیمپسون

simpson(y, x,dx)

```
from scipy import integrate
import numpy as np
x = np.linspace(0, 2, num=10)
y = np.power(x,3)
I =integrate.simpson(y, x)
print(I)
```

انتگرال گیری به روش رومبرگ

romb(y, dx)

```
from scipy import integrate
import numpy as np
y = np.arange(3, 12)
 I =integrate.romb(y)
print(I)
```

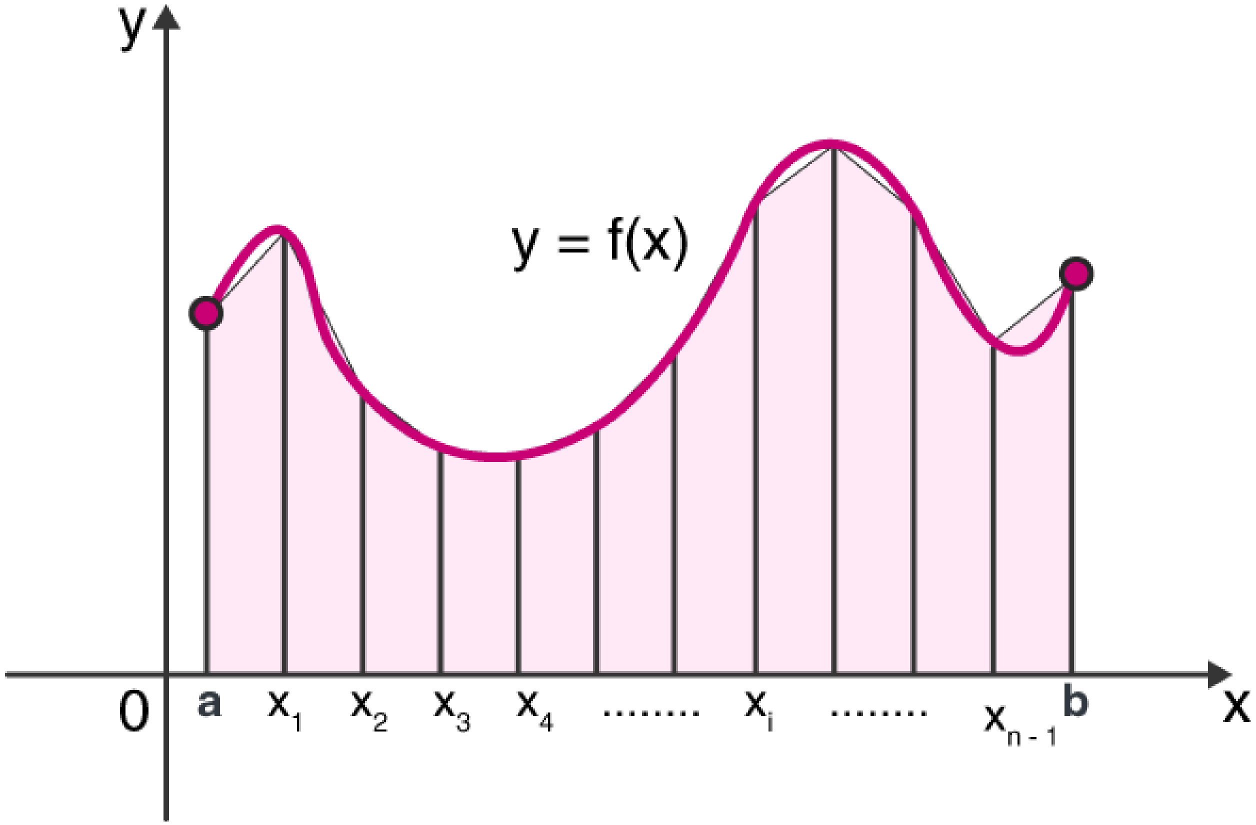
```
from scipy import integrate
import numpy as np
x = np.arange(10, 14.25, 0.25)
y = np.power(x,3)
I =integrate.romb(y)
print(I)
```

