人工智能程序设计

M3 人工智能基本方法

3 数值计算与优化

张 莉

数值计算

- 数值计算指有效使用数字计算机求数学问题近似解的方法与过程，以及由相关理论构成的学科，主要包括：
  - 方程求解
  - 矩阵特征值和特征向量求解
  - 插值方法
  - 最优化
  - 数值积分
  - …

numerical computation

数值计算与优化

1. 方程求根与最小二乘法
2. 极值与梯度下降
3. 特征值分解与奇异值分解

人工智能程序设计

1 方程求根与最小二乘法

线性方程组求解

```
import numpy as np
a = np.array([[2, -3, 1], [3, 2, 0], [1, 7, -1]])
b = np.array([1, 13, 16])
x = np.linalg.solve(a, b)
print(x)
print(np.allclose(np.dot(a, x), b))
```

$$2x_0-3x_1+x_2=1$$
$$3x_0+2x_1=13$$
$$x_0+7x_1-x_2=16$$

```
a = np.matrix([[2, -3, 1], [3, 2, 0], [1, 7, -1]])
b = np.matrix([1, 13, 16]).T
```

```
from scipy.linalg import solve
x = solve(a, b)
```

超定线性方程组求解

```
import scipy
a = np.array([[1,2], [4,5], [2,1]])
b = np.array([3, 6, 0])
x = scipy.linalg.solve(a, b)
print(x)
```

$$x_0+2x_1=3$$
$$4x_0+5x_1=6$$
$$2x_0+x_1=0$$

```
x = scipy.linalg.lstsq(a, b)
```

```
`a` must be square and of full-rank, i.e., all rows (or, equivalently,
columns) must be linearly independent; if either is not true, use
`lstsq` for the least-squares best "solution" of the
system/equation.
```

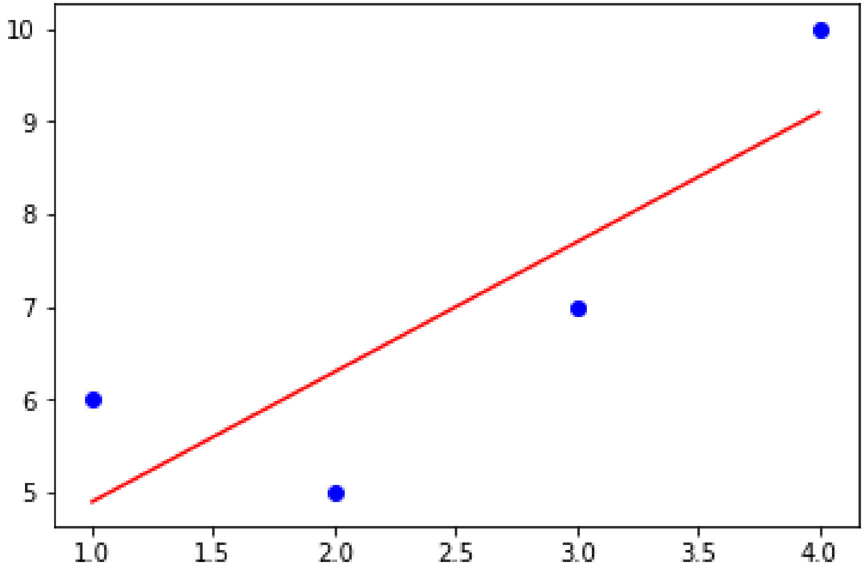
最小二乘法Least Squares

- 最小二乘法（又称最小平方法）是一种数学优化技术，通过**最小化误差的平方和**寻找数据的最佳函数匹配，最小平方所涵义的最佳拟合即残差residual （观测值与模型提供的拟合值之间的差距）平方总和的最小化。
- 最小二乘法是对超定方程组以回归分析求得近似解的标准方法，最重要的应用是在**曲线拟合**上。
- 利用最小二乘法可以简便地**求得未知的数据**，并使得这些求得的数据与实际数据之间误差的平方和为最小。

