科学与工程数值算法系列丛书

科学与工程数值算法
(Visual Basic 版)

周长发　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

本书介绍了在科学与工程实际工作中常用的数值计算算法原理和 Visual Basic 编程方法。本书包括复数运算、矩阵运算、线性代数方程组的求解、非线性方程与方程组的求解、插值和数值积分等 6 章，涉及使用频率非常高的近 90 个基本算法。在介绍每一算法原理的基础上，讨论了算法的 Visual Basic 编程方法，并用常用的数据介绍了算法程序的调用方式。

本书适合科学与工程数值计算工作的科研人员、工程技术人员、管理人员以及大专院校相关专业的师生参考阅读。

图书在版编目(CIP)数据

科学与工程数值算法：Visual Basic 版/周长发编著.北京：清华大学出版社，2002
ISBN 7-302-06010-X

I.科… II.周… III.BASIC 语言－数值计算－计算方法 IV.TP312

中国版本图书馆 CIP 数核字(2002)第 084401 号

出 版 者： 清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www.tup.com.cn
http://www.tup.tsinghua.edu.cn
责任编辑： 张彦青
印 刷 者：
发 行 者： 新华书店总店北京发行所
开 本： 787×1092 1/16 印张：19.5 字数：462 千字
版 次： 2002 年 11 月第 1 版 2002 年 11 月第 1 次印刷
书 号： ISBN 7-302-06010-X/TP・3587
印 数： 0001～5000
定 价： 32.00 元

前　言

数值计算是科学研究和实际工程中的重要技术手段，而数值计算中使用最频繁的是一些数值计算的经典算法。Visual Basic 是在 Windows 系列操作系统中一种流行的编程语言，具有可视化、交互式和简洁性等特征，具有强大简便的数据库访问能力，大大简化了程序设计的复杂度，从根本上改变了传统的程序设计模式。因此 Visual Basic 在科学与工程实践中得到广泛的使用。本书的目标就是使用流行的 Visual Basic 语言来描述和实现经典的数值计算算法。

读者对象

如果您需要使用 Visual Basic 进行数值计算方面的编程，那么本书可能就是您寻觅已久的助手。

本书假定读者具有一定的 Visual Basic 编程经验。如果你还不熟悉如何用 Visual Basic 来创建工程，我建议您在阅读一本 Visual Basic 编程的教科书之后再阅读本书。

内容结构

本书介绍了近 90 个实用经典算法，每一算法都独立成节。每节都包括算法原理、算法实现和示例 3 部分。算法原理部分分别讨论了每一算法的计算原理；算法实现部分讨论了用 Visual Basic 实现算法的技巧，并给出了完整的算法函数源程序；示例部分介绍了算法函数的调用方式，给出了调用算法的示例源程序、验证算法的示例数据和运算结果。本书所有源程序都可以从 www.wenyuan.com.cn 网站下载。

本书包括复数运算、矩阵运算、线性方程组的求解、非线性方程与方程组的求解、插值、数值积分等 6 章。各章都是在介绍算法的基础上用 Visual Basic 实现其相关算法，其中包括如下内容：

第 1 章：复数运算，本章介绍了与复数相关运算的算法。

第 2 章：矩阵运算，本章在介绍各种矩阵运算原理的基础上实现了矩阵基础运算。

第 3 章：线性代数方程组的求解，本章分析了线性代数方程组的常用求解算法。

第 4 章：非线性方程与方程组的求解，本章介绍了非线性方程和方程组的求解方法。

第 5 章：插值，本章介绍并用 Visual Basic 实现了常用的插值算法。

第 6 章：数值积分，本章介绍了常用的数值积分算法。

致谢

本书的成稿是许多人努力的结果。在此，我要感谢所有对本书的写作有过帮助的人：周建欣、林琳、黄家辉、辛平、邓优、李秀鹃、刘烁和何西涛等参加了本书有关内容的讨论和资料收集；周建欣、彭文、黄娅丽测试了本书的所有示例程序；何晓、李丽帮助输入了部分文稿。

我要感谢清华大学出版社所有对本书的出版付出过劳动的人，特别是彭欣女士，她为本书的策划和组织付出了大量的劳动并提供了很多很好的建议。

限于笔者的能力，本书中错误之处在所难免，请读者批评指正。

作者

2002 年 8 月 19 日

目　　录

第1章　复 数 运 算

本章将介绍复数的运算算法，包括复数的乘法、除法、模、根、实幂指数、复幂指数、自然对数、正弦、余弦和正切的计算方法。

在 Visual Basic 的数据类型中没有定义复数类型，因此需要用 Type 来定义自己的复数类型。对于复数 $x + \mathrm{j}y$，定义复数类型 Complex 为：

```
Public Type Complex
    x as Double          ' 复数实部
    y as Double          ' 复数虚部
End Type
```

其中 x 为复数的实部，y 为复数的虚部。

1.1　复 数 乘 法

1. 算法原理

对复数 $z_1 = a + \mathrm{j}b$，$z_2 = c + \mathrm{j}d$，$z_1 \times z_2$ 可用如下公式计算：

$$z_1 \times z_2 = (a + \mathrm{j}b)(c + \mathrm{j}d) = ac - bd + \mathrm{j}(ad + bc)$$

2. 算法实现

根据上述算法，可以定义计算复数乘法的 Visual Basic 函数 CMul，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  功能：  复数的运算
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' 定义复数类型
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Type Complex
   x As Double     ' 复数实部
   y As Double     ' 复数虚部
End Type

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CMul
'  功能：  计算复数的乘法
'  参数：  cpxZ1   - Complex型变量
`          cpxZ2   - Complex型变量
```

```
' 返回值：Complex型，乘积
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CMul(cpxZ1 As Complex, cpxZ2 As Complex) As Complex
    CMul.x = cpxZ1.x * cpxZ2.x - cpxZ1.y * cpxZ2.y
    CMul.y = cpxZ1.x * cpxZ2.y + cpxZ1.y * cpxZ2.x
End Function
```

3. 示例

调用函数 CMul **来求解** (9 + 11j) × (56+3j)**，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

    ' 乘法
    Dim cpxZ1 As Complex, cpxZ2 As Complex
    Dim sComplex1 As String, sComplex2 As String
    cpxZ1.x = 9
    cpxZ1.y = 11
    cpxZ2.x = 56
    cpxZ2.y = 3
    sComplex1 = "复数 " & cpxZ1.x & " + " & cpxZ1.y & "j"
    sComplex2 = "复数 " & cpxZ2.x & " + " & cpxZ2.y & "j"

    cpxZ = CMul(cpxZ1, cpxZ2)
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"

     sMsg = sComplex1 & "乘以" & sComplex2 & " 的积为 " & sComplex
     MsgBox sMsg

End Sub
```

其输出结果如图 1.1 **所示。**

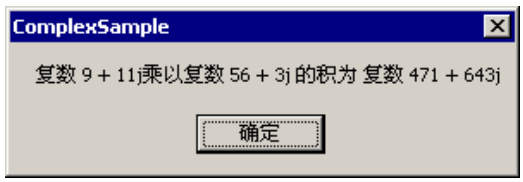

图 1.1　程序输出结果

1.2　复 数 除 法

1. 算法原理

对复数 $z_1 = a + \mathrm{j}b$**，** $z_2 = c + \mathrm{j}d$**，** z_1 **除以** z_2 **可用如下公式计算：**

$$\frac{z_1}{z_2}=\frac{a+\mathrm{j}b}{c+\mathrm{j}d}=\frac{(ac+bd)+\mathrm{j}(bc-ad)}{c^2+d^2}=\begin{cases}\dfrac{\left(a+b\dfrac{d}{c}\right)+\mathrm{j}\left(b-a\dfrac{d}{c}\right)}{c+d\dfrac{d}{c}}, & |c|\geq|d| \\ \dfrac{\left(a\dfrac{c}{d}+b\right)+\mathrm{j}\left(b\dfrac{c}{d}-a\right)}{c\dfrac{c}{d}+d} & |c|<|d|\end{cases}$$

后面的等式用于防止上溢和下溢。

2. 算法实现

根据上述算法，可以定义计算复数除法的 Visual Basic 函数 CDiv，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CDiv
'  功能：  计算复数的除法
'  参数：  cpxZ1  - Complex型变量，被除复数
`          cpxZ2  - Complex型变量，除数
'  返回值：Complex型，商
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CDiv(cpxZ1 As Complex, cpxZ2 As Complex) As Complex
    ' 局部变量
    Dim e As Double, f As Double

    If Abs(cpxZ2.x) >= Abs(cpxZ2.y) Then
        e = cpxZ2.y / cpxZ2.x
        f = cpxZ2.x + e * cpxZ2.y

        CDiv.x = (cpxZ1.x + cpxZ1.y * e) / f
        CDiv.y = (cpxZ1.y - cpxZ1.x * e) / f
    Else
        e = cpxZ2.x / cpxZ2.y
        f = cpxZ2.y + e * cpxZ2.x

        CDiv.x = (cpxZ1.x * e + cpxZ1.y) / f
        CDiv.y = (cpxZ1.y * e - cpxZ1.x) / f
    End If

End Function
```

3. 示例

调用函数 CDiv 来求解 (4 + 8j)/(7+9j)，程序代码如下：

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String
```

```
    ' 除法
    Dim cpxZ1 As Complex, cpxZ2 As Complex
    Dim sComplex1 As String, sComplex2 As String
    cpxZ1.x = 4
    cpxZ1.y = 8
    cpxZ2.x = 7
    cpxZ2.y = 9
    sComplex1 = "复数 " & cpxZ1.x & " + " & cpxZ1.y & "j"
    sComplex2 = "复数 " & cpxZ2.x & " + " & cpxZ2.y & "j"

    cpxZ = CDiv(cpxZ1, cpxZ2)
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
    sMsg = sComplex1 & "除以" & sComplex2 & " 的商为 " & sComplex
    MsgBox sMsg

End Sub
```

其输出结果如图 1.2 所示。

图 1.2 程序输出结果

1.3 复 数 的 模

1. 算法原理

对复数 $z = x + \mathrm{j}y$，其模 $|z|$ 定义如下：

$$|z| = \sqrt{x^2 + y^2} = \begin{cases} |x|\sqrt{1+\left(\dfrac{y}{x}\right)^2}, & |x| > |y| \\ |y|\sqrt{1+\left(\dfrac{x}{y}\right)^2}, & |x| \le |y| \end{cases}$$

后面的等式用于防止上溢和下溢。

2. 算法实现

根据上述算法，可以定义计算复数的模的 Visual Basic 函数 CAbs，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'   模块名：ComplexModule.bas
'   函数名：CAbs
'   功能：   计算复数的模
'   参数：   cpxZ   - Complex型变量，用于求模的复数
'   返回值：Double型，指定复数的模
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Function CAbs(cpxZ As Complex) As Double
    ' 特殊值处理
    If cpxZ.x = 0 Then
        CAbs = cpxZ.y
        Exit Function
    End If
    If cpxZ.y = 0 Then
        CAbs = cpxZ.x
        Exit Function
    End If

    '求取实部和虚部的绝对值
    cpxZ.x = Abs(cpxZ.x)
    cpxZ.y = Abs(cpxZ.y)

    ' 计算模
    If cpxZ.x > cpxZ.y Then
        CAbs = cpxZ.x * Sqr(1 + (cpxZ.y / cpxZ.x) * (cpxZ.y / cpxZ.x))
        Exit Function
    End If

    CAbs = cpxZ.y * Sqr(1 + (cpxZ.x / cpxZ.y) * (cpxZ.x / cpxZ.y))

End Function
```

3. 示例

调用函数 CAbs **来求解复数** 55.6 + 68.2j **的模，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim dblM As Double

    ' 定义复数
    cpxZ.x = 55.6
    cpxZ.y = 68.2

    ' 求模
    dblM = CAbs(cpxZ)

    MsgBox "复数 " & cpxZ.x & " + " & cpxZ.y & "j 的模为 " & dblM
End Sub
```

其输出结果如图 1.3 **所示。**

图 1.3　运行结果

1.4 复 数 的 根

1. 算法原理

对复数 $z = x + \mathrm{j}y$，其全部 n 次根的计算方法如下：

$$
\begin{aligned}
z^{\frac{1}{n}} &= (x+\mathrm{j}y)^{\frac{1}{n}} = [r(\cos\phi + \mathrm{j}\sin\phi)]^{\frac{1}{n}} \\
&= r^{\frac{1}{n}}\left(\cos\frac{\phi+2k\pi}{n} + \mathrm{j}\sin\frac{\phi+2k\pi}{n}\right) \\
&= r^{\frac{1}{n}}(\cos(\theta+k\omega) + \mathrm{j}\sin(\theta+k\omega)) \\
&= r^{\frac{1}{n}}\cos(\theta+k\omega) + \mathrm{j}r^{\frac{1}{n}}\sin(\theta+k\omega) \\
&= r^{\frac{1}{n}}(\cos\theta\cos k\omega - \sin\theta\cos k\omega) + \mathrm{j}r^{\frac{1}{n}}(\sin\theta\cos k\omega + \cos\theta\sin k\omega)
\end{aligned}
$$

其中，

$k = 0,\ 1,\ \ldots,\ n-1$

$$\omega = \frac{2\pi}{n}$$

$$r = \sqrt{x^2 + y^2}$$

$$
\theta = \begin{cases}
0 & a > 0,\ b = 0 \\
\dfrac{\pi}{n} & a < 0,\ b = 0 \\
\dfrac{\pi}{2n} & a = 0,\ b > 0 \\
-\dfrac{\pi}{2n} & a = 0,\ b < 0 \\
\dfrac{\arctan(\frac{b}{a})}{n} & a > 0,\ b \neq 0 \\
\dfrac{\arctan(\frac{b}{a}) + \pi}{n} & a < 0,\ b \neq 0
\end{cases}
$$

因此复数 $z = x + \mathrm{j}y$ 的全部 n 次根可表示为：

$$x_k + \mathrm{j}y_k, k = 0,\ 1,\ \ldots,\ ,\ n-1$$

其中，

$$x_k = r^{\frac{1}{n}}(\cos\theta\cos k\omega - \sin\theta\cos k\omega)$$

$$y_k = r^{\frac{1}{n}}(\sin\theta\cos k\omega + \cos\theta\sin k\omega)$$

2. 算法实现

根据上述方法，可以定义计算复数的根的 Visual Basic **函数** CRoot。**为了减少正弦和余弦的运算量，**CRoot **函数使用了如下递推公式：**

$$\begin{cases}\sin k\omega = \sin\,(k-1)\omega\cos\,\omega + \cos\,(k-1)\omega\sin\,\omega \\ \cos k\omega = \cos\,(k-1)\omega\cos\,\omega - \sin\,(k-1)\omega\sin\,\omega\end{cases}$$

CRoot **函数其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CRoot
'  功能：  计算复数的根
'  参数：  cpxZ   - Complex型变量，用于求根的复数
'          n      - Integer型变量，复数的根次
'          cpxZR  - Complex型一维数组，用于存放求得的复数的根
'  返回值：Integer型，复数的根的个数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CRoot(cpxZ As Complex, n As Integer, cpxZR() As Complex) As Integer
    ' 局部变量
    Dim i As Integer
    Dim r As Double
    Dim c As Double, c1 As Double, ck As Double, cs As Double
    Dim s As Double, s1 As Double, sk As Double, sc As Double

    ' 特殊值处理
    If cpxZ.y = 0 Then
        r = Exp(Log(Abs(cpxZ.x)) / n)
        cs = 0
    Else
        If cpxZ.x = 0 Then
            r = Exp(Log(Abs(cpxZ.y)) / n)
            If cpxZ.y > 0 Then
                cs = 1.5707963268
            Else
                cs = -1.5707963268
            End If
        Else
            r = Exp(Log(cpxZ.x * cpxZ.x + cpxZ.y * cpxZ.y) / n / 2)
            cs = Atn(cpxZ.y / cpxZ.x)
        End If
    End If

    If cpxZ.x < 0 Then
        cs = cs + PI
    End If

    ' 中间值，用于提高运算速度
    cs = cs / n
    c = Cos(cs)
    s = Sin(cs)

    sc = 2 * PI / n
    c1 = Cos(sc)
```

```
    s1 = Sin(sc)

    ck = 1
    sk = 0

    ' 第一个根
    cpxZR(1).x = c * r
    cpxZR(1).y = s * r

    ' 其余n-1个根
    For i = 2 To n
        cs = ck * c1 - sk * s1
        sc = sk * c1 + ck * s1

        ' 递推公式
        cpxZR(i).x = (c * cs - s * sc) * r
        cpxZR(i).y = (s * cs - c * sc) * r

        ck = cs
        sk = sc
    Next i

    ' 共有n个根
    CRoot = n

End Function
```

3. 示例

下面的代码调用函数 CRoot **求解复数** 16 + 81j **的** 4 **次根：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim i As Integer, n As Integer
    Dim sComplex As String, sMsg As String
    Dim cpxZR() As Complex

    ' 定义复数
    cpxZ.x = 16
    cpxZ.y = 81
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"

    ' 求根
    n = 4
    ReDim cpxZR(n) As Complex
    n = CRoot(cpxZ, n, cpxZR)

    sMsg = sComplex & " 的 " & n & " 个 " & n & " 次根为:" & Chr(13) & Chr(13)
    For i = 1 To n
        sMsg = sMsg & i & ": " & cpxZR(i).x & " + " & cpxZR(i).y & "j" & Chr(13)
    Next i
```

```
    MsgBox sMsg

End Sub
```

其输出结果如图 1.4 **所示。**

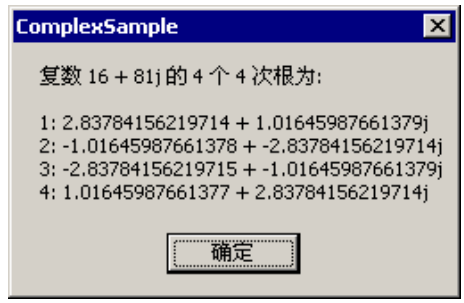

图 1.4　程序输出结果

1.5　复数的实幂指数

1. 算法原理

复数 $z = x + \mathrm{j}y$ **的** w(**为实数**)**次幂的计算方法如下：**

$$z^w = (x + \mathrm{j}y)^w = [r(\cos\phi + \mathrm{j}\sin\phi)]^w = r^w(\cos w\phi + \mathrm{j}\sin w\phi)$$

其中 r **为复数** $z = x + \mathrm{j}y$ **的模，** ϕ **为复数** $z = x + \mathrm{j}y$ **的幅角。**

2. 算法实现

根据上述方法，可以定义计算复数的实幂指数的 Visual Basic **函数** CPow**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CPow
'  功能：  计算复数的实幂指数
'  参数：  cpxZ   - Complex型变量，用于求实幂指数的复数
'          w      - Double型变量，复数的实幂指数幂次
'  返回值：Complex型变量，为求得的复数的实幂指数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CPow(cpxZ As Complex, w As Double) As Complex
    ' 局部变量
    Dim r As Double, t As Double

     ' 特殊值处理
    If cpxZ.x = 0 And cpxZ.y = 0 Then
       CPow.x = 0
       CPow.y = 0
       Exit Function
    End If

    ' 幂运算公式中的三角函数运算
```

```
    If cpxZ.x = 0 Then
       If cpxZ.y > 0 Then
          t = 1.5707963268
       Else
          t = -1.5707963268
       End If

    Else
       If cpxZ.x > 0 Then
          t = Atn(cpxZ.y / cpxZ.x)
       Else
          If cpxZ.y >= 0 Then
             t = Atn(cpxZ.y / cpxZ.x) + PI
          Else
             t = Atn(cpxZ.y / cpxZ.x) - PI
          End If
       End If
    End If

    ' 模的幂
    r = Exp(w * Log(Sqr(cpxZ.x * cpxZ.x + cpxZ.y * cpxZ.y)))

    ' 复数的实幂指数
    CPow.x = r * Cos(w * t)
    CPow.y = r * Sin(w * t)

End Function
```

3. 示例

调用函数 CPow **求解复数** 16 + 81j **的** 4 **次幂，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim n As Integer
    Dim sComplex As String, sMsg As String

    ' 定义复数
    cpxZ.x = 16
    cpxZ.y = 81
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"

    ' 求实幂指数
    Dim cpxPow As Complex
    n = 3
    cpxPow = CPow(cpxZ, 3)
    sMsg = sComplex & " 的 " & n & " 次幂为:" & cpxPow.x & " + " & _
           cpxPow.y & "j"
    MsgBox sMsg

End Sub
```

其输出结果如图 1.5 所示。

图 1.5　程序输出结果

1.6　复数的复幂指数

1. 算法原理

复数 $z=x+\mathrm{j}y$ 的 $c+\mathrm{j}d$ 次幂的计算方法如下：

$$
\begin{aligned}
z^{(c+\mathrm{j}d)} &= (x+\mathrm{j}y)^{(c+\mathrm{j}d)} \\
&= \mathrm{e}^{(c+\mathrm{j}d)\ln(x+\mathrm{j}y)} \\
&= \mathrm{e}^{(c+\mathrm{j}d)(\ln(x+\mathrm{j}y)+2n\pi(c+\mathrm{j}d))} \\
&= \mathrm{e}^{c\ln|x+\mathrm{j}y|-d(\arg u+2n\pi)}(\cos\ v+\sin\ v)
\end{aligned}
$$

其中，

$$v=d\ln|x+\mathrm{j}y|+c(\arg\ u+2n\pi)$$

$$
\arg u=\begin{cases}
\dfrac{b}{|b|}\cdot\dfrac{\pi}{2} & a=0 \\
\arctan\left(\dfrac{b}{a}\right) & a>0 \\
\arctan\left(\dfrac{b}{a}\right)+\pi & a<0,\ b\geq 0 \\
\arctan\left(\dfrac{b}{a}\right)-\pi & a<0,\ b<0
\end{cases}
$$

当 $n=0$ 时，可以求得复数复幂指数的主值。

2. 算法实现

根据上述方法，可以定义计算复数的复幂指数的 Visual Basic 函数 CCPow，代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CCPow
'  功能：  计算复数的复幂指数
'  参数：  cpxZ   - Complex型变量，用于求复幂指数的复数
'          cpxN   - Complex型变量，复数的复幂指数幂次
'          n      - Integer型变量，2*PI的整数倍数
'  返回值：Complex型变量，为求得的复数的复幂指数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CCPow(cpxZ As Complex, cpxN As Complex, n As Integer) As Complex
```

```
    ' 局部变量
    Dim r As Double, s As Double, u As Double, v As Double

    ' 特殊值处理
    If cpxZ.x = 0 Then
       If cpxZ.y = 0 Then
          CCPow.x = 0
          CCPow.y = 0
          Exit Function
       End If

       s = 1.5707963268 * (Abs(cpxZ.y) / cpxZ.y + 4 * n)
    Else
       s = 2 * PI * n + Atn(cpxZ.y / cpxZ.x)

       If cpxZ.x < 0 Then
          If cpxZ.y > 0 Then
             s = s + PI
          Else
             s = s - PI
          End If
       End If
    End If

    ' 求幂运算公式
    r = 0.5 * Log(cpxZ.x * cpxZ.x + cpxZ.y * cpxZ.y)
    v = cpxN.y * r + cpxN.x * s
    u = Exp(cpxN.x * r - cpxN.y * s)
    CCPow.x = u * Cos(v)
    CCPow.y = u * Sin(v)

End Function
```

3. 示例

调用函数 CCPow **求解复数** 1 + j **的** 1 + j **次幂，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

    ' 求复幂指数
    cpxZ.x = 1
    cpxZ.y = 1
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
    Dim cpxCPow As Complex
    Dim cpxN As Complex
    cpxN.x = 1
    cpxN.y = 1
    cpxCPow = CCPow(cpxZ, cpxN, 0)
    sMsg = sComplex & " 的 " & cpxN.x & " + " & cpxN.y & "j" & _
```

```
                " 次幂为:" & cpxCPow.x & " + " & cpxCPow.y & "j"
        MsgBox sMsg

    End Sub
```

示例中 n 值取为 0，求得的结果为复幂指数的主值。其输出结果如图 1.6 所示。

图 1.6　程序输出结果

1.7　复数的自然对数

1. 算法原理

复数 $z = x + \mathrm{j}y$ 的自然对数的计算方法如下：

$$\ln z = \ln|z| + \mathrm{j}\arg z$$

其中，

$$\ln|z| = \ln\sqrt{x^2 + y^2}$$

$$= \begin{cases} \dfrac{1}{2}\left[\ln(|2a|+|2b|) + \ln\left(8\dfrac{a}{|2a|+|2b|}a + 8\dfrac{b}{|2a|+|2b|}b\right)\right] - \ln\sqrt{8} & |a|<1,\ \ |b|<1 \\ \dfrac{1}{2}\left[\ln(\dfrac{|a|}{4}) + \ln\left(\dfrac{1}{2}\dfrac{\frac{a}{2}}{\frac{|a|}{4}+\frac{|b|}{4}}\dfrac{a}{2} + \dfrac{1}{2}\dfrac{\frac{b}{2}}{\frac{|a|}{4}+\frac{|b|}{4}}\dfrac{b}{2}\right)\right] + \ln\sqrt{8} & |a|\geq 1,\ |b|\geq 1 \end{cases}$$

$$\arg z = \begin{cases} \arctan(\dfrac{b}{a}) & a>0,\ |a|\geq b \\ \arctan(\dfrac{b}{a}) + \pi & a<0,\ b\geq 0,\ \ |a|\geq b \\ \arctan(\dfrac{b}{a}) - \pi & a<0,\ \ b<0 \\ -\arctan(\dfrac{b}{a}) + \dfrac{b}{|b|}\dfrac{\pi}{2} & |a|<|b| \text{ 或 } a=0 \end{cases}$$

2. 算法实现

根据上述方法，可以定义复数的自然对数的 Visual Basic 函数 CLn，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CLn
'  功能：  计算复数的自然对数
'  参数：  cpxZ   - Complex型变量，用于求自然对数的复数
```

```
' 返回值：Complex型变量，为求得的复数的自然对数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CLn(cpxZ As Complex) As Complex
    ' 局部变量
    Dim e As Double, f As Double

     ' 特殊值处理
    If cpxZ.x = 0 And cpxZ.y = 0 Then
       CLn.x = 0
       CLn.y = 0
       Exit Function
    End If

    ' 根据不同的情况计算复数的自然对数
    If Abs(cpxZ.x) < 1 And Abs(cpxZ.y) < 1 Then
       CLn.x = Abs(cpxZ.x + cpxZ.x) + Abs(cpxZ.y + cpxZ.y)
       CLn.y = 8 * cpxZ.x / CLn.x * cpxZ.x + 8 * cpxZ.y / CLn.x * cpxZ.y
    Else
       e = 0.5 * cpxZ.x
       f = 0.5 * cpxZ.y
       CLn.x = Abs(0.5 * e) + Abs(0.5 * f)
       CLn.y = 0.5 * e / CLn.x * e + 0.5 * f / CLn.x * f
       CLn.x = 0.5 * (Log(CLn.x) + Log(CLn.y)) + 1.03972077084
    End If

    If cpxZ.x <> 0 And Abs(cpxZ.x) >= Abs(cpxZ.y) Then
       If cpxZ.x >= 0 Then
          CLn.y = Atn(cpxZ.y / cpxZ.x)
       Else
          If cpxZ.y >= 0 Then
             CLn.y = Atn(cpxZ.y / cpxZ.x) + PI
          Else
             CLn.y = Atn(cpxZ.y / cpxZ.x) - PI
          End If
       End If
    Else
       CLn.y = -Atn(cpxZ.x / cpxZ.y) + PI / 2 * cpxZ.y / Abs(cpxZ.y)
    End If

End Function
```

3. 示例

调用函数 CLn **求解复数** 3 +2 j **的自然对数，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

    ' 求自然对数
    cpxZ.x = 3
    cpxZ.y = 2
```

```
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
    Dim cpxLn As Complex
    cpxLn = CLn(cpxZ)
    sMsg = sComplex & " 的自然对数为:" & cpxLn.x & " + " & cpxLn.y & "j"
    MsgBox sMsg

End Sub
```

输出结果如图 1.7 所示。

图 1.7　程序输出结果

1.8　复数的正弦

1. 算法原理

复数 $z = x + \mathrm{j}y$ 的正弦的计算方法如下：

$$\sin z = \sin\,(x + \mathrm{j}y) = \sin\,x\cosh y + \mathrm{j}\cos\,x\sinh y$$

其中，

$$\cosh y = \frac{\mathrm{e}^{y} + \mathrm{e}^{-y}}{2}$$

$$\sinh y = \frac{\mathrm{e}^{y} - \mathrm{e}^{-y}}{2}$$

当 $|y| < 1$ 时，为了避免有效数字的损失，sinh 改用切比雪夫公式计算，即

$$\sinh y = y[-c_5 y_1^5 + (c_5 - c_4)y_1^4 + (4c_5 + c_4 - c_3)y_1^3 + (-3c_5 + c_4 + c_3 - c_2)y_1^2 + (-3c_5 - 2c_4 + 2c_3 c_2 - c_1)y_1 + (c_5 - c_4 - c_3 + c_2 + c_1 - c_0)$$

其中，

$c_0 = 1.13031820798497$

$c_1 = 0.04433684984866$

$c_2 = 0.00054292631191$

$c_3 = 0.00000319843646$

$c_4 = 0.00000001103607$

$c_5 = 0.00000000002498$

$y_1 = 2(2y^2 - 1)$

2. 算法实现

根据上述方法，可以定义复数的正弦的 Visual Basic 函数 CSin，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CSin
'  功能：  计算复数的正弦
'  参数：  cpxZ   - Complex型变量，用于求正弦的复数
'  返回值：Complex型变量，为求得的复数的正弦
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CSin(cpxZ As Complex) As Complex
    ' 局部变量
    Dim i As Integer
    Dim y1 As Double, br As Double, b1 As Double, b2 As Double
    Dim c(6) As Double

    ' 切比雪夫公式的常数系数
    c(1) = 1.13031820798497
    c(2) = 0.04433684984866
    c(3) = 0.00054292631191
    c(4) = 0.00000319843646
    c(5) = 0.00000001103607
    c(6) = 0.00000000002498

    y1 = Exp(cpxZ.y)
    CSin.x = 0.5 * (y1 + 1 / y1)
    If Abs(cpxZ.y) >= 1 Then
        CSin.y = 0.5 * (y1 - 1 / y1)
    Else
        b1 = 0
        b2 = 0
        y1 = 2 * (2 * cpxZ.y * cpxZ.y - 1)
        For i = 6 To 1 Step -1
            br = y1 * b1 - b2 - c(i)
            If i <> 1 Then
                b2 = b1
                b1 = br
            End If
        Next i

        CSin.y = cpxZ.y * (br - b1)
    End If

    ' 组合计算结果
    CSin.x = CSin.x * Sin(cpxZ.x)
    CSin.y = CSin.y * Cos(cpxZ.x)

End Function
```

3. **示例**

调用函数 CSin **求解复数** 5 +7 j **的正弦，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

    ' 求正弦
    cpxZ.x = 5
    cpxZ.y = 7
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
    Dim cpxSin As Complex
    cpxSin = CSin(cpxZ)
    sMsg = sComplex & " 的正弦为:" & cpxSin.x & " + " & cpxSin.y & "j"
    MsgBox sMsg

End Sub
```

其输出结果如图 1.8 **所示。**

图 1.8　程序输出结果

1.9　复数的余弦

1. **算法原理**

复数 $z = x + \mathrm{j}y$ **的余弦的计算方法如下：**

$$\cos z = \cos(x + \mathrm{j}y) = \cos x \cosh y - \mathrm{j}\sin x \sinh y$$

其中，

$$\cosh y = \frac{\mathrm{e}^{y} + \mathrm{e}^{-y}}{2}$$

$$\sinh y = \frac{\mathrm{e}^{y} - \mathrm{e}^{-y}}{2}$$

当 $|y| < 1$ **时，为了避免有效数字的损失，**sinh **改用切比雪夫公式计算，即**

$$\sinh y = y[-c_5 y_1^5 + (c_5 - c_4)y_1^4 + (4c_5 + c_4 - c_3)y_1^3 + (-3c_5 + c_4 + c_3 - c_2)y_1^2 + (-3c_5 - 2c_4 + 2c_3 c_2 - c_1)y_1 + (c_5 - c_4 - c_3 + c_2 + c_1 - c_0)$$

其中，

$c_0 = 1.13031820798497$

$c_1 = 0.04433684984866$

$c_2 = 0.00054292631191$

$c_3 = 0.00000319843646$

$c_4 = 0.00000001103607$

$c_5 = 0.00000000002498$

$y_1 = 2(2y^2 - 1)$

2. 算法实现

根据上述方法，可以定义计算复数的余弦的 Visual Basic 函数 CCos，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：ComplexModule.bas
'  函数名：CCos
'  功能：  计算复数的余弦
'  参数：  cpxZ   - Complex型变量，用于求余弦的复数
'  返回值：Complex型变量，为求得的复数的余弦
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CCos(cpxZ As Complex) As Complex
    ' 局部变量
    Dim i As Integer
    Dim y1 As Double, br As Double, b1 As Double, b2 As Double
    Dim c(6) As Double

    ' 切比雪夫公式的常数系数
    c(1) = 1.13031820798497
    c(2) = 0.04433684984866
    c(3) = 0.00054292631191
    c(4) = 0.00000319843646
    c(5) = 0.00000001103607
    c(6) = 0.00000000002498

    y1 = Exp(cpxZ.y)
    CCos.x = 0.5 * (y1 + 1 / y1)
    If Abs(cpxZ.y) >= 1 Then
        CCos.y = 0.5 * (y1 - 1 / y1)
    Else
        b1 = 0
        b2 = 0
        y1 = 2 * (2 * cpxZ.y * cpxZ.y - 1)
        For i = 6 To 1 Step -1
            br = y1 * b1 - b2 - c(i)
            If i <> 1 Then
                b2 = b1
                b1 = br
            End If
        Next i
```

```
        CCos.y = cpxZ.y * (br - b1)
    End If

    ′ 组合计算结果
    CCos.x = CCos.x * Cos(cpxZ.x)
    CCos.y = -CCos.y * Sin(cpxZ.x)

End Function
```

3. 示例

调用函数 CCos **求解复数** 0.75 +0.25 j **的余弦，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

′ 求余弦
cpxZ.x = 0.75
cpxZ.y = 0.25
sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
Dim cpxCos As Complex
cpxCos = CCos(cpxZ)
sMsg = sComplex & " 的余弦为:" & cpxCos.x & " + " & cpxCos.y & "j"
MsgBox sMsg

End Sub
```

其输出结果如图 1.9 **所示。**

图 1.9　程序输出结果

1.10　复数的正切

1. 算法原理

复数 $z = x + \mathrm{j}y$ **的正切的计算方法如下：**

$$\tan z = \frac{\sin z}{\cos z} = \frac{\sin(x + \mathrm{j}y)}{\cos(x + \mathrm{j}y)}$$

2. 算法实现

根据上述方法，可以定义计算复数的正切的 Visual Basic **函数** CTan**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
′  模块名：ComplexModule.bas
′  函数名：CTan
```

```
' 功能： 计算复数的正切
' 参数： cpxZ  - Complex型变量，用于求正切的复数
' 返回值：Complex型变量，为求得的复数的正切
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function CTan(cpxZ As Complex) As Complex
   ' 局部变量
   Dim i As Integer
   Dim y1 As Double, br As Double, b1 As Double, b2 As Double
   Dim c(6) As Double
   Dim cpxSin As Complex, cpxCos As Complex

   ' 切比雪夫公式的常数系数
   c(1) = 1.13031820798497
   c(2) = 0.04433684984866
   c(3) = 0.00054292631191
   c(4) = 0.00000319843646
   c(5) = 0.00000001103607
   c(6) = 0.00000000002498

   y1 = Exp(cpxZ.y)
   CTan.x = 0.5 * (y1 + 1 / y1)
   If Abs(cpxZ.y) >= 1 Then
      CTan.y = 0.5 * (y1 - 1 / y1)
   Else
      b1 = 0
      b2 = 0
      y1 = 2 * (2 * cpxZ.y * cpxZ.y - 1)
      For i = 6 To 1 Step -1
         br = y1 * b1 - b2 - c(i)
         If i <> 1 Then
            b2 = b1
            b1 = br
         End If
      Next i

      CTan.y = cpxZ.y * (br - b1)
   End If

   ' 组合计算结果
   cpxSin.x = CTan.x * Sin(cpxZ.x)
   cpxSin.y = CTan.y * Cos(cpxZ.x)
   cpxCos.x = CTan.x * Cos(cpxZ.x)
   cpxCos.y = -CTan.y * Sin(cpxZ.x)

   CTan = CDiv(cpxSin, cpxCos)

End Function
```

3. 示例

调用函数 CTan **求解复数** 0.25 +0.25 j **的正切，程序代码如下：**

```
Sub Main()
    Dim cpxZ As Complex
    Dim sComplex As String, sMsg As String

    ' 求正切
    cpxZ.x = 0.25
    cpxZ.y = 0.25
    sComplex = "复数 " & cpxZ.x & " + " & cpxZ.y & "j"
    Dim cpxTan As Complex
    cpxTan = CTan(cpxZ)
    sMsg = sComplex & " 的正切为:" & cpxTan.x & " + " & cpxTan.y & "j"
    MsgBox sMsg

End Sub
```

输出结果如图 1.10 **所示。**

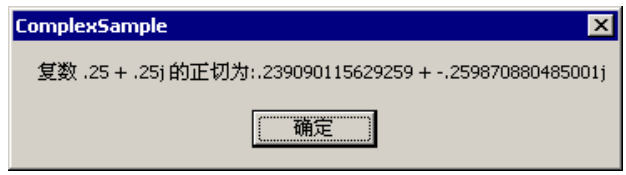

图 1.10　程序输出结果

第 2 章　矩 阵 运 算

矩阵就是由 $m\times n$ 个数 $a_{ij}(i=1,2,\Lambda\ m;\ j=1,2,\Lambda\ ,n)$ 排成 m 行 n 列的矩形表：

$$\begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \Lambda & \Lambda & \Lambda & \Lambda \\ a_{m1} & a_{m2} & \Lambda & a_{mn} \end{bmatrix}$$

矩阵是重要的数学概念，在科学、工程和社会经济等领域都具有重要的应用。在 Visual Basic 中一般用 $m\times n$ 二维数组来表示 $m\times n$ 矩阵。

本章将介绍矩阵的运算算法，包括矩阵基础运算(加法、转置、乘法和复矩阵乘法)、矩阵求逆的全选主元高斯-约当法、对称正定矩阵的求逆、托伯利兹矩阵求逆的特兰持方法、求行列式值的全选主元高斯消去法、求矩阵秩的全选主元高斯消去法、对称正定矩阵的乔里斯基分解与行列式的求值、矩阵的三角分解、一般实矩阵的 QR 分解、一般实矩阵的奇异值分解、求广义逆的奇异值分解法、约化对称矩阵为对称三对角阵的豪斯荷尔德变换法、实对称三对角阵的全部特征值与特征向量的计算、约化一般实矩阵为赫申伯格矩阵的初等相似变换法、求赫申伯格矩阵全部特征值的 QR 方法、求实对称矩阵特征值与特征向量的雅可比法和求实对称矩阵特征值与特征向量的雅可比过关法等。

为了示例的方便，先定义 3 个显示矩阵的函数 MatrixToString，MatrixRowToString 和 MatrixColToString，分别用于将矩阵、矩阵的某一行和某一列的所有元素转换为排列整齐的字符串。它们的代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  功能：  矩阵运算
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MatrixToString
'  功能：  将矩阵转换为显示字符串
'  参数：  m   - Integer型变量，矩阵的行数
'          n   - Integer型变量，矩阵的列数
'          mtxA  - Double型m x n二维数组，待显示的矩阵
'          sFormat - 显示矩阵各元素的格式控制字符串
'  返回值：String型，显示矩阵的字符串
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MatrixToString(m As Integer, n As Integer, mtxA() As Double, _
sFormat As String) As String
    Dim i As Integer, j As Integer
    Dim s As String
```

```
    s = ""
    For i = 1 To m
        For j = 1 To n
            s = s + Format(mtxA(i, j), sFormat) + "  "
        Next j
        s = s + Chr(13)
    Next i

    MatrixToString = s

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MatrixRowToString
'  功能：  将矩阵的指定行转换为显示字符串
'  参数：  n   - Integer型变量，矩阵的列数
'          r   - Integer型变量，要显示的矩阵的行数
'          mtxA  - Double型m x n二维数组，待显示的矩阵
'          sFormat - 显示矩阵各元素的格式控制字符串
'  返回值：String型，显示矩阵指定的行向量
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MatrixRowToString(n As Integer, r As Integer, mtxA() As Double, _
sFormat As String) As String
    Dim i As Integer, j As Integer
    Dim s As String

    s = ""
    For j = 1 To n
        s = s + Format(mtxA(r, j), sFormat) + "  "
    Next j

    MatrixRowToString = s

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MatrixColToString
'  功能：  将矩阵的指定列转换为显示字符串
'  参数：  m   - Integer型变量，矩阵的行数
'          c   - Integer型变量，要显示的矩阵的列数
'          mtxA  - Double型m x n二维数组，待显示的矩阵
'          sFormat - 显示矩阵各元素的格式控制字符串
'  返回值：String型，显示矩阵指定的列向量
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MatrixColToString(m As Integer, c As Integer, mtxA() As Double, _
sFormat As String) As String
    Dim i As Integer, j As Integer
    Dim s As String
```

```
    s = ""
    For i = 1 To m
       s = s + Format(mtxA(i, c), sFormat) + "  "
    Next i

    MatrixColToString = s

End Function
```

2.1 矩阵基础运算

1. **算法原理**

矩阵的基础运算包括加法、减法、转置、乘法和复矩阵乘法，它们的运算法则如下：

(1) **加法：** $m\times n$ **阶矩阵** $\boldsymbol{A}$ **与** $m\times n$ **阶矩阵** $\boldsymbol{B}$ **的和矩阵** $\boldsymbol{C}$ **定义为**

$$c_{ij}=a_{ij}+b_{ij},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ n$$

(2) **减法：** $m\times n$ **阶矩阵** $\boldsymbol{A}$ **与** $m\times n$ **阶矩阵** $\boldsymbol{B}$ **的差矩阵** $\boldsymbol{C}$ **定义为**

$$c_{ij}=a_{ij}-b_{ij},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ n$$

(3) **数乘：** $m\times n$ **阶矩阵** $\boldsymbol{A}$ **与数** λ **的乘积矩阵** $\boldsymbol{B}$ **定义为**

$$b_{ij}=\lambda a_{ij},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ n$$

(4) **转置：** $m\times n$ **阶矩阵** $\boldsymbol{A}$ **的转置矩阵** $\boldsymbol{A}^{\mathrm{T}}$ **定义为**

$$a_{ij}^{\mathrm{T}}=a_{ji},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ n$$

$\boldsymbol{A}^{\mathrm{T}}$ **是** $n\times m$ **阶矩阵。**

(5) **乘法：** $m\times n$ **阶矩阵** $\boldsymbol{A}$ **与** $n\times k$ **阶矩阵** $\boldsymbol{B}$ **的乘积矩阵** $\boldsymbol{C}$ **定义为**

$$c_{ij}=\sum_{t=1}^{n}a_{it}b_{tj},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ k$$

矩阵乘法要求左边的矩阵的列数与右边矩阵的行数是相等的。

(6) **复矩阵乘法：** $m\times n$ **阶复矩阵** $\boldsymbol{A}$ **与** $n\times k$ **阶复矩阵** $\boldsymbol{B}$ **的乘积矩阵** $\boldsymbol{C}$ **定义为**

$$c_{ij}=\sum_{t=1}^{n}a_{it}b_{tj},\ i=1,\ 2,\ \Lambda,\ m;\ \ j=1,\ 2,\ \Lambda,\ k$$

其中，

$\boldsymbol{A}=\boldsymbol{AR}+\mathrm{j}\boldsymbol{AI}$

$\boldsymbol{B}=\boldsymbol{BR}+\mathrm{j}\boldsymbol{BI}$

$\boldsymbol{C}=\boldsymbol{CR}+\mathrm{j}\boldsymbol{CI}$

a_{ij},b_{ij},c_{ij} **都是复数。**

两个复数相乘采用如下算法：

设： $e+\mathrm{j}f=(a+\mathrm{j}b)(c+\mathrm{j}d)$

令： $p=ac,\ q=bd,\ s=(a+b)(c+d)$

则： $e=p-q,\ f=s-p-q$

2. 算法实现

根据上述算法，可以定义计算矩阵加法、减法、转置、乘法和复矩阵乘法的 Visual Basic **函数** MAdd、MSub、MTrans、MMul **和** MCmul**，它们的代码如下：**

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MAdd
'  功能：  计算矩阵的加法
'  参数：  m   - Integer型变量，矩阵的行数
'          n   - Integer型变量，矩阵的列数
'          mtxA  - Double型m x n二维数组，存放相加的左边矩阵
'          mtxB  - Double型m x n二维数组，存放相加的右边矩阵
'          mtxC  - Double型m x n二维数组，返回和矩阵
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MAdd(m As Integer, n As Integer, mtxA() As Double, mtxB() As Double, _
mtxC() As Double)
    Dim i As Integer, j As Integer

    For i = 1 To m
        For j = 1 To n
            mtxC(i, j) = mtxA(i, j) + mtxB(i, j)
        Next j
    Next i

End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MSub
'  功能：  计算矩阵的减法
'  参数：  m   - Integer型变量，矩阵的行数
'          n   - Integer型变量，矩阵的列数
'          mtxA  - Double型m x n二维数组，存放相减的左边矩阵
'          mtxB  - Double型m x n二维数组，存放相减的右边矩阵
'          mtxC  - Double型m x n二维数组，返回差矩阵
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MSub(m As Integer, n As Integer, mtxA() As Double, mtxB() As Double, _
mtxC() As Double)
    Dim i As Integer, j As Integer

    For i = 1 To m
        For j = 1 To n
            mtxC(i, j) = mtxA(i, j) + mtxB(i, j)
        Next j
    Next i

End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MNmul
'  功能：  计算矩阵的数乘
'  参数：  m      - Integer型变量，矩阵的行数
'          n      - Integer型变量，矩阵的列数
'          dblNum  - Double型变量，乘数
'          mtxA    - Double型m x n二维数组，存放乘数矩阵
'          mtxB    - Double型m x n二维数组，存放数乘的结果矩阵
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MNmul(m As Integer, n As Integer, dblNum As Double, _
mtxA() As Double, mtxB() As Double)
    Dim i As Integer, j As Integer

    For i = 1 To m
        For j = 1 To n
            mtxB(i, j) = dblNum * mtxA(i, j)
        Next j
    Next i

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MTrans
'  功能：  计算矩阵的转置
'  参数：  m   - Integer型变量，矩阵的行数
'          n   - Integer型变量，矩阵的列数
'          mtxA  - Double型m x n二维数组，存放原矩阵
'          mtxAT  - Double型n x m二维数组，返回转置矩阵
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MTrans(m As Integer, n As Integer, mtxA() As Double, mtxAT() As Double)
    Dim i As Integer, j As Integer

    For i = 1 To m
        For j = 1 To n
            mtxAT(i, j) = mtxAT(j, i)
        Next j
    Next i

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MMul
'  功能：  计算矩阵的乘法
'  参数：  m   - Integer型变量，相乘的左边矩阵的行数
'          n   - Integer型变量，相乘的左边矩阵的列数和右边矩阵的行数
'          l   -  Integer型变量，相乘的右边矩阵的列数
'          mtxA  - Double型m x n二维数组，存放相乘的左边矩阵
```

```
'         mtxB  - Double型n x l二维数组，存放相乘的右边矩阵
'         mtxC  - Double型m x l二维数组，返回矩阵乘积矩阵
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MMul(m As Integer, n As Integer, l As Integer, mtxA() As Double, _
mtxB() As Double, mtxC() As Double)
    Dim i As Integer, j As Integer, k As Integer

    For i = 1 To m
        For j = 1 To l
            mtxC(i, j) = 0#
            For k = 1 To n
                mtxC(i, j) = mtxC(i, j) + mtxA(i, k) * mtxB(k, j)
            Next k
        Next j
    Next i

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MCmul
'  功能：  计算复矩阵乘法
'  参数：  m   - Integer型变量，相乘的左边矩阵的行数
'          n   - Integer型变量，相乘的左边矩阵的列数和右边矩阵的行数
'          l   -  Integer型变量，相乘的右边矩阵的列数
'          mtxAR  - Double型m x n二维数组，存放相乘的左边矩阵的实部
'          mtxAI  - Double型m x n二维数组，存放相乘的左边矩阵的虚部
'          mtxBR  - Double型n x l二维数组，存放相乘的右边矩阵的实部
'          mtxBI  - Double型n x l二维数组，存放相乘的右边矩阵的虚部
'          mtxCR  - Double型m x l二维数组，返回矩阵乘积矩阵的实部
'          mtxCI  - Double型m x l二维数组，返回矩阵乘积矩阵的虚部
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MCmul(m As Integer, n As Integer, l As Integer, mtxAR() As Double, _
mtxAI() As Double, mtxBR() As Double, mtxBI() As Double, _
mtxCR() As Double, mtxCI() As Double)
    Dim i As Integer, j As Integer, k As Integer
    Dim p As Double, q As Double, s As Double

    For i = 1 To m
        For j = 1 To l
            mtxCR(i, j) = 0#
            mtxCI(i, j) = 0#
            For k = 1 To n
                p = mtxAR(i, k) * mtxBR(k, j)
                q = mtxAI(i, k) * mtxBI(k, j)
                s = (mtxAR(i, k) + mtxAI(i, k)) * (mtxBR(k, j) + mtxBI(k, j))
                mtxCR(i, j) = mtxCR(i, j) + p - q
                mtxCI(i, j) = mtxCI(i, j) + s - p - q
            Next k
        Next j
    Next i
```

```
End Sub
```

3. 示例

调用函数 MMul 来求解如下矩阵的乘积：

$$\begin{bmatrix} 1 & 3 & -2 & 0 & 4 \\ -2 & -1 & 5 & -7 & 2 \\ 0 & 8 & 4 & 1 & -5 \\ 3 & -3 & 2 & -4 & 1 \end{bmatrix} \begin{bmatrix} 4 & 5 & -1 \\ 2 & -2 & 6 \\ 7 & 8 & 1 \\ 0 & 3 & -5 \\ 9 & 8 & -6 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 5) As Double
    Dim mtxB(5, 3) As Double
    Dim mtxC(4, 3) As Double

    mtxA(1,1)=1 : mtxA(1,2)=3 : mtxA(1,3)=-2: mtxA(1,4)=0 : mtxA(1,5)=4
    mtxA(2,1)=-2: mtxA(2,2)=-1: mtxA(2,3)=5 : mtxA(2,4)=-7: mtxA(2,5)=2
    mtxA(3,1)=0 : mtxA(3,2)=8 : mtxA(3,3)=4 : mtxA(3,4)=1 : mtxA(3,5)=-5
    mtxA(4,1)=3 : mtxA(4,2)=-3: mtxA(4,3)=2 : mtxA(4,4)=-4: mtxA(4,5)=1

    mtxB(1, 1) = 4: mtxB(1, 2) = 5: mtxB(1, 3) = -1
    mtxB(2, 1) = 2: mtxB(2, 2) = -2: mtxB(2, 3) = 6
    mtxB(3, 1) = 7: mtxB(3, 2) = 8: mtxB(3, 3) = 1
    mtxB(4, 1) = 0: mtxB(4, 2) = 3: mtxB(4, 3) = -5
    mtxB(5, 1) = 9: mtxB(5, 2) = 8: mtxB(5, 3) = -6

    '求解
    Call MMul(4, 5, 3, mtxA, mtxB, mtxC)

    MsgBox "求解成功！" & Chr$(13) & Chr$(13) & MatrixToString(4, 3, _
    mtxC, "######0.0000000")

End Sub
```

其输出结果如图 2.1 所示。

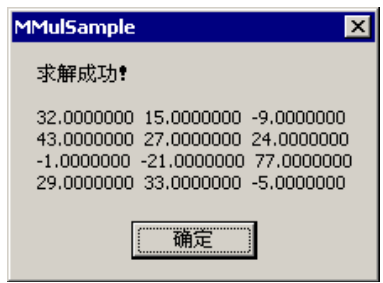

图 2.1　程序输出结果

2.2　实矩阵求逆的全选主元高斯-约当法

1. 算法原理

设 A 为 n 阶方阵，若存在 n 阶方阵 B 使 $AB=BA=I$ 成立，则称 A 可逆，B 是 A 的逆矩阵，记为 A^{-1}。逆矩阵是惟一的。

使用高斯-约当法(全选主元)求逆时对 k 从 0 到 n-1 作如下几步：

(1) 从第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素，并记住此元素所在的行号和列号，再通过行交换与列交换将它交换到主元素位置上。这一步称为全选主元。

(2) $1/a_{kk} \Rightarrow a_{kk}$

(3) $a_{kj}a_{kk} \Rightarrow a_{kj},\ j=0,1,\Lambda\ ,n-1;\ j\neq k$

(4) $a_{ij}-a_{ik}a_{kj} \Rightarrow a_{ij},\ i,\ j=0,1,\Lambda\ ,n-1;\ i,\ j\neq k$

(5) $-a_{ik}a_{kk} \Rightarrow a_{ik},\ i=0,1,\Lambda\ ,n-1;\ i\neq k$

根据在全选主元过程中所记录的行、列交换信息进行恢复，恢复的原则如下：

在全选主元过程中，先交换的行、列后进行恢复；原来的行(列)交换用列(行)交换来恢复。

2. 算法实现

根据上述算法，可以定义矩阵求逆的 Visual Basic 函数 MRinv，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MRinv
'  功能：  矩阵求逆
'  参数：  n      - Integer型变量，矩阵的阶数
'          mtxA   - Double型二维数组，体积为n x n。存放原矩阵A；
'                   返回时存放其逆矩阵A-1。
'  返回值：Boolean型，失败为False，成功为True
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MRinv(n As Integer, mtxA() As Double) As Boolean
    ' 局部变量
    ReDim nIs(n) As Integer, nJs(n) As Integer
    Dim i As Integer, j As Integer, k As Integer
    Dim d As Double, p As Double

    ' 全选主元，消元
    For k = 1 To n
        d = 0#
        For i = k To n
            For j = k To n
                p = Abs(mtxA(i, j))
                If (p > d) Then
                    d = p
```

```
                nIs(k) = i
                nJs(k) = j
            End If
        Next j
    Next i

    ' 求解失败
    If (d + 1# = 1#) Then
        MRinv = False
        Exit Function
    End If

    If (nIs(k) <> k) Then
        For j = 1 To n
            p = mtxA(k, j)
            mtxA(k, j) = mtxA(nIs(k), j)
            mtxA(nIs(k), j) = p
        Next j
    End If

    If (nJs(k) <> k) Then
        For i = 1 To n
            p = mtxA(i, k)
            mtxA(i, k) = mtxA(i, nJs(k))
            mtxA(i, nJs(k)) = p
        Next i
    End If

    mtxA(k, k) = 1# / mtxA(k, k)
    For j = 1 To n
        If (j <> k) Then mtxA(k, j) = mtxA(k, j) * mtxA(k, k)
    Next j
    For i = 1 To n
        If (i <> k) Then
            For j = 1 To n
                If (j <> k) Then mtxA(i, j) = mtxA(i, j) - _
                                              mtxA(i, k) * mtxA(k, j)
            Next j
        End If
    Next i
    For i = 1 To n
        If (i <> k) Then mtxA(i, k) = -mtxA(i, k) * mtxA(k, k)
    Next i
Next k

' 调整恢复行列次序
For k = n To 1 Step -1
    If (nJs(k) <> k) Then
      For j = 1 To n
         p = mtxA(k, j)
         mtxA(k, j) = mtxA(nJs(k), j)
```

```
            mtxA(nJs(k), j) = p
          Next j
        End If
        If (nIs(k) <> k) Then
          For i = 1 To n
            p = mtxA(i, k)
            mtxA(i, k) = mtxA(i, nIs(k))
            mtxA(i, nIs(k)) = p
          Next i
        End If
    Next k

    ' 求解成功
MRinv = True

End Function
```

3. 示例

调用函数 MRInv 求下列 4 阶矩阵的逆矩阵，并计算 AA^{-1} 以检验其结果的正确性：

$$\boldsymbol{A}=\begin{bmatrix}0.2368 & 0.2471 & 0.2568 & 1.2671\\ 1.1161 & 0.1254 & 0.1397 & 0.1490\\ 0.1582 & 1.1675 & 0.1768 & 0.1871\\ 0.1968 & 0.2071 & 1.2168 & 0.2271\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 4) As Double
    Dim mtxB(4, 4) As Double
    Dim mtxC(4, 4) As Double

    ' 原矩阵
    mtxA(1,1)=0.2368:mtxA(1,2)=0.2471:mtxA(1,3)=0.2568:mtxA(1,4)=1.2671
    mtxA(2,1)=1.1161:mtxA(2,2)=0.1254:mtxA(2,3)=0.1397:mtxA(2,4)=0.149
    mtxA(3,1)=0.1582:mtxA(3,2)=1.1675:mtxA(3,3)=0.1768:mtxA(3,4)=0.1871
    mtxA(4,1)=0.1968:mtxA(4,2)=0.2071:mtxA(4,3)=1.2168:mtxA(4,4)=0.2271

    ' 备份原矩阵
    For i = 1 To 4
        For j = 1 To 4
            mtxB(i, j) = mtxA(i, j)
        Next j
    Next i

    '求解
    If MRinv(4, mtxA) Then

        ' 矩阵乘法AA-
        Call MMul(4, 4, 4, mtxB, mtxA, mtxC)
```

```
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "矩阵A为" & Chr$(13) & Chr$(13) & _
              MatrixToString(4, 4, mtxB, "######0.0000000") & Chr$(13) & _
              Chr$(13) & "逆矩阵A-为" & Chr$(13) & Chr$(13) & _
              MatrixToString(4, 4, mtxA, "######0.0000000") & Chr$(13) & _
              Chr$(13) & "矩阵A与其逆矩阵A-之乘积矩阵为" & Chr$(13) & _
              Chr$(13) & MatrixToString(4, 4, mtxC, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

输出结果如图 2.2 所示。

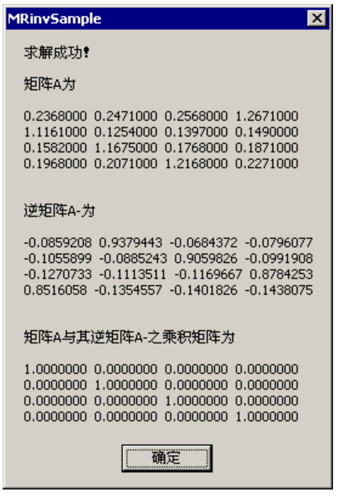

图 2.2　程序输出结果

2.3　复矩阵求逆的全选主元高斯-约当法

1. 算法原理

复矩阵求逆的全选主元高斯-约当法的算法同实矩阵的求逆，参见 2.2 节。其中复矩阵分成实部与虚部两部分，即：

$$\boldsymbol{A}=\boldsymbol{AR}+\mathrm{j}\boldsymbol{AI}$$

算法中的所有运算均为复数运算。有关复数相乘与复数相除的算法参见 1.1 和 1.2 节的相关内容。

2. 算法实现

复矩阵求逆的 Visual Basic 函数 MCinv 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MCinv
```

```
' 功能：  复矩阵求逆
' 参数：  n      - Integer型变量，矩阵的阶数
'         mtxAR - Double型二维数组，体积为n x n。
'                 存放原矩阵A的实部；返回时存放其逆矩阵A^-1的实部。
'         mtxAI - Double型二维数组，体积为n x n。
'                 存放原矩阵A的虚部；返回时存放其逆矩阵A^-1的虚部。
' 返回值：Boolean型，失败为False，成功为True
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MCinv(n As Integer, mtxAR() As Double, mtxAI() As Double) As Boolean
    ' 局部变量
    ReDim nIs(n) As Integer, nJs(n) As Integer
    Dim i As Integer, j As Integer, k As Integer
    Dim d As Double, p As Double, s As Double, t As Double, q As Double, _
        b As Double

    ' 全选主元，消元
    For k = 1 To n
        d = 0#
        For i = k To n
            For j = k To n
                p = mtxAR(i, j) * mtxAR(i, j) + mtxAI(i, j) * mtxAI(i, j)
                If (p > d) Then
                    d = p
                    nIs(k) = i
                    nJs(k) = j
                End If
            Next j
        Next i

        ' 求解失败
        If (d + 1# = 1#) Then
            MCinv = False
            Exit Function
        End If

        If (nIs(k) <> k) Then
            For j = 1 To n
                t = mtxAR(k, j)
                mtxAR(k, j) = mtxAR(nIs(k), j)
                mtxAR(nIs(k), j) = t

                t = mtxAI(k, j)
                mtxAI(k, j) = mtxAI(nIs(k), j)
                mtxAI(nIs(k), j) = t
             Next j
        End If
        If (nJs(k) <> k) Then
            For i = 1 To n
                t = mtxAR(i, k)
                mtxAR(i, k) = mtxAR(i, nJs(k))
                mtxAR(i, nJs(k)) = t
```

```
            t = mtxAI(i, k)
            mtxAI(i, k) = mtxAI(i, nJs(k))
            mtxAI(i, nJs(k)) = t
        Next i
    End If

    mtxAR(k, k) = mtxAR(k, k) / d
    mtxAI(k, k) = -mtxAI(k, k) / d
    For j = 1 To n
        If (j <> k) Then
            p = mtxAR(k, j) * mtxAR(k, k)
            q = mtxAI(k, j) * mtxAI(k, k)
            s = (mtxAR(k, j) + mtxAI(k, j)) * (mtxAR(k, k) + mtxAI(k, k))
            mtxAR(k, j) = p - q
            mtxAI(k, j) = s - p - q
        End If
    Next j
    For i = 1 To n
        If (i <> k) Then
          For j = 1 To n
               If (j <> k) Then
                  p = mtxAR(k, j) * mtxAR(i, k)
                  q = mtxAI(k, j) * mtxAI(i, k)
                  s = (mtxAR(k, j) + mtxAI(k, j)) * (mtxAR(i, k) + _
                       mtxAI(i, k))
                  t = p - q
                  b = s - p - q
                  mtxAR(i, j) = mtxAR(i, j) - t
                  mtxAI(i, j) = mtxAI(i, j) - b
               End If
            Next j
        End If
    Next i

    For i = 1 To n
        If (i <> k) Then
            p = mtxAR(i, k) * mtxAR(k, k)
            q = mtxAI(i, k) * mtxAI(k, k)
            s = (mtxAR(i, k) + mtxAI(i, k)) * (mtxAR(k, k) + mtxAI(k, k))
            mtxAR(i, k) = q - p
            mtxAI(i, k) = p + q - s
        End If
    Next i
Next k

' 调整恢复行列次序
For k = n To 1 Step -1
    If (nJs(k) <> k) Then
      For j = 1 To n
         t = mtxAR(k, j)
         mtxAR(k, j) = mtxAR(nJs(k), j)
```

```
                mtxAR(nJs(k), j) = t
                t = mtxAI(k, j)
                mtxAI(k, j) = mtxAI(nJs(k), j)
                mtxAI(nJs(k), j) = t
            Next j
        End If
        If (nIs(k) <> k) Then
            For i = 1 To n
                t = mtxAR(i, k)
                mtxAR(i, k) = mtxAR(i, nIs(k))
                mtxAR(i, nIs(k)) = t
                t = mtxAI(i, k)
                mtxAI(i, k) = mtxAI(i, nIs(k))
                mtxAI(i, nIs(k)) = t
            Next i
        End If
    Next k

    ' 求解成功
MCinv = True

End Function
```

3. 示例

调用函数 MCinv **求下列** 4 **阶复矩阵** $\boldsymbol{A}$ **的逆矩阵** $\boldsymbol{A}^{-1}$**，并计算** $\boldsymbol{A}\boldsymbol{A}^{-1}$ **以检验其结果的正确性：**

$$\boldsymbol{A} = \boldsymbol{AR} + \mathrm{j}\boldsymbol{AI}$$

$$\boldsymbol{AR} = \begin{bmatrix} 0.236800 & 0.247100 & 0.256800 & 1.26710 \\ 1.11610 & 0.125400 & 0.139700 & 0.149000 \\ 0.158200 & 1.16750 & 0.176800 & 0.187100 \\ 0.196800 & 0.207100 & 1.216800 & 0.227100 \end{bmatrix}$$

$$\boldsymbol{AI} = \begin{bmatrix} 0.134500 & 0.167800 & 0.182500 & 1.11610 \\ 1.26710 & 0.201700 & 0.702400 & 0.272100 \\ -0.283600 & -1.19670 & 0.355800 & -0.207800 \\ 0.357600 & -1.23450 & 2.11850 & 0.477300 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxAR(4, 4) As Double, mtxAI(4, 4) As Double
    Dim mtxBR(4, 4) As Double, mtxBI(4, 4) As Double
    Dim mtxCR(4, 4) As Double, mtxCI(4, 4) As Double
    Dim s1 As String, s2 As String

    ' 原矩阵
mtxAR(1,1)=0.2368:mtxAR(1,2)=0.2471:mtxAR(1,3)=0.2568:mtxAR(1,4)=1.2671
mtxAR(2,1)=1.1161:mtxAR(2,2)=0.1254:mtxAR(2,3)=0.1397:mtxAR(2,4)=0.149
```

```
mtxAR(3,1)=0.1582:mtxAR(3,2)=1.1675:mtxAR(3,3)=0.1768:mtxAR(3,4)=0.1871
mtxAR(4,1)=0.1968:mtxAR(4,2)=0.2071:mtxAR(4,3)=1.2168:mtxAR(4,4)=0.2271

mtxAI(1,1)=0.1345:mtxAI(1,2)=0.1678:mtxAI(1,3)=0.1875:mtxAI(1,4)=1.1161
mtxAI(2,1)=1.2671:mtxAI(2,2)=0.2017:mtxAI(2,3)=0.7024:mtxAI(2,4)=0.2721
mtxAI(3,1)=-0.2836:mtxAI(3,2)=-1.1967:mtxAI(3,3)=0.3558:mtxAI(3,4)=-0.20
78
mtxAI(4,1)=0.3576:mtxAI(4,2)=-1.2345:mtxAI(4,3)=2.1185:mtxAI(4,4)=0.4773

   ' 备份原矩阵
   For i = 1 To 4
      For j = 1 To 4
         mtxBR(i, j) = mtxAR(i, j)
         mtxBI(i, j) = mtxAI(i, j)
      Next j
   Next i

   '求解
   If MCinv(4, mtxAR, mtxAI) Then

      ' 复矩阵乘法
      Call MCmul(4, 4, 4, mtxBR, mtxBI, mtxAR, mtxAI, mtxCR, mtxCI)

      s1 = "求解成功！" & Chr$(13) & Chr$(13)
      s1 = s1 + "矩阵AR为" & Chr$(13) & Chr$(13)
      s1 = s1 + MatrixToString(4, 4, mtxBR, "######0.0000000") & _
           Chr$(13) & Chr$(13)
      s1 = s1 + "矩阵AI为" & Chr$(13) & Chr$(13)
      s1 = s1 + MatrixToString(4, 4, mtxBI, "######0.0000000") & _
           Chr$(13) & Chr$(13)
      s1 = s1 + "逆矩阵AR-为" & Chr$(13) & Chr$(13)
      s1 = s1 + MatrixToString(4, 4, mtxAR, "######0.0000000") & _
           Chr$(13) & Chr$(13)
      s1 = s1 + "逆矩阵AI-为" & Chr$(13) & Chr$(13)
      s1 = s1 + MatrixToString(4, 4, mtxAI, "######0.0000000") & _
           Chr$(13) & Chr$(13)

      s2 = "矩阵AR与其逆矩阵AR-之乘积矩阵为" & Chr$(13) & Chr$(13)
      s2 = s2 + MatrixToString(4, 4, mtxCR, "######0.0000000") & _
           Chr$(13) & Chr$(13)
      s2 = s2 + "矩阵AI与其逆矩阵AI-之乘积矩阵为" & Chr$(13) & Chr$(13)
      s2 = s2 + MatrixToString(4, 4, mtxCI, "######0.0000000")