انتگرال گیری عددی

$$\int_a^b f(x)\,dx$$

$$I = \sum_{i=1}^{n} A_i f(x_i)$$

قاعده ذوزنقه



$$f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{h}(x - x_i)$$

$$S = \int_{x_1}^{x_n} f(x)dx = \sum_{i=1}^{n-1} \int_{x_i}^{x_i+h} \left[f_i + \frac{f_{i+1} - f_i}{h}(x - x_i) \right] dx$$

$$S = \sum_{i=1}^{n-1} \left[f_i h + \frac{(f_{i+1} - f_i)}{h} \frac{h^2}{2} \right]$$

$$S = \sum_{i=1}^{n-1} \frac{(f_i + f_{i+1})}{2} h$$

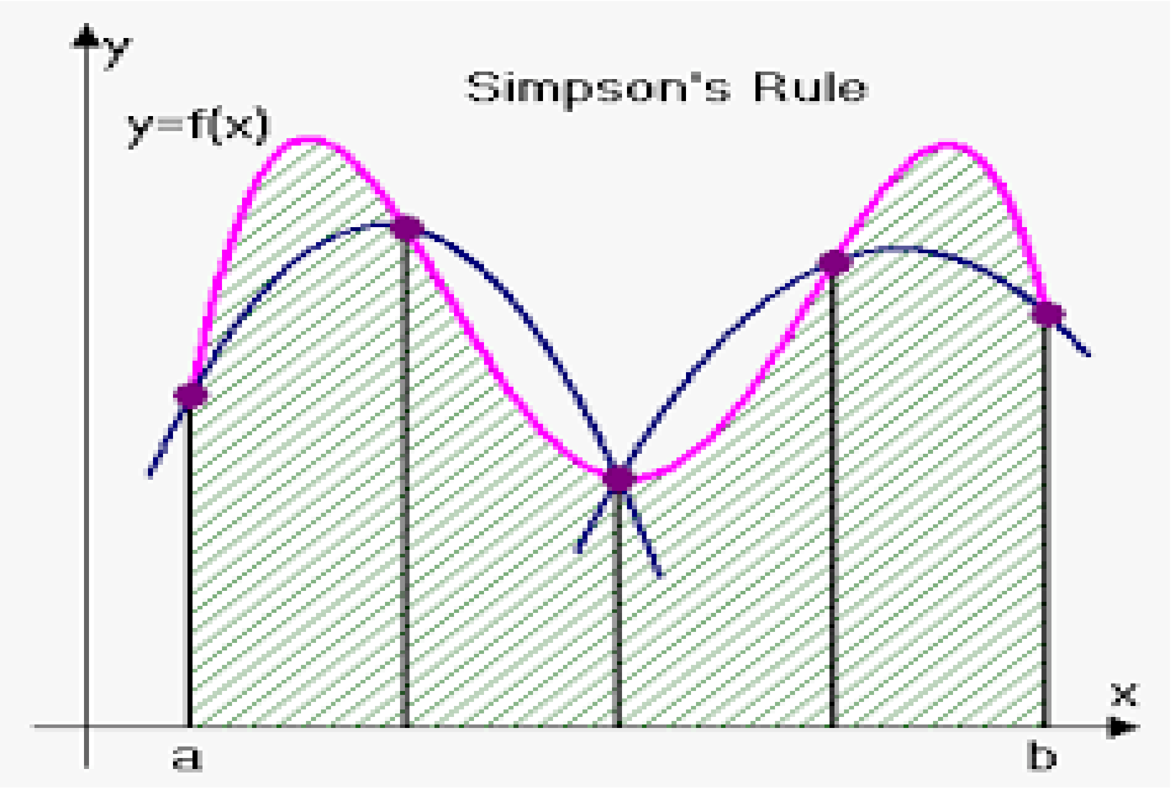
$$S = \left(\frac{f_1}{2} + f_2 + f_3 + \cdots + \frac{f_n}{2} \right) h$$

قاعده سیمپسون

$$f(x) = f(x_i) + \frac{\Delta f_i}{h}(x - x_i) + \frac{\Delta^2 f_i}{2h^2}(x - x_i)(x - x_i - h)$$

$$S = \int_{x_i}^{x_i+2h} \left[f(x_i) + \frac{\Delta f_i}{h}(x - x_i) + \frac{\Delta^2 f_i}{2h^2}(x - x_i)(x - x_i - h) \right] d$$