最小二乘法和线性拟合

残差平方和

$$S_r = \sum_{i=1}^{n}(y_i - f(x_i))^2$$



(x,y): (1,6), (2,5), (3,7), (4,10)

问题：寻找最合适的直线$y=a_0+a_1x$

$$S_r = \sum_{i=1}^{n}(y_i - a_0 - a_1x_i)^2$$

$$\frac{\partial S_r}{\partial a_0} = 0$$

$$\Rightarrow -2\sum_{i=1}^{n}(y_i - a_0 - a_1x_i) = 0$$

$$\Rightarrow \sum_{i=1}^{n}(y_i - a_0 - a_1x_i) = 0$$

$$\frac{\partial S_r}{\partial a_1} = 0$$

$$\Rightarrow -2\sum_{i=1}^{n}x_i(y_i - a_0 - a_1x_i) = 0$$

$$\Rightarrow \sum_{i=1}^{n}(x_iy_i - a_0x_i - a_1x_i^2) = 0$$

$$a_1 = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{n\sum x^2 - \left(\sum x\right)^2}$$

$$a_0 = \bar{y} - a_1\bar{x}$$

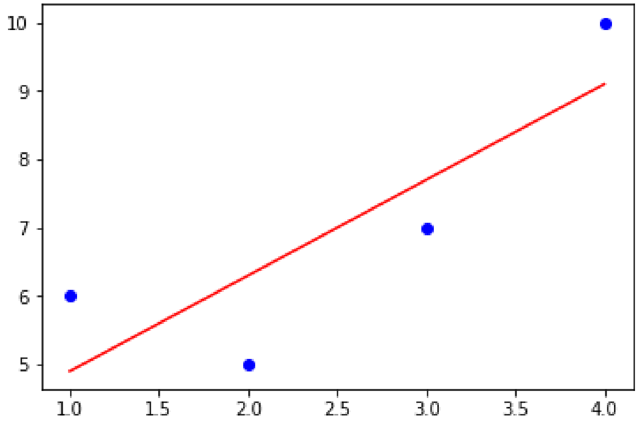
最小二乘法和线性拟合

(x,y): (1,6), (2,5), (3,7), (4,10)

$$a_1 = \frac{n\sum xy - (\sum x)(\sum y)}{n\sum x^2 - (\sum x)^2}$$

$$a_0 = \bar{y} - a_1\bar{x}$$

```
X = np.array([1,2,3,4])
y = np.array([6,5,7,10])
n = len(y)
a1 = (n*sum(X*y)-sum(X)*sum(y))/(n*sum(X*X)-sum(X)**2)
a0 = np.mean(y) - a1*np.mean(X)
```



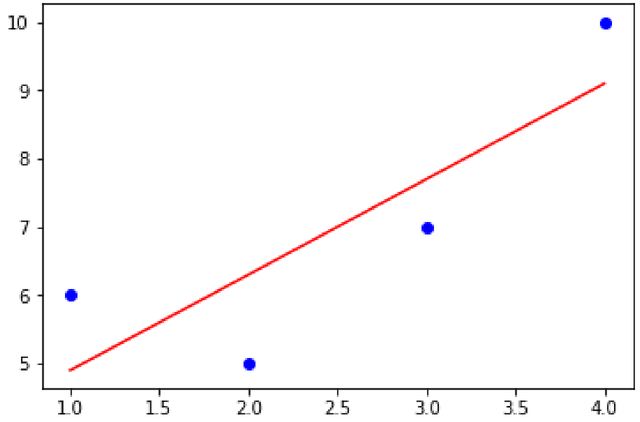
y = 3.5+1.4x

最小二乘法和线性拟合

(x,y): (1,6), (2,5), (3,7), (4,10)

