# 插值

插值是数值分析里面逼近的重要方法，利用它可通过函数在有限个点处的取值状况，估算出函数在其他点处的近似值。

# 线性插值

线性插值是用一系列首尾相连的线段依次连接相邻各点，每条线段内的点的高度作为插值获得的高度值。 以$(x_i, y_i)$表示某条线段的前一个端点，$(x_{i+1}, y_{i+1})$表示该线段的后一个端点，则对于在$[x_i, x_{i+1}]$范围内的横坐标为$x$的点，其高度$y$为

$$y = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i)$$

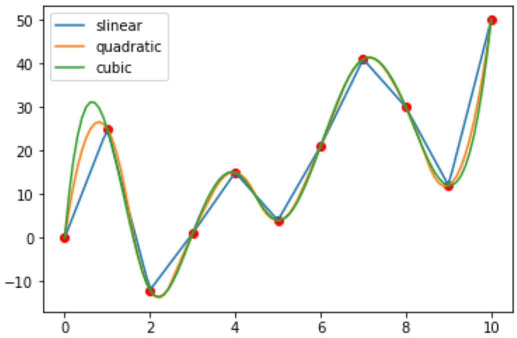
# 二次插值

如果按照线性插值的形式，以每3个相邻点做插值，就得到了二次插值：

$$y = \frac{(x - x_{i+1})(x - x_{i+2})}{(x_i - x_{i+1})(x_i - x_{i+2})} \cdot y_i + \frac{(x - x_i)(x - x_{i+2})}{(x_{i+1} - x_i)(x_{i+1} - x_{i+2})} \cdot y_{i+1} + \frac{(x - x_i)(x - x_{i+1})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \cdot y_{i+2}$$

In [3]:

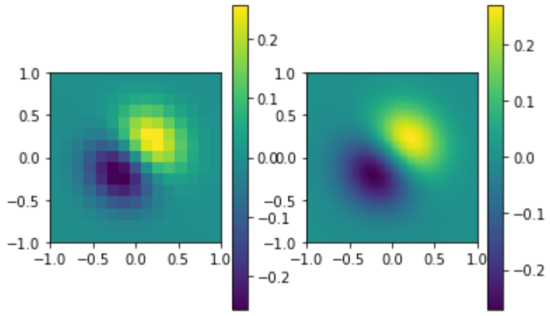
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy import interpolate

n = 10
x = np.linspace(0,10,n)
y = np.sin(x) + np.random.rand(n)*5
x = [0, 1, 2, 3, 4, 5, 6, 7, 8 ,9, 10]
y = [0, 25, -12, 1, 15, 4, 21, 41, 30, 12, 50]
x_new = np.linspace(0,10,100)

fig = plt.figure()

plt.plot(x,y,"ro")

for kind in ["slinear","quadratic","cubic"]:
    # slinear 线性插值,"quadratic","cubic" 为2阶、3阶B样条曲线插值

    f = interpolate.interp1d(x,y,kind=kind)
    y_new=f(x_new)

    plt.plot(x_new,y_new,label=str(kind))

plt.legend()
plt.show()
```



In [4]:

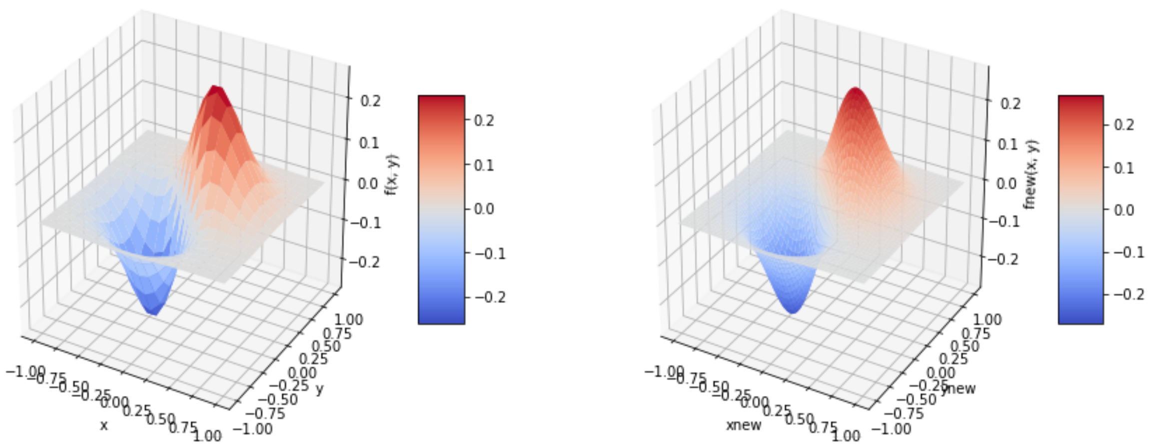
```
# -*- coding: utf-8 -*-
"""
演示二维插值。
"""
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