قاعده سیمپسون



$$S = \sum_{i=1,3,5,\ldots,n-2} \frac{h}{3}\left(f_i + 4f_{i+1} + f_{i+2}\right)$$

$$= \frac{h}{3}\{f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + 2f_5 + 4f_6 + \cdots + f_{n+1}\}$$

نکته: تعداد نقاط مورد نیاز برای قاعده سیمپسون فرد است.

خطای قاعده ذوزنقه

- از بسط تیلور f(x) در فاصله [0,h]

$$f(x) = f(0) + xf'(0) + \frac{x^2}{2!}f''(0) + \frac{x^3}{3!}f'''(0) + \cdots$$

$$\int_0^h f(x)dx = hf(0) + \frac{h^2}{2!}f'(0) + \frac{h^3}{3!}f''(0) + \frac{h^2}{4!}f'''(0) + \cdots \quad (*)$$

با استفاده از قاعده ذوزنقه، انتگرال برابر است با

$$\frac{h}{2}[f(0) + f(h)]$$

از بسط تیلور f(h) داریم:

$$\frac{h}{2}[f(0)+f(h)]$$
$$=\frac{h}{2}\left[f(0)+\left\{f(0)+hf'(0)+\frac{h^2}{2!}f''(0)+\frac{h^3}{3!}f'''(0)+\cdots\right\}\right] \quad (**)$$

از تفاضل (*) و (**) داریم

$$e=-\frac{h^3}{12}f''(0)-\frac{h^4}{24}f'''(0)$$

با نصف شدن h خطایک هشتم می شود

خطای روش سیمپسون

$$e = -\frac{h^5}{90} f^{iv}(0)$$

اگر h نصف شود خطا یک سی ودوم می شود.

دوره جامع پایتون:

بخش محاسبات عددی

جلسه ۵

دکتر ذبیح اله ذبیحی

حل معادله دیفرانسیل مرتبه اول

حل عددی معادلات دیفرانسیل از نوع:

$$\frac{\mathrm{dy}}{\mathrm{dx}} = \mathrm{f}(\mathrm{x}, \mathrm{y})$$

$$\mathrm{x}_1 \leq \mathrm{x} \leq \mathrm{x}_\mathrm{f}$$

مسئله مقدار اولیه

$$\mathrm{y}(\mathrm{x}_1) = y_0$$

مسئله مقدار مرزی

$$\mathrm{y}(\mathrm{x}_\mathrm{f}) = y_0$$

روش اویلر

$$\frac{\mathrm{dy}}{\mathrm{dx}} = \mathrm{f(x,y)} \quad , y(x_1) = y_1$$

$$y_{i+1} = y_i + hf(x_i, y_i)$$

خطای روش اویلر

$$y_{i+1} = y_{i+1}^* + e_{i+1} = (y_i^* + e_i) + hf(x_i, y_i^* + e_i)$$

$$y_{i+1}^* + e_{i+1} = y_i^* + e_i + \{f(x_i, y_i^*) + e_i \frac{\partial f}{\partial y} |_{x_i, y_i^*}$$

$$= y_i^* + hf(x_i, y_i^*) + e_i \left(1 + h\frac{\partial f}{\partial y}\right)$$

$$y_{i+1}^* = y_i^* + hf(x_i, y_i)$$

$$e_{i+1} = e_i \left(1 + h\frac{\partial f}{\partial y}\right)$$

اگر $\left|1 + h\frac{\partial f}{\partial y}\right| < 1$ باشد، آنگاه خطاها در تکرارهای پیاپی، مستهلک می شوند. در این حالت می گویند که روش اویلر پایدار است. در غیر این صورت خطاها در تکرارهای پیاپی افزایش یافته و روال مذکور ناپایدار است.