$$a_0+1a_1=6$$
$$a_0+2a_1=5$$
$$a_0+3a_1=7$$
$$a_0+4a_1=10$$

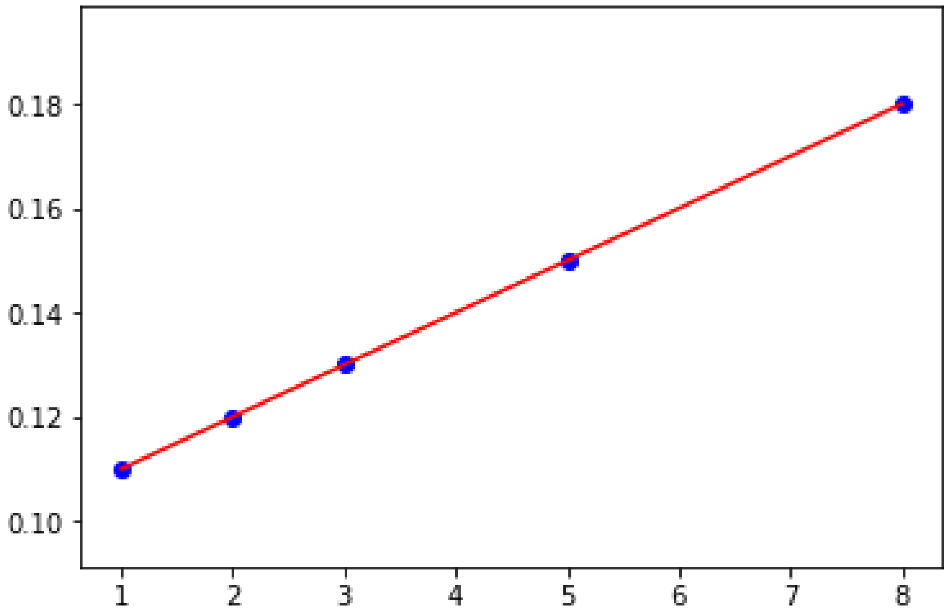
```
X = np.array([[1,1],[1,2],[1,3],[1,4]])
y = np.array([6,5,7,10])
r = scipy.linalg.lstsq(X, y)
print(r)
```



y = 3.5+1.4x

超定线性方程组和线性拟合

[维基百科] 假设五个国家的国民生产总值分别是1、2、3、5、8（单位10亿美元），又假设这五个国家的贫困比例分别是11%、12%、13%、15%、18%。



y = 0.1+0.01x

```
...
X = np.array([1,2,3,5,8])
y = np.array([0.11,0.12,0.13,0.15,0.18])
plt.scatter(X, y, color = 'blue')
plt.plot(X, r[0][0]+r[0][1]*X, color = 'red')
plt.show()
```

一元线性回归预览

```
from sklearn import linear_model

clf = linear_model.LinearRegression()
X = np.array([1,2,3,4]).reshape(-1,1)
y = np.array([6,5,7,10])
clf.fit(X, y)       # 训练模型
b, a = clf.coef_, clf.intercept_
print(b, a)
x = [[4]]
print(clf.predict(x))     # 预测
```

[1.4] 3.5000000000000001

最小二乘法和超定线性方程组

$X_0+2x_1=3$
$4x_0+5x_1=6$
$2x_0+x_1=0$

$y=a_0+a_1x$

$X_0+2x_1=3$
$x_0+5/4*x_1=6/4$
$x_0+1/2*x_1=0$

```
X = np.array([2,5/4,1/2])
y = np.array([3,6/4,0])
n = len(y)
a1 = (n*sum(X*y)-sum(X)*sum(y))/(n*sum(X*X)-sum(X)**2)
a0 = np.mean(y) - a1*np.mean(X)
```

```
>>> a0,a1
(-1.0, 2.0)
```

```
X = np.array([[1,2],[1,5/4],[1,1/2]])
y = np.array([3,6/4,0])
r = scipy.linalg.lstsq(X,y)
```

