In [1]:

```python
#导入库
import numpy as np
import pandas as pd
from sympy import *
import re
from sympy.abc import x, y
import math
import matplotlib.pyplot as plt
```

数据如下 PPT 53页 例15-3

例 15.3 已知$x^{(0)}=(41,49,61,78,96,104)$，试建立 GM(2,1)模型。

In [2]:

```python
#导入数据
# data = np.array([71.1,72.4,72.4,72.1,71.4,72.0,71.6])
data=np.array([41,49,61,78,96,104])
```

##step1

定义 15.2 设原始序列

$$x^{(0)}=(x^{(0)}(1),x^{(0)}(2),\cdots,x^{(0)}(n)),$$

其 1 次累加生成序列（1—AGO）$x^{(1)}$和 1 次累减生成序列（1—IAGO）$\alpha^{(1)}x^{(0)}$分别为

$$x^{(1)}=(x^{(1)}(1),x^{(1)}(2),\cdots,x^{(1)}(n)),$$

和

$$\alpha^{(1)}x^{(0)}=(\alpha^{(1)}x^{(0)}(2),\cdots,\alpha^{(1)}x^{(0)}(n)),$$

其中

$$\alpha^{(1)}x^{(0)}(k)=x^{(0)}(k)-x^{(0)}(k-1),\quad k=2,3,\cdots,n,$$

In [3]:

```python
#计算累加生成序列 x1
data_cumsum = np.cumsum(data)
data_cumsum
```

Out[3]:

```
array([ 41,  90, 151, 229, 325, 429])
```

In [4]:

```
#生成累减序列 a1x0
data_diff = [ data[i+1]-data[i] for i in range(len(data)-1)]
data_diff
```

Out[4]:

```
[8, 12, 17, 18, 8]
```

##step2

$x^{(1)}$的均值生成序列为

$$z^{(1)} = (z^{(1)}(2), z^{(1)}(3), \cdots, z^{(1)}(n)),$$

In [5]:

```
#均值生成序列 z1
data_mean = (data_cumsum[1:]+data_cumsum[:-1])/2
data_mean
```

Out[5]:

```
array([ 65.5, 120.5, 190. , 277. , 377. ])
```

##step3

且

$$B = \begin{bmatrix} -x^{(0)}(2) & -z^{(1)}(2) & 1 \\ -x^{(0)}(3) & -z^{(1)}(3) & 1 \\ \vdots & \vdots & \vdots \\ -x^{(0)}(n) & -z^{(1)}(n) & 1 \end{bmatrix},$$

$$Y = \begin{bmatrix} \alpha^{(1)}x^{(0)}(2) \\ \alpha^{(1)}x^{(0)}(3) \\ \vdots \\ \alpha^{(1)}x^{(0)}(n) \end{bmatrix} = \begin{bmatrix} x^{(0)}(2) - x^{(0)}(1) \\ x^{(0)}(3) - x^{(0)}(2) \\ \vdots \\ x^{(0)}(n) - x^{(0)}(n-1) \end{bmatrix},$$

In [6]:

```
#构建矩阵B
B= np.array([ [-data[i+1],-data_mean[i],1] for i in range(len(data)-1)])
B
```

Out[6]:

```
array([[ -49. ,  -65.5,    1. ],
       [ -61. , -120.5,    1. ],
       [ -78. , -190. ,    1. ],
       [ -96. , -277. ,    1. ],
       [-104. , -377. ,    1. ]])
```

In [7]:

```
#构建Y矩阵
Y = np.array([ [data[i+1] - data[i]] for i in range(len(data)-1)])
Y
```

Out[7]:

```
array([[ 8],
       [12],
       [17],
       [18],
       [ 8]])
```

##step4

$$\hat{u} = \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \hat{b} \end{bmatrix} = (B^T B)^{-1} B^T Y$$

In [8]:

```
#使用最小二乘法求解系数 a1 a2 b
a1,a2,b = np.linalg.inv(B.T.dot(B)).dot(B.T).dot(Y)
a1,a2,b
```

Out[8]:

```
(array([-1.09219635]), array([0.19590335]), array([-31.79834712]))
```

In [9]:

```
init_printing()
# 定义符号常量x 与 f(x) g(x)。这里的f g还可以用其他字母替换，用于表示函数
f, g = symbols('f g', cls=Function)
```

##step5

定义 15.3 称

$$\frac{d^2x^{(1)}}{dt^2}+a_1\frac{dx^{(1)}}{dt}+a_2x^{(1)}=b \qquad (15.9)$$

为 GM(2,1)模型的白化方程。

In [10]:

```
#sympy解微分方程官方文档
#https://www.osgeo.cn/sympy/tutorial/solvers.html#solving-differential-equations
```