رانگ کوتا مرتبه دوم

$$y_{i+1} = y_i + \frac{h}{2}(s_i + s_{i+1})$$

$s_{i+1} = f(x_{i+1}, y_i + s_i h)$ و $s_i = f(x_i, y_i)$

رانگ کوتای مرتبه چهارم

$$y_{i+1} = y_i + \frac{h}{2}(s_1 + 2s_2 + 2s_3 + s_4)$$

$$s_1 = f(x_i\,, y_i)$$

$$s_2 = f\left(x_i + \frac{h}{2}\,, y_i + s_1\frac{h}{2}\right)$$

$$s_3 = f\left(x_i + \frac{h}{2}\,, y_i + s_2\frac{h}{2}\right)$$

$$s_4 = f(x_i + h\,, y_i + s_3 h)$$

حل معادلات دیفرانسیل همزمان

$$\frac{dy}{dx} = f(x, y, z) \text{ و } y_0 = Y_0$$

$$\frac{dz}{dx} = g(x, y, z) \text{ و } z_0 = Z_0$$

روش هیون

$$y_{i+1} = y_i + \frac{\mathrm{h}}{2}(s_i + s_{i+1})$$
$$s_i = f(x_i\,, y_i, z_i)$$
$$s_{i+1} = f(x_i + h\,, y_i + hs_i, z_i + hp_i)$$
$$p_i = g(x_i\,, y_i, z_i)$$
$$z_{i+1} = z_i + \frac{\mathrm{h}}{2}(p_i + p_{i+1})$$
$$p_{i+1} = g(x_i + h\,, y_i + hs_i, z_i + hp_i)$$

حل معادلات دیفرانسیل مراتب بالاتر

$$\frac{d^2y}{dx^2} = f\left(x, y, \frac{dy}{dx}\right)$$

$$\left(\frac{dy}{dx}\right)_{x_1} = y_1'$$

$$y(x_1) = y_1$$

$$\frac{dy}{dx} = z$$

$$\frac{dz}{dx} = f(x, y, z)$$

- $y_{i+1} = y_i + \frac{\mathrm{h}}{2}(s_i + s_{i+1})$
- $s_i = z_i$
- $p_i = f(x_i , y_i, z_i)$
- $s_{i+1} = z_i + hp_i$
- $p_{i+1} = f(x_i + h , y_i + hs_i, z_i + hp_i)$
- $z_{i+1} = z_i + \frac{\mathrm{h}}{2}(p_i + p_{i+1})$

روش تفاضل متناهی

- $y_i' = \frac{y_{i+1} - y_{i-1}}{2h}$
- $y_i' = \frac{y_{i+1} - y_i}{h}$
- $y_i' = \frac{y_i - y_{i-1}}{h}$
- $y_i^{"} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$

- $\nabla^2 u(x,y,z,t) = \frac{1}{c^2}\frac{\partial u(x,y,z,t)}{\partial t}$, $c^2 = \frac{K}{\sigma\rho}$

که در آن u دما ، K رسانایی حرارتی، σ گرمای ویژه و ρ چگالی است. اگر یک تیغه فلزی در نظر بگیریم، مسئله به دو بعد کاهش می یابد و در آن صورت توزیع حالت پایدار دما در این تیغه در معادله لاپلاس دو بعدی صدق می کند.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

- $x_i = ih_x \qquad i = 0,1,\ldots\ldots,n_x$
- $y_j = jh_y \qquad j = 0,1,\ldots\ldots,n_y$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} = 0$$