def func(x, y):
    return (x+y)*np.exp(-5.0*(x**2 + y**2))

# X-Y轴分为15*15的网格
y,x= np.mgrid[-1:1:15j, -1:1:15j]

fvals = func(x,y) # 计算每个网格点上的函数值 15*15的值
print(len(fvals[0]))

#三次样条二维插值
newfunc = interpolate.interp2d(x, y, fvals, kind='cubic')

# 计算100*100的网格上的插值
xnew = np.linspace(-1,1,100)#x
ynew = np.linspace(-1,1,100)#y
fnew = newfunc(xnew, ynew)#仅仅是y值 100*100的值

# 绘图
# 为了更明显地比较插值前后的区别，使用关键字参数interpolation='nearest'
# 关闭imshow()内置的插值运算。
plt.subplot(121)
im1=plt.imshow(fvals, extent=[-1,1,-1,1], interpolation='nearest', origin="lower")#pl.cm.jet
#extent=[-1,1,-1,1]为x,y范围 favals为
plt.colorbar(im1)

plt.subplot(122)
im2=plt.imshow(fnew, extent=[-1,1,-1,1], interpolation='nearest', origin="lower")
plt.colorbar(im2)
plt.show()
```

15



In [3]:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
from scipy import interpolate
import matplotlib.cm as cm
import matplotlib.pyplot as plt

def func(x, y):
 return (x+y)*np.exp(-5.0*(x**2 + y**2))

# X-Y轴分为20*20的网格
x = np.linspace(-1, 1, 20)
y = np.linspace(-1,1,20)
x, y = np.meshgrid(x, y)#20*20的网格数据

fvals = func(x,y) # 计算每个网格点上的函数值 15*15的值

fig = plt.figure(figsize=(15, 6))
#Draw sub-graph1
ax=plt.subplot(1, 2, 1,projection = '3d')
surf = ax.plot_surface(x, y, fvals, cmap=cm.coolwarm)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
plt.colorbar(surf, shrink=0.5, aspect=5)#标注

#二维插值
newfunc = interpolate.interp2d(x, y, fvals, kind='cubic')#newfunc为一个函数

# 计算100*100的网格上的插值
xnew = np.linspace(-1,1,100)#x
ynew = np.linspace(-1,1,100)#y
fnew = newfunc(xnew, ynew)#仅仅是y值 100*100的值 np.shape(fnew) is 100*100
xnew, ynew = np.meshgrid(xnew, ynew)
ax2=plt.subplot(1, 2, 2,projection = '3d')
surf2 = ax2.plot_surface(xnew, ynew, fnew, cmap=cm.coolwarm)
ax2.set_xlabel('xnew')
ax2.set_ylabel('ynew')
ax2.set_zlabel('fnew(x, y)')
plt.colorbar(surf2, shrink=0.5, aspect=5)#标注

plt.show()
```



# 牛顿插值

1. 已知$n$个点的坐标$(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$,求一个$n-1$次多项式经过这些点
2. 定义如下内容：

- 一阶差商

$$f[x_i,x_j]=\frac{f(x_i)-f(x_j)}{x_i-x_j}(i\neq j,x_i\neq x_j)$$

- 二阶差商

$$f[x_i,x_j,x_k]=\frac{f[x_i,x_j]-f[x_j,x_k]}{x_i-x_k}(i\neq k)$$

- $n$阶差商

$$f[x_0,x_1,\cdots,x_n]=\frac{f[x_0,x_1,\cdots,x_{n-1}]-f[x_1,x_2,\cdots,x_n]}{x_0-x_n}$$