      MsgBox s1
      MsgBox s2

   Else
      MsgBox "求解失败！"
End If

End Sub
```

输出结果如图 2.3 与图 2.4 所示。

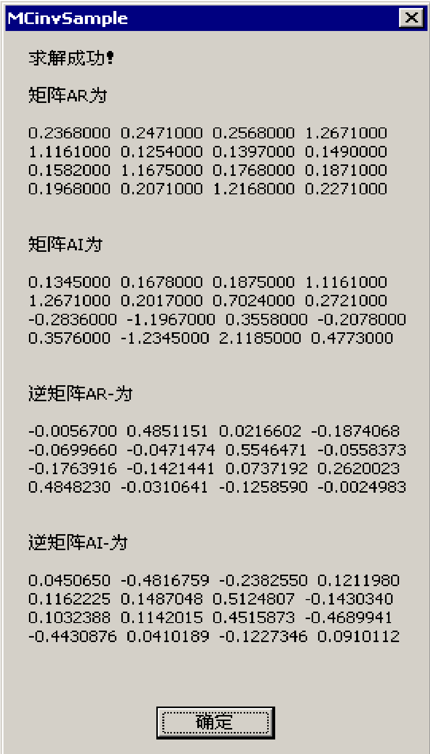

图 2.3　程序显示结果

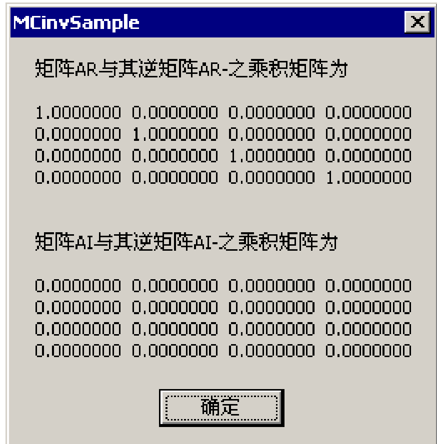

图 2.4　程序显示结果

2.4　对称正定矩阵的求逆

1. 算法原理

一般采用“变量循环重新编号法”来计算求 n 阶对称正定矩阵 A 的逆矩阵 A^{-1}，计算公式如下：

$$a'_{nn} = 1/a_{11}$$

$$a'_{n,j-1} = -a_{1j}/a_{11},\quad j = 2, 3, \Lambda, n$$

$$a'_{i-1,n} = a_{i1}/a_{11},\quad i = 2, 3, \Lambda, n$$

$$a'_{i-1,j-1} = a_{ij} - a_{i1}a_{1j}/a_{11},\quad i, j = 2, 3, \Lambda, n$$

当 A 为对称正定矩阵时，其逆矩阵 A^{-1} 也是对称正定矩阵。

2. 算法实现

根据上述方法，可以定义计算对称正定矩阵的逆的 Visual Basic 函数 MSsgj，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' 模块名：MatrixModule.bas
' 函数名：MSsgj
' 功能： 计算对称正定矩阵的逆
' 参数： n    - Integer型变量，矩阵阶数。
'        mtxA - Double型二维数组，体积为n x n。存放实对称正定矩阵A；
'               返回时存放逆矩阵A^-1。
' 返回值：Boolean型，成功为True，失败为False。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MSsgj(n As Integer, mtxA() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j, k, m
    Dim w As Double, g As Double
    ReDim b(n) As Double

    ' 循环重新编号求解
    For k = 1 To n
        w = a(1, 1)
        ' 求解失败
        If (Abs(w) + 1# = 1#) Then
            MSsgj = False
            Exit Function
        End If

        m = n - k - 1
        For i = 2 To n
            g = a(i, 1)
            b(i) = g / w
            If (i <= m) Then b(i) = -b(i)
            For j = 2 To i
                a(i - 1, j - 1) = a(i, j) + g * b(j)
            Next j
        Next i
        a(n, n) = 1# / w
        For i = 2 To n
            a(n, i - 1) = b(i)
        Next i
    Next k

    For i = 1 To n - 1
        For j = i + 1 To n
            a(i, j) = a(j, i)
        Next j
    Next i

    ' 求解成功
    MSsgj = True

End Function
```

3. 示例

调用函数 MSsgj **求下列** 4 **阶对称正定矩阵** A **的逆矩阵** A^{-1}**，并计算** AA^{-1} **以检验结果的正确性：**

$$\boldsymbol{A}=\begin{bmatrix}5 & 7 & 6 & 5\\ 7 & 10 & 8 & 7\\ 6 & 8 & 10 & 9\\ 5 & 7 & 9 & 10\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 4) As Double
    Dim mtxB(4, 4) As Double
    Dim mtxC(4, 4) As Double

    ' 原矩阵
    mtxA(1, 1) = 5:    mtxA(1, 2) = 7:    mtxA(1, 3) = 6:    mtxA(1, 4) = 5
    mtxA(2, 1) = 7:    mtxA(2, 2) = 10:    mtxA(2, 3) = 8:    mtxA(2, 4) = 7
    mtxA(3, 1) = 6:    mtxA(3, 2) = 8:    mtxA(3, 3) = 10:    mtxA(3, 4) = 9
    mtxA(4, 1) = 5:    mtxA(4, 2) = 7:    mtxA(4, 3) = 9:    mtxA(4, 4) = 10

    ' 备份原矩阵
    For i = 1 To 4
        For j = 1 To 4
            mtxB(i, j) = mtxA(i, j)
        Next j
    Next i

    '求解
    If MRinv(4, mtxA) Then

        ' 矩阵乘法AA-
        Call MMul(4, 4, 4, mtxB, mtxA, mtxC)

        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "矩阵A为" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 4, mtxB, "######0.0000000") & Chr$(13) & _
               Chr$(13) & "逆矩阵A-为" & Chr$(13) & Chr$(13) & _
               MatrixToString(4,4, mtxA, "######0.0000000") & Chr$(13) & _
               Chr$(13) & "矩阵A与其逆矩阵A-之乘积矩阵为" & Chr$(13) & _
               Chr$(13) & MatrixToString(4, 4, mtxC, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

输出结果如图 2.5 **所示。**

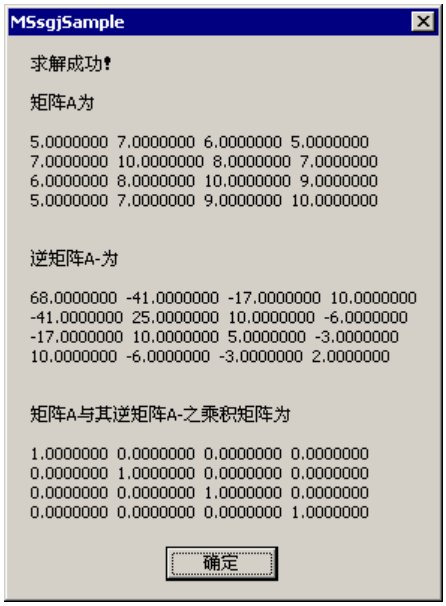

图 2.5　程序输出结果

2.5　托伯利兹矩阵求逆的特兰持方法

1. 算法原理

本节介绍用特兰持(Trench)方法求托伯利兹(Toeplitz)矩阵的逆矩阵的算法。
设 n 阶托伯利兹矩阵为

$$\boldsymbol{T}^{(n)}=\begin{bmatrix} t_0 & t_1 & t_2 & \Lambda & t_{n-1} \\ \tau_1 & t_0 & t_1 & \Lambda & t_{n-2} \\ \tau_2 & \tau_1 & t_0 & \Lambda & t_{n-3} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ \tau_{n-1} & \tau_{n-2} & \tau_{n-3} & \Lambda & t_0 \end{bmatrix}$$

简称 n 阶 T 型矩阵。其求逆矩阵过程如下：

(1) **取初值** $a_0=t_0$，$c_0^{(o)}=\tau_1/t_0$，$r_0^{(0)}=t_1/t_0$。

(2) **对于 k 从 0 到 n-3 作如下计算：**

$$c_i^{(k+1)}=c_i^{(k)}+\frac{r_{k-i}^{(k)}}{a_k}\left(\sum_{j=1}^{k+1}c_{k+1-j}^{(k)}\tau_j-\tau_{k+2}\right),\ i=0,1,\Lambda\ ,k$$

$$c_{k+1}^{(k+1)}=\frac{1}{a_k}\left(\tau_{k+2}-\sum_{j=1}^{k+1}c_{k+1-j}^{(k)}\tau_j\right)$$

$$r_i^{(k+1)} = r_i^{(k)} + \frac{c_{k-i}^{(k)}}{a_k}(\sum_{j=1}^{k+1} r_{k+1-j}^{(k)} t_j - t_{k+2}),\ \ i = 0, 1, \Lambda\ , k$$

$$r_{k+1}^{(k+1)} = \frac{1}{a_k}(t_{k+2} - \sum_{j=1}^{k+1} r_{k+1-j} t_j)$$

$$a_{k+1} = t_0 - \sum_{j=1}^{k+2} t_j\ c_{j-1}^{(k+1)}$$

最后可算出 a_{n-2} **以及** $c_i^{(n-2)}$ **和** $r_i^{(n-2)} (i = 0, 1, \Lambda\ , n-2)$ **。**

(3)　**计算逆矩阵 $B^{(n)}$ 中的各元素：**

$$b_{00} = 1/a_{n-2}$$

$$b_{0,\ j+1} = -r_j^{(n-2)} / a_{n-2},\ \ j = 0, 1, \Lambda\ , n-2$$

$$b_{i+1,\ 0} = -c_i^{(n-2)} / a_{n-2},\ \ i = 0, 1, \Lambda\ , n-2$$

$$b_{i+1,\ j+1} = b_{ij} + \frac{1}{a_{n-2}}\left[c_i^{(n-2)} r_j^{(n-2)} - r_{n-i-2}^{(n-2)} c_{n-j-2}^{(n-2)}\right],\ \ i,\ j = 0, 1, \Lambda\ , n-2$$

2. 算法实现

根据上述方法，可以定义用特兰持方法求托伯利兹矩阵的逆矩阵的 Visual Basic **函数** MTinv**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MTinv
'  功能：  用特兰持(Trench)方法求托伯利兹(Toeplitz)矩阵的逆矩阵
'  参数：  n      - Integer型变量，T型矩阵阶数。
'          dblT  - Double型一维数组，长度为n。存放n阶T型矩阵中的上三角元素
'                 t0,t1,…tn-1。
'          dblTT  - Double型一维数组，长度为n。其中后n-1个元素tt(1),…,tt(n-1)依次
'                  存放n阶T型矩阵中的元素。
'          dblB    - Double型二维数组，体积为n x n。返回n阶T型矩阵的逆矩阵。
'  返回值：Boolean型，成功为True，失败为False。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MTinv(n As Integer, dblT() As Double, dblTT() As Double, _
dblB() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim a As Double, s As Double
    ReDim c(n) As Double, r(n) As Double, p(n) As Double

    ' 矩阵非T型矩阵
    If (Abs(dblT(1)) + 1# = 1#) Then
        MTinv = False
        Exit Function
    End If

    ' 取初值
```

```
a = dblT(1)
c(1) = dblTT(2) / dblT(1)
r(1) = dblT(2) / dblT(1)

' 循环计算
For k = 1 To n - 2
   s = 0#
   For j = 2 To k + 1
      s = s + c(k + 1 - j + 1) * dblTT(j)
   Next j

   s = (s - dblTT(k + 2)) / a

   For i = 1 To k
      p(i) = c(i) + s * r(k - i + 1)
   Next i

   c(k + 1) = -s
   s = 0#
   For j = 2 To k + 1
      s = s + r(k + 1 - j + 1) * dblT(j)
   Next j

   s = (s - dblT(k + 2)) / a
   For i = 1 To k
      r(i) = r(i) + s * c(k - i + 1)
      c(k - i + 1) = p(k - i + 1)
   Next i

   r(k + 1) = -s
   a = 0#
   For j = 2 To k + 2
      a = a + dblT(j) * c(j - 1)
   Next j

   a = dblT(1) - a
   If (Abs(a) + 1# = 1#) Then
      MTinv = False
      Exit Function
   End If
Next k

dblB(1, 1) = 1# / a
For i = 1 To n - 1
   dblB(1, i + 1) = -r(i) / a
   dblB(i + 1, 1) = -c(i) / a
Next i

' 计算逆矩阵中的各元素
For i = 1 To n - 1
   For j = 1 To n - 1
```

```
            dblB(i + 1, j + 1) = dblB(i, j) - c(i) * dblB(1, j + 1)
            dblB(i + 1, j + 1) = dblB(i + 1, j + 1) + _
                                 c(n - j) * dblB(1, n - i + 1)
        Next j
    Next i

    ' 求解成功
    MTinv = True

End Function
```

3. 示例

调用函数 MTinv **求下列** 6 **阶** T **型矩阵** $\boldsymbol{T}^{(6)}$**的逆矩阵：**

$$\boldsymbol{T}^{(6)}=\begin{bmatrix}10 & 5 & 4 & 3 & 2 & 1\\ -1 & 10 & 5 & 4 & 3 & 2\\ -2 & -1 & 10 & 5 & 4 & 3\\ -3 & -2 & -1 & 10 & 5 & 4\\ -4 & -3 & -2 & -1 & 10 & 5\\ -5 & -4 & -3 & -2 & -1 & 10\end{bmatrix}$$

其中，

$\boldsymbol{T}$=(10，5，4，3，2，1)

$\boldsymbol{TT}$=(0，-1，-2，-3，-4，-5)

n=6

程序代码如下：

```
Sub Main()
    Dim t(6) As Double
    Dim tt(6) As Double '={0.0,-1.0,-2.0,-3.0,-4.0,-5.0}
    Dim mtxB(6, 6) As Double

    t(1) = 10
    t(2) = 5
    t(3) = 4
    t(4) = 3
    t(5) = 2
    t(6) = 1

    tt(1) = 0
    tt(2) = -1
    tt(3) = -2
    tt(4) = -3
    tt(5) = -4
    tt(6) = -5

    '求解
```

```
    If MTinv(6, t, tt, mtxB) Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
                "逆矩阵为" & Chr$(13) & Chr$(13) & _
                MatrixToString(6, 6, mtxB, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

输出结果如图 2.6 所示。

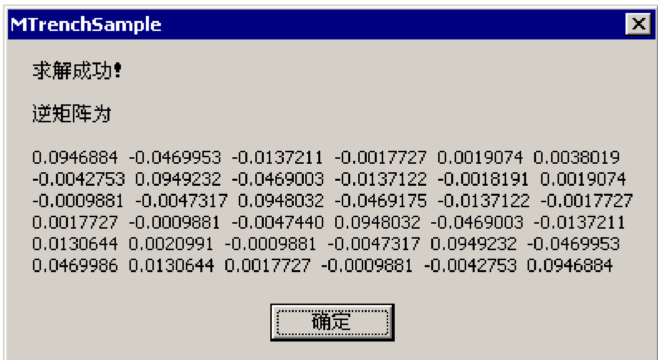

图 2.6　程序输出结果

2.6　求行列式值的全选主元高斯消去法

1. 算法原理

用全选主元高斯(Guass)消去法计算 n 阶方阵 A 所对应的行列式值的基本思路是：用高斯消去法对矩阵 A 进行一系列变换，使之成为上三角矩阵，其主对角线上的各元素的乘积即为行列式的值。

变换过程如下：

对于 $k=0,\ 1,\ \Lambda\ ,\ n-2$ 作变换

$$a_{ij}-a_{ik}a_{ki}/a_{kk} \Rightarrow a_{ij},\ i,\ j=k+1,\Lambda\ ,n-1$$

为保证数值计算的稳定性，在实际变换过程中采用全选主元。

2. 算法实现

根据上述方法，可以定义用全选主元高斯消去法计算 n 阶方阵 A 所对应的行列式值的 Visual Basic 函数 MDetGauss，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MDetGauss
'  功能：  用全选主元高斯消去法求行列式的值
'  参数：  n      - Integer型变量，方阵的阶数。
'         mtxA    - Double型二维数组，体积为n x n，存放方阵。
'  返回值：Double型，行列式的值。
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MDetGauss(n As Integer, mtxA() As Double) As Double
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, nIs As Integer, nJs As Integer
    Dim f As Double, det As Double, q As Double, d As Double

    f = 1#
    det = 1#

    ' 选主元
    For k = 1 To n - 1
        q = 0#
        For i = k To n
            For j = k To n
                d = Abs(mtxA(i, j))
                If (d > q) Then
                    q = d
                    nIs = i
                    nJs = j
                End If
            Next j
        Next i

        ' 求解失败
        If (q + 1# = 1#) Then
            MDetGauss = 0
            Exit Function
        End If

        If (nIs <> k) Then
            f = -f
            For j = k To n
                d = mtxA(k, j)
                mtxA(k, j) = mtxA(nIs, j)
                mtxA(nIs, j) = d
            Next j
        End If

        ' 调整
        If (nJs <> k) Then
            f = -f
            For i = k To n
                d = mtxA(i, nJs)
                mtxA(i, nJs) = mtxA(i, k)
                mtxA(i, k) = d
            Next i
        End If

        ' 计算行列式的值
        det = det * mtxA(k, k)
```

```
        ' 调整方阵为上三角矩阵
        For i = k + 1 To n
            d = mtxA(i, k) / mtxA(k, k)
            For j = k + 1 To n
                mtxA(i, j) = mtxA(i, j) - d * mtxA(k, j)
            Next j
        Next i
    Next k

    ' 计算行列式的值
    det = f * det * mtxA(n, n)

    ' 求解成功
    MDetGauss = det

End Function
```

3. 示例

调用函数 MDetGauss **求下列** 2 **个方阵** $\boldsymbol{A}$ **与** $\boldsymbol{B}$ **的行列式值** $\det \boldsymbol{A}$ **与** $\det \boldsymbol{B}$**：**

$$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}3 & -3 & -2 & 4\\ 5 & -5 & 1 & 8\\ 11 & 8 & 5 & -7\\ 5 & -1 & -3 & -1\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 4) As Double
    Dim mtxB(4, 4) As Double
    Dim dblDetA As Double, dblDetB As Double

    mtxA(1, 1) = 1:  mtxA(1, 2) = 2:  mtxA(1, 3) = 3:  mtxA(1, 4) = 4
    mtxA(2, 1) = 5:  mtxA(2, 2) = 6:  mtxA(2, 3) = 7:  mtxA(2, 4) = 8
    mtxA(3, 1) = 9:  mtxA(3, 2) = 10:  mtxA(3, 3) = 11:  mtxA(3, 4) = 12
    mtxA(4, 1) = 13:  mtxA(4, 2) = 14:  mtxA(4, 3) = 15:  mtxA(4, 4) = 16

    mtxB(1, 1) = 3:  mtxB(1, 2) = -3:  mtxB(1, 3) = -2:  mtxB(1, 4) = 4
    mtxB(2, 1) = 5:  mtxB(2, 2) = -5:  mtxB(2, 3) = 1:  mtxB(2, 4) = 8
    mtxB(3, 1) = 11: mtxB(3, 2) = 8:  mtxB(3, 3) = 5:  mtxB(3, 4) = -7
    mtxB(4, 1) = 5:  mtxB(4, 2) = -1: mtxB(4, 3) = -3:  mtxB(4, 4) = -1

    '求解
    dblDetA = MDetGauss(4, mtxA)
    dblDetB = MDetGauss(4, mtxB)

    MsgBox "det(A) = " & dblDetA & Chr$(13) & Chr$(13) & "det(B) = " & dblDetB

End Sub
```

输出结果如图 2.7 所示。

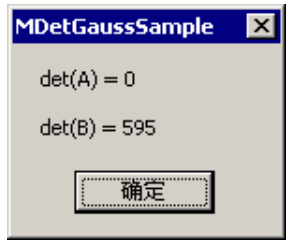

图 2.7　程序输出结果

2.7　求矩阵秩的全选主元高斯消去法

1. 算法原理

若矩阵 $\boldsymbol{A}$ 中，k 阶子式不等于零，而一切 $k+1$ 阶子式均为零，则称矩阵的秩为 k，记为 $\mathrm{rank}\boldsymbol{A}=k$ 或秩$(A)=k$，若 n 阶方阵 $\boldsymbol{A}$ 可逆，则 $\mathrm{rank}\boldsymbol{A}=n$。本节介绍用全选主元高斯消去法计算矩阵 $\boldsymbol{A}$ 的秩的算法。

设有 $m\times n$ 阶矩阵

$$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ a_{m1} & a_{m2} & \Lambda & a_{mn} \end{bmatrix}$$

取 $k=\min\{m,\ n\}$。对于 $r=1,\Lambda\ ,k$，用全选主元高斯消去法将 $\boldsymbol{A}$ 变为上三角矩阵，直到某次 $a_{r,\ r}=0$ 为止，矩阵 $\boldsymbol{A}$ 的秩为 r。

2. 算法实现

根据上述方法，可以定义用全选主元高斯消去法计算矩阵的秩的 Visual Basic 函数 MRank，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MRank
'  功能：  用全选主元高斯消去法求矩阵的秩
'  参数：  m       - Integer型变量，矩阵的行数。
'          n       - Integer型变量，矩阵的列数。
'          mtxA    - Double型二维数组，体积为m x n，存放待求秩的矩阵。
'  返回值：Integer型，矩阵的秩。
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MRank(m As Integer, n As Integer, mtxA() As Double) As Integer
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, l As Integer, _
        nIs As Integer, nJs As Integer, nn As Integer
    Dim q As Double, d As Double
```

```
' 基数
nn = m
If (m >= n) Then nn = n

' 消元求解
k = 0
For l = 1 To nn
   q = 0#
   For i = 2 To m
      For j = 2 To n
         d = Abs(mtxA(i, j))
         If (d > q) Then
            q = d
            nIs = i
            nJs = j
         End If
      Next j
   Next i

   ' 求解失败
   If (q + 1# = 1#) Then
      MRank = k
      Exit Function
   End If

   k = k + 1
   If (nIs <> l) Then
      For j = l To n
         d = mtxA(l, j)
         mtxA(l, j) = mtxA(nIs, j)
         mtxA(nIs, j) = d
      Next j
   End If

   If (nJs <> l) Then
      For i = l To m
         d = mtxA(i, nJs)
         mtxA(i, nJs) = mtxA(i, l)
         mtxA(i, l) = d
       Next i
   End If

   For i = l + 1 To n
      d = mtxA(i, l) / mtxA(l, l)
      For j = l + 1 To n
         mtxA(i, j) = mtxA(i, j) - d * mtxA(l, j)
      Next j
   Next i
Next l

' 求解成功
```

```
    MRank = k

End Function
```

3. 示例

调用函数 MRank **求解下列** 5×4 **阶矩阵** $\boldsymbol{A}$ **的秩：**

$$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\\ 17 & 18 & 19 & 20\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(5, 4) As Double
    Dim nRankA As Integer

    mtxA(1, 1)=1:   mtxA(1, 2)=2:   mtxA(1, 3)=3:   mtxA(1, 4) = 4
    mtxA(2, 1)=5:   mtxA(2, 2)=6:   mtxA(2, 3)=7:   mtxA(2, 4) = 8
    mtxA(3, 1)=9:   mtxA(3, 2)=10:  mtxA(3, 3)=11:  mtxA(3, 4) = 12
    mtxA(4, 1)=13:  mtxA(4, 2)=14:  mtxA(4, 3)=15:  mtxA(4, 4) = 16
    mtxA(5, 1)=-17: mtxA(5, 2)=18:  mtxA(5, 3)=19:  mtxA(5, 4) = 20

    '求解
    nRankA = MRank(5, 4, mtxA)

    MsgBox "Rank(A) = " & nRankA

End Sub
```

图 2.8　程序输出结果

输出结果如图 2.8 **所示。**

2.8　对称正定矩阵的乔里斯基分解与行列式的求值

1. 算法原理

本节介绍用乔里斯基(Cholesky)**分解法求对称正定矩阵的三角分解，并求行列式值的算法。**

设 n **阶矩阵** $\boldsymbol{A}$ **为对称正定，则存在一个实的非奇异的下三角阵** $\boldsymbol{L}$**，使** $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$**，其中，**

$$\boldsymbol{L}=\begin{bmatrix}l_{00} & & & \\ l_{10} & l_{11} & 0 & \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \\ l_{n-1,0} & l_{n-1,1} & \Lambda & l_{n-1,n-1}\end{bmatrix}$$

乔里斯基分解的步骤为：

对于 $j = 0,\ 1,\ \Lambda\ ,\ n-1$

(1) $l_{jj} = (a_{jj} - \sum_{k=0}^{j-1} l_{jk}^2)^{\frac{1}{2}}$

(2) $l_{ij} = (a_{ij} - \sum_{k=0}^{j-1} l_{ik} l_{jk}) / l_{jj}\ ,\ i = j+1,\ \Lambda\ ,\ n-1$

A **的行列式值为** $\det(\boldsymbol{A}) = (\prod_{k=0}^{n-1} l_{kk})^2$

2. **算法实现**

根据上述方法，可以实现用乔里斯基分解法求对称正定矩阵的三角分解，并定义行列式值的 Visual Basic **函数** MDetChol**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MDetChol
'  功能：  用乔里斯基分解法求对称正定矩阵行列式的值
'  参数：  n      - Integer型变量，对称正定矩阵的阶数。
'          mtxA   - Double型二维数组，体积为n x n，存放对称正定矩阵，
'                   返回时，其下三角部分存放分解后的下三角矩阵，其余元素为0。
'          dblDet - Double型变量，返回对称正定矩阵行列式的值。
'  返回值：Boolean型，成功为True，失败为False。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MDetChol(n As Integer, mtxA() As Double, det As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim d As Double

    ' 矩阵校验失败
    If ((mtxA(1, 1) + 1# = 1#) Or (mtxA(1, 1) < 0#)) Then
        MDetChol = False
        Exit Function
    End If

    ' 开始，赋初值
    mtxA(1, 1) = Sqr(mtxA(1, 1))
    d = mtxA(1, 1)

    For i = 2 To n
        mtxA(i, 1) = mtxA(i, 1) / mtxA(1, 1)
    Next i

    ' 循环计算
    For j = 2 To n
        For k = 1 To j - 1
            mtxA(j, j) = mtxA(j, j) - mtxA(j, k) * mtxA(j, k)
        Next k
```

```
        If ((mtxA(j, j) + 1# = 1#) Or (mtxA(j, j) < 0#)) Then
            MDetChol = False
            Exit Function
        End If

        mtxA(j, j) = Sqr(mtxA(j, j))
        d = d * mtxA(j, j)

        For i = j + 1 To n
            For k = 1 To j - 1
                mtxA(i, j) = mtxA(i, j) - mtxA(i, k) * mtxA(j, k)
            Next k

            mtxA(i, j) = mtxA(i, j) / mtxA(j, j)
        Next i
    Next j

    ' 计算行列式值
    det = d * d

    ' 下三角矩阵
    For i = 1 To n - 1
        For j = i + 1 To n
            mtxA(i, j) = 0#
        Next j
    Next i

    ' 求解成功
    MDetChol = True

End Function
```

3. 示例

调用函数 MDetChol **求下列** 4 **阶对称正定矩阵** $\boldsymbol{A}$ **的乔里斯基分解式，并求行列式值** $\det \boldsymbol{A}$**：**

$$\boldsymbol{A}=\begin{bmatrix}5 & 7 & 6 & 5\\ 7 & 10 & 8 & 7\\ 6 & 8 & 10 & 9\\ 5 & 7 & 9 & 10\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 4) As Double
    Dim nDetA As Double

    mtxA(1, 1) = 5:  mtxA(1, 2) = 7:  mtxA(1, 3) = 6:  mtxA(1, 4) = 5
    mtxA(2, 1) = 7:  mtxA(2, 2) = 10:  mtxA(2, 3) = 8:  mtxA(2, 4) = 7
    mtxA(3, 1) = 6:  mtxA(3, 2) = 8:  mtxA(3, 3) = 10:  mtxA(3, 4) = 9
```

```
    mtxA(4, 1) = 5:  mtxA(4, 2) = 7:  mtxA(4, 3) = 9:  mtxA(4, 4) = 10

    '求解
    If MDetChol(4, mtxA, nDetA) Then
       MsgBox "Det(A) = " & nDetA & Chr$(13) & Chr$(13) & _
              "分解后的下三角矩阵为" & Chr$(13) & Chr$(13) & _
              MatrixToString(4, 4, mtxA, "#######0.0000000")
    Else
       MsgBox "求解失败"
End If

End Sub
```

输出结果如图 2.9 所示。

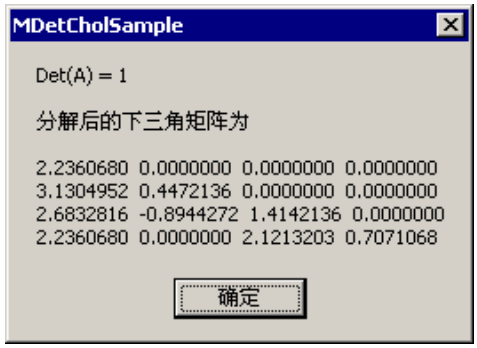

图 2.9 程序输出结果

2.9 矩阵的三角分解

1. 算法原理

对 n 阶实矩阵进行 LU 分解，即 $\boldsymbol{A}=\boldsymbol{LU}$，其中，

$$\boldsymbol{L}=\begin{bmatrix} 1 & & & & & \\ l_{10} & 1 & & & & \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & & & \\ l_{k0} & l_{k1} & \Lambda & 1 & & \\ \mathrm{M} & \mathrm{M} & & \mathrm{M} & \mathrm{O} & \\ l_{n-1,\ 0} & l_{n-1,\ 1} & \Lambda & l_{n-1,\ k} & \Lambda & 1 \end{bmatrix}$$

$$\boldsymbol{U}=\begin{bmatrix} u_{00} & u_{01} & \Lambda & u_{0,\ n-1} \\ & u_{11} & \Lambda & u_{1,\ n-1} \\ & & \mathrm{O} & \mathrm{M} \\ & & & u_{n-1,\ n-1} \end{bmatrix}$$

令：

$$
\boldsymbol{Q} = \boldsymbol{L} + \boldsymbol{U} - \boldsymbol{I}_n = \begin{bmatrix} u_{00} & u_{01} & \Lambda & u_{0k} & \Lambda & u_{0,\ n-1} \\ l_{10} & u_{11} & \Lambda & u_{1k} & \Lambda & u_{1,\ n-1} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ l_{k0} & l_{k1} & \Lambda & u_{kk} & \Lambda & u_{k,\ n-1} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ l_{n-1,\ 0} & l_{n-1,\ 1} & \Lambda & l_{n-1,\ k} & \Lambda & u_{n-1,\ n-1} \end{bmatrix}
$$

则 LU **分解问题变为求矩阵** Q **的问题。**

由 n **阶实矩阵求矩阵** Q **的计算步骤为：**

对于 $k = 0,\ 1,\ \Lambda\ ,\ n-2$

(1)　$a_{ik} / a_{kk} \Rightarrow a_{ik}$,　$i = k+1,\ \Lambda\ ,\ n-1$

(2)　$a_{ij} - a_{ik}a_{kj} \Rightarrow a_{ij}$,　$i,\ j = k+1,\ \Lambda\ ,\ n-1$

由于本方法没有选主元，因此数值计算是不稳定的。

2.. **算法实现**

根据上述方法，可以定义矩阵的三角分解的 Visual Basic **函数** MLU**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MLU
'  功能：  矩阵的三角分解
'  参数：  n       - Integer型变量，矩阵的阶数。
'          mtxA    - Double型二维数组，体积为n x n，存放n阶矩阵，返回时存放Q矩阵。
'          mtxL    - Double型二维数组，体积为n x n，返回时存放下三角矩阵L。
'          mtxU    - Double型二维数组，体积为n x n，返回时存放上三角矩阵U。
'  返回值：Boolean型，成功为True，失败为False。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MLU(n As Integer, mtxA() As Double, mtxL() As Double, mtxU() As Double)
As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer

    For k = 1 To n - 1
        ' 分解失败
        If (Abs(mtxA(k, k)) + 1# = 1#) Then
            MLU = False
            Exit Function
        End If

        For i = k + 1 To n
            mtxA(i, k) = mtxA(i, k) / mtxA(k, k)
        Next i

        For i = k + 1 To n
```

```
            For j = k + 1 To n
               mtxA(i, j) = mtxA(i, j) - mtxA(i, k) * mtxA(k, j)
            Next j
         Next i
      Next k

      For i = 1 To n
         For j = 1 To i
            mtxL(i, j) = mtxA(i, j)
            mtxU(i, j) = 0#
         Next j

         mtxL(i, i) = 1#
         mtxU(i, i) = mtxA(i, i)
         For j = i + 1 To n
            mtxL(i, j) = 0#
            mtxU(i, j) = mtxA(i, j)
         Next j
      Next i

      ' 分解成功
      MLU = True
End Function
```

3. 示例

调用函数 MLU **求下列** 4 **阶矩阵的** LU **分解：**

$$\boldsymbol{A}=\begin{bmatrix}2 & 4 & 4 & 2\\ 3 & 3 & 12 & 6\\ 2 & 4 & -1 & 2\\ 4 & 2 & 1 & 1\end{bmatrix}$$

程序代码如下：

```
Sub Main()
   Dim mtxA(4, 4) As Double
   Dim mtxL(4, 4) As Double
   Dim mtxU(4, 4) As Double

   ' 原矩阵
   mtxA(1, 1) = 2:  mtxA(1, 2) = 4:  mtxA(1, 3) = 4:  mtxA(1, 4) = 2
   mtxA(2, 1) = 3:  mtxA(2, 2) = 3:  mtxA(2, 3) = 12:  mtxA(2, 4) = 6
   mtxA(3, 1) = 2:  mtxA(3, 2) = 4:  mtxA(3, 3) = -1:  mtxA(3, 4) = 2
   mtxA(4, 1) = 4:  mtxA(4, 2) = 2:  mtxA(4, 3) = 1:  mtxA(4, 4) = 1

   '求解
   If MLU(4, mtxA, mtxL, mtxU) Then
      MsgBox "分解后的下三角矩阵L为" & Chr$(13) & Chr$(13) & _
```

```
            MatrixToString(4, 4, mtxL, "######0.0000000") & _
            Chr$(13) & Chr$(13) & "分解后的上三角矩阵U为" & Chr$(13) & _
            Chr$(13) & MatrixToString(4, 4, mtxU, "######0.0000000")
    Else
        MsgBox "求解失败"
    End If

End Sub
```

输出结果如图 2.10 **所示。**

图 2.10　程序输出结果

2.10　一般实矩阵的 QR 分解

1. **算法原理**

设 $m\times n$ **的实矩阵** $\boldsymbol{A}$ **列线性无关，则可以将** $\boldsymbol{A}$ **分解为** $\boldsymbol{A}=\boldsymbol{QR}$ **的形式。其中** $\boldsymbol{Q}$ **为** $m\times n$ **的正交矩阵，** $\boldsymbol{R}$ **为** $m\times n$ **的上三角矩阵。**

豪斯荷尔德(Householder)**变换是对一般** $m\times n$ **阶的实矩阵进行** $\boldsymbol{QR}$ **分解的重要方法。豪斯荷尔德方法对** $\boldsymbol{A}$ **进行** $\boldsymbol{QR}$ **分解的步骤如下：**

令 $s=\min(m-1,\ n)$，　$\boldsymbol{Q}=\boldsymbol{I}_{m\times m}$

对于 k **从** 0 **到** s-1 **作以下几步：**

(1)　**确定豪斯荷尔德变换：**

$$\boldsymbol{Q}^{(k)}=\begin{bmatrix}\boldsymbol{I}_k & 0\\ 0 & \tilde{\boldsymbol{Q}}_{m-k}\end{bmatrix}$$

其中，

$$\tilde{\boldsymbol{Q}}_{m-k}=\begin{bmatrix} (1-2u_k^2) & -2u_k u_{k+1} & \Lambda & -2u_k u_{m-1} \\ -2u_{k+1}u_k & (1-2u_{k+1}^2 & \Lambda & -2u_{k+1}u_{m-1} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ -2u_{m-1}u_k & -2u_{m-1}u_{k+1} & \Lambda & (1-2u_{m-1}^2) \end{bmatrix}$$

矩阵中 $u_j\,(j=k,\ k+1,\ \Lambda\ ,\ m-1)$ 的计算公式如下：

$$\eta=\max_{k\le i\le m-1}|a_{ik}|$$

$$\alpha=-\mathrm{sign}(a_{kk})\sqrt{\sum_{i=k}^{m-1}(a_{ik}/\eta)^2}\cdot\eta$$

$$\rho=\sqrt{2\alpha(\alpha-a_{kk})}$$

$$u_k=\frac{1}{\rho}(a_{kk}-\alpha)\Rightarrow a_{kk}$$

$$u_i=\frac{1}{\rho}a_{ik}\Rightarrow a_{ik},\quad i=k+1,\Lambda\ ,m-1$$

(2)　用 $\boldsymbol{Q}^{(k)}$左乘 $\boldsymbol{Q}$。即 $\boldsymbol{Q}^{(k)}\boldsymbol{Q}\Rightarrow\boldsymbol{Q}$

其计算公式如下：

$$\left.\begin{aligned} & t=\sum_{l=k}^{m-1}a_{lk}\cdot q_{l} \\ & q_{i}-2a_{ik}t\Rightarrow q_j \end{aligned}\right\}\quad \begin{aligned} & j=0_j,\ 1,\ \Lambda\ ,\ m-1 \\ & i=k,\ \underset{j}{k}+1,\ \Lambda\ ,\ m-1 \end{aligned}$$

(3)　用 $\boldsymbol{Q}^{(k)}$左乘 $\boldsymbol{A}$。即 $\boldsymbol{Q}^{(k)}\ \boldsymbol{A}\Rightarrow\boldsymbol{A}$

其计算公式如下：

$$\left.\begin{aligned} & t=\sum_{l=k}^{m-1}a_{lk}\cdot a_{lj} \\ & a_{ij}-2a_{ik}t\Rightarrow a_{ij} \end{aligned}\right\}\quad \begin{aligned} & j=k+1,\Lambda\ ,n-1 \\ & i=k,\Lambda\ ,m-1 \end{aligned}$$

$$\alpha\Rightarrow a_{kk}$$

经过上述运算，矩阵 $\boldsymbol{A}$ 的右上三角部分即为 $\boldsymbol{R}$，而矩阵 $\boldsymbol{Q}^{\mathrm{T}}$ 为正交矩阵。

2. 算法实现

根据上述方法，可以定义用豪斯荷尔德方法对矩阵进行 QR 分解的 Visual Basic 函数 MMqr，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MMqr
'  功能：  用豪斯荷尔德变换法对矩阵进行QR分解
'  参数：   m    - Integer型变量。矩阵的行数， m>=n
```

```
'          n    - Integer型变量。矩阵的列数，n<=m
'         dblA  - Double型二维数组，体积维n x n。
'                 存放待分解矩阵；返回时，存放分解式中的R矩阵.
'         dblQ  - Double型二维数组，体积为m x m。返回时，存放分解式中的Q矩阵
'  返回值： Boolean型。False，失败；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MMqr(m As Integer, n As Integer, dblA() As Double, _
dblQ() As Double) As Boolean
    Dim i As Integer, j As Integer, k As Integer, nn As Integer, jj As Integer
    Dim u As Double, alpha As Double, w As Double, t As Double

    If (m < n) Then
        MMqr = False
        Exit Function
    End If

    For i = 1 To m
        For j = 1 To m
            dblQ(i, j) = 0#
            If (i = j) Then
                dblQ(i, j) = 1#
            End If
        Next j
    Next i

    nn = n
    If (m = n) Then
        nn = m - 1
    End If

    For k = 1 To nn
        u = 0#
        For i = k To m
            w = Abs(dblA(i, k))
            If (w > u) Then
                u = w
            End If
        Next i

        alpha = 0#
        For i = k To m
            t = dblA(i, k) / u
            alpha = alpha + t * t
        Next i

        If (dblA(k, k) > 0#) Then
            u = -u
        End If

        alpha = u * Sqr(alpha)
```

```
        If (Abs(alpha) + 1# = 1#) Then
            MMqr = False
            Exit Function
        End If

        u = Sqr(2# * alpha * (alpha - dblA(k, k)))
        If ((u + 1#) <> 1#) Then
            dblA(k, k) = (dblA(k, k) - alpha) / u
            For i = k + 1 To m
                dblA(i, k) = dblA(i, k) / u
            Next i

            For j = 1 To m
                t = 0#
                For jj = k To m
                    t = t + dblA(jj, k) * dblQ(jj, j)
                Next jj
                For i = k To m
                    dblQ(i, j) = dblQ(i, j) - 2# * t * dblA(i, k)
                Next i
            Next j

            For j = k + 1 To n
                t = 0#
                For jj = k To m
                    t = t + dblA(jj, k) * dblA(jj, j)
                Next jj
                For i = k To m
                    dblA(i, j) = dblA(i, j) - 2# * t * dblA(i, k)
                Next i
            Next j

            dblA(k, k) = alpha
            For i = k + 1 To m
                dblA(i, k) = 0#
            Next i
        End If
    Next k

    For i = 1 To m - 1
        For j = i + 1 To m
            t = dblQ(i, j)
            dblQ(i, j) = dblQ(j, i)
            dblQ(j, i) = t
        Next j
    Next i

    MMqr = True
End Function
```

3. 示例

调用函数 MMqr **对下列** 4×3 **阶的矩阵** A **进行** QR **分解：**

$$A=\begin{bmatrix}1 & 1 & -1\\ 2 & 1 & 0\\ 1 & -1 & 0\\ -1 & 2 & 1\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 3) As Double
    Dim mtxQ(4, 4) As Double

    mtxA(1, 1) = 1: mtxA(1, 2) = 1: mtxA(1, 3) = -1
    mtxA(2, 1) = 2: mtxA(2, 2) = 1: mtxA(2, 3) = 0
    mtxA(3, 1) = 1: mtxA(3, 2) = -1: mtxA(3, 3) = 0
    mtxA(4, 1) = -1: mtxA(4, 2) = 2: mtxA(4, 3) = 1

    '求解
    If MMqr(4, 3, mtxA, mtxQ) Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "矩阵Q" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 4, mtxQ, "######0.0000000") & Chr$(13) _
               & Chr$(13) & "矩阵A" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 3, mtxA, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

输出结果如图 2.11 **所示。**

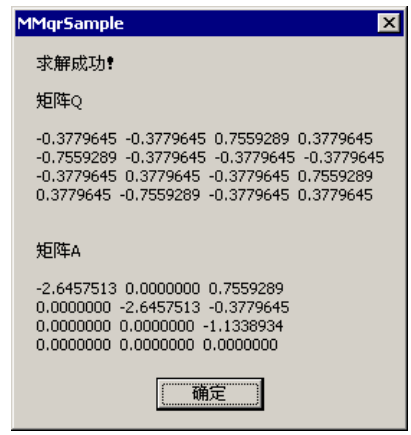

图 2.11　程序输出结果

2.11 一般实矩阵的奇异值分解

1. 算法原理

本节介绍用豪斯荷尔德(Householder)变换及变形 QR 算法对一般实矩阵进行奇异值分解的算法。

设 $\boldsymbol{A}$ 为 $m\times n$ 的实矩阵,则存在一个 $m\times m$ 的列正交矩阵 $\boldsymbol{U}$ 和为 $n\times n$ 的列正交矩阵 $\boldsymbol{V}$,使:

$$\boldsymbol{A}=\boldsymbol{U}\begin{bmatrix}\Sigma & 0\\ 0 & 0\end{bmatrix}\mathrm{V}^{\mathrm{T}}$$

成立。其中,$\Sigma=\mathrm{diag}(\sigma_0,\sigma_1,\Lambda\ ,\sigma_p)(p\le\min(m,n)-1)$ 且 $\sigma_0\ge\sigma_1\ge\Lambda\ \ge\sigma_p>0$。

上式称为实矩阵 $\boldsymbol{A}$ 的奇异值分解式,$\sigma_i(i=0,1,\Lambda\ ,p)$ 称为 $\boldsymbol{A}$ 的奇异值。

利用 $\boldsymbol{A}$ 的奇异值分解式,可以计算 $\boldsymbol{A}$ 的广义逆 $\boldsymbol{A}^{+}$,利用 $\boldsymbol{A}$ 的广义逆可以求解线性最小二乘问题。

奇异值分解分两大步:

(1) 用豪斯荷尔德变换将 $\boldsymbol{A}$ 约化为双对角线矩阵。即:

$$\boldsymbol{B}=\tilde{U}^{\mathrm{T}}\boldsymbol{A}\tilde{V}=\begin{bmatrix}s_0 & \mathrm{e}_0 & & & \\ & s_1 & \mathrm{e}_1 & 0 & \\ & & \mathrm{O} & \mathrm{O} & \\ & 0 & & s_{p-1} & \mathrm{e}_{p-1}\\ & & & & s_p\end{bmatrix}$$

其中,

$\tilde{U}=U_0U_1\Lambda\ U_{k-1},\ \ k=\min(n,m-1)$

$\tilde{V}=V_0V_1\Lambda\ V_{l-1},\ \ l=\min(m,n-2)$

$\tilde{U}$ 中的每一个变换 $U_j(j=0,1,\Lambda\ ,k-1)$ 将 $\boldsymbol{A}$ 中第 j 列主对角线以下的元素变为零 ;而 $\tilde{V}$ 中的每一个变换 $V_j(j=0,1,\Lambda\ ,l-1)$ 将 $\boldsymbol{A}$ 中的第 j 行中与主对角线紧邻的右次对角线元素右边的元素变为零。

对于每一个变换 V_j 具有如下形式:

$I-\rho V_jV_j^{\mathrm{T}}$

其中 ρ 为比例因子,用来避免计算过程中的溢出和误差累计。V_j 是一个列向量,即:

$V_j=(v_0,v_1,\Lambda\ ,v_{n-1})^{\mathrm{T}}$

则:$\boldsymbol{A}V_j=\boldsymbol{A}-\rho\boldsymbol{A}V_jV_j^{\mathrm{T}}=\boldsymbol{A}=-WV_j^{\mathrm{T}}$

其中，

$$W=\rho \boldsymbol{A} V_j=\rho(\sum_{i=0}^{n-1} v_i a_{0i},\ \sum_{i=0}^{n-1} v_i a_{1i}, \Lambda\ , \sum_{i=0}^{n-1} v_i a_{m-1,\,i})^{\mathrm{T}}$$

(2) 用变形的 QR 算法进行迭代，计算所有的奇异值。即：用一系列的平面旋转变换将双对角线矩阵 $\boldsymbol{B}$ 逐步变成对角矩阵。

在每一次的迭代中，用变换

$$\boldsymbol{B}^{'}=U_{p-1,\ \ p}^{\mathrm{T}}\Lambda\ U_{12}^{\mathrm{T}}U_{01}^{\mathrm{T}}\boldsymbol{B}V_{01}V_{12}\Lambda\ V_{m-2,\,m-1}$$

其中交换 $U_{j,j+1}^{\mathrm{T}}$ 将 B 中第 j 列主对角线下的一个非零元素变为 0，同时在第 j 行的次对角线元素的右边出现一个非零元素；而变换 $V_{j,j+1}$ 将第 j-1 行的次对角线元素右边的一个非零元素变为零，同时在第 j 列的主对角线元素的下方出现一个非零元素。由此可知，经过一次迭代 $(j=0,1,\Lambda\ , p-1)$ 后，$\boldsymbol{B}'$ 仍为双对角线矩阵。但随着迭代的进行，最后收敛为对角矩阵，其对角线上的元素即为奇异值。

在每次迭代时，经过初始变换 V_{01} 后，将在第 0 列的主对角线下方出现一个非零元素。在变换 V_{01} 中，选择位移值 u 的计算公式如下：

$$b=\left[(s_{p-1}+s_p)(s_{p-1}-s_p)+\mathrm{e}_{p-1}^2\right]/2$$

$$c=(s_p\mathrm{e}_{p-1})^2$$

$$d=\mathrm{sign}(b)\sqrt{b^2+c}$$

$$c=s_p^2-c/(b+d)$$

最后还需要对奇异值按非递增次序进行排序。

在上述变换过程中，若对于某个次对角线元素 e_j 满足 $|\mathrm{e}_j|\le\varepsilon(|s_{j+1}|+|s_j|)$ 则可以认为 e_j 为零；

若对角线元素 s_j 满足 $|s_j|\le\varepsilon(|\mathrm{e}_{j-1}|+|\mathrm{e}_j|)$，则可以认为 s_j 为零(即为零奇异值)。其中 ε 为给定的精度要求。

2. 算法实现

根据上述方法，可以实现用豪斯荷尔德变换及变形 QR 算法对一般实矩阵进行奇异值分解算法的 Visual Basic 函数 MUav，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MUav
'  功能：  用豪斯荷尔德变换及变形QR算法对矩阵进行奇异值分解
'  参数：   m    - Integer型变量。系数矩阵的行数， m>=n
'           n    - Integer型变量。系数矩阵的列数，n<=m
'          dblA  - Double型二维数组，体积为m x n。存放待分解矩阵；
'                        返回时，其对角线存放矩阵的奇异值(以非递增次序排列)，其余元素为0。
'          dblU  - Double型二维数组，体积为m x m。返回时，存放奇异值分解式中的
```

```
'                左奇异向量U。
'         dblV  - Double型二维数组，体积为n x n。返回时，存放奇异值分解式中的
'                右奇异向量VT。
'          ka  - Integer型变量。ka=max(m,n)+1
'         eps  - Double型变量。奇异值分解函数中的控制精度参数。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MUav(m As Integer, n As Integer, dblA() As Double, _
dblU() As Double, dblV() As Double, ka As Integer, eps As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, l As Integer, it As Integer
    Dim ll As Integer, kk As Integer, mm As Integer, nn As Integer, _
         m1 As Integer, ks As Integer
    Dim d As Double, dd As Double, t As Double, sm As Double, _
         sm1 As Double, em1 As Double, sk As Double, ek As Double
    Dim b As Double, c As Double, shh As Double, fg(2) As Double, _
         cs(2) As Double
    ReDim s(ka) As Double, e(ka) As Double, w(ka) As Double

    ' 迭代次数为60
    it = 60
    k = n

    If (m - 1 < n) Then
       k = m - 1
    End If

    l = m
    If (n - 2 < m) Then
       l = n - 2
    End If
    If (l < 0) Then
       l = 0
    End If

    ll = k

    If (l > k) Then
       ll = l
    End If

    If (ll >= 1) Then
       For kk = 1 To ll
          If (kk <= k) Then
             d = 0#
             For i = kk To m
                d = d + dblA(i, kk) * dblA(i, kk)
             Next i

             s(kk) = Sqr(d)
             If s(kk) <> 0# Then
```

```
            If (dblA(kk, kk) <> 0#) Then
                s(kk) = Abs(s(kk))
                If (dblA(kk, kk) < 0#) Then
                    s(kk) = -s(kk)
                End If
            End If
            For i = kk To m
                dblA(i, kk) = dblA(i, kk) / s(kk)
            Next i
            dblA(kk, kk) = 1# + dblA(kk, kk)
        End If
        s(kk) = -s(kk)
    End If

    If (n >= kk + 1) Then
        For j = kk + 1 To n
            If ((kk <= k) And (s(kk) <> 0#)) Then
                d = 0#
                For i = kk To m
                    d = d + dblA(i, kk) * dblA(i, j)
                Next i
                d = -d / dblA(kk, kk)
                For i = kk To m
                    dblA(i, j) = dblA(i, j) + d * dblA(i, kk)
                Next i
            End If
            e(j) = dblA(kk, j)
        Next j
    End If

    If (kk <= k) Then
        For i = kk To m
            dblU(i, kk) = dblA(i, kk)
        Next i
    End If

    If (kk <= l) Then
        d = 0#
        For i = kk + 1 To n
            d = d + e(i) * e(i)
        Next i

        e(kk) = Sqr(d)
        If (e(kk) <> 0#) Then
            If (e(kk + 1) <> 0#) Then
                e(kk) = Abs(e(kk))
                If (e(kk + 1) < 0#) Then
                    e(kk) = -e(kk)
                End If
            End If
            For i = kk + 1 To n
```

```
                e(i) = e(i) / e(kk)
            Next i
            e(kk + 1) = 1# + e(kk + 1)
        End If

        e(kk) = -e(kk)
        If ((kk + 1 <= m) And (e(kk) <> 0#)) Then
            For i = kk + 1 To m
                w(i) = 0#
            Next i
            For j = kk + 1 To n
                For i = kk + 1 To m
                    w(i) = w(i) + e(j) * dblA(i, j)
                Next i
            Next j
            For j = kk + 1 To n
                For i = kk + 1 To m
                    dblA(i, j) = dblA(i, j) - w(i) * e(j) / e(kk + 1)
                Next i
            Next j
        End If
        For i = kk + 1 To n
            dblV(i, kk) = e(i)
        Next i
    End If
Next kk
End If

mm = n
If (m + 1 < n) Then mm = m + 1
If (k < n) Then s(k + 1) = dblA(k + 1, k + 1)
If (m < mm) Then s(mm) = 0#
If (l + 1 < mm) Then e(l + 1) = dblA(l + 1, mm)

e(mm) = 0#
nn = m
If (m > n) Then nn = n
If (nn >= k + 1) Then
    For j = k + 1 To nn
        For i = 1 To m
            dblU(i, j) = 0#
        Next i
        dblU(j, j) = 1#
    Next j
End If

If (k >= 1) Then
    For ll = 1 To k
        kk = k - ll + 1
        If (s(kk) <> 0#) Then
            If (nn >= kk + 1) Then
```

```
                For j = kk + 1 To nn
                    d = 0#
                    For i = kk To m
                        d = d + dblU(i, kk) * dblU(i, j) / dblU(kk, kk)
                    Next i
                    d = -d
                    For i = kk To m
                        dblU(i, j) = dblU(i, j) + d * dblU(i, kk)
                    Next i
                Next j
            End If

            For i = kk To m
                dblU(i, kk) = -dblU(i, kk)
            Next i

            dblU(kk, kk) = 1# + dblU(kk, kk)
            If (kk - 1 >= 1) Then
                For i = 1 To kk - 1
                  dblU(i, kk) = 0#
                Next i
            End If
        Else
            For i = 1 To m
                dblU(i, kk) = 0#
            Next i
            dblU(kk, kk) = 1#
        End If
    Next ll
End If

For ll = 1 To n
    kk = n - ll + 1
    If ((kk <= l) And (e(kk) <> 0#)) Then
        For j = kk + 1 To n
            d = 0#
            For i = kk + 1 To n
                d = d + dblV(i, kk) * dblV(i, j) / dblV(kk + 1, kk)
            Next i
            d = -d
            For i = kk + 1 To n
                dblV(i, j) = dblV(i, j) + d * dblV(i, kk)
            Next i
        Next j
    End If

    For i = 1 To n
        dblV(i, kk) = 0#
    Next i

    dblV(kk, kk) = 1#
```

```
Next ll

For i = 1 To m
   For j = 1 To n
      dblA(i, j) = 0#
   Next j
Next i

m1 = mm
it = 60
While (True)
   ' 计算成功，返回
   If (mm = 0) Then
      Call Cal1(dblA, e, s, dblV, m, n)
      MUav = True
      Exit Function
   End If

   ' 计算失败，返回
   If (it = 0) Then
      Call Cal1(dblA, e, s, dblV, m, n)
      MUav = False
      Exit Function
   End If

   ' 开始下一次迭代
   kk = mm - 1
   While ((kk <> 0) And (Abs(e(kk)) <> 0#))
      d = Abs(s(kk)) + Abs(s(kk + 1))
      dd = Abs(e(kk))
      If (dd > eps * d) Then
         kk = kk - 1
      Else
         e(kk) = 0#
      End If
   Wend

   If (kk = mm - 1) Then
      kk = kk + 1
      If (s(kk) < 0#) Then
         s(kk) = -s(kk)
         For i = 1 To n
            dblV(i, kk) = -dblV(i, kk)
         Next i
      End If

      While ((kk <> m1) And (s(kk) < s(kk + 1)))
         d = s(kk)
         s(kk) = s(kk + 1)
         s(kk + 1) = d
         If (kk < n) Then
```

```
            For i = 1 To n
                d = dblV(i, kk)
                dblV(i, kk) = dblV(i, kk + 1)
                dblV(i, kk + 1) = d
            Next i
        End If
        If (kk < m) Then
            For i = 1 To m
                d = dblU(i, kk)
                dblU(i, kk) = dblU(i, kk + 1)
                dblU(i, kk + 1) = d
            Next i
        End If
        kk = kk + 1
    Wend
    it = 60
    mm = mm - 1
Else
    ks = mm
    While ((ks > kk) And (Abs(s(ks)) <> 0#))
        d = 0#
        If (ks <> mm) Then d = d + Abs(e(ks))
        If (ks <> kk + 1) Then d = d + Abs(e(ks - 1))
        dd = Abs(s(ks))
        If (dd > eps * d) Then
            ks = ks - 1
        Else
            s(ks) = 0#
        End If
    Wend
    If (ks = kk) Then
        kk = kk + 1
        d = Abs(s(mm))
        t = Abs(s(mm - 1))
        If (t > d) Then d = t
        t = Abs(e(mm - 1))
        If (t > d) Then d = t
        t = Abs(s(kk))
        If (t > d) Then d = t
        t = Abs(e(kk))
        If (t > d) Then d = t
        sm = s(mm) / d
        sm1 = s(mm - 1) / d
        em1 = e(mm - 1) / d
        sk = s(kk) / d
        ek = e(kk) / d
        b = ((sm1 + sm) * (sm1 - sm) + em1 * em1) / 2#
        c = sm * em1
        c = c * c
        shh = 0#
        If ((b <> 0#) Or (c <> 0#)) Then
```

```
            shh = Sqr(b * b + c)
            If (b < 0#) Then shh = -shh
            shh = c / (b + shh)
         End If
         fg(1) = (sk + sm) * (sk - sm) - shh
         fg(2) = sk * ek
         For i = kk To mm - 1
            Call Cal2(fg, cs)
            If (i <> kk) Then e(i - 1) = fg(1)
            fg(1) = cs(1) * s(i) + cs(2) * e(i)
            e(i) = cs(1) * e(i) - cs(2) * s(i)
            fg(2) = cs(2) * s(i + 1)
            s(i + 1) = cs(1) * s(i + 1)
            If ((cs(1) <> 1#) Or (cs(2) <> 0#)) Then
               For j = 1 To n
                 d = cs(1) * dblV(j, i) + cs(2) * dblV(j, i + 1)
                 dblV(j, i + 1) = -cs(2) * dblV(j, i) + _
                                  cs(1) * dblV(j, i + 1)
                 dblV(j, i) = d
               Next j
            End If
            Call Cal2(fg, cs)
            s(i) = fg(1)
            fg(1) = cs(1) * e(i) + cs(2) * s(i + 1)
            s(i + 1) = -cs(2) * e(i) + cs(1) * s(i + 1)
            fg(2) = cs(2) * e(i + 1)
            e(i + 1) = cs(1) * e(i + 1)
            If (i < m) Then
               If ((cs(1) <> 1#) Or (cs(2) <> 0#)) Then
                  For j = 1 To m
                     d = cs(1) * dblU(j, i) + cs(2) * dblU(j, i + 1)
                     dblU(j, i + 1) = -cs(2) * dblU(j, i) + _
                                      cs(1) * dblU(j, i + 1)
                     dblU(j, i) = d
                  Next j
               End If
            End If
         Next i
         e(mm - 1) = fg(1)
         it = it - 1
      Else
         If (ks = mm) Then
            kk = kk + 1
            fg(2) = e(mm - 1)
            e(mm - 1) = 0#
            For ll = kk To mm - 1
               i = mm + kk - ll - 1
               fg(1) = s(i)
               Call Cal2(fg, cs)
               s(i) = fg(1)
               If (i <> kk) Then
```

```
                    fg(2) = -cs(2) * e(i - 1)
                    e(i - 1) = cs(1) * e(i - 1)
                End If
                If ((cs(1) <> 1#) Or (cs(2) <> 0#)) Then
                  For j = 1 To n
                     d = cs(1) * dblV(j, i) + cs(2) * dblV(j, mm)
                     dblV(j, mm) = -cs(2) * dblV(j, i) + _
                                   cs(1) * dblV(j, mm)
                     dblV(j, i) = d
                   Next j
                End If
             Next ll
          Else
             kk = ks + 1
             fg(2) = e(kk - 1)
             e(kk - 1) = 0#
             For i = kk To mm
                fg(1) = s(i)
                Call Cal2(fg, cs)
                s(i) = fg(1)
                fg(2) = -cs(2) * e(i)
                e(i) = cs(1) * e(i)
                If ((cs(1) <> 1#) Or (cs(2) <> 0#)) Then
                  For j = 1 To m
                     d = cs(1) * dblU(j, i) + cs(2) * dblU(j, kk - 1)
                     dblU(j, kk - 1) = -cs(2) * dblU(j, i) + _
                                       cs(1) * dblU(j, kk - 1)
                     dblU(j, i) = d
                  Next j
                End If
             Next i
          End If
       End If
    End If
 Wend

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：Cal1
'  功能：  内部过程，供MUav函数调用
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub Cal1(dblA() As Double, e() As Double, s() As Double, _
dblV() As Double, m As Integer, n As Integer)
   Dim i As Integer, j As Integer, p As Integer, q As Integer
   Dim d As Double

   If (m >= n) Then
      i = n
   Else
```

```
        i = m
    End If

    For j = 1 To i - 1
        dblA(j, j) = s(j)
        dblA(j, j + 1) = e(j)
    Next j

    dblA(i, i) = s(i)

    If (m < n) Then dblA(i, i + 1) = e(i)

    For i = 1 To n - 1
        For j = i + 1 To n
            d = dblV(i, j)
            dblV(i, j) = dblV(j, i)
            dblV(j, i) = d
        Next j
    Next i
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：Cal2
'  功能：  内部过程，供MUav函数调用
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub Cal2(fg() As Double, cs() As Double)
    Dim r As Double, d As Double

    If ((Abs(fg(1)) + Abs(fg(2))) = 0#) Then
        cs(1) = 1#
        cs(2) = 0#
        d = 0#
    Else
        d = Sqr(fg(1) * fg(1) + fg(2) * fg(2))
        If (Abs(fg(1)) > Abs(fg(2))) Then
            d = Abs(d)
            If (fg(1) < 0#) Then d = -d
        End If

        If (Abs(fg(2)) >= Abs(fg(1))) Then
            d = Abs(d)
            If (fg(2) < 0#) Then d = -d
        End If
        cs(1) = fg(1) / d
        cs(2) = fg(2) / d
    End If

    r = 1#
    If (Abs(fg(1)) > Abs(fg(2))) Then
```

```
        r = cs(2)
    Else
        If (cs(1) <> 0#) Then
            r = 1# / cs(1)
        End If
    End If

    fg(1) = d
    fg(2) = r

End Sub
```

3. 示例

调用函数 MUav **求下列矩阵** $\boldsymbol{A}$ **的奇异值分解式** UAV (**取** ε =0.000001)。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 1 & -1 & 0 \\ -1 & 2 & 1 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(4, 3) As Double
    Dim mtxU(4, 4) As Double
    Dim mtxV(3, 3) As Double
    Dim mtxB(4, 3) As Double
    Dim mtxC(4, 3) As Double

    ' 原矩阵赋值
    mtxA(1, 1) = 1: mtxA(1, 2) = 1: mtxA(1, 3) = -1
    mtxA(2, 1) = 2: mtxA(2, 2) = 1: mtxA(2, 3) = 0
    mtxA(3, 1) = 1: mtxA(3, 2) = -1: mtxA(3, 3) = 0
    mtxA(4, 1) = -1: mtxA(4, 2) = 2: mtxA(4, 3) = 1

    '求解
    If MUav(4, 3, mtxA, mtxU, mtxV, 5, 0.000001) Then

        ' 矩阵乘法 U*A
        Call MMul(4, 4, 3, mtxU, mtxA, mtxB)
        ' 矩阵乘法 (U*A)*V
        Call MMul(4, 3, 3, mtxB, mtxV, mtxC)

        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "矩阵U" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 4, mtxU, "######0.0000000") & Chr$(13) & _
               "矩阵V" & Chr$(13) & Chr$(13) & _
               MatrixToString(3, 3, mtxV, "######0.0000000") & Chr$(13) & _
               "奇异值矩阵" & Chr$(13) & Chr$(13) & _
```

```
            MatrixToString(4, 3, mtxA, "######0.0000000") & Chr$(13) & _
            "UAV" & Chr$(13) & Chr$(13) & _
            MatrixToString(4, 3, mtxC, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

输出结果如图 2.12 所示。

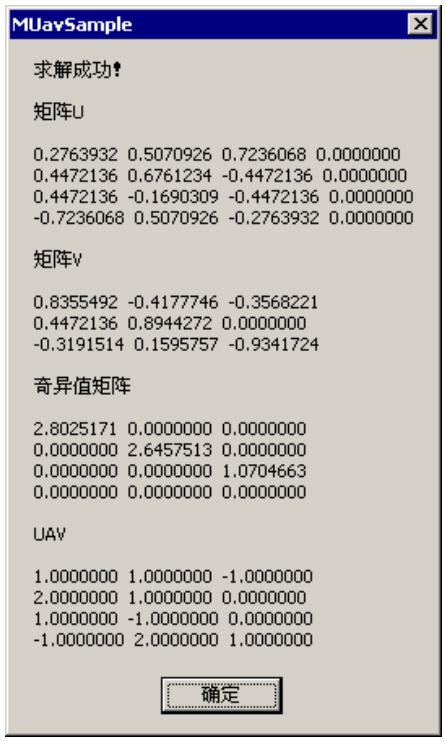

图 2.12 程序输出结果

2.12 求广义逆的奇异值分解法

1. 算法原理

设 $\boldsymbol{A}$ 为 $m\times n$ 的实矩阵，则存在一个 $m\times m$ 的列正交矩阵 $\boldsymbol{U}$ 和为 $n\times n$ 的列正交矩阵 $\boldsymbol{V}$，使

$$\boldsymbol{A}=\boldsymbol{U}\begin{bmatrix}\Sigma & 0\\ 0 & 0\end{bmatrix}\boldsymbol{V}^{\mathrm{T}}$$

成立。其中 $\Sigma=\mathrm{diag}(\sigma_0,\sigma_1,\Lambda\ ,\sigma_p)(p\le\min(m,n)-1)$ 且 $\sigma_0\ge\sigma_1\ge\Lambda\ \ge\sigma_p>0$。

上式称为实矩阵 $\boldsymbol{A}$ 的奇异值分解式，$\sigma_i(i=0,1,\Lambda\ ,p)$ 称为 $\boldsymbol{A}$ 的奇异值。关于奇异值分解请参见 2.11 节的有关内容。

利用 A 的奇异值分解式，可以计算 A 的广义逆 A^+。设(U=U_1，U_2)，其中 U_1 为 U 中前 $p+1$ 列列正交向量组构成的 $m\times(p+1)$矩阵；V=(V_1，V_2)，其中 V_1 为 V 中前 $p+1$ 列列正交向量组构成的 $n\times(p+1)$矩阵。则 A 的广义逆定义为：

$$A^{+}=V_1\sum{}^{-1}U_1^{\mathrm{T}}$$

2. 算法实现

根据上述方法，可以定义求广义逆的奇异值分解法的 Visual Basic 函数 MInv，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MInv
'  功能：  求矩阵的广义逆
'  参数：   m    - Integer型变量。系数矩阵的行数， m>=n
'           n    - Integer型变量。系数矩阵的列数，n<=m
'          dblA  - Double型二维数组，体积为m x n。存放待分解矩阵；
'                  返回时，其对角线存放矩阵的奇异值(以非递增次序排列)，其余元素为0。
'          dblAP - Double型二维数组，体积为n x m。返回时存放矩阵的广义逆。
'          dblU  - Double型二维数组，体积为m x m。返回时，存放奇异值分解式中的
'                  左奇异向量U。
'          dblV  - Double型二维数组，体积为n x n。返回时，存放奇异值分解式中的
'                  右奇异向量VT。
'           ka   - Integer型变量。ka=max(m,n)+1
'          eps   - Double型变量。奇异值分解函数中的控制精度参数。
'  返回值： Boolean型。False，失败无解；True，成功
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MInv(m As Integer, n As Integer, dblA() As Double, _
dblAP() As Double, dblU() As Double, dblV() As Double, _
ka As Integer, eps As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, l As Integer

    ' 求解奇异值失败
    If Not MUav(m, n, dblA, dblU, dblV, ka, eps) Then
        MInv = False
        Exit Function
    End If

    ' 计算广义逆
    j = n
    If (m < n) Then j = m
    j = j - 1
    k = 0
    While (k <= j)
        If (dblA(k + 1, k + 1) = 0#) Then GoTo o_lable
        k = k + 1
    Wend
```

```
o_lable:
    k = k - 1
    For i = 0 To n - 1
        For j = 0 To m - 1
            dblAP(i + 1, j + 1) = 0#
            For l = 0 To k
                dblAP(i + 1, j + 1) = dblAP(i + 1, j + 1) + _
                  dblV(l + 1, i + 1) * dblU(j + 1, l + 1) / dblA(l + 1, l + 1)
            Next l
        Next j
    Next i

    ' 求解成功
    MInv = True

End Function
```