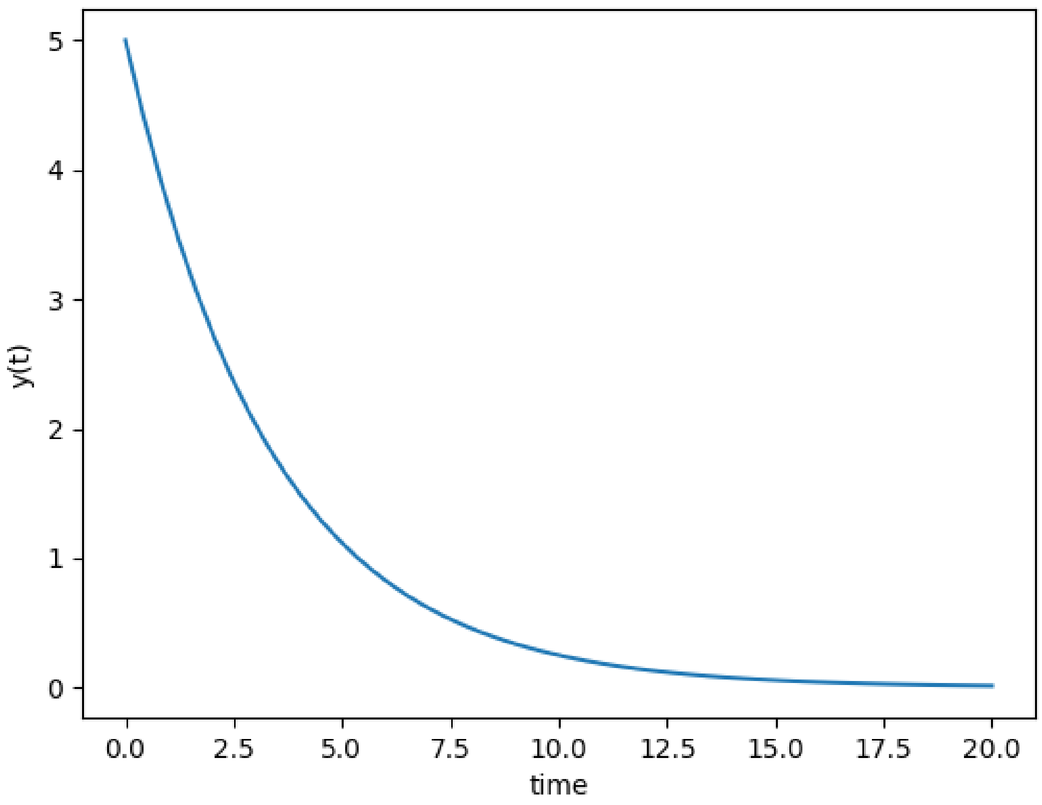
$$u_{i,j} = \frac{h_x^2 h_y^2}{2h_x^2 + 2h_y^2}\left[\frac{u_{i+1,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} + u_{i,j-1}}{h_y^2}\right]$$

اگر $h_x = h_y$ باشد

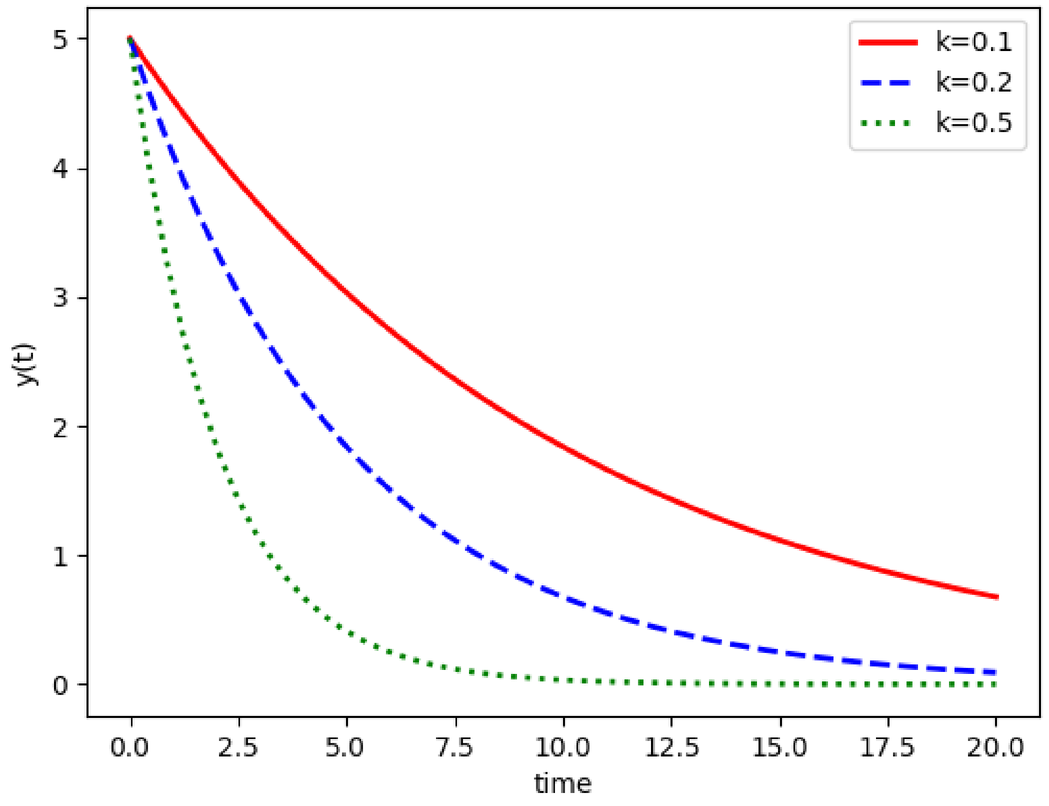
$$u_{i,j} = \frac{1}{4}\left[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}\right]$$