非线性方程组求解

```
import numpy as np
from scipy.optimize import fsolve, leastsq

def f(x):
      return [x[0]*np.cos(x[1])-4, x[1]*x[0]-x[1]-5]
result = fsolve(f, [1,1])
print("==========")
print("Function name:", fsolve.__name__)
print("Result:", result)
print()
result = leastsq(f, [1,1])
print("==========")
print("Function name:", leastsq.__name__)
print("Result:", result[0])
```

$x_0+\cos(x_1)=4$

$x_0x_1-x_1=5$

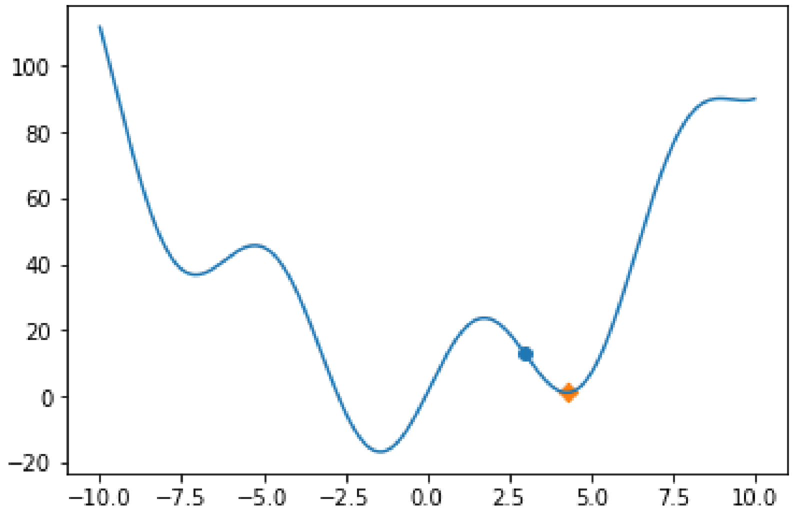
```
==========
Function name: fsolve
Result: [6.50409711 0.90841421]

==========
Function name: leastsq
Result: [6.50409711 0.90841421]
```

