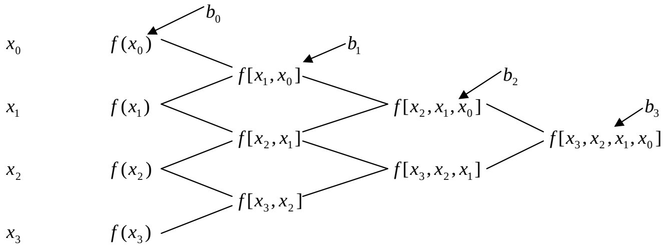
1. Newton divided difference formula 生成了牛顿插值多项式的各项系数。所以我们先定义 Newton_formula(x, y)函数来计算各项系数。其中输入的 x 是一个包含了$x_0, x_1, \ldots, x_{n-1}$的列表，而 y 是一个包含了$f(x_0), f(x_1), \ldots, f(x_{n-1})$的列表。将 x,y 添加到$n \times (n+1)$矩阵的前两列。$\begin{pmatrix} x_0 & f(x_0) & \ldots & 0 \\ x_1 & f(x_1) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1} & f(x_{n-1}) & \ldots & 0 \end{pmatrix}$

通过以下步骤计算 divided difference，并确保我们所需要的系数在对角元上。



$$\begin{pmatrix} x_0 & f(x_0) & 0 & 0 & \ldots & 0 \\ x_1 & f(x_1) & f(x_1, x_0) & 0 & \ldots & 0 \\ x_2 & f(x_2) & f(x_2, x_1) & f(x_2, x_1, x_0) & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n-1} & f(x_{n-1}) & f(x_{n-1}, x_{n-2}) & f(x_{n-1}, x_{n-2}, x_{n-3}) & \ldots & 0 \end{pmatrix}$$

然后取对角元素$f(x_0), f(x_1, x_0), f(x_2, x_1, x_0)$ ...存入矩阵 result 并返回即可。

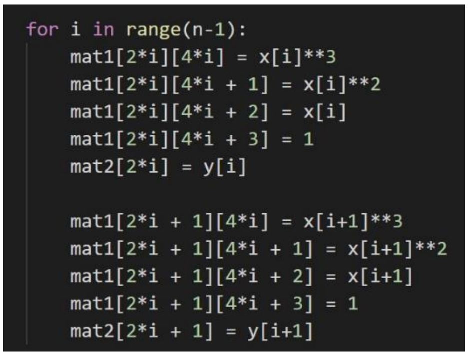
再定义函数 polynomial(x, b)将系数组合成多项式，其中 x 是给定的一系列$x_0, x_1, \ldots, x_{n-1}$构成的矩阵，b 是$b_0, b_1, \ldots, b_n$构成的系数矩阵。将对应位置的元素取出构成$f(t) = b_0 + b_1(t - x_0) + b_2(t - x_0)(t - x_1) + \cdots$返回即可。

再定义 spline(x, y)函数来计算 cubic spline interpolation 的每个区间内的函数。$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$。对于 n 个点，共有(n-1)个函数，4(n-1)个系数，所以构造$4(n-1) \times (4n-1)$的矩阵 mat1，以及一个$4(n-1)$维列矩阵 mat2，用 numpy.linalg.solve()函数暴力求解。

其中 for i 循环设置了以下(2n-2)个方程：

$$f_i(x_i) = a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i = f(x_i) \quad \text{for } i = 1, 2, \cdots, n-1$$

$$f_i(x_{i+1}) = a_i x_{i+1}^3 + b_i x_{i+1}^2 + c_i x_{i+1} + d_i = f(x_{i+1}) \quad \text{for } i = 1, 2, \cdots, n-1$$



```
for i in range(n-1):
    mat1[2*i][4*i] = x[i]**3
    mat1[2*i][4*i + 1] = x[i]**2
    mat1[2*i][4*i + 2] = x[i]
    mat1[2*i][4*i + 3] = 1
    mat2[2*i] = y[i]

    mat1[2*i + 1][4*i] = x[i+1]**3
    mat1[2*i + 1][4*i + 1] = x[i+1]**2
    mat1[2*i + 1][4*i + 2] = x[i+1]
    mat1[2*i + 1][4*i + 3] = 1
    mat2[2*i + 1] = y[i+1]
```