3. 示例

调用函数 MInv **求下列** 5×4 **阶的矩阵** $\boldsymbol{A}$ **的广义逆** $\boldsymbol{A}^{+}$**，再求** $\boldsymbol{A}^{+}$**的广义逆**$(\boldsymbol{A}^{+})^{+}$**，以验证求解的正确性：**

$$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3 & 4\\ 6 & 7 & 8 & 9\\ 1 & 2 & 13 & 0\\ 16 & 17 & 8 & 9\\ 2 & 4 & 3 & 4\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(5, 4) As Double
    Dim mtxAP(4, 5) As Double
    Dim mtxU(5, 5) As Double
    Dim mtxV(4, 4) As Double
    Dim mtxB(5, 4) As Double
    Dim mtxC(5, 4) As Double
    Dim mtxD(4, 5) As Double
    Dim i As Integer, j As Integer

    ' 原矩阵
    mtxA(1, 1) = 1: mtxA(1, 2) = 2: mtxA(1, 3) = 3: mtxA(1, 4) = 4
    mtxA(2, 1) = 6: mtxA(2, 2) = 7: mtxA(2, 3) = 8: mtxA(2, 4) = 9
    mtxA(3, 1) = 1: mtxA(3, 2) = 2: mtxA(3, 3) = 13: mtxA(3, 4) = 0
    mtxA(4, 1) = 16: mtxA(4, 2) = 17: mtxA(4, 3) = 8: mtxA(4, 4) = 9
    mtxA(5, 1) = 2: mtxA(5, 2) = 4: mtxA(5, 3) = 3: mtxA(5, 4) = 4

    ' 备份原矩阵
    For i = 1 To 5
        For j = 1 To 4
            mtxC(i, j) = mtxA(i, j)
```

```
        Next j
    Next i

    '求解A+
    If MInv(5, 4, mtxA, mtxAP, mtxU, mtxV, 6, 0.000001) Then
        ' 备份矩阵A+
        For i = 1 To 4
            For j = 1 To 5
                mtxD(i, j) = mtxAP(i, j)
            Next j
        Next i

        '求解(A+)+
        If MInv(4, 5, mtxAP, mtxB, mtxV, mtxU, 6, 0.000001) Then
            MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
                   "原矩阵A" & Chr$(13) & Chr$(13) & _
                   MatrixToString(5, 4, mtxC, "######0.0000000") & _
                   Chr$(13) & Chr$(13) & "矩阵A+" & Chr$(13) & Chr$(13) & _
                   MatrixToString(4, 5, mtxD, "######0.0000000") & _
                   Chr$(13) & Chr$(13) & "矩阵(A+)+" & Chr$(13) & _
                   Chr$(13) & MatrixToString(5, 4, mtxB, "######0.0000000")
        Else
            MsgBox "验证失败！"
        End If

    Else
        MsgBox "求解失败！"
    End If
End Sub
```

输出结果如图 2.13 **所示。**

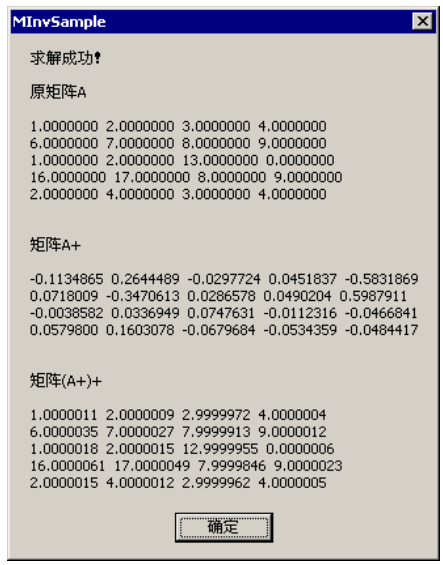

图 2.13　程序输出结果

2.13 约化对称矩阵为对称三对角阵的豪斯荷尔德变换法

1. 算法原理

本节介绍用豪斯荷尔德(Householder)变换法将 n 阶实对称矩阵约化为对称三对角阵的算法。

用豪斯荷尔德变换将 n 阶实对称矩阵 $\boldsymbol{A}$ 约化为对称三对角阵的基本方法就是，使 $\boldsymbol{A}$ 经过 n-2 次正交变换后变换成三对角阵 $\boldsymbol{A}_{n-2}$，即：

$$\boldsymbol{A}_{n-2} = P_{n-3} \Lambda \ P_1 P_0 \boldsymbol{A} P_0 P_1 \Lambda \ P_{n-3}$$

每一次正交变换 $P_i\ (i = 0, 1, 2, \Lambda\ , n-3)$ 都具有如下形式：

$$P_i = I - U_i U_i^T / H_i$$

其中 P_i 为对称正交矩阵，而且

$$\begin{aligned} H_i &= \frac{1}{2} U_i U_i^T \\ U_i &= (a_{l0}^{(i)}, a_{l1}^{(i)}, \Lambda\ , a_{l,l-2}^{(i)}, a_{l,l-1}^{(i)} \pm \sigma_i^{1/2}, 0, \Lambda\ , 0)^T \\ \sigma_i &= (a_{l0}^{(i)})^2 + (a_{l1}^{(i)})^2 + \Lambda\ + (a_{l,l-1}^{(i)})^2 \end{aligned}$$

式中 l = n-i-1。

对 A 的每一次变换为

$$A_{i+1} = P_i A_i P_i = (I - U_i U_i^T / H_i) A_i\ (I - U_i U_i^T / H_i)$$

若令

$$\begin{cases} s_i = A_i U_i / H_i \\ k_i = U_i^T s_i / 2H_i \\ q_i = s_i - k_i U_i \end{cases}$$

则

$$A_{i+1} = A_i - U_i q_i^T - q_i U_i^T$$

其中 s_i 具有如下形式：

$$s_i = (s_{i0}, s_{i1}, \Lambda\ , s_{il}, 0, \Lambda\ , 0)^T$$

q_i 的形式为

$$q_i = (s_{i0} - k_i u_{i0}, s_{i1} - k_{i1} u_{i1}, \Lambda\ , s_{i,l-1} - k_i u_{i,l-1}, s_{il}, 0, \Lambda\ , 0)^T$$

2. **算法实现**

根据上述方法，可以定义用豪斯荷尔德变换法约化对称矩阵为对称三对角矩阵的 Visual Basic **函数** MSymTri**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MSymTri
'  功能：  用豪斯荷尔德变换约化对称矩阵为对称三对角矩阵
'  参数：   n    - Integer型变量，对称矩阵的阶数。
'         dblA  - Double型二维数组，体积为n x n。存放n阶对称矩阵。
'         dblQ  - Double型二维数组，体积为n x n。
'                 返回时存放豪斯荷尔德变换的乘积矩阵Q。
'         dblT - Double型二维数组，体积为n x n。返回时存放对称三对角矩阵T。
'         dblB - Double型一维数组，长度为n。
'                返回时存放对称三对角矩阵T主对角线上的元素。
'         dblC - Double型一维数组，长度为n。
'                返回时前n-1个元素存放对称三对角矩阵T次对角线上的元素。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MSymTri(n As Integer, dblA() As Double, dblQ() As Double, _
dblT() As Double, dblB() As Double, dblC() As Double)
    Dim i As Integer, j As Integer, k As Integer
    Dim h As Double, f As Double, g As Double, h2 As Double

    ' 初始化
    For i = 1 To n
        For j = 1 To n
            dblQ(i, j) = dblA(i, j)
        Next j
    Next i

    For i = n To 2 Step -1
        h = 0#
        If (i > 2) Then
            For k = 1 To i - 1
                h = h + dblQ(i, k) * dblQ(i, k)
            Next k
        End If

        If (h + 1# = 1#) Then
            dblC(i) = 0#
            If (i = 2) Then dblC(i) = dblQ(i, i - 1)
            dblB(i) = 0#
        Else
            dblC(i) = Sqr(h)

            If (dblQ(i, i - 1) > 0#) Then dblC(i) = -dblC(i)

            h = h - dblQ(i, i - 1) * dblC(i)
            dblQ(i, i - 1) = dblQ(i, i - 1) - dblC(i)
```

```
        f = 0#

        For j = 1 To i - 1
           dblQ(j, i) = dblQ(i, j) / h
           g = 0#
           For k = 1 To j
             g = g + dblQ(j, k) * dblQ(i, k)
           Next k

           If (j + 1 <= i - 1) Then
              For k = j + 1 To i - 1
                 g = g + dblQ(k, j) * dblQ(i, k)
              Next k
           End If

           dblC(j) = g / h
           f = f + g * dblQ(j, i)
        Next j

        h2 = f / (h + h)
        For j = 1 To i - 1
           f = dblQ(i, j)
           g = dblC(j) - h2 * f
           dblC(j) = g
           For k = 1 To j
              dblQ(j, k) = dblQ(j, k) - f * dblC(k) - g * dblQ(i, k)
           Next k
        Next j

        dblB(i) = h
     End If
  Next i

  For i = 1 To n - 1
     dblC(i) = dblC(i + 1)
  Next i

  dblC(n) = 0#
  dblB(1) = 0#
  For i = 1 To n
     If ((dblB(i) <> 0#) And (i - 1 >= 0)) Then
       For j = 1 To i - 1
           g = 0#
           For k = 1 To i - 1
              g = g + dblQ(i, k) * dblQ(k, j)
           Next k
           For k = 1 To i - 1
              dblQ(k, j) = dblQ(k, j) - g * dblQ(k, i)
           Next k
       Next j
```

```
        End If

        dblB(i) = dblQ(i, i)
        dblQ(i, i) = 1#
        If (i - 1 >= 0) Then
            For j = 1 To i - 1
                dblQ(i, j) = 0#
                dblQ(j, i) = 0#
            Next j
        End If
    Next i

    ' 构造对称三对角矩阵
    For i = 1 To n
        For j = 1 To n
            dblT(i, j) = 0
            k = i - j
            If k = 0 Then dblT(i, j) = dblB(j)
            If k = 1 Then dblT(i, j) = dblC(j)
            If k = -1 Then dblT(i, j) = dblC(i)
        Next j
    Next i
End Sub
```

3. 示例

调用函数 MSymTri **将下列** 5×5 **阶矩阵** $\boldsymbol{A}$ **约化为对称三对角阵，并显示豪斯荷尔德变换的乘积矩阵** $\boldsymbol{Q}$**：**

$$\boldsymbol{A}=\begin{bmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & -1 \\ 4 & -3 & -5 & -1 & 15 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(5, 5) As Double
    Dim mtxQ(5, 5) As Double
    Dim mtxT(5, 5) As Double
    Dim dblB(5) As Double, dblC(5) As Double

    ' 原矩阵
    mtxA(1,1)=10:  mtxA(1,2)=1 : mtxA(1,3)=2 :  mtxA(1,4)=3 : mtxA(1,5)=4
    mtxA(2,1)=1 :  mtxA(2,2)=9 : mtxA(2,3)=-1:  mtxA(2,4)=2 : mtxA(2,5)=-3
    mtxA(3,1)=2 :  mtxA(3,2)=-1: mtxA(3,3)=7 :  mtxA(3,4)=3 : mtxA(3,5)=-5
    mtxA(4,1)=3 :  mtxA(4,2)=2 : mtxA(4,3)=3 :  mtxA(4,4)=12: mtxA(4,5)=-1
    mtxA(5,1)=4 :  mtxA(5,2)=-3: mtxA(5,3)=-5:  mtxA(5,4)=-1: mtxA(5,5)=15
```

```
    '求解
    Call MSymTri(5, mtxA, mtxQ, mtxT, dblB, dblC)

    MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
           "原矩阵A" & Chr$(13) & Chr$(13) & _
           MatrixToString(5, 5, mtxA, "######0.0000000") & Chr$(13) & _
           Chr$(13) & "矩阵Q" & Chr$(13) & Chr$(13) & _
           MatrixToString(5, 5, mtxQ, "######0.0000000") & Chr$(13) & _
           Chr$(13) & "对称三对角矩阵T" & Chr$(13) & Chr$(13) & _
           MatrixToString(5, 5, mtxT, "######0.0000000")

End Sub
```

其输出结果如图 2.14 **所示。**

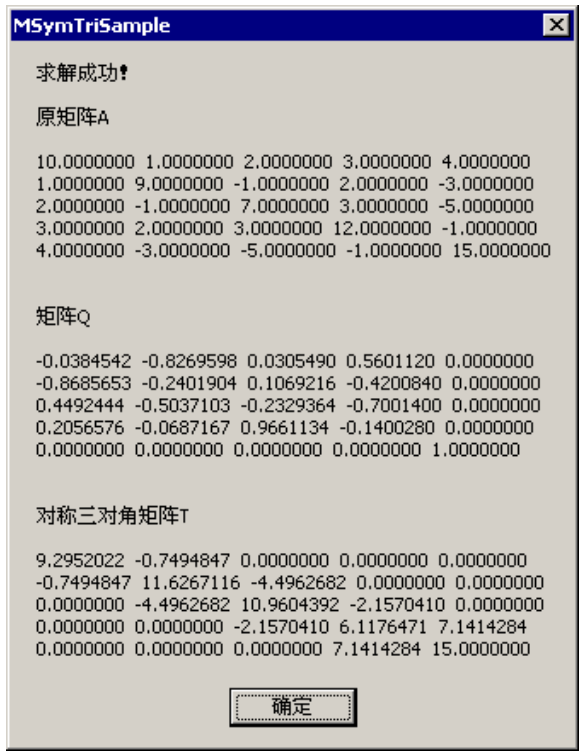

图 2.14　程序输出结果

2.14　实对称三对角阵的全部特征值与特征向量的计算

1. 算法原理

计算实对称矩阵的特征值和特征向量的一种常用方法是：先通过豪斯荷尔德变换法将对称矩阵约化为对称三对角阵(参见 2.13 **节)，再通过变形** QR **方法(参见** 2.16 **节)计算对称三对角阵的特征值和特征向量。**

2. **算法实现**

上述计算实对称矩阵的特征值和特征向量方法的 Visual Basic **函数** MSymTriEigenv **可定义如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MSymTriEigenv
'  功能：  用变形QR方法计算对称三对角矩阵的全部特征值和特征向量
'  参数：   n    - Integer型变量，对称三对角矩阵的阶数。
'         dblB - Double型一维数组，长度为n，存放对称三对角矩阵T主对角线上的元素。
'                返回时，存放全部特征值。
'         dblC - Double型一维数组，长度为n，前n-1个元素存放对称三对角矩阵T次对角线
'                上的元素。
'         dblQ - Double型二维数组，体积为n x n。
'               1)如果存放单位矩阵，则返回n阶对称三对角矩阵的特征向量组。
'               2)如果存放对称矩阵A的豪斯荷尔德变换的乘积矩阵Q
'                 (可由函数MSymTri求得)，则返回n阶对称矩阵A的特征向量组。
'                  其中dblQ中的第j列为与数组dblB中第j个特征值对应的特征向量。
'         eps  - Double型变量。迭代过程中的控制精度参数。
'         nMaxItNum    - Integer。为求得一个特征值所允许的最大迭代次数。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MSymTriEigenv(n As Integer, dblB() As Double, dblC() As Double, _
dblQ() As Double, eps As Double, nMaxItNum As Integer) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, m As Integer, it As Integer
    Dim d As Double, f As Double, h As Double, g As Double, p As Double, _
         r As Double, e As Double, s As Double

    dblC(n) = 0#
    d = 0#
    f = 0#
    For j = 1 To n
        it = 0
        h = eps * (Abs(dblB(j)) + Abs(dblC(j)))
        If (h > d) Then d = h
        m = j
        While ((m <= n) And (Abs(dblC(m)) > d))
            m = m + 1
        Wend
        If (m <> j) Then
            Do
                If (it = nMaxItNum) Then
                    MSymTriEigenv = False
                    Exit Function
                End If
                it = it + 1
                g = dblB(j)
                p = (dblB(j + 1) - g) / (2# * dblC(j))
                r = Sqr(p * p + 1#)
```

```
            If (p >= 0#) Then
               dblB(j) = dblC(j) / (p + r)
            Else
               dblB(j) = dblC(j) / (p - r)
            End If
            h = g - dblB(j)
            For i = j + 1 To n
              dblB(i) = dblB(i) - h
            Next i
            f = f + h
            p = dblB(m)
            e = 1#
            s = 0#
            For i = m - 1 To j Step -1
               g = e * dblC(i)
               h = e * p
               If (Abs(p) >= Abs(dblC(i))) Then
                  e = dblC(i) / p
                  r = Sqr(e * e + 1#)
                  dblC(i + 1) = s * p * r
                  s = e / r
                  e = 1# / r
               Else
                  e = p / dblC(i)
                  r = Sqr(e * e + 1#)
                  dblC(i + 1) = s * dblC(i) * r
                  s = 1# / r
                  e = e / r
               End If
               p = e * dblB(i) - s * g
               dblB(i + 1) = h + s * (e * g + s * dblB(i))
               For k = 1 To n
                  h = dblQ(k, i + 1)
                  dblQ(k, i + 1) = s * dblQ(k, i) + e * h
                  dblQ(k, i) = e * dblQ(k, i) - s * h
               Next k
            Next i
            dblC(j) = s * p
            dblB(j) = e * p
         Loop While (Abs(dblC(j)) > d)
      End If
      dblB(j) = dblB(j) + f
   Next j
   For i = 1 To n
      k = i
      p = dblB(i)
      If (i + 1 <= n) Then
         j = i + 1
         While ((j <= n) And (dblB(j) <= p))
            k = j
            p = dblB(j)
```

```
            j = j + 1
         Wend
      End If
      If (k <> i) Then
         dblB(k) = dblB(i)
         dblB(i) = p

         For j = 1 To n
            p = dblQ(j, i)
            dblQ(j, i) = dblQ(j, k)
            dblQ(j, k) = p
         Next j
      End If
   Next i

   MSymTriEigenv = True

End Function
```

3. 示例

调用函数 MSymTriEigenv **计算下面** 5×5 **阶矩阵** A **的特征值和特征向量：**

$$\boldsymbol{A}=\begin{bmatrix}10 & 1 & 2 & 3 & 4\\ 1 & 9 & -1 & 2 & -3\\ 2 & -1 & 7 & 3 & -5\\ 3 & 2 & 3 & 12 & -1\\ 4 & -3 & -5 & -1 & 15\end{bmatrix}$$

程序代码如下：

```
Sub Main()
   Dim mtxA(5, 5) As Double
   Dim mtxQ(5, 5) As Double
   Dim mtxT(5, 5) As Double
   Dim dblB(5) As Double, dblC(5) As Double
   Dim sEigenValue As String, sEigenVector As String
   Dim i As Integer

  ' 原矩阵
   mtxA(1,1)=10:  mtxA(1,2)=1: mtxA(1,3)=2:  mtxA(1,4)=3: mtxA(1,5)=4
   mtxA(2,1)=1:  mtxA(2,2)=9 : mtxA(2,3)=-1:  mtxA(2,4)=2 : mtxA(2,5)=-3
   mtxA(3,1)=2:  mtxA(3,2)=-1: mtxA(3,3)=7 :  mtxA(3,4)=3 : mtxA(3,5)=-5
   mtxA(4,1)=3:  mtxA(4,2)=2 : mtxA(4,3)=3 :  mtxA(4,4)=12: mtxA(4,5)=-1
   mtxA(5,1)=4:  mtxA(5,2)=-3: mtxA(5,3)=-5:  mtxA(5,4)=-1: mtxA(5,5)=15

   '求解对称三对角矩阵
   Call MSymTri(5, mtxA, mtxQ, mtxT, dblB, dblC)
   '求解对称三对角矩阵的特征值和特征向量
   If MSymTriEigenv(5, dblB, dblC, mtxQ, 0.000001, 60) Then
   ' 特征值和特征向量
```

```
        For i = 1 To 5
            sEigenValue = sEigenValue & i & ": " & _
                          Format(dblB(i), "######0.0000000") & Chr(13)
            sEigenVector = sEigenVector & i & ": " & _
                        MatrixColToString(5, i, mtxQ, "######0.0000000") & _
                        Chr(13)
        Next i
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "原矩阵A" & Chr$(13) & Chr$(13) & _
               MatrixToString(5, 5, mtxA, "######0.0000000") & Chr$(13) & _
               "特征值" & Chr$(13) & Chr$(13) & _
               sEigenValue & Chr$(13) & _
               "特征向量" & Chr$(13) & Chr$(13) & _
               sEigenVector
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

其输出结果如图 2.15 **所示。**

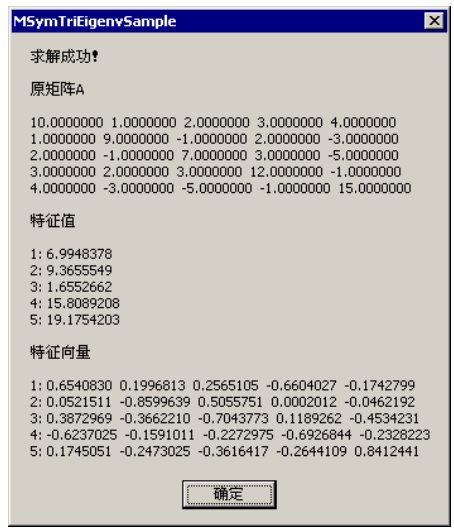

图 2.15　程序输出结果

2.15　约化一般实矩阵为赫申伯格矩阵的初等相似变换法

1. 算法原理

计算一般矩阵的特征值和特征向量时，一般先把矩阵变换为赫申伯格(Hessen Berg)**矩阵，以减少计算量。赫申伯格矩阵简称为上** H **阵，具有如下的性质：**

$$h_{ij} = 0,\ i > j+1$$

初等相似变换法是约化一般实矩阵为赫申伯格矩阵的重要方法。约化一个实矩阵 $\boldsymbol{A}$ 为上 H 阵需要经过两个主要步骤，他们依次把 $\boldsymbol{A}$ 的第 1，2，…，n-2 列化为上 H 阵。

设 $\boldsymbol{A}_1$ 经过 r-1 步后变化为 $\boldsymbol{A}_r$，则第 r 步的计算方法如下：

(1) 选取 $|a_{(r+1)'r}^{(r)}|$，使：

$$|a_{(r+1)'r}^{(r)}| = \max_{i=r+1}^{n}\{a_{ir}^{(r)}\}$$

如果存在多个 $|a_{(r+1)'r}^{(r)}|$，则选取首先遇到的那个。

(2) 交换第 $(r+1)'$ 行与第 $(r+1)$ 行，交换第 $(r+1)'$ 列与第 $(r+1)$ 列。

(3) 对于每一个 $i(i=r+2，\ldots，n)$，计算

$$n_{i,r+1} = a_{ir}^{(r)} \Big/ a_{i+1,r}^{(r)}$$

(第i行) $- n_{i,r+1} \times$ (第$r+1$行)

$n_{i,r+1} \times$ (第i列) + (第$r+1$列)

因此，

$$\boldsymbol{A}_{r+1} = \boldsymbol{N}_{r+1}^{-1}\boldsymbol{I}_{r+1,(r+1)'}\boldsymbol{A}_r\boldsymbol{I}_{r+1,(r+1)'}\boldsymbol{N}_{r+1}$$

其中，

$\boldsymbol{I}_{r+1,(r+1)'}$ 是初等置换矩阵，

$\boldsymbol{N}_{r+1}$ 是初等矩阵，即：$(\boldsymbol{N}_{r+1})_{i,\,r+1} = n_{i,\,r+1}$，$i = r+2, \Lambda\ , n$，其余元素与单位矩阵相同。

2. 算法实现

根据上述方法，可以定义初等相似变换法约化一般实矩阵为赫申伯格矩阵的 Visual Basic 函数 MHberg，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MHberg
'  功能：  用初等相似变换约化一般矩阵为赫申伯格(Hessen Berg)矩阵
'  参数：   n    - Integer型变量，矩阵的阶数。
'          mtxA  - Double型二维数组，体积为n×n。存放n阶矩阵；
'                  返回时存放矩阵的H矩阵。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MHberg(n As Integer, mtxA() As Double)
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim d As Double, t As Double

    For k = 2 To n - 1
       ' 选取绝对值最大的元素
        d = 0#
        For j = k To n
```

```
            t = mtxA(j, k - 1)
            If (Abs(t) > Abs(d)) Then
                d = t
                i = j
            End If
        Next j
        If (Abs(d) + 1# <> 1#) Then
            ' 交换i,k行，j,k列
            If (i <> k) Then
                For j = k - 1 To n
                    t = mtxA(i, j)
                    mtxA(i, j) = mtxA(k, j)
                    mtxA(k, j) = t
                Next j
                For j = 1 To n
                    t = mtxA(j, i)
                    mtxA(j, i) = mtxA(j, k)
                    mtxA(j, k) = t
                Next j
            End If
            ' 变换
            For i = k + 1 To n
                t = mtxA(i, k - 1) / d
                mtxA(i, k - 1) = 0#
                For j = k To n
                    mtxA(i, j) = mtxA(i, j) - t * mtxA(k, j)
                Next j
                For j = 1 To n
                    mtxA(j, k) = mtxA(j, k) + t * mtxA(j, i)
                Next j
            Next i
        End If
    Next k
End Function
```

3. 示例

调用函数 MHBerg **求下列** 5×5 **阶的矩阵** A **的上** H **阵：**

$$A=\begin{bmatrix} 1 & 6 & -3 & -1 & 7 \\ 8 & -15 & 18 & 5 & 4 \\ -2 & 11 & 9 & 15 & 20 \\ -3 & 2 & 21 & 30 & -6 \\ 17 & 22 & -5 & 3 & 6 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(5, 5) As Double
    Dim mtxB(5, 5) As Double
```

```
    Dim i As Integer, j As Integer

    ' 原矩阵
    mtxA(1,1)=1  :  mtxA(1,2)=6  : mtxA(1,3)=-3:  mtxA(1,4)=-1: mtxA(1,5)=7
    mtxA(2,1)=8  :  mtxA(2,2)=-15: mtxA(2,3)=18:  mtxA(2,4)=5: mtxA(2,5)=4
    mtxA(3,1)=-2 :  mtxA(3,2)=11 : mtxA(3,3)=9:  mtxA(3,4)=15: mtxA(3,5)=20
    mtxA(4,1)=-13: mtxA(4,2)=2 :  mtxA(4,3)=21:  mtxA(4,4)=30: mtxA(4,5)=-6
    mtxA(5,1)=17 :  mtxA(5,2)=22: mtxA(5,3)=-5:  mtxA(5,4)=3: mtxA(5,5)=6

    ' 备份原矩阵
    For i = 1 To 5
       For j = 1 To 5
          mtxB(i, j) = mtxA(i, j)
       Next j
    Next i

    '求解
    Call MHberg(5, mtxA)
    MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
           "原矩阵A" & Chr$(13) & Chr$(13) & _
           MatrixToString(5, 5, mtxB, "######0.0000000") & Chr$(13) & _
           Chr$(13) & "H矩阵" & Chr$(13) & Chr$(13) & _
           MatrixToString(5, 5, mtxA, "######0.0000000")
End Sub
```

其输出结果如图 2.16 **所示。**

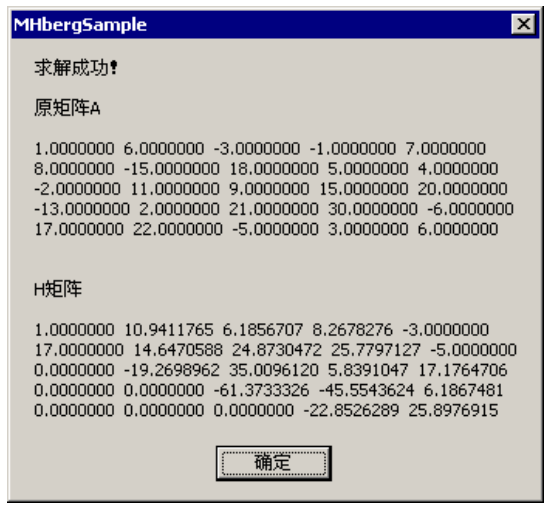

图 2.16　程序输出结果

2.16　求赫申伯格矩阵全部特征值的 QR 方法

1. 算法原理

本节介绍用带原点位移的双重步 QR **方法计算实上** H **阵的全部特征值的算法。**

设上 H 阵 $\boldsymbol{A}$ 为不可约的，通过一系列双重步 QR 变换，使上 H 阵变为对角块全部是一阶块或二阶块，进而从中解出全部特征值。

双重步 QR 变换的计算步骤可以归纳如下：

(1) 确定一个初等正交对称矩阵 $\boldsymbol{Q}_0$，对 $\boldsymbol{A}$ 作相似变换

$$\boldsymbol{A}_1 = \boldsymbol{Q}_0 \boldsymbol{A} \boldsymbol{Q}_0$$

其中，$\boldsymbol{Q}_0$ 为对称正交矩阵，具有如下形式

$$\boldsymbol{Q}_0 = \begin{bmatrix} \tilde{\boldsymbol{Q}}_0 & 0 \\ 0 & I_{n-3} \end{bmatrix}$$

且 $\tilde{\boldsymbol{Q}}_0$ 为 3×3 的矩阵。若令

$$\begin{cases} \alpha = a_{n-2,\,n-2} + a_{n-1,\,n-1} \\ \beta = a_{n-2,\,n-2} \cdot a_{n-1,\,n-1} - a_{n-2,\,n-1} \cdot a_{n-1,\,n-2} \end{cases}$$

$$\begin{cases} p_0 = a_{00}(a_{00} - \alpha) + a_{01}a_{10} + \beta \\ q_0 = a_{10}(a_{00} + a_{11} - \alpha) \\ r_0 = a_{10}a_{21} \end{cases}$$

则 $\tilde{\boldsymbol{Q}}_0$ 中的各元素为

$$\tilde{\boldsymbol{Q}}_0 = \begin{bmatrix} -\dfrac{p_0}{s_0} & -\dfrac{q_0}{s_0} & -\dfrac{r_0}{s_0} \\ -\dfrac{q_0}{s_0} & \dfrac{p_0}{s_0} + \dfrac{r_0^2}{s_0(p_0+s_0)} & -\dfrac{q_0 r_0}{s_0(p_0+s_0)} \\ -\dfrac{r_0}{s_0} & -\dfrac{q_0 r_0}{s_0(p_0+s_0)} & \dfrac{p_0}{s_0} + -\dfrac{q_0^2}{s_0(p_0+s_0)} \end{bmatrix}$$

其中，$s_0 = \operatorname{sign}(p_0)\sqrt{p_0^2 + q_0^2 + r_0^2}$。

由此可得

$$\boldsymbol{A}_1 = \boldsymbol{Q}_0 \boldsymbol{A} \boldsymbol{Q}_0 = \begin{bmatrix} * & * & * & * & \Lambda & * \\ p_1 & * & * & * & \Lambda & * \\ \underline{q_1} & * & * & * & \Lambda & * \\ \underline{r_1} & \underline{*} & * & * & \Lambda & * \\ & & & \mathrm{O} & \mathrm{O} & \mathrm{M} \\ & & 0 & & * & * \end{bmatrix}$$

其中，有下划线的元素为次对角线以下新增加的 3 个非 0 元素。

(2) 利用同样的方法，依次确定正交对称矩阵 $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \Lambda, \boldsymbol{Q}_{n-2}$，对 $\boldsymbol{A}_1, \boldsymbol{A}_2, \Lambda, \boldsymbol{A}_{n-2}$ 作相似变换

$$\boldsymbol{A}_{i+1}=\boldsymbol{Q}_i\boldsymbol{A}_i\boldsymbol{Q}_i,\qquad i=1,2,\Lambda\ ,n-2$$

最后可得上 H 阵

$$\boldsymbol{A}_{n-1}=\boldsymbol{Q}_{n-1}\boldsymbol{A}_{n-2}\boldsymbol{Q}_{n-2}$$

在这一过程中，一般有：

$$\begin{bmatrix}
* & * & * & \Lambda & * & * & * & * & \Lambda & * \\
* & * & * & \Lambda & * & * & * & * & \Lambda & * \\
 & * & * & \Lambda & * & * & * & * & \Lambda & * \\
 & & \mathrm{O} & \mathrm{O} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\
 & & & * & * & * & * & * & \Lambda & * \\
 & & & & p_i & * & * & * & \Lambda & * \\
 & & & & q_i & * & * & * & \Lambda & * \\
 & & & & \bar{r}_i & * & * & * & \Lambda & * \\
 & & & & \bar{\ } & \bar{\ } & & \mathrm{O} & \mathrm{O} & * \\
 & & & & & & & & * & *
\end{bmatrix}$$

为了消去 $\boldsymbol{A}_i$ 中次对角线以下有下划线的 3 个非 0 元素，$\boldsymbol{Q}_i$ 可以取如下形式的正交对称矩阵：

$$\boldsymbol{Q}_i=\begin{bmatrix} I_i & 0 & 0 \\ 0 & \tilde{\boldsymbol{Q}}_i^{(0)} & 0 \\ 0 & 0 & I_{n-i-3} \end{bmatrix}$$

其中 $\tilde{\boldsymbol{Q}}_i^{(0)}$ 为如下 3 阶方阵：

$$\tilde{\boldsymbol{Q}}_i^{(0)}=\begin{bmatrix}
-\dfrac{p_i}{s_i} & -\dfrac{q_i}{s_i} & -\dfrac{r_i}{s_i} \\
-\dfrac{q_i}{s_i} & \dfrac{p_i}{s_i}+\dfrac{r_i^{\,2}}{s_i(p_i+s_i)} & -\dfrac{q_ir_i}{s_i(p_i+s_i)} \\
-\dfrac{r_i}{s_i} & -\dfrac{q_ir_i}{s_i(p_i+s_i)} & \dfrac{p_i}{s_i}+\dfrac{q_i^{\,2}}{s_i(p_i+s_i)}
\end{bmatrix}$$

其中， $s_i=\mathrm{sign}(p_i)\sqrt{p_i^2+q_i^2+r_i^2}$

上式中的 p_i, q_i, $r_i\,(i=1,2,\Lambda\ ,n-2)$ 可从相应的 A_i 的第 $i-1$ 列中取得，即：

$$p_i = a_{i,\,i-1}^{(i)}$$
$$q_i = a_{i+1,\,i-1}^{(i)}$$
$$r_i = a_{i+2,\,i-1}^{(i)}$$
$$p_{n-2} = a_{n-2,\,n-3}^{(n-2)}$$
$$q_{n-2} = a_{n-1,\,n-3}^{(n-2)}$$
$$r_{n-2} = 0$$

反复计算上述两步，直到将上 H 阵变换为对角块全部是一阶块或二阶块为止。此时，就可以直接从各一阶块或二阶块中求出全部的特征值。

QR 方法是一种迭代方法。在每次迭代过程中，如果次对角线元素的模小到一定程度，就可以把他们看成 0，即如果$|a_{k,\,k-1}| \le \varepsilon(|a_{k-1,\,k-1}|+|a_{kk}|)$，则$a_{k,\,k-1}=0$（$\varepsilon$ 为指定的精度控制参数)。这样，将矩阵分割为各不可约的上 H 阵，以便逐步降低主子阵的阶数，从而减少计算量。

注意，当上 H 阵类似正交矩阵时，本算法可能失败或误差较大。

2. 算法实现

根据上述方法，可以定义求赫申伯格矩阵全部特征值的 QR 方法的 Visual Basic 函数 MHbergEigenv，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MHbergEigenv
'  功能：  用QR方法计算赫申伯格(Hessen Berg)矩阵的全部特征值
'  参数：   n    - Integer型变量，赫申伯格矩阵的阶数。
'          dblA - Double型二维数组，体积为n x n。存放赫申伯格矩阵。
'          dblUR - Double型一维数组，长度为n，存放n个特征值的实部。
'          dblUI - Double型一维数组，长度为n，存放n个特征值的虚部。。
'          eps  - Double型变量。迭代过程中的控制精度参数。
'          nMaxItNum    - Integer。为求得一个特征值所允许的最大迭代次数。
'  返回值： Boolean型。False，失败无解；True, 成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MHbergEigenv(n As Integer, dblA() As Double, _
dblUR() As Double, dblUI() As Double, eps As Double, _
nMaxItNum As Integer) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer, l As Integer, _
        m As Integer, it As Integer
    Dim ii As Integer, jj As Integer, kk As Integer, ll As Integer
    Dim b As Double, c As Double, w As Double, g As Double, _
        xy As Double, p As Double, q As Double
    Dim r As Double, x As Double, s As Double, e As Double, f As Double, _
        z As Double, y As Double

    ' 初值
    it = 0
```

```
m = n + 1
While (m <> 1)
    l = m - 1
    While ((l > 1) And (Abs(dblA(l, l - 1)) > eps * _
                       (Abs(dblA(l - 1, l - 1)) + Abs(dblA(l, l)))))
        l = l - 1
    Wend
    If (l = m - 1) Then
        dblUR(m - 1) = dblA(m - 1, m - 1)
        dblUI(m - 1) = 0#
        m = m - 1
        it = 0
    Else
        If (l = m - 2) Then
            b = -(dblA(m - 1, m - 1) + dblA(m - 2, m - 2))
            c = dblA(m - 1, m - 1) * dblA(m - 2, m - 2) - _
                 dblA(m - 1, m - 2) * dblA(m - 2, m - 1)
            w = b * b - 4# * c
            y = Sqr(Abs(w))
            If (w > 0#) Then
                xy = 1#
                If (b < 0#) Then xy = -1#
                dblUR(m - 1) = (-b - xy * y) / 2#
                dblUR(m - 2) = c / dblUR(m - 1)
                dblUI(m - 1) = 0#
                dblUI(m - 2) = 0#
            Else
                dblUR(m - 1) = -b / 2#
                dblUR(m - 2) = dblUR(m - 1)
                dblUI(m - 1) = y / 2#
                dblUI(m - 2) = -dblUI(m - 1)
            End If
            m = m - 2
            it = 0
        Else
            ' 经过指定的迭代次数后，精度仍不能达到要求，计算失败，返回
            If (it >= nMaxItNum) Then
                MHbergEigenv = False
                Exit Function
            End If
            ' 下一次迭代
            it = it + 1
            For j = l + 2 To m - 1
              dblA(j, j - 2) = 0#
            Next j
            For j = l + 3 To m - 1
              dblA(j, j - 3) = 0#
            Next j
            For k = l To m - 2
                If (k <> l) Then
```

```
    p = dblA(k, k - 1)
    q = dblA(k + 1, k - 1)
    r = 0#
    If (k <> m - 2) Then r = dblA(k + 2, k - 1)
Else
    x = dblA(m - 1, m - 1) + dblA(m - 2, m - 2)
    y = dblA(m - 2, m - 2) * dblA(m - 1, m - 1) - _
         dblA(m - 2, m - 1) * dblA(m - 1, m - 2)
    p = dblA(l, l) * (dblA(l, l) - x) + _
         dblA(l, l + 1) * dblA(l + 1, l) + y
    q = dblA(l + 1, l) * (dblA(l, l) + _
         dblA(l + 1, l + 1) - x)
    r = dblA(l + 1, l) * dblA(l + 2, l + 1)
End If
If ((Abs(p) + Abs(q) + Abs(r)) <> 0#) Then
    xy = 1#
    If (p < 0#) Then xy = -1#
    s = xy * Sqr(p * p + q * q + r * r)
    If (k <> l) Then dblA(k, k - 1) = -s
    e = -q / s
    f = -r / s
    x = -p / s
    y = -x - f * r / (p + s)
    g = e * r / (p + s)
    z = -x - e * q / (p + s)
    For j = k To m - 1
        p = x * dblA(k, j) + e * dblA(k + 1, j)
        q = e * dblA(k, j) + y * dblA(k + 1, j)
        r = f * dblA(k, j) + g * dblA(k + 1, j)
        If (k <> m - 2) Then
            p = p + f * dblA(k + 2, j)
            q = q + g * dblA(k + 2, j)
            r = r + z * dblA(k + 2, j)
            dblA(k + 2, j) = r
        End If
        dblA(k + 1, j) = q
        dblA(k, j) = p
    Next j
    j = k + 3
    If (j >= m - 1) Then j = m - 1
    For i = l To j
        p = x * dblA(i, k) + e * dblA(i, k + 1)
        q = e * dblA(i, k) + y * dblA(i, k + 1)
        r = f * dblA(i, k) + g * dblA(i, k + 1)

        If (k <> m - 2) Then
            p = p + f * dblA(i, k + 2)
            q = q + g * dblA(i, k + 2)
            r = r + z * dblA(i, k + 2)
            dblA(i, k + 2) = r
        End If
```

```
                    dblA(i, k + 1) = q
                    dblA(i, k) = p
                Next i
            End If
        Next k
    End If
End If
Wend

' 计算成功，返回
MHbergEigenv = True

End Function
```

3. 示例

调用函数 MHBergEigenv **求下列** 5×5 **阶的矩阵** A **的全部特征值：**

$$A=\begin{bmatrix} 1 & 6 & -3 & -1 & 7 \\ 8 & -15 & 18 & 5 & 4 \\ -2 & 11 & 9 & 15 & 20 \\ -3 & 2 & 21 & 30 & -6 \\ 17 & 22 & -5 & 3 & 6 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim mtxA(5, 5) As Double
    Dim mtxB(5, 5) As Double
    Dim mtxC(5, 5) As Double
    Dim dblUR(5) As Double
    Dim dblUI(5) As Double
    Dim i As Integer, j As Integer
    Dim s As String

    ' 原矩阵
    mtxA(1,1)=1:  mtxA(1,2)=6:   mtxA(1,3)=-3:mtxA(1,4)=-1: mtxA(1,5)=7
    mtxA(2,1)=8:  mtxA(2,2)=-15:mtxA(2,3)=18: mtxA(2,4)=5:  mtxA(2,5)=4
    mtxA(3,1)=-2: mtxA(3,2)=11: mtxA(3,3)=9:  mtxA(3,4)=15: mtxA(3,5)=20
    mtxA(4,1)=-13:mtxA(4,2)=2:  mtxA(4,3)=21: mtxA(4,4)=30: mtxA(4,5)=-6
    mtxA(5,1)=17: mtxA(5,2)=22: mtxA(5,3)=-5: mtxA(5,4)=3:  mtxA(5,5)=6

    ' 备份原矩阵
    For i = 1 To 5
        For j = 1 To 5
            mtxB(i, j) = mtxA(i, j)
        Next j
    Next i

    '求解H矩阵
    Call MHberg(5, mtxA)
```

```
    ' 备份H矩阵
    For i = 1 To 5
        For j = 1 To 5
            mtxC(i, j) = mtxA(i, j)
        Next j
    Next i

    ' 求解特征值
    If MHbergEigenv(5, mtxA, dblUR, dblUI, 0.000001, 60) Then
        ' 生成特征值显示字符串
        For i = 1 To 5
            s = s + Format(dblUR(i), "######0.0000000") + " + " + _
                Format(dblUI(i), "######0.0000000") + "j" & Chr$(13)
        Next i
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "原矩阵A" & Chr$(13) & Chr$(13) & _
               MatrixToString(5, 5, mtxB, "######0.0000000") & Chr$(13) & _
               "H矩阵" & Chr$(13) & Chr$(13) & _
               MatrixToString(5, 5, mtxC, "######0.0000000") & Chr$(13) & _
               "特征值为" & Chr$(13) & Chr$(13) & s
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

其输出结果如图 2.17 **所示。**

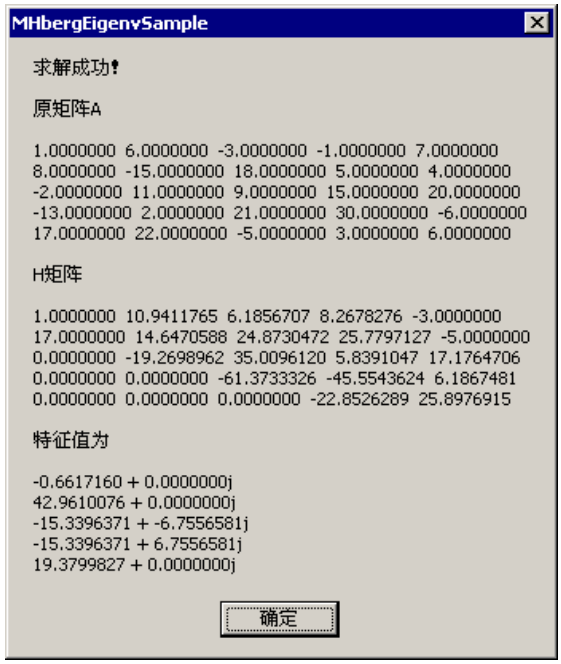

图 2.17　程序输出结果

2.17　求实对称矩阵特征值与特征向量的雅可比法

1. 算法原理

雅可比(Jacobi)方法是求实对称矩阵特征值与特征向量的常用方法。其基本思路如下：

设 n 阶矩阵 $\boldsymbol{A}$ 为对称矩阵，在其非对角线元素中选取一个绝对值最大的元素，设为 a_{pq}；利用平面旋转变换矩阵 $\boldsymbol{R}_0(p,q,\theta)$ 对 $\boldsymbol{A}$ 进行正交相似变换：

$$\boldsymbol{A}_1 = \boldsymbol{R}_0(p,q,\theta)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{R}_0(p,q,\theta)$$

其中，$\boldsymbol{R}_0(p,q,\theta)$ 的元素为

$$r_{pp} = \cos\theta, \quad r_{qq} = \cos\theta, \quad r_{pq} = -\sin\theta, \quad r_{qp} = \sin\theta$$
$$r_{ij} = 0, \quad i, j \neq p, q$$

如果按下式确定角度 θ：

$$tg\,2\theta = \frac{2a_{pq}}{a_{pp} - a_{qq}}$$

则对称矩阵 $\boldsymbol{A}$ 经上述变换后，其非对角线元素的平方和将减少 $2a_{pq}^2$，对角线元素的平方和将增加 $2a_{pq}^2$，而矩阵中所有元素的平方和保持不变。由此可知，对称矩阵 $\boldsymbol{A}$ 每经过一次这样的正交相似变换，其对角线元素的平方和将向 0 趋近一步。因此，只要反复进行上述正交变换，就可以将对称矩阵 $\boldsymbol{A}$ 变换为对角矩阵。对角矩阵中对角线上的元素 $\lambda_0, \lambda_1, \Lambda, \lambda_{n-1}$ 即是矩阵的特征值，而每一步中的平面旋转矩阵的乘积的第 i 列($i = 0, 1, 2, \ldots, n-1$)即为与 λ_i 对应的特征向量。

综上所述，用雅可比方法求 n 阶对称矩阵 $\boldsymbol{A}$ 的特征值和特征向量的步骤如下：

(1) 令 $\boldsymbol{S}=\boldsymbol{I}_n$，$\boldsymbol{I}_n$ 为单位矩阵；

(2) 在 $\boldsymbol{A}$ 中选取非对角线元素中绝对值最大的元素 a_{pq}；

(3) 对于给定的精度控制参数 ε，如果 $|a_{pq}| < \varepsilon$，则迭代过程结束。此时，对角线元素 $a_{ij}(i = 0, 1, \Lambda, n-1)$ 即为特征值 λ_i，矩阵 $\boldsymbol{S}$ 的第 i 列为与 λ_i 对应的特征向量；如果 $|a_{pq}| \geq \varepsilon$，则继续下一步迭代计算。

(4) 用如下公式计算平面旋转矩阵的元素及用其进行变换后的矩阵 $\boldsymbol{A}_1$ 的元素：

$$x = -a_{pq}$$

$$y = \frac{1}{2}(a_{qq} - a_{pp})$$

$$w = \text{sign}(y)\frac{x}{\sqrt{x^2 + y^2}}$$

$$\sin 2\theta = w$$

$$\sin\theta = \frac{w}{\sqrt{2(1+\sqrt{1-w^2})}}$$

$$\cos\theta = \sqrt{1-\sin^2\theta}$$

$$a_{pp}^{(1)} = a_{pp}\cos^2\theta + a_{qq}\sin^2\theta + a_{pq}\cos 2\theta$$

$$a_{qq}^{(1)} = a_{pp}\sin^2\theta - a_{qq}\cos^2\theta - a_{pq}\sin 2\theta$$

$$a_{pq}^{(1)} = a_{qp}^{(1)} = 0$$

$$a_{pj}^{(1)} = a_{pj}\cos\theta + a_{qj}\sin\theta, \quad j \neq p, q$$

$$a_{qj}^{(1)} = -a_{pj}\sin\theta + a_{qj}\cos\theta, \quad j \neq p, q$$

$$a_{ip}^{(1)} = a_{ip}\cos\theta + a_{iq}\sin\theta, \quad i \neq p, q$$

$$a_{ip}^{(1)} = -a_{ip}\sin\theta + a_{iq}\cos\theta, \quad i \neq p, q$$

$$a_{ij}^{(1)} = a_{ij}, \quad i, j \neq p, q$$

$$S = S \cdot R(p, q, \theta)$$

(5) **重复**(2)**—**(5)**步计算。**

2. **算法实现**

根据上述方法，可以定义求实对称矩阵特征值与特征向量的雅可比法的 Visual Basic **函数** MJacobiEigenv**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MJacobiEigenv
'  功能：  用雅可比法(Jacobi)计算对称矩阵的特征值和特征向量
'  参数：   n    - Integer型变量，对称矩阵的阶数。
'          dblA - Double型二维数组，体积为n x n。存放对称矩阵；
'                 返回时，对角线上存放求得的n个特征值。
'          dblV - Double型二维数组，体积为n x n。
'                 返回n个特征向量，其中第i列为第i个特征值dblA(i,i)对应的特征向量。
'          eps  - Double型变量。迭代过程中的控制精度参数。
'          nMaxItNum    - Integer。为求得一个特征值所允许的最大迭代次数。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function MJacobiEigenv(n As Integer, dblA() As Double, dblV() As Double, _
eps As Double, nMaxItNum As Integer) As Boolean
    '局部变量
```

```
Dim i As Integer, j As Integer, p As Integer, q As Integer, l As Integer
Dim fm As Double, cn As Double, sn As Double, omega As Double, _
     x As Double, y As Double, d As Double

' 赋初值，建立单位矩阵
l = 1
For i = 1 To n
   dblV(i, i) = 1#
   For j = 1 To n
     If (i <> j) Then dblV(i, j) = 0#
   Next j
Next i

' 循环迭代计算
While (True)
   ' 选取绝对值最大的对角线元素
   fm = 0#
   For i = 2 To n
      For j = 1 To i - 1
         d = Abs(dblA(i, j))
         If ((i <> j) And (d > fm)) Then
            fm = d
            p = i
            q = j
         End If
      Next j
   Next i
   ' 符合精度控制要求，计算成功，结束返回
   If (fm < eps) Then
      MJacobiEigenv = True
      Exit Function
   End If
   ' 经过设定的迭代次数后仍然不能达到精度要求，计算失败，返回
   If (l > nMaxItNum) Then
      MJacobiEigenv = False
      Exit Function
   End If
   ' 正交变换，计算变换后矩阵的各元素
   l = l + 1
   x = -dblA(p, q)
   y = (dblA(q, q) - dblA(p, p)) / 2#
   omega = x / Sqr(x * x + y * y)
   If (y < 0#) Then omega = -omega
   sn = 1# + Sqr(1# - omega * omega)
   sn = omega / Sqr(2# * sn)
   cn = Sqr(1# - sn * sn)
   fm = dblA(p, p)
   dblA(p, p) = fm * cn * cn + dblA(q, q) * sn * sn + dblA(p, q) * omega
   dblA(q, q) = fm * sn * sn + dblA(q, q) * cn * cn - dblA(p, q) * omega
   dblA(p, q) = 0#
```

```
        dblA(q, p) = 0#
        For j = 1 To n
            If ((j <> p) And (j <> q)) Then
                fm = dblA(p, j)
                dblA(p, j) = fm * cn + dblA(q, j) * sn
                dblA(q, j) = -fm * sn + dblA(q, j) * cn
            End If
        Next j
        For i = 1 To n
          If ((i <> p) And (i <> q)) Then
              fm = dblA(i, p)
              dblA(i, p) = fm * cn + dblA(i, q) * sn
              dblA(i, q) = -fm * sn + dblA(i, q) * cn
          End If
        Next i
        For i = 1 To n
            fm = dblV(i, p)
            dblV(i, p) = fm * cn + dblV(i, q) * sn
            dblV(i, q) = -fm * sn + dblV(i, q) * cn
        Next i
    Wend
End Function
```

3. 示例

调用函数 MJacobiEigenv **求下列** 3 **阶矩阵** $\boldsymbol{A}$ **的全部特征值和特征向量：**

$$\boldsymbol{A}=\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim i As Integer, j As Integer
    Dim n As Integer, nIt As Integer
    Dim eps As Double
    n = 3
    nIt = 60
    eps = 0.000001
    ReDim mtxA(n, n) As Double
    ReDim mtxB(n, n) As Double
    ReDim mtxV(n, n) As Double
    Dim sEigenVector As String, sEigenValue As String
    ' 原矩阵
    mtxA(1, 1) = 2:  mtxA(1, 2) = -1: mtxA(1, 3) = 0
    mtxA(2, 1) = -1:  mtxA(2, 2) = 2: mtxA(2, 3) = -1
    mtxA(3, 1) = 0:  mtxA(3, 2) = -1: mtxA(3, 3) = 2
    ' 备份原矩阵
    For i = 1 To n
```

```
        For j = 1 To n
            mtxB(i, j) = mtxA(i, j)
        Next j
    Next i
    ' 求解特征值和特征向量
    If MJacobiEigenv(n, mtxA, mtxV, eps, nIt) Then
        For i = 1 To n
            For j = 1 To n
                If (i = j) Then
                    sEigenValue = sEigenValue + _
                          Format(mtxA(i, j), "######0.0000000") + "  "
                End If
            Next j
            sEigenVector = sEigenVector & i & ": " & _
                MatrixColToString(n, i, mtxV, "######0.0000000") & Chr(13)
        Next i
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) &
_
               "原矩阵为" & Chr$(13) & Chr$(13) &
_
              MatrixToString(n, n, mtxB,
"######0.0000000") & Chr$(13) & _
               Chr$(13) & "特征值为" & Chr$(13)
& Chr$(13) & _
               sEigenValue & Chr$(13) & Chr$(13)
& _
               "特征向量为" & Chr$(13) & Chr$(13)
& _
               sEigenVector
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

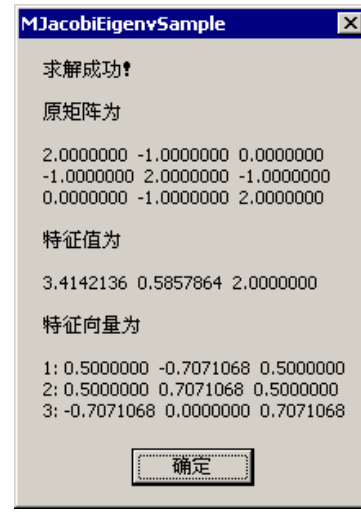

图 2.18　程序输出结果

其输出结果如图 2.18 **所示。**

2.18　求实对称矩阵特征值与特征向量的雅可比过关法

1. **算法原理**

求实对称矩阵特征值与特征向量的雅可比过关法的基本方法与 2.17 **节介绍的求实对称矩阵特征值与特征向量的雅可比法基本相同，只是在选取非对角线上的绝对值最大的元素时采用如下方法：**

(1) **计算实对称矩阵** A **的非对角线元素的平方和的平方根：**

$$v_0 = \left(2\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} a_{ij}^2 \right)^{\frac{1}{2}}$$

(2) **设置阈值** $v_1 = v_0/n$ **，在非对角线元素中按行扫描选取第一个绝对值大于或等于** v_1 **的元素** a_{pq} **进行平面旋转变换，直到所有非对角线元素的绝对值都小于** v_1 **为止。**

(3) **再设置阈值** $v_2 = v_1/n$ **，重复以上过程。**

(4) **依此类推，直到对于某个** $v_k < \varepsilon$ **为止。**

2. **算法实现**

根据上述方法，可以定义求实对称矩阵特征值与特征向量的雅可比过关法的 Visual Basic **函数** MHbergEigenv2**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：MatrixModule.bas
'  函数名：MJacobiEigenv2
'  功能：  用雅可比(Jacobi)过关法计算对称矩阵的特征值和特征向量
'  参数：   n    - Integer型变量，对称矩阵的阶数。
'         dblA - Double型二维数组，体积为n x n。存放对称矩阵；
'                返回时，对角线上存放求得的n个特征值。
'         dblV - Double型二维数组，体积为n x n。返回n个特征向量，
'                其中第i列为第i个特征值dblA(i,i)对应的特征向量。
'         eps  - Double型变量。迭代过程中的控制精度参数。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MJacobiEigenv2(n As Integer, dblA() As Double, dblV() As Double, _
eps As Double)
    ' 局部变量
    Dim i As Integer, j As Integer, p As Integer, q As Integer
    Dim ff As Double, fm As Double, cn As Double, sn As Double, _
        omega As Double, x As Double, y As Double, d As Double

    ' 赋初值，建立单位矩阵
    For i = 1 To n
        dblV(i, i) = 1#
        For j = 1 To n
            If (i <> j) Then dblV(i, j) = 0#
        Next j
    Next i

    ff = 0#
    For i = 2 To n
        For j = 1 To i
            d = dblA(i, j)
            ff = ff + d * d
        Next j
    Next i
```

```
    ff = Sqr(2# * ff)

    ' 循环迭代计算
loop_0:

    ff = ff / (1# * n)

loop_1:

    ' 选取对角线元素中的绝对值最大的元素
    For i = 2 To n
        For j = 1 To i - 1
            d = Abs(dblA(i, j))
            If (d > ff) Then
                p = i
                q = j
                GoTo loop_2
            End If
        Next j
    Next i

' 计算已达到精度要求，返回
 If (ff < eps) Then Exit Sub

    GoTo loop_0

loop_2:

     ' 计算正交变换后矩阵的各元素
     x = -dblA(p, q)
     y = (dblA(q, q) - dblA(p, p)) / 2#
     omega = x / Sqr(x * x + y * y)
     If (y < 0#) Then omega = -omega
     sn = 1# + Sqr(1# - omega * omega)
     sn = omega / Sqr(2# * sn)
     cn = Sqr(1# - sn * sn)
     fm = dblA(p, p)
     dblA(p, p) = fm * cn * cn + dblA(q, q) * sn * sn + dblA(p, q) * omega
     dblA(q, q) = fm * sn * sn + dblA(q, q) * cn * cn - dblA(p, q) * omega
     dblA(p, q) = 0#
     dblA(q, p) = 0#

     For j = 1 To n
         If ((j <> p) And (j <> q)) Then
             fm = dblA(p, j)
             dblA(p, j) = fm * cn + dblA(q, j) * sn
             dblA(q, j) = -fm * sn + dblA(q, j) * cn
         End If
     Next j

     For i = 1 To n
```

```
        If ((i <> p) And (i <> q)) Then
            fm = dblA(i, p)
            dblA(i, p) = fm * cn + dblA(i, q) * sn
            dblA(i, q) = -fm * sn + dblA(i, q) * cn
        End If
    Next i

    For i = 1 To n
        fm = dblV(i, p)
        dblV(i, p) = fm * cn + dblV(i, q) * sn
        dblV(i, q) = -fm * sn + dblV(i, q) * cn
    Next i

   GoTo loop_1

End Function
```

3. 示例

调用函数 MJacobiEigenv2 **求下列** 5 **阶实对称矩阵** A **的全部特征值和特征向量：**

$$A=\begin{bmatrix}10 & 1 & 2 & 3 & 4\\ 1 & 9 & -1 & 2 & -3\\ 2 & -1 & 7 & 3 & -5\\ 3 & 2 & 3 & 12 & -1\\ 4 & -3 & -5 & -1 & 15\end{bmatrix}$$

程序代码如下：

```
Sub Main()
   Dim i As Integer, j As Integer
   Dim n As Integer, nIt As Integer
   Dim eps As Double

   n = 5
   nIt = 60
   eps = 0.000001

   ReDim mtxA(n, n) As Double
   ReDim mtxB(n, n) As Double
   ReDim mtxV(n, n) As Double
   Dim sEigenVector As String, sEigenValue As String

   ' 原矩阵
   mtxA(1,1)=10:  mtxA(1,2)=1: mtxA(1,3)=2: mtxA(1,4)=3: mtxA(1,5)=4
   mtxA(2,1)=1:  mtxA(2,2)=9: mtxA(2,3)=-1: mtxA(2,4)=2: mtxA(2,5)=-3
   mtxA(3,1)=2:  mtxA(3,2)=-1: mtxA(3,3)=7: mtxA(3,4)=3: mtxA(3,5)=-5
   mtxA(4,1)=3:  mtxA(4,2)=2: mtxA(4,3)=3:  mtxA(4,4)=12: mtxA(4,5)=-1
   mtxA(5,1)=4:  mtxA(5,2)=-3: mtxA(5,3)=-5: mtxA(5,4)=-1: mtxA(5,5)=15
```

```
    ' 备份原矩阵
    For i = 1 To n
        For j = 1 To n
            mtxB(i, j) = mtxA(i, j)
        Next j
    Next i

    ' 求解特征值和特征向量
    Call MJacobiEigenv2(n, mtxA, mtxV, eps)

    For i = 1 To n
        For j = 1 To n
            If (i = j) Then sEigenValue = sEigenValue + _
                                Format(mtxA(i, j), "######0.0000000") + "
"
        Next j

        sEigenVector = sEigenVector & i & ": " & _
             MatrixColToString(n, i, mtxV, "######0.0000000") & Chr(13)
    Next i

    MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
           "原矩阵为" & Chr$(13) & Chr$(13) & _
           MatrixToString(n, n, mtxB, "######0.0000000") & Chr$(13) & _
            Chr$(13) & "特征值为" & Chr$(13) & Chr$(13) & _
           sEigenValue & Chr$(13) & Chr$(13) & _
           "特征向量为" & Chr$(13) & Chr$(13) & _
           sEigenVector
End Sub
```

其输出结果如图 2.19 **所示。**

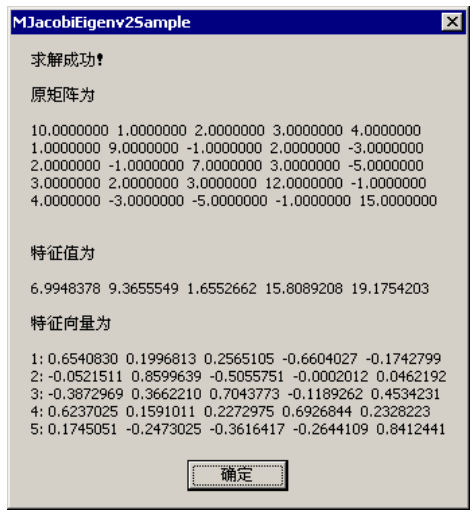

图 2.19　程序输出结果

第 3 章　线性代数方程组的求解

线性代数方程组又称为线性方程组。$m\times n$(阶)线性(代数)方程组包括 m 个方程 n 个未知数，其基本形式如下：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{\mathrm{m1}}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

矩阵 $\boldsymbol{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$ 为方程组的系数矩阵，矩阵 $\boldsymbol{B}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n}b_1 \\ a_{21} & a_{22} & \cdots & a_{2n}b_2 \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn}b_m \end{pmatrix}$ 为方程组的增广矩阵。

列向量 $\boldsymbol{b}=\begin{bmatrix} b_1 \\ b_2 \\ \mathrm{M} \\ b_n \end{bmatrix}$ 称为方程组的右端项、常向量或右端向量，列向量 $\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \\ \mathrm{M} \\ x_n \end{bmatrix}$ 称为方程组的未知向量。

线性代数方程组在科学和工程计算中有着广泛的应用。本章将介绍科学和工程应用中常用的线性代数方程组的求解方法，包括全选主元高斯消去法、全选主元高斯-约当消去法、复系数方程组的全选主元高斯消去法、复系数方程组的全选主元高斯-约当消去法、求解三对角线方程组的追赶法、一般带型方程组的求解、求解对称方程组的分解法、求解对称正定方程组的平方根法、求解大型稀疏方程组的全选主元高斯-约当消去法、求解托伯利兹方程组的列文逊方法、高斯-赛德尔迭代法、求解对称正定方程组的共轭梯度法、求解线性最小二乘问题的豪斯荷尔德变换法、求解线性最小二乘问题的广义逆法和病态方程组的求解等方法。

3.1　全选主元高斯消去法

1. 算法原理

本算法用全选主元高斯(Gauss)消去法求解 n 阶线性代数方程组 $\boldsymbol{AX}=\boldsymbol{B}$，其中，

参数矩阵 $A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$，常数向量 $B=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$，变量向量 $X=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$。

全选主元高斯消去法的求解过程可分为消元、回代和次序调整 3 个过程。

1)　消元过程

对于 k 从 1 到 n-1 作如下 3 步操作：

(1)　从系数矩阵 A 的第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素，并通过行交换与列交换，将它交换到主元素的位置上。

(2)　归一化

$$a_{kj}/a_{kk} \Rightarrow a_{kj},\quad j=k+1,\cdots,n$$
$$b_k/a_{kk} \Rightarrow b_k$$

(3)　消元

$$a_{ij}-a_{ik}a_{kj} \Rightarrow a_{ij},\quad i,\ j=k+1,\cdots,n$$
$$b_i-a_{ik}b_k \Rightarrow b_i,\quad i=k+1,\cdots,n$$

2)　回代过程

(1)　$b_n/a_{nn} \Rightarrow x_n$

(2)　$b_i-\sum\limits_{j=i+1}^{n} a_{ij}x_j \Rightarrow x_i,\quad i=n-1,\cdots,1,0$

3)　最后对解向量中的元素顺序进行调整。

全选主元高斯消去法需要通过大量运算来寻找最大模元素和交换行列，因此速度较慢。但该方法的求解精度较高，在实际中应用很广。

2. 算法实现

根据上述的运算方法，可以定义全选主元高斯消去法的 Visual Basic 函数 LEGauss，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  功能：  求解线性方程组
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEGauss
'  功能：  使用全选主元高斯消去法求解线性代数方程组
'  参数    n     - Integer型变量，线性代数方程组的阶数
'         dblA   - Double型 n x n 二维数组，线性代数方程组的系数矩阵
'         dblB   - Double型长度为 n 的一维数组，线性代数方程组的常数向量，
'                  返回方程组的解向量
```