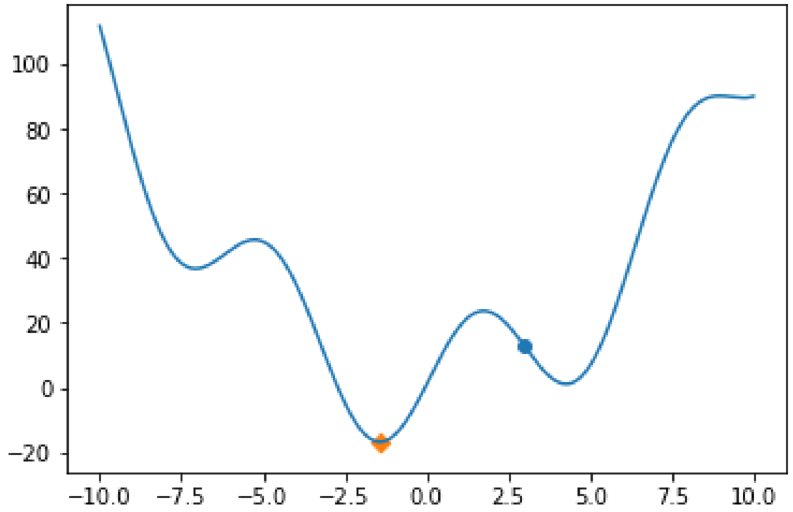
人工智能程序设计

2 极值与梯度下降法

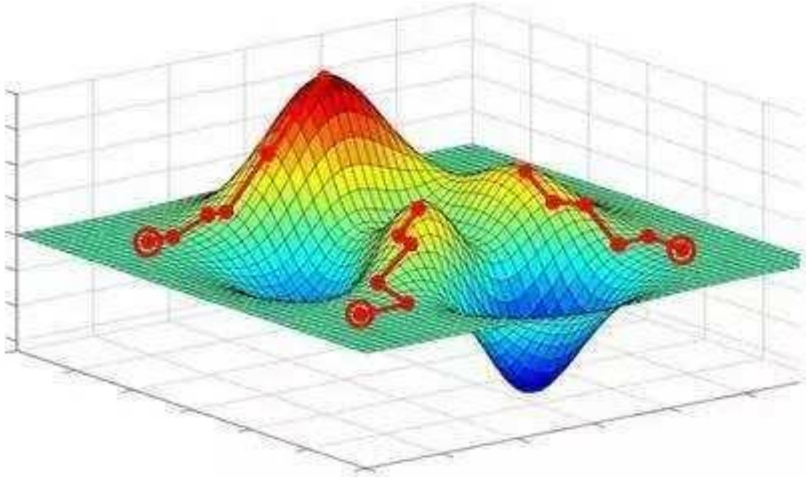
极值

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import minimize, basinhopping
def f(x):
    return x**2+1
    # return x**2+20*np.sin(x)+1
x = np.linspace(-10, 10, 1000)
x0 = 3
x_min = minimize(f, x0).x
# x_min = basinhopping(f, x0, stepsize = 3).x
plt.plot(x, f(x))
plt.scatter(x0, f(x0), marker='o')
plt.scatter(x_min, f(x_min), marker='D')
plt.show()
```



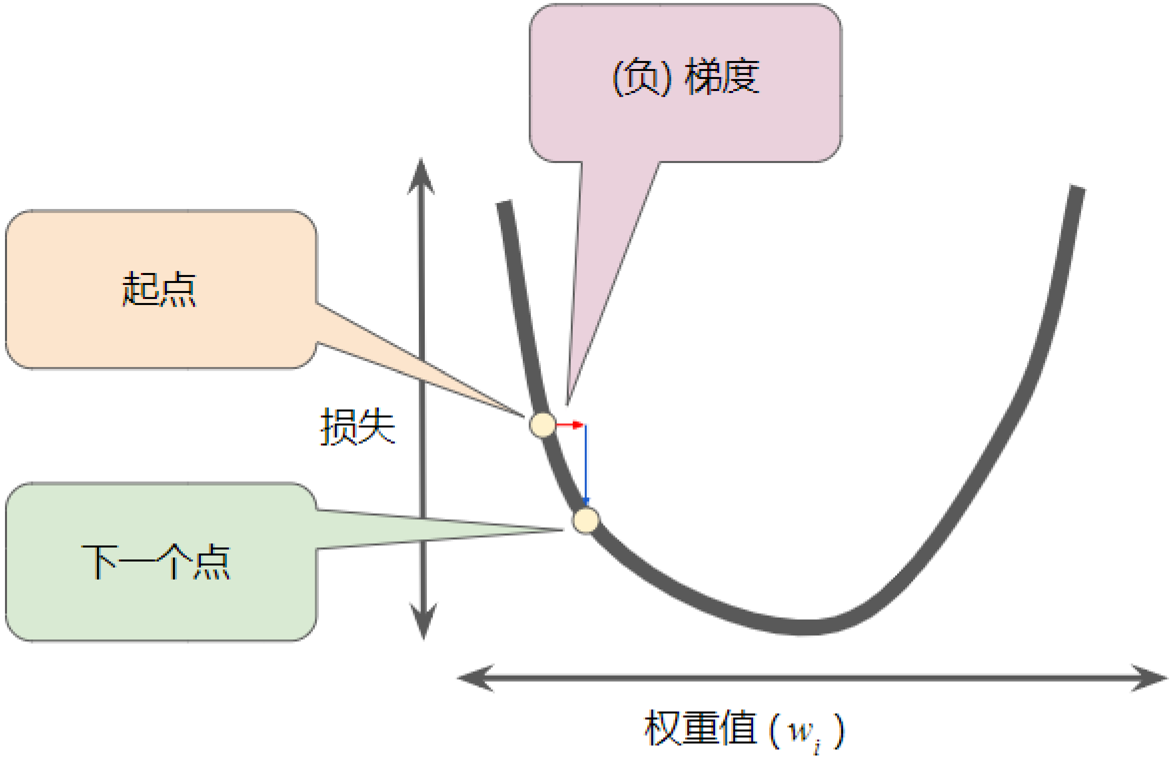


梯度下降法Gradient descent求极值



- [维基百科]梯度下降法是一个一阶最优化算法，通常也称为最速下降法。 要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索
- 求解无约束最优化问题最常用的方法
- 每一步主要的操作是求解目标函数的梯度向量，将当前位置的负梯度方向作为搜索方向
- 三要素：出发点，下降方向，下降步长

Gradient descent



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$ 损失函数

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

From: Google机器学习速成课程