接着 for j 循环设置了以下(n-2)个方程：

$$3a_i x_{i+1}^2 + 2b_i x_{i+1} + c_i = 3a_{i+1} x_{i+1}^2 + 2b_{i+1} x_{i+1} + c_{i+1} \quad \text{for } i = 1,2,\cdots,n-2$$

```
for j in range(n-2):
    mat1[2*n - 2 + j][4*j] = 3 * (x[j+1]**2)
    mat1[2*n - 2 + j][4*j + 1] = 2 * x[j+1]
    mat1[2*n - 2 + j][4*j + 2] = 1

    mat1[2*n - 2 + j][4*j + 4] = -3 * (x[j+1]**2)
    mat1[2*n - 2 + j][4*j + 5] = -2 * x[j+1]
    mat1[2*n - 2 + j][4*j + 6] = -1
```

for k 循环设置了以下(n-2)个方程：

$$6a_i x_{i+1} + 2b_i = 6a_{i+1} x_{i+1} + 2b_{i+1} \quad \text{for } i = 1,2,\cdots,n-2$$

```
for k in range(n-2):
    mat1[3*n - 4 + k][4*k] = 6 * x[k+1]
    mat1[3*n - 4 + k][4*k + 1] = 2

    mat1[3*n - 4 + k][4*k + 4] = -6 * x[k+1]
    mat1[3*n - 4 + k][4*k + 5] = -2
```

最后一部分设置了最后两个方程：

$$6a_1 x_1 + 2b_1 = 0$$

$$6a_{n-1} x_n + 2b_{n-1} = 0$$

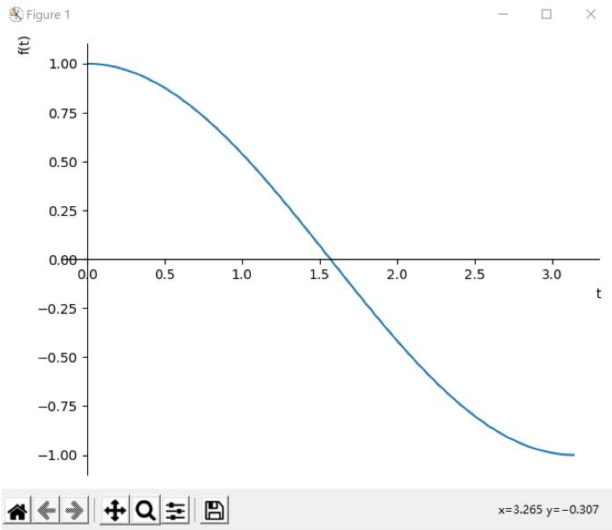
```
mat1[4*n - 6][0] = 6 * x[0]
mat1[4*n - 6][1] = 2
mat1[4*n - 5][4*n -8] = 6 * x[n-1]
mat1[4*n - 5][4*n -7] = 2
```

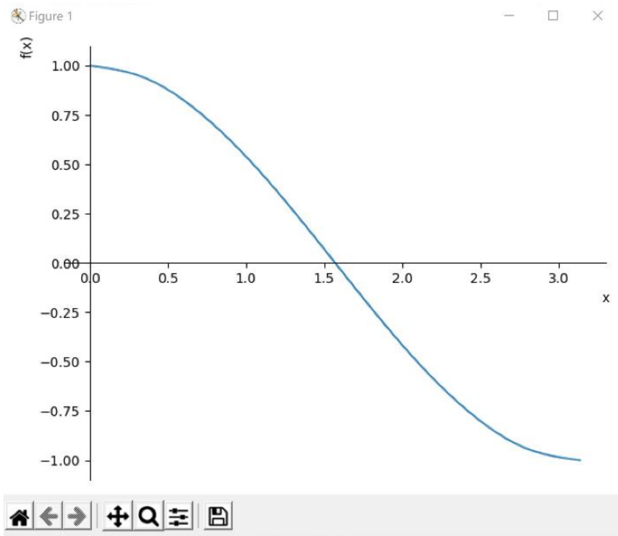
求解后得到全部$(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2 \ldots a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1})$返回。

最后定义 equation(x)函数，将a_i, b_i, c_i, d_i组合成$f_i(t) = a_i t^3 + b_i t^2 + c_i t + d_i$，返回一个列表，其中存储了(n-1)个函数。

这样将 Newton_formula(x, y)函数和 polynomial(x, b)函数组合起来可实现 Newton interpolation。将 spline(x, y)函数和 equation(x)函数组合起来可实现 cubic spline interpolation。test1()函数对比了两种方法对 cos(x)在$[0, \pi]$范围内进行插值得到的结果；test2()函数对比了两种方法对$\frac{1}{1+25x^2}$在$[-1,1]$范围内进行插值得到的结果。

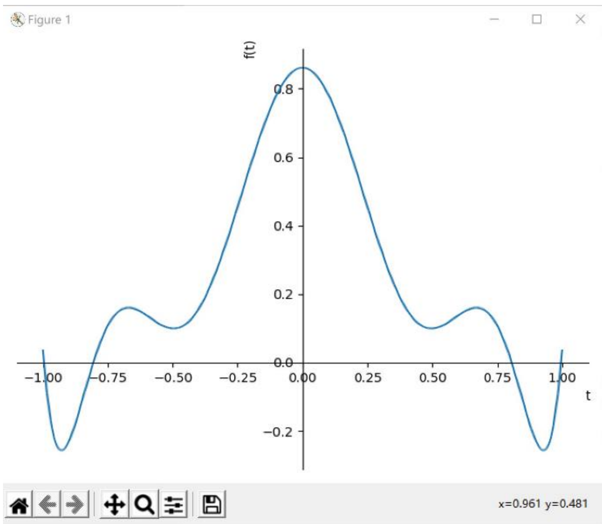
cos(x)在$[0, \pi]$范围内的结果（左图为 Newton interpolation，右图为 cubic spline interpolation）：