حل معادلات دیفرانسیل در پایتون

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def f(y,t):
   k = 0.3
   dydt = -k * y
   return dydt
y0 = 5
t = np.linspace(0,20)
y = odeint(f,y0,t)
plt.plot(t,y)
plt.xlabel('time')
plt.ylabel('y(t)')
plt.show()
```



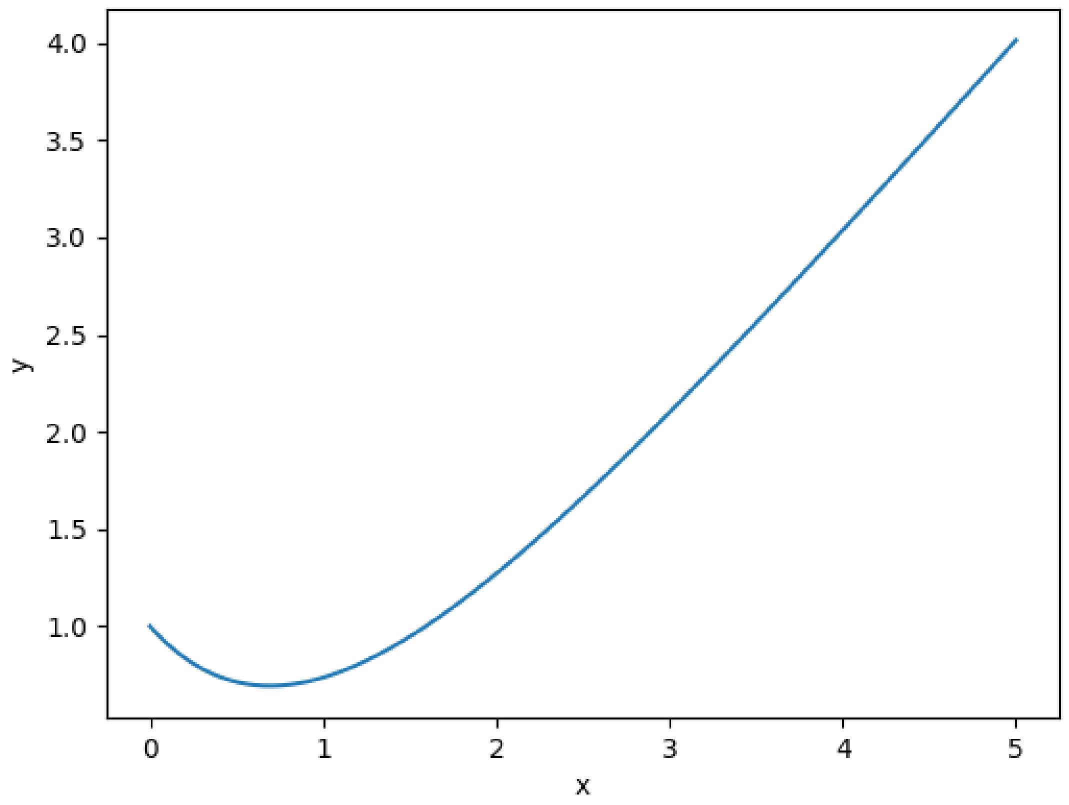
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def f(y,t,k):
    dydt = -k * y
    return dydt
y0 = 5
t = np.linspace(0,20)
k = 0.1
y1 = odeint(f,y0,t,args=(k,))
k = 0.2
y2 = odeint(fy0,t,args=(k,))
k = 0.5
y3 = odeint(f,y0,t,args=(k,))
plt.plot(t,y1,'r-',linewidth=2,label='k=0.1')
plt.plot(t,y2,'b--',linewidth=2,label='k=0.2')
plt.plot(t,y3,'g:',linewidth=2,label='k=0.5')
plt.xlabel('time')
plt.ylabel('y(t)')
plt.legend()
plt.show()
```