In [11]:

```
#参考资料：https://www.codenong.com/cs106740712/
#利用sympy库求解微分方程
#计算白化方程 (dx1)^2 + a1*(dx1/dt) + a2*x1 = b
# 用diffeq代表微分方程
#参考资料：https://vimsky.com/examples/usage/python-sympy-diff-method.html
#f(x).diff(x, x) 二阶微分
#sympy.Eq() 用于创建等式
#sympy.diff() 用于求导 具体参数为 (expression, reference variable)
#sympy.diff(f(x),x) 对f(x)求一阶偏导或导数
#sympy.diff(f(x),x,x) 对f(x)求二阶偏导或导数
diffeq = Eq(f(x).diff(x, x) + a1[0] * f(x).diff(x) + a2[0] * f(x), b[0])
# 调用dsolve函数,返回一个Eq对象,并提取带参数方程
sol = dsolve(diffeq, f(x))
sol
```

Out[11]:

$$f(x)=C_1e^{0.226223404169048x}+C_2e^{0.865972945416752x}-162.316507096965$$

In [12]:

```
# 提取方程中f(x)的表达式
differential_equation = str(sol.args[1])
differential_equation
```

Out[12]:

```
'C1*exp(0.226223404169048*x) + C2*exp(0.865972945416752*x) - 162.316507096965'
```

In [13]:

```
 # 使用正则表达式提取齐次微分方程的根与非齐次微分方程的特解
equation_parameter = re.findall("-?\d+.?\d+", differential_equation.replace(' ', ''))
equation_parameter
```

Out[13]:

```
['0.226223404169048', '0.865972945416752', '-162.316507096965']
```

In [14]:

```
def solving_equation(x1, equation_parameter):
    # 二阶齐次微分方程的根分为两个不同的实根、两个相同的实根以及两个虚根
    # 不同情况方程的形式不同，根据predict函数中求得的带参数的方程来决定使用的策略
    # 下面以两个不同的实根为例
    parameter = solve(
        [x * math.exp(equation_parameter[0] * 0) + y * math.exp(equation_parameter[1] * 0) + equ
         x * math.exp(equation_parameter[0] * (len(x1)-1)) + y * math.exp(equation_parameter[1]
         equation_parameter[2] - x1[len(x1) - 1]])
    print(parameter)
    # 返回X1的预测值
    return [parameter[x] * math.exp(equation_parameter[0] * i) + parameter[y] * math.exp(equati
            equation_parameter[2] for i in range(len(x1))]
```

In [15]:

```
for i in range(len(equation_parameter)):
    equation_parameter[i] = float(equation_parameter[i])
    # 利用边界条件，取X1中第一个数和最后一个数,构造方程组，求参数C1和C2，并返回预测值
predict_data = solving_equation(data_cumsum, equation_parameter)
np.array(predict_data)
```

```
{x: 203.849012866393, y: -0.532505769428176}
```

Out[15]:

```
array([41.0000000000000, 92.0148144078690, 155.156052197629,
       232.367192369491, 324.521982077498, 429.000000000000], dtype=object)
```

In [16]:

```
result = np.ediff1d(predict_data)
result = np.insert(result,0,predict_data[0])
result
```

Out[16]:

```
array([41.0000000000000, 51.0148144078690, 63.1412377897597,
       77.2111401718622, 92.1547897080068, 104.478017922502], dtype=object)
```

In [17]:

```
#计算残差
residual = data - result
```

In [18]:

```
#计算相对误差
relative_error = residual/data
```

In [19]:

```
#合并数据
all = np.vstack((data,result,residual,relative_error)).T
```

In [20]:

```
#生成表格
df = pd.DataFrame(all,columns=['原始数据','预测数据','残差','相对误差'])
df
```

Out[20]:

	原始数据	预测数据	残差	相对误差
0	41	41.0000000000000	2.84217094304040e-14	6.93212425131805e-16
1	49	51.0148144078690	-2.01481440786898	-0.0411186613850813
2	61	63.1412377897597	-2.14123778975966	-0.0351022588485190
3	78	77.2111401718622	0.788859828137845	0.0101135875402288
4	96	92.1547897080068	3.84521029199320	0.0400542738749291
5	104	104.478017922502	-0.478017922502204	-0.00459632617790581

In [21]:

```
#绘制图像
plt.scatter(data,result)  #如果在 x-y = 0  (x=y)直线上，说明预测值与真实值误差很小
#绘制出散点的拟合曲线
plt.plot(data,data,'r')
plt.xlabel('y_test')
plt.ylabel('predictions')
plt.show()
```