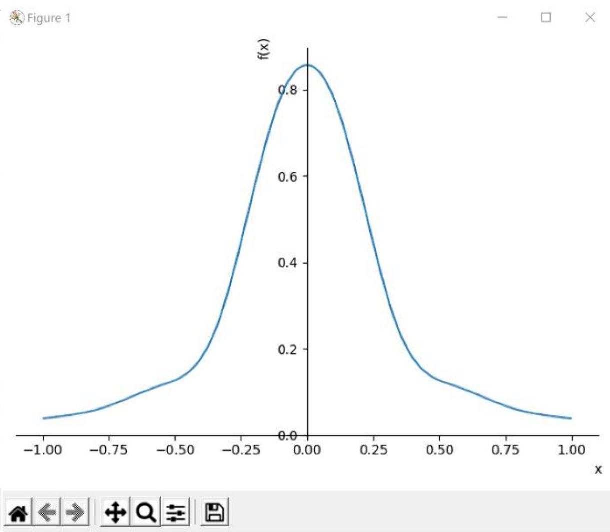




结果从函数图像上来看并无太大不同，都与 cos(x)符合较好。

$\frac{1}{1+25x^2}$在$[-1,1]$范围内的结果（左图为 Newton interpolation，右图为 cubic spline interpolation）：





可以看到，两种插值方法结果差别较大，cubic spline interpolation 更好。

Input: $(x_0, y_0), (x_1, y_1) \ldots (x_{n-1}, y_{n-1})$

Output: Newton interpolation of points $f(t) = b_0 + b_1(t - x_0) + b_2(t - x_0)(t - x_1) + \cdots$

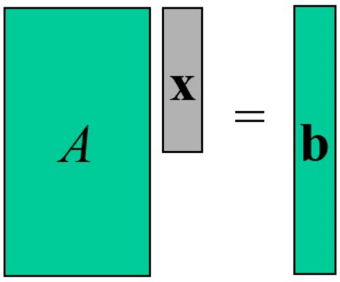
1. **Function** Newton_formula(x, y)
2. $n \leftarrow \dim(x)$
3. $mat \leftarrow \begin{pmatrix} x_0 & f(x_0) & \ldots & 0 \\ x_1 & f(x_1) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1} & f(x_{n-1}) & \ldots & 0 \end{pmatrix}$
4. **For** $j \leftarrow 2$ to n **Do**
5. **For** $k \leftarrow (j-1)$ to $(n-1)$ **Do**
6. $mat[k,j] \leftarrow \frac{mat[k,j-1] - mat[k-1,j-1]}{mat[k,0] - mat[k-j+1,0]}$
7. $result \leftarrow (0,0 \ldots 0)$
8. **For** $l \leftarrow 0$ to $(n-1)$ **Do**
9. $result[l] \leftarrow mat[l, l+1]$ /*$result = (b_0, b_1, \ldots, b_n)$*/
10. **Return** $result$

spline(x, y)函数内容就是用矩阵解$4(n-1)$个未知数的方程组，故省略这一部分的伪代码。

拟合的方程为：$\frac{PV}{RT} = 1 + \frac{a}{V} + \frac{b}{V^2}$

转化为：$P = RT(\frac{1}{V} + \frac{a}{V^2} + \frac{b}{V^3})$，即：$y = 8.314 \times 303 \times \left(\frac{1}{x} + \frac{a}{x^2} + \frac{b}{x^3}\right) = C\left(\frac{1}{x} + \frac{a}{x^2} + \frac{b}{x^3}\right)$

待解的矩阵为：$Ax = b\ \rightarrow$



$$\begin{pmatrix} \frac{C}{{x_1}^2} & \frac{C}{{x_1}^3} \\ \frac{C}{{x_2}^2} & \frac{C}{{x_2}^3} \\ \frac{C}{{x_3}^2} & \frac{C}{{x_3}^3} \\ \frac{C}{{x_4}^2} & \frac{C}{{x_4}^3} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 - \frac{C}{x_1} \\ y_2 - \frac{C}{x_2} \\ y_3 - \frac{C}{x_3} \\ y_4 - \frac{C}{x_4} \end{pmatrix}$$