3. 生成差商表

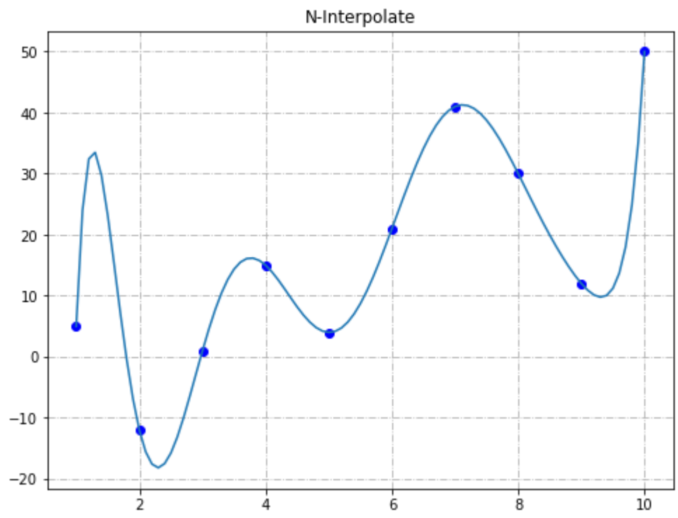
$$\begin{matrix} x_0 & f(x_0) & & & \\ x_1 & f(x_1) & f[x_1,x_0] & & \\ x_2 & f(x_2) & f[x_2,x_1] & f[x_2,x_1,x_0] & \\ x_3 & f(x_3) & f[x_3,x_2] & f[x_3,x_2,x_1] & f[x_3,x_2,x_1,x_0] \end{matrix}$$

1. 最终根据差商表生成的牛顿插值公式为：

$$\begin{aligned} N(x)=&f(x_0)\\ &+f[x_0,x_1](x-x_0)\\ &+f[x_0,x_1,x_2](x-x_0)(x-x_1)\\ &+f[x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1}) \end{aligned}$$

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

def divided_diff(x, y):
    '''
    计算差商表
    '''
    n = len(y)
    coef = np.zeros([n, n])

    coef[:,0] = y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    return coef

def newton_poly(coef, x_data, x):
    '''
    计算新的插值点的值
    '''
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p

x = [1, 2, 3, 4, 5, 6, 7, 8 ,9, 10]
y = [5, -12, 1, 15, 4, 21, 41, 30, 12, 50]

# 计算差商
a_s = divided_diff(x, y)[0, :]

# 计算新点的值
x_new = np.arange(1, 10.1, .1)
y_new = newton_poly(a_s, x, x_new)

plt.figure(figsize = (8, 6))
plt.plot(x, y, 'bo')
plt.plot(x_new, y_new)
plt.grid(True, linestyle='-.')
plt.title('N-Interpolate')
```

Out[4]: Text(0.5,1,'N-Interpolate')



# 拉格朗日插值法

1. 已知$n$个点的坐标$(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$,求一个$n-1$次多项式经过这些点
2. 假设$n-1$次多项式为$y=a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}$
3. 讲$n$个点带入多项式得：

$$
\begin{aligned}
&y_1=a_0+a_1x_1+a_2x_1^2+\cdots+a_{n-1}x_1^{n-1}\\
&y_2=a_0+a_1x_2+a_2x_2^2+\cdots+a_{n-1}x_2^{n-1}\\
&\cdots\cdots\\
&y_n=a_0+a_1x_n+a_2x_n^2+\cdots+a_{n-1}x_n^{n-1}
\end{aligned}
$$

4. 直接求方程组，得拉格朗日多项式为：

$$
L(x)=y_1\frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)}+y_2\frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)}+\\
\cdots y_n\frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})}
$$

1. 或者直接写为如下形式：

$$
L(x)=\sum_{i=1}^{n}y_i\prod_{k=1,k\neq i}^{n}\frac{x-x_k}{x_i-x_k}
$$

# 第四次作业，请补全下述拉格朗日插值代码

In [5]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt


plt.figure(figsize = (8, 6))
plt.plot(x, y, "bo", markersize=8, label="init")


x = [1, 2, 3, 4, 5, 6, 7, 8 ,9, 10]
y = [5, -12, 1, 15, 4, 21, 41, 30, 12, 50]



plt.title('L-interpolate')
plt.grid(True, linestyle='-.')

plt.show()
```