```
' 返回值：Boolean型，求解成功为True，无解或求解失败为False
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LEGauss(n As Integer, dblA() As Double, _
dblB() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim nIs As Integer
    ReDim nJs(n) As Integer
    Dim d As Double, t As Double

    ' 开始求解
    For k = 1 To n - 1
        d = 0#

        ' 归一
        For i = k To n
            For j = k To n
                t = Abs(dblA(i, j))
                If t > d Then
                    d = t
                    nJs(k) = j
                    nIs = i
                End If
            Next j
        Next i

        ' 无解，返回
        If d + 1# = 1# Then
            LEGauss = False
            Exit Function
        End If

        ' 消元
        If nJs(k) <> k Then
            For i = 1 To n
                t = dblA(i, k)
                dblA(i, k) = dblA(i, nJs(k))
                dblA(i, nJs(k)) = t
            Next i
        End If

        If nIs <> k Then
            For j = k To n
                t = dblA(k, j)
                dblA(k, j) = dblA(nIs, j)
                dblA(nIs, j) = t
            Next j
            t = dblB(k)
            dblB(k) = dblB(nIs)
            dblB(nIs) = t
        End If
```

```
        d = dblA(k, k)
        For j = k + 1 To n
            dblA(k, j) = dblA(k, j) / d
        Next j

        dblB(k) = dblB(k) / d
        For i = k + 1 To n
            For j = k + 1 To n
                dblA(i, j) = dblA(i, j) - dblA(i, k) * dblA(k, j)
            Next j
            dblB(i) = dblB(i) - dblA(i, k) * dblB(k)
        Next i
    Next k

    d = dblA(n, n)

    ' 无解，返回
    If Abs(d) + 1# = 1# Then
        LEGauss = False
        Exit Function
    End If

    ' 回代
    dblB(n) = dblB(n) / d
    For i = n - 1 To 1 Step -1
        t = 0#
        For j = i + 1 To n
          t = t + dblA(i, j) * dblB(j)
        Next j
        dblB(i) = dblB(i) - t
    Next i

    ' 调整解的次序
    nJs(n) = n
    For k = n To 1 Step -1
        If nJs(k) <> k Then
            t = dblB(k)
            dblB(k) = dblB(nJs(k))
            dblB(nJs(k)) = t
        End If
    Next k

    ' 求解成功
    LEGauss = True

End Function
```

3. 示例

调用上述的 LEGauss **函数类求解如下的** 4 **阶方程组：**

$$\begin{cases} 0.2368x_0 + 0.2471x_1 + 0.2568x_2 + 1.2671x_3 = 1.8471 \\ 0.1968x_0 + 0.2071x_1 + 1.2168x_2 + 0.2271x_3 = 1.7471 \\ 0.1581x_0 + 1.1675x_1 + 0.1768x_2 + 0.1871x_3 = 1.6471 \\ 1.1161x_0 + 0.1254x_1 + 0.1397x_2 + 0.1490x_3 = 1.5471 \end{cases}$$

程序代码如下：

```
Sub Main()
    Dim dblA(4, 4) As Double
    Dim dblB(4) As Double

    '系数矩阵
    dblA(1,1)=0.2368:dblA(1,2)=0.2471:dblA(1,3)=0.2568:dblA(1,4)=1.2671
    dblA(2,1)=0.1968:dblA(2,2)=0.2071:dblA(2,3)=1.2168:dblA(2,4)=0.2271
    dblA(3,1)=0.1581:dblA(3,2)=1.1675:dblA(3,3)=0.1768:dblA(3,4)=0.1871
    dblA(4,1)=1.1161:dblA(4,2)=0.1254:dblA(4,3)=0.1397:dblA(4,4)=0.149
    '常数向量
    dblB(1) = 1.8471:  dblB(2) = 1.7471:  dblB(3) = 1.6471:  dblB(4) = 1.5471

    '求解
    If LEGauss(4, dblA, dblB) = True Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & "x1
= " & dblB(1) & _
               Chr$(13) & "x2 = " & dblB(2) & Chr$(13)
& _
               "x3 = " & dblB(3) & Chr$(13) & "x4 = "
& dblB(4)
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

图 3.1 程序输出结果

输出结果如图 3.1 **所示。**

3.2 全选主元高斯-约当消去法

1. 算法原理

本算法用全选主元高斯-约当(Gauss-Jordan)**消去法求解** m **个** n **阶线性代数方程组** $\boldsymbol{AX}=\boldsymbol{B}$**，其中，**

参数矩阵 $\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ a_{n1} & a_{n2} & \Lambda & a_{nn} \end{bmatrix}$**，常数矩阵** $\boldsymbol{B}=\begin{bmatrix} b_{11}\Lambda & b_{1m} \\ b_{21}\Lambda & b_{2m} \\ \mathrm{M}\,\mathrm{M} & \mathrm{M} \\ b_{n1}\Lambda & b_{nm} \end{bmatrix}$**，变量矩阵**

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \Lambda & x_{1m} \\ x_{21} & \Lambda & x_{2m} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} \\ x_{n1} & \Lambda & x_{nm} \end{bmatrix}。$$

全选主元高斯-约当消去法的求解过程如下：

1)　**消元过程**

对于 k **从** 1 **到** n **作如下** 3 **步操作：**

(1)　**从系数矩阵** $\boldsymbol{A}$ **的第** k **行、第** k **列开始的右下角子阵中选取绝对值最大的元素，并通过行交换与列交换，将它交换到主元素的位置上。**

(2)　**归一化**

$$a_{kj} / a_{kk} \Rightarrow a_{kj} \qquad j = k+1, \Lambda, n$$

$$b_{kj} / a_{kk} \Rightarrow b_{kj} \qquad j = 1, 2, \Lambda, m$$

(3)　**消元**

$$a_{ij} - a_{ik}a_{kj} \Rightarrow a_{ij}, \qquad j = k+1, \Lambda, n, \qquad i = 1, 2, \Lambda, n, \qquad i \neq k$$

$$b_{ij} - a_{ik}b_{kj} \Rightarrow b_{ij}, \qquad j = 1, 2, \Lambda, m, \qquad i = 1, 2, \Lambda, n, \qquad i \neq k$$

2)　**回代过程**

最后对矩阵 $\boldsymbol{B}$ **中的元素顺序进行调整，则** $\boldsymbol{B}$ **中的每一列即是对应的方程组的解向量。**

2. 算法实现

根据上述的运算方法,可以定义全选主元高斯-约当消去法的 Visual Basic **函数** LEGauss Jordan**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEGaussJordan
'  功能：  使用全选主元高斯-约当消去法求解线性代数方程组
'  参数     n     - Integer型变量，线性代数方程组的阶数
'           m    - Integer型变量，线性代数方程组的个数，即右端常数矩阵列向量的个数
'          dblA   - Double型 n x n 二维数组，线性代数方程组的系数矩阵
'          dblB   - Double型n x m二维数组，线性代数方程组的常数矩阵，
'                      返回方程组的解矩阵
'  返回值：Boolean型，求解成功为True，无解或求解失败为False
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LEGaussJordan(n As Integer, m As Integer, _
dblA() As Double, dblB() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim nIs As Integer
    ReDim nJs(n) As Integer
    Dim d As Double, q As Double

    ' 开始求解
    For k = 1 To n
        q = 0#
```

```
' 归一
For i = k To n
   For j = k To n
      If Abs(dblA(i, j)) > q Then
         q = Abs(dblA(i, j))
         nJs(k) = j
         nIs = i
      End If
   Next j
Next i

' 无解，返回
If q + 1# = 1# Then
   LEGaussJordan = False
   Exit Function
End If

' 消元
' A->
For j = k To n
   d = dblA(k, j)
   dblA(k, j) = dblA(nIs, j)
   dblA(nIs, j) = d
Next j
' B->
For j = 1 To m
   d = dblB(k, j)
   dblB(k, j) = dblB(nIs, j)
   dblB(nIs, j) = d
Next j
'A->
For i = 1 To n
   d = dblA(i, k)
   dblA(i, k) = dblA(i, nJs(k))
   dblA(i, nJs(k)) = d
Next i

For j = k + 1 To n
   dblA(k, j) = dblA(k, j) / dblA(k, k)
Next j
For j = 1 To m
   dblB(k, j) = dblB(k, j) / dblA(k, k)
Next j

' 回代
For j = k + 1 To n
  For i = 1 To n
    If i <> k Then
      dblA(i, j) = dblA(i, j) - dblA(i, k) * dblA(k, j)
    End If
```

```
        Next i
      Next j

      For j = 1 To m
        For i = 1 To n
          If i <> k Then
            dblB(i, j) = dblB(i, j) - dblA(i, k) * dblB(k, j)
          End If
        Next i
      Next j
   Next k

   ' 调整解的次序
   For k = n To 1 Step -1
      For j = 1 To m
         d = dblB(k, j)
         dblB(k, j) = dblB(nJs(k), j)
         dblB(nJs(k), j) = d
      Next j
   Next k

   ' 求解成功
   LEGaussJordan = True

End Function
```

3. 示例

调用上述 LEGaussJordan 函数类求解如下的 4 阶方程组：

$$\begin{bmatrix} 1 & 3 & 2 & 13 \\ 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -3 & 11 & 5 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 6 & 4 \\ 11 & 7 \\ -2 & -1 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
   Dim dblA(4, 4) As Double
   Dim dblB(4, 2) As Double

   '系数矩阵
   dblA(1, 1) = 1:   dblA(1, 2) = 3:   dblA(1, 3) = 2:   dblA(1, 4) = 13
   dblA(2, 1) = 7:   dblA(2, 2) = 2:   dblA(2, 3) = 1:   dblA(2, 4) = -2
   dblA(3, 1) = 9:   dblA(3, 2) = 15:   dblA(3, 3) = 3:   dblA(3, 4) = -2
   dblA(4, 1) = -2:   dblA(4, 2) = -2:  dblA(4, 3) = 11:  dblA(4, 4) = 5
   '常数矩阵
   dblB(1, 1) = 9:   dblB(1, 2) = 0
   dblB(2, 1) = 6:   dblB(2, 2) = 4
   dblB(3, 1) = 11:   dblB(3, 2) = 7
   dblB(4, 1) = -2:   dblB(4, 2) = -1
```

```
    '求解
    If LEGaussJordan(4, 2, dblA, dblB) = True Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x11 = " & Round(dblB(1, 1), 5) & "   x12 = " & _
              Round(dblB(1, 2), 5) & Chr$(13) & _
              "x21 = " & Round(dblB(2, 1), 5) & "   x22 = " & _
              Round(dblB(2, 2), 5) & Chr$(13) & _
              "x31 = " & Round(dblB(3, 1), 5) & "
x32 = " & _
              Round(dblB(3, 2), 5) & Chr$(13) & _
              "x41 = " & Round(dblB(4, 1), 5) & "
x42 = " & _
              Round(dblB(4, 2), 5)
    Else
       MsgBox "求解失败！"
    End If

End Sub
```

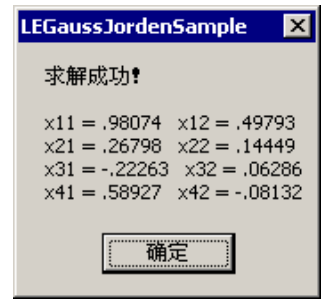

图 3.2 程序输出结果

输出结果如图 3.2 所示。

3.3 复系数方程组的全选主元高斯消去法

1. 算法原理

本算法用全选主元高斯消去法求解复系数线性代数方程组 $\boldsymbol{AX} = \boldsymbol{B}$，其中，

$$\boldsymbol{A} = \boldsymbol{AR} + \mathrm{j}\boldsymbol{AI}$$
$$\boldsymbol{B} = \boldsymbol{BR} + \mathrm{j}\boldsymbol{BI}$$
$$\boldsymbol{X} = \boldsymbol{XR} + \mathrm{j}\boldsymbol{XI}$$

与 3.1 节中的实系数线性方程组的求解类似，全选主元高斯消去法包括消元、回代和次序调整 3 个过程，只是所有运算都是复数运算。

复数的乘法采用如下的快速算法：

$$(a + \mathrm{j}b)(c + \mathrm{j}d) = (p - q) + \mathrm{j}(s - p - q)$$

其中，

$p = ac$

$q = bd$

$s = (a + b)(c + d)$

复数的除法采用如下的快速算法：

$$(c + \mathrm{j}d)/(a + \mathrm{j}b) = (p - q)/w + (s - p - q)/w$$

其中，

$p = ac$

$q = -bd$

$s = (a - b)(c + d)$

$w = a^2 + b^2$

2. 算法实现

根据上述的运算方法，可以定义复系数方程组的全选主元高斯消去法的 Visual Basic **函数** LECpxGauss**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LECpxGauss
'  功能：  使用全选主元高斯消去法求解复系数线性代数方程组
'  参数    n     - Integer型变量，线性代数方程组的阶数
'        dblAR  - Double型 n x n 二维数组，线性代数方程组的系数矩阵的实部
'        dblAI   - Double型 n x n 二维数组，线性代数方程组的系数矩阵的虚部
'        dblBR  - Double型长度为 n 的一维数组，线性代数方程组的常数向量的实部，
'                 返回方程组的解向量的实部
'        dblBI   - Double型长度为 n 的一维数组，线性代数方程组的常数向量的虚部，
'                 返回方程组的解向量的虚部
'  返回值：Boolean型，求解成功为True，无解或求解失败为False
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LECpxGauss(n As Integer, dblAR() As Double, _
dblAI() As Double, dblBR() As Double, dblBI() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim nIs As Integer
    ReDim nJs(n) As Integer
    Dim d As Double, p As Double, q As Double, s As Double

    ' 开始求解
    For k = 1 To n - 1
        d = 0#

        ' 归一
        For i = k To n
            For j = k To n
                p = dblAR(i, j) * dblAR(i, j) + dblAI(i, j) * dblAI(i, j)
                If p > d Then
                    d = p
                    nJs(k) = j
                    nIs = i
                End If
            Next j
        Next i

        ' 无解，返回
```

```
If d + 1# = 1# Then
    LECpxGauss = False
    Exit Function
End If

' 消元
For j = k To n
    p = dblAR(k, j)
    dblAR(k, j) = dblAR(nIs, j)
    dblAR(nIs, j) = p
    p = dblAI(k, j)
    dblAI(k, j) = dblAI(nIs, j)
    dblAI(nIs, j) = p
Next j

p = dblBR(k)
dblBR(k) = dblBR(nIs)
dblBR(nIs) = p
p = dblBI(k)
dblBI(k) = dblBI(nIs)
dblBI(nIs) = p

For i = 1 To n
    p = dblAR(i, k)
    dblAR(i, k) = dblAR(i, nJs(k))
    dblAR(i, nJs(k)) = p
    p = dblAI(i, k)
    dblAI(i, k) = dblAI(i, nJs(k))
    dblAI(i, nJs(k)) = p
Next i

' 复数运算
For j = k + 1 To n
    p = dblAR(k, j) * dblAR(k, k)
    q = -dblAI(k, j) * dblAI(k, k)
    s = (dblAR(k, k) - dblAI(k, k)) * (dblAR(k, j) + dblAI(k, j))
    dblAR(k, j) = (p - q) / d
    dblAI(k, j) = (s - p - q) / d
Next j

p = dblBR(k) * dblAR(k, k)
q = -dblBI(k) * dblAI(k, k)
s = (dblAR(k, k) - dblAI(k, k)) * (dblBR(k) + dblBI(k))
dblBR(k) = (p - q) / d
dblBI(k) = (s - p - q) / d

For i = k + 1 To n
    For j = k + 1 To n
        p = dblAR(i, k) * dblAR(k, j)
        q = dblAI(i, k) * dblAI(k, j)
        s = (dblAR(i, k) + dblAI(i, k)) * (dblAR(k, j) + dblAI(k, j))
```

```
            dblAR(i, j) = dblAR(i, j) - p + q
            dblAI(i, j) = dblAI(i, j) - s + p + q
        Next j
        p = dblAR(i, k) * dblBR(k)
        q = dblAI(i, k) * dblBI(k)
        s = (dblAR(i, k) + dblAI(i, k)) * (dblBR(k) + dblBI(k))
        dblBR(i) = dblBR(i) - p + q
        dblBI(i) = dblBI(i) - s + p + q
    Next i
Next k

d = dblAR(n, n) * dblAR(n, n) + dblAI(n, n) * dblAI(n, n)

' 无解，返回
If d + 1# = 1# Then
    LECpxGauss = False
    Exit Function
End If

p = dblAR(n, n) * dblBR(n)
q = -dblAI(n, n) * dblBI(n)
s = (dblAR(n, n) - dblAI(n, n)) * (dblBR(n) + dblBI(n))
dblBR(n) = (p - q) / d
dblBI(n) = (s - p - q) / d

' 回代
For i = n - 1 To 1 Step -1
    For j = i + 1 To n
        p = dblAR(i, j) * dblBR(j)
        q = dblAI(i, j) * dblBI(j)
        s = (dblAR(i, j) + dblAI(i, j)) * (dblBR(j) + dblBI(j))
        dblBR(i) = dblBR(i) - p + q
        dblBI(i) = dblBI(i) - s + p + q
    Next j
Next i

' 调整解的次序
nJs(n) = n
For k = n To 1 Step -1
    p = dblBR(k)
    dblBR(k) = dblBR(nJs(k))
    dblBR(nJs(k)) = p

    p = dblBI(k)
    dblBI(k) = dblBI(nJs(k))
    dblBI(nJs(k)) = p
Next k

' 求解成功
LECpxGauss = True
```

```
End Function
```

3. 示例

调用上述 LECpxGauss **函数求解复系数方程组** $\boldsymbol{AX}=\boldsymbol{B}$，**其中，**

$$\boldsymbol{A}=\boldsymbol{AR}+\mathrm{j}\boldsymbol{AI},\qquad \boldsymbol{B}=\boldsymbol{BR}+\mathrm{j}\boldsymbol{BI}$$

$$\boldsymbol{AR}=\begin{bmatrix}1 & 3 & 2 & 13\\ 7 & 2 & 1 & -2\\ 9 & 15 & 3 & -2\\ -2 & -2 & 11 & 5\end{bmatrix},\ \boldsymbol{AI}=\begin{bmatrix}3 & -2 & 1 & 6\\ -2 & 7 & 5 & 8\\ 9 & -3 & 15 & 1\\ -2 & -2 & 7 & 6\end{bmatrix},\ \boldsymbol{BR}=\begin{bmatrix}2\\7\\3\\9\end{bmatrix},\ \boldsymbol{BI}=\begin{bmatrix}1\\2\\-2\\3\end{bmatrix}。$$

程序代码如下：

```
Sub Main()
    Dim dblAR(4, 4) As Double, dblAI(4, 4) As Double
    Dim dblBR(4) As Double, dblBI(4) As Double
    Dim sCpx1 As String, sCpx2 As String, sCpx3 As String, sCpx4 As String

    '系数矩阵
    dblAR(1, 1) = 1:  dblAR(1, 2) = 3:  dblAR(1, 3) = 2:  dblAR(1, 4) = 13
    dblAR(2, 1) = 7:  dblAR(2, 2) = 2:  dblAR(2, 3) = 1:  dblAR(2, 4) = -2
    dblAR(3, 1) = 9:  dblAR(3, 2) = 15:  dblAR(3, 3) = 3:  dblAR(3, 4) = -2
    dblAR(4, 1) = -2: dblAR(4, 2) = -2:  dblAR(4, 3) = 11: dblAR(4, 4) = 5

    dblAI(1, 1) = 3:  dblAI(1, 2) = -2:  dblAI(1, 3) = 1:  dblAI(1, 4) = 6
    dblAI(2, 1) = -2: dblAI(2, 2) = 7:   dblAI(2, 3) = 5:   dblAI(2, 4) = 8
    dblAI(3, 1) = 9:  dblAI(3, 2) = -3:   dblAI(3, 3) = 15: dblAI(3, 4) = 1
    dblAI(4, 1) = -2: dblAI(4, 2) = -2:   dblAI(4, 3) = 7:  dblAI(4, 4) = 6
    '常数向量
    dblBR(1) = 2:    dblBR(2) = 7:    dblBR(3) = 3:    dblBR(4) = 9
    dblBI(1) = 1:    dblBI(2) = 2:    dblBI(3) = -2:    dblBI(4) = 3

    '求解
    If LECpxGauss(4, dblAR, dblAI, dblBR, dblBI) = True Then
        sCpx1 = dblBR(1) & " + " & dblBI(1) & "j"
        sCpx2 = dblBR(2) & " + " & dblBI(2) & "j"
        sCpx3 = dblBR(3) & " + " & dblBI(3) & "j"
        sCpx4 = dblBR(4) & " + " & dblBI(4) & "j"
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & "x1 = " & sCpx1 & _
            Chr$(13) & "x2 = " & sCpx2 & Chr$(13) & "x3 = " & sCpx3 & _
            Chr$(13) & "x4 = " & sCpx4
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

输出结果如图 3.3 所示。

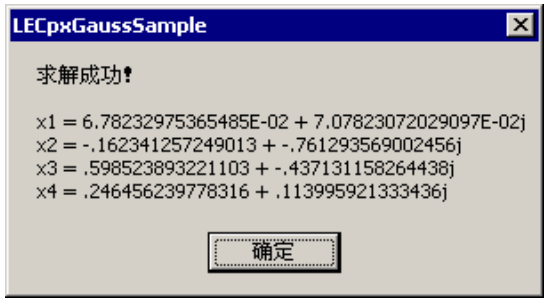

图 3.3　程序输出结果

3.4　复系数方程组的全选主元高斯-约当消去法

1. 算法原理

本算法用全选主元高斯-约当消去法求解复系数线性代数方程组 $\boldsymbol{AX}=\boldsymbol{B}$，其中，

$$\boldsymbol{A} = \boldsymbol{AR} + j\boldsymbol{AI}$$
$$\boldsymbol{B} = \boldsymbol{BR} + j\boldsymbol{BI}$$
$$\boldsymbol{X} = \boldsymbol{XR} + j\boldsymbol{XI}$$

与 3.2 节中的实系数线性方程组的求解类似，全选主元高斯消去法包括消元和回代两个过程，只是所有运算都是复数运算。

复数的乘法采用如下的快速算法：

$$(a + \mathrm{j}b)(c + \mathrm{j}d) = (p - q) + \mathrm{j}(s - p - q)$$

其中，

$p = ac$
$q = bd$
$s = (a + b)(c + d)$

复数的除法采用如下的快速算法：

$$(c + \mathrm{j}d)/(a + \mathrm{j}b) = (p - q)/w + (s - p - q)/w$$

其中，

$p = ac$
$q = -bd$
$s = (a - b)(c + d)$
$w = a^2 + b^2$

2. 算法实现

根据上述的运算方法，可以定义复系数方程组的全选主元高斯-约当消去法的 Visual

Basic 函数 LECpxGaussJordan，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LECpxGaussJordan
'  功能：  使用全选主元高斯-约当消去法求解复系数线性代数方程组
'  参数    n    - Integer型变量，线性代数方程组的阶数
'          m    - Integer型变量，方程组右端复常数向量的个数
'         dblAR - Double型 n x n 二维数组，线性代数方程组的系数矩阵的实部
'         dblAI - Double型 n x n 二维数组，线性代数方程组的系数矩阵的虚部
'         dblBR - Double型长度为 n X m 的二维数组，
'                 存放方程组右端的m组常数向量的实部，返回时存放m组解向量的实部
'         dblBI - Double型长度为 n x m 的二维数组，
'                 存放方程组右端的m组常数向量的虚部，返回时存放m组解向量的虚部
'  返回值：Boolean型，求解成功为True，无解或求解失败为False
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LECpxGaussJordan(n As Integer, m As Integer, _
dblAR() As Double, dblAI() As Double, dblBR() As Double, _
dblBI() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim nIs As Integer
    ReDim nJs(n) As Integer
    Dim d As Double, p As Double, q As Double, s As Double

    ' 开始求解
    For k = 1 To n
        d = 0#

        ' 归一
        For i = k To n
            For j = k To n
                p = dblAR(i, j) * dblAR(i, j) + dblAI(i, j) * dblAI(i, j)
                If p > d Then
                    d = p
                    nJs(k) = j
                    nIs = i
                End If
            Next j
        Next i

        ' 无解，返回
        If d + 1# = 1# Then
            LECpxGaussJordan = False
            Exit Function
        End If

        ' 消元
        If nIs <> k Then
            ' A->
            For j = k To n
```

```
            p = dblAR(k, j)
            dblAR(k, j) = dblAR(nIs, j)
            dblAR(nIs, j) = p
            p = dblAI(k, j)
            dblAI(k, j) = dblAI(nIs, j)
            dblAI(nIs, j) = p
        Next j
        ' B ->
        For j = 1 To m
            p = dblBR(k, j)
            dblBR(k, j) = dblBR(nIs, j)
            dblBR(nIs, j) = p
            p = dblBI(k, j)
            dblBI(k, j) = dblBI(nIs, j)
            dblBI(nIs, j) = p
        Next j
    End If
    If nJs(k) <> k Then
        ' A->
        For i = 1 To n
            p = dblAR(i, k)
            dblAR(i, k) = dblAR(i, nJs(k))
            dblAR(i, nJs(k)) = p
            p = dblAI(i, k)
            dblAI(i, k) = dblAI(i, nJs(k))
            dblAI(i, nJs(k)) = p
        Next i
    End If

    ' 复数运算
    For j = k + 1 To n
        p = dblAR(k, j) * dblAR(k, k)
        q = -dblAI(k, j) * dblAI(k, k)
        s = (dblAR(k, k) - dblAI(k, k)) * (dblAR(k, j) + dblAI(k, j))
        dblAR(k, j) = (p - q) / d
        dblAI(k, j) = (s - p - q) / d
    Next j
    For j = 1 To m
        p = dblBR(k, j) * dblAR(k, k)
        q = -dblBI(k, j) * dblAI(k, k)
        s = (dblAR(k, k) - dblAI(k, k)) * (dblBR(k, j) + dblBI(k, j))
        dblBR(k, j) = (p - q) / d
        dblBI(k, j) = (s - p - q) / d
    Next j

    For i = 1 To n
        If i <> k Then
            For j = k + 1 To n
                p = dblAR(i, k) * dblAR(k, j)
                q = dblAI(i, k) * dblAI(k, j)
                s = (dblAR(i, k) + dblAI(i, k)) * (dblAR(k, j) + _
```

```
                    dblAI(k, j))
                dblAR(i, j) = dblAR(i, j) - p + q
                dblAI(i, j) = dblAI(i, j) - s + p + q
            Next j
            For j = 1 To m
                p = dblAR(i, k) * dblBR(k, j)
                q = dblAI(i, k) * dblBI(k, j)
                s = (dblAR(i, k) + dblAI(i, k)) * (dblBR(k, j) + _
                    dblBI(k, j))
                dblBR(i, j) = dblBR(i, j) - p + q
                dblBI(i, j) = dblBI(i, j) - s + p + q
            Next j
        End If
    Next i
Next k

' 调整解的次序
For k = n To 1 Step -1
    If nJs(k) <> k Then
        For j = 1 To m
            p = dblBR(k, j)
            dblBR(k, j) = dblBR(nJs(k), j)
            dblBR(nJs(k), j) = p

            p = dblBI(k, j)
            dblBI(k, j) = dblBI(nJs(k), j)
            dblBI(nJs(k), j) = p
        Next j
    End If
Next k

' 求解成功
LECpxGaussJordan = True

End Function
```

3. 示例

调用上述 LECpxGaussJordan **函数求解复系数方程组** $\boldsymbol{AX}=\boldsymbol{B}$，**其中，**

$$\boldsymbol{A}=\boldsymbol{AR}+\mathrm{j}\boldsymbol{AI},\ \boldsymbol{B}=\boldsymbol{BR}+\mathrm{j}\boldsymbol{BI}$$

$$\boldsymbol{AR}=\begin{bmatrix}1&3&2&13\\7&2&1&-2\\9&15&3&-2\\-2&-2&11&5\end{bmatrix},\ \boldsymbol{AI}=\begin{bmatrix}3&-2&1&6\\-2&7&5&8\\9&-3&15&1\\-2&-2&7&6\end{bmatrix},\ \boldsymbol{BR}=\begin{bmatrix}2&-2\\7&3\\3&2\\9&1\end{bmatrix},\ \boldsymbol{BI}=\begin{bmatrix}1&3\\2&7\\-2&9\\3&2\end{bmatrix}。$$

程序代码如下：

```
Sub Main()
    Dim dblAR(4, 4) As Double, dblAI(4, 4) As Double
```

```
    Dim dblBR(4, 2) As Double, dblBI(4, 2) As Double
    Dim sCpx11 As String, sCpx12 As String, sCpx21 As String, sCpx22 As String
    Dim sCpx31 As String, sCpx32 As String, sCpx41 As String, sCpx42 As String

    '系数矩阵
    dblAR(1, 1) = 1:  dblAR(1, 2) = 3:  dblAR(1, 3) = 2:  dblAR(1, 4) = 13
    dblAR(2, 1) = 7:  dblAR(2, 2) = 2:  dblAR(2, 3) = 1:  dblAR(2, 4) = -2
    dblAR(3, 1) = 9:  dblAR(3, 2) = 15:  dblAR(3, 3) = 3:  dblAR(3, 4) = -2
    dblAR(4, 1) = -2: dblAR(4, 2) = -2:  dblAR(4, 3) = 11: dblAR(4, 4) = 5

    dblAI(1, 1) = 3:  dblAI(1, 2) = -2:  dblAI(1, 3) = 1:  dblAI(1, 4) = 6
    dblAI(2, 1) = -2: dblAI(2, 2) = 7:  dblAI(2, 3) = 5:  dblAI(2, 4) = 8
    dblAI(3, 1) = 9:   dblAI(3, 2) = -3:  dblAI(3, 3) = 15:  dblAI(3, 4) = 1
    dblAI(4, 1) = -2:  dblAI(4, 2) = -2:  dblAI(4, 3) = 7:  dblAI(4, 4) = 6

    '常数向量
    dblBR(1, 1) = 2:    dblBR(1, 2) = -2
    dblBR(2, 1) = 7:    dblBR(2, 2) = 3
    dblBR(3, 1) = 3:    dblBR(3, 2) = 2
    dblBR(4, 1) = 9:    dblBR(4, 2) = 1

    dblBI(1, 1) = 1:    dblBI(1, 2) = 3
    dblBI(2, 1) = 2:    dblBI(2, 2) = 7
    dblBI(3, 1) = -2:    dblBI(3, 2) = 9
    dblBI(4, 1) = 3:    dblBI(4, 2) = 2

    '求解
    If LECpxGaussJordan(4, 2, dblAR, dblAI, dblBR, dblBI) = True Then
       sCpx11 = Round(dblBR(1, 1), 5) & " + " & Round(dblBI(1, 1), 5) & "j"
       sCpx12 = Round(dblBR(1, 2), 5) & " + " & Round(dblBI(1, 2), 5) & "j"
       sCpx21 = Round(dblBR(2, 1), 5) & " + " & Round(dblBI(2, 1), 5) & "j"
       sCpx22 = Round(dblBR(2, 2), 5) & " + " & Round(dblBI(2, 2), 5) & "j"
       sCpx31 = Round(dblBR(3, 1), 5) & " + " & Round(dblBI(3, 1), 5) & "j"
       sCpx32 = Round(dblBR(3, 2), 5) & " + " & Round(dblBI(3, 2), 5) & "j"
       sCpx41 = Round(dblBR(4, 1), 5) & " + " & Round(dblBI(4, 1), 5) & "j"
       sCpx42 = Round(dblBR(4, 2), 5) & " + " & Round(dblBI(4, 2), 5) & "j"

       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x11 = " & sCpx11 & Chr$(13) & _
              "x21 = " & sCpx21 & Chr$(13) & _
              "x31 = " & sCpx31 & Chr$(13) & _
              "x41 = " & sCpx41 & Chr$(13) & Chr$(13) & _
              " x12 = " & sCpx12 & Chr$(13) & _
              " x22 = " & sCpx22 & Chr$(13) & _
              " x32 = " & sCpx32 & Chr$(13) & _
              " x42 = " & sCpx42
    Else
       MsgBox "求解失败！"
End If

End Sub
```

输出结果如图 3.4 所示。

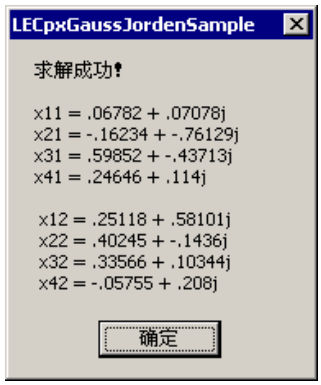

图 3.4 程序输出结果

3.5 求解三对角线方程组的追赶法

1. 算法原理

本节讨论用追赶法求解 n 阶三对角线方程组 $AX=D$，其中，

$$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & 0 & \\ & a_{32} & a_{33} & a_{34} & & \\ & & \mathrm{O} & \mathrm{O} & \mathrm{O} & \\ & 0 & & a_{n-1,\,n-2} & a_{n-1,\,n-1} & a_{n-1,\,n} \\ & & & & a_{n,\,n-1} & a_{n,\,n} \end{bmatrix}$$

$$\boldsymbol{X}=(x_1,\ x_2, x_3\Lambda\ , x_{n-1},\ x_n)^{\mathrm{T}}$$

$$\boldsymbol{D}=(d_1 d_2,\ d_3\Lambda\ ,\ d_{n-1},\ d_n)^{\mathrm{T}}$$

追赶法的本质是没有选主元的高斯消去法，只是在计算过程中考虑了三对角线矩阵的特点，对于绝大部分的零不再作处理。其步骤如下：

(1) 对于 $k=1, 2, \Lambda\ , n-1$ 作归一化及消元处理。即：

$$a_{k,\ k+1}/a_{kk} \Rightarrow a_{k,\ k+1}$$

$$d_k/a_{kk} \Rightarrow d_k$$

$$a_{k+1,\ k+1}-a_{k+1,\ k}a_{k,\ k+1} \Rightarrow a_{k+1,\ k+1}$$

$$d_{k+1}-a_{k+1,\ k}d_k \Rightarrow d_{k+1}$$

(2) 进行回代。则：

$$d_n/a_{n,\ n} \Rightarrow x_n$$

$$d_k-a_{k,\ k+1}x_{k+1} \Rightarrow x_k,\ \ k=n-1, \Lambda\ , 2, 1$$

本函数中，为便于三对角线矩阵的输入，用一个一维数组 B 以行为主存放三对角线矩

阵 A 中的三对角线上的非 0 元素。三对角线矩阵 A 和一维数组 B 的关系如下：

$$A(i,j)=\begin{cases}B(2i+j), & i-1\le j\le i+1\\ 0, & \text{其他}\end{cases}$$

其中 $i, j = 1, 2, \ldots, n$；n 为矩阵 A 的阶数。

根据上述关系，追赶法的计算过程如下：

(1) 对于 $k = 1, 2, \ldots, n-1$ 作归一化和消元处理；

(2) 回代。

最终的计算结果存放在一维数组 D 中。

2. 算法实现

根据上述的运算方法，可以定义求解三对角线方程组的追赶法的 Visual Basic 函数 LETrid，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LETrid
'  功能：  使用追赶法求解三对角线线性代数方程组
'  参数    n   - Integer型变量，线性代数方程组的阶数
'          m   - Integer型变量，n阶三对角线矩阵三对角线上元素的个数，即数组b的长度。
'                它的值应为m = 3n -2。函数应对此值进行检验。
'          dblB －Double型一维数组，长度为m。以行为主存放三对角线矩阵中三对角线上的
'                元素，即b中依次存放下列元素：'a11，a12，a21，a22，a23，a32，a33，
                a34，…，an,n-1，an,n
'          dblD －Double型一维数组，长度为n。作为传入参数，存放方程组右端的常数向量。
'                 函数返回时，此数组中存放着方程组的解向量。
'  返回值： Integer型。小于0，m的值不正确；为0，求解失败，无解；大于0，成功
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LETrid(n As Integer, m As Integer, dblB() As Double, _
dblD() As Double) As Integer
    ' 局部变量
    Dim k As Integer, j As Integer
    Dim s As Double

    ' 参数校验
    If (m <> (3 * n - 2)) Then
        LETrid = -1
        Exit Function
    End If

    ' 求解
    For k = 1 To n - 1
        j = 3 * (k - 1) + 1
        s = dblB(j)

        ' 无解，返回
        If (Abs(s) + 1# = 1#) Then
            LETrid = 0
```

```
            Exit Function
        End If

        dblB(j + 1) = dblB(j + 1) / s
        dblD(k) = dblD(k) / s
        dblB(j + 3) = dblB(j + 3) - dblB(j + 2) * dblB(j + 1)
        dblD(k + 1) = dblD(k + 1) - dblB(j + 2) * dblD(k)
    Next k

    s = dblB(3 * n - 2)

    ' 无解，返回
    If (Abs(s) + 1# = 1#) Then
        LETrid = 0
        Exit Function
    End If

    dblD(n) = dblD(n) / s
    For k = n - 1 To 1 Step -1
      dblD(k) = dblD(k) - dblB(3 * (k - 1) + 2) * dblD(k + 1)
    Next k

    ' 求解成功
    LETrid = 1

End Function
```

3. 示例

调用上述 LETrid 函数求解如下三对角线线性方程组：

$$\begin{bmatrix} 1 & 1 & & & \\ 1 & 2 & 1 & & \\ & 1 & 3 & 1 & \\ & & 1 & 4 & 1 \\ & & & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \\ 15 \\ 24 \\ 29 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim dblB(13) As Double, dblD(5) As Double

    '系数矩阵
    dblB(1) = 1
    dblB(2) = 1
    dblB(3) = 1
    dblB(4) = 2
    dblB(5) = 1
    dblB(6) = 1
    dblB(7) = 3
    dblB(8) = 1
```

```
    dblB(9) = 1
    dblB(10) = 4
    dblB(11) = 1
    dblB(12) = 1
    dblB(13) = 5
    '常数向量
    dblD(1) = 3
    dblD(2) = 8
    dblD(3) = 15
    dblD(4) = 24
    dblD(5) = 29

    '求解
    If LETrid(5, 13, dblB, dblD) > 0 Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & "x1 = "
& dblD(1) & _
          Chr$(13) & "x2 = " & dblD(2) & Chr$(13) & "x3
= " & dblD(3) & _
          Chr$(13) & "x4 = " & dblD(4) & Chr$(13) & "x5
= " & dblD(5)
    Else
       MsgBox "求解失败！"
    End If

End Sub
```

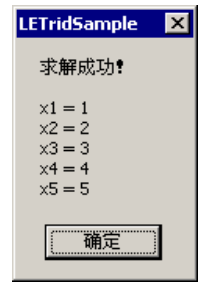

图 3.5　程序输出结果

输出结果如图 3.5 **所示。**

3.6　一般带型方程组的求解

1. 算法原理

一般采用用列选主元高斯消去法来求解右端具有 m **组常数向量的** n **阶一般带型方程组** $AX=D$**。其中** A **为** n **阶带型矩阵，其元素满足：**

$$a_{ij}\begin{cases}\neq 0, & i-l\le j\le i+1\\ =0, & \text{其他}\end{cases}$$

即

$$\boldsymbol{A}=\begin{bmatrix} a_{00} & \Lambda & a_{0l} & & & & \\ \mathrm{M} & & \mathrm{M} & \mathrm{O} & & & 0 \\ a_{l0} & \Lambda & a_{ll} & \Lambda & a_{l,\,2l} & & \\ & \mathrm{O} & & \mathrm{O} & & \mathrm{O} & \\ & & a_{n-l-1,\,n-2l-1} & \Lambda & a_{n-l-1,\;n-l-1} & \Lambda & a_{n-l-1,\,n-1} \\ & & \mathrm{O} & & \mathrm{M} & & \mathrm{M} \\ & 0 & & & a_{n-1,\,n-l-1} & \Lambda & a_{n-1,\,n-1} \end{bmatrix}$$

l 称为半带宽，带宽为 $il = 2l+1$。

$$
\boldsymbol{D}=\begin{bmatrix} d_{00} & \Lambda & d_{0,\,m-1} \\ d_{10} & \Lambda & d_{1,\,m-1} \\ \mathrm{M} & & \mathrm{M} \\ d_{n-1,\,0} & \Lambda & d_{n-1,\,m-1} \end{bmatrix}
$$

由于系数矩阵中非 0 元素是带状分布的，带区外都为 0，因此，求解时可以只考虑带区内的非 0 元素。这样可以减少很大的运算量。对于带区矩阵也只需要存储带区内的非 0 元素，可以用一个 $n\times(2l+1)$的二维数组 B 来表示。为了在消元过程中不在带区外产生新的非 0 元素，对于带区矩阵 $\boldsymbol{A}$ 中的前 $l+1$ 行和后 $l+1$ 行的元素均以左边对齐放在数组 B 的相应行，并将最右边的空余部分填入 0。

带型矩阵 $\boldsymbol{A}$ 可用二维数组 B 表示如下：

$$
\boldsymbol{B}=\begin{bmatrix} b_{00} & \Lambda & b_{0,\ l} & \Lambda & b_{0,\ 2l} \\ \mathrm{M} & & \mathrm{M} & & \mathrm{M} \\ b_{l,\,0} & \Lambda & b_{l,\ l} & \Lambda & b_{l,\ 2l} \\ \mathrm{M} & & \mathrm{M} & & \mathrm{M} \\ b_{n-l-1,\ 0} & \Lambda & b_{n-l-1,\ l} & \Lambda & b_{n-l-1,\ 2l} \\ \mathrm{M} & & \mathrm{M} & & \mathrm{M} \\ b_{n-1,\ 0} & \Lambda & b_{n-1,\ l} & \Lambda & b_{n-1,\ 2l} \end{bmatrix}
$$

$$
=\begin{bmatrix} a_{00} & \Lambda & a_{0,\ l} & & 0 \\ \mathrm{M} & \mathrm{O} & \mathrm{M} & \mathrm{O} & \\ a_{l,\ 0} & \Lambda & a_{l,\ l} & \Lambda & b_{l,\ 2l} \\ \mathrm{M} & & \mathrm{M} & & \mathrm{M} \\ a_{n-l-1,\ n-2l-1} & \Lambda & a_{n-l-1,\ n-l-1} & \Lambda & a_{n-l-1,\ n-1} \\ \mathrm{M} & \mathrm{N} & \mathrm{M} & \mathrm{N} & \\ a_{n-1,\ n-l-1} & \Lambda & a_{n-1,\ n-1} & & 0 \end{bmatrix}
$$

整个求解过程都在二维数组 B 中进行。

2. 算法实现

根据上述的运算方法，可以定义求解一般带型方程组的 Visual Basic 函数 LEBand，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEBand
'  功能：   一般带型方程组的求解
'  参数     n   - Integer型变量，线性代数方程组的阶数
'           m  - Integer型变量，为方程组右端的常数向量的个数
'           l   - Integer型变量，为系数矩阵的半带宽。
'           il   - Integer型变量，为系数矩阵的带宽。
```

```
'        dblB - Double型n x il二维数组，存放带型矩阵A中带区内的元素
'        dblD - Double型n x m二维数组，作为传入参数，存放方程组右端的m组常数向量。
'               函数返回时，其中存放着m组解向量。
'  返回值： Integer型。小于0，参数中半带宽l与带宽il的关系不对；
'                       等于0，系数矩阵A奇异，无解；大于0成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LEBand(n As Integer, m As Integer, l As Integer, _
il As Integer, dblB() As Double, dblD() As Double) As Integer
    ' 局部变量
    Dim ls As Integer, k As Integer, i As Integer, j As Integer, nIs As Integer
    Dim p As Double, t As Double

    ' 参数校验
    If (il <> (2 * l + 1)) Then
        LEBand = -1
        Exit Function
    End If

    ' 求解
    ls = l
    For k = 1 To n - 1
        p = 0#
        For i = k To ls
            t = Abs(dblB(i, 1))
            If t > p Then
                p = t
                nIs = i
            End If
        Next i

        ' 无解，返回
        If (p + 1# = 1#) Then
            LEBand = 0
            Exit Function
        End If

        For j = 1 To m
            t = dblD(k, j)
            dblD(k, j) = dblD(nIs, j)
            dblD(nIs, j) = t
        Next j

        For j = 1 To il
            t = dblB(k, j)
            dblB(k, j) = dblB(nIs, j)
            dblB(nIs, j) = t
        Next j

        For j = 1 To m
          dblD(k, j) = dblD(k, j) / dblB(k, 1)
        Next j
```

```
        For j = 2 To il
          dblB(k, j) = dblB(k, j) / dblB(k, 1)
        Next j

        For i = k + 1 To ls + 1
            t = dblB(i, 1)
            For j = 1 To m
                dblD(i, j) = dblD(i, j) - t * dblD(k, j)
            Next j

            For j = 2 To il
                dblB(i, j - 1) = dblB(i, j) - t * dblB(k, j)
            Next j

            dblB(i, il) = 0#
        Next i
        If (ls <> n - 1) Then
            ls = ls + 1
        End If
    Next k

    p = dblB(n, 1)

    ' 无解，返回
    If (Abs(p) + 1# = 1#) Then
        LEBand = 0
        Exit Function
    End If

    For j = 1 To m
        dblD(n, j) = dblD(n, j) / p
    Next j

    ls = 1
    For i = n - 1 To 1 Step -1
        For k = 1 To m
            For j = 2 To ls + 1
                dblD(i, k) = dblD(i, k) - dblB(i, j) * dblD(i + j - 1, k)
            Next j
        Next k
        If (ls <> (il - 1)) Then
            ls = ls + 1
        End If
    Next i

    ' 求解成功
    LEBand = 1

End Function
```

3. 示例

调用上述 LEBand 函数求解如下 8 阶五对角线方程组 $\boldsymbol{A}X=\boldsymbol{D}$，其中，

$$\boldsymbol{A}=\begin{bmatrix} 3 & -4 & 1 & & & & & \\ -2 & -5 & 6 & 1 & & & & \\ 1 & 3 & -1 & 2 & -3 & & & \\ & 2 & 5 & -5 & 6 & -1 & & \\ & & -3 & 1 & -1 & 2 & -5 & \\ & & & 6 & 1 & -3 & 2 & -9 \\ & & & & -4 & 1 & -1 & 2 \\ & & & & & 5 & 1 & -7 \end{bmatrix}，\quad \boldsymbol{D}=\begin{bmatrix} 13 & 29 & -13 \\ -6 & 17 & -21 \\ -31 & -6 & 4 \\ 64 & 3 & 16 \\ -20 & 1 & -5 \\ -22 & -41 & 56 \\ -29 & 10 & -21 \\ 7 & -24 & 20 \end{bmatrix}。$$

在本例中，$n=8$，$m=3$，$l=2$，$il=5$，与带型矩阵 $\boldsymbol{A}$ 所对应的二维数组 $\boldsymbol{B}$ 为

$$\boldsymbol{B}=\begin{bmatrix} 3 & -4 & 1 & 0 & 0 \\ -2 & -5 & 6 & 1 & 0 \\ 1 & 3 & -1 & 2 & -3 \\ 2 & 5 & -5 & 6 & -1 \\ -3 & 1 & -1 & 2 & -5 \\ 6 & 1 & -3 & 2 & -9 \\ -4 & 1 & -1 & 2 & 0 \\ 5 & 1 & -7 & 0 & 0 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim dblB(8, 5) As Double, dblD(8, 3) As Double

    '系数矩阵
    dblB(1,1)=3: dblB(1,2)=-4:dblB(1,3)=1: dblB(1,4)=0: dblB(1,5)=0
    dblB(2,1)=-2:dblB(2,2)=-5:dblB(2,3)=6: dblB(2,4)=1: dblB(2,5)=0
    dblB(3,1)=1: dblB(3,2)=3: dblB(3,3)=-1:dblB(3,4)=2: dblB(3,5)=-3
    dblB(4,1)=2: dblB(4,2)=5: dblB(4,3)=-5:dblB(4,4)=6: dblB(4,5)=-1
    dblB(5,1)=-3: dblB(5,2)=1:dblB(5,3)=-1:dblB(5,4)=2:dblB(5,5)=-5
    dblB(6,1)=6:  dblB(6,2)=1:dblB(6,3)=-3:dblB(6,4)=2:dblB(6,5)=-9
    dblB(7,1)=-4: dblB(7,2)=1:dblB(7,3)=-1:dblB(7,4)=2:dblB(7,5)=0
    dblB(8,1)=5:  dblB(8,2)=1:dblB(8,3)=-7:dblB(8,4)=0:dblB(8,5)=0

    '常数向量
    dblD(1, 1) = 13:   dblD(1, 2) = 29:   dblD(1, 3) = -13
    dblD(2, 1) = -6:   dblD(2, 2) = 17:   dblD(2, 3) = -21
    dblD(3, 1) = -31:  dblD(3, 2) = -6:   dblD(3, 3) = 4
    dblD(4, 1) = 64:   dblD(4, 2) = 3:    dblD(4, 3) = 16
    dblD(5, 1) = -20:  dblD(5, 2) = 1:    dblD(5, 3) = -5
    dblD(6, 1) = -22:  dblD(6, 2) = -41:  dblD(6, 3) = 56
    dblD(7, 1) = -29:  dblD(7, 2) = 10:   dblD(7, 3) = -21
```

```
    dblD(8, 1) = 7:   dblD(8, 2) = -24:  dblD(8, 3) = 20

    '求解
    If LEBand(8, 3, 2, 5, dblB, dblD) > 0 Then
       For i = 0 To 7
          Form1.Label2(i) = "x" & (i + 1) & "1 ="
          Form1.Label3(i) = "x" & (i + 1) & "2 ="
          Form1.Label5(i) = "x" & (i + 1) & "3 ="

          Form1.s1(i).Caption = Round(dblD(i + 1, 1), 5)
          Form1.s2(i).Caption = Round(dblD(i + 1, 2), 5)
          Form1.s3(i).Caption = Round(dblD(i + 1, 3), 5)
       Next I

       Form1.Show

    Else
       MsgBox "求解失败！"
    End If

End Sub
```

为了显示整齐的输出结果，本例设计了一个窗体 Form1，并在其中设计了相应的一系列文本框用于显示 3 组解向量，其输出结果如图 3.6 所示。

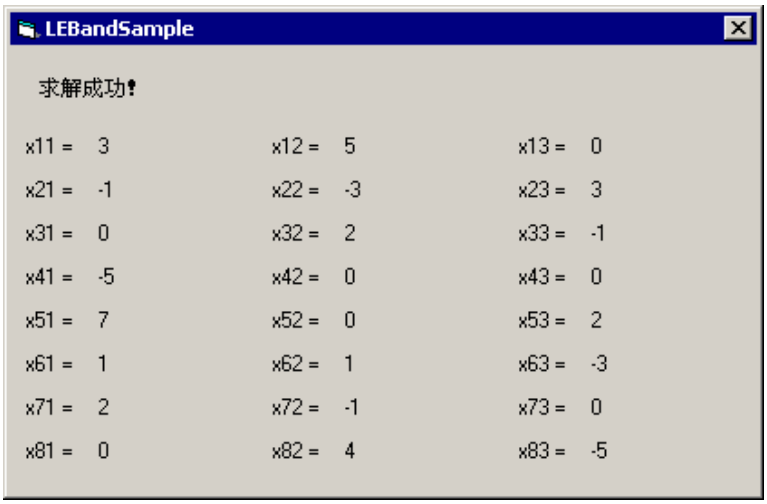

图 3.6 程序输出结果

3.7 求解对称方程组的分解法

1. 算法原理

本节介绍用分解法求解系数矩阵为对称、且右端具有 m 组常数向量的线性代数方程组 $AX=C$，其中 A 为 n 阶对称矩阵，C 为：

$$
\boldsymbol{C}=\begin{bmatrix} c_{11} & c_{12} & \Lambda & c_{1m} \\ c_{21} & c_{22} & \Lambda & c_{2m} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ c_{n1} & c_{n2} & \Lambda & c_{nm} \end{bmatrix}
$$

对称矩阵 A 可以分解为一个下三角矩阵 L、一个对角线矩阵 D 和一个上三角矩阵 L^{T} 的乘积，即 $A=LDL^{\mathrm{T}}$，其中：

$$
\boldsymbol{L}=\begin{bmatrix} 1 & & \\ l_{21} & 1 & 0 \\ \mathrm{M} & \mathrm{M} & \mathrm{M} \\ l_{n1} & l_{n2} & 1 \end{bmatrix},\quad \boldsymbol{D}=\begin{bmatrix} \mathrm{d}_{11} & & \\ & & 0 \\ & \mathrm{d}_{22} & \\ & 0 & \\ & & \mathrm{d}_{\mathrm{nm}} \end{bmatrix}
$$

矩阵 L 和 D 中的各元素由下列计算公式确定：

$$
\begin{aligned}
&d_{11}=a_{11}\\
&d_{ii}=a_{ii}-\sum_{k=1}^{i-1} l_{ik}^{2} d_{kk}\\
&l_{ij}=(a_{ij}-\sum_{k=1}^{i-1} l_{ik} l_{ik} d_{kk})/d_{jj},\quad j<i\\
&l_{ij}=0,\quad j>i
\end{aligned}
$$

对于方程组 AX=C 来说，当 L 和 D 确定后，令 $DL^{\mathrm{T}}X=Y$，则可由回代过程求解方程组 LY=B 而得到 Y，再由方程组 $DL^{\mathrm{T}}X=Y$ 解出 X。计算公式如下：

$$
\begin{aligned}
&y_{1}=b_{1}\\
&y_{i}=b_{i}-\sum_{k=1}^{i-1} l_{ik} y_{k},\quad i>1\\
&x_{n}=y_{n}/d_{nn}\\
&x_{i}=\left(y_{i}-\sum_{k=i+1}^{n} d_{ii} l_{ki} x_{k}\right)\Big/ d_{ii}
\end{aligned}
$$

2. 算法实现

根据上述的运算方法，可以定义用分解法求解对称方程组的 Visual Basic 函数 LEDjn，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEDjn
'  功能：  用分解法求解对称方程组
```

```
' 参数    n   - Integer型变量，线性代数方程组的阶数
'          m  - Integer型变量，为方程组右端的常数向量的个数
'         dblA - Double型n x n二维数组，存放系数矩阵
'         dblC - Double型n x m二维数组，作为传入参数，存放方程组右端的m组常数向量。
'                函数返回时，其中存放着m组解向量。
' 返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LEDjn(n As Integer, m As Integer, dblA() As Double, _
dblC() As Double) As Integer
   ' 局部变量
   Dim i As Integer, j As Integer, k As Integer, k1 As Integer, _
      k2 As Integer, k3 As Integer

   ' 无解，返回
   If (Abs(dblA(1, 1)) + 1# = 1#) Then
      LEDjn = False
      Exit Function
   End If

   For i = 2 To n
     dblA(i, 1) = dblA(i, 1) / dblA(1, 1)
   Next i

   For i = 2 To n - 1
      For j = 2 To i
         dblA(i, i) = dblA(i, i) - _
                      dblA(i, j - 1) * dblA(i, j - 1) * dblA(j - 1, j - 1)
      Next j

      ' 无解，返回
      If (Abs(dblA(i, i)) + 1# = 1#) Then
         LEDjn = False
         Exit Function
      End If

      For k = i + 1 To n
         For j = 2 To i
            dblA(k, i) = dblA(k, i) - _
                 dblA(k, j - 1) * dblA(i, j - 1) * dblA(j - 1, j - 1)
         Next j
         dblA(k, i) = dblA(k, i) / dblA(i, i)
      Next k
   Next i

   For j = 2 To n
      dblA(n, n) = dblA(n, n) - _
                   dblA(n, j - 1) * dblA(n, j - 1) * dblA(j - 1, j - 1)
   Next j

   ' 无解，返回
   If (Abs(dblA(n, n)) + 1# = 1#) Then
```

```
        LEDjn = False
        Exit Function
    End If

    For j = 1 To m
        For i = 2 To n
            For k = 2 To i
                dblC(i, j) = dblC(i, j) - dblA(i, k - 1) * dblC(k - 1, j)
            Next k
        Next i
    Next j

    For i = 2 To n
        For j = i To n
            dblA(i - 1, j) = dblA(i - 1, i - 1) * dblA(j, i - 1)
        Next j
    Next i

    For j = 1 To m
        dblC(n, j) = dblC(n, j) / dblA(n, n)

        For k = 2 To n
            k1 = n - k + 2
            For k2 = k1 To n
                k3 = n - k + 1
                dblC(k3, j) = dblC(k3, j) - dblA(k3, k2) * dblC(k2, j)
            Next k2
            dblC(k3, j) = dblC(k3, j) / dblA(k3, k3)
        Next k
    Next j

    ' 求解成功
    LEDjn = True

End Function
```

3. 示例

调用上述 LEDjn **函数求解如下** 5 **阶对称方程组** $AX = C$**，其中**

$$\boldsymbol{A}=\begin{bmatrix}5 & 7 & 6 & 5 & 1\\ 7 & 10 & 8 & 7 & 2\\ 6 & 8 & 10 & 9 & 3\\ 5 & 7 & 9 & 10 & 4\\ 1 & 2 & 3 & 4 & 5\end{bmatrix},\quad \boldsymbol{C}=\begin{bmatrix}24 & 96\\ 34 & 136\\ 36 & 144\\ 35 & 140\\ 15 & 60\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim dblA(5, 5) As Double
```

```
    Dim dblB(5, 2) As Double

    '系数矩阵
    dblA(1,1)=5:dblA(1,2)=7: dblA(1,3)=6: dblA(1,4)=5: dblA(1,5)=1
    dblA(2,1)=7:dblA(2,2)=10:dblA(2,3)=8: dblA(2,4)=7: dblA(2,5)=2
    dblA(3,1)=6:dblA(3,2)=8: dblA(3,3)=10:dblA(3,4)=9: dblA(3,5)=3
    dblA(4,1)=5:dblA(4,2)=7: dblA(4,3)=9: dblA(4,4)=10:dblA(4,5)=4
    dblA(5,1)=1:dblA(5,2)=2: dblA(5,3)=3: dblA(5,4)=4: dblA(5,5)=5
    '常数矩阵
    dblB(1,1)=24:dblB(2,1)=34: dblB(3,1)=36: dblB(4,1)=35: dblB(5,1)=15
    dblB(1,2)=96:dblB(2,2)=136:dblB(3,2)=144:dblB(4,2)=140:dblB(5,2)=60

    '求解
    If LEDjn(5, 2, dblA, dblB) = True Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x11 = " & Round(dblB(1, 1), 5) & "      x12 = " & _
              Round(dblB(1, 2), 5) & Chr$(13) & _
              "x21 = " & Round(dblB(2, 1), 5) & "      x22 = " & _
              Round(dblB(2, 2), 5) & Chr$(13) & _
              "x31 = " & Round(dblB(3, 1), 5) & "
x32 = " & _
              Round(dblB(3, 2), 5) & Chr$(13) & _
              "x41 = " & Round(dblB(4, 1), 5) & "
x42 = " & _
              Round(dblB(4, 2), 5) & Chr$(13) & _
              "x51 = " & Round(dblB(5, 1), 5) & "
x52 = " & _
              Round(dblB(5, 2), 5)
    Else
       MsgBox "求解失败！"
End If

End Sub
```

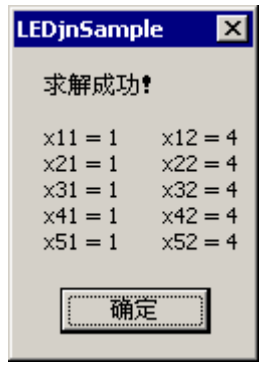

图 3.7　程序输出结果

输出结果如图 3.7 **所示。**

3.8　求解对称正定方程组的平方根法

1. 算法原理

乔里斯基(Cholesky)**分解法或称为平均根法是求解系数矩阵为对称正定、且右端具有** m **组常数向量的** n **阶线性代数方程组的常用算法。**

对线性方程组 $\boldsymbol{AX}=\boldsymbol{D}$**，当系数矩阵** $\boldsymbol{A}$ **为对称正定时，可以惟一地分解为** $\boldsymbol{A}=\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}$**，其中** $\boldsymbol{U}$ **为上三角矩阵。即：**

$$\begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ a_{n1} & a_{n2} & \Lambda & a_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & & & \\ u_{12} & u_{22} & & 0 \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \\ u_{n1} & u_{n2} & & u_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \Lambda & u_{1n} \\ & u_{22} & \Lambda & u_{2n} \\ & 0 & \mathrm{O} & \mathrm{M} \\ & & & u_{nn} \end{bmatrix}$$

且 $u_{ij}=u_{ji}(i,\ j=1, 2, ..., n)$。

U 矩阵中的各元素由以下公式计算：

$$u_{11}=\sqrt{a_{11}}$$

$$u_{ii}=(a_{ii}-\sum_{k=1}^{i-1}u_{ki}^2), i=2, ..., n$$

$$u_{ij}=\left(a_{ij}-\sum_{k=1}^{i-1}u_{ki}u_{kj}\right)\Big/u_{ii}\ ,\quad j>i$$

方程组 $AX=B$ 的解可由如下公式求得：

$$y_i=(b_i-\sum_{k=1}^{i-1}u_{ki}y_k)\Big/u_{ii}$$

$$x_i=(y_i-\sum_{k=i+1}^{n-1}u_{ik}x_k)\Big/u_{ii}$$

2. 算法实现

根据上述的运算方法，可以定义乔里斯基分解法求解正定方程组的 Visual Basic 函数 LECholesky，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LECholesky
'  功能：  用乔里斯基分解法求解正定方程组
'  参数    n   - Integer型变量，线性代数方程组的阶数
'          m  - Integer型变量，为方程组右端的常数向量的个数
'         dblA - Double型n x n二维数组，存放系数矩阵(应为对称正定矩阵)；
'                返回时，其上三角部分存放分解后的U矩阵
'         dblD - Double型n x m二维数组，作为传入参数，存放方程组右端的m组常数向量。
'                函数返回时，其中存放着m组解向量。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LECholesky(n As Integer, m As Integer, dblA() As Double, _
dblD() As Double) As Integer
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer

    ' 矩阵非正定，求解失败
    If ((dblA(1, 1) + 1# = 1#) Or (dblA(1, 1) < 0#)) Then
        LECholesky = False
        Exit Function
    End If

    dblA(1, 1) = Sqr(dblA(1, 1))
    For j = 2 To n
        dblA(1, j) = dblA(1, j) / dblA(1, 1)
```

```
    Next j

    For i = 2 To n
       For j = 2 To i
          dblA(i, i) = dblA(i, i) - dblA(j - 1, i) * dblA(j - 1, i)
       Next j

       ' 求解失败
       If ((dblA(i, i) + 1# = 1#) Or (dblA(i, i) < 0#)) Then
          LECholesky = False
          Exit Function
       End If

       dblA(i, i) = Sqr(dblA(i, i))
       If (i <> n) Then
          For j = i + 1 To n
             For k = 2 To i
               dblA(i, j) = dblA(i, j) - dblA(k - 1, i) * dblA(k - 1, j)
             Next k
             dblA(i, j) = dblA(i, j) / dblA(i, i)
          Next j
       End If
    Next i

    For j = 1 To m
       dblD(1, j) = dblD(1, j) / dblA(1, 1)

       For i = 2 To n
          For k = 2 To i
             dblD(i, j) = dblD(i, j) - dblA(k - 1, i) * dblD(k - 1, j)
          Next k

          dblD(i, j) = dblD(i, j) / dblA(i, i)
       Next i
    Next j

    For j = 1 To m
       dblD(n, j) = dblD(n, j) / dblA(n, n)

       For k = n To 2 Step -1
          For i = k To n
             dblD(k - 1, j) = dblD(k - 1, j) - dblA(k - 1, i) * dblD(i, j)
          Next i
          dblD(k - 1, j) = dblD(k - 1, j) / dblA(k - 1, k - 1)
       Next k
    Next j

    ' 求解成功
    LECholesky = True

End Function
```

3. 示例

调用上述 LECholesky 函数求解如下 4 阶正定方程组：

$$\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 23 & 92 \\ 32 & 128 \\ 33 & 132 \\ 31 & 124 \end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim dblA(4, 4) As Double
    Dim dblB(4, 2) As Double

    '系数矩阵
    dblA(1, 1) = 5:   dblA(1, 2) = 7:    dblA(1, 3) = 6:    dblA(1, 4) = 5
    dblA(2, 1) = 7:   dblA(2, 2) = 10:   dblA(2, 3) = 8:    dblA(2, 4) = 7
    dblA(3, 1) = 6:   dblA(3, 2) = 8:    dblA(3, 3) = 10:   dblA(3, 4) = 9
    dblA(4, 1) = 5:   dblA(4, 2) = 7:    dblA(4, 3) = 9:    dblA(4, 4) = 10
    '常数矩阵
    dblB(1, 1) = 23:  dblB(2, 1) = 32:   dblB(3, 1) = 33:   dblB(4, 1) = 31
    dblB(1, 2) = 92:  dblB(2, 2) = 128:  dblB(3, 2) = 132:  dblB(4, 2) = 124

    '求解
    If LECholesky(4, 2, dblA, dblB) = True Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x11 = " & Round(dblB(1, 1), 5) & "      x12 = " & _
              Round(dblB(1, 2), 5) & Chr$(13) & _
              "x21 = " & Round(dblB(2, 1), 5) & "
x22 = " & _
              Round(dblB(2, 2), 5) & Chr$(13) & _
              "x31 = " & Round(dblB(3, 1), 5) & "
x32 = " & _
              Round(dblB(3, 2), 5) & Chr$(13) & _
              "x41 = " & Round(dblB(4, 1), 5) & "
x42 = " & _
              Round(dblB(4, 2), 5)
    Else
       MsgBox "求解失败！"
    End If

End Sub
```

图 3.8　程序输出结果

本例的输出结果如图 3.8 所示。

3.9 求解大型稀疏方程组的全选主元高斯-约当消去法

1. 算法原理

对线性方程组 $AX=B$，如果其系数矩阵为稀疏矩阵，即矩阵中大部分元素为 0，则称该方程组为稀疏方程组。用 3.2 节中介绍的全选主元高斯-约当(Gauss-Jordan)消去法求解稀疏方程组时，在消元过程中避免了对零元素的运算，从而可以大大减少运算次数。因此，全选主元高斯-约当消去法非常适合用于求解大型稀疏方程组。

2. 算法实现

用全选主元高斯-约当消去法求解稀疏方程组的 Visual Basic 函数 LEGgje 的代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEGgje
'  功能：  用全选主元高斯-约当消去法求解稀疏方程组
'  参数    n    - Integer型变量，线性代数方程组的阶数
'         dblA  - Double型n x n二维数组，存放系数矩阵(应为稀疏矩阵)
'         dblB  - Double一维数组，长度为n，存放方程组右端的常数向量；
'                 返回时存放方程组的解
'  返回值： Boolean型。False，失败无解；True，成功
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LEGgje(n As Integer, dblA() As Double, dblB() As Double) _
As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim nIs As Integer
    ReDim nJs(n) As Integer
    Dim d As Double, q As Double

    ' 开始求解
    For k = 1 To n
        q = 0#

        ' 归一
        For i = k To n
            For j = k To n
                If Abs(dblA(i, j)) > q Then
                    q = Abs(dblA(i, j))
                    nJs(k) = j
                    nIs = i
                End If
            Next j
        Next i

        ' 无解，返回
        If q + 1# = 1# Then
```

```
        LEGgje = False
        Exit Function
    End If

    ' 消元
    ' A->
    For j = k To n
        d = dblA(k, j)
        dblA(k, j) = dblA(nIs, j)
        dblA(nIs, j) = d
    Next j
    ' B->
    d = dblB(k)
    dblB(k) = dblB(nIs)
    dblB(nIs) = d
    'A->
    For i = 1 To n
        d = dblA(i, k)
        dblA(i, k) = dblA(i, nJs(k))
        dblA(i, nJs(k)) = d
    Next i

    For j = k + 1 To n
        dblA(k, j) = dblA(k, j) / dblA(k, k)
    Next j

    dblB(k) = dblB(k) / dblA(k, k)

    ' 回代
    For j = k + 1 To n
      For i = 1 To n
        If i <> k Then
          dblA(i, j) = dblA(i, j) - dblA(i, k) * dblA(k, j)
        End If
      Next i
    Next j

    For i = 1 To n
    If i <> k Then
      dblB(i) = dblB(i) - dblA(i, k) * dblB(k)
    End If
    Next i
Next k

' 调整解的次序
For k = n To 1 Step -1
    d = dblB(k)
    dblB(k) = dblB(nJs(k))
    dblB(nJs(k)) = d
Next k
```

```
    ' 求解成功
    LEGgje = True

End Function
```