奇异值分解矩阵：$A = U\Sigma V^T$

即可得到：$x = V\Sigma^{-1}U^Tb$，其中：$\Sigma^{-1} = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 \end{pmatrix}$

源代码中利用了两种方法来完成最小二乘法拟合，分别是奇异值分解 SVD 方法和 SciPy 库自带的 leastsq()函数。

func()定义了$f(x) = 8.314 \times 303 \times \left(\frac{1}{x} + \frac{a}{x^2} + \frac{b}{x^3}\right)$，error()定义了拟合的残差。

```
def func(p, x):
# The function that needs to be fitted
    a, b = p
    fx = (8.314 * 303 * ((1/x) + (a/(x**2) + (b/(x**3)))))
    return fx

def error(p, x, y):
# The residual error of the fitting
    return func(p,x) - y
```

SVD(x,y)函数将$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$和$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$作为输入，通过 for 循环将元素填入 A 矩阵和 b 矩阵，使得：

```
def SVD(x, y):
# Solving least-squares system with SVD
    m = len(x)
    A = np.zeros([m,2])
    b = np.zeros(m)
    mat = np.zeros([2,m])

    for i in range(m):
        A[i][0] = (8.314*303) / (x[i]**2)
        A[i][1] = (8.314*303) / (x[i]**3)
        b[i] = y[i] - (8.314*303) / x[i]

    U, sigma, V = np.linalg.svd(A)
    mat[0][0] = 1 / sigma[0]
    mat[1][1] = 1 / sigma[1]
    solution = np.dot(np.dot(np.dot(V.T,mat),U.T),b)

    return solution
```

$$A = \begin{pmatrix} \frac{C}{{x_1}^2} & \frac{C}{{x_1}^3} \\ \frac{C}{{x_2}^2} & \frac{C}{{x_2}^3} \\ \frac{C}{{x_3}^2} & \frac{C}{{x_3}^3} \\ \frac{C}{{x_4}^2} & \frac{C}{{x_4}^3} \end{pmatrix}, \qquad b = \begin{pmatrix} y_1 - \frac{C}{x_1} \\ y_2 - \frac{C}{x_2} \\ y_3 - \frac{C}{x_3} \\ y_4 - \frac{C}{x_4} \end{pmatrix}$$

使用 numpy.linalg.svd()函数将 A 矩阵奇异值分解，并将分解得到的U矩阵赋值给 U，V^T矩阵赋值给 V，奇异值σ_1, σ_2存入 sigma。最后完成矩阵乘法：

$$x = V \begin{pmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 \end{pmatrix} U^T b$$

并将结果存入 solution 并返回，solution[0]即为 a，solution[1]即为 b。

在 main()函数中对比了奇异值分解 SVD 方法和 SciPy 库的 leastsq()方法得到的结果，将结果输出，得到：

```
国际单位制下: a=37413.502527, b=-342760.322904
国际单位制下，奇异值分解得到:  a=37413.502652, b=-342760.324984
未转换单位时: a=143672.636462, b=-10181.462224
未转换单位时，奇异值分解得到:  a=143672.636285, b=-10181.462372
```

输入 P，V 时采用了国际单位制和未转换单位制的两种情况，可以看到结果非常接近，拟合结果良好。

Input: $x = (x_1, x_2, x_3, x_4), y = (y_1, y_2, y_3, y_4), f(x) = 8.314 \times 303 \times \left(\frac{1}{x} + \frac{a}{x^2} + \frac{b}{x^3}\right)$

Output: The value of a and b

Function SVD(x, y)

$m \leftarrow$ dim (x)

$A \leftarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, B \leftarrow \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, mat \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

For $i \leftarrow$ 0 to (m − 1) **Do**

    $A[i, 0] \leftarrow \frac{8.314 \times 303}{x[i]^2}$

    $A[i, 1] \leftarrow \frac{8.314 \times 303}{x[i]^3}$

17. $b[i] \leftarrow \mathrm{y[i]} - \frac{8.314\times303}{x[i]}$ /* $A\leftarrow\begin{pmatrix} \frac{C}{{x_1}^2} & \frac{C}{{x_1}^3} \\ \frac{C}{{x_2}^2} & \frac{C}{{x_2}^3} \\ \frac{C}{{x_3}^2} & \frac{C}{{x_3}^3} \\ \frac{C}{{x_4}^2} & \frac{C}{{x_4}^3} \end{pmatrix}, b\leftarrow\begin{pmatrix} y_1 - \frac{C}{x_1} \\ y_2 - \frac{C}{x_2} \\ y_3 - \frac{C}{x_3} \\ y_4 - \frac{C}{x_4} \end{pmatrix}$ */

18. $U, \sum, V \leftarrow \mathrm{SVD(A)}$ /* $A = U\sum V$ */

19. $\binom{a}{b} \leftarrow V^T\sum^{-1}U^T b$

20. **Return** $\binom{a}{b}$