```python
def f(x):
    r = x**2+20*np.sin(x)+1
    # r = x**2+1
    return r

def dr_f(x):
    r = 2*x+20*np.cos(x)
    # r = 2*x
    return r
```

```python
def GD(eps, max_iters):
    alpha = 0.02    # 学习率/步长
    x = 3   # 初始权重
    iters = 0
    y1 = f(x)
    y2 = y1+0.1
    while abs(y1 - y2) > eps and iters < max_iters:
        y1 = y2
        x = x - alpha * dr_f(x)
        print(x)
        y2 = f(x)
        iters += 1
    return x, y2

if __name__ == '__main__':
    x_min, f_xmin = GD(1e-8, 1000)
    print(x_min)
```

人工智能程序设计

3 特征值分解与奇异值分解

特征值分解

```
>>> x = np.array([[3, 2],[1, 4]])

>>> e, v = scipy.linalg.eig(x)
>>> e
array([2.+0.j, 5.+0.j])
>>> v
array([[-0.89442719, -0.70710678],
       [ 0.4472136 , -0.70710678]])
```

$$\begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}$$

方阵适用

速查表：scipy 和 numpy 库中与特征值问题相关的各函数

函数名	所属库类	特征值问题的类型 [1]				函数特性						
		广义		标准		矩阵形式 [4]				部分求解 [2]	返回值	
		左	右	左	右	一般	对称	对称带状	非正定 [5]		只求特征值	参数复用 [3]
eig	numpy.linalg											
eigh	numpy.linalg											
eigvals	numpy.linalg											
eigvalsh	numpy.linalg											
eig	scipy.linalg					A,B	A,B	A,B	A,B			
eig_banded	scipy.linalg											
eigh	scipy.linalg						A,B	A,B				
eigvals	scipy.linalg					A,B	A,B	A,B	A,B			
eigvals_banded	scipy.linalg											
eigvalsh	scipy.linalg						A,B	A,B				
eigs	scipy.sparse.linalg					A	A,B	A,B	A,B			
eigsh	scipy.sparse.linalg						A,B	A,B	B			

待考证 From: 范雨https://fanyublog.com/2015/11/15/eig_in_numpy_scipy/

奇异值分解

```
>>> x = np.array([[3, 2, 1], [1, 3, 4]])
>>> s,u,v = scipy.linalg.svd(x)
>>> s
array([[-0.53895353, -0.8423355 ],
       [-0.8423355 ,  0.53895353]])
>>> u
array([5.85814143, 2.38373214])
>>> v
array([[-0.41979118, -0.61536813, -0.66715622],
       [-0.83400854, -0.02844715,  0.55101771],
       [ 0.35805744, -0.78772636,  0.50128041]])
```

$$A = U\Sigma V^T$$

```
a : (M, N) array_like
    Matrix to decompose.
```

M3.3小结

01 方程求根与最小二乘法

02 极值与梯度下降

03 特征值分解与奇异值分解