3. 示例

调用上述 LEGgje 函数求解 8 阶稀疏方程组 $AX = B$，其中，

$$\boldsymbol{A}=\begin{bmatrix}0&0&-1&0&0&0&2&0\\0&6&0&0&0&-6&0&0\\0&0&0&2&0&0&0&-4\\3&0&0&0&-2&0&1&0\\0&0&6&0&0&0&5&0\\1&0&0&0&-3&0&0&2\\0&4&0&-1&0&0&0&0\\0&0&1&0&-1&0&0&-2\end{bmatrix}$$

$$\boldsymbol{B}=(4,6,-8,-2,27,-9,2,-4)^{\mathrm{T}}$$

程序代码如下：

```
Sub Main()
    Dim a(8, 8) As Double
    Dim b(8) As Double

    '系数矩阵
a(1,1)=0:a(1,2)=0:a(1,3)=-1:a(1,4)=0:a(1,5)=0:a(1,6)=0:a(1,7)=2:a(1,8)=0
a(2,1)=0:a(2,2)=6:a(2,3)=0:a(2,4)=0:a(2,5)=0:a(2,6)=-6:a(2,7)=0:a(2,8)=0
a(3,1)=0:a(3,2)=0:a(3,3)=0:a(3,4)=2:a(3,5)=0:a(3,6)=0:a(3,7)=0:a(3,8)=-4
a(4,1)=3:a(4,2)=0:a(4,3)=0:a(4,4)=0:a(4,5)=-2:a(4,6)=0:a(4,7)=1:a(4,8)=0
a(5,1)=0:a(5,2)=0:a(5,3)=6:a(5,4)=0:a(5,5)=0:a(5,6)=0:a(5,7)=5:a(5,8)=0
a(6,1)=1:a(6,2)=0:a(6,3)=0:a(6,4)=0:a(6,5)=-3:a(6,6)=0:a(6,7)=0:a(6,8)=2
a(7,1)=0:a(7,2)=4:a(7,3)=0:a(7,4)=-1:a(7,5)=0:a(7,6)=0:a(7,7)=0:a(7,8)=0
a(8,1)=0:a(8,2)=0:a(8,3)=1:a(8,4)=0:a(8,5)=-1:a(8,6)=0:a(8,7)=0:a(8,8)=-2

    '常数向量
    b(1) = 4
    b(2) = 6
    b(3) = -8
    b(4) = -2
    b(5) = 27
    b(6) = -9
    b(7) = 2
    b(8) = -4

    '求解
    If LEGgje(8, a, b) = True Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x1 = " & Round(b(1), 5) & Chr$(13) & _
```

```
                "x2 = " & Round(b(2), 5) & Chr$(13) & _
                "x3 = " & Round(b(3), 5) & Chr$(13) & _
                "x4 = " & Round(b(4), 5) & Chr$(13) & _
                "x5 = " & Round(b(5), 5) & Chr$(13) & _
                "x6 = " & Round(b(6), 5) & Chr$(13) & _
                "x7 = " & Round(b(7), 5) & Chr$(13) & _
                "x8 = " & Round(b(8), 5)
    Else
        MsgBox "求解失败！"
    End If
End Sub
```

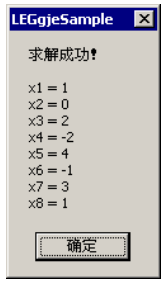

图 3.9　程序输出结果

本例的输出结果如图 3.9 **所示。**

3.10　求解托伯利兹方程组的列文逊方法

1. 算法原理

本节介绍用列文逊(Levinson)**递推算法求解对称托伯利兹**(Toeplite)**型方程组的算法。**
n **阶对称托伯利兹矩阵的形式如下：**

$$\boldsymbol{T}^{(n)}=\begin{bmatrix} t_1 & t_2 & t_3 & \Lambda & t_n \\ t_2 & t_1 & t_2 & \Lambda & t_{n-1} \\ t_3 & t_2 & t_1 & \Lambda & t_{n-2} \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & & \mathrm{M} \\ t_n & t_{n-1} & t_{n-2} & & t_1 \end{bmatrix}$$

这种矩阵简称为 *n* **阶对称** T **型矩阵。**
设线性代数方程组 $\boldsymbol{AX}=\boldsymbol{B}$ **的系数矩阵为** *n* **阶对称** T **型矩阵，即** $\boldsymbol{A}=\boldsymbol{T}^{(n)}$**。**
假设已知

$$\boldsymbol{T}^{(k)}\begin{bmatrix} y_1^{(k)} \\ \mathrm{M} \\ y_{k-1}^{(k)} \\ y_k^{(k)} \end{bmatrix}=\begin{bmatrix} 0 \\ \mathrm{M} \\ 0 \\ \alpha_k \end{bmatrix}$$

令

$$\beta_k=y_k^{(k)}t_{k+1}+y_{k-1}^{(k)}t_k+\Lambda+y_1^{(k)}t_2=\sum_{=1}^{k}t_{+1}y^{(k)}\qquad(1)$$

由于 $\boldsymbol{T}^{(k)}$**和** $\boldsymbol{T}^{(k+1)}$**都是对称** T **型矩阵，因此，**

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} 0 \\ y_1^{(k)} \\ \mathrm{M} \\ y_{k-1}^{(k)} \\ y_k^{(k)} \end{bmatrix}=\begin{bmatrix} \beta_k \\ 0 \\ \mathrm{M} \\ 0 \\ \alpha_k \end{bmatrix}\qquad(2)$$

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} y_k^{(k)} \\ y_{k-1}^{(k)} \\ \mathrm{M} \\ y_1^{(k)} \\ 0 \end{bmatrix}=\begin{bmatrix} \alpha_k \\ 0 \\ \mathrm{M} \\ 0 \\ \beta_k \end{bmatrix} \tag{3}$$

若将方程组(2)减去方程组(3)的 β_k / α_k 倍，并且令

$$c_k = -\beta_k / \alpha_k \tag{4}$$

则可得到如下方程组：

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} c_k y_k^{(k)} \\ y_1^{(k)} + c_k y_{k-1}^{(k)} \\ \mathrm{M} \\ y_{k-1}^{(k)} + c_k y_1^{(k)} \\ y_k^{(k)} \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \mathrm{M} \\ 0 \\ \alpha_k + c_k \beta_k \end{bmatrix} \tag{5}$$

若再令：

$$\begin{cases} y_1^{(k+1)} = c_k y_k^{(k)} \\ y_i^{(k+1)} = y_{i-1}^{(k)} + c_{k-1} y_{k-i-1}^{(k)}, \qquad i = 2, 3, \Lambda\ , k \\ y_{k+1}^{(k+1)} = y_k^{(k)} \end{cases} \tag{6}$$

及

$$\alpha_{k+1} = \alpha_k + c_k \beta_k \tag{7}$$

则方程组(5)变为：

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} y_1^{(k+1)} \\ y_2^{(k+1)} \\ \mathrm{M} \\ y_k^{(k+1)} \\ y_{k+1}^{(k+1)} \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \mathrm{M} \\ 0 \\ \alpha_{k+1} \end{bmatrix} \tag{8}$$

显然，式(6)与式(7)是递推公式。若取 $\alpha_1 = t_1$，则 $y_1^{(1)} = 1$。

下面再考虑方程组

$$\boldsymbol{T}^{(k)} X^{(k)} = \boldsymbol{B}^{(k)}, k = 2, 3, \Lambda\ , n \tag{9}$$

其中，

$$\boldsymbol{X}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \Lambda\ , x_k^{(k)})^{\mathrm{T}}$$

$$\boldsymbol{B}^{(k)} = (b_1, b_2, \Lambda\ , b_k)^{\mathrm{T}}$$

现假设对于某个 k，方程组(9)已经解出，则：

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} X^{(k+1)} \\ 0 \end{bmatrix}=\begin{bmatrix} B^{(k)} \\ q_{k+1} \end{bmatrix} \tag{10}$$

$$q_{k+1}=x_1^{(k)}t_{k+1}+\Lambda\ +x_k^{(k)}t_2=\sum_{=1}^{k}x^{(k)}t_{k-\ +2} \qquad {}_{j} \qquad {}_{j} \tag{11}$$

又因为：

$$T^{(k+1)}X^{(k+1)}=B^{(k+1)} \tag{12}$$

于是，方程组(12)减去(10)得

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} x_1^{(k+1)}-x_1^{(k)} \\ x_2^{(k+1)}-x_2^{(k)} \\ - \quad \mathrm{M} \\ x_k^{(k+1)}-x_k^{(k)} \\ x_{k+1}^{(k+1)} \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \mathrm{M} \\ 0 \\ b_{k+1}-q_{k+1} \end{bmatrix} \tag{13}$$

在上述过程中，若 k=1 时有：

$$x_1^{(1)}=b_1/t_1$$

如果用

$$w_{k+1}=(b_{k+1}-q_{k+1})/\alpha_{k+1} \tag{14}$$

乘方程组(8)则可得：

$$\boldsymbol{T}^{(k+1)}\begin{bmatrix} w_{k+1}y_1^{(k+1)} \\ w_{k+1}y_2^{(k+1)} \\ \mathrm{M} \\ w_{k+1}y_k^{(k+1)} \\ w_{k+1}y_{k+1}^{(k+1)} \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \mathrm{M} \\ 0 \\ b_{k+1}-q_{k+1} \end{bmatrix} \tag{15}$$

比较方程组(13)与(15)，可以得到由 $X^{(k)}$ 计算 $X^{(k+1)}$ 的计算公式：

$$\begin{cases} x_i^{(k+1)}=x_i^{(k)}+w_{k+1}y_i^{(k+1)} & i=1,2,3,\Lambda\ k \\ x_{k+1}^{(k+1)}=w_{k+1}y_{k+1}^{(k+1)} \end{cases} \tag{16}$$

综上所述，由式(11)、(3)、(4)、(6)、(7)、(14)和(16)可以得到求解对称 T 型方程组的递推算法：

(1) 取初值：$\alpha_1=t_1, y_1^{(1)}=1, x_1^{(1)}=b_1/t_1$ ；

(2) 对于 k 从 1 到 n-1 依次计算如下公式：

$$q_k = \sum_{j=1}^{k} x_j^{(k)} t_{k-j+2}$$

$$\beta_k = \sum_{j=1}^{k} y_j^{(k)} t_{j+1}$$

$$c_k = -\beta_k / \alpha_k$$

$$\begin{cases} y_1^{(k+1)} = c_k y_k^{(k)} \\ y_i^{(k+1)} = y_{i-1}^{(k)} + c_k y_{k-i}^{(k)} \quad i = 2, 3, 4, \Lambda\ , k \\ y_{k+1}^{(k+1)} = y_k^{(k)} \end{cases}$$

$$\alpha_{k+1} = \alpha_k + c_k \beta_k$$

$$w_{k+1} = (b_{k+1} - q_{k+1}) / \alpha_{k+1}$$

$$\begin{cases} x_i^{(k+1)} = x_i^{(k)} + w_{k+1} y_i^{(k+1)} \quad i = 1, 2, 3, \Lambda\ , k \\ x_{k+1}^{(k+1)} = w_{k+1} y_{k+1}^{(k+1)} \end{cases}$$

2. **算法实现**

根据上述计算方法，可以定义用列文逊递推算法求解对称托伯利兹方程组的 Visual Basic **函数** LETlvs**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LETlvs
'  功能：  用列文逊递推算法求解对称托伯利兹方程组
'  参数    n    - Integer型变量，线性代数方程组的阶数
'         dblT  - Double型一维数组，长度为n ，存放对称T型矩阵中的元素
'         dblB  - Double一维数组，长度为n，存放方程组右端的常数向量
'         dblX  - Double一维数组，长度为n，返回存放方程组的解
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function LETlvs(n As Integer, dblT() As Double, dblB() As Double, _
dblX() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer, k As Integer
    Dim a As Double, beta As Double, q As Double, c As Double, h As Double
    ReDim y(n) As Double, s(n) As Double

    a = dblT(1)
    If (Abs(a) + 1# = 1#) Then
        LETlvs = False
        Exit Function
    End If

    y(1) = 1#
    dblX(1) = dblB(1) / a

    For k = 1 To n - 1
        beta = 0#
        q = 0#
```

```
        For j = 1 To k
            beta = beta + y(j) * dblT(j + 1)
            q = q + dblX(j) * dblT(k - j + 2)
        Next j

        If (Abs(a) + 1# = 1#) Then
            LETlvs = False
            Exit Function
        End If

        c = -beta / a
        s(1) = c * y(k)
        y(k + 1) = y(k)
        If (k <> 1) Then
            For i = 2 To k
                s(i) = y(i - 1) + c * y(k - i + 1)
            Next i
        End If

        a = a + c * beta
        If (Abs(a) + 1# = 1#) Then
            LETlvs = False
            Exit Function
        End If

        h = (dblB(k + 1) - q) / a
        For i = 1 To k
            dblX(i) = dblX(i) + h * s(i)
            y(i) = s(i)
        Next i

        dblX(k + 1) = h * y(k + 1)
    Next k

    LETlvs = True

End Function
```

3. 示例

调用上述 LETlvs **函数求解** 6 **阶** T **型方程组** $\boldsymbol{A}X = \boldsymbol{B}$，**其中，**

$$\boldsymbol{A} = \begin{bmatrix} 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 6 & 5 & 4 & 3 & 2 \\ 4 & 5 & 6 & 5 & 4 & 3 \\ 3 & 4 & 5 & 6 & 5 & 4 \\ 2 & 3 & 4 & 5 & 6 & 5 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} = \boldsymbol{T}^{(6)}$$

$$\boldsymbol{T} = (6, 5, 4, 3, 2, 1)$$

$$\boldsymbol{B} = (11, 9, 9, 9, 13, 17)^T$$

程序代码如下：

```
Sub Main()
    Dim dblT(6) As Double
    Dim dblB(6) As Double
    Dim x(6) As Double

    '系数矩阵
    dblT(1) = 6
    dblT(2) = 5
    dblT(3) = 4
    dblT(4) = 3
    dblT(5) = 2
    dblT(6) = 1

    '常数向量
    dblB(1) = 11
    dblB(2) = 9
    dblB(3) = 9
    dblB(4) = 9
    dblB(5) = 13
dblB(6) = 17

    '求解
    If LETlvs(6, dblT, dblB, x) = True Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "x1 = " & Round(x(1), 5) & Chr$(13) & _
               "x2 = " & Round(x(2), 5) & Chr$(13) & _
               "x3 = " & Round(x(3), 5) & Chr$(13) & _
               "x4 = " & Round(x(4), 5) & Chr$(13) & _
               "x5 = " & Round(x(5), 5) & Chr$(13) & _
               "x6 = " & Round(x(6), 5)
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

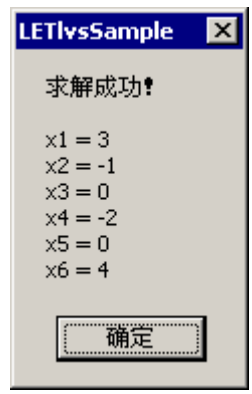

图 3.10　程序输出结果

本例的输出结果如图 3.10 **所示。**

3.11　高斯-赛德尔迭代法

1. 算法原理

对线性代数方程组 $AX=B$**，如果其系数矩阵** A **满足如下的条件：**

$$\sum_{\substack{j=0\\ j\neq i}}^{n-1}\left|a_{ij}\right|<\left|a_{ii}\right|,\quad i=0,1,\Lambda\ ,n-1$$

即系数矩阵主对角线占绝对优势，则可以采用高斯-赛德尔(Gauss-Seidel)**迭代法来求**

解。

高斯-赛德尔迭代格式为：

$$x_i^{(k+1)} = (b_i - \sum_{=0}^{i-1} a_i\, x^{(k+1)} - \sum_{\neq i+1}^{n-1} a_{ij} x_j^{(k)}) \ \big/ a_{ii},\ i_j =_j 0, 1, \Lambda\ , n-1$$

其中，初值取 $x_i^{(0)} = 0\ (i = 0, 1, \Lambda\ , n-1)$。

结束迭代的条件为：

$$\max_{0 \le i \le n-1} \frac{\left| x_i^{(k+1)} - x_i^{(k)} \right|}{1 + \left| x_i^{(k+1)} \right|} < \varepsilon$$

其中，ε 为给定的精度要求。

2. 算法实现

用高斯-赛德尔迭代法求解系数矩阵主对角线占绝对优势的线性方程组的 Visual Basic 函数 LESeidel 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LESeidel
'  功能：  用高斯-赛德尔迭代求解系数矩阵主对角线占绝对优势线性方程组
'  参数：   n    - Integer型变量，线性代数方程组的阶数
'         dblA  - Double型n x n二维数组，存放系数矩阵
'         dblB  - Double一维数组，长度为n，存放方程组右端的常数向量
'         dblX  - Double型一维数组，长度为n。返回方程组的解。
'         eps   - Double型变量。给定的精度要求。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function LESeidel(n As Integer, dblA() As Double, dblB() As Double, _
dblX() As Double, eps As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer
    Dim p As Double, q As Double, s As Double, t As Double

    ' 校验系数矩阵是否主对角线占绝对优势
    For i = 1 To n
        p = 0#
        dblX(i) = 0#
        For j = 1 To n
            If (i <> j) Then
                p = p + Abs(dblA(i, j))
            End If
        Next j
        If (p >= Abs(dblA(i, i))) Then
            LESeidel = False
            Exit Function
        End If
    Next i
```

```
    ' 迭代求解
    p = eps + 1#
    While (p >= eps)
       p = 0#
       For i = 1 To n
          t = dblX(i)
          s = 0#
          For j = 1 To n
             If (j <> i) Then
                s = s + dblA(i, j) * dblX(j)
             End If
          Next j

          dblX(i) = (dblB(i) - s) / dblA(i, i)
          q = Abs(dblX(i) - t) / (1# + Abs(dblX(i)))
          If (q > p) Then
             p = q
          End If
       Next i
    Wend

    ' 求解成功
    LESeidel = True

End Function
```

3. 示例

调用上述函数 LESeidel **求解下列** 4 **阶方程组：**

$$\begin{cases} 7x_0 + 2x_1 + x_2 - 2x_3 = 4 \\ 9x_0 + 15x_1 + 3x_2 - 2x_3 = 7 \\ -2x_0 - 2x_1 + 11x_2 + 5x_3 = -1 \\ x_0 + 3x_1 + 2x_2 + 13x_3 = 0 \end{cases}$$

取 ε =0.000001。

程序代码如下：

```
Sub Main()
    Dim dblA(4, 4) As Double
    Dim dblB(4) As Double
    Dim x(4) As Double

    '系数矩阵
    '系数矩阵
    dblA(1, 1) = 7:   dblA(1, 2) = 2:    dblA(1, 3) = 1:    dblA(1, 4) = -2
    dblA(2, 1) = 9:   dblA(2, 2) = 15:   dblA(2, 3) = 3:    dblA(2, 4) = -2
    dblA(3, 1) = -2:  dblA(3, 2) = -2:   dblA(3, 3) = 11:   dblA(3, 4) = 5
    dblA(4, 1) = 1:   dblA(4, 2) = 3:    dblA(4, 3) = 2:    dblA(4, 4) = 13
```

```
    '常数向量
    dblB(1) = 4
    dblB(2) = 7
    dblB(3) = -1
    dblB(4) = 0

    '求解
    If LESeidel(4, dblA, dblB, x, 0.000001) = True Then
       MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
              "x1 = " & Round(x(1), 7) & Chr$(13)
& _
              "x2 = " & Round(x(2), 7) & Chr$(13)
& _
              "x3 = " & Round(x(3), 7) & Chr$(13)
& _
              "x4 = " & Round(x(4), 7)
    Else
       MsgBox "求解失败！"
    End If

End Sub
```

图 3.11　程序输出结果

本例的输出结果如图 3.11 所示。

3.12　求解对称正定方程组的共轭梯度法

1. 算法原理

共轭梯度法是求解 n 阶对称正定方程组的重要方法。共轭梯度法的递推计算公式如下：

(1) 取初值 $X_0 = (0, 0, \Lambda\ , 0)^{\mathrm{T}}$，$R_0 = P_0 = B$。

(2) 对于 $i = 0, 1, \Lambda\ , n-1$ 作如下计算：

$$\alpha_i = \frac{(P_i, B)}{(P_i, AP_i)}$$

$$X_{i+1} = X_i + \alpha_i P_i$$

$$R_{i+1} = B - AX_{i+1}$$

$$\beta_i = \frac{(P_{i+1}, AP_i)}{(P_i, AP_i)}$$

$$P_{i+1} = R_{i+1} - \beta_i P_i$$

上述过程一直作到 $\|R_i\| < \varepsilon$ 或 $i = n-1$ 为止。

特别要指出，本方法只适用于对称正定方程组。

2. 算法实现

基于上述方法，可以定义用共轭梯度法求解 n 阶对称正定方程组的 Visual Basic 函数 LEGrad，其中调用了 2.1 节介绍的矩阵乘法函数 MMul。LEGrad 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEGrad
'  功能：  用共轭梯度法是求解n阶对称正定方程组
'  参数：  n    - Integer型变量，线性代数方程组的阶数
'         dblA  - Double型n x n二维数组，存放对称正定系数矩阵
'         dblB  - Double一维数组，长度为n，存放方程组右端的常数向量
'         dblX  - Double型一维数组，长度为n。返回方程组的解。
'         eps   - Double型变量。给定的精度要求。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub LEGrad(n As Integer, dblA() As Double, dblB() As Double, dblX() As Double,
eps As Double)
    ' 局部变量
    Dim i As Integer, k As Integer
    ReDim p(n, 1) As Double, r(n) As Double, s(n, 1) As Double, _
         q(n, 1) As Double, x(n, 1) As Double
    Dim alpha As Double, beta As Double, d As Double, e As Double

    ' 初始化
    For i = 1 To n
       x(i, 1) = 0#
       p(i, 1) = dblB(i)
       r(i) = dblB(i)
    Next i

    ' 循环求解
    i = 1
    While (i <= n)

       ' 矩阵乘法
       Call MMul(n, n, 1, dblA, p, s)

       d = 0#
       e = 0#

       For k = 1 To n
          d = d + p(k, 1) * dblB(k)
          e = e + p(k, 1) * s(k, 1)
       Next k

       alpha = d / e
       For k = 1 To n
         x(k, 1) = x(k, 1) + alpha * p(k, 1)
       Next k

       ' 矩阵乘法
       Call MMul(n, n, 1, dblA, x, q)

       d = 0#
       For k = 1 To n
          r(k) = dblB(k) - q(k, 1)
```

```
            d = d + r(k) * s(k, 1)
        Next k

        beta = d / e
        d = 0#
        For k = 1 To n
            d = d + r(k) * r(k)
        Next k

        d = Sqr(d)

        ' 求解结束，返回
        If (d < eps) Then
            For k = 1 To n
               dblX(k) = x(k, 1)
            Next k
            Exit Sub
        End If

        For k = 1 To n
           p(k, 1) = r(k) - beta * p(k, 1)
        Next k

        i = i + 1
    Wend

    ' 求解结束，返回
    For k = 1 To n
       dblX(k) = x(k, 1)
    Next k

End Function
```

3. 示例

调用上述函数 LEGrad **求解** 4 **阶对称正定方程组** $\boldsymbol{AX}=\boldsymbol{B}$**。其中，**

$$\boldsymbol{A}=\begin{bmatrix}5 & 7 & 6 & 5\\ 7 & 10 & 8 & 7\\ 6 & 8 & 10 & 9\\ 5 & 7 & 9 & 10\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}23\\ 32\\ 33\\ 31\end{bmatrix}$$

取 ε =0.000001。

程序代码如下：

```
Sub Main()
    Dim dblA(4, 4) As Double
    Dim dblB(4) As Double
    Dim x(4) As Double

    '系数矩阵
```

```
    '系数矩阵
    dblA(1, 1) = 5:   dblA(1, 2) = 7:    dblA(1, 3) = 6:    dblA(1, 4) = 5
    dblA(2, 1) = 7:   dblA(2, 2) = 10:   dblA(2, 3) = 8:    dblA(2, 4) = 7
    dblA(3, 1) = 6:   dblA(3, 2) = 8:    dblA(3, 3) = 10:    dblA(3, 4) = 9
    dblA(4, 1) = 5:   dblA(4, 2) = 7:    dblA(4, 3) = 9:    dblA(4, 4) = 10

    '常数向量
    dblB(1) = 23
    dblB(2) = 32
    dblB(3) = 33
    dblB(4) = 31

    '求解
    Call LEGrad(4, dblA, dblB, x, 0.000001)

    MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
           "x1 = " & Round(x(1), 6) & Chr$(13) & _
           "x2 = " & Round(x(2), 6) & Chr$(13) & _
           "x3 = " & Round(x(3), 6) & Chr$(13) & _
           "x4 = " & Round(x(4), 6)

End Sub
```

图 3.12 程序输出结果

输出结果如图 3.12 所示。

3.13 求解线性最小二乘问题的豪斯荷尔德变换法

1. 算法原理

本节介绍用豪斯荷尔德(Householder)变换法求解线性最小二乘问题的算法。

设超定方程组为 $\boldsymbol{A}X=\boldsymbol{B}$，其中 $\boldsymbol{A}$ 为 $m\times n(m\geq n)$列线性无关的矩阵，X 为 n 维列向量，B 为 m 维列向量。

用豪斯荷尔德变换将 $\boldsymbol{A}$ 进行 QR 分解。即：$\boldsymbol{A}$=QR

其中，$\boldsymbol{Q}$ 为 $m\times m$ 的正交矩阵，$\boldsymbol{R}$ 为上三角矩阵。具体分解过程见 2.11 节介绍的方法。

设 $E = B-\boldsymbol{A}X$，用 $\boldsymbol{Q}^{\mathrm{T}}$ 乘上式两端得：

$$\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{E}=\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{B}-\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X}=\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{B}-\boldsymbol{R}\boldsymbol{X}$$

因为 $\boldsymbol{Q}^{\mathrm{T}}$ 为正交矩阵，所以，

$$\| E \|_2^2=\| Q^{\mathrm{T}}E \|_2^2=\| Q^{\mathrm{T}}B-RX \|_2^2$$

若令

$$Q^{\mathrm{T}}B=\begin{bmatrix}C\\D\end{bmatrix},\ RX=\begin{bmatrix}R_1\\0\end{bmatrix}X$$

其中，C 为 n 维列向量，D 为 m-n 维列向量，$\boldsymbol{R}_1$ 为 $n\times n$ 上三角方阵，$\boldsymbol{0}$ 为 $(m-n)\times n$ 的零矩阵。则：

$$\| E \|_2^2 = \| C - R_1 X \|_2^2 + \| D \|_2^2$$

上式中，当 X 满足 $R_1 X = C$ 时，$\| E \|_2^2$ 将取最小值。

综上所述，求解最小二乘问题线性方程组 ***A***X=B 的步骤如下：

(1) 对 ***A*** 进行 QR 分解。即 A=QR，其中 ***Q*** 为 $m \times m$ 的正交矩阵，R 为上三角矩阵。令：

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix}$$

其中，$\boldsymbol{R}_1$ 为 $n \times n$ 上三角方阵，***0*** 为 $(m-n) \times n$ 的零矩阵。

(2) 计算

$$\begin{bmatrix} C \\ D \end{bmatrix} = \boldsymbol{Q}^{\mathrm{T}} B$$

其中 C 为 n 维列向量。

(3) 利用回代求解方程组 $\boldsymbol{R}_1 X = C$。

2. 算法实现

基于上述的计算方法，可以定义用豪斯荷尔德变换法求解线性最小二乘问题方程组的 Visual Basic 函数 LEMqr，其中调用了 2.11 节介绍的矩阵 QR 函数 MMqr。LEMqr 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEMqr
'  功能：  用豪斯荷尔德变换法求解线性最小二乘问题方程组
'  参数：   m    - Integer型变量。系数矩阵的行数， m>=n
'           n    - Integer型变量。系数矩阵的列数，n<=m
'          dblA  - Double型二维数组，体积维n x n。存放系数矩阵；
'                 返回时，存放分解式中的R矩阵.
'          dblB  - Double型一维数组，长度为m。存放方程组右端常数向量；
'                 返回时，前n个元素存放方程组的最小二乘解。
'          dblQ  - Double型二维数组，体积为m x m。返回时，存放分解式中的Q矩阵
'  返回值： Boolean型。False，失败无解；True， 成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function LEMqr(m As Integer, n As Integer, dblA() As Double, _
dblB() As Double, dblQ() As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer
    Dim d As Double
    ReDim c(n) As Double

    ' QR分解失败，返回
    If (Not MMqr(m, n, dblA, dblQ)) Then
        LEMqr = False
        Exit Function
```

```
    End If

    For i = 1 To n
       d = 0#
       For j = 1 To m
          d = d + dblQ(j, i) * dblB(j)
       Next j
       c(i) = d
    Next i

    dblB(n) = c(n) / dblA(n, n)
    For i = n - 1 To 1 Step -1
       d = 0#
       For j = i + 1 To n
          d = d + dblA(i, j) * dblB(j)
       Next j
       dblB(i) = (c(i) - d) / dblA(i, i)
    Next i

    ' 求解结束，返回
    LEMqr = True

End Function
```

3. 示例

调用上述函数 LEMqr **求解超定方程组** $\boldsymbol{A}$X=$\boldsymbol{B}$，**其中，**

$$\boldsymbol{A}=\begin{bmatrix}1 & 1 & -1\\ 2 & 1 & 0\\ 1 & -1 & 0\\ -1 & 2 & 1\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}2\\ -3\\ 1\\ 4\end{bmatrix}$$

程序代码如下：

```
Sub Main()
    Dim dblA(4, 3) As Double
    Dim dblQ(4, 4) As Double
    Dim dblB(4) As Double

    '系数矩阵
    dblA(1, 1) = 1:   dblA(1, 2) = 1:    dblA(1, 3) = -1
    dblA(2, 1) = 2:   dblA(2, 2) = 1:   dblA(2, 3) = 0
    dblA(3, 1) = 1:    dblA(3, 2) = -1:   dblA(3, 3) = 0
    dblA(4, 1) = -1:    dblA(4, 2) = 2:    dblA(4, 3) = 1

    '常数向量
    dblB(1) = 2
    dblB(2) = -3
    dblB(3) = 1
    dblB(4) = 4
```

```
    '求解
    If LEMqr(4, 3, dblA, dblB, dblQ) Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "x1 = " & Round(dblB(1), 6) & Chr$(13) & _
               "x2 = " & Round(dblB(2), 6) & Chr$(13) & _
               "x3 = " & Round(dblB(3), 6) & Chr$(13) & Chr$(13) & _
               "矩阵Q" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 4, dblQ, "######0.0000000") & _
               Chr$(13) & Chr$(13) & "矩阵R" & Chr$(13) & Chr$(13) & _
               MatrixToString(4, 3, dblA, "######0.0000000")
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

输出结果如图 3.13 **所示。**

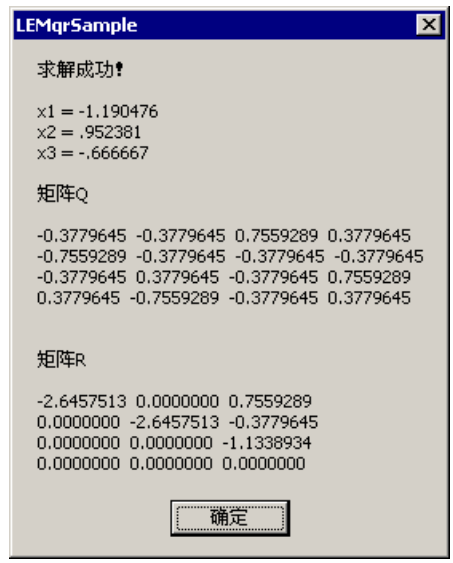

图 3.13　程序输出结果

3.14　求解线性最小二乘问题的广义逆法

1. **算法原理**

本节介绍利用广义逆求超定方程组 $\boldsymbol{A}$X=B **的最小二乘解，其中** $\boldsymbol{A}$ **为** $m\times n(m\geq n)$**的矩阵，且列线性无关。当** $m=n$**时，即为求线性代数方程组的解。**

(1)　**首先对矩阵** $\boldsymbol{A}$ **进行奇异值分解，即：**

$$\boldsymbol{A}=U\begin{bmatrix}\Sigma & 0\\ 0 & 0\end{bmatrix}V^{\mathrm{T}}$$

(2) **然后利用奇异值分解式计算 A 的广义逆 A^+。即**

$$\boldsymbol{A}^+ = V_1\Sigma^{-1}U_1^{\mathrm{T}}$$

以上两步请参看 2.12 **节的方法说明。**

(3) **最后利用广义逆 A^+求超定方程组 $\boldsymbol{AX}=B$ 的最小二乘解。即 $X=\boldsymbol{A}^+B$**

2. **算法实现**

基于上述方法，可以定义用广义逆法求解线性最小二乘问题方程组的 Visual Basic **函数** LEMiv，**其中调用了** 2.13 **节中求矩阵广义逆的函数** MInv，MInv **又调用了** 2.12 **节介绍的矩阵奇异值分解函数** MUav。LEMiv **的代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：LEModule.bas
'  函数名：LEMiv
'  功能：   用广义逆法求解线性最小二乘问题方程组
'  参数：    m    - Integer型变量。系数矩阵的行数， m>=n
'            n    - Integer型变量。系数矩阵的列数，n<=m
'           dblA  - Double型二维数组，体积维n x n。存放超定方程组系数矩阵；
'                   返回时，其对角线存放矩阵的奇异值，其余元素为0。
'           dblB  - Double型一维数组，长度为m。存放超定方程组右端常数向量
'           dblX  - Double型一维数组，长度为n。返回时，存放超定方程组的最小二乘解。
'           dblAP - Double型二维数组，体积维n x m。
'                   返回时，存放超定方程组系数矩阵A的广义逆A+。
'           dblU  - Double型二维数组，体积维m x m。
'                   返回时，存放超定方程组系数矩阵A的奇异值分解式中的左奇异向量U。
'           dblV  - Double型二维数组，体积维n x n。
'                   返回时，存放超定方程组系数矩阵A的奇异值分解式中的右奇异向量VT。
'            ka  - Integer型变量。ka=max(m,n)+1
'           eps  - Double型变量。奇异值分解函数中的控制精度参数。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function LEMiv(m As Integer, n As Integer, dblA() As Double, _
dblB() As Double, dblX() As Double, dblAP() As Double, _
dblU() As Double, dblV() As Double, ka As Integer, eps As Double) As Boolean
    ' 局部变量
    Dim i As Integer, j As Integer

    If (Not MInv(m, n, dblA, dblAP, dblU, dblV, ka, eps)) Then
        LEMiv = False
        Exit Function
    End If

    For i = 1 To n
        dblX(i) = 0#
        For j = 1 To m
          dblX(i) = dblX(i) + dblAP(i, j) * dblB(j)
        Next j
    Next i
```

```
LEMiv = True

End Function
```

3. 示例

调用上述函数 LEMiv **求解超定方程组** $\boldsymbol{A}X=\boldsymbol{B}$**，其中，**

$$\boldsymbol{A}=\begin{bmatrix}1 & 1 & -1\\ 2 & 1 & 0\\ 1 & -1 & 0\\ -1 & 2 & 1\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}2\\ -3\\ 1\\ 4\end{bmatrix}$$

程序代码如下：

```
Sub Main()
   Dim dblA(4, 3) As Double
   Dim dblAP(3, 4) As Double
   Dim dblU(4, 4) As Double
   Dim dblV(3, 3) As Double
   Dim dblB(4) As Double
   Dim dblX(3) As Double

   '系数矩阵
   dblA(1, 1) = 1:   dblA(1, 2) = 1:    dblA(1, 3) = -1
   dblA(2, 1) = 2:   dblA(2, 2) = 1:    dblA(2, 3) = 0
   dblA(3, 1) = 1:   dblA(3, 2) = -1:   dblA(3, 3) = 0
   dblA(4, 1) = -1:  dblA(4, 2) = 2:    dblA(4, 3) = 1

   '常数向量
   dblB(1) = 2
   dblB(2) = -3
   dblB(3) = 1
   dblB(4) = 4

   '求解
   If LEMiv(4, 3, dblA, dblB, dblX, dblAP, dblU, dblV, 5, 0.000001) Then
      MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
             "x1 = " & Round(dblX(1), 6) & Chr$(13) & _
             "x2 = " & Round(dblX(2), 6) & Chr$(13) & _
             "x3 = " & Round(dblX(3), 6) & Chr$(13) & Chr$(13) & _
             "矩阵A+" & Chr$(13) & Chr$(13) & _
             MatrixToString(3, 4, dblAP, "######0.0000000")
   Else
      MsgBox "求解失败！"
End If

End Sub
```

输出结果如图 3.14 **所示。**

图 3.14　程序输出结果

3.15　病态方程组的求解

1. 算法原理

设线性代数方程组 $\boldsymbol{A}$X=B **是病态的。其求解的步骤如下：**

(1)　**用全选主元高斯消去法求解，得到一组近似值** $X^{(1)}=(x_0^{(1)}, x_1^{(2)}, \Lambda , x_{n-1}^{(1)})^{\mathrm{T}}$ **。**

(2)　**计算剩余向量**

$$\boldsymbol{R}=\boldsymbol{B}-\boldsymbol{AX}^{(1)}$$

(3)　**用全选主元高斯消去法求解线性代数方程组**

$$\boldsymbol{A}\mathrm{E}=\mathrm{R}$$

解出 $\mathrm{e}=(\mathrm{e}_0,\ \mathrm{e}_1,\ \Lambda\ ,\mathrm{e}_{n-1})^{\mathrm{T}}$ **。**

(4)　**计算**

$$X^{(2)}=X^{(1)}+E$$

(5)　**令** $X^{(1)}=X^{(2)}$ **，转**(2)**重复这个过程。直到**

$$\max_{0\le i\le n-1}\frac{\left|x_i^{(2)}-x_i^{(1)}\right|}{1+\left|x_i^{(2)}\right|}<\varepsilon$$

为止。

2. 算法实现

基于上述的计算方法，可以定义求解病态方程组的 Visual Basic **函数** LEMorbid**，其中调用了** 3.1 **节中的用全选主元高斯消去法求解线性代数方程组的函数** LEGauss **和** 2.1 **节的矩阵乘法函数** MMul**。**LEMorbid **的代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'　模块名：LEModule.bas
'　函数名：LEMorbid
```

```
'  功能：  求解病态方程组
'  参数：   n    - Integer型变量，方程组的阶数。
'          dblA  - Double型二维数组，体积维n x n。存放病态方程组系数矩阵。
'          dblB  - Double型一维数组，长度为n，存放方程组右端常数向量。
'          dblX  - Double型一维数组，长度为n。返回时，存放方程组的解向量。
'          eps   - Double型变量。控制精度参数。
'  返回值： Boolean型。False，失败无解；True，成功
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function LEMorbid(n As Integer, dblA() As Double, dblB() As Double, _
dblX() As Double, eps As Double) As Boolean
   ' 局部变量
   Dim i As Integer, j As Integer, k As Integer, kk As Integer
   Dim q As Double, qq As Double
   ReDim p(n, n) As Double, r(n) As Double, e(n, 1) As Double, _
        x(n, 1) As Double, xx(n) As Double

   i = 60
   For k = 1 To n
      For j = 1 To n
         p(k, j) = dblA(k, j)
      Next j
   Next k

   For k = 1 To n
      x(k, 1) = dblB(k)
   Next k

   For k = 1 To n
      xx(k) = x(k, 1)
   Next k

   ' 全选主元高斯消去法
   If (Not LEGauss(n, p, xx)) Then
      LEMorbid = False
      Exit Function
   End If

   For k = 1 To n
      x(k, 1) = xx(k)
   Next k

   q = 1# + eps
   While (q >= eps)
      If (i = 0) Then
         LEMorbid = False
         Exit Function
      End If

      i = i - 1

      ' 矩阵乘法
```

```
        Call MMul(n, n, 1, dblA, x, e)

        For k = 1 To n
            r(k) = dblB(k) - e(k, 1)
        Next k

        For k = 1 To n
            For j = 1 To n
                p(k, j) = dblA(k, j)
            Next j
        Next k

        ' 全选主元高斯消去法
        If (Not LEGauss(n, p, r)) Then
            LEMorbid = False
            Exit Function
        End If

        q = 0#
        For k = 1 To n
            qq = Abs(r(k)) / (1# + Abs(x(k, 1) + r(k)))
            If (qq > q) Then q = qq
        Next k

        For k = 1 To n
            x(k, 1) = x(k, 1) + r(k)
        Next k
    Wend

    ' 解赋值返回
    For k = 1 To n
        dblX(k) = x(k, 1)
    Next k

    LEMorbid = True

End Function
```

3. 示例

调用上述函数 LEMorbid **求解下列** 4 **阶病态方程组**

$$\begin{bmatrix} 3.4336 & -0.5238 & 067105 & -0.15272 \\ -0.5238 & 3.28326 & -0.73051 & -0.2689 \\ 0.67105 & -0.73051 & 4.02612 & 0.01835 \\ -0.15272 & -0.2680 & 0.01835 & 2.75702 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.5 \\ 2.5 \\ -2.0 \end{bmatrix}$$

取 ε =0.000001。

程序代码如下：

```
Sub Main()
    Dim dblA(4, 4) As Double
    Dim dblB(4) As Double
    Dim dblX(4) As Double

    '系数矩阵
dblA(1,1)=3.4336: dblA(1,2)=-0.5238:dblA(1,3)=0.67105:dblA(1,4)=-0.15271
dblA(2,1)=-0.5238:dblA(2,2)=3.28326:dblA(2,3)=-0.73051:dblA(2,4)=-0.2689
dblA(3,1)=0.67105:dblA(3,2)=-0.73051:dblA(3,3)=4.02612:dblA(3,4)=0.01835
dblA(4,1)=-0.15272:dblA(4,2)=-0.268:dblA(4,3)=0.01835:dblA(4,4)=2.75702

    '常数向量
    dblB(1) = -1#
    dblB(2) = 1.5
    dblB(3) = 2.5
    dblB(4) = -2#

    '求解
    If LEMorbid(4, dblA, dblB, dblX, 0.000001) Then
        MsgBox "求解成功！" & Chr$(13) & Chr$(13) & _
               "x1 = " & Round(dblX(1), 6) & Chr$(13) & _
               "x2 = " & Round(dblX(2), 6) & Chr$(13) &
_
               "x3 = " & Round(dblX(3), 6) & Chr$(13) &
_
               "x4 = " & Round(dblX(4), 6)
    Else
        MsgBox "求解失败！"
    End If

End Sub
```

图 3.15　程序输出结果

输出结果如图 3.15 **所示。**

第 4 章　非线性方程与方程组的求解

与求解线性问题不同，非线性方程和方程组的求解都是通过迭代法来逐步逼近的。从一些近似实验解出发，当一些预定的收敛规则满足时，将得到一个求解的实用算法。

非线性方程和方程组的求解也是在实际工作中经常遇到的问题。本章将介绍科学和工程应用中常用的非线性方程和方程组的求解方法，包括求非线性方程实根的对分法、求非线性方程一个实根的牛顿法、求非线性方程一个实根的埃特金迭代法、求非线性方程一个实根的连分式解法、求实系数代数方程全部根的 QR 方法、求实系数代数方程全部根的牛顿-下山法、求复系数代数方程全部根的牛顿-下山法、求非线性方程组一组实根的剃度法、求非线性方程组一组实根的拟牛顿法、求非线性方程组最小二乘解的广义逆法、求非线性方程一个实根的蒙特卡洛法、求实函数或复函数方程一个复根的蒙特卡洛法和求非线性方程组一组实根的蒙特卡洛法等。

4.1　求非线性方程实根的对分法

1. 算法原理

对分法(又叫做二分法)是求解非线性方程实根的常用方法。用对分法搜索方程 $f(x)=0$ 在区间[a，b]内的实根的方法如下：

(1) 从端点 $x_0=a$ 开始，以 h 为步长，逐步往后进行搜索。

(2) 对于每一个子区间 $[x_i, x_{i+1}]$(其中 $x_{i+1}=x_i+h$)：

① 若 $f(x_i)=0$，则 x_i 为一个实根，且从 $x_i+\frac{h}{2}$ 开始再往后搜索；

② 若 $f(x_{i+1})=0$，则 x_{i+1} 为一个实根，且从 $x_{i+1}+\frac{h}{2}$ 开始再往后搜索；

③ 若 $f(x_i)f(x_{i+1})>0$，则说明当前子区间内无实根，从 x_{i+1} 开始再往后搜索；

④ 若 $f(x_i)f(x_{i+1})<0$，则说明当前子区间内有实根。此时，反复将子区间减半，直到发现一个实根或子区间长度小于 ε 为止。在后一种情况下，子区间的中点即取为方程的一个实根。然后再从 x_{i+1} 开始再往后搜索。其中 ε 为预先给定的精度要求。

(3) 上述过程一直进行到区间右端点 b 为止。

在使用本方法时，要注意步长 h 的选择。若步长 h 选得过大，可能会导致某些实根的丢失；若步长 h 选得过小，则会增加计算工作量。

2. 算法实现

根据上述方法，可以定义对分法的 Visual Basic **函数** NLBisectRoot**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  功能：  求解非线性方程和方程组
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLBisectRoot
'  功能：  使用对分法求解非线性方程的实根，本函数需要调用计算方程左端函数f(x)值的函数
'          Func，其形式为：
'           Function Func(x as double) as double
'  参数：  m    - Integer型变量，在[a, b]内实根个数的预估值
'          a    - Double型变量，求根区间的左端点
'          b    - Double型变量，求根区间的右端点
'          h    - Double型变量，搜索求根时采用的步长
'          x    - Double型一维数组，长度为m。
'                 返回在区间[a, b]内搜索到的实根，实根个数由函数值返回
'        eps    - Double型变量，精度控制参数
'  返回值：Integer型，求得的实根的个数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLBisectRoot(m As Integer, a As Double, b As Double, _
h As Double, x() As Double, eps As Double) As Integer
    Dim n As Integer, js As Integer
    Dim z As Double, y As Double, z1 As Double, y1 As Double, _
        z0 As Double, y0 As Double

    ' 根的个数清0
    n = 0

    ' 左边界函数值
    z = a
    y = Func(z)

    ' 迭代求解，直到到达右边界
    While ((z <= b + h / 2#) And (n <> m))
        ' 如果精度满足要求，则求得一个实根，继续计算下一步
        If (Abs(y) < eps) Then
            n = n + 1
            x(n) = z
            z = z + h / 2#
            y = Func(z)
        Else
            z1 = z + h
            y1 = Func(z1)
            If (Abs(y1) < eps) Then
                n = n + 1
```

```
                x(n) = z1
                z = z1 + h / 2#
                y = Func(z)
            Else
                If (y * y1 > 0#) Then
                    y = y1
                    z = z1
                Else
                    js = 0
                    While (js = 0)
                        If (Abs(z1 - z) < eps) Then
                            n = n + 1
                            x(n) = (z1 + z) / 2#
                            z = z1 + h / 2#
                            y = Func(z)
                            js = 1
                        Else
                            z0 = (z1 + z) / 2#
                            y0 = Func(z0)
                            If (Abs(y0) < eps) Then
                                x(n) = z0
                                n = n + 1
                                js = 1
                                z = z0 + h / 2#
                                y = Func(z)
                            Else
                                If ((y * y0) < 0#) Then
                                    z1 = z0
                                    y1 = y0
                                Else
                                    z = z0
                                    y = y0
                                End If
                            End If
                        End If
                    Wend
                End If
            End If
        End If
    Wend

    ' 返回求得的根的个数
    NLBisectRoot = n

End Function
```

3. 示例

调用上述函数 NLBisectRoot **求方程**

$$f(x)=x^6-5x^5+3x^4+x^3-7x^2+7x-20=0$$

在区间[-2，5]**内的所有实根。取步长** h=0.2，ε=0.000001。
由于本方程为 6 **次代数方程，最多有** 6 **个实根，因此取** m=6。
程序代码如下：

```
Sub Main()
    Dim m As Integer, n As Integer
    Dim s As String

    m = 6
    ReDim x(m) As Double

    '求解
    n = NLBisectRoot(m, -2, 5, 0.2, x, 0.000001)

    s = ""
    For i = 1 To n
        s = s + "x(" & i & ") = " & x(i) & Chr(13)
    Next i

    MsgBox "共求解 " & n & " 个实根，分别为：" & Chr(13) & Chr(13) & s

End Sub

' 待求解的方程的函数
Function Func(x As Double) As Double
    Func = x * x * x * x * x * x - 5 * x * x * x * x * x + _
           3 * x * x * x * x + x * x * x - 7 * x * x + 7 * x - 20
End Function
```

其输出结果如图 4.1 **所示。**

图 4.1　程序输出结果

4.2　求非线性方程一个实根的牛顿法

1. 算法原理

本节介绍用牛顿(Newton)**迭代法求方程** $f(x)=0$ **的一个实根的算法。**
设方程 $f(x)=0$ **满足以下条件：**

(1) $f(x)$ 在区间 $[a,\ b]$ 上，其 $f'(x)$ 与 $f''(x)$ 均存在，且各自保持固定符号；

(2) $f(a)f(b)<0$；

(3) $f(x_0)f''(x)>0$ 且 $x,\ x_0\in[a,\ b]$。

则方程 $f(x)=0$ 在区间 $[a,b]$ 上有且只有一个实根，取初值 x_0，由牛顿迭代格式

$$x_{n+1}=x_n-f(x_n)/f'(x_n)$$

计算得到的序列 $x_0, x_1, \Lambda, x_n, \Lambda$ 收敛于方程 $f(x)=0$ 的根。

结束迭代过程的条件为

$$\begin{cases}|f(x_{n+1})|<\varepsilon\\ |x_{n+1}-x_n|<\varepsilon\end{cases}$$

同时成立，其中 ε 为预先给定的精度要求。

2. 算法实现

根据上述方法，可以定义牛顿迭代法的 Visual Basic 函数 NLNewtonRoot，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLNewtonRoot
'  功能：  使用牛顿迭代法求解非线性方程的实根，本函数需要调用计算方程左端函数f(x)值的
'          过程Func，其形式为：
'          Sub Func(x As Double, y() as double)
'          y(1) 返回f(x)的值
'          y(2) 返回f'(x)的值
'  参数：  x    - Double型变量，输入时存放迭代初值；返回时存放迭代终值，即方程的根
'         js    - Integer型变量，最大迭代次数
'         eps   - Double型变量，精度控制参数
'  返回值：Integer型，若小于0，则表示在求解失败；若等于最大迭代次数js，则表示迭代了js
'          次还未满足精度要求，返回的实根只作为参考；若大于等于0且小于最大迭代次数js，
'          则表示正常返回。
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLNewtonRoot(x As Double, js As Integer, eps As Double) As Integer
    Dim k As Integer, l As Integer
    Dim y(2) As Double, d As Double, p As Double, x0 As Double, x1 As Double

    ' 初值
    l = js
    k = 1
    x0 = x

    ' 计算f(x)和f'(x)
    Call Func(x0, y)

    d = eps + 1#

    ' 迭代计算
    While ((d >= eps) And (l <> 0))
```

```
        ' 求解失败，返回
        If (Abs(y(2)) + 1# = 1#) Then
            NLNewtonRoot = -1
            Exit Function
        End If

        x1 = x0 - y(1) / y(2)

        Call Func(x1, y)

        d = Abs(x1 - x0)
        p = Abs(y(1))

        If (p > d) Then d = p

        x0 = x1
        l = l - 1
    Wend

    ' 求解结束
    x = x1
    k = js - l

    NLNewtonRoot = k

End Function
```

3. 示例

调用上述 NLNewtonRoot **函数求方程** $f(x)=x^3-x^2-1=0$ **在** $x_0=1.5$ **附近的一个实根。取** ε =0.000001**，最大迭代次数为** 60**。其中** $f'(x)=3x^2-2x$ **。**

程序代码如下：

```
Sub Main()
    Dim n As Integer
    Dim x As Double

    ' 迭代初值
    x = 1.5

    '求解
    n = NLNewtonRoot(x, 60, 0.000001)

    If n >= 0 Then
        MsgBox "x = " & x
    Else
        MsgBox "求解失败"
    End If

End Sub
```

```
' 待求解的方程的函数
Sub Func(x As Double, y() As Double)
    y(1) = x * x * x - x * x - 1
    y(2) = 3 * x * x - 2 * x
End Sub
```

其输出结果如图 4.2 所示。

图 4.2 程序输出结果

4.3 求非线性方程一个实根的埃特金迭代法

1. 算法原理

埃特金(Aitken)迭代法是求解求非线性方程 $x = \varphi(x)$ 的一个实根的常用方法。

设非线性方程为：

$$x = \varphi(x)$$

取初值 x_0，埃特金迭代一次的过程如下：

预报 $u = \varphi(x_k)$

再预报 $v = \varphi(u)$

校正 $x_{k+1} = v - \dfrac{(v-u)^2}{v - 2u + x_k}$

结束迭代过程的条件为：

$$|v - u| < \varepsilon$$

此时，v 即为非线性方程的一个实根。其中 ε 为预先给定的精度要求。

2. 算法实现

根据上述方法，可以定义埃特金迭代法的 Visual Basic 函数 NLAitkenRoot，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLAitkenRoot
'  功能：  使用埃特金迭代法求解非线性方程的一个实根，本函数需要调用计算方程左端函数
'          f(x)值的过程Func，其形式为：
'          Function Func(x as double) as double
'  参数：  x    - Double型变量，输入时存放迭代初值；返回时存放迭代终值，即方程的根
'         js    - Integer型变量，最大迭代次数
'         eps   - Double型变量，精度控制参数
'  返回值：Integer型，若为0，则表示迭代了js次还未满足精度要求，返回的实根只作为参考；
'                    若大于0，则表示正常返回。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLAitkenRoot(x As Double, js As Integer, eps As Double) As Integer
    Dim flag As Integer, l As Integer
```

```
    Dim u As Double, v As Double, x0 As Double

    ' 迭代初值
    l = 0
    x0 = x
    flag = 0

    ' 迭代求解
    While ((flag = 0) And (l <> js))
        l = l + 1
        u = Func(x0)
        v = Func(u)
        If (Abs(u - v) < eps) Then
            x0 = v
            flag = 1
        Else
            x0 = v - (v - u) * (v - u) / (v - 2# * u + x0)
        End If
    Wend

    ' 求解结束
    x = x0
    l = js - l

    NLAitkenRoot = l

End Function
```

3. 示例

调用上述 NLAitkenRoot **函数求方程** $f(x)=6-x^2=0$ **在** $x_0=0.0$ **附近的一个实根。取** ε =0.000001**，最大迭代次数为** 60**。**

程序代码如下：

```
Sub Main()
    Dim n As Integer
    Dim x As Double

    ' 迭代初值
    x = 0

    '求解
    n = NLAitkenRoot(x, 60, 0.000001)

    If n > 0 Then
        MsgBox "x = " & x
    Else
        MsgBox "求解失败"
    End If

End Sub
```

```
' 待求解的方程的函数
Function Func(x As Double) As Double
    Func = 6 - x * x
End Function
```

图 4.3 程序输出结果

其输出结果如图 4.3 所示。

4.4 求非线性方程一个实根的连分式解法

1. 算法原理

连分式解法也是求解非线性方程一个实根的常用方法。

设非线性方程为 $f(x)=0$，而 $f(x)$ 的反函数为 $x=F(y)$，并用连分式表示为

$$x= a_0+ \frac{y-y_0}{a_1} +\frac{y-y_1}{a_2} + \cdots + \frac{y-y_{i-1}}{a_i} +\cdots$$

其中，$y_i = f(x_i)$。因此，满足方程 $f(x)=0$ 的根为：

$$x= a_0 - \frac{y_0}{a_1} - \frac{y_1}{a_2} - \cdots - \frac{y_{i-1}}{a_i} - \cdots$$

由上所述，求非线性方程 $f(x)=0$ 的一个实根的过程如下：

选取初值 x_0 及 x_1，并计算出 $y_0 = f(x_0)$，$y_i = f(x_i)$，根据 (x_0, y_0) 及 (x_1, y_1) 可以计算出：

$$a_0 = x_0$$
$$a_1 = (y_1 - y_0)/(x_1 - x_0)$$

由此可以计算出 $x_2 = a_0 - \dfrac{y_0}{a_1}$。

一般来说，若已知点列 $(x_0,\ y_0)$，$(x_1,\ y_1), \Lambda\ , (x_{i-1},\ y_{i-1})$，并已求得 $a_0, a_1, \Lambda\ , a_{i-1}$，则可以计算出：

$$x_i= a_0 - \frac{y_0}{a_1} - \frac{y_1}{a_2} - \cdots - \frac{y_{i-2}}{a_{i-1}} - \cdots$$

及

$$y_i = f(x_i)。$$

然后通过下列递推公式计算出 a_i：

$$a_0 = x_0$$
$$a_{0i} = x_i$$
$$a_{j+1,i} = \frac{y_i - y_j}{a_{ji} - a_j}, \qquad j = 0, 1, 2, \Lambda\ , i-1$$
$$a_i = x_{ii}$$

以上过程一直进行到$|y_i| < \varepsilon$为止。其中ε为预先给定的精度要求。

在实际计算构成中，上述的连分式最多作到七节为止。如果此时还不满足精度要求，则将最后得到的实根近似值作为新的初值再重新计算。

本方法对函数$f(x)$的要求比较低，在有实根的情况下，一般都能通过本方法求出一个实根。

2. 算法实现

根据上述方法，可以定义连分式解法的 Visual Basic 函数 NLPqRoot，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLPqRoot
'  功能：  使用连分式解法求解非线性方程的一个实根，本函数需要调用计算方程左端函数
'          f(x)值的过程Func，其形式为：
'          Function Func(x as double) as double
'  参数：  x    - Double型变量，输入时存放迭代初值；返回时存放迭代终值，即方程的根
'          eps  - Double型变量，精度控制参数
'  返回值：Boolean型，若为False，则表示迭代了10次还未满足精度要求，返回的实根只作为
'          参考；若为True，则表示正常返回。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLPqRoot(x As Double, eps As Double) As Boolean
    Dim i As Integer, j As Integer, m As Integer, it As Integer, l As Integer
    Dim a(10) As Double, y(10) As Double, z As Double, h As Double, _
         x0 As Double, q As Double

    ' 迭代初值
    l = 10
    q = 1E+35
    x0 = x
    h = 0#

    ' 迭代计算
    While (l <> 0)
        l = l - 1
        j = 0
        it = l

        ' 最多迭代到第7节连分式
        While (j <= 7)
            If (j <= 2) Then
               z = x0 + 0.1 * j
            Else
```

```
            z = h
        End If

        ' 函数值
        y(j + 1) = Func(z)
        h = z
        If (j = 0) Then
          a(1) = z
        Else
          m = 0
          i = 0
          While ((m = 0) And (i <= j - 1))
            If (Abs(h - a(i + 1)) + 1# = 1#) Then
              m = 1
            Else
              h = (y(j + 1) - y(i + 1)) / (h - a(i + 1))
            End If

            i = i + 1
          Wend
          a(j + 1) = h
          If (m <> 0) Then a(j + 1) = q
          h = 0#
          For i = j To 1 Step -1
            If (Abs(a(i + 1) + h) + 1# = 1#) Then
              h = q
            Else
              h = -y(i) / (a(i + 1) + h)
            End If
          Next i
          h = h + a(1)
        End If

        ' 精度达到要求，则结束迭代
        If (Abs(y(j + 1)) >= eps) Then
          j = j + 1
        Else
          j = 10
          l = 0
        End If

    Wend

    x0 = h

Wend

' 求解结束
x = h

' 判断解的合理性
```

```
    If it = 0 Then
        NLPqRoot = False
        Exit Function
    End If

    ' 正常解
    NLPqRoot = True

End Function
```

3. **示例**

调用上述 NLPqRoot **函数求解方程** $f(x)=x^3-x^2-1=0$ **在** $x_0=1.0$ **附近的一个实根，取** $\varepsilon=0.000001$。

程序代码如下：

```
Sub Main()
    Dim x As Double

    ' 迭代初值
    x = 1

    '求解
    If NLPqRoot(x, 0.000001) Then
        MsgBox "x = " & x
    Else
        MsgBox "求解失败"
    End If

End Sub

' 待求解的方程的函数
Function Func(x As Double) As Double
    Func = x * x * x - x * x - 1
End Function
```

图 4.4　程序输出结果

其输出结果如图 4.4 **所示。**

4.5　求实系数代数方程全部根的 QR 方法

1.**算法原理**

本节介绍用 QR **方法求实系数** n **次多项式方程**

$$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\Lambda\ +a_1x+a_0=0$$

的全部根的算法。

令　　$b_i=a_i/a_n,\ \ i=n-1,\Lambda\ ,1,0$

则一般实系数 n **次多项式方程** $P_n(x)=0$ **化为** n **次首一多项式方程**

$$Q_n(x)=x^n+b_{n-1}x^{n-1}+\Lambda\ +b_1x+b_0=0$$

由线性代数可知，$Q_n(x)=0$ 可以看成是如下实矩阵

$$\boldsymbol{Q}=\begin{bmatrix} -b_{n-1} & -b_{n-2} & -b_{n-3} & \Lambda & -b_1 & -b_0 \\ 1 & 0 & 0 & \Lambda & 0 & 0 \\ 0 & 1 & 0 & \Lambda & 0 & 0 \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ 0 & 0 & 0 & \Lambda & 0 & 0 \\ 0 & 0 & 0 & \Lambda & 1 & 0 \end{bmatrix}$$

的特征多项式方程

$$Q_n(\lambda)=\lambda^n+b_{n-1}\lambda^{n-1}+\Lambda\ +b_1\lambda+b_0=0$$

因此，求方程 $Q_n(x)=0$ 的全部根就变为求上述矩阵 $\boldsymbol{Q}$ 的全部特征值。矩阵 $\boldsymbol{Q}$ 为一个上 H 阵，可以直接用 QR 方法求出全部特征值。

有关用 QR 方法求上 H 阵全部特征值的方法可参阅 2.16 节。

2. 算法实现

根据上述方法，可以定义 QR 方法的 Visual Basic 函数 NLQrRoot，其中调用了 2.16 节中求上 H 阵全部特征值的函数 MHbergEigenv。NLQrRoot 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLQrRoot
'  功能：  使用QR方法求解实系数代数方程的全部根，本函数需要调用用QR方法计算上H阵全部
'          特征值的函数MhbergEigenv
'  参数：  n     - 多项式方程的次数
'          a     - Double型一维数组，长度为n+1，按降幂次序依次存放n次多项式方程的n+1
'                   个系数
'          xr    - Double型一维数组，长度为n，返回n个根的实部
'          xi    - Double型一维数组，长度为n，返回n个根的虚部
'         eps   - Double型变量，精度控制参数
'         nMaxItNum    - Integer型变量，控制QR方法的最大迭代次数
'  返回值：Boolean型，若为False，则表示在QR方法中迭代已超过最大迭代次数但还未满足精
'          度要求；若为True，则表示求解成功。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLQrRoot(n As Integer, a() As Double, xr() As Double, _
xi() As Double, eps As Double, nMaxItNum As Integer) As Boolean
    Dim i As Integer, j As Integer
    ReDim q(n, n) As Double

    ' 用最高幂系数约化其余系数
    For j = 1 To n
        q(1, j) = -a(j + 1) / a(1)
    Next j

    ' 构造上H阵
    For i = 2 To n
        For j = 1 To n
            q(i, j) = 0#
```

```
        Next j
    Next i

    For i = 2 To n
        q(i, i - 1) = 1#
    Next i

    ' 求上H阵的特征值，即方程的全部根
    NLQrRoot = MHbergEigenv(n, q, xr, xi, eps, nMaxItNum)

End Function
```

3. **示例**

调用上述 NLQrRoot **函数用** QR **方法求** 6 **次多项式方程**

$$P_6(x) = x^6 - 5x^5 + 3x^4 + x^3 - 7x^2 + 7x - 20 = 0$$

的全部根，取 ε =0.000001**，最大迭代次数为** 60**。**

程序代码如下：

```
Sub Main()
    Dim n As Integer
    Dim s As String

    ' 6次方程
    n = 6

    ' 分配存放系数和解的内存
    ReDim a(n + 1) As Double, xr(n) As Double, xi(n) As Double

    ' 系数数组
    a(1) = 1
    a(2) = -5
    a(3) = 3
    a(4) = 1
    a(5) = -7
    a(6) = 7
    a(7) = -20

    '求解
    If NLQrRoot(n, a, xr, xi, 0.000001, 60) Then
        s = ""
        For i = 1 To n
            s = s & "x(" & i & ") = " & xr(i) & " + (" & xi(i) & ")j" & Chr(13)
        Next i

        MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
        MsgBox "求解失败"
    End If
```

