# 第一章

## 使用IDL命令

### IDL命令解析

#### 定位参数

Contour,peak,lon,lat,XStyle=1,YStyle=1, /Follow,$

Levels=vals,C\_Label=[1,0,1,0,0,1,1,0]

一个定位参数在命令名的右边有其确定的顺序。

例如，下面这两条命令的格式是不正确的并会导致错误。第一条命令的定位参数顺序被改变，第二条命令遗漏了第二个定位参数。

Contour, lon, peak, lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals, C\_Labels=[1,0,1,0,0,1,1,0]

Contour, peak, , lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals, C\_Labels=[1,0,1,0,0,1,1,0]

一般情况下，命令的定位参数必须给定参数，但并不总是如此。例如，在上面正确的命令中，peak是Contour命令必需的参数，但是lon和lat是可选定位参数。

#### 关键字参数

关键字参数能任何顺序出现在命令名右边。

Contour, peak, Level=vals, lon, XStyle=1, YStyle=1, $

/Follow, lat, C\_Lavels=[1,0,1,0,0,1,1,0]

关键字能设置为一个特定值（例如，XStyle=1），一个变量（例如，Levels=vals），一个数组（例如，C\_Labels=[1,0,1,0,0,1,1,0]），甚至可以用一个斜杠字符来设定（例如，/Follow）。

有些关键字有二进制特性。语法/Keyword等同于语法Keyword=1。

#### IDL过程和函数

IDL命令要么是过程，要么是函数。如下的IDL命令BytScl就是一个函数：

scaled=BytScl(image, Top=199, Min=0, Max=maxValue)

函数命令总是显示地返回一个值，这个数值必须赋予给一个变量。函数返回值可能是任一种IDL变量，包括数值，数组和结构。

函数和过程写在一起，例如

scaled=BytScl(image,Top=199,Min=0,Max=maxValue)

TV,scaled

第一个命令是一个函数命令，另一个是过程命令，此过程使用函数的返回值作为其定位参数，两个命令写成如下这样在IDL中很常见:

TV,BytScl(image,Top=199,Min=0,Max=maxValue)

在这种情况下，BytScl命令首先被执行并得到一个返回值，此返回值作为TV命令的定位参数。

#### 用IDL命令帮助

IDL>?

#### 创建命令日志

IDL>Journal, 'book\_commands'

随后所有在IDL命令行上键入的命令都将写入这个日志文件。

IDL>a=[3,5,7,3,6,9]

IDL>Help, a

IDL>Plot, a

当想关闭日志文件时，再次在IDL命令行键入Journal命令，如下：

IDL>Journal

当想再次执行日志文件中的命令时，在IDL命令行键入@作为开头字母

IDL>@book\_commands

如果每次建立日志文件时都想要一个唯一文件名，可用下列的IDL程序完成：

PRO Journal\_Unique

Journal, String('journal\_',Bin\_Data(SysTime()),'.pro',$

Format='(A,I4,5I2.2,A)')

END

然后，用Journal\_Unique代替Journal,就可以建立每次都具有唯一文件名的日志文件。

### 创建变量

变量名必须以字母开头。它们可以包括其它字母，数字，下划线，$。一个变量名最长可达255个字符。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 字节数 | 创建变量 | 数据类型函数 |
| 字节型 | 1 | Var=0B | thisVar=Byte(variable) |
| 16位有符号整型 | 2 | Var=0 | thisVar=Fix(variable) |
| 32位有符号长整型 | 4 | Var=0L | thisVar=Long(variable) |
| 64位有符号整型 | 8 | Var=0LL | thisVar=Long64(variable) |
| 16位无符号整型 | 2 | Var=0U | thisVar=UInt(variable) |
| 32位无符号长整型 | 4 | Var=0UL | thisVar=ULong(variable) |
| 64位无符号整型 | 8 | Var=0ULL | thisVar=Ulong64(variable) |
| 浮点型 | 4 | Var=0.0 | thisVar=Float(variable) |
| 双精度浮点型 | 8 | Var=0.0D | thisVar=Double(variable) |
| 复数 | 8 | Var=Complex(0.0,0.0) | thisVar=Complex(variable) |
| 双精度复数 | 16 | Var=Dcomplex(0.0D,0.0D) | thisVar=DComplex(variable) |
| 字符串 | 0-32767 | Var=’’或Var=”” | thisVar=String(variable) |
| 指针 | 4 | Var=Ptr\_New() | None |
| 对象 | 4 | Var=Obj\_New() | None |

**表1：IDL中的14种基本数据类型。表中显示了每种数据类型的字节数，创建变量的方法，用语数据类型之间强制转换的IDL函数**

创建100\*100初始值为零的字节型数组，输入：

IDL>array=BytArr(100,100)

创建一个有100个元素的浮点型矢量，初始数值为从0到99，输入：

IDL>vector=FIndGen(100)

#### 动态改变变量的属性

num=3 ; Initialize NUM as a scalar integer.

num=num\*5.2 ; Variable NUM changes to a float!