مثال

$$\frac{dy}{dx} + y = x \quad , y(0) = 1$$

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def f(y, x):
    return x - y
y0 = 1.0
xs = np.linspace(0,5,100)
ys = odeint(f, y0, xs)
plt.xlabel("x")
plt.ylabel("y")
plt.plot(xs, ys)
plt.show()
```



$$y'' + 2y' + 2y = \cos(2x) \quad , y(0) = 0 \, , y'(0) = 0$$

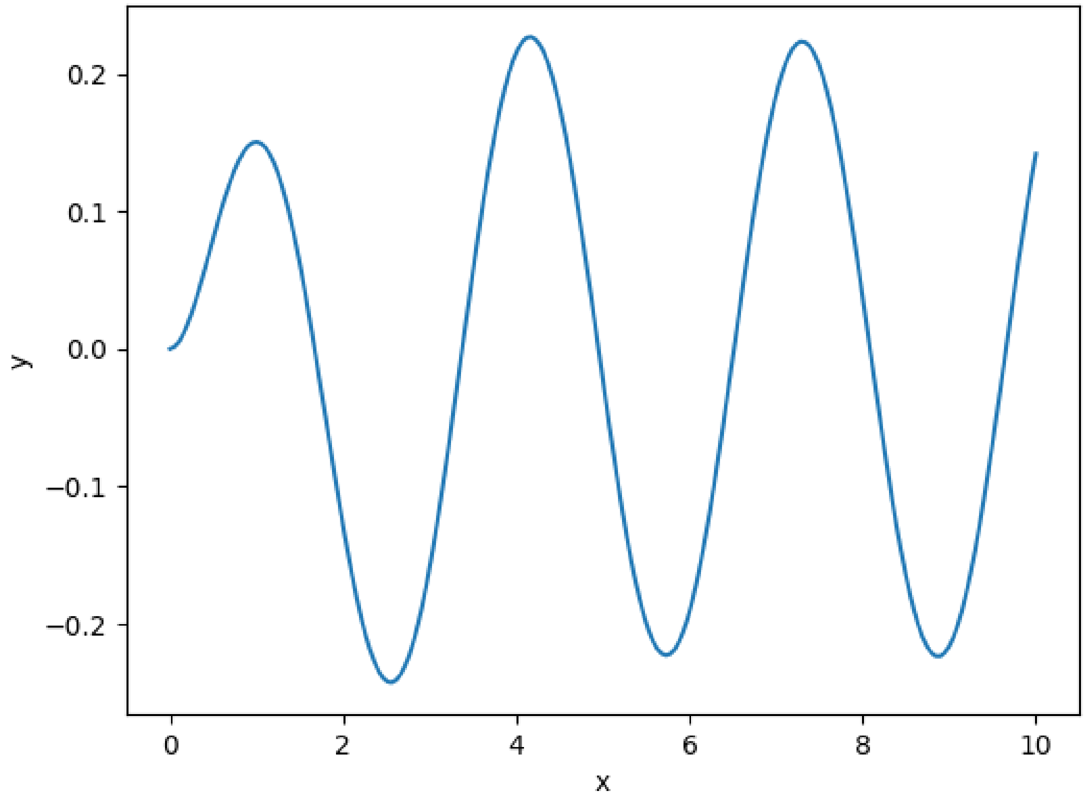
اگر

$$z = y'$$

بنابراین

$$z' + 2z + 2y = \cos(2x) \, , z(0) = 0, y(0) = 0$$

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def dU_dx(U, x):
    return [U[1], -2*U[1] - 2*U[0] + np.cos(2*x)]
U0 = [0, 0]
xs = np.linspace(0, 10, 200)
Us = odeint(dU_dx, U0, xs)
ys = Us[:,0]
plt.xlabel("x")
plt.ylabel("y")
plt.plot(xs,ys)
plt.show()
```