```
End Sub
```

其输出结果如图 4.5 所示。

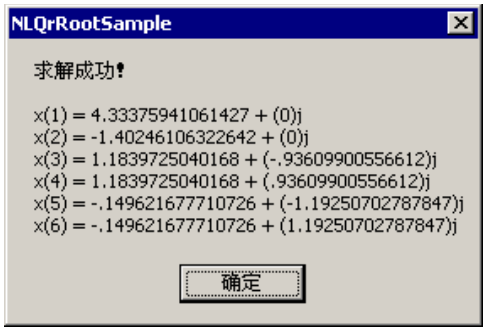

图 4.5 程序输出结果

4.6 求实系数代数方程全部根的牛顿-下山法

1. 算法原理

牛顿-下山法是求解实系数代数方程全部根的重要方法。

对实系数代数方程

$$f(z)=a_n z^n+a_{n-1}z^{n-1}+\Lambda+a_1 z+a_0=0$$

牛顿-下山法的迭代格式

$$z_{i+1}=z_i-tf(z_i)/f'(z_i)$$

选取适当的 t 可以保证有

$$|f(z_{i+1})|^2<|f(z_i)|^2$$

迭代计算直到$|f(z_i)|^2<\varepsilon$为止。

迭代格式在鞍点或接近重根点时，可能因为$f'(z)\approx 0$而$|f(z)|^2\neq 0$而失败。在本算法中，采用撒网络的方法来解决这一问题。即选取适当的 d 与 c，用

$$x_{i+1}=x_i+d\cos(c)$$
$$y_{i+1}=y_i+d\sin(c)$$

计算，使$|f(z_{i+1})|^2<|f(z_i)|^2$而冲过鞍点，或使$|f(z)|^2<\varepsilon$而求得一个根。

每当求得一个根 z^* 后，在 $f(z)$ 中除去因子$(z-z^*)$，再求另一个根。

重复上述过程，直到求得全部根为止。

在实际计算中，每求一个根，都要作变换$z=\sqrt[n-1]{|a_0|}z'$，使得$a_n=1$时，$|a_0|=1$，以保证寻根在单位圆内进行。

2. 算法实现

根据上述方法，可以定义牛顿-下山法的 Visual Basic 函数 NLNdhRoot，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLNdhRoot
'  功能：  使用牛顿-下山法求解实系数代数方程的全部根
'  参数：  n    - 多项式方程的次数
'          a    - Double型一维数组，长度为n+1，按降幂次序依次存放n次多项式方程的n+1
个系数
'          xr    - Double型一维数组，长度为n，返回n个根的实部
'          xi    - Double型一维数组，长度为n，返回n个根的虚部
'  返回值：Boolean型，若为False，则表示在QR方法中迭代已超过最大迭代次数但还未满足
'          精度要求；若为True，则表示求解成功。
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLNdhRoot(n As Integer, a() As Double, xr() As Double, _
xi() As Double) As Boolean
    Dim m As Integer, i As Integer, jt As Integer, k As Integer, _
         nIs As Integer, it As Integer
    Dim t As Double, x As Double, y As Double, x1 As Double, _
         y1 As Double, dx As Double, dy As Double
    Dim p As Double, q As Double, w As Double, dd As Double, _
         dc As Double, c As Double
    Dim g As Double, u As Double, v As Double, pq As Double, _
         g1 As Double, u1 As Double, v1 As Double

    ' 系数个数
    m = n + 1

    ' 用最高幂系数约化其余系数
    k = a(1)
    For i = 1 To m
       a(i) = a(i) / k
    Next i

    ' 迭代求解
    k = m
    nIs = 0
    w = 1#
    jt = 1
    While (jt = 1)
       pq = Abs(a(k))
       While (pq < 0.000000000001)
          xr(k - 1) = 0#
          xi(k - 1) = 0#
          k = k - 1

          ' 求解结束
          If (k = 2) Then
             xr(1) = -a(2) * w / a(1)
             xi(1) = 0#
             NLNdhRoot = True
             Exit Function
```

```
            End If

            pq = Abs(a(k))
        Wend

        q = Log(pq)
        q = q / (1# * k)
        q = Exp(q)
        p = q
        w = w * p
        For i = 2 To k
            a(i) = a(i) / q
            q = q * p
        Next i

        x = 0.0001
        x1 = x
        y = 0.2
        y1 = y
        dx = 1#
        g = 1E+37
l40:
        u = a(1)
        v = 0#
        For i = 2 To k
            p = u * x1
            q = v * y1
            pq = (u + v) * (x1 + y1)
            u = p - q + a(i)
            v = pq - p - q
        Next i

        g1 = u * u + v * v
        If (g1 >= g) Then
            If (nIs <> 0) Then
                it = 1
                Call g65(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
                If (it = 0) Then GoTo l40
            Else
                Call g60(t, x, y, x1, y1, dx, dy, p, q, k, it)
                If (t >= 0.001) Then GoTo l40
                If (g > 1E-18) Then
                    it = 0
                    Call g65(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
                    If (it = 0) Then GoTo l40
                End If
            End If
            Call g90(xr, xi, a, x, y, p, q, w, k)
        Else
            g = g1
            x = x1
```

```
            y = y1
            nIs = 0
            If (g <= 1E-22) Then
                Call g90(xr, xi, a, x, y, p, q, w, k)
            Else
                u1 = k * a(1)
                v1 = 0#
                For i = 3 To k
                    p = u1 * x
                    q = v1 * y
                    pq = (u1 + v1) * (x + y)
                    u1 = p - q + (k - i + 1) * a(i - 1)
                    v1 = pq - p - q
                Next i

                p = u1 * u1 + v1 * v1
                If (p <= 1E-20) Then
                    it = 0
                    Call g65(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
                    If (it = 0) Then GoTo l40
                    Call g90(xr, xi, a, x, y, p, q, w, k)
                Else
                    dx = (u * u1 + v * v1) / p
                    dy = (u1 * v - v1 * u) / p
                    t = 1# + 4# / k
                    Call g60(t, x, y, x1, y1, dx, dy, p, q, k, it)
                    If (t >= 0.001) Then GoTo l40
                    If (g > 1E-18) Then
                        it = 0
                        Call g65(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
                        If (it = 0) Then GoTo l40
                    End If
                    Call g90(xr, xi, a, x, y, p, q, w, k)
                End If
            End If
        End If

        If (k = 2) Then
            jt = 0
        Else
            jt = 1
        End If
    Wend

    ' 迭代结束
    NLNdhRoot = True

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
```

```
' 函数名：g60
' 功能： 内部函数
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g60(t As Double, x As Double, y As Double, x1 As Double, _
y1 As Double, dx As Double, dy As Double, p As Double, q As Double, _
k As Integer, it As Integer)

   it = 1
   While (it = 1)
      t = t / 1.67
      it = 0
      x1 = x - t * dx
      y1 = y - t * dy
      If (k > 50) Then
         p = Sqr(x1 * x1 + y1 * y1)
         q = Exp(85# / k)
         If (p >= q) Then it = 1
      End If
   Wend

End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' 模块名：NLModule.bas
' 函数名：g90
' 功能： 内部函数
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g90(xr() As Double, xi() As Double, a() As Double, x As Double, _
y As Double, p As Double, q As Double, w As Double, k As Integer)

   Dim i As Integer

   If (Abs(y) <= 0.000001) Then
      p = -x
      y = 0#
      q = 0#
   Else
      p = -2# * x
      q = x * x + y * y
      xr(k - 1) = x * w
      xi(k - 1) = -y * w
      k = k - 1
   End If

   For i = 2 To k
      a(i) = a(i) - a(i - 1) * p
      a(i + 1) = a(i + 1) - a(i - 1) * q
   Next i

   xr(k - 1) = x * w
   xi(k - 1) = y * w
```

```
    k = k - 1

    If (k = 2) Then
        xr(1) = -a(2) * w / a(1)
        xi(1) = 0#
    End If

 End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：g65
'  功能：  内部函数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g65(x As Double, y As Double, x1 As Double, y1 As Double, _
dx As Double, dy As Double, dd As Double, dc As Double, c As Double, _
k As Integer, nIs As Integer, it As Integer)

    If (it = 0) Then
        nIs = 1
        dd = Sqr(dx * dx + dy * dy)
        If (dd > 1#) Then dd = 1#
        dc = 6.28 / (4.5 * k)
        c = 0#
    End If

    While (True)
        c = c + dc
        dx = dd * Cos(c)
        dy = dd * Sin(c)
        x1 = x + dx
        y1 = y + dy
        If (c <= 6.29) Then
            it = 0
            Exit Sub
        End If
        dd = dd / 1.67
        If (dd <= 0.0000001) Then
            it = 1
            Exit Sub
        End If
        c = 0#
    Wend

End Sub
```

3. 示例

调用上述 NLNdhRoot **函数求** 6 **次多项式方程**

$$f(z)=z^6-5z^5+3z^4+z^3-7z^2+7z-20=0$$

的全部根。

程序代码如下：

```
Sub Main()
    Dim n As Integer
    Dim s As String

    ' 6次方程
    n = 6

    ' 分配存放系数和解的内存
    ReDim a(n + 1) As Double, xr(n) As Double, xi(n) As Double

    ' 系数数组
    a(1) = 1
    a(2) = -5
    a(3) = 3
    a(4) = 1
    a(5) = -7
    a(6) = 7
    a(7) = -20

    '求解
    If NLNdhRoot(n, a, xr, xi) Then
        s = ""
        For i = 1 To n
            s = s & "x(" & i & ") = " & xr(i) & " + (" & xi(i) & ")j" & Chr(13)
        Next i

        MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
        MsgBox "求解失败"
    End If

End Sub
```

其输出结果如图 4.6 所示。

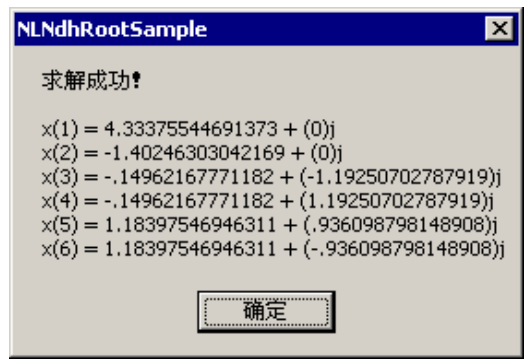

图 4.6 程序输出结果

4.7　求复系数代数方程全部根的牛顿-下山法

1. 算法原理

牛顿-下山法也是求解复系数代数方程全部根的重要方法。对复系数代数方程

$$f(z)=a_n z^n+a_{n-1}z^{n-1}+\Lambda+a_1 z+a_0=0$$

其中，a_k 为复数，可表示为

$$a_k=ar(k)+\mathrm{j}ai(k),\ k=0,\ 1,\ \Lambda\ ,\ n$$

具体的求解方法参见 4.5 节。

2. 算法实现

根据上述方法，可以定义牛顿-下山法求解复系数方程的 Visual Basic 函数 NLNdhcRoot，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLNdhcRoot
'  功能：  使用牛顿-下山法求解复系数代数方程的全部根
'  参数：  n    - 多项式方程的次数
'          ar   - Double型一维数组，长度为n+1，按降幂次序依次存放n次多项式方程的n+1
'                 个系数的实部
'          ai   - Double型一维数组，长度为n+1，按降幂次序依次存放n次多项式方程的n+1
'                 个系数的虚部
'          xr    - Double型一维数组，长度为n，返回n个根的实部
'          xi    - Double型一维数组，长度为n，返回n个根的虚部
'  返回值：Boolean型，若为False，则表示在QR方法中迭代已超过最大迭代次数但还未满足
'          精度要求；若为True，则表示求解成功。
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLNdhcRoot(n As Integer, ar() As Double, ai() As Double, _
xr() As Double, xi() As Double) As Boolean
    Dim m As Integer, i As Integer, jt As Integer, k As Integer, _
        nIs As Integer, it As Integer
    Dim t As Double, x As Double, y As Double, x1 As Double, _
        y1 As Double, dx As Double, dy As Double
    Dim p As Double, q As Double, w As Double, dd As Double, _
        dc As Double, c As Double
    Dim g As Double, u As Double, v As Double, pq As Double, _
        g1 As Double, u1 As Double, v1 As Double

    ' 系数个数
    m = n + 1

    ' 用最高幂系数约化其余系数
    p = Sqr(ar(1) * ar(1) + ai(1) * ai(1))
    For i = 1 To m
        ar(i) = ar(i) / p
```

```
            ai(i) = ai(i) / p
        Next i

        ' 迭代求解
        k = m
        nIs = 0
        w = 1#
        jt = 1

        While (jt = 1)
            pq = Sqr(ar(k) * ar(k) + ai(k) * ai(k))

            While (pq < 0.000000000001)
                xr(k - 1) = 0#
                xi(k - 1) = 0#
                k = k - 1

                ' 求解结束
                If (k = 2) Then
                    p = ar(1) * ar(1) + ai(1) * ai(1)
                    xr(1) = -w * (ar(1) * ar(2) + ai(1) * ai(2)) / p
                    xi(1) = w * (ar(2) * ai(1) - ar(1) * ai(2)) / p
                    NLNdhcRoot = True
                    Exit Function
                End If

                pq = Sqr(ar(k) * ar(k) + ai(k) * ai(k))
            Wend

            q = Log(pq)
            q = q / (1# * k)
            q = Exp(q)
            p = q
            w = w * p
            For i = 2 To k
                ar(i) = ar(i) / q
                ai(i) = ai(i) / q
                q = q * p
            Next i

            x = 0.0001
            x1 = x
            y = 0.2
            y1 = y
            dx = 1#
            g = 1E+37
    l40:
            u = ar(1)
            v = ai(1)
            For i = 2 To k
                p = u * x1
```

```
    q = v * y1
    pq = (u + v) * (x1 + y1)
    u = p - q + ar(i)
    v = pq - p - q + ai(i)
Next i

g1 = u * u + v * v
If (g1 >= g) Then
    If (nIs <> 0) Then
        it = 1
        Call g65c(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
        If (it = 0) Then GoTo l40
    Else
        Call g60c(t, x, y, x1, y1, dx, dy, p, q, k, it)
        If (t >= 0.001) Then GoTo l40
        If (g > 1E-18) Then
            it = 0
            Call g65c(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
            If (it = 0) Then GoTo l40
        End If
    End If

    Call g90c(xr, xi, ar, ai, x, y, p, w, k)
Else
    g = g1
    x = x1
    y = y1
    nIs = 0
    If (g <= 1E-22) Then
        Call g90c(xr, xi, ar, ai, x, y, p, w, k)
    Else
        u1 = k * ar(1)
        v1 = ai(1)
        For i = 3 To k
            p = u1 * x
            q = v1 * y
            pq = (u1 + v1) * (x + y)
            u1 = p - q + (k - i + 1) * ar(i - 1)
            v1 = pq - p - q + (k - i + 1) * ai(i - 1)
        Next i

        p = u1 * u1 + v1 * v1
        If (p <= 1E-20) Then
            it = 0
            Call g65c(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
            If (it = 0) Then GoTo l40
            Call g90c(xr, xi, ar, ai, x, y, p, w, k)
        Else
            dx = (u * u1 + v * v1) / p
            dy = (u1 * v - v1 * u) / p
            t = 1# + 4# / k
```

```
                Call g60c(t, x, y, x1, y1, dx, dy, p, q, k, it)
                If (t >= 0.001) Then GoTo l40
                If (g > 1E-18) Then
                   it = 0
                   Call g65c(x, y, x1, y1, dx, dy, dd, dc, c, k, nIs, it)
                   If (it = 0) Then GoTo l40
                End If
                Call g90c(xr, xi, ar, ai, x, y, p, w, k)
             End If
          End If
       End If

       If (k = 2) Then
          jt = 0
       Else
          jt = 1
       End If
    Wend

    ' 迭代结束
    NLNdhcRoot = True

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：g60c
'  功能：  内部函数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g60c(t As Double, x As Double, y As Double, x1 As Double, _
y1 As Double, dx As Double, dy As Double, p As Double, q As Double, _
k As Integer, it As Integer)

    it = 1
    While (it = 1)
       t = t / 1.67
       it = 0
       x1 = x - t * dx
       y1 = y - t * dy
       If (k >= 30) Then
          p = Sqr(x1 * x1 + y1 * y1)
          q = Exp(75# / k)
          If (p >= q) Then it = 1
       End If
    Wend

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：g90c
```

```
'  功能：  内部函数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g90c(xr() As Double, xi() As Double, ar() As Double, ai() As Double, _
x As Double, y As Double, p As Double, w As Double, k As Integer)

    Dim i As Integer

    For i = 2 To k
        ar(i) = ar(i) + ar(i - 1) * x - ai(i - 1) * y
        ai(i) = ai(i) + ar(i - 1) * y + ai(i - 1) * x
    Next i

    xr(k - 1) = x * w
    xi(k - 1) = y * w
    k = k - 1
    If (k = 2) Then
        p = ar(1) * ar(1) + ai(1) * ai(1)
        xr(1) = -w * (ar(1) * ar(2) + ai(1) * ai(2)) / p
        xi(1) = w * (ar(2) * ai(1) - ar(1) * ai(2)) / p
    End If

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：g65c
'  功能：  内部函数
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub g65c(x As Double, y As Double, x1 As Double, y1 As Double, _
dx As Double, dy As Double, dd As Double, dc As Double, c As Double, _
k As Integer, nIs As Integer, it As Integer)

    If (it = 0) Then
        nIs = 1
        dd = Sqr(dx * dx + dy * dy)
        If (dd > 1#) Then dd = 1#
        dc = 6.28 / (4.5 * k)
        c = 0#
    End If

    While (True)
        c = c + dc
        dx = dd * Cos(c)
        dy = dd * Sin(c)
        x1 = x + dx
        y1 = y + dy
        If (c <= 6.29) Then
            it = 0
            Exit Sub
        End If
        dd = dd / 1.67
```

```
        If (dd <= 0.0000001) Then
            it = 1
            Exit Sub
        End If
        c = 0#
    Wend

End Sub
```

3. 示例

调用上述 NLNdhcRoot 函数求解如下复系数代数方程的全部根：

$$f(z)=z^5+(3+3j)z^4-0.01jz^3+(4.9-19j)z^2+21.33z+(0.1-100j)=0$$

程序代码如下：

```
Sub Main()
    Dim n As Integer
    Dim s As String

    ' 5次方程
    n = 5

    ' 分配存放系数和解的内存
    ReDim ar(n + 1) As Double, ai(n + 1) As Double, xr(n) As Double, _
          xi(n) As Double

    ' 系数数组
    ar(1) = 1#
    ar(2) = 3#
    ar(3) = 0#
    ar(4) = 4.9
    ar(5) = 21.33
    ar(6) = 0.1

    ai(1) = 0#
    ai(2) = 2#
    ai(3) = -0.01
    ai(4) = -19#
    ai(5) = 0#
    ai(6) = -100#

    '求解
    If NLNdhcRoot(n, ar, ai, xr, xi) Then
        s = ""
        For i = 1 To n
            s = s & "x(" & i & ") = " & xr(i) & " + (" & xi(i) & ")j" & Chr(13)
        Next i

        MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
```

```
        MsgBox "求解失败"
End If

End Sub
```

其输出结果如图 4.7 所示。

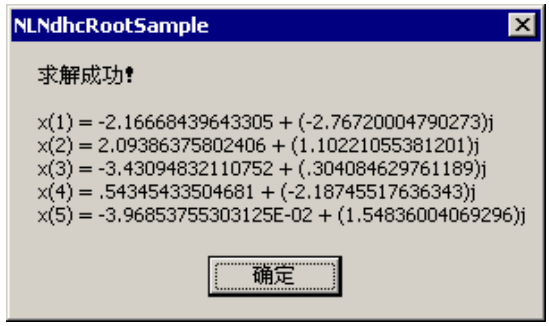

图 4.7　程序输出结果

4.8　求非线性方程组一组实根的梯度法

1. 算法原理

设非线性方程组为：

$$f_i = f_i(x_0,\ x_1, \Lambda\ ,\ x_{n-1}) = 0,\quad i = 0,\ 1,\ \Lambda\ ,\ n-1$$

并定义目标函数为：

$$F = F(x_0, x_1, \Lambda\ , x_{n-1}) = \sum_{i=0}^{n-1} f_i^{\,2}$$

则梯度法的计算过程如下：

(1)　**选取一组初值** $x_0, x_1, \Lambda\ , x_{n-1}$

(2)　**计算目标函数值**

$$F = F(x_0, x_1, \Lambda\ , x_{n-1}) = \sum_{i=0}^{n-1} f_i^{\,2}$$

(3)　**若** $F < \varepsilon$ **，则** $X = (x_0, x_1, \Lambda\ , x_{n-1})^{\mathrm{T}}$ **即为方程组的一组实根，过程结束；否则继续。**

(4)　**计算目标函数在** $(x_0, x_1, \Lambda\ , x_{n-1})$ **点的偏导数**

$$\frac{\partial F}{\partial x_i} = 2\sum_{j=0}^{n-1} f_j \cdot \frac{\partial f_j}{\partial x_i},\quad j = 0,\ 1,\ \Lambda\ ,\ n-1$$

(5)　**计算**

$$D = \sum_{j=0}^{n-1}\left(\frac{\partial F}{\partial x_j}\right)^2$$

(6) 计算

$$x_i - \lambda \frac{\partial F}{\partial x_i} \Rightarrow x_i,\ i = 0,\ 1,\ \Lambda\ ,\ n-1$$

其中 $\lambda = F/D$

(7) 重复(1)－(6)步，直到满足精度要求为止。

在上述过程中，如果 D = 0，则说明遇到了目标函数的局部极值点，此时可改变初值再重新计算。

2. 算法实现

根据上述方法,可以定义梯度法求解非线性方程组一组解的 Visual Basic 函数 NLGrad。本函数需要调用计算目标函数值的函数 Func，由用户自编，其形式为：

Function Func(x() as double) as double

其返回值为：

$$\sum_{i=0}^{n-1} f_i^{\,2}(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1}),\ i = 0,\ 1,\ \Lambda\ ,\ n-1$$

NLGrad 的代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLGrad
'  功能：  使用梯度法求解非线性方程组的一组解，本函数需要调用计算目标函数值的函数Func，
'          其形式为：
'          Function Func(x() as double) as double
'  参数:   n    - Integer型变量，方程的个数，也是未知数的个数
'          x    - Double型一维数组，长度为n，存放一组初值x1, x2, …, xn，
'                 返回时存放方程组的一组实根
'        nMaxIt - Integer型变量，最大迭代次数
'        eps    - Double型变量，精度控制参数
'          h    - Double型变量，收敛控制参数，一般取1e-5至1e-10之间
'  返回值：Boolean型，False则求解失败，True则求解成功。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLGrad(n As Integer, x() As Double, nMaxIt As Integer, _
eps As Double, h As Double) As Boolean
    Dim i As Integer, j As Integer
    Dim f As Double, fh As Double, r As Double, sum As Double
    ReDim hx(n) As Double, df(n) As Double

    For j = 1 To nMaxIt
        For i = 1 To n
            If (x(i) = 0) Then
                hx(i) = h
            Else
                hx(i) = h * x(i)
            End If
```

```
        Next i

        f = Func(x)

        If (f <= eps) Then
            NLGrad = True
            Exit Function
        End If

        sum = 0

        For i = 1 To n
            x(i) = x(i) + hx(i)
            fh = Func(x)
            df(i) = (fh - f) / hx(i)
            sum = sum + df(i) * df(i)
            x(i) = x(i) - hx(i)
        Next i

        r = f / sum

        For i = 1 To n
            x(i) = x(i) - r * df(i)
        Next i

    Next j

NLGrad = False

End Function
```

3. 示例

调用上述 NLGrad **函数求解如下方程组的一组实根：**

$$\begin{cases} x_1 - 5x_2^2 + 7x_3^2 + 12 = 0 \\ 3x_1x_2 + x_1x_3 - 11x_1 = 0 \\ 2x_2x_3 + 40x_1 = 0 \end{cases}$$

取 $\varepsilon = 1^{-10}$，$h = 1^{-10}$，**初值设为**(1.5，7.5，-6.0)。

程序代码如下：

```
Sub Main()
    Dim n As Integer, nMaxIt As Integer
    Dim s As String

    ' 3次方程
    n = 3

    ' 分配初值和解的内存
```

```
    ReDim x(n) As Double

    ' 初值数组
    x(1) = 1.5
    x(2) = 7.5
    x(3) = -6

    ' 迭代次数
    nMaxIt = 1200

    '求解
    If NLGrad(n, x, nMaxIt, 0.0000000001, 0.0000000001) Then
       s = ""
       For i = 1 To n
          s = s & "x(" & i & ") = " & x(i) & Chr(13)
       Next i

       MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
       MsgBox "求解失败"
    End If

End Sub

Function Func(x() As Double) As Double
    Func = (x(1) - 5 * x(2) * x(2) + 7 * x(3) * x(3)
+ 12) * (x(1) - _
          5 * x(2) * x(2) + 7 * x(3) * x(3) + 12)
+ (3 * x(1) * x(2) + _
          x(1) * x(3) - 11 * x(1)) * (3 * x(1) * x(2)
+ x(1) * x(3) - _
          11 * x(1)) + (2 * x(2) * x(3) + _
          40 * x(1)) * (2 * x(2) * x(3) + 40 * x(1))
End Function
```

图 4.8 程序输出结果

其输出结果如图 4.8 **所示。**

4.9 求非线性方程组一组实根的拟牛顿法

1. 算法原理

本节介绍用拟牛顿法求非线性方程组

$$f_i(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})=0,\quad i=0,\ 1,\ \Lambda\ ,\ n-1$$

的一组实数解的算法。

设非线性方程组

$$f_i(X)=0,\ i=0,\ 1,\ \Lambda\ ,\ n-1 \tag{1}$$

其中 $X=(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})^{\mathrm{T}}$ **。**

若假设 X 的第 k 次迭代近似值为

$$X^{(k)}=(x_0^{(k)},\ x_1^{(k)},\ \Lambda\ ,\ x_{n-1}^{(k)})^{\mathrm{T}}$$

则计算第 k+1 次迭代值的牛顿迭代格式为：

$$X^{(k+1)}=X^{(k)}-F(X^{(k)})^{-1}f(X^{(k)})$$

其中，

$$f(X^{(k)})=(f_0^{(k)},\ f_1^{(k)},\ \Lambda\ ,\ f_{n-1}^{(k)})^{\mathrm{T}}$$
$$f_i^{(k)}=f_i(X^{(k)})$$

$F(X)$ 为雅可比(Jacobi)矩阵，即：

$$F(X)=\begin{bmatrix}\dfrac{\partial f_0(X)}{\partial x_0} & \dfrac{\partial f_0(X)}{\partial x_1} & \Lambda & \dfrac{\partial f_0(X)}{\partial x_{n-1}}\\ \dfrac{\partial f_1(X)}{\partial x_0} & \dfrac{\partial f_1(X)}{\partial x_1} & \Lambda & \dfrac{\partial f_1(X)}{\partial x_{n-1}}\\ \mathrm{M} & \mathrm{M} & & \mathrm{M}\\ \dfrac{\partial f_{n-1}(X)}{\partial x_0} & \dfrac{\partial f_{n-1}(X)}{\partial x_1} & \Lambda & \dfrac{\partial f_{n-1}(X)}{\partial x_{n-1}}\end{bmatrix}$$

令：$\delta^{(k)}=F(X^{(k)})^{-1}f(X^{(k)})$，其中，$\delta^{(k)}=(\delta_0^{(k)},\ \delta_1^{(k)},\ \Lambda\ ,\ \delta_{n-1}^{(k)})^{\mathrm{T}}$

则有

$$F(X^{(k)})\delta^{(k)}=f(X^{(k)}) \tag{2}$$

$$X^{(k+1)}=X^{(k)}-\delta^{(k)} \tag{3}$$

若在雅可比矩阵中，用差商代替偏导数，即：

$$\frac{\partial f_i(X^{(k)})}{\partial x_j}\approx\frac{f_i(X_j^{(k)})-f_i(X^{(k)})}{h}$$

其中，h 足够小，且

$$f_i(X_j^{(k)})=f_i(x_0^{(k)},\ \Lambda\ ,\ x_{j-1}^{(k)},\ x_j^{(k)}+h,\ x_{j+1}^{(k)},\ \Lambda\ ,\ x_{n-1}^{(k)})$$

则方程组(2)变为

$$\sum_{=0}^{n-1}f_i(X^{(k)})z_j^{(k)}=f\ (X_j^{(k)}),\ i=0,\ 1,\ \Lambda\ ,\ n-1$$

其中，

$$z_j^{(k)}=\frac{\delta_j^{(k)}}{h+\sum_{s=0}^{n-1}\delta_s^{(k)}},\ j=0,\ 1,\ \Lambda\ ,\ n-1。$$

综上所述，拟牛顿法求解非线性方程组的计算过程如下：

(1) **取初值** $X=(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})^T,\ h>0,\ 0<t<1$。

(2) **计算** $f_i(X)\Rightarrow B(i),\ i=0,\ 1,\ \Lambda\ ,\ n-1$。

(3) **若满足**

$$\max_{0\le i\le n-1}\left|B(i)\right|\le\varepsilon$$

则方程组的一组实数解为：

$$X=(x_0, x_1, \cdots, x_{n-1})^{\mathrm{T}}$$

计算过程结束；否则继续。

(4) **计算**

$$f_i(X_j)\Rightarrow A(i,\ \ j),\ i,\ \ j=0,\ 1,\ \Lambda\ ,\ n-1$$

其中，

$$X=(x_0,\ x_1,\ \Lambda\ ,\ x_{j-1},\ x_j+h,\ x_{j+1}\Lambda\ ,\ x_{n-1})^{\mathrm{T}}。$$

(5) **解线性代数方程组** $AZ=B$**，其中** $Z=(z_0,\ z_1,\ \Lambda\ ,\ z_{n-1})^{\mathrm{T}}$。**且计算** $\beta=1-\sum_{j=0}^{n-1}z_j$ 。

(6) **计算** $x_i-hz_i/\beta\Rightarrow x_i,\ i=0,\ 1,\ \Lambda\ ,\ n-1$。

(7) $t*h\Rightarrow h$, **转**(1)。

以上过程一直作到 X **满足精度要求为止。**

在计算过程中，若发现线性代数方程组 $AZ=B$ **奇异或** $\beta=0$ (**即** $\sum_{j=0}^{n-1}z_j=1$)**的情况，可以采取以下一些措施试一试：**

放宽精度要求 ε **；**

适当改变 t **与** h **的初值；**

改变 X **的初值；**

改变方程组中各方程的顺序。

2. **算法实现**

根据上述方法，可以定义用拟牛顿法求非线性方程组的 Visual Basic **函数** NLNewtonA**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLNewtonA
'  功能：  使用拟牛顿法求非线性方程组
'          1)本函数要调用全选主元高斯消去法求解线性代数方程组的函数LEGauss;
'          2)本函数需要调用计算目标函数值的函数Func，其形式为：
'          Sub Func(x() as Double, y() as Double)
'          y(i) 为方程组中各左边多项式与一组x(i)对应的函数值
'  参数：  n    - Integer型变量，方程的个数，也是未知数的个数
'          x    - Double型一维数组，长度为n，存放一组初值x1, x2, …, xn,
'                   返回时存放方程组的一组实根
'         nMaxIt - Integer型变量，最大迭代次数
'         eps    - Double型变量，精度控制参数
```

```
'         h    - Double型变量，增量初值，在本函数中要被破坏
'         t    - Double型变量，控制h大小的变量，0<t<1
'  返回值：Boolean型，True则求解成功；False则求解失败。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NLNewtonA(n As Integer, x() As Double, nMaxIt As Integer, _
eps As Double, h As Double, t As Double) As Boolean
    Dim i As Integer, j As Integer, l As Integer
    Dim am As Double, z As Double, beta As Double, d As Double
    ReDim y(n) As Double, a(n, n) As Double, b(n) As Double

    ' 最大迭代次数
    l = nMaxIt

    '精度控制
    am = 1# + eps

    ' 迭代求解
    While (am >= eps)

        ' 函数值
        Call Func(x, b)

        am = 0#
        For i = 1 To n
            z = Abs(b(i))
            If (z > am) Then am = z
        Next i

        If (am >= eps) Then
            l = l - 1

            ' 达到最大迭代次数，精度未达到要求，求解失败，返回
            If (l = 0) Then
                NLNewtonA = False
                Exit Function
            End If

            For j = 1 To n
                z = x(j)
                x(j) = x(j) + h

                Call Func(x, y)

                For i = 1 To n
                    a(i, j) = y(i)
                Next i

                x(j) = z

            Next j
```

```
            ' 高斯消元失败，求解失败，返回
            If Not LEGauss(n, a, b) Then
               NLNewtonA = False
               Exit Function
            End If

            beta = 1#
            For i = 1 To n
               beta = beta - b(i)
            Next i

            ' 方程求解失败，返回
            If (Abs(beta) + 1# = 1#) Then
               NLNewtonA = False
               Exit Function
            End If

            d = h / beta
            For i = 1 To n
               x(i) = x(i) - d * b(i)
            Next i

            h = t * h

         End If
      Wend

      ' 方程求解成功，返回
      NLNewtonA = True

End Function
```

3. 示例

调用上述 NLNewtonA **函数求解如下方程组的一组实根：**

$$\begin{cases} f_0 = x_0^2 + x_1^2 + x_2^2 - 1 = 0 \\ f_1 = 2x_0^2 + x_1^2 - 4x_2 = 0 \\ f_2 = 3x_0^2 - 4x_1 + x_2^2 = 0 \end{cases}$$

取初值为 $(1.0,\ 1.0,\ 1.0)^{\mathrm{T}}$，t=0.1，h=0.1，$\varepsilon$ =0.0000001, **最大迭代次数** k=100。

```
Sub Main()
   Dim n As Integer, nMaxIt As Integer
   Dim t As Double, h As Double
   Dim s As String

   ' 3次方程
   n = 3

   ' 分配初值和解的内存
```

```
    ReDim x(n) As Double

    ' 初值数组
    x(1) = 1
    x(2) = 1
    x(3) = 1

    ' 最大迭代次数
    nMaxIt = 100

    ' 增量及控制参数
    t = 0.1
    h = 0.1

    '求解
    If NLNewtonA(n, x, nMaxIt, 0.0000001, h, t) Then
        s = ""
        For i = 1 To n
            s = s & "x(" & i & ") = " & x(i) & Chr(13)
        Next i

        MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
        MsgBox "求解失败"
    End If

End Sub

Sub Func(x() As Double, y() As Double)
    y(1) = x(1) * x(1) + x(2) * x(2) + x(3) * x(3)
- 1#
    y(2) = 2# * x(1) * x(1) + x(2) * x(2) - 4# * x(3)
    y(3) = 3# * x(1) * x(1) - 4# * x(2) + x(3) * x(3)
End Sub
```

图 4.9　程序输出结果

其输出结果如图 4.9 所示。

4.10　求非线性方程组最小二乘解的广义逆法

1. 算法原理

利用广义逆求解无约束条件下的优化问题

$$f_i(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1}) = 0,\ \ i = 0,\ 1,\ \Lambda\ ;\ \ m-1,\ \ m \geq n$$

当 $m=n$ 时，即为求解非线性方程组。

设非线性方程组为：

$$f_i(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1}) = 0,\ \ i = 0,\ 1,\ \Lambda\ ;\ \ m-1,\ \ m \geq n$$

其雅可比(Jacobi)矩阵为：

$$F(X)=\begin{bmatrix} \dfrac{\partial f_0}{\partial x_0} & \dfrac{\partial f_0}{\partial x_1} & \Lambda & \dfrac{\partial f_0}{\partial x_{n-1}} \\ \dfrac{\partial f_1}{\partial x_0} & \dfrac{\partial f_1}{\partial x_1} & \Lambda & \dfrac{\partial f_1}{\partial x_{n-1}} \\ \mathrm{M} & \mathrm{M} & & \mathrm{M} \\ \dfrac{\partial f_{m-1}}{\partial x_0} & \dfrac{\partial f_{m-1}}{\partial x_1} & \Lambda & \dfrac{\partial f_{m-1}}{\partial x_{n-1}} \end{bmatrix}$$

计算非线性方程组最小二乘解的迭代公式为

$$X^{(k+1)}=X^{(k)}-\alpha_k Z^{(k)}$$

其中，$Z^{(k)}$ 为线性代数方程组 $A^{(k)}Z^{(k)}=F^{(k)}$ 的线性最小二乘解，即：

$$Z^{(k)}=(A^{(k)})+F^{(k)}$$

式中 $A^{(k)}$ 为 k 次迭代值 $X^{(k)}$ 的雅可比矩阵；$F^{(k)}$ 为 k 次迭代值的左端函数值，即

$$F^{(k)}=(f_0^{(k)},\ f_1^{(k)},\ \Lambda,\ f_{m-1}^{(k)})^{\mathrm{T}}$$

$$f_i^{(k)}=f_i(x_0^{(k)},\ x_1^{(k)},\ \Lambda,\ x_{m\backslash n-1}^{(k)}),\ i=0,\ 1,\ \Lambda,\ m-1$$

α_k 为使 α 的一元函数 $\sum_{i=0}^{m-1}(f_i^{(k+1)})^2$ 达到极小值的点。在本算法中利用有理极值法计算 α_k。

2. 算法实现

根据上述方法，可以定义用最小二乘解的广义逆法求非线性方程组的 Visual Basic 函数 NLMiv，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLMiv
'  功能：  用最小二乘解的广义逆法求非线性方程组
'          1)本函数要调用求解线性最小乘问题的广义逆法的函数LEMiv；
'          2)本函数需要调用计算目标函数值的函数Func，其形式为：
'             Sub Func(x() as Double, y() as Double)
'             y(i) 为方程组中各左边多项式与一组x(i)对应的函数值
'          3)本函数需要调用计算雅可比矩阵函数，其形式为：
'             Sub FuncMJ(x() as Double, p() as Double)
'           p(i, j) 为方程组中各左边多项式fi的对变元xj的偏导数与一组x(i)对应的函数值
'  参数：  m   - Integer型变量，非线性方程组中方程的个数
'          n   - Integer型变量，非线性方程组中未知数的个数
'          x   - Double型一维数组，长度为n，存放一组初值x1, x2, …, xn，要求不全为0，
'                返回时存放方程组的最小二乘解，当m=n时，即是非线性方程组的解
'         eps1   - Double型变量，最小二乘解的精度控制参数
'         eps2   - Double型变量，奇异值分解的精度控制参数
'         ka     - Integer型变量，ka=max(m, n)+1
'  返回值：Boolean型，True则求解成功；False则求解失败。
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Function NLMiv(m As Integer, n As Integer, x() As Double, _
eps1 As Double, eps2 As Double, ka As Integer) As Boolean
    Dim i As Integer, j As Integer, k As Integer, l As Integer, kk As Integer
    Dim jt As Boolean
    Dim y(10) As Double, b(10) As Double, alpha As Double, z As Double, _
         h2 As Double, y1 As Double, y2 As Double
    Dim y3 As Double, y0 As Double, h1 As Double

    ' 分配内存
    ReDim p(m, n) As Double, d(m) As Double, pp(n, m) As Double, _
         dx(n) As Double, u(m, m) As Double
    ReDim v(n, n) As Double, w(ka) As Double

    ' 迭代次数设为60
    l = 60

    ' 控制初值
    alpha = 1#

    While (l > 0)
        ' 调用计算目标函数值的函数
        Call Func(x, d)

        ' 调用计算雅可比矩阵函数
        Call FuncMJ(x, p)

        ' 调用求解线性最小乘问题的广义逆法的函数LEMiv
        jt = LEMiv(m, n, p, d, dx, pp, u, v, ka, eps2)

        ' 求解失败，返回
        If (jt = False) Then
            NLMiv = False
            Exit Function
        End If

        ' 继续迭代计算
        j = 0
        jt = True
        h2 = 0#

        While (jt)
            jt = False
            If (j <= 2) Then
                z = alpha + 0.01 * j
            Else
                z = h2
            End If

            For i = 1 To n
                w(i) = x(i) - z * dx(i)
            Next i
```

```
Call Func(w, d)

y1 = 0#

For i = 1 To m
   y1 = y1 + d(i) * d(i)
Next i

For i = 1 To n
  w(i) = x(i) - (z + 0.00001) * dx(i)
Next i

Call Func(w, d)

y2 = 0#
For i = 1 To m
   y2 = y2 + d(i) * d(i)
Next i

y0 = (y2 - y1) / 0.00001

If (Abs(y0) > 0.0000000001) Then
   h1 = y0
   h2 = z
   If (j = 0) Then
      y(1) = h1
      b(1) = h2
   Else
      y(j + 1) = h1
      kk = 0
      k = 0
      While ((kk = 0) And (k <= j - 1))
         y3 = h2 - b(k + 1)
         If (Abs(y3) + 1# = 1#) Then
            kk = 1
         Else
            h2 = (h1 - y(k + 1)) / y3
         End If

         k = k + 1
      Wend

      b(j + 1) = h2
      If (kk <> 0) Then b(j + 1) = 1E+35
      h2 = 0#
      For k = j - 1 To 0 Step -1
        h2 = -y(k + 1) / (b(k + 2) + h2)
      Next k

      h2 = h2 + b(1)
```

```
                End If
                j = j + 1
                If (j <= 7) Then
                   jt = True
                Else
                   z = h2
                End If
            End If
        Wend

        alpha = z
        y1 = 0#
        y2 = 0#
        For i = 1 To n
            dx(i) = -alpha * dx(i)
            x(i) = x(i) + dx(i)
            y1 = y1 + Abs(dx(i))
            y2 = y2 + Abs(x(i))
        Next i

        ' 达到精度要求，求解成功，返回
        If (y1 < eps1 * y2) Then
            NLMiv = True
            Exit Function
        End If

        l = l - 1
    Wend

    ' 迭代60次后仍未达到精度要求，求解失败，返回
    NLMiv = False

End Function
```

3. **示例**

调用上述 NLMiv **函数求解**

$$\begin{cases} x_0^2 + 7x_0x_1 + 3x_1^2 + 0.5 = 0 \\ x_0^2 - 2x_0x_1 + x_1^2 - 1 = 0 \\ x_0 + x_1 + 1 = 0 \end{cases}$$

的最小二乘解。其中 m=3，n=2，**取初值**(1.0，-1.0)，$\varepsilon 1 = \varepsilon 2$=0.000001。

程序代码如下：

```
Sub Main()
    Dim n As Integer, m As Integer
    Dim s As String

    ' 3个未知数
```

```
    m = 3
    ' 2次方程
    n = 2

    ' 分配初值和解的内存
    ReDim x(n) As Double

    ' 初值数组
    x(1) = 1
    x(2) = -1

    '求解
    If NLMiv(m, n, x, 0.000001, 0.000001, 4) Then
       s = ""
       For i = 1 To n
          s = s & "x(" & i & ") = " & x(i) & Chr(13)
       Next i

       MsgBox "求解成功！" & Chr(13) & Chr(13) & s
    Else
       MsgBox "求解失败"
    End If

End Sub

Sub Func(x() As Double, y() As Double)
    y(1) = x(1) * x(1) + 7# * x(1) * x(2) + 3# * x(2) * x(2) + 0.5
    y(2) = x(1) * x(1) - 2# * x(1) * x(2) + x(2) * x(2) - 1#
    y(3) = x(1) + x(2) + 1#
End Sub

Sub FuncMJ(x() As Double, p() As Double)
    p(1, 1) = 2# * x(1) + 7# * x(2)
    p(1, 2) = 7# * x(1) + 6# * x(2)
    p(2, 1) = 2# * x(1) - 2# * x(2)
    p(2, 2) = -2# * x(1) + 2# * x(2)
    p(3, 1) = 1#
    p(3, 2) = 1#
End Sub
```

图 4.10　程序输出结果

其输出结果如图 4.10 **所示。**

4.11　求非线性方程一个实根的蒙特卡洛法

1. **算法原理**

本节介绍用蒙特卡洛(Monte Carlo)法求非线性方程 $f(x)=0$ **的一个实根的算法。**

设实函数方程为：

$$f(x)=0$$

用蒙特卡洛法求一个实根的过程如下：

(1) 选取一个初值 x，并计算 $F_0 = f(x)$。

(2) 选取一个 $b>0$，在区间[$-b$，b]上反复产生均匀分布的随机数 r，对于每一个 r 计算 $F_1 = f(x+r)$，直到发现一个 r 使 $|F_1| < |F_0|$ 为止，此时 $x+r \Rightarrow x,\ F_1 \Rightarrow F_0$。如果连续产生 m 个随机数 r 还不满足 $|F_1| < |F_0|$，则将 b 减半再进行。

(3) 重复上述过程，直到 $|F_0| < \varepsilon$ 为止，此时的 x 即为非线性方程 $f(x)=0$ 的一个实根。

在使用本方法时，如遇到迭代不收敛，则可以适当调整 b 与其调节参数 m 的值。

2. 算法实现

根据上述方法，可以定义用蒙特卡洛法求非线性方程一个实根的 Visual Basic 函数 NLMtclRoot，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLMtclRoot
'  功能：  用蒙特卡洛法求非线性方程一个实根，本函数需要调用计算方程左端函数f(x)值的
'          函数Func，其形式为：
'          Function Func(x As Double) As Double
'  参数：  x    - Double型变量，存放初值，返回时存放方程的一个实根
'          b    - Double型变量，均匀分布随机数的端点初值
'          m    - Integer型变量，控制调节b的参数
'         eps   - Double型变量，精度控制参数
'  返回值：无
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub NLMtclRoot(x As Double, b As Double, m As Integer, eps As Double)
    Dim k As Integer
    Dim xx As Double, a As Double, y As Double, x1 As Double, y1 As Double

    ' 初值
    a = b
    k = 1
    xx = x

    ' 函数值
    y = Func(xx)

    ' 迭代求解
    While (a >= eps)

        ' 随机数
        Call Randomize(1)
        x1 = Rnd()

        x1 = -a + 2# * a * x1
        x1 = xx + x1
        y1 = Func(x1)

        k = k + 1
```

```
            If (Abs(y1) >= Abs(y)) Then
                If (k > m) Then
                    k = 1
                    a = a / 2#
                End If
            Else
                k = 1
                xx = x1
                y = y1

                ' 精度达到要求，求解结束
                If (Abs(y) < eps) Then
                    x = xx
                    Exit Sub
                End If
            End If
        Wend

        ' 迭代求解结束
        x = xx

    End Sub
```

3. 示例

调用上述 NLMtclRoot **函数求实函数方程**

$$f(x)=\mathrm{e}^{-x^3}-\frac{\sin x}{\cos x}+800=0$$

在区间 $(0,\ \frac{\pi}{2})$ **内的一个实根。取初值** $x=0.5,\ b=1.0,\ m=10,\ \varepsilon=0.000001$ **。**

程序代码如下：

```
Sub Main()
    Dim b As Double
    Dim m As Integer
    Dim x As Double

    ' 迭代初值
    x = 0.5

    ' 随机数初始值与控制参数
    b = 1#
    m = 10


    '求解
    Call NLMtclRoot(x, b, m, 0.00001)
```

```
    MsgBox "x = " & x

End Sub

' 待求解的方程的函数
Function Func(x As Double) As Double
    Func = Exp(-x * x * x) - Sin(x) / Cos(x) + 800#
End Function
```

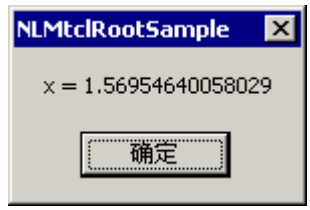

图 4.11　程序输出结果

其输出结果如图 4.11 所示。

4.12　求实函数或复函数方程的一个复根的蒙特卡洛法

1. 算法原理

蒙特卡洛(Monte Carlo)法是求实函数或复函数方程 $f(z)=0$ 的一个复根的重要方法。设非线性方程为：

$$f(z)=0$$

左端函数 $f(z)$ 的模函数记为 $\|f(z)\|$。

用蒙特卡洛法求一个复根的过程如下：

(1) 选取一个初值 $z=x+\mathrm{j}y$，其中 $\mathrm{j}=\sqrt{-1}$。并计算 $F_0=\|f(z)\|$。

(2) 再选取一个 $b>0$，在区间[$-b$，b]上反复产生均匀分布的随机数 r_x 与 r_y，对于每一对 $(r_x,\ r_y)$ 计算

$$F_1=\|f(x+r_x+\mathrm{j}(y+r_y))\|$$

直到发现一对 $(r_x,\ r_y)$ 使 $F_1<F_0$ 为止，此时

$$x+r_x+\mathrm{j}(y+r_y)\Rightarrow z,\ F_1\Rightarrow F_0$$

如果连续产生了 m 对随机数 $(r_x,\ r_y)$ 还不满足 $F_1<F_0$，则将 b 减半再进行。

(3) 重复上述过程，直到 $F_0<\varepsilon$ 为止，此时的 z 即为 $f(z)=0$ 的一个复根。

使用本方法时，如遇到迭代不收敛，则可以适当调整 b 和其控制参数 m 的值。

本方法可用于求只包含两个未知量的非线性方程组

$$\begin{cases} f_1(x,\ y)=0 \\ f_2(x,\ y)=0 \end{cases}$$

的一组实根。此时将 x 作为实部，y 当作虚部。

2. 算法实现

根据上述方法，可以定义用蒙特卡洛法求非线性方程一个实根的 Visual Basic 函数

NLMtclcRoot，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLMtclcRoot
'  功能：  用蒙特卡洛法求非线性方程一个复根，本函数需要调用计算方程左端函数f(x)值的
'          函数Func，其形式为：
'          Function Func(x As Double, y As Double) As Double
'          其返回值为||f(x+jy)||
'  参数：  x    - Double型变量，存放初值的实部，返回时存放方程的一个复根的实部
'          y    - Double型变量，存放初值的虚部，返回时存放方程的一个复根的虚部
'          b    - Double型变量，均匀分布随机数的端点初值
'          m   - Integer型变量，控制调节b的参数
'        eps   - Double型变量，精度控制参数
'  返回值：无
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub NLMtclcRoot(x As Double, y As Double, b As Double, m As Integer, _
eps As Double)
    Dim k As Integer
    Dim xx As Double, yy As Double, a As Double, z As Double
    Dim x1 As Double, y1 As Double, z1 As Double

    ' 初值
    a = b
    k = 1
    xx = x
    yy = y

    ' 函数值
    z = Func(xx, yy)

    ' 迭代求解
    While (a >= eps)

        ' 随机数
        Call Randomize(1)
        x1 = -a + 2# * a * Rnd()

        x1 = xx + x1

        ' 随机数
        Call Randomize(1)
        y1 = -a + 2# * a * Rnd()

        y1 = yy + y1
        z1 = Func(x1, y1)
        k = k + 1
        If (z1 >= z) Then
            If (k > m) Then
                k = 1
                a = a / 2#
```

```
            End If
        Else
            k = 1
            xx = x1
            yy = y1
            z = z1

            ' 精度达到要求，求解结束
            If (z < eps) Then
                x = xx
                y = yy
                Exit Sub
            End If
        End If
    Wend

    ' 迭代求解结束
    x = xx
    y = yy

End Sub
```

3. 示例

调用上述 NLMtclcRoot **函数求实函数方程**

$$f(z)=z^2-6z+13=0$$

的一个复根。取初值 $z=0.5+0.5\mathrm{j},\ b=1.0,\ m=10,\ \varepsilon=0.00001$**，其中，**

$$f(z)=(x^2-y^2-6x+13)+\mathrm{j}(2xy-6y)$$

程序代码如下：

```
Sub Main()
    Dim b As Double
    Dim m As Integer
    Dim x As Double, y As Double

    ' 迭代初值
    x = 0.5
    y = 0.5

    ' 随机数初始值与控制参数
    b = 1#
    m = 10

    '求解
    Call NLMtclcRoot(x, y, b, m, 0.00001)

    MsgBox "x = " & x & " + (" & y & ")j"
```

```
End Sub

' 待求解的方程的函数
Function Func(x As Double, y As Double) As Double
    Dim u As Double, v As Double
    u = x * x - y * y - 6# * x + 13#
    v = 2# * x * y - 6# * y
    Func = Sqr(u * u + v * v)
End Function
```

其输出结果如图 4.12 所示。

图 4.12 程序输出结果

4.13 求非线性方程组一组实根的蒙特卡洛法

1. 算法原理

本节介绍用蒙特卡洛(Monte Carlo)法求非线性方程组

$$f_i(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})=0,\ \ i=0,\ 1,\ \Lambda\ ,\ n-1$$

的一组实根。

设非线性方程为：

$$f_i(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})=0,\ \ i=0,\ 1,\ \Lambda\ ,\ n-1$$

定义模函数为：

$$\|F\|=\sqrt{\sum_{i=0}^{n-1}f_i^2}$$

用蒙特卡洛法求一组实根的过程如下：

(1) 选取一个初值 $X=(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})^{\mathrm{T}}$，并计算模函数 $F_0=\|F\|$。

(2) 再选取一个 $b>0$，在区间[$-b$，b]上反复产生均匀分布的随机数 $(r_0,\ r_1,\ \Lambda\ ,\ r_{n-1})$，对于每一组 $(r_0,\ r_1,\ \Lambda\ ,\ r_{n-1})$ 计算 $X=(x_0+r_0,\ x_1+r_1,\ \Lambda\ ,\ x_{n-1}+r_{n-1})^{\mathrm{T}}$ 的模函数值 F_1，直到发现一组使 $F_1<F_0$ 为止，此时令

$$x_i+r_i\Rightarrow x_i,\ \ i=0,\ 1,\ \Lambda\ ,\ n-1$$
$$F_1\Rightarrow F_0$$

如果连续产生了 m 组随机数还不满足 $F_1<F_0$，则将 b 减半再进行。

(3) 重复上述过程，直到 $F_0<\varepsilon$ 为止，此时的 $X=(x_0,\ x_1,\ \Lambda\ ,\ x_{n-1})^{\mathrm{T}}$ 即为非线性方程

组的一组实根。

使用本方法时，如遇到迭代不收敛，则可以适当调整 b 和其控制参数 m 的值。

2. 算法实现

根据上述方法，可以定义用蒙特卡洛法求非线性方程组一组实根的 Visual Basic 函数 NLMtcl，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NLModule.bas
'  函数名：NLMtcl
'  功能：  用蒙特卡洛法求非线性方程组一组实根，本函数需要调用计算方程组模函数值||F||的
'          函数Func，其形式为：
'          Function Func(x As Double) As Double
'          其返回值为Sqr(f1*f1 + f2*f2 + … + fn*fn)
'  参数：  n   - Integer型变量，方程的个数，也是未知数的个数
'          x    - Double型一维数组，长度为n，存放一组初值，返回时存放方程的一组实根
'          b    - Double型变量，均匀分布随机数的端点初值
'          m   - Integer型变量，控制调节b的参数
'         eps   - Double型变量，精度控制参数
'  返回值：无
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub NLMtcl(n As Integer, x() As Double, b As Double, m As Integer, _
eps As Double)
    Dim k As Integer, i As Integer
    Dim a As Double, z As Double, z1 As Double
    ReDim y(n) As Double

    ' 初值
    a = b
    k = 1

    ' 函数值
    z = Func(x)

    ' 迭代求解
    While (a >= eps)
        For i = 1 To n
            ' 随机数
            Call Randomize(1)
            y(i) = -a + 2# * a * Rnd() + x(i)
        Next i

        z1 = Func(y)

        k = k + 1

        If (z1 >= z) Then
            If (k > m) Then
                k = 1
                a = a / 2#
```

```
            End If
        Else
            k = 1
            For i = 1 To n
                x(i) = y(i)
            Next i

            z = z1

            ' 精度达到要求，求解结束
            If (z < eps) Then Exit Sub

        End If
    Wend

End Sub
```

3. 示例

调用上述 NLMtcl **函数求非线性方程组**

$$\begin{cases} 3x_1 + x_2 2x_3^2 - 3 = 0 \\ -3x_1 + 5x_2^2 + 2x_1 x_3 - 1 = 0 \\ 25x_1 x_2 + 20x_3 + 12 = 0 \end{cases}$$

的一组实根。取初值 $X = (0,\ 0,\ 0)^{\mathrm{T}}$，$b = 2.0$，$m = 10$，$\varepsilon = 0.00001$。

程序代码如下：

```
Sub Main()
    Dim b As Double
    Dim m As Integer, n As Integer
    Dim y As Double
    Dim s As String

    ' 3元方程组
    n = 3

    ' 分配内存
    ReDim x(n) As Double

    ' 迭代初值
    x(1) = 0
    x(2) = 0
    x(3) = 0

    ' 随机数初始值与控制参数
    b = 2#
    m = 10
```

```
    '求解
    Call NLMtcl(n, x, b, m, 0.00001)

    For i = 1 To n
       s = s & "x(" & i & ") = " & x(i) & Chr(13)
    Next i

    MsgBox s

End Sub

' 待求解的方程组的函数
Function Func(x() As Double) As Double
    Dim f1 As Double, f2 As Double, f3 As Double

    f1 = 3# * x(1) + x(2) + 2# * x(3) * x(3) - 3#
    f2 = -3# * x(1) + 5# * x(2) * x(2) + 2# * x(1)
* x(3) - 1#
    f3 = 25# * x(1) * x(2) + 20# * x(3) + 12#

Func = Sqr(f1 * f1 + f2 * f2 + f3 * f3)
End Function
```

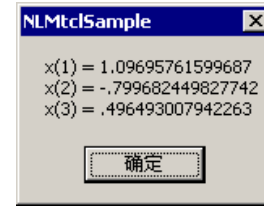

图 4.13　程序输出结果

其输出结果如图 4.13 所示。

第 5 章　插　　值

插值是确定某个函数在两个采样值之间的数值时采用的运算过程。插值在离散数据之间补充一些数据，使这组离散数据能够符合某个连续函数。插值是计算数学中最基本和最常用的手段，是函数逼近理论中的重要方法。利用它，可通过函数在有限个点处的取值状况，估算该函数在别处的值，即通过有限的数据，得出完整的数学描述。

插值是实际工作中应用最广泛的方法之一。本章我们将介绍常用的插值算法，包括一元全区间不等距插值、一元全区间等距插值、一元三点不等距插值、一元三点等距插值、连分式不等距插值、连分式等距插值、埃尔米特不等距插值、埃尔米特等距插值、埃特金不等距逐步插值、埃特金等距逐步插值、光滑不等距插值、光滑等距插值、第 1 种边界条件的三次样条函数插值、第 2 种边界条件的三次样条函数插值、第 3 种边界条件的三次样条函数插值、二元三点插值和二元全区间插值等。

5.1　一元全区间不等距插值

1. 算法原理

一元全区间不等距插值是指给定 n 个不等距结点 $x_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i=f(x_i)$，用拉格朗日(Lagrange)插值公式，计算指定插值点 t 处的函数近似值 $z=f(t)$。

为了避免龙格(Runge)现象对计算结果的影响，在 n 个结点中自动选择 8 个结点进行插值，且使指定插值点 t 位于它们的中间。即选取满足 $x_k<x_{k+1}<x_{k+2}<x_{k+3}<t<x_{k+4}<x_{k+5}<x_{k+6}<x_{k+7}$ 的 8 个结点，利用 7 次拉格朗日插值多项式计算插值点 t 处的函数近似值 $z=f(t)$，即

$$z=\sum_{i=k}^{k+7} y_i \prod_{\substack{j=k\\ j\neq i}}^{k+7}\left[(t-x_j)/(x_i-x_j)\right]$$

当插值点 t 位于包含 n 个结点的区间外时，仅取区间某端的 4 个结点进行插值；而当插值点 t 靠近 n 个结点的两端时，选取的结点数将少于 8 个。

2. 算法实现

根据上述算法，可以定义一元全区间不等距插值的 Visual Basic 函数 INLagrn，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INLagrn
'  功能：  用拉格朗日插值公式进行一元全区间不等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)，
```

```
'                要求x(1)<x(2)<...<x(n)
'         y     - Double型一维数组，长度为n，存放给定的n个结点的函数值y(i)，
'                y(i) = f(x(i)), i=1,2,...,n
'         t   - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INLagrn(n As Integer, x() As Double, y() As Double, _
t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer
    Dim z As Double, s As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INLagrn = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INLagrn = z
        Exit Function
    End If

    If (n = 2) Then
        z = (y(1) * (t - x(2)) - y(2) * (t - x(1))) / (x(1) - x(2))
        INLagrn = z
        Exit Function
    End If

    ' 开始插值
    i = 0
    While ((x(i) < t) And (i <= n))
        i = i + 1
    Wend

    k = i - 4
    If (k < 0) Then k = 0

    m = i + 3
    If (m > n - 1) Then m = n - 1

    For i = k To m
        s = 1#
        For j = k To m
            ' 拉格朗日插值公式
            If (j <> i) Then s = s * (t - x(j + 1)) / (x(i + 1) - x(j + 1))
        Next j
```

```
        z = z + s * y(i + 1)
    Next i

    ' 返回结果
    INLagrn = z

End Function
```

3. 示例

已知一元不等距列表函数如下：

i	1	2	3	4	5
x_i	0.10	0.15	0.25	0.40	0.50
y_i	0.904837	0.860708	0.778801	0.670320	0.606531

I	6	7	8	9	10
x_i	0.57	0.70	0.85	0.93	1.00
y_i	0.565525	0.496585	0.427415	0.394554	0.367879

调用函数 INLagrn **计算插值点** t=0.63 **处的函数近似值，程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, x(10) As Double, y(10) As Double

    x(1) = 0.1
    x(2) = 0.15
    x(3) = 0.25
    x(4) = 0.4
    x(5) = 0.5
    x(6) = 0.57
    x(7) = 0.7
    x(8) = 0.85
    x(9) = 0.93
    x(10) = 1#

    y(1) = 0.904837
    y(2) = 0.860708
    y(3) = 0.778801
    y(4) = 0.67032
    y(5) = 0.606531
    y(6) = 0.565525
    y(7) = 0.496585
    y(8) = 0.427415
    y(9) = 0.394554
    y(10) = 0.367879

    ' 插值位置
    t = 0.63
```

```
    ' 插值
    z = INLagrn(10, x, y, t)

    MsgBox "f(0.63) = " & z

End Sub
```

图 5.1　程序输出结果

其输出结果如图 5.1 所示。

5.2　一元全区间等距插值

1. 算法原理

本节介绍一元全区间等距插值，即给定 n 个等距结点 $x_i = x_0 + ih(i = 0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i = f(x_i)$，用拉格朗日(Lagrange)插值公式，计算指定插值点 t 处的函数近似值 $z = f(t)$。其算法原理同 5.1 节。

2. 算法实现

根据上述算法，可以定义一元全区间等距插值的 Visual Basic 函数 INEdLagrn，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdLagrn
'  功能：  用拉格朗日插值公式进行一元全区间等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          t   - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INEdLagrn(n As Integer, h As Double, x0 As Double, _
y() As Double, t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer
    Dim z As Double, s As Double, xi As Double, xj As Double
    Dim p As Double, q As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INEdLagrn = z
        Exit Function
    End If
```

```
    If (n = 1) Then
       z = y(1)
       INEdLagrn = z
       Exit Function
    End If

    If (n = 2) Then
       z = (y(2) * (t - x0) - y(1) * (t - x0 - h)) / h
       INEdLagrn = z
       Exit Function
    End If

    ' 开始插值
    If (t > x0) Then
       p = (t - x0) / h
       i = Int(p)
       q = i
       If (p > q) Then i = i + 1
    Else
       i = 0
    End If

    k = i - 4
    If (k < 0) Then k = 0

    m = i + 3
    If (m > n - 1) Then m = n - 1

    For i = k To m
       s = 1#
       xi = x0 + i * h
       For j = k To m
          If (j <> i) Then
             xj = x0 + j * h
             ' 拉格朗日插值公式
             s = s * (t - xj) / (xi - xj)
          End If
       Next j
       z = z + s * y(i + 1)
    Next i

    ' 返回结果
    INEdLagrn = z

End Function
```

3. 示例

设函数 $y=e^{-x}$ 的列表如下：

i	1	2	3	4	5
x_i	0.1	0.2	0.3	0.4	0.5
y_i	0.904837	0.818731	0.740818	0.670320	0.606531

I	6	7	8	9	10
x_i	0.6	0.7	0.8	0.9	1.0
y_i	0.548812	0.496585	0.449329	0.406570	0.367879

调用函数 INEdLagrn **来计算在插值点** t=0.25，0.63，0.95 **处的函数近似值，取** x_0=0.1，h=0.1，n=10。**程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, x0 As Double, h As Double, y(10) As Double
    Dim s As String

    ' 第一个结点
    x0 = 0.1

    ' 步长
    h = 0.1

    ' 结点的函数值
    y(1) = 0.904837
    y(2) = 0.818731
    y(3) = 0.740818
    y(4) = 0.67032
    y(5) = 0.606531
    y(6) = 0.548812
    y(7) = 0.496585
    y(8) = 0.449329
    y(9) = 0.40657
    y(10) = 0.367879

    ' 插值位置
    t = 0.25
    ' 插值
    z = INEdLagrn(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.63
    ' 插值
    z = INEdLagrn(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.95
```

```
    ' 插值
    z = INEdLagrn(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z

    MsgBox s

End Sub
```

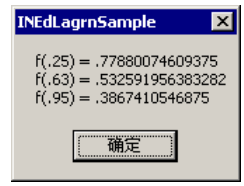

图 5.2 程序输出结果

其输出结果如图 5.2 所示。

5.3 一元三点不等距插值

1. 算法原理

一元三点不等距插值是指给定 n 个不等距结点 $x_i\,(i=0,\ 1,\ 2,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i=f(x_i)$，用抛物插值公式，计算指定插值点 t 处的函数近似值 $z=f(t)$。

设 n 个不等距结点为 $x_0<x_1<\Lambda\ <x_{n-1}$，其相应的函数值为 $y_i\,(i=0,\ 1,\ 2,\ \Lambda\ ,\ n-1)$。

为了计算指定插值点 t 处的函数近似值，选取最靠近插值点 t 的 3 个结点，即如果 $x_k<t<x_{k+1}$，则当 $|x_k-t|>|t-x_{k+1}|$ 时，取 3 个结点 x_k，x_{k+1}，x_{k+2}，而当 $|x_k-t|<|t-x_{k+1}|$ 时，取 3 个结点 x_{k-1}，x_k，x_{k+1}，然后用抛物插值公式计算 t 点的函数近似值，即：

$$z=\sum_{i=m}^{m+2}y_i\prod_{\substack{j=m\\ j\neq i}}^{m+2}\left[(t-x_j)/(x_i-x_j)\right]$$

其中：

当 $|x_k-t|>|t-x_{k+1}|$ 时，$m=k$；

当 $|x_k-t|<|t-x_{k+1}|$ 时，$m=k-1$。

如果插值点 t 位于包含 n 个结点的区间外，则取区间某端的两个结点作线性插值。

2. 算法实现

根据上述算法，可以定义一元三点不等距插值的 Visual Basic 函数 INLagrn3，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INLagrn3
'  功能：  进行一元三点不等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          t     - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Function INLagrn3(n As Integer, x() As Double, y() As Double, _
t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer
    Dim z As Double, s As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INLagrn3 = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INLagrn3 = z
        Exit Function
    End If

    If (n = 2) Then
        z = (y(1) * (t - x(2)) - y(2) * (t - x(1))) / (x(1) - x(2))
        INLagrn3 = z
        Exit Function
    End If

    ' 开始插值
    If (t <= x(2)) Then
        k = 0
        m = 2
    Else
        If (t >= x(n - 1)) Then
            k = n - 3
            m = n - 1
        Else
            k = 1
            m = n

            While (m - k <> 1)
                i = (k + m) / 2
                If (t < x(i)) Then
                    m = i
                Else
                    k = i
                End If
            Wend

            k = k - 1
            m = m - 1
            If (Abs(t - x(k + 1)) < Abs(t - x(m + 1))) Then
                k = k - 1
```

```
            Else
                m = m + 1
            End If
        End If
    End If

    z = 0#
    For i = k To m
        s = 1#
        For j = k To m
            If (j <> i) Then
                ' 抛物线插值公式
                s = s * (t - x(j + 1)) / (x(i + 1) - x(j + 1))
            End If
        Next j

        z = z + s * y(i + 1)
    Next i

    ' 返回结果
    INLagrn3 = z

End Function
```

3. 示例

给定列表函数如下：

x	1.615	1.634	1.702	1.828	1.921
y=f(x)	2.41450	2.46459	2.65271	3.03035	3.34066

调用函数 INLagrn3 **来求** f(x)**在** x=1.682, 1.813 **处的函数近似值。程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, x(5) As Double, y(5) As Double
    Dim s As String

    ' 给定结点
    x(1) = 1.615
    x(2) = 1.634
    x(3) = 1.702
    x(4) = 1.828
    x(5) = 1.921

    y(1) = 2.4145
    y(2) = 2.46459
    y(3) = 2.65271
    y(4) = 3.03035
    y(5) = 3.34066

    ' 插值位置
```

```
    t = 1.682
    ' 插值
    z = INLagrn3(5, x, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 1.813
    ' 插值
    z = INLagrn3(5, x, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

图 5.3　程序输出结果

其输出结果如图 5.3 所示。

5.4　一元三点等距插值

1. 算法原理

一元三点等距插值是指给定 n 个等距结点 $x_i = x_0 + ih(i = 0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i = f(x_i)$，用抛物插值公式，计算指定插值点 t 处的函数近似值。其算法原理同 5.3 节。

2. 算法实现

根据上述算法，可以定义一元三点等距插值的 Visual Basic 函数 INEdLagrn3，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdLagrn3
'  功能：  进行一元三点等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                   y(i) = f(x(i)), i=1,2,...,n
'          t     - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INEdLagrn3(n As Integer, h As Double, x0 As Double, _
y() As Double, t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer
    Dim z As Double, s As Double, xi As Double, xj As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
```

```
    INEdLagrn3 = z
    Exit Function
End If

If (n = 1) Then
    z = y(1)
    INEdLagrn3 = z
    Exit Function
End If

If (n = 2) Then
    z = (y(2) * (t - x0) - y(1) * (t - x0 - h)) / h
    INEdLagrn3 = z
    Exit Function
End If

' 开始插值
If (t <= x0 + h) Then
    k = 0
    m = 2
Else
    If (t >= x0 + (n - 3) * h) Then
        k = n - 3
        m = n - 1
    Else
        i = Int((t - x0) / h) + 1
        If (Abs(t - x0 - i * h) >= Abs(t - x0 - (i - 1) * h)) Then
            k = i - 2
            m = i
        Else
            k = i - 1
            m = i + 1
        End If
    End If
End If

z = 0#
For i = k To m
    s = 1#
    xi = x0 + i * h

    For j = k To m
        If (j <> i) Then
            xj = x0 + j * h
            ' 抛物线插值公式
            s = s * (t - xj) / (xi - xj)
        End If
    Next j

    z = z + s * y(i + 1)
Next i
```

```
    ' 返回结果
    INEdLagrn3 = z

End Function
```

3. 示例

设函数 $f(x)=\mathrm{e}^{-x}$ **在** 10 **个等距结点上的函数值如下表：**

x	0.1	0.2	0.3	0.4	0.5
e^{-x}	0.904837	0.818731	0.740818	0.670320	0.606531

x	0.6	0.7	0.8	0.9	1.0
e^{-x}	0.548812	0.496585	0.449329	0.406570	0.367879

调用函数 INEdLagrn3 **计算在插值点** t=0.23，0.63，0.95 **处的函数近似值，取** x_0=0.1，h=0.1，n=10。**程序代码如下。**

```
Sub Main()
    Dim t As Double, z As Double, x0 As Double, h As Double, y(10) As Double
    Dim s As String

    ' 第一个结点
    x0 = 0.1

    ' 步长
    h = 0.1

    ' 结点的函数值
    y(1) = 0.904837
    y(2) = 0.818731
    y(3) = 0.740818
    y(4) = 0.67032
    y(5) = 0.606531
    y(6) = 0.548812
    y(7) = 0.496585
    y(8) = 0.449329
    y(9) = 0.40657
    y(10) = 0.367879

    ' 插值位置
    t = 0.25
    ' 插值
    z = INEdLagrn3(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.63
    ' 插值
    z = INEdLagrn3(10, h, x0, y, t)
```

```
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.95
    ' 插值
    z = INEdLagrn3(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z

    MsgBox s

End Sub
```

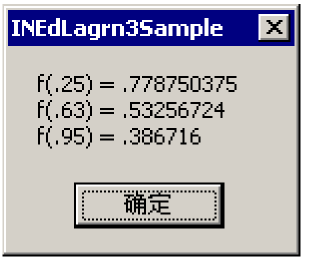

图 5.4　程序输出结果

其输出结果如图 5.4 所示。

5.5　连分式不等距插值

1. 算法原理

本节介绍给定 n 个不等距结点 $x_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i = f(x_i)$，用连分式插值法，计算指定插值点 t 处的函数近似值的算法。

给定 n 个不等距结点为 $x_0 < x_1 < \Lambda < x_{n-1}$，其相应的函数值为 $y_i(i=0，1，\cdots，n-1)$，则可以构造一个 n 节连分式：

$$\boldsymbol{\Phi}(x)=b_0+\cfrac{x-x_0}{b_1+\cfrac{x-x_1}{b_2+\cdots+\cfrac{x-x_{n-2}}{b_{n-1}}}}$$

其中，常数项 b_0 及部分分母 $b_1,\ b_2,\ \Lambda\ ,\ b_{n-1}$ 由下列递推公式计算：

$$\varphi_0(x_j)=y_j,\qquad j=0,\ 1,\ \Lambda\ ,\ n-1$$

$$\varphi_{i+1}(x_j)=\frac{x_j-x_i}{\varphi_i(x_j)-\varphi_i(x_i)},\quad \begin{array}{l} i=0,\ 1,\ \Lambda\ ,\ n-1 \\ j=i+1,\ \Lambda\ ,\ n-1 \end{array}$$

$$b_i=\varphi_i(x_i),\qquad i=0,\ 1,\ \Lambda\ ,\ n-1$$

在实际进行插值计算时，一般在指定插值点 t 的前后各取 4 个结点就够了。此时，计算八节连分式的值 $\varphi(t)$，将其作为插值点 t 处的函数近似值。

2. 算法实现

根据上述算法，可以定义连分式不等距插值的 Visual Basic 函数 INPq，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INPq
'  功能：  进行连分式不等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个结点的函数值y(i)，
'                   y(i) = f(x(i)), i=1,2,...,n
```

```
'         t     - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INPq(n As Integer, x() As Double, y() As Double, _
t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer, l As Integer
    Dim z As Double, h As Double, b(8) As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INPq = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INPq = z
        Exit Function
    End If

    ' 开始插值
    If (n <= 8) Then
        k = 0
        m = n
    Else
        If (t < x(5)) Then
            k = 0
            m = 8
        Else
            If (t > x(n - 4)) Then
                k = n - 8
                m = 8
            Else
                k = 1
                j = n
                While (j - k <> 1)
                    i = (k + j) / 2
                    If (t < x(i)) Then
                        j = i
                    Else
                        k = i
                    End If
                Wend

                k = k - 4
                m = 8
            End If
        End If
```

```
    End If

    b(1) = y(k + 1)

    For i = 2 To m
       h = y(i + k)
       l = 0
       j = 1

       While ((l = 0) And (j <= i - 1))
          If (Abs(h - b(j)) + 1# = 1#) Then
             l = 1
          Else
             h = (x(i + k) - x(j + k)) / (h - b(j))
          End If

          j = j + 1
       Wend

       b(i) = h

       If (l <> 0) Then b(i) = 1E+35
    Next i

    z = b(m)
    For i = m - 1 To 1 Step -1
       z = b(i) + (t - x(i + k)) / z
    Next i

    ' 返回结果
    INPq = z

End Function
```

3. 示例

设函数 $f(x)=\dfrac{1}{1+25x^2}$ **在** 10 **个不等距结点上的函数值如下表：**

x	−1.00	−0.80	−0.65	−0.40	−0.30
$f(x)$	0.0384615	0.0588236	0.0864865	0.200000	0.307692

x	0.00	0.20	0.45	0.80	1.00
$f(x)$	1.00000	0.500000	0.164948	0.0588236	0.0384615

调用函数 INPq **计算** t=-0.85 **及** t=0.25 **处的函数近似值。程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, x(10) As Double, y(10) As Double
    Dim s As String

    ' 给定结点
```

```
    x(1) = -1#
    x(2) = -0.8
    x(3) = -0.65
    x(4) = -0.4
    x(5) = -0.3
    x(6) = 0
    x(7) = 0.2
    x(8) = 0.45
    x(9) = 0.8
    x(10) = 1#

    y(1) = 0.0384615
    y(2) = 0.0588236
    y(3) = 0.0864865
    y(4) = 0.2
    y(5) = 0.307692
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.164948
    y(9) = 0.0588236
    y(10) = 0.0384615

    ' 插值位置
    t = -0.85
    ' 插值
    z = INPq(10, x, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.25
    ' 插值
    z = INPq(10, x, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

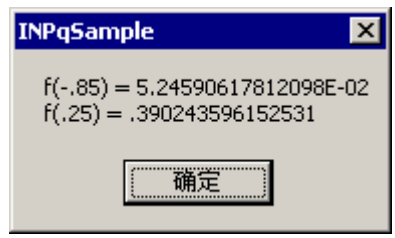

图 5.5　程序输出结果

其输出结果如图 5.5 **所示。**

5.6　连分式等距插值

1. **算法原理**

本节介绍给定 n **个等距结点** $x_i = x_0 + ih(i = 0,\ 1,\ \Lambda\ ,\ n-1)$ **上的函数值** $y_i = f(x_i)$ **，利用连分式插值公式计算指定插值点** t **处的函数近似值的算法。其算法原理同** 5.5 **节。**

2. **算法实现**

根据上述算法，可以定义连分式等距插值的 Visual Basic **函数** INEdPq**，其代码如下：**

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdPq
'  功能：  进行连分式等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          t     - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INEdPq(n As Integer, h As Double, x0 As Double, y() As Double, _
t As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer, l As Integer
    Dim z As Double, hh As Double, xi As Double, xj As Double, b(8) As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INEdPq = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INEdPq = z
        Exit Function
    End If

    ' 开始插值
    If (n <= 8) Then
        k = 0
        m = n
    Else
        If (t < (x0 + 4# * h)) Then
            k = 0
            m = 8
        Else
            If (t > (x0 + (n - 5) * h)) Then
                k = n - 8
                m = 8
            Else
                k = Int((t - x0) / h) - 3
                m = 8
            End If
        End If
    End If
```

```
    b(1) = y(k + 1)
    For i = 2 To m
       hh = y(i + k)
       l = 0
       j = 1

       While ((l = 0) And (j <= i - 1))
          If (Abs(hh - b(j)) + 1# = 1#) Then
             l = 1
          Else
             xi = x0 + (i + k - 1) * h
             xj = x0 + (j + k - 1) * h
             hh = (xi - xj) / (hh - b(j))
          End If

          j = j + 1
       Wend

       b(i) = hh
       If (l <> 0) Then b(i) = 1E+35
    Next i

    z = b(m)
    For i = m - 1 To 1 Step -1
      z = b(i) + (t - (x0 + (i + k - 1) * h)) / z
    Next i

    ' 返回结果
    INEdPq = z

End Function
```

3. 示例

设函数 $f(x)=\dfrac{1}{1+25x^2}$ **在** 11 **个等距结点上的函数值如下表：**

x	−1.00	−0.80	−0.60	−0.40	−0.20	0.00
$f(x)$	0.0384615	0.0588236	0.100000	0.200000	0.500000	1.00000

x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

调用函数 INEdPq **计算在插值点** t=-0.75 **及** t=-0.05 **处的函数近似值，取** x_0=-1.00，h=0.20，n=11。**程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, x0 As Double, h As Double, y(11) As Double
    Dim s As String

    ' 第一个结点
```

```
    x0 = -1#

    ' 步长
    h = 0.2

    ' 结点的函数值
    y(1) = 0.0384615
    y(2) = 0.0588236
    y(3) = 0.1
    y(4) = 0.2
    y(5) = 0.5
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.2
    y(9) = 0.1
    y(10) = 0.0588236
    y(11) = 0.0384615

    ' 插值位置
    t = -0.75
    ' 插值
    z = INEdPq(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = -0.05
    ' 插值
    z = INEdPq(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

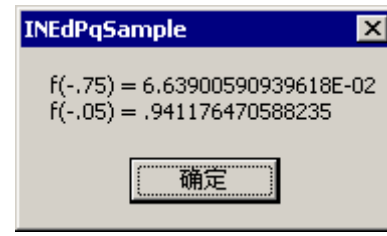

图 5.6 程序运行结果

其输出结果如图 5.6 **所示。**

5.7 埃尔米特不等距插值

1. 算法原理

埃尔米特不等距插值是指给定 n **个不等距结点** $x_i\ (i=0,\ 1,\ \Lambda\ ,\ n-1)$ **上的函数值** $f(x_i)$ **及一阶导数值** $f'(x_i)$**，用埃米尔特**(Hermite)**插值法计算指定插值点** t **处的函数近似值** $f(t)$**。**

设函数 $f(x)$**在** n **个不等距结点** $x_0 < x_1 < \Lambda < x_{n-1}$ **上的函数值及一阶导数值分别为：**

$$y_i = f(x_i),\ \ y_i' = f'(x_i),\ \ i=0,\ 1,\ \Lambda\ ,\ n-1$$

则 $f(x)$ **可用埃米尔特插值多项式**

$$P_{2n-1}(x) = \sum_{i=0}^{n-1}\left[y_i + (x-x_i)(y_i' - 2y_i l_i'(x_i))\right]t_i^2(x)$$

近似代替。其中：

$$l_i(x)=\prod_{\substack{j=0\\ j\neq i}}^{n-1}\left[(x-x_j)/(x_i-x_j)\right]$$

$$l_i'(x_i)=\sum_{\substack{j=0\\ j\neq i}}^{n-1}\frac{1}{x_i-x_j}$$

在实际进行插值计算时，为了减少计算量，用户可以适当地将远离插值点的结点抛弃。通常，只需取插值点 t 的前后各 4 个结点就够了。

2. 算法实现

根据上述算法，可以定义埃尔米特不等距插值的 Visual Basic 函数 INHermite，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INHermite
'  功能：  进行埃尔米特不等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'         dy     - Double型一维数组，长度为n，存放给定的n个结点的一阶导数值y'(i)，
'                  y'(i) = f'(x(i)), i=1,2,...,n
'          t     - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INHermite(n As Integer, x() As Double, y() As Double, _
dy() As Double, t As Double) As Double
    Dim i As Integer, j As Integer
    Dim z As Double, p As Double, q As Double, s As Double

    ' 初值
    z = 0#

    ' 循环计算
    For i = 1 To n
        s = 1#

        For j = 1 To n
          If (j <> i) Then s = s * (t - x(j)) / (x(i) - x(j))
        Next j

        s = s * s
        p = 0#
        For j = 1 To n
          If (j <> i) Then p = p + 1# / (x(i) - x(j))
        Next j
```

```
        q = y(i) + (t - x(i)) * (dy(i) - 2# * y(i) * p)
        z = z + q * s
    Next i

    ' 返回结果
    INHermite = z

End Function
```

3. 示例

设函数 $f(x)=\mathrm{e}^{-x}$ **在** 10 **个不等距结点上的函数值与一阶导数值如下表：**

x	0.10	0.15	0.30	0.45	0.55
$f(x)$	0.904837	0.860708	0.740818	0.637628	0.576950
$f'(x)$	−0.904837	−0.860708	−0.740818	−0.637628	−0.576950

x	0.60	0.70	0.85	9	100
$f(x)$	0.548812	0.496585	0.427415	0.406570	0.367879
$f'(x)$	−0.548812	−0.496585	−0.427415	−0.406570	−0.367879

调用函数 INHermite **计算在插值点** t=0.356 **处的函数近似值** $f(t)$**。程序代码如下：**

```
Sub Main()
    Dim i As Integer
    Dim t As Double, z As Double, x(10) As Double, y(10) As Double, _
        dy(10) As Double
    Dim s As String

    ' 给定结点
    x(1) = 0.1
    x(2) = 0.15
    x(3) = 0.3
    x(4) = 0.45
    x(5) = 0.55
    x(6) = 0.6
    x(7) = 0.7
    x(8) = 0.85
    x(9) = 0.9
    x(10) = 1#

    y(1) = 0.904837
    y(2) = 0.860708
    y(3) = 0.740818
    y(4) = 0.637628
    y(5) = 0.57695
    y(6) = 0.548812
    y(7) = 0.496585
    y(8) = 0.427415
    y(9) = 0.40657
```

```
    y(10) = 0.367879

    ' 求示例函数的一阶导数值
    For i = 1 To 10
       dy(i) = -y(i)
    Next i

    ' 插值位置
    t = 0.356
    ' 插值
    z = INHermite(10, x, y, dy, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

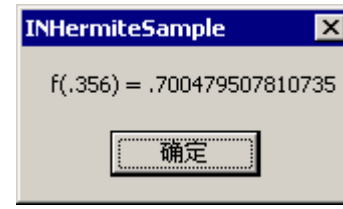

图 5.7　程序输出结果

其输出结果如图 5.7 **所示。**

5.8　埃尔米特等距插值

1. 算法原理

埃尔米特等距插值是对已知等距结点的一种插值算法，即给定 n **个等距结点** $x_i(i=0, 1, \cdots n-1)$**上的函数值** $f(x_i)$**及一阶导数值，利用连分式插值公式，计算指定插值点** t **处的函数近似值** $z=f(t)$**。其算法原理同** 5.7 **节。**

2. 算法实现

根据上述算法，可以定义埃尔米特等距插值的 Visual Basic **函数** INEdHermite**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdHermite
'  功能：  进行埃尔米特等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'         dy     - Double型一维数组，长度为n，存放给定的n个结点的一阶导数值y'(i)，
'                  y'(i) = f'(x(i)), i=1,2,...,n
'          t   - Double型变量，存放指定的插值点的值
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INEdHermite(n As Integer, h As Double, x0 As Double, y() As Double,
dy() As Double, t As Double) As Double
    Dim i As Integer, j As Integer
    Dim z As Double, s As Double, p As Double, q As Double
```

```
    ' 初值
    z = 0#

    ' 循环计算
    For i = 1 To n
       s = 1#
       q = x0 + (i - 1) * h

       For j = 1 To n
          p = x0 + (j - 1) * h
          If (j <> i) Then s = s * (t - p) / (q - p)
       Next j

       s = s * s
       p = 0#

       For j = 1 To n
         If (j <> i) Then p = p + 1# / (q - (x0 + (j - 1) * h))
       Next j

       q = y(i) + (t - q) * (dy(i) - 2# * y(i) * p)
       z = z + q * s
    Next i

   ' 返回结果
    INEdHermite = z

End Function
```

3. 示例

设函数 $f(x)=\mathrm{e}^{-x}$ **在** 10 **个等距结点上的函数值与一阶导数值如下表：**

x	0.10	0.20	0.30	0.40	0.55
$f(x)$	0.904837	0.818731	0.740818	0.670320	0.606531
$f'(x)$	−0.904837	−0.818731	−0.740818	−0.670320	−0.606531

x	0.60	0.70	0.80	0.9	1.00
$f(x)$	0.548812	0.496585	0.449329	0.406570	0.367879
$f'(x)$	−0.548812	−0.496585	−449329	−0.406570	−0.367879

调用函数 INEdHermite **计算在插值点** t=0.752 **处的函数近似值** $f(t)$，**取** $x_0=0.1$，$h=0.1$。**程序代码如下：**

```
Sub Main()
   Dim i As Integer
   Dim t As Double, z As Double, x0 As Double, h As Double, y(10) As Double,
dy(10) As Double
   Dim s As String
```

```
    ' 第一个结点
    x0 = 0.1

    ' 步长
    h = 0.1

    ' 结点的函数值
    y(1) = 0.904837
    y(2) = 0.818731
    y(3) = 0.740818
    y(4) = 0.67032
    y(5) = 0.606531
    y(6) = 0.548812
    y(7) = 0.496585
    y(8) = 0.449329
    y(9) = 0.40657
    y(10) = 0.367879

    ' 给定函数的一阶导数值
    For i = 1 To 10
       dy(i) = -y(i)
    Next i

    ' 插值位置
    t = 0.752
    ' 插值
    z = INEdPq(10, h, x0, y, t)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

图 5.8　程序输出结果

其输出结果如图 5.8 **所示。**

5.9　埃特金不等距逐步插值

1. **算法原理**

本节介绍给定 n **个不等距结点** $x_i\,(i=0,\ 1,\ \Lambda\ ,\ n-1)$ **上的函数值** $y_i = f(x_i)$ **及精度要求，利用埃特金**(Aitken)**逐步插值法，计算指定插值点** t **处的函数近似值的算法。**

给定的 n **个不等距结点为** $x_0 < x_1 < \Lambda < x_{n-1}$**，其中函数值为** $y_i\,(i=0,\ 1,\ \Lambda\ ,\ n-1)$**。**

首先，从给定的 n **个结点中选取最靠近插值点** t **的** m **个结点** $x_0^{'},\ x_1^{'},\ \Lambda\ ,\ x_{m-1}^{'}$**，相应的函数值为** $y_0^{'},\ y_1^{'},\ \Lambda\ ,\ y_{m-1}^{'}$**，其中** $m \le n$**。在本函数程序中，**m **的最大值取为** 10**。**

然后用此 m **个结点作埃特金逐步线性插值。**

埃特金逐步插值的步骤如下：

(1)　$y_0^{'} \Rightarrow p_0,\ 1 \Rightarrow i$ **；**

(2) $y_i' \Rightarrow z$,

$$p_j + \frac{t - x_j'}{x_j' - x_i'}\left[p_j - z\right] \Rightarrow z, \quad j = 0,\ 1,\ \Lambda\ ,\ i-1$$

(3) $z \Rightarrow p_i$;

(4) **若** $i = m-1$ **或** $|p_i - p_{i-1}| < \varepsilon$，**则结束，** p_i **即为** $f(t)$ **的近似值；否则，** $i+1 \Rightarrow i$，**转** (2)。

2. **算法实现**

根据上述算法，可以定义埃特金不等距逐步插值的 Visual Basic **函数** INAitken**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INAitken
'  功能：  进行埃特金不等距逐步插值
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          t     - Double型变量，存放指定的插值点的值
'        eps     - Double型变量，精度控制参数
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INAitken(n As Integer, x() As Double, y() As Double, _
t As Double, eps As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer, l As Integer
    Dim z As Double, xx(10) As Double, yy(10) As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INAitken = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INAitken = z
        Exit Function
    End If

    ' 开始插值
    m = 10
    If (m > n) Then m = n
```

```
If (t <= x(1)) Then
   k = 1
Else
   If (t >= x(n)) Then
      k = n
   Else
      k = 1
      j = n

      While ((k - j <> 1) And (k - j <> -1))
         l = (k + j) / 2
         If (t < x(l)) Then
            j = l
         Else
            k = l
         End If
      Wend

      If (Abs(t - x(l)) > Abs(t - x(j))) Then k = j
   End If
End If

j = 1
l = 0

For i = 1 To m
   k = k + j * l
   If ((k < 1) Or (k > n)) Then
      l = l + 1
      j = -j
      k = k + j * l
   End If

   xx(i) = x(k)
   yy(i) = y(k)
   l = l + 1
   j = -j
Next i

i = 0
Do
   i = i + 1
   z = yy(i + 1)
   For j = 1 To i
     z = yy(j) + (t - xx(j)) * (yy(j) - z) / (xx(j) - xx(i + 1))
   Next j

   yy(i + 1) = z
Loop While ((i <> m - 1) And (Abs(yy(i + 1) - yy(i)) > eps))

' 返回结果
```

```
    INAitken = z

End Function
```

3. 示例

设函数 $f(x)$在 11 个结点上的函数值如下表：

x	−1.00	−0.80	−0.65	−0.40	−0.30	0.00
$f(x)$	0.0384615	0.0588236	0.0864865	0.200000	0.307692	1.00000

x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

调用函数 INAitken **计算在插值点** t=-0.75 **及** t=0.05 **处的函数近似值，取** $\varepsilon=10^{-6}$**。程序代码如下：**

```
Sub Main()
    Dim t As Double, z As Double, eps As Double, x(11) As Double, y(11) As
Double
    Dim s As String

    eps = 0.000001

    ' 给定结点
    x(1) = -1#
    x(2) = -0.8
    x(3) = -0.65
    x(4) = -0.4
    x(5) = -0.3
    x(6) = 0#
    x(7) = 0.2
    x(8) = 0.4
    x(9) = 0.6
    x(10) = 0.8
    x(11) = 1#

    y(1) = 0.0384615
    y(2) = 0.0588236
    y(3) = 0.0864865
    y(4) = 0.2
    y(5) = 0.307692
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.2
    y(9) = 0.1
    y(10) = 0.0588236
    y(11) = 0.0384615

    ' 插值位置
    t = -0.75
```

```
    ' 插值
    z = INAitken(10, x, y, t, eps)
    s = s & "f(" & t & ") = " & z & Chr(13)

    ' 插值位置
    t = 0.05
    ' 插值
    z = INAitken(10, x, y, t, eps)
    s = s & "f(" & t & ") = " & z & Chr(13)

    MsgBox s

End Sub
```

其输出结果如图 5.9 所示。

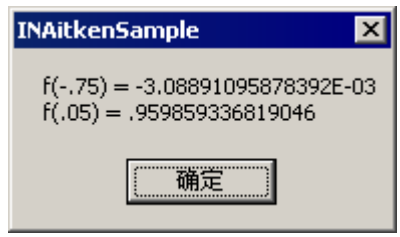

图 5.9　程序输出结果

5.10　埃特金等距逐步插值

1. 算法原理

本节介绍给定 n 个等距结点 $x_i = x_0 + ih(i = 0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i = f(x_i)$ 及精度要求，利用埃特金(Aitken)逐步插值法，计算指定插值点 t 处的函数近似值 z=$f(t)$。计算方法同 5.9 节。

2. 算法实现

根据上述算法，可以定义埃特金等距逐步插值的 Visual Basic 函数 INEdAitken，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdAitken
'  功能：  进行埃特金等距逐步插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'         dy     - Double型一维数组，长度为n，存放给定的n个结点的一阶导数值y'(i)，
'                  y'(i) = f'(x(i)), i=1,2,...,n
'          t   - Double型变量，存放指定的插值点的值
```

```
'        eps     - Double型变量，精度控制参数
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INEdAitken(n As Integer, h As Double, x0 As Double, _
y() As Double, t As Double, eps As Double) As Double
    Dim i As Integer, j As Integer, k As Integer, m As Integer, l As Integer
    Dim z As Double, xx(10) As Double, yy(10) As Double

    ' 初值
    z = 0#

    ' 特例处理
    If (n < 1) Then
        INEdAitken = z
        Exit Function
    End If

    If (n = 1) Then
        z = y(1)
        INEdAitken = z
        Exit Function
    End If

    ' 开始插值
    m = 10
    If (m > n) Then m = n

    If (t <= x0) Then
        k = 1
    Else
        If (t >= x0 + (n - 1) * h) Then
            k = n
        Else
            k = 1
            j = n

            While ((k - j <> 1) And (k - j <> -1))
                l = (k + j) / 2
                If (t < x0 + (l - 1) * h) Then
                    j = l
                Else
                    k = l
                End If
            Wend

            If (Abs(t - (x0 + (l - 1) * h)) > Abs(t - (x0 + (j - 1) * h))) Then
                k = j
            End If

        End If
    End If
```

```
    j = 1
    l = 0
    For i = 1 To m
       k = k + j * l
       If ((k < 1) Or (k > n)) Then
          l = l + 1
          j = -j
          k = k + j * l
       End If

       xx(i) = x0 + (k - 1) * h
       yy(i) = y(k)
       l = l + 1
       j = -j
    Next i

    i = 0

    Do
       i = i + 1
       z = yy(i + 1)

       For j = 1 To i
          z = yy(j) + (t - xx(j)) * (yy(j) - z) / (xx(j) - xx(i + 1))
       Next j

       yy(i + 1) = z
    Loop While ((i <> m - 1) And (Abs(yy(i + 1) - yy(i)) > eps))

    ' 返回结果
    INEdAitken = z

End Function
```

3. 示例

设函数 $f(x)$ 在 10 个等距结点上的函数值如下表：

x	0.10	0.20	0.30	0.40	0.50
$f(x)$	0.904837	0.818731	0.740818	0.670320	0.606531

x	0.60	0.70	0.80	0.90	1.00
$f(x)$	0.548812	0.496585	0.449329	0.406570	0.367879

调用函数 INEdAitken **计算在插值点** t=0.15 **及** t=0.55 **处的函数近似值，取** x_0=0.1，h=0.1，$\varepsilon = 10^{-6}$。**程序代码如下：**

```
Sub Main()
    Dim i As Integer
    Dim t As Double, z As Double, x0 As Double, h As Double, _
```

```
    y(10) As Double, eps As Double
Dim s As String

' 第一个结点
x0 = 0.1

' 步长
h = 0.1

' 精度
eps = 0.000001

' 结点的函数值
y(1) = 0.904837
y(2) = 0.818731
y(3) = 0.740818
y(4) = 0.67032
y(5) = 0.606531
y(6) = 0.548812
y(7) = 0.496585
y(8) = 0.449329
y(9) = 0.40657
y(10) = 0.367879

' 插值位置
t = 0.15
' 插值
z = INEdAitken(10, h, x0, y, t, eps)
s = s & "f(" & t & ") = " & z & Chr(13)

' 插值位置
t = 0.55
' 插值
z = INEdAitken(10, h, x0, y, t, eps)
s = s & "f(" & t & ") = " & z & Chr(13)

MsgBox s

End Sub
```

图 5.10 程序输出结果

其输出结果如图 5.10 所示。

5.11 光滑不等距插值

1. 算法原理

光滑不等距插值是指给定 n 个不等距结点 $x_i(i=0，1，\cdots，n-1)$上的函数值 $y_i=f(x_i)$及精度要求，利用阿克玛(Akima)方法，计算指定子区间上的三次插值多项式与指定插值点上的函数值。

设给定的 n 个不等距结点为 $x_0<x_1<\cdots<x_{n-1}$，相应的函数值为 $y_i(i=0，1，\cdots，n-1)$。若在子区间 $[x_k，\ x_{k+1}](k=0,\ 1,\ \Lambda\ ,\ n-2)$ 上的两个端点处满足以下 4 个条件：

$$\begin{cases} y_k = f(x_k) \\ y_{k+1} = f(x_{k+1}) \\ y_k^{'} = g_k \\ y_{k+1}^{'} = g_{k+1} \end{cases}$$

则在此区间上可以惟一确定一个三次多项式：

$$s(x) = s_0 + s_1(x-x_k) + s_2(x-x_k)^2 + s_3(x-x_k)^3$$

利用此三次多项式可计算该子区间中的插值点 t 处的函数近似值。根据阿克玛几何条件，g_k 与 g_{k+1} 由下式计算：

$$g_k = \frac{|u_{k+1}-u_k|u_{k-1}+|u_{k-1}-u_{k-2}|u_k}{|u_{k+1}-u_k|+|u_{k-1}-u_{k-2}|}$$

$$g_{k+1} = \frac{|u_{k+2}-u_{k+1}|u_k+|u_k-u_{k-1}|u_{k+1}}{|u_{k+2}-u_{k+1}|+|u_k-u_{k-1}|}$$

其中，

$$u_k = \frac{y_{k+1}-y_k}{x_{k+1}-x_k}$$

并且在端点处有：

$$u_{-1} = 2u_0 - u_1, u_{-2} = 2u_{-1} - u_0$$

$$u_{n-1} = 2u_{n-2} - u_{n-3}, u_n = 2u_{n-1} - u_{n-2}$$

当 $u_{k+1} = u_k$ 与 $u_{k-1} = u_{k-2}$ 时，

$$g_k = \frac{u_{k-1}+u_k}{2}$$

当 $u_{k+2} = u_{k+1}$ 与 $u_k = u_{k-1}$ 时，

$$g_{k+1} = \frac{u_k+u_{k+1}}{2}$$

最后可以得到区间 $[x_k，\ x_{k+1}](k=0,1,\Lambda\ ,n-2)$ 上的三次多项式的系数：

$$s_0 = y_k$$

$$s_1 = g_k$$

$$s_2 = (3u_k - 2g_k - g_{k+1})/(x_{k+1}-x_k)$$

$$s_3 = (g_{k+1} + g_k - 2u_k)/(x_{k+1}-x_k)^2$$

插值点 $t(t \in [x_k, x_{k+1}])$ 处的函数近似值为：

$$s(t) = s_0 + s_1(t-x_k) + s_2(t-x_k)^2 + s_3(t-x_k)^3$$

2. 算法实现

根据上述算法，可以定义光滑不等距插值的 Visual Basic 函数 INAkima，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INAkima
'  功能：  光滑不等距插值
'  参数：  n    - Integer型变量，给定结点的点数
'          x    - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y    - Double型一维数组，长度为n，存放给定的n个结点的函数值y(i)，
'                 y(i) = f(x(i)), i=1,2,...,n
'          k    - Integer型变量，控制参数，若k>=0，则只计算第k个子区间[x(k), x(k+1)]
'                 上的三次多项式的系数s1，s2，s3，s4；若k<0，则需要计算指定插值点t处
'                 的函数近似值f(t)，并计算所在子区间的三次多项式系数s1，s2，s3，s4
'          t    - Double型变量，存放指定的插值点的值，若k>=0，此参数不起作用，可为任
'                 意值
'          s    - Double型一维数组，长度为5，其中s1，s2，s3，s4返回三次多项式的系
'                 数，s5返回指定插值点t处的函数近似值f(t)(k<0时)或任意值(k>=0时)'
返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub INAkima(n As Integer, x() As Double, y() As Double, k As Integer, _
t As Double, s() As Double)
    Dim kk As Integer, l As Integer, m As Integer
    Dim u(5) As Double, p As Double, q As Double

    ' 初值
    s(5) = 0#
    s(1) = 0#
    s(2) = 0#
    s(3) = 0#
    s(4) = 0#

    ' 特例处理
    If (n < 1) Then
       Exit Sub
    End If

    If (n = 1) Then
       s(1) = y(1)
       s(5) = y(1)
       Exit Sub
    End If

    If (n = 2) Then
       s(1) = y(1)
       s(2) = (y(2) - y(1)) / (x(2) - x(1))
       If (k < 0) Then
         s(5) = (y(1) * (t - x(2)) - y(2) * (t - x(1))) / (x(1) - x(2))
       End If
       Exit Sub
```

```
End If

' 开始插值
If (k < 0) Then
   If (t <= x(2)) Then
      kk = 0
   Else
      If (t >= x(n)) Then
         kk = n - 2
      Else
         kk = 1
         m = n
         While (((kk - m) <> 1) And ((kk - m) <> -1))
            l = (kk + m) / 2
            If (t < x(l)) Then
               m = l
            Else
               kk = l
            End If
         Wend

         kk = kk - 1
      End If
   End If
Else
   kk = k
End If

If (kk >= n - 1) Then kk = n - 2

' 调用Akima公式
u(3) = (y(kk + 2) - y(kk + 1)) / (x(kk + 2) - x(kk + 1))
If (n = 3) Then
   If (kk = 0) Then
      u(4) = (y(3) - y(2)) / (x(3) - x(2))
      u(5) = 2# * u(4) - u(3)
      u(2) = 2# * u(3) - u(4)
      u(1) = 2# * u(2) - u(3)
   Else
      u(2) = (y(2) - y(1)) / (x(2) - x(1))
      u(1) = 2# * u(2) - u(3)
      u(4) = 2# * u(3) - u(2)
      u(5) = 2# * u(4) - u(3)
   End If
Else
   If (kk <= 1) Then
      u(4) = (y(kk + 3) - y(kk + 2)) / (x(kk + 3) - x(kk + 2))
      If (kk = 1) Then
         u(2) = (y(2) - y(1)) / (x(2) - x(1))
         u(1) = 2# * u(2) - u(3)
         If (n = 4) Then
```

```
                u(5) = 2# * u(4) - u(3)
            Else
                u(5) = (y(5) - y(4)) / (x(5) - x(4))
            End If
        Else
            u(2) = 2# * u(3) - u(4)
            u(1) = 2# * u(2) - u(3)
            u(5) = (y(4) - y(3)) / (x(4) - x(3))
        End If
    Else
        If (kk >= (n - 3)) Then
            u(2) = (y(kk + 1) - y(kk)) / (x(kk + 1) - x(kk))
            If (kk = (n - 3)) Then
                u(4) = (y(n) - y(n - 1)) / (x(n) - x(n - 1))
                u(5) = 2# * u(4) - u(3)
                If (n = 4) Then
                    u(1) = 2# * u(2) - u(3)
                Else
                    u(1) = (y(kk) - y(kk - 1)) / (x(kk) - x(kk - 1))
                End If
            Else
                u(4) = 2# * u(3) - u(2)
                u(5) = 2# * u(4) - u(3)
                u(1) = (y(kk) - y(kk - 1)) / (x(kk) - x(kk - 1))
            End If
        Else
            u(2) = (y(kk + 1) - y(kk)) / (x(kk + 1) - x(kk))
            u(1) = (y(kk) - y(kk - 1)) / (x(kk) - x(kk - 1))
            u(4) = (y(kk + 3) - y(kk + 2)) / (x(kk + 3) - x(kk + 2))
            u(5) = (y(kk + 4) - y(kk + 3)) / (x(kk + 4) - x(kk + 3))
        End If
    End If
End If

s(1) = Abs(u(4) - u(3))
s(2) = Abs(u(1) - u(2))
If ((s(1) + 1# = 1#) And (s(2) + 1# = 1#)) Then
    p = (u(2) + u(3)) / 2#
Else
    p = (s(1) * u(2) + s(2) * u(3)) / (s(1) + s(2))
End If

s(1) = Abs(u(4) - u(5))
s(2) = Abs(u(3) - u(2))
If ((s(1) + 1# = 1#) And (s(2) + 1# = 1#)) Then
    q = (u(3) + u(4)) / 2#
Else
    q = (s(1) * u(3) + s(2) * u(4)) / (s(1) + s(2))
End If

s(1) = y(kk + 1)
```

```
    s(2) = p
    s(4) = x(kk + 2) - x(kk + 1)
    s(3) = (3# * u(3) - 2# * p - q) / s(4)
    s(4) = (q + p - 2# * u(3)) / (s(4) * s(4))

    If (k < 0) Then
       p = t - x(kk + 1)
       s(5) = s(1) + s(2) * p + s(3) * p * p + s(4) * p * p * p
    End If

End Sub
```

3. 示例

设函数 $f(x)$ **在** 11 **个结点上的函数值如下表：**

x	−1.00	−0.95	−0.75	−0.55	−0.30	0.00
$f(x)$	0.0384615	0.0424403	0.0663900	0.116788	0.307692	1.00000

x	0.20	0.45	0.60	0.80	1.00	
$f(x)$	0.500000	0.164948	0.100000	0.0588236	0.0384615	

调用函数 INAkima **计算指定插值点** t=-0.85 **与** t=0.15 **处的函数近似值及插值点所在子区间的三次多项式**

$$s(t) = s_0 + s_1(t - x_k) + s_2(t - x_k)^2 + s_3(t - x_k)^3$$

的系数 s_0，s_1，s_2，s_3。**其中** $t \in [x_k,\ x_{k+1}]$。**程序代码如下：**

```
Sub Main()
    Dim k As Integer, n As Integer
    Dim t As Double, x(11) As Double, y(11) As Double, s(5) As Double
    Dim str As String

    ' 给定结点
    x(1) = -1#
    x(2) = -0.95
    x(3) = -0.75
    x(4) = -0.55
    x(5) = -0.3
    x(6) = 0#
    x(7) = 0.2
    x(8) = 0.45
    x(9) = 0.6
    x(10) = 0.8
    x(11) = 1#

    y(1) = 0.0384615
    y(2) = 0.0424403
    y(3) = 0.06639
    y(4) = 0.116788
```

```
    y(5) = 0.307692
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.164948
    y(9) = 0.1
    y(10) = 0.0588236
    y(11) = 0.0384615

    ' 计算插值的控制参数
    k = -1

    ' 结点数
    n = 11

    ' 插值位置
    t = -0.85
    ' 插值
    Call INAkima(10, x, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)

    ' 插值位置
    t = 0.15
    ' 插值
    Call INAkima(10, x, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)

    MsgBox str

End Sub
```

其输出结果如图 5.11 **所示。**

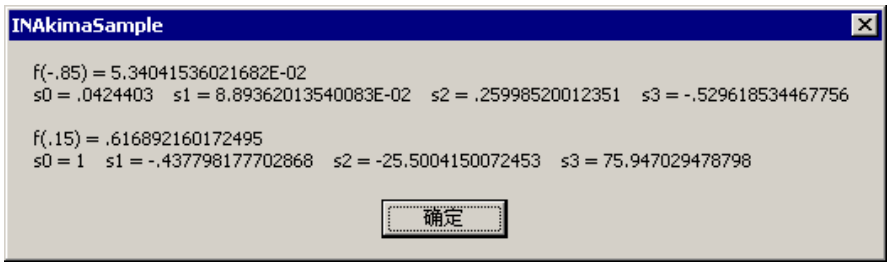

图 5.11 程序输出结果

5.12 光滑等距插值

1. **算法原理**

本节介绍光滑等距插值，即给定 n **个等距结点** $x_i = x_0 + ih(i = 0,\ 1,\ \Lambda\ ,\ n-1)$ **上的函数值** $y_i = f(x_i)$ **及精度要求，利用阿克玛**(Akima)**方法计算指定子区间上的三次插值多项式与指定插值点上的函数近似值。其算法原理同** 5.11 **节。**

2. 算法实现

根据上述算法，可以定义光滑等距插值的 Visual Basic 函数 INEdAkima，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INEdAkima
'  功能：  光滑等距插值
'  参数：  n     - Integer型变量，给定结点的点数
'          h     - Integer型变量，等距结点的步长
'          x0    - Double型变量，存放等距n个结点中第一个结点的值
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          k     - Integer型变量,控制参数,若k>=0,则只计算第k个子区间[x(k), x(k+1)]
'                  上的三次多项式的系数s1，s2，s3，s4；若k<0，则需要计算指定插值点t
'                  处的函数近似值f(t)，并计算所在子区间的三次多项式系数s1，s2，s3，s4
'          t     - Double型变量，存放指定的插值点的值,若k>=0，此参数不起作用，可为任
'                  意值
'          s     - Double型一维数组，长度为5，其中s1，s2，s3，s4返回三次多项式的系
'                  数，s5返回指定插值点t处的函数近似值f(t)(k<0时)或任意值(k>=0时)
'  返回值：Double型，指定的查指点t的函数近似值f(t)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub INEdAkima(n As Integer, h As Double, x0 As Double, y() As Double, _
k As Integer, t As Double, s() As Double)
    Dim kk As Integer, m As Integer, l As Integer
    Dim u(5) As Double, p As Double, q As Double

    ' 初值
    s(5) = 0#
    s(1) = 0#
    s(2) = 0#
    s(3) = 0#
    s(4) = 0#

    ' 特例处理
    If (n < 1) Then
       Exit Sub
    End If

    If (n = 1) Then
       s(1) = y(1)
       s(5) = y(5)
       Exit Sub
    End If

    If (n = 2) Then
       s(1) = y(1)
       s(2) = (y(2) - y(1)) / h
       If (k < 0) Then
         s(5) = (y(2) * (t - x0) - y(1) * (t - x0 - h)) / h
       End If
```

```
    Exit Sub
End If

' 开始插值
If (k < 0) Then
    If (t <= x0 + h) Then
        kk = 0
    Else
        If (t >= x0 + (n - 1) * h) Then
            kk = n - 2
        Else
            kk = 1
            m = n
            While (((kk - m) <> 1) And ((kk - m) <> -1))
                l = (kk + m) / 2
                If (t < x0 + (l - 1) * h) Then
                    m = l
                Else
                    kk = l
                End If
            Wend

            kk = kk - 1
        End If
    End If
Else
    kk = k
End If

If (kk >= n - 1) Then kk = n - 2

' 调用Akima公式
u(3) = (y(kk + 2) - y(kk + 1)) / h

If (n = 3) Then
    If (kk = 0) Then
        u(4) = (y(3) - y(2)) / h
        u(5) = 2# * u(4) - u(3)
        u(2) = 2# * u(3) - u(4)
        u(1) = 2# * u(2) - u(3)
    Else
        u(2) = (y(2) - y(1)) / h
        u(1) = 2# * u(2) - u(3)
        u(4) = 2# * u(3) - u(2)
        u(5) = 2# * u(4) - u(3)
    End If
Else
    If (kk <= 1) Then
        u(4) = (y(kk + 3) - y(kk + 2)) / h
        If (kk = 1) Then
            u(2) = (y(2) - y(1)) / h
```

```
            u(1) = 2# * u(2) - u(3)
            If (n = 4) Then
               u(5) = 2# * u(4) - u(3)
            Else
               u(5) = (y(5) - y(4)) / h
            End If
         Else
            u(2) = 2# * u(3) - u(4)
            u(1) = 2# * u(2) - u(3)
            u(5) = (y(4) - y(3)) / h
         End If
      Else
         If (kk >= (n - 3)) Then
            u(2) = (y(kk + 1) - y(kk)) / h

            If (kk = (n - 3)) Then
               u(4) = (y(n) - y(n - 1)) / h
               u(5) = 2# * u(4) - u(3)
               If (n = 4) Then
                  u(1) = 2# * u(2) - u(3)
               Else
                  u(1) = (y(kk) - y(kk - 1)) / h
               End If
            Else
               u(4) = 2# * u(3) - u(2)
               u(5) = 2# * u(4) - u(3)
               u(1) = (y(kk) - y(kk - 1)) / h
            End If
         Else
            u(2) = (y(kk + 1) - y(kk)) / h
            u(1) = (y(kk) - y(kk - 1)) / h
            u(4) = (y(kk + 3) - y(kk + 2)) / h
            u(5) = (y(kk + 4) - y(kk + 3)) / h
         End If
      End If
   End If

   s(1) = Abs(u(4) - u(3))
   s(2) = Abs(u(1) - u(2))
   If ((s(1) + 1# = 1#) And (s(2) + 1# = 1#)) Then
       p = (u(2) + u(3)) / 2#
   Else
      p = (s(1) * u(2) + s(2) * u(3)) / (s(1) + s(2))
   End If

   s(1) = Abs(u(4) - u(5))
   s(2) = Abs(u(3) - u(2))
   If ((s(1) + 1# = 1#) And (s(2) + 1# = 1#)) Then
       q = (u(3) + u(4)) / 2#
   Else
      q = (s(1) * u(3) + s(2) * u(4)) / (s(1) + s(2))
```

```
    End If

    s(1) = y(kk + 1)
    s(2) = p
    s(4) = h
    s(3) = (3# * u(3) - 2# * p - q) / s(4)
    s(4) = (q + p - 2# * u(3)) / (s(4) * s(4))
    If (k < 0) Then
       p = t - (x0 + kk * h)
       s(5) = s(1) + s(2) * p + s(3) * p * p + s(4) * p * p * p
    End If

End Sub
```

3. 示例

设函数 $f(x)$ **在** 11 **个等距结点上的函数值如下表：**

x	−1.00	−0.80	−0.60	−0.4	−0.30	0.00
$f(x)$	0.0384615	0.0588236	0.100000	0.200000	0.500000	1.00000

x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

调用函数 INEdAkima **计算指定插值点** t=-0.65，0.25 **处的函数近似值及插值点所在子区间的三次多项式**

$$s(x)=s_0+s_1(x-x_k)+s_2(x-x_k)^2+s_3(x-x_k)^3$$

的系数 s_0，s_1，s_2，s_3。**其中** $t\in[x_k,\ x_{k+1}]$，x_0=-1.0，h=0.2。**程序代码如下：**

```
Sub Main()
    Dim k As Integer, n As Integer
    Dim t As Double, x0 As Double, h As Double, y(11) As Double, s(5) As Double
    Dim str As String

    ' 第一个结点
    x0 = -1#

    ' 步长
    h = 0.2

    ' 计算插值的控制参数
    k = -1

    ' 结点数
    n = 11

    ' 结点的函数值
    y(1) = 0.0384615
    y(2) = 0.0588236
    y(3) = 0.1
```

```
    y(4) = 0.2
    y(5) = 0.5
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.2
    y(9) = 0.1
    y(10) = 0.0588236
    y(11) = 0.0384615

    ' 插值位置
    t = -0.65
    ' 插值
    Call INEdAkima(n, h, x0, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)

    ' 插值位置
    t = 0.25
    ' 插值
    Call INEdAkima(n, h, x0, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)

    MsgBox str

End Sub
```

其输出结果如图 5.12 **所示。**

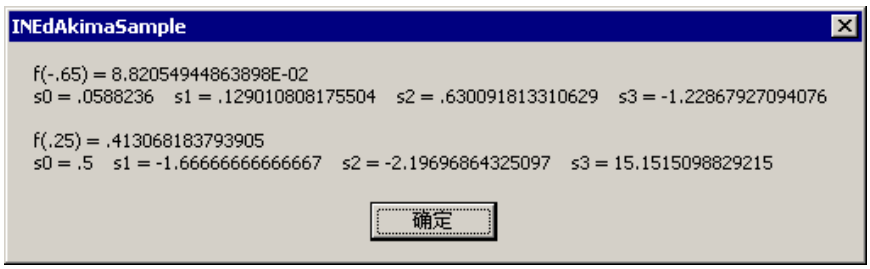

图 5.12　程序输出结果

5.13　第一种边界条件的三次样条函数插值、微商与积分

1. 算法原理

本节介绍第一种边界条件的三次样条函数插值、微商与积分算法，即给定 n **个结点** $x_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$ **上的函数值** $y_i=f(x_i)$ **及两端点上的一阶导数值** $y_0^{'}=f^{'}(x_0),\ y_{n-1}^{'}=f^{'}(x_{n-1})$**，利用三次样条函数，计算各结点上的数值导数及插值区间** $[x_0,\ x_{n-1}]$**上的积分近似值** $s=\int_{x_0}^{x_1} f(x)\,\mathrm{d}x$**，并对函数** $f(x)$ **进行成组插值及成组微商。**

设给定的 n 个结点为 $x_0 < x_1 < \Lambda < x_{n-1}$，其函数值为 $y_i\ (i = 0,\ 1,\ \Lambda,\ n-1)$，第一种边界条件为：

$$\begin{cases} y_0' = f'(x_0) \\ y_{n-1}' = f'(x_{n-1}) \end{cases}$$

计算 n-2 个结点上的一阶导数值 $y_j'\ (j = 1,\ 2,\ \Lambda,\ n-2)$ 的公式如下：

$$a_0 = 0$$

$$b_0 = y_0'$$

$$h_j = x_{j+1} - x_j,\ \ j = 0,\ 1,\ \Lambda,\ n-2$$

$$\alpha_j = h_{j-1}/(h_{j-1} + h_j),\ \ j = 1,\ 2,\ \Lambda,\ n-2$$

$$\beta j = 3\left[(1-\alpha_j)(y_j - y_{j-1})/h_{j-1} + \alpha_j(y_{j+1} - y_j)/h_j\right],\ \ j = 1,\ 2,\ \Lambda,\ n-2$$

$$a_j = -\alpha_j/\left[2 + (1-\alpha_j)a_{j-1}\right],\ \ j = 1,\ 2,\ \Lambda,\ n-2$$

$$b_j = \left[\beta_j - (1-\alpha_j)b_{j-1}\right]/\left[2 + (1-\alpha_j)a_{j-1}\right],\ \ j = 1,\ 2,\ \Lambda,\ n-2$$

$$y_j' = a_j y_{j+1}' + b_j,\ \ j = n-2,\ n-3,\ \Lambda,\ 1$$

计算 n 个结点上的二阶导数值 $y_j''\ (j = 0,\ 1,\ \Lambda,\ n-1)$ 的公式如下：

$$y_j'' = 6(y_{j+1} - y_j)/h_j^2 - 2(2y_j' + y_{j+1}')/h_j,\ \ j = 0,\ 1,\ \Lambda,\ n-2$$

$$y_{n-1}'' = 6(y_{n-2} - y_{n-1})/h_{n-2}^2 + 2(2y_{n-1}' + y_{n-2}')/h_{n-2}$$

利用各结点上的数值导数，并根据辛卜生(Simpson)公式，可以得到在插值区间 $[x_0,\ x_{n-1}]$ 上的求积公式：

$$s = \int_{x_0}^{x_{n-1}} f(x)\,\mathrm{d}x$$

$$= \frac{1}{2}\sum_{i=0}^{n-2}(x_{i+1} - x_i)(y_i + y_{i+1}) - \frac{1}{24}\sum_{i=0}^{n-2}(x_{i+1} - x_i)^3(y_i'' + y_{i+1}'')$$

利用各结点上的函数值 y_i、一阶导数值 $y_i'(i = 0,\ 1,\ \Lambda,\ n-1)$，计算插值点 t 处的函数值、一阶导数值及二阶导数值的公式如下(其中 $t \in [x_i,\ x_{i+1}]$)：

$$y(t) = \left[\frac{3}{h_i^2}(x_{i+1} - t)^2 - \frac{2}{h_i^3}(x_{i+1} - t)^3\right]y_i + \left[\frac{3}{h_i^2}(t - x_i)^2 - \frac{2}{h_i^3}(t - x_i)^3\right]y_{i+1} +$$

$$h_i\left[\frac{1}{h_i^2}(x_{i+1} - t)^2 - \frac{1}{h_i^3}(x_{i+1} - t)^3\right]y_i' - h_i\left[\frac{1}{h_i^2}(t - x_i)^2 - \frac{1}{h_i^3}(t - x_i)^3\right]y_{i+1}'$$

$$y'(t) = \frac{6}{h_i}\left[\frac{1}{h_i^2}(x_{i+1} - t)^2 - \frac{1}{h_i}(x_{i+1} - t)\right]y_i - \frac{6}{h_i}\left[\frac{1}{h_i^2}(t - x_i)^2 - \frac{1}{h_i^3}(t - x_i)\right]y_{i+1} +$$

$$\left[\frac{3}{h_i^2}(x_{i+1} - t)^2 - \frac{2}{h_i}(x_{i+1} - t)\right]y_i' + \left[\frac{3}{h_i^2}(t - x_i)^2 - \frac{2}{h_i}(t - x_i)\right]y_{i+1}'$$

$$y''(t)=\frac{1}{h_i^2}\left[6-\frac{12}{h_i}(x_{i+1}-t)\right]y_i+\frac{1}{h_i^2}\left[6-\frac{12}{h_i}(t-x_i)\right]y_{i+1}+$$
$$\frac{1}{h_i}\left[2-\frac{6}{h_i}(x_{i+1}-t)\right]y_i'-\frac{1}{h_i}\left[2-\frac{6}{h_i}(t-x_i)\right]y_{i+1}'$$

2. **算法实现**

根据上述算法，可以定义第一种边界条件的三次样条函数插值、微商与积分的 Visual Basic **函数** INSpline1**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INSpline1
'  功能：  实现第一种边界条件的三次样条函数插值、微商与积分
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n
'          dy    - Double型一维数组，长度为n，调用时，dy(1)存放给定区间的左端点处的
'                  一阶导数值，dy(n)存放给定区间的右端点处的一阶导数值。返回时，存放n
'                  个给定点处的一阶导数值y'(i)，i=1,2,…n
'          ddy   - Double型一维数组，长度为n，返回时，存放n个给定点处的二阶导数值
'                  y''(i)，i=1,2,…n
'          m     - Integer型变量，指定插值点的个数
'          t     - Double型一维数组，长度为m，存放m个指定的插值点的值。要求
'                  x(1)<t(j)<x(n), j=1,2,…,m-1
'          z     - Double型一维数组，长度为m，存放m个指定的插值点处的函数值
'          dz    - Double型一维数组，长度为m，存放m个指定的插值点处的一阶导数值
'          ddz   - Double型一维数组，长度为m，存放m个指定的插值点处的二阶导数值
'  返回值：Double型，指定函数的x(1)到x(n)的定积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INSpline1(n As Integer, x() As Double, y() As Double, _
dy() As Double, ddy() As Double, m As Integer, t() As Double, _
z() As Double, dz() As Double, ddz() As Double) As Double
    Dim i As Integer, j As Integer
    Dim h0 As Double, h1 As Double, alpha As Double, beta As Double, _
         g As Double
    ReDim s(n) As Double

    ' 初值
    s(1) = dy(1)
    dy(1) = 0#
    h0 = x(2) - x(1)

    For j = 2 To n - 1
        h1 = x(j + 1) - x(j)
        alpha = h0 / (h0 + h1)
        beta = (1# - alpha) * (y(j) - y(j - 1)) / h0
        beta = 3# * (beta + alpha * (y(j + 1) - y(j)) / h1)
        dy(j) = -alpha / (2# + (1# - alpha) * dy(j - 1))
```

```
   s(j) = (beta - (1# - alpha) * s(j - 1))
   s(j) = s(j) / (2# + (1# - alpha) * dy(j - 1))
   h0 = h1
Next j

' 一阶导数值
For j = n - 1 To 1 Step -1
   dy(j) = dy(j) * dy(j + 1) + s(j)
Next j

For j = 1 To n - 1
   s(j) = x(j + 1) - x(j)
Next j

' 二阶导数值
For j = 1 To n - 1
   h1 = s(j) * s(j)
   ddy(j) = 6# * (y(j + 1) - y(j)) / h1 - 2# * (2# * dy(j) + _
            dy(j + 1)) / s(j)
Next j

h1 = s(n - 1) * s(n - 1)
ddy(n) = 6# * (y(n - 1) - y(n)) / h1 + 2# * (2# * dy(n) + _
         dy(n - 1)) / s(n - 1)
g = 0#
For i = 1 To n - 1
   h1 = 0.5 * s(i) * (y(i) + y(i + 1))
   h1 = h1 - s(i) * s(i) * s(i) * (ddy(i) + ddy(i + 1)) / 24#
   g = g + h1
Next i

' 插值
For j = 1 To m
   If (t(j) >= x(n - 1)) Then
      i = n - 1
   Else
      i = 1
      While (t(j) > x(i + 1))
         i = i + 1
      Wend
   End If

   h1 = (x(i + 1) - t(j)) / s(i)
   h0 = h1 * h1
   z(j) = (3# * h0 - 2# * h0 * h1) * y(i)
   z(j) = z(j) + s(i) * (h0 - h0 * h1) * dy(i)
   dz(j) = 6# * (h0 - h1) * y(i) / s(i)
   dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i)
   ddz(j) = (6# - 12# * h1) * y(i) / (s(i) * s(i))
   ddz(j) = ddz(j) + (2# - 6# * h1) * dy(i) / s(i)
   h1 = (t(j) - x(i)) / s(i)
```

```
        h0 = h1 * h1
        z(j) = z(j) + (3# * h0 - 2# * h0 * h1) * y(i + 1)
        z(j) = z(j) - s(i) * (h0 - h0 * h1) * dy(i + 1)
        dz(j) = dz(j) - 6# * (h0 - h1) * y(i + 1) / s(i)
        dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i + 1)
        ddz(j) = ddz(j) + (6# - 12# * h1) * y(i + 1) / (s(i) * s(i))
        ddz(j) = ddz(j) - (2# - 6# * h1) * dy(i + 1) / s(i)
    Next j

    ' 返回积分值
    INSpline1 = g

End Function
```

3. 示例

已知某直升飞机旋转机翼外形曲线上的部分坐标值如下表：

x	0.52	8.0	17.95	28.65	50.65	104.6
y	5.28794	13.8400	20.2000	24.9000	31.1000	36.5000

x	156.6	260.7	364.4	468.0	507.0	520.0
y	36.6000	31.0000	20.9000	7.80000	1.50000	0.200000

且两端点上的一阶导数值为：$y_0' = 1.86548,\ y_n' = -0.046115$**。调用函数** INSpline1 **计算各结点处的一阶及二阶导数值，在区间**[0.52，520.0]**内的积分值。并计算在** 8 **个插值点** 4.0，14.0，30.0，60.0，130.0，230.0，450.0，515.0 **处的函数值、一阶导数值及二阶导数值。取** n=12，m=8**。程序代码如下：**

```
Sub Main()
    Dim k As Integer, n As Integer
    Dim t As Double, x0 As Double, h As Double, y(11) As Double, s(5) As Double
    Dim str As String

    ' 第一个结点
    x0 = -1#

    ' 步长
    h = 0.2

    ' 计算插值的控制参数
    k = -1

    ' 结点数
    n = 11

    ' 结点的函数值
    y(1) = 0.0384615
    y(2) = 0.0588236
    y(3) = 0.1
    y(4) = 0.2
```

```
    y(5) = 0.5
    y(6) = 1#
    y(7) = 0.5
    y(8) = 0.2
    y(9) = 0.1
    y(10) = 0.0588236
    y(11) = 0.0384615

    ' 插值位置
    t = -0.65
    ' 插值
    Call INEdAkima(n, h, x0, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)
    For i = 1 To 4
       str = str & "s" & (i - 1) & " = " & s(i) & "    "
    Next i

    str = str & Chr(13) & Chr(13)

    ' 插值位置
    t = 0.25
    ' 插值
    Call INEdAkima(n, h, x0, y, k, t, s)
    str = str & "f(" & t & ") = " & s(5) & Chr(13)
    For i = 1 To 4
       str = str & "s" & (i - 1) & " = " & s(i) & "    "
    Next i

    MsgBox str

End Sub
```

其输出结果如图 5.13 **所示。**

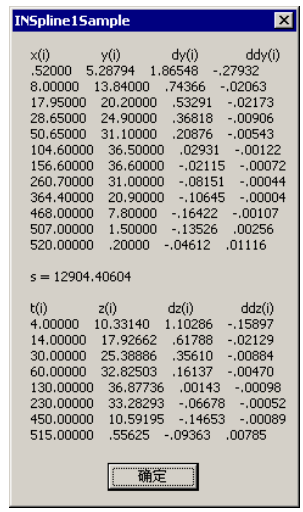

图 5.13　程序输出结果

5.14　第二种边界条件的三次样条函数插值、微商与积分

1. 算法原理

本节介绍第 2 种边界条件的三次样条函数插值、微商与积分算法，即给定 n 个结点 $x_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i=f(x_i)$ 及两端点上的二阶导数值 $y_0^{''}=f^{''}(x_0),\ y_{n-1}^{''}=f^{''}(x_{n-1})$，利用三次样条函数，计算各结点上的数值导数 $y_i^{'},y_i^{''}(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 及插值区间 $[x_0,\ x_{n-1}]$ 上的积分近似值 $s=\int_{x_0}^{x_{n-1}} f(x)\,\mathrm{d}x$，并对函数 $f(x)$ 进行成组插值及成组微商。

设给定的 n 个结点为 $x_0<x_1<\Lambda<x_{n-1}$，其函数值为 $y_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$，第 2 种边界条件为：

$$\begin{cases} y_0^{''}=f^{''}(x_0) \\ y_{n-1}^{''}=f^{''}(x_{n-1}) \end{cases}$$

计算 n 个结点上的一阶导数值 $y_j^{'}(j=1,\ 2,\ \Lambda\ ,\ n-2)$ 的公式如下：

$$a_0=-0.5$$

$$b_0=3\cdot\frac{y_1-y_0}{2(x_1-x_0)}-y_0^{''}(x_1-x_0)/4$$

$$h_j=x_{j+1}-x_j,\quad j=0,\ 1,\ \Lambda\ ,\ n-2$$

$$\alpha_j=h_{j-1}/(h_{j-1}+h_j),\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$\beta_i=3\left[(1-\alpha_j)(y_j-y_{j-1})/h_{j-1}+\alpha_j(y_{j+1}-y_i)/h_j\right],\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$a_j=-\alpha_j/\left[2+(1-\alpha_j)a_{j-1}\right],\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$b_j=\left[\beta_j-(1-\alpha_j)b_{j-1}\right]/\left[2+(1-\alpha_j)a_{j-1}\right],\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$y_{n-1}^{'}=\left[3(y_{n-1}-y_{n-2})/h_{n-2}+y_{n-1}^{''}h_{n-2}/2-b_{n-2}\right]/(2+a_{n-2})$$

$$y_j^{'}=a_jy_{j+1}^{'}+b_j,\quad j=n-2,\ n-3,\ \Lambda\ ,\ 0$$

计算 n 个结点上的二阶导数值 $y_j^{''}(j=0,\ 1,\ \Lambda\ ,\ n-1)$：插值区间 $[x_0,\ x_{n-1}]$ 上的积分值。插值点 t 处的函数值、一阶导数值及二阶导数值的公式同 5.13 节。

2. 算法实现

根据上述算法，可以定义第 2 种边界条件的三次样条函数插值、微商与积分的 Visual Basic 函数 INSpline2，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INSpline2
'  功能：  实现第二种边界条件的三次样条函数插值、微商与积分
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
```

```
'         y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                 y(i) = f(x(i)), i=1,2,...,n
'         dy    - Double型一维数组，长度为n，返回时，存放n个给定点处的一阶导数值
'                 y'(i)，i=1,2,…n
'         ddy   - Double型一维数组，长度为n，调用时，ddy(1)存放给定区间的左端点处的
'                 二阶导数值，ddy(n)存放给定区间的右端点处的二阶导数值。返回时，存放n
'                 个给定点处的二阶导数值y''(i)，i=1,2,…n
'         m     - Integer型变量，指定插值点的个数
'         t     - Double型一维数组，长度为m，存放m个指定的插值点的值。要求
'                 x(1)<t(j)<x(n), j=1,2,…,m-1
'         z     - Double型一维数组，长度为m，存放m个指定的插值点处的函数值
'         dz    - Double型一维数组，长度为m，存放m个指定的插值点处的一阶导数值
'         ddz   - Double型一维数组，长度为m，存放m个指定的插值点处的二阶导数值
'  返回值：Double型，指定函数的x(1)到x(n)的定积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INSpline2(n As Integer, x() As Double, y() As Double, _
dy() As Double, ddy() As Double, m As Integer, t() As Double, _
z() As Double, dz() As Double, ddz() As Double) As Double
    Dim i As Integer, j As Integer
    Dim h0 As Double, h1 As Double, alpha As Double, beta As Double, _
         g As Double
    ReDim s(n) As Double

    ' 初值
    dy(1) = -0.5
    h0 = x(2) - x(1)
    s(1) = 3# * (y(2) - y(1)) / (2# * h0) - ddy(1) * h0 / 4#

    For j = 2 To n - 1
        h1 = x(j + 1) - x(j)
        alpha = h0 / (h0 + h1)
        beta = (1# - alpha) * (y(j) - y(j - 1)) / h0
        beta = 3# * (beta + alpha * (y(j + 1) - y(j)) / h1)
        dy(j) = -alpha / (2# + (1# - alpha) * dy(j - 1))
        s(j) = (beta - (1# - alpha) * s(j - 1))
        s(j) = s(j) / (2# + (1# - alpha) * dy(j - 1))
        h0 = h1
    Next j

    ' 一阶导数值
    dy(n) = (3# * (y(n) - y(n - 1)) / h1 + ddy(n) * h1 / 2# - _
             s(n - 1)) / (2# + dy(n - 1))
    For j = n - 1 To 1 Step -1
      dy(j) = dy(j) * dy(j + 1) + s(j)
    Next j

    For j = 1 To n - 1
        s(j) = x(j + 1) - x(j)
    Next j

    ' 二阶导数值
```

```
    For j = 1 To n - 1
       h1 = s(j) * s(j)
       ddy(j) = 6# * (y(j + 1) - y(j)) / h1 - 2# * (2# * dy(j) + _
                dy(j + 1)) / s(j)
    Next j

    h1 = s(n - 1) * s(n - 1)
    ddy(n) = 6# * (y(n - 1) - y(n)) / h1 + 2# * (2# * dy(n) + _
            dy(n - 1)) / s(n - 1)
    g = 0#
    For i = 1 To n - 1
       h1 = 0.5 * s(i) * (y(i) + y(i + 1))
       h1 = h1 - s(i) * s(i) * s(i) * (ddy(i) + ddy(i + 1)) / 24#
       g = g + h1
    Next i

    ' 插值
    For j = 1 To m
       If (t(j) >= x(n)) Then
          i = n - 2
       Else
          i = 0
          While (t(j) > x(i + 1))
             i = i + 1
          Wend
       End If

       h1 = (x(i + 1) - t(j)) / s(i)
       h0 = h1 * h1
       z(j) = (3# * h0 - 2# * h0 * h1) * y(i)
       z(j) = z(j) + s(i) * (h0 - h0 * h1) * dy(i)
       dz(j) = 6# * (h0 - h1) * y(i) / s(i)
       dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i)
       ddz(j) = (6# - 12# * h1) * y(i) / (s(i) * s(i))
       ddz(j) = ddz(j) + (2# - 6# * h1) * dy(i) / s(i)
       h1 = (t(j) - x(i)) / s(i)
       h0 = h1 * h1
       z(j) = z(j) + (3# * h0 - 2# * h0 * h1) * y(i + 1)
       z(j) = z(j) - s(i) * (h0 - h0 * h1) * dy(i + 1)
       dz(j) = dz(j) - 6# * (h0 - h1) * y(i + 1) / s(i)
       dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i + 1)
       ddz(j) = ddz(j) + (6# - 12# * h1) * y(i + 1) / (s(i) * s(i))
       ddz(j) = ddz(j) - (2# - 6# * h1) * dy(i + 1) / s(i)
    Next j

    ' 返回积分值
    INSpline2 = g

End Function
```

3. 示例

本例中，设初始条件同 5.13 **节的示例，其中边界条件改为：**

$$y_o'' = -0.279319,\ \ y_{n-1}'' = 0.0111560$$

调用函数 INSpline2 **计算各结点处的一阶及二阶导数值，在区间**[0.52，520.0]**内的积分值。并计算在** 8 **个插值点** 4.0，14.0，30.0，60.0，130.0，230.0，450.0，515.0 **处的函数值、一阶导数值及二阶导数值。取** n=12，m=8**。程序代码如下：**