تابع دیگر برای حل معادلات دیفرانسیل

scipy.integrate.solve_ivp(fun, t_span, y0, method='RK45‘)

Method:

‘RK45’ (default): Explicit Runge-Kutta method of order 5(4)

‘RK23’: Explicit Runge-Kutta method of order 3(2)

‘DOP853’: Explicit Runge-Kutta method of order 8

‘Radau’: Implicit Runge-Kutta method of the Radau IIA family of order 5

‘BDF’: Implicit multi-step variable-order (1 to 5) method based on a backward differentiation formula

‘LSODA’: Adams/BDF method with automatic stiffness detection and switching

کمینه سازی

scipy.optimize.minimize(fun, x0, method=None)

Type of solver. Should be one of

‘Nelder-Mead’

‘Powell’

‘CG’ ‘:Conjugate-Gradient

BFGS’: Broyden-Fletcher-Goldfarb-Shanno algorithm

‘Newton-CG’ :Newton-Conjugate-Gradient

‘L-BFGS-B’

‘TNC’

‘COBYLA’

‘SLSQP’ :Sequential Least SQuares Programming

‘trust-constr’:Trust-Region Constrained Algorithm

‘dogleg’

‘trust-ncg’

‘trust-exact’

‘trust-krylov’

مثال

```python
from scipy.optimize import minimize
def f(x):
  return x**2 + x + 2
mymin = minimize(f, 0, method='BFGS')
print(mymin)
```

پاسخ

```
 fun: 1.75
 hess_inv: array([[0.50000001]])
      jac: array([0.])
  message: 'Optimization terminated successfully.'
     nfev: 8
      nit: 2
     njev: 4
   status: 0
  success: True
        x: array([-0.50000001])
```

fun مقدار بهینه تابع هدف مسأله است.
slack مقادیر متغیرهای کمکی یا لنگی مربوط به محدودیت‌های کوچکترمساوی را نمایش می‌دهد.
con مقادیر متغیرهای مصنوعی یا باقی‌مانده محدودیت‌های تساوی را نشان می‌دهد.
success زمانی که نقطه بهینه یافته شود، مقدار True را برمی‌گرداند.
int نمایانگر وضعیت خروجی الگوریتم است و اعداد صحیح ۰ تا ۴ را نمایش وضعیت‌های زیر نشان می‌دهد:
مقدار ۰ برای پایان موفقیت‌آمیز بهینه‌سازی
مقدار ۱ برای توقف الگوریتم به دلیل رسیدن به حداکثر تعداد تکرار
مقدار ۲ برای شرایط پاسخ نشدنی (منطقه موجه نشدنی)
مقدار ۳ برای شرایط پاسخ نامحدود
مقدار ۴ برای مواجهه الگوریتم با دشواری‌های عددی
nit تعداد تکرارهای الگوریتم را نشان می‌دهد.
message وضعیت خروجی الگوریتم را توصیف می‌کند.

روش حداقل مربعات

```
import numpy as np
def f(x):
   return np.array([10 * (x[1] - x[0]**2), (1 - x[0])])

from scipy.optimize import least_squares
input = np.array([2, 2])
res = least_squares(f, input)

print (res)
```

ماژول datetime

```
import datetime
a=datetime.datetime.now()
for I in range(10000):
        print (i)
b=datetime.datetime.now()
c=b-a
print(c.total_seconds())
```

```
import datetime
a=datetime.datetime.now()
for I in range(10000):
        b=datetime.datetime.now()
        c=b-a
        d=c.total_seconds()
         if d<15:
               print (i)
```