```
Sub Main()
    Dim m As Integer, n As Integer, i As Double
    Dim s As Double
    Dim str As String

    ' 结点数
    n = 12

    ' 插值点数
    m = 8

    ' 分配内存
    ReDim x(n) As Double, y(n) As Double, dy(n) As Double, ddy(n) As Double
    ReDim t(m) As Double, z(m) As Double, dz(m) As Double, ddz(m) As Double

    ' 结点的值
    x(1) = 0.52
    x(2) = 8#
    x(3) = 17.95
    x(4) = 28.65
    x(5) = 50.65
    x(6) = 104.6
    x(7) = 156.6
    x(8) = 260.7
    x(9) = 364.4
    x(10) = 468#
    x(11) = 507#
    x(12) = 520#

    ' 结点的函数值
    y(1) = 5.28794
    y(2) = 13.84
    y(3) = 20.2
    y(4) = 24.9
    y(5) = 31.1
    y(6) = 36.5
    y(7) = 36.6
    y(8) = 31#
    y(9) = 20.9
    y(10) = 7.8
    y(11) = 1.5
```

```
    y(12) = 0.2

    ' 待插值点的值
    t(1) = 4#
    t(2) = 14#
    t(3) = 30#
    t(4) = 60#
    t(5) = 130#
    t(6) = 230#
    t(7) = 450#
    t(8) = 515#

    ' 二阶导数的左右端点值
    ddy(1) = -0.279319
    ddy(12) = 0.011156

    ' 插值
    s = INSpline2(n, x, y, dy, ddy, m, t, z, dz, ddz)

    str = "x(i)            y(i)            dy(i)           ddy(i)" & Chr(13)
    For i = 1 To n
        str = str & Format(x(i), "#####.00000") & "    " & _
              Format(y(i), "#####.00000") & "    " & _
              Format(dy(i), "#####.00000") & "    " & _
              Format(ddy(i), "#####.00000") & Chr(13)
    Next i

    str = str & Chr(13) & "s = " & Format(s, "#####.00000") & Chr(13) & Chr(13)

    str = str + "t(i)            z(i)
dz(i)           ddz(i)" & Chr(13)
    For i = 1 To m
        str = str & Format(t(i), "#####.00000")
& "    " & _
              Format(z(i), "#####.00000") & "
" & _
              Format(dz(i), "#####.00000") & "
" & _
              Format(ddz(i), "#####.00000") &
Chr(13)
    Next i

    MsgBox str

End Sub
```

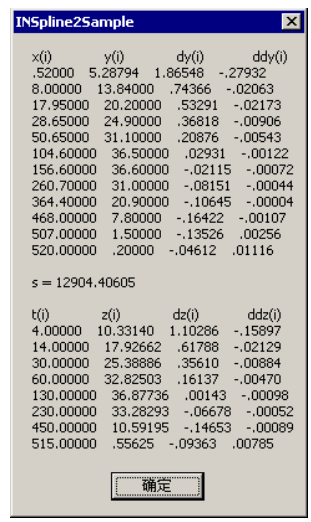

图 5.14　程序输出结果

其输出结果如图 5.14 **所示。**

5.15　第三种边界条件的三次样条函数插值、微商与积分

1. 算法原理

本节介绍第 3 种边界条件的三次样条函数插值、微商与积分算法，即给定 n 个结点 $x_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 上的函数值 $y_i = f(x_i)$ 以及第 3 种边界条件，利用三次样条函数，计算各给定结点上的一阶导数值 y_i'、二阶导数值 $y_i''(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 及插值区间 $[x_0,\ x_{n-1}]$ 上的积分近似值 $s=\int_{x_0}^{x_{n-1}} f(x)\,\mathrm{d}x$，并对函数 $f(x)$ 进行成组插值及成组微商。

设给定的 n 个结点为 $x_0 < x_1 < \Lambda < x_{n-1}$，其函数值为 $y_i(i=0,\ 1,\ \Lambda\ ,\ n-1)$，第 3 种边界条件为：

$$y_0 = y_{n-1},\quad y_0' = y_{n-1}',\quad y_0'' = y_{n-1}''$$

计算 n 个结点上的一阶导数值 $y_j'(j=1,\ 2,\ \Lambda\ ,\ n-2)$ 的公式如下：

$$a_0 = 0,\ b_0 = 1,\ c_1 = 0$$

$$h_0 = h_{n-1},\ y_0 - y_{-1} = y_{n-1} - y_{n-2}$$

$$h_j = x_{j+1} - x_j,\ j=0,\ 1,\ \Lambda\ ,\ n-2$$

$$\alpha_j = h_{j-1}/(h_{j-1}+h_j),\ j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$\beta_j = 3\left[(1-\alpha_j)(y_j - y_{j-1})/h_{j-1} + \alpha_j(y_{j+1}-y_j)/h_j\right],\ j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$a_j = -\alpha_{j-1}/\left[2+(1-\alpha_{j-1})a_{j-1}\right],\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$b_j = -(1-\alpha_{j-1})b_{j-1}/\left[2+(1-\alpha_{j-1})a_{j-1}\right],\quad j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$c_j = \left[\beta_{j-1}-(1-\alpha_{j-1})c_{j-1}\right]/\left[2+(1-\alpha_{j-1})a_{j-1}\right],\ j=1,\ 2,\ \Lambda\ ,\ n-2$$

$$u_{n-1} = 1,\quad v_{n-1} = 0$$

$$u_j = a_j u_{j+1} + b_j,\quad j=n-2,\ n-3,\ \Lambda\ ,\ 1$$

$$v_j = a_j v_{j+1} + c_j,\quad j=n-2,\ n-3,\ \Lambda\ ,\ 1$$

$$y_{n-2}' = \left[\beta_{n-2} - \alpha_{n-2}v_1 - (1-a_{n-2})v_{n-2}\right]/\left[2+\alpha_{n-2}u_1 + (1-\alpha_{n-2})u_{n-2}\right]$$

$$y_j' = u_{j+1}y_{n-2}' + v_{j+1},\quad j=0,\ 1,\ \Lambda\ ,\ n-3$$

$$y_{n-1}' = y_0'$$

计算 n 个给定结点上的二阶导数值 $y_j''(j=0,\ 1,\ \Lambda\ ,\ n-1)$。插值区间 $[x_0, x_{n-1}]$ 上的积分值、指定插值点 t 处的函数值、一阶导数值及二阶导数值的公式同 5.13 节。

2. 算法实现

根据上述算法，可以定义第 3 种边界条件的三次样条函数插值、微商与积分的 Visual Basic 函数 INSpline3，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
```

```
'  函数名：INSpline3
'  功能：  实现第三种边界条件的三次样条函数插值、微商与积分
'  参数：  n     - Integer型变量，给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定的n个结点的值x(i)
'          y     - Double型一维数组，长度为n，存放给定的n个等距结点的函数值y(i)，
'                  y(i) = f(x(i)), i=1,2,...,n. 要求y(1)=y(n)
'          dy    - Double型一维数组，长度为n，返回时，存放n个给定点处的一阶导数值
'                  y'(i)，i=1,2,…n
'          ddy   - Double型一维数组，长度为n，返回时，存放n个给定点处的二阶导数值
'                  y''(i)，i=1,2,…n
'          m     - Integer型变量，指定插值点的个数
'          t     - Double型一维数组，长度为m，存放m个指定的插值点的值。要求
'                  x(1)<t(j)<x(n), j=1,2,…,m-1
'          z     - Double型一维数组，长度为m，存放m个指定的插值点处的函数值
'          dz    - Double型一维数组，长度为m，存放m个指定的插值点处的一阶导数值
'          ddz   - Double型一维数组，长度为m，存放m个指定的插值点处的二阶导数值
'  返回值：Double型，指定函数的x(1)到x(n)的定积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INSpline3(n As Integer, x() As Double, y() As Double, _
dy() As Double, ddy() As Double, m As Integer, t() As Double, _
z() As Double, dz() As Double, ddz() As Double) As Double
    Dim i As Integer, j As Integer
    Dim y0 As Double, y1 As Double, h0 As Double, h1 As Double, _
       alpha As Double, beta As Double, u As Double, g As Double
    ReDim s(n) As Double

    ' 初值
    h0 = x(n) - x(n - 1)
    y0 = y(n) - y(n - 1)
    dy(1) = 0#
    ddy(1) = 0#
    ddy(n) = 0#
    s(1) = 1#
    s(n) = 1#

    For j = 2 To n
       h1 = h0
       y1 = y0
       h0 = x(j) - x(j - 1)
       y0 = y(j) - y(j - 1)
       alpha = h1 / (h1 + h0)
       beta = 3# * ((1# - alpha) * y1 / h1 + alpha * y0 / h0)
       If (j < n) Then
          u = 2# + (1# - alpha) * dy(j - 1)
          dy(j) = -alpha / u
          s(j) = (alpha - 1#) * s(j - 1) / u
          ddy(j) = (beta - (1# - alpha) * ddy(j - 1)) / u
       End If
    Next j

    ' 二阶导数值
```

```
For j = n - 1 To 2 Step -1
   s(j) = dy(j) * s(j + 1) + s(j)
   ddy(j) = dy(j) * ddy(j + 1) + ddy(j)
Next j

' 一阶导数值
dy(n - 1) = (beta - alpha * ddy(2) - (1# - alpha) * ddy(n - 1)) / _
           (alpha * s(2) + (1# - alpha) * s(n - 1) + 2#)
For j = 3 To n
   dy(j - 2) = s(j - 1) * dy(n - 1) + ddy(j - 1)
Next j

dy(n) = dy(1)
For j = 1 To n - 1
   s(j) = x(j + 1) - x(j)
Next j

For j = 1 To n - 1
   h1 = s(j) * s(j)
   ddy(j) = 6# * (y(j + 1) - y(j)) / h1 - 2# * (2# * dy(j) + _
            dy(j + 1)) / s(j)
Next j

h1 = s(n - 1) * s(n - 1)
ddy(n) = 6# * (y(n - 1) - y(n)) / h1 + 2# * (2# * dy(n) + _
          dy(n - 1)) / s(n - 1)
g = 0#
For i = 1 To n - 1
   h1 = 0.5 * s(i) * (y(i) + y(i + 1))
   h1 = h1 - s(i) * s(i) * s(i) * (ddy(i) + ddy(i + 1)) / 24#
   g = g + h1
Next i

' 插值
For j = 1 To m
   h0 = t(j)
   While (h0 >= x(n))
      h0 = h0 - (x(n) - x(1))
   Wend

   While (h0 < x(1))
      h0 = h0 + (x(n) - x(1))
   Wend

   i = 1
   While (h0 > x(i + 1))
      i = i + 1
   Wend

   u = h0
   h1 = (x(i + 1) - u) / s(i)
```

```
        h0 = h1 * h1
        z(j) = (3# * h0 - 2# * h0 * h1) * y(i)
        z(j) = z(j) + s(i) * (h0 - h0 * h1) * dy(i)
        dz(j) = 6# * (h0 - h1) * y(i) / s(i)
        dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i)
        ddz(j) = (6# - 12# * h1) * y(i) / (s(i) * s(i))
        ddz(j) = ddz(j) + (2# - 6# * h1) * dy(i) / s(i)
        h1 = (u - x(i)) / s(i)
        h0 = h1 * h1
        z(j) = z(j) + (3# * h0 - 2# * h0 * h1) * y(i + 1)
        z(j) = z(j) - s(i) * (h0 - h0 * h1) * dy(i + 1)
        dz(j) = dz(j) - 6# * (h0 - h1) * y(i + 1) / s(i)
        dz(j) = dz(j) + (3# * h0 - 2# * h1) * dy(i + 1)
        ddz(j) = ddz(j) + (6# - 12# * h1) * y(i + 1) / (s(i) * s(i))
        ddz(j) = ddz(j) - (2# - 6# * h1) * dy(i + 1) / s(i)
    Next j

    ' 返回积分值
    INSpline3 = g

End Function
```

3. 示例

给定间隔为10°**的** sin x **函数表，调用函数** INSpline3 **计算间隔为**5°**的** sin x **函数表，并计算其一阶、二阶导数值以及在一个周期内的积分值，取** n=37，m=36**。程序代码如下：**

```
Sub Main()
    Dim m As Integer, n As Integer, i As Double, j As Double
    Dim s As Double, u As Double
    Dim str As String

    ' 结点数
    n = 37

    ' 插值点数
    m = 36

    ' 分配内存
    ReDim x(n) As Double, y(n) As Double, dy(n) As Double, ddy(n) As Double
    ReDim t(m) As Double, z(m) As Double, dz(m) As Double, ddz(m) As Double

    ' 结点的值与结点的函数值
    For i = 1 To n
        x(i) = (i - 1) * 6.2831852 / 36#
        y(i) = Sin(x(i))
    Next i

    ' 待插值点的值
    For i = 1 To m
        t(i) = (0.5 + i - 1) * 6.2831852 / 36#
```

```
    Next i

    ' 插值
    s = INSpline3(n, x, y, dy, ddy, m, t, z, dz, ddz)

    str = "x(i)      y(i)=sin(x)      dy(i)=cos(x)      ddy(i)=-sin(x)" & _
          Chr(13) & Chr(13)
    str = str & Format(x(1), "#####0.00000") & "    " & _
          Format(y(1), "#####0.00000") & "    " & _
          Format(dy(1), "#####0.00000") & "    " & _
          Format(ddy(1), "#####0.00000") & Chr(13)
    For i = 1 To m
       u = t(i) * 36# / 0.62831852
       str = str & Format(u, "#####0.00000") & "    " & _
             Format(z(i), "#####0.00000") & "    " & _
             Format(dz(i), "#####0.00000") & "    " & _
             Format(ddz(i), "#####0.00000") & Chr(13)
       u = x(i + 1) * 36# / 0.62831852
       j = i + 1
       str = str & Format(u, "#####0.00000") & "    " & _
             Format(y(j), "#####0.00000") & "    " & _
             Format(dy(j), "#####0.00000") & "    " & _
             Format(ddy(j), "#####0.00000") & Chr(13)
    Next i

    str = str & Chr(13) & "s = " & s & Chr(13) & Chr(13)
    ' 在窗体的RichText控件中显示结果
    Form1.RichTextBox1.Text = str
    Form1.Show
End Sub
```

其输出结果如图 5.15 所示。注意,由于输出内容太多,在工程中加入了一个窗体 Form1,并在窗体中设计了一个 RichText 框,所有输出结果都显示在该框中。

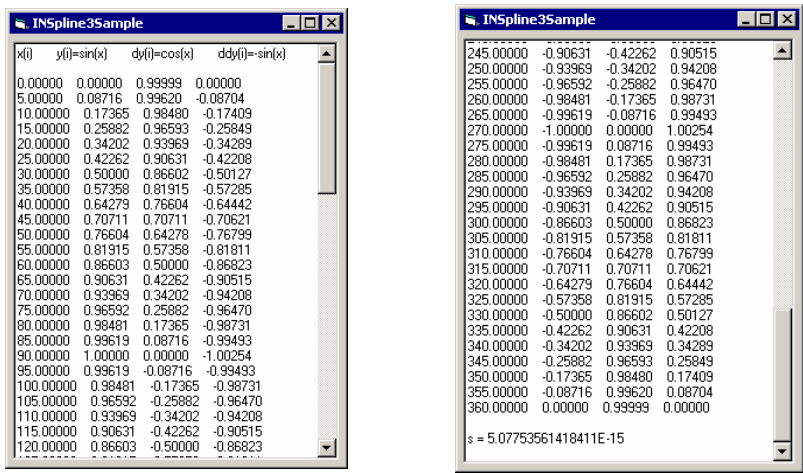

图 5.15 程序输出结果

5.16　二元三点插值

1. 算法原理

二元三点插值是指，对于给定矩形域上 $n\times m$ 个结点$(x_i, y_i)(i=0, 1, \cdots, n-1; j=0, 1, \cdots m-1)$**上的函数值 $z_{ij}=z(x_i, y_j)$，利用二元三点插值公式计算指定插值点**(u, v)**处的函数近似值 $w=z(u, v)$。**

设给定矩形域上 $n\times m$ 个结点在两个方向上的坐标分别为

$$x_0 < x_1 < \Lambda < x_{n-1}$$
$$y_0 < y_1 < \Lambda < y_{m-1}$$

相应的函数值为：

$$z_{ij}=z(x_i, y_j),\ i=0,\ 1,\ \Lambda,\ n-1;\ \ j=0,\ 1,\ \Lambda,\ m-1$$

选取最靠近插值点 (u, v) **的** 9 **个结点，其两个方向上的坐标分别为**

$$x_p < x_{p+1} < \Lambda < x_{p+2}$$
$$y_q < y_{q+1} < \Lambda < y_{q+2}$$

然后用二元三点插值公式

$$z(x, y)=\sum_{i=p}^{p+2}\sum_{j=q}^{q+2}\left(\prod_{\substack{k=p\\k\neq i}}^{p+2}\frac{x-x_k}{x_i-x_k}\right)\left(\prod_{\substack{l=q\\l\neq j}}^{q+2}\frac{y-y_l}{y_j-y_l}\right)z_{ij}$$

计算插值点 (u, v) **处的函数近似值。**

2. 算法实现

根据上述算法，可以定义二元三点插值插值的 Visual Basic **函数** INTqip**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INTqip
'  功能：  实现二元三点插值
'  参数：  n     - Integer型变量，x方向上给定结点的点数
'          m     - Integer型变量，y方向上给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定n x m 个结点x方向上的n个值x(i)
'          y     - Double型一维数组，长度为m，存放给定n x m 个结点y方向上的m个值y(i)
'          z     - Double型n x m二维数组，存放给定的n x m个结点的函数值z(i,j)，
'                  z(i,j) = f(x(i), y(j)), i=1,2,...,n, j=1,2,...,m
'          u     - Double型变量，存放插值点x坐标
'          v     - Double型变量，存放插值点y坐标
'  返回值：Double型，指定函数值f(u, v)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INTqip(n As Integer, m As Integer, x() As Double, _
y() As Double, z() As Double, u As Double, v As Double) As Double
    Dim nn As Integer, mm As Integer, ip As Integer, iq As Integer, _
         i As Integer, j As Integer, k As Integer, l As Integer
    Dim b(3) As Double, h As Double, w As Double
```

```
nn = 3
If (n <= 3) Then
   ip = 0
   nn = n
Else
   If (u <= x(2)) Then
      ip = 0
   Else
      If (u >= x(n - 1)) Then
         ip = n - 3
      Else
         i = 1
         j = n
         While (((i - j) <> 1) And ((i - j) <> -1))
            l = (i + j) / 2
            If (u < x(l)) Then
               j = l
            Else
               i = l
            End If
         Wend
         If (Abs(u - x(i)) < Abs(u - x(j))) Then
            ip = i - 2
         Else
            ip = i - 1
         End If
      End If
   End If
End If

mm = 3
If (m <= 3) Then
   iq = 0
   mm = m
Else
   If (v <= y(2)) Then
      iq = 0
   Else
      If (v >= y(m - 1)) Then
         iq = m - 3
      Else
         i = 1
         j = m
         While (((i - j) <> 1) And ((i - j) <> -1))
            l = (i + j) / 2
            If (v < y(l)) Then
               j = l
            Else
               i = l
            End If
```

```
            Wend
            If (Abs(v - y(i)) < Abs(v - y(j))) Then
               iq = i - 2
            Else
               iq = i - 1
            End If
         End If
      End If
   End If

   For i = 1 To nn
      b(i) = 0#
      For j = 1 To mm
         h = z(ip + i, iq + j)
         For k = 1 To mm
            If (k <> j) Then
               ' 二元拉格朗日公式
               h = h * (v - y(iq + k)) / (y(iq + j) - y(iq + k))
            End If
         Next k
         b(i) = b(i) + h
      Next j
   Next i

   w = 0#
   For i = 1 To nn
      h = b(i)
      For j = 1 To nn
         If (j <> i) Then
            ' 二元拉格朗日公式
            h = h * (u - x(ip + j)) / (x(ip + i) - x(ip + j))
         End If
      Next j
      w = w + h
   Next i

   ' 返回函数值
   INTqip = w

End Function
```

3. 示例

设二元函数

$$z(x,y)=\mathrm{e}^{-(x-y)}$$

取以下 6×5 **个结点：**

$$x_i=0.2i,\quad i=0,\ 1,\ \Lambda\ ,\ 5$$
$$y_i=0.25j,\quad j=0,\ 1,\ \Lambda\ ,\ 4$$

其函数值为：

$$z(i,\ j)=\mathrm{e}^{-(x_i-y_j)},\ i=0,\ 1,\ \Lambda,\ 5;\ \ j=0,\ 1,\ \Lambda,\ 4$$

调用函数 INTqip **来计算插值点**(0.9，0.8)**以及**(0.3，0.9)**处的函数近似值。程序代码如下：**

```
Sub Main()
    Dim m As Integer, n As Integer, i As Double, j As Double
    Dim u As Double, v As Double, w As Double
    Dim str As String

    ' x方向结点数
    n = 6

    ' y方向结点数
    m = 5

    ' 分配内存
    ReDim x(n) As Double, y(m) As Double, z(n, m) As Double

    ' 结点的值与结点的函数值
    For i = 1 To n
        x(i) = 0.2 * (i - 1)
    Next i

    For j = 1 To m
        y(j) = 0.25 * (j - 1)
    Next j

    For i = 1 To n
        For j = 1 To m
            z(i, j) = Exp(-(x(i) - y(j)))
        Next j
    Next i

    ' 待插值点的值
    u = 0.9
    v = 0.8
    ' 插值
    w = INTqip(n, m, x, y, z, u, v)
    str = str & "z(" & u & ", " & v & ") = " & w & Chr(13)

    ' 待插值点的值
    u = 0.3
    v = 0.9
    ' 插值
    w = INTqip(n, m, x, y, z, u, v)
    str = str & "z(" & u & ", " & v & ") = " & w

    MsgBox str

End Sub
```

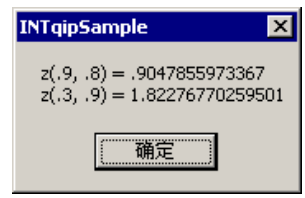

图 5.16 程序输出结果

其输出结果如图 5.16 **所示。**

5.17　二元全区间插值

1. 算法原理

二元全区间插值是指，对于给定矩形域上 $n\times m$ 个结点$(x_i, y_i)(i=0, 1, \cdots, n-1; j=0, 1, \cdots, m-1)$**上的函数值** $z_{ij}=z(x_i, y_j)$**，利用二元插值公式计算指定插值点** (u, v) **处的函数近似值** $w=z(u, v)$**。**

设给定矩形域上 $n\times m$ 个结点在两个方向上的坐标分别为：

$$x_0<x_1<\Lambda<x_{n-1}$$
$$y_0<y_1<\Lambda<y_{m-1}$$

相应的函数值为：

$$z_{ij}=z(x_i, y_j),\quad i=0, 1, \Lambda, n-1;\quad j=0, 1, \Lambda, m-1$$

计算插值点 (u, v) **处的函数值** $w=z(u, v)$**。**

以插值点 (u, v) **为中心，在** X **方向上，前后各取** 4 **个坐标：**

$$x_p<x_{p+1}<x_{p+2}<x_{p+3}<x_{p+4}<x_{p+5}<x_{p+6}<x_{p+7}$$

在 Y **方向上，前后也各取** 4 **个坐标：**

$$x_q<x_{q+1}<x_{q+2}<x_{q+3}<x_{q+4}<x_{q+5}<x_{q+6}<x_{q+7}$$

然后用二元插值公式

$$z(x, y)=\sum_{i=p}^{p+7}\sum_{j=q}^{q+7}\left(\prod_{\substack{k=p\\k\neq i}}^{p+7}\frac{x-x_k}{x_i-x_k}\right)\left(\prod_{\substack{l=q\\l\neq j}}^{q+7}\frac{y-y_l}{y_j-y_l}\right)z_{ij}$$

计算插值点 (u, v) **处的函数近似值。**

2. 算法实现

根据上述算法，可以定义二元全区间插值的 Visual Basic **函数** INLagrn2**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：InterpModule.bas
'  函数名：INLagrn2
'  功能：  实现二元全区间插值
'  参数：  n     - Integer型变量，x方向上给定结点的点数
'          m     - Integer型变量，y方向上给定结点的点数
'          x     - Double型一维数组，长度为n，存放给定n x m 个结点x方向上的n个值x(i)
'          y     - Double型一维数组，长度为m，存放给定n x m 个结点y方向上的m个值y(i)
'          z     - Double型n x m二维数组，存放给定的n x m个结点的函数值z(i,j)，
'                  z(i,j) = f(x(i), y(j)), i=1,2,...,n, j=1,2,...,m
'          u     - Double型变量，存放插值点x坐标
'          v     - Double型变量，存放插值点y坐标
```

```
' 返回值：Double型，指定函数值f(u, v)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function INLagrn2(n As Integer, m As Integer, x() As Double, _
y() As Double, z() As Double, u As Double, v As Double) As Double
    Dim ip As Integer, ipp As Integer, iq As Integer, iqq As Integer, _
         i As Integer, j As Integer, k As Integer, l As Integer
    Dim b(10) As Double, h As Double, w As Double

    If (u <= x(1)) Then
       ip = 1
       ipp = 4
    Else
       If (u >= x(n)) Then
          ip = n - 3
          ipp = n
       Else
          i = 1
          j = n
          While (((i - j) <> 1) And ((i - j) <> -1))
             l = (i + j) / 2
             If (u < x(l)) Then
                j = l
             Else
                i = l
             End If
          Wend
          ip = i - 3
          ipp = i + 4
       End If
    End If

    If (ip < 1) Then
       ip = 1
    End If

    If (ipp > n) Then
       ipp = n
    End If

    If (v <= y(1)) Then
       iq = 1
       iqq = 4
    Else
       If (v >= y(m)) Then
          iq = m - 3
          iqq = m
       Else
          i = 1
          j = m
          While (((i - j) <> 1) And ((i - j) <> -1))
             l = (i + j) / 2
```

```
            If (v < y(l)) Then
               j = l
            Else
               i = l
            End If
         Wend
         iq = i - 3
         iqq = i + 4
      End If
   End If

   If (iq < 1) Then
      iq = 1
   End If

   If (iqq > m) Then
      iqq = m
   End If

   For i = ip To ipp
      b(i - ip + 1) = 0#
      For j = iq To iqq
         h = z(i, j)
         For k = iq To iqq
            ' 二元拉格朗日公式
            If (k <> j) Then h = h * (v - y(k)) / (y(j) - y(k))
         Next k
         b(i - ip + 1) = b(i - ip + 1) + h
      Next j
   Next i

   w = 0#
   For i = ip To ipp
      h = b(i - ip + 1)
      For j = ip To ipp
        ' 二元拉格朗日公式
        If (j <> i) Then h = h * (u - x(j)) / (x(i) - x(j))
      Next j
      w = w + h
   Next i

   ' 返回函数值
   INLagrn2 = w

End Function
```

3. 示例

设二元函数

$$z(x,\ y)=\mathrm{e}^{-(x-y)}$$

取以下 11×11 **个结点：**

$$x_i = 0.1i, \quad i = 0,\ 1,\ \Lambda\ ,\ 10$$
$$y_i = 0.1j, \quad j = 0,\ 1,\ \Lambda\ ,\ 10$$

其函数值为：

$$z(i,\ j) = \mathrm{e}^{-(x_i - y\ )}, \quad i = 0,\ 1,\ \Lambda\ ,\ 10\ ;\ j = 0,\ 1,\ \Lambda\ ,\ 10$$

调用函数 INLagrn2 **计算插值点**(0.35，0.65)**以及**(0.45，0.55)**处的函数近似值。程序代码如下：**

```
Sub Main()
   Dim m As Integer, n As Integer, i As Double, j As Double
   Dim u As Double, v As Double, w As Double
   Dim str As String

   ' x方向结点数
   n = 11

   ' y方向结点数
   m = 11

   ' 分配内存
   ReDim x(n) As Double, y(m) As Double, z(n, m) As Double

   ' 结点的值与结点的函数值
   For i = 1 To n
      x(i) = 0.1 * (i - 1)
   Next i

   For j = 1 To m
      y(j) = x(j)
   Next j

   For i = 1 To n
      For j = 1 To m
         z(i, j) = Exp(-(x(i) - y(j)))
      Next j
   Next i

   ' 待插值点的值
   u = 0.35
   v = 0.65
   ' 插值
   w = INLagrn2(n, m, x, y, z, u, v)
   str = str & "z(" & u & ", " & v & ") = " & w & Chr(13)

   ' 待插值点的值
   u = 0.45
```

```
    v = 0.55
    ' 插值
    w = INLagrn2(n, m, x, y, z, u, v)
    str = str & "z(" & u & ", " & v & ") = " &
w

    MsgBox str

End Sub
```

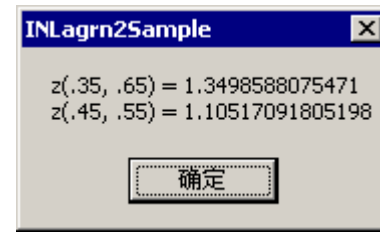

图 5.17　程序输出结果

其输出结果如图 5.17 **所示。**

第6章　数 值 积 分

数值积分即是求函数曲线在一定区间内与横坐标所围区域的面积，是微积分计算的重要内容。数值积分的基本方法是在积分区间内的一系列横坐标上把被积函数值相加。这种方法的要点是用最少数目的被积函数值尽可能精确地积分。

本章我们将介绍常用的数值积分算法，包括变步长梯形求积法、变步长辛卜生求积法、自适应梯形求积法、龙贝格求积法、计算一维积分的连分式法、高振荡函数求积法、勒让德-高斯求积法、拉盖尔-高斯求积法和埃尔米特-高斯求积法等。

6.1　变步长梯形求积法

1. 算法原理

计算定积分 $T=\int_a^b f(x)\,\mathrm{d}x$，基本步骤如下：

(1) 用梯形公式计算：

$$T_n=\frac{h}{2}\left[f(a)+f(b)\right]$$

其中 $n=1$，$h=b-a$。

(2) 用下列递推公式计算

$$T_{2n}=\frac{1}{2}T_n+\frac{h}{2}\sum_{i=0}^{n-1}f(x_i+0.5h)$$

$$2n\Rightarrow n,\ \ 0.5h\Rightarrow h$$

直至 $\left|T_{2n}-T_n\right|<\varepsilon$ 为止。

2. 算法实现

根据上述算法，可以定义变步长梯形求积法的 Visual Basic 函数 NITrapzd，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NITrapzd
'  功能：  用变步长梯形求积法求积，本函数需要调用计算函数f(x)值的函数Func，其形式为：
'          Function Func(x As Double) As Double
'  参数：  a     - Double型变量，积分下限
'          b     - Double型变量，积分上限，要求b>a
'          eps   - Double型变量，积分精度要求
'  返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NITrapzd(a As Double, b As Double, eps As Double) As Double
    Dim n As Integer, k As Integer
```

```
    Dim fa As Double, fb As Double, h As Double, t1 As Double, _
       p As Double, s As Double, x As Double, t As Double

    ' 积分区间端点的函数值
    fa = Func(a)
    fb = Func(b)

    ' 迭代初值
    n = 1
    h = b - a
    t1 = h * (fa + fb) / 2#
    p = eps + 1#

    ' 迭代计算
    While (p >= eps)
       s = 0#

       For k = 0 To n - 1
          x = a + (k + 0.5) * h
          s = s + Func(x)
       Next k

       t = (t1 + h * s) / 2#
       p = Abs(t1 - t)
       t1 = t
       n = n + n
       h = h / 2#
    Wend

    ' 返回满足精度的积分值
    NITrapzd = t

End Function
```

3. 示例

调用函数 NITrapzd **计算定积分**

$$T=\int_0^1 \mathrm{e}^{-x^2}\mathrm{d}x$$

取 ε =0.000001**。程序代码如下：**

```
Sub Main()
    Dim a As Double, b As Double, eps As Double, v As Double

    a = 0#
    b = 1#
    eps = 0.000001


    ' 计算积分
```

```
    v = NITrapzd(a, b, eps)

    MsgBox "积分值 = " & v

End Sub

` 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = Exp(-x * x)
End Function
```

其输出结果如图 6.1 所示。

图 6.1　程序输出结果

6.2　变步长辛卜生求积法

1. 算法原理

变步长辛卜生(Simpson)求积法是计算定积分 $S=\int_a^b f(x)\,\mathrm{d}x$ 的经典方法，其计算步骤如下：

(1)　用梯形公式计算 $T_n = h[f(a)+f(b)]/2$，其中 n=1，h=b-a。且令 $S_n = T_n$。

(2)　用变步长梯形法则计算

$$T_{2n} = \frac{1}{2}T_n + \frac{h}{2}\sum_{k=0}^{n-1} f(x_k + h/2)$$

(3)　用辛卜生求积公式计算：

$$S_{2n} = (4T_{2n} - T_n)/3$$

若 $\left|S_{2n}-S_n\right| \geq \varepsilon \geq \varepsilon$，则令 $2n \Rightarrow n,\ h/2 \Rightarrow h$，转到步骤(2)继续进行计算；否则结束，$S_{2n}$ 即为所求积分的近似值。其中 ε 为事先给定的求积精度。

2. 算法实现

根据上述算法，可以定义变步长辛卜生求积法的 Visual Basic 函数 NISimpson，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'   模块名：NIModule.bas
'   函数名：NISimpson
'   功能：  用变步长辛卜生求积法求积，本函数需要调用计算函数f(x)值的函数Func，其形式为：
```

```
'         Function Func(x As Double) As Double
'  参数：  a     - Double型变量，积分下限
'         b     - Double型变量，积分上限，要求b>a
'         eps   - Double型变量，积分精度要求
'  返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NISimpson(a As Double, b As Double, eps As Double) As Double
    Dim n As Integer, k As Integer
    Dim h As Double, t1 As Double, t2 As Double, s1 As Double, s2 As Double
    Dim ep As Double, p As Double, x As Double

    ' 迭代初值
    n = 1
    h = b - a
    t1 = h * (Func(a) + Func(b)) / 2#
    s1 = t1
    ep = eps + 1#

    ' 迭代计算
    While (ep >= eps)
        p = 0#

        For k = 0 To n - 1
            x = a + (k + 0.5) * h
            p = p + Func(x)
        Next k

        t2 = (t1 + h * p) / 2#
        s2 = (4# * t2 - t1) / 3#
        ep = Abs(s2 - s1)
        t1 = t2
        s1 = s2
        n = n + n
        h = h / 2#
    Wend

    ' 返回满足精度的积分值
    NISimpson = s2

End Function
```

3. 示例

调用函数 NISimpson **来计算定积分**

$$S=\int_0^1 \frac{1\mathrm{n}\,(1+x)}{1+x^2}\,\mathrm{d}x$$

取 ε =0.000001**。程序代码如下：**

```
Sub Main()
    Dim a As Double, b As Double, eps As Double, v As Double
```

```
    a = 0#
    b = 1#
    eps = 0.000001

    ' 计算积分
    v = NISimpson(a, b, eps)


    MsgBox "积分值 = " & v

End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = Log(1# + x) / (1# + x * x)
End Function
```

其输出结果如图 6.2 所示。

图 6.2 程序输出结果

6.3 自适应梯形求积法

1. 算法原理

计算被积函数 $f(x)$ 在积分区间内有强峰的定积分 $T=\int_a^b f(x)\,\mathrm{d}x$，过程如下：

(1) 将积分区间[a，b]分割成两个相等的子区间(称为 1 级子区间) $\Delta_0^{(1)}$ 和 $\Delta_1^{(1)}$，并在每一个子区间上分别用梯形公式计算积分近似值，设其结果分别为 $t_0^{(1)}$ 和 $t_1^{(1)}$。

(2) 将子区间 $\Delta_0^{(1)}$ 再分割为两个相等的子区间(称为 2 级子区间) $\Delta_0^{(2)}$ 和 $\Delta_1^{(2)}$，并在每一个子区间上也分别用梯形公式计算积分近似值，设分别为 $t_0^{(2)}$ 和 $t_1^{(2)}$。如果下列不等式

$$\left|t_0^{(1)}-(t_0^{(2)}+t_1^{(2)})\right|<\varepsilon/1.4$$

成立，则保留 $t_0^{(2)}$ 和 $t_1^{(2)}$。再将 $\Delta_1^{(1)}$ 也分割为两个相等的 2 级子区间 $\Delta_2^{(2)}$ 和 $\Delta_3^{(2)}$，其相应的积分近似值为 $t_2^{(2)}$ 和 $t_3^{(2)}$。

(3) 如果下列不等式

$$\left|t_1^{(1)}-(t_2^{(2)}+t_3^{(2)})\right|<\varepsilon/1.4$$

成立，则保留 $t_2^{(2)}$ 和 $t_3^{(2)}$。最后可得到满足精度要求的积分近似值：

$$t = t_0^{(2)} + t_1^{(2)} + t_2^{(2)} + t_3^{(2)}$$

(4) 如果上述不等式中有一个不成立，则将对应的 2 级子区间再分割成两个相等的 3 级子区间。在考虑 3 级子区间时，其精度要求变为 $\varepsilon/1.4^2$。

(5) 同样，对 3 级子区间中不满足精度要求的子区间又可以分割成两个相等的 4 级子区间，其精度要求变为 $\varepsilon/1.4^3$。依此类推，这个过程一直进行到在所考虑的所有子区间上都满足精度要求为止。

本算法为递归算法。

2. 算法实现

根据上述算法，可以定义自适应梯形求积法的 Visual Basic 函数 NIATrapzd，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NIATrapzd
'  功能：  用自适应梯形求积法求积，本函数需要调用计算函数f(x)值的函数Func，其形式为：
'          Function Func(x As Double) As Double
'  参数：  a     - Double型变量，积分下限
'          b     - Double型变量，积分上限，要求b>a
'          eps   - Double型变量，积分精度要求
'          d     - Double型变量，对积分区间进行分割的最小步长，当子区间的宽度小于d时，
'                  即使没有满足精度要求，也不再往下进行分割
'  返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NIATrapzd(a As Double, b As Double, eps As Double, _
d As Double) As Double
    Dim h As Double, t As Double, f0 As Double, f1 As Double, t0 As Double

    h = b - a
    t = 0#

    f0 = Func(a)
    f1 = Func(b)

    t0 = h * (f0 + f1) / 2#

    Call ppp(a, b, h, f0, f1, t0, eps, d, t)

    NIATrapzd = t

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：ppp
'  功能：  供函数NIATrapzd内部调用的递归过程
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub ppp(x0 As Double, x1 As Double, h As Double, f0 As Double, _
f1 As Double, t0 As Double, eps As Double, d As Double, t As Double)
    Dim x As Double, f As Double, t1 As Double, t2 As Double, _
```

```
        p As Double, g As Double, eps1 As Double

    x = x0 + h / 2#
    f = Func(x)
    t1 = h * (f0 + f) / 4#
    t2 = h * (f + f1) / 4#
    p = Abs(t0 - (t1 + t2))

    If ((p < eps) Or (h / 2# < d)) Then
        t = t + (t1 + t2)
    Else
        g = h / 2#
        eps1 = eps / 1.4

        Call ppp(x0, x, g, f0, f, t1, eps1, d, t)

        Call ppp(x, x1, g, f, f1, t2, eps1, d, t)
    End If

End Sub
```

3. 示例

调用函数 NIATrapzd **计算定积分**

$$T=\int_{-1}^{1}\frac{1}{1+25x^2}\,\mathrm{d}x$$

取 ε =0.000001，**最小步长为** 0.0001。**程序代码如下：**

```
Sub Main()
    Dim a As Double, b As Double, eps As Double, d As Double, v As Double

    a = -1#
    b = 1#
    eps = 0.000001
    d = 0.0001

    ' 计算积分
    v = NIASimpson(a, b, eps, d)

    MsgBox "积分值 = " & v

End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = 1# / (1# + 25# * x * x)
End Function
```

其输出结果如图 6.3 **所示。**

图 6.3　程序输出结果

6.4　龙贝格求积法

1. 算法原理

龙贝格(Romberg)求积法是计算定积分 $T=\int_a^b f(x)\,\mathrm{d}x$ 的最常用算法之一。

设 $T_m(h)$ 为步长为 h 时利用 $2m$-2 阶牛顿-柯特斯(Newton-Cotes)公式计算得到的结果；$T_m(\frac{h}{2})$ 为将步长 h 减半后用 $2m$-2 阶牛顿-柯特斯公式计算得到的结果。将它们进行线性组合，便得到步长为 h 的 $2m$ 阶牛顿-柯特斯公式，即：

$$T_{m+1}(h)=\frac{4^m T_m(h/2)-T_m(h)}{4^m-1}$$

其中，$T_1(h)$ 为步长为 h 时的梯形公式计算得到的结果，$T_1(h/2)$ 为步长为 $h/2$ 时的梯形公式计算得到的结果。

在实际进行计算时，龙贝格求积法按下表所示的计算格式进行，直到 $|T_{m+1}(h)-T_m(h)|<\varepsilon$ 为止。

梯形法则	二阶公式	四阶公式	六阶公式	八阶公式	…
$T_1(h)$					
$T_1(\frac{h}{2})$	$T_2(h)$				
$T_1(\frac{h}{2^2})$	$T_2(\frac{h}{2})$	$T_3(h)$			
$T_1(\frac{h}{2^3})$	$T_2(\frac{h}{2^2})$	$T_3(\frac{h}{2})$	$T_4(h)$		
$T_1(\frac{h}{2^4})$	$T_2(\frac{h}{2^3})$	$T_3(\frac{h}{2^2})$	$T_4(\frac{h}{2})$	$T_5(h)$	…
⋮	⋮	⋮	⋮	⋮	

本算法中，最多可以算到 m=10，如果此时还达不到精度要求，就取 T_{10} 作为最后结果。

2. 算法实现

根据上述算法，可以定义龙贝格求积法的 Visual Basic 函数 NIRomberg，其代码如下：

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NIRomberg
'  功能：  用龙贝格求积法求积，本函数需要调用计算函数f(x)值的函数Func，其形式为：
'          Function Func(x As Double) As Double
'  参数：  a     - Double型变量，积分下限
'          b     - Double型变量，积分上限，要求b>a
'          eps   - Double型变量，积分精度要求
'  返回值：Double型，积分值
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NIRomberg(a As Double, b As Double, eps As Double) As Double
```

```
    Dim m As Integer, n As Integer, i As Integer, k As Integer
    Dim y(10) As Double, h As Double, ep As Double, p As Double, _
       x As Double, s As Double, q As Double

    ' 初值
    h = b - a
    y(1) = h * (Func(a) + Func(b)) / 2#
    m = 1
    n = 1
    ep = eps + 1#

    ' 循环计算
    While ((ep >= eps) And (m <= 9))

       p = 0#

       For i = 0 To n - 1
          x = a + (i + 0.5) * h
          p = p + Func(x)
       Next i

       p = (y(1) + h * p) / 2#
       s = 1#

       For k = 1 To m
          s = 4# * s
          q = (s * p - y(k)) / (s - 1#)
          y(k) = p
          p = q
       Next k

       ep = Abs(q - y(m))
       m = m + 1
       y(m) = q
       n = n + n
       h = h / 2#
    Wend

    ' 求得满意的结果，返回
    NIRomberg = q

End Function
```

3. 示例

调用函数 NIRomberg **计算定积分**

$$T=\int_0^1 \frac{x}{4+x^2}\,\mathrm{d}x$$

取 ε =0.000001。**程序代码如下：**

```
Sub Main()
    Dim a As Double, b As Double, eps As Double, v As Double

    a = 0
    b = 1#
    eps = 0.000001

    ' 计算积分
    v = NIRomberg(a, b, eps)

    MsgBox "积分值 = " & v

End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
Func = x / (4# + x * x)
End Function
```

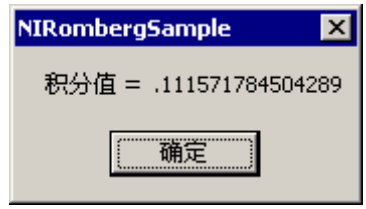

图 6.4　程序输出结果

其输出结果如图 6.4 所示。

6.5　计算一维积分的连分式法

1. 算法原理

计算定积分 $s=\int_a^b f(x)\,\mathrm{d}x$ 的算法如下：

利用变步长梯形法计算出步长分别为

$$h_i=(b-a)/2^i,\ i=0,\ 1,\ \Lambda$$

的一系列积分近似值 $s_i\,(i=0,\ 1,\ \Lambda\,)$。显然，积分近似值 s 是步长 h 的函数 $s(h)$。

将函数 $s(h)$用如下连分式表示：

$$s(h)=b_0+\cfrac{h-h_0}{b_1+\cfrac{h-h_1}{b_2+\cdots+\cfrac{h-h_{i-1}}{b_i+\cdots}}}$$

其中，b_0，b_1，Λ，b_i，Λ 可以由一系列的积分近似值 $(h_i,\ s_i)(i=0,\ 1,\ \Lambda\,)$ 来确定，其计算公式如下：

$$b_0=s_0$$

$$b_1=\frac{h_1-h_0}{s_1-b_0}$$

$$b_{21}=\frac{h_2-h_0}{s_2-b_0},\ b_2=\frac{h_2-h_1}{b_{21}-b_1}$$

$$b_{31}=\frac{h_3-h_0}{s_3-b_0},\ b_{32}=\frac{h_3-h_1}{b_{31}-b_1},\ b_3=\frac{h_3-h_2}{b_{32}-b_2},$$

一般来说，如果已知 b_0，b_1，Λ，b_{i-1}，则在计算出一个新的积分近似值点 (h_i,s_i) 后，

可以用以下递推公式计算 b_i：

$$\begin{cases} b_{i1} = s_i \\ b_{ij} = \dfrac{h_i - h_{j-1}}{b_{i,j-1} - b_{j-1,j-1}}, \quad j = 1, 2, \Lambda, i \\ b_i = b_{ii} \end{cases}$$

当 h=0 时，$s(0)$即为积分的精确值。即：

$$s = s(0) = b_0 - \cfrac{h_0}{b_1 - \cfrac{h_1}{b_2 - \cdots - \cfrac{h_{i-1}}{b_i - \cdots}}}$$

在实际进行计算时，一般取到 7 节就能满足精度要求。在本函数中，最多可以取到 10 节，如果此时还不满足精度要求，就直接取当前的积分近似值。

2. 算法实现

根据上述算法，可以定义计算一维积分的连分式法的 Visual Basic 函数 NIPq，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' 模块名：NIModule.bas
' 函数名：NIPq
' 功能：  用连分式法计算一维积分，本函数需要调用计算函数f(x)值的函数Func，其形式为：
'         Function Func(x As Double) As Double
' 参数：  a     - Double型变量，积分下限
'         b     - Double型变量，积分上限，要求b>a
'         eps   - Double型变量，积分精度要求
' 返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NIPq(a As Double, b As Double, eps As Double) As Double
    Dim m As Integer, n As Integer, k As Integer, l As Integer, j As Integer
    Dim h(10) As Double, bb(10) As Double, hh As Double, t1 As Double, _
        s1 As Double
    Dim ep As Double, s As Double, x As Double, t2 As Double, g As Double

    ' 初值
    m = 1
    n = 1
    hh = b - a
    h(1) = hh
    t1 = hh * (Func(a) + Func(b)) / 2#
    s1 = t1
    bb(1) = s1
    ep = 1# + eps

    ' 循环计算
    While ((ep >= eps) And (m <= 9))
        s = 0#
        For k = 0 To n - 1
            x = a + (k + 0.5) * hh
```

```
            s = s + Func(x)
        Next k

        t2 = (t1 + hh * s) / 2#
        m = m + 1
        h(m) = h(m - 1) / 2#
        g = t2
        l = 0
        j = 2

        While ((l = 0) And (j <= m))
            s = g - bb(j - 1)

            If (Abs(s) + 1# = 1#) Then
                l = 1
            Else
                g = (h(m) - h(j - 1)) / s
            End If

            j = j + 1
        Wend

        bb(m) = g

        If (l <> 0) Then bb(m) = 1E+35

        g = bb(m)

        For j = m To 2 Step -1
           g = bb(j - 1) - h(j - 1) / g
        Next j

        ep = Abs(g - s1)
        s1 = g
        t1 = t2
        hh = hh / 2#
        n = n + n
    Wend

    ' 结果返回
    NIPq = g

End Function
```

3. 示例

调用函数 NIPq **计算定积分**

$$s=\int_0^{4.3} \mathrm{e}^{-x^2}\mathrm{d}x$$

取 ε =0.000001**。程序代码如下：**

```
Sub Main()
```

```
    Dim a As Double, b As Double, eps As Double, v As Double

    a = 0
    b = 4.3
    eps = 0.000001

    ' 计算积分
    v = NIPq(a, b, eps)

    MsgBox "积分值 = " & v
End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = Exp(-x * x)
End Function
```

图 6.5 程序输出结果

其输出结果如图 6.5 所示。

6.6 高振荡函数求积法

1. 算法原理

当 m 较大时，积分 $s_1(m)=\int_a^b f(x)\cos mx\,\mathrm{d}x$ 与 $s_2(m)=\int_a^b f(x)\sin mx\,\mathrm{d}x$ 称为高振荡积分。分部积分法是计算高振荡函数的积分的重要方法。

令：$s(m)=\int_a^b f(x)\,e^{\mathrm{j}mx}\,\mathrm{d}x$

其中，$e^{\mathrm{j}mx}=\cos mx+\mathrm{j}\sin mx$，则有：

$$s(m)=s_1(m)+\mathrm{j}s_2(m)$$

反复利用分部积分法可以得到

$$s(m)=\int_a^b f(x)\,e^{mx\mathrm{j}}\mathrm{d}x$$

$$\approx-\sum_{k=0}^{n-1}\left(\frac{\mathrm{j}}{m}\right)^{k+1}\left[\left(f^{(k)}(b)\cos mb-f^{(k)}(a)\cos ma\right)+\mathrm{j}\left(f^{(k)}(b)\sin mb-f^{(k)}(a)\sin ma\right)\right]$$

分离出实部和虚部后就得到：

$$s_1(m)=\int_a^b f(x)\cos mx\,\mathrm{d}x$$

$$\approx\sum_{k=0}^{n-1}\frac{1}{m^{k+1}}\left[\left(f^{(k)}(b)\sin\left(\frac{k\pi}{2}+mb\right)-f^{(k)}(a)\sin\left(\frac{k\pi}{2}+ma\right)\right)\right]$$

$$s_2(m)=\int_a^b f(x)\sin mx\,\mathrm{d}x$$

$$\approx\sum_{k=0}^{n-1}\frac{-1}{m^{k+1}}\left[\left(f^{(k)}(b)\cos\left(\frac{k\pi}{2}+mb\right)-f^{(k)}(a)\cos\left(\frac{k\pi}{2}+ma\right)\right)\right]$$

其中，

$$\sin\left(\frac{k\pi}{2}+x\right)=\begin{cases}\sin x, & \mathrm{mod}(k,4)=0\\ \cos x, & \mathrm{mod}(k,4)=1\\ -\sin x, & \mathrm{mod}(k,4)=2\\ -\cos x, & \mathrm{mod}(k,4)=3\end{cases}$$

$$\cos\left(\frac{k\pi}{2}+x\right)=\begin{cases}\cos x, & \mathrm{mod}(k,4)=0\\ -\sin x, & \mathrm{mod}(k,4)=1\\ -\cos x, & \mathrm{mod}(k,4)=2\\ \sin x, & \mathrm{mod}(k,4)=3\end{cases}$$

n 的大小可根据 $f(x)$ 及 m 的大小而定。一般取 n=3 左右。

2. 算法实现

根据上述算法，可以定义分步积分法的 Visual Basic **函数** NIPart**，其代码如下：**

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NIPart
'  功能：  用分步积分法计算高振荡函数的积分
'  参数：  a  - Double型变量，积分下限
'          b  - Double型变量，积分上限，要求 b>a
'          m  - Double型变量，被积函数中振荡函数的角频率
'          n  - 给定积分区间两端点上的导数最高阶数＋1
'          fa - Double型一维数组，长度为n，存放f(x)在积分区间端点x=a处的各阶导数值
'          fb - Double型一维数组，长度为n，存放f(x)在积分区间端点x=b处的各阶导数值
'          s  - Double型一维数组，长度为2，其中s(1)返回f(x)cos(mx)在积分区间的积分
'               值，s(2) 返回f(x)sin(mx)在积分区间的积分值
'  返回值：无
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub NIPart(a As Double, b As Double, m As Integer, n As Integer, _
fa() As Double, fb() As Double, s() As Double)
    Dim mm As Long, k As Integer, j As Integer
    Dim sa(4) As Double, sb(4) As Double, ca(4) As Double, cb(4) As Double
    Dim sma As Double, smb As Double, cma As Double, cmb As Double

    ' 初值
    sma = Sin(m * a)
    smb = Sin(m * b)
    cma = Cos(m * a)
    cmb = Cos(m * b)

    sa(1) = sma
    sa(2) = cma
    sa(3) = -sma
    sa(4) = -cma
    sb(1) = smb
    sb(2) = cmb
    sb(3) = -smb
    sb(4) = -cmb
```

```
    ca(1) = cma
    ca(2) = -sma
    ca(3) = -cma
    ca(4) = sma
    cb(1) = cmb
    cb(2) = -smb
    cb(3) = -cmb
    cb(4) = smb

    s(1) = 0#
    s(2) = 0#

    mm = 1

    ' 循环计算
    For k = 0 To n - 1
       j = k
       While (j >= 4)
          j = j - 4
       Wend

       mm = mm * m
       s(1) = s(1) + (fb(k + 1) * sb(j + 1) - _
                fa(k + 1) * sa(j + 1)) / (1# * mm)
       s(2) = s(2) + (fb(k + 1) * cb(j + 1) - _
                fa(k + 1) * ca(j + 1)) / (1# * mm)
    Next k

    s(2) = -s(2)

End Sub
```

3. 示例

调用函数 NIPart **计算下列高振荡积分**

$$s_1 = \int_0^{2\pi} x\cos x\cos 30x \,\mathrm{d}x$$

与

$$s_2 = \int_0^{2\pi} x\cos \sin 30x \,\mathrm{d}x$$

其中 a=0.0，b=6.2831852，m=30。

取 n=4 $\quad f(x) = x\cos x, \quad f'(x) = \cos x - x\sin x$

$$f''(x) = -2\sin x - x\cos x, \quad f'''(x) = -3\cos x + x\sin x$$

则有，

$$fa(0) = f^{(0)}(0) = 0.0, \quad fa(1) = f^{(1)}(0) = 1.0$$

$$fa(2) = f^{(2)}(0) = 0.0, \quad fa(3) = f^{(3)}(0) = -3.0$$

$$fb(0) = f^{(0)}(2\pi) = 6.2831852, \quad fb.(1) = f^{(1)}(2\pi) = 1.0$$

$fb(2) = f^{(2)}(0) = -6.2831852,$　　$fb(3) = f^{(3)}(2\pi) = -3.0$

程序代码如下：

```
Sub Main()
    Dim a As Double, b As Double
    Dim m As Integer, n As Integer
    Dim s(2) As Double, fa(4) As Double, fb(4) As Double

    fa(1) = 0#
    fa(2) = 1#
    fa(3) = 0#
    fa(4) = -3#

    fb(1) = 6.2831852
    fb(2) = 1#
    fb(3) = -6.2831852
    fb(4) = -3#

    a = 0#
    b = 6.2831852
    m = 30
    n = 4

    ' 计算积分
    Call NIPart(a, b, m, n, fa, fb, s)

    MsgBox "s(1) = " & s(1) & Chr(13) & "s(2) = " & s(2)

End Sub
```

图 6.6　程序输出结果

其输出结果如图 6.6 **所示。**

6.7　勒让德-高斯求积法

1. 算法原理

勒让德-高斯(Legendre-Guass)**求积法计算定积分** $G = \int_a^b f(x)\,\mathrm{d}x$ **的算法如下：**

对积分变量 x **作变换** $x = \frac{b-a}{2}t + \frac{b+a}{2}$，**将原积分化为区间**[-1，1]**上的积分，即：**

$$G = \int_a^b f(x)\,\mathrm{d}x = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)\mathrm{d}t = \frac{b-a}{2}\int_{-1}^{1}\varphi(t)\,\mathrm{d}t$$

根据差值求积公式有：

$$\int_{-1}^{1}\varphi(t)\,\mathrm{d}t = \sum_{k=0}^{n-1}\lambda_k\ \varphi(t_k)$$

其中， $t_k\,(k = 0,\ 1,\ \Lambda\ ,\ n-1)$ **为区间**[-1，1]**上的** n **个求积结点，且**

$$\lambda_k = \int_{-1}^{1} A_k(t)\,\mathrm{d}t$$

$$A_k(t) = \prod_{\substack{j=0 \\ j\neq k}}^{n-1} [(t-t_j)/(t_k-t_j)]$$

如果 n **个结点** $t_k\ (k=0,\ 1,\ 2,\ \Lambda,\ n-1)$ **取勒让德**(Legendre)**多项式**

$$\frac{1}{2^n n!}\frac{d^n}{dt^n}\left[\left(t^2-1\right)^n\right]$$

在区间[-1，1]**上的** n **个零点，则上述差值求积公式称为勒让德-高斯求积公式，其代数精确度为** $2n-1$**。**

下表列出了 n **从** 2 **到** 10 **之间的求积结点** t_k **和求积系数** λ_k **。**

n	求积结点 t_k	求积系数 λ_k
2	±0.5773502692	1.0
3	0 ±0.7745966692	0.8888888889 0.5555555556
4	±0.3399810436 ±0.8611363116	0.6521451549 0.3478548451
5	0.0 ±0.5384693101 ±0.9061798459	0.5688888889 0.4786286705 0.2369268851
6	±0.2386191861 ±0.6612093865 ±0.9324695142	0.4679139346 0.3607615731 0.1713244924
7	0.0 ±0.4058451514 ±0.7415311856 ±0.9491079123	0.4179591837 0.3818300505 0.2797053915 0.1294849662
8	±0.1834346425 ±0.5255324099 ±0.7966664774 ±0.9602898565	0.3626837834 0.3137066459 0.2223810345 0.1012285363
9	0.0 ±0.3242534234 ±0.6133714327 0.8360311073 ±0.9681602395	0.3302393550 0.3123470070 0.2606106964 0.1806481607 0.0812743884
10	±0.1488743390 ±0.4333953941 ±0.6794095683 ±0.8650633667 ±0.9739065285	0.2955242247 0.2692667193 0.2190863625 0.1494513492 0.0666713443

在本函数中，取 n=5，并采用变步长求积法。

2. **算法实现**

根据上述算法，可以定义勒让德-高斯求积法的 Visual Basic **函数** NILgdGauss**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NILgdGauss
'  功能：  用勒让德-高斯求积法计算一维积分，本函数需要调用计算函数f(x)值的函数Func，
'          其形式为：
'          Function Func(x As Double) As Double
'  参数：  a     - Double型变量，积分下限
'          b     - Double型变量，积分上限，要求 b>a
'          eps   - Double型变量，积分精度要求
'  返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NILgdGauss(a As Double, b As Double, eps As Double) As Double
    Dim m As Long, i As Integer, j As Integer
    Dim s As Double, p As Double, ep As Double, h As Double, _
        aa As Double, bb As Double
    Dim w As Double, x As Double, g As Double
    Dim t(5) As Double, c(5) As Double

    ' 勒让德-高斯求积系数
    t(1) = -0.9061798459
    t(2) = -0.5384693101
    t(3) = 0#
    t(4) = 0.5384693101
    t(5) = 0.9061798459

    c(1) = 0.2369268851
    c(2) = 0.4786286705
    c(3) = 0.5688888889
    c(4) = 0.4786286705
    c(5) = 0.2369268851

    ' 初值
    m = 1
    h = b - a
    s = Abs(0.001 * h)
    p = 1E+35
    ep = eps + 1#

    ' 循环计算
    While ((ep >= eps) And (Abs(h) > s))
        g = 0#

        For i = 1 To m
            aa = a + (i - 1#) * h
```

```
            bb = a + i * h
            w = 0#

            For j = 1 To 5
                x = ((bb - aa) * t(j) + (bb + aa)) / 2#
                w = w + Func(x) * c(j)
            Next j

            g = g + w
        Next i

        g = g * h / 2#
        ep = Abs(g - p) / (1# + Abs(g))
        p = g
        m = m + 1
        h = (b - a) / m
    Wend

    ' 返回结果
    NILgdGauss = g

End Function
```

3. 示例

调用函数 NILgdGauss **计算定积分**

$$G=\int_{2.5}^{8.4}(x^2+\sin\ x)\ \mathrm{d}x$$

取 ε =0.000001**。程序代码如下：**

```
Sub Main()
    Dim a As Double, b As Double, eps As Double, v As Double

    a = 2.5
    b = 8.4
    eps = 0.000001

    ' 计算积分
    v = NILgdGauss(a, b, eps)

    MsgBox "积分值 = " & v

End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = x * x + Sin(x)
End Function
```

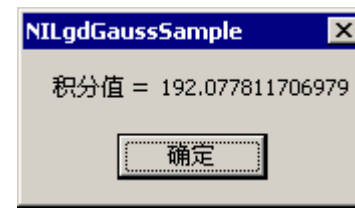

图 6.7　程序输出结果

其输出结果如图 6.7 **所示。**

6.8　拉盖尔-高斯求积法

1. 算法原理

用拉盖尔-高斯(Laguerre-Gauss)求积法计算半无限区间$[0,\infty]$上的积分

$$G=\int_0^{\infty} f(x)\,\mathrm{d}x$$

的算法如下：

设半无限区间$[0,\ \infty]$上的积分为：

$$G=\int_0^{\infty} f(x)\,\mathrm{d}x$$

n 点拉盖尔－高斯求积公式为：

$$G=\sum_{i=0}^{n-1}\lambda_i f(x_i)$$

其中，$x_i\,(i=0,\ 1,\ \Lambda\ ,\ n-1)$ 为 n 阶拉盖尔多项式

$$L_n(x)=\mathrm{e}^x\frac{\mathrm{d}^n}{\mathrm{d}x^n}(x^n\mathrm{e}^{-x}),\ \ 0\le x\le+\infty$$

的 n 个零点；λ_i 为求积系数。

在本函数中，取 n=5。5 阶拉盖尔-高斯求积公式的结点为：

$x_0=0.26355990$　　$x_1=1.41340290$　　$x_2=3.59642600$

$x_3=7.08580990$　　$x_4=12.64080000$

求积系数为：

$\lambda_0=0.6790941054$　　$\lambda_1=1.638487956$　　$\lambda_2=2.769426772$

$\lambda_3=4.315944000$　　$\lambda_4=7.104896230$

本方法特别适应于计算如下形式的积分：

$$\int_0^{\infty}\mathrm{e}^{-}\ \ddot{g}(x)\,\mathrm{d}x$$

2. 算法实现

根据上述算法，可以定义拉盖尔-高斯求积法的 Visual Basic 函数 NILgreGauss，其代码如下：

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NILgreGauss
'  功能：  用拉盖尔-高斯求积法计算一维积分，本函数需要调用计算函数f(x)值的函数Func，
'          其形式为：
'          Function Func(x As Double) As Double
'  参数： 无
'  返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NILgreGauss() As Double
    Dim i As Integer
    Dim x As Double, g As Double
```

```
    Dim t(5) As Double, c(5) As Double

    ' 拉盖尔－高斯求积系数
    t(1) = 0.2635599
    t(2) = 1.4134029
    t(3) = 3.596426
    t(4) = 7.0858099
    t(5) = 12.6408

    c(1) = 0.6790941054
    c(2) = 1.638487956
    c(3) = 2.769426772
    c(4) = 4.315944
    c(5) = 7.10489623

    ' 循环计算
    g = 0#
    For i = 1 To 5
        x = t(i)
        g = g + c(i) * Func(x)
    Next i

    ' 返回结果
    NILgreGauss = g

End Function
```

3. 示例

调用函数 NILgreGauss **计算半无限区间的积分**

$$G = \int_0^{\infty} x\,\mathrm{e}^{-x}\,\mathrm{d}x$$

程序代码如下：

```
Sub Main()
    Dim v As Double

    ' 计算积分
    v = NILgreGauss()

    MsgBox "积分值 = " & v

End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = x * Exp(-x)
End Function
```

其输出结果如图 6.8 **所示。**

图 6.8　程序输出结果

6.9　埃尔米特-高斯求积法

1. **算法原理**

用埃尔米特-高斯(Hermite-Gauss)**求积公式计算无限区间** $(-\infty, +\infty)$ **上的积分**

$$G = \int_0^{\infty} f(x)\,\mathrm{d}x$$

的算法如下：

设无限区间 $(-\infty, +\infty)$ **上的积分为：**

$$G = \int_0^{\infty} f(x)\,\mathrm{d}x$$

n **点埃尔米特-高斯求积公式为：**

$$G = \sum_{i=0}^{n-1} \lambda_i f(x_i)$$

其中， $x_i\,(i = 0, 1, \Lambda, n-1)$ **为** n **阶埃尔米特多项式**

$$H_n(x) = (-1)^n \mathrm{e}^{x^2} \frac{\mathrm{d}^n}{\mathrm{d}x^n}(\mathrm{e}^{-x^2}),\ \ 0 \le x \le +\infty$$

的 n **个零点；** λ_i **为求积系数。**

在本函数中，取 n=5.5 **阶埃尔米特-高斯求积公式的结点为：**

$x_0 = -2.02018200$，　　$x_1 = -0.95857190$，　　$x_2 = 0.0$

$x_3 = 0.95857190$，　　$x_4 = 2.02018200$

求积系数为

$\lambda_0 = 1.181469599$，　　$\lambda_1 = 0.9865791417$，　　$\lambda_2 = 0.0$

$\lambda_3 = 0.9865791417$，　　$\lambda_4 = 1.181469599$

本方法特别适合于计算如下形式的积分：

$$\int_{-\infty}^{\infty} \mathrm{e}^{-x^2} g(x)\,\mathrm{d}x$$

2. **算法实现**

根据上述算法，可以定义埃尔米特-高斯求积法的 Visual Basic **函数** NIHermiteGauss**，其代码如下：**

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  模块名：NIModule.bas
'  函数名：NIHermiteGauss
'  功能：  用埃尔米特-高斯求积法计算一维积分，本函数需要调用计算函数f(x)值的函数Func，
'          其形式为：
```

```
'         Function Func(x As Double) As Double
' 参数： 无
' 返回值：Double型，积分值
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function NIHermiteGauss() As Double
    Dim i As Integer
    Dim x As Double, g As Double
    Dim t(5) As Double, c(5) As Double

    ' 拉盖尔-高斯求积系数
    t(1) = -2.020182
    t(2) = -0.9585719
    t(3) = 0#
    t(4) = 0.9585719
    t(5) = 2.020182

    c(1) = 1.181469599
    c(2) = 0.9865791417
    c(3) = 0.9453089237
    c(4) = 0.9865791417
    c(5) = 1.181469599

    g = 0#

    ' 循环计算
    For i = 1 To 5
        x = t(i)
        g = g + c(i) * Func(x)
    Next i

    ' 返回结果
    NIHermiteGauss = g

End Function
```

3. 示例

调用函数 NIHermiteGauss **来计算无限区间的积分**

$$G=\int_{-\infty}^{\infty}x^2\ \mathrm{e}^{-\ ^2x}\mathrm{d}x$$

程序代码如下：

```
Sub Main()
    Dim v As Double

    ' 计算积分
    v = NIHermiteGauss()

    MsgBox "积分值 = " & v
```

```
End Sub

' 计算被积函数的函数值
Function Func(x As Double) As Double
    Func = x * x * Exp(-x * x)
End Function
```

其输出结果如图 6.9 **所示。**

图 6.9　程序输出结果