|  |  |  |
| --- | --- | --- |
| 数据类型 | 初始化函数 | 产生索引值的函数 |
| 字节型 | BytArr | BIndGen |
| 16位有符号整型 | IntArr | IndGen |
| 32位有符号长整型 | LonArr | LIndGen |
| 64位有符号整型 | Lon64Arr | L64IndGen |
| 16位无符号整型 | UIntArr | UIndGen |
| 32位无符号长整型 | ULonArr | ULIndGen |
| 64位无符号整型 | ULon64Arr | UL64IndGen |
| 浮点型 | FltArr | FIndGen |
| 双精度浮点型 | DblArr | DIndGen |
| 复数 | ComplexArr | CIndGen |
| 双精度复数 | DComplexArr | DCIndGen |
| 字符串 | StrArr | SIndGen |
| 指针 | PtrArr | None |
| 对象 | ObjArr | None |

**表2:　IDL函数可以创建矢量和多维数组，并将其每个元素初始为0或为它们本身的索引号码。**

### 使用矢量和数组

#### 创建矢量

IDL命令行：

IDL>vector=[1,2,3]

IDL>Help,vector

VECTOR INT =Array[3]

增加从第四个元素到矢量中：

IDL>vector=[vector,4]

IDL>Print,vector

1 2 3 4

#### 数组下标的应用

IDL>Print, vector[0:2]

1 2 3

用数组下标将另一个元素插入第二和第三个元素之间

IDL>vector=[vector[0:1],5,vector[2:3]]

IDL>Print, vector

1 2 5 3 4

建立一个值在0到50之间的6个元素浮点矢量

IDL>vector=FIndGen(6)\*10

IDL>Print,vector

0.000000 10.0000 20.0000 30.0000 40.0000 50.0000

#### 数组的建立

在IDL命令行中建立

IDL>array=[[1,2,3],[4,5,6]]

IDL>Print, array

1 2 3

4 5 6

等同于先建立一个矢量，然后Reform命令将此变形为一个三行二列的数组，

IDL>vector=IndGen(6)+1

IDL>array=Reform(vector,3,2)

IDL>Print, array

这表明矢量的排列是以行的顺序存储在IDL中的

#### 数组中元素的存取

读出刚建立的数组中位于第一列第二行的元素：

IDL>Print,array[0,1]

注意，下标的顺序先是列标，后是行标。

获得数组中的第四个元素：

IDL>Print, array[3]

存取数组中的第一，二，四和第六个元素：

IDL>indices=[0,1,3,5]

IDL>Print,array[indices]

#### 矢量和子数组的提取

IDL>data = RandomU(seed, 10, 20)

提取出第6-10列和第12-15行的数据：

IDL>subarray = data[5:9, 11:14]

将第8列的数据提取出来，使用下标\*代表所有的行，如下所示：

IDL>Plot, data[7,\*]

建立一个第14行的矢量：

IDL>vector = data[\*,13]

建立一个数据为数组中最后5行的数组：

IDL>subarray = data[\*,15:19]

子数组是一个10列\*5行的数组。

用数组的最后5列建立一个子数组：

IDL>subarray = data[5:\*,\*]

### 使用IDL图形窗口

#### 图形窗口的建立

建立并启动一个窗口：

IDL>Window

窗口的标题栏中有一个0，这是此窗口的索引号。当图形窗口建立后，每个图形窗口都有唯一的一个图形窗口索引号。在IDL的一次运行中，最少可同时打开128个图形窗口。可以为0到31号图形窗口指定一个索引号。对于32到127号图形窗口，可以用Window命令带上Free关键字来创建

创建一个索引号为10的图形窗口：

　　 IDL>Window, 10

　　若某个索引号图形窗口的窗口已经存在，再用Window命令创建相同索引号图形窗口，Window命令将首先删除旧窗口，然后建立一个带有此索引号的新窗口。

IDL>Window, /Free

用关键字Free建立的图形窗口，将会具有一个大于31的索引号。关键字Free是建立索引号大于31的常规图形窗口的唯一途径。

#### 确定当前图形窗口

当前图形窗口的索引号总是存储在!D.Window系统变量中。如果没有创建和打开图形窗口，系统变量!D.Window的值为－1。

IDL＞Window, /Free

IDL> thisWindowIndex = !D.Window

#### 使图形窗口成为当前窗口

使当前图形窗口为10号窗口：

IDL＞Wset, 10

随后所有的图形命令的结果都将显示到10号窗口内。

在某个窗口内绘制图形，该窗口必须是当前图形窗口。

#### 删除图形窗口

删除窗口10：

IDL＞Wdelete, 10

删除当前显示器上的所有图形窗口：

IDL＞WHILE　!D.Window NE –1 DO Wdelete, !D.Window

#### 图形窗口的位置和尺寸

用关键字XSize 和YSize创建一个宽200像素，高300像素的窗口：

IDL＞Window, 1, XSize=200, YSize=300

用关键字XPos 和YPos将窗口的左上角定位于显示器（75，150）处，键入:

IDL>Window, 2, XPos=75, YPos=150

#### 将图形窗口设置到显示器最前面

将图形窗口显示在前面，用Wshow命令+图形窗口索引号。

IDL＞Wshow, 1

将该窗口显示在前面，又将其变为当前窗口，可同时键入Wshow 和Wset命令：

IDL＞Wshow, 2

IDL>Wset, 2

如果输入不带参数的Wshow命令，在显示器上将当前窗口拖到前面。

IDL＞Wshow

注意，在PC机和Macintosh机器上，可以用ALT-TAB键或者OPTION-TAB键来循环选择已经在显示器上打开的窗口，让其可见并拥有窗口焦点。

#### 在图形窗口上设置标题

IDL＞Window, Title=’Example IDL Graphics Commands’

#### 清除图形窗口内容

IDL＞Erase

可以用以下命令实现用炭灰色清除当前图形显示：

IDL>TVLCT, 70, 70, 100

IDL>Erase, Color=100

　　清除非当前图形窗口（系统变量!D.Window指向的窗口）的内容，必须使该窗口成为当前图形窗口

# 第二章 简单的图形显示

## 创建线画图

用LoadData命令打开时序数据集。

IDL>curve=LoadData(1)

IDL>Help, curve

CURVE FLOAT =Array[101]

curve是一个具有101个元素的浮点矢量（或一维数组）。

绘出该矢量：

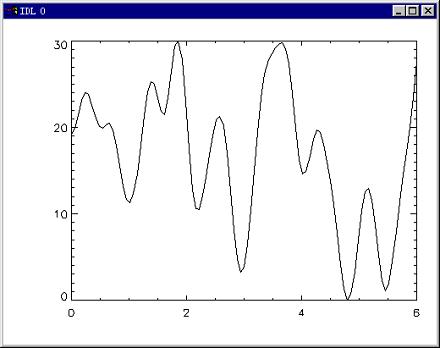
IDL>Plot, curve

x轴或水平轴被标识为从0到100，这与矢量中的元素个数相对应。而y轴或垂直轴则是用数据坐标来标识（取决于数据的坐标轴）。

但大多数情况下，线画图用于显示一组数据（独立数据）相对另外一组数据（非独立数据）的关系。例如，上面的曲线可能代表在某段时间内采集数据的信号。可能需要绘制某个时刻的信号值。在这种情况下，需要一条与该曲线矢量具有相同元素个数的矢量（这样可以获得一一对应的相关性），并将该矢量转换为实验中所用的时间单位。例如，可以创建一个时间矢量，并绘出它与上述曲线矢量的关系图：

IDL>time=FIndGen(101)\*(6.0/100)

IDL>Plot, time, curve

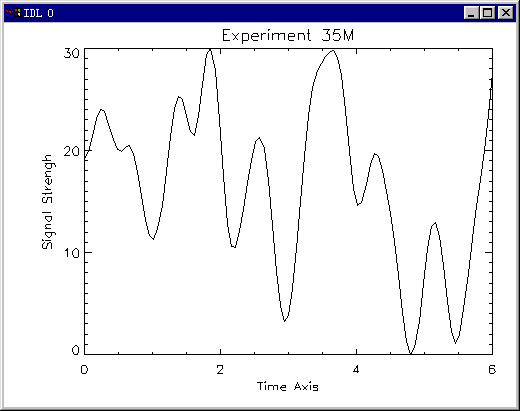
FIndGen命令创建一个元素值为0到100的共101个元素的矢量。乘法因子按比例缩小每个元素的大小，最后的结果是一个元素值为0到6之间的共101个元素的矢量。

**图1：独立数据（时间）与非独立数据（曲线）关系图。**

为此曲线图加上标题：

IDL>Plot,time,curve,XTitle='Time Axis',　$

YTitle='Signal Strengh',Title='Experiment 35M'

输出结果应与图2一样。

**图2：简单的带坐标轴标题和图形标题的线画图**

可以用CharSize关键字改变所有图形注记的大小。

将坐标轴标题的字符放大50%：

IDL>Plot, time, curve, XTitle='Time Axis', $

YTitle='Signal Strength', Title='Experiment 35M', $

CharSize=1.5

如果希望所有的图形显示的字符比正常情况下大，可以通过绘图系统变量上设置CharSize的大小，如下：

IDL>!P.CharSize=1.5

使Y轴的注记比X轴的大两倍：

IDL>Plot,　time,　curve,　XTitle='Time Axis',　XCharSize=1.0,　$

YTitle='Signal Strength',　YCharSize=2.0

如果设置XCharSize关键字为2，当!P.CharSize系统变量也被设置为2时，字符将比平常大四倍。

## 定制线画图

改变线条的线型和粗细

画一条线型为长虚线的线条：

IDL>Plot, time, curve, LineStyle=5

使用虚线画出曲线：

IDL>Plot, time, curve, LineStyle=2

|  |  |
| --- | --- |
| 索引号 | 线型 |
| 0 | 实线 |
| 1 | 点线 |
| 2 | 虚线 |
| 3 | 划点线 |
| 4 | 划点点线 |
| 5 | 长虚线 |

**表3：可以通过赋予LineStyle这个关键字不同索引号来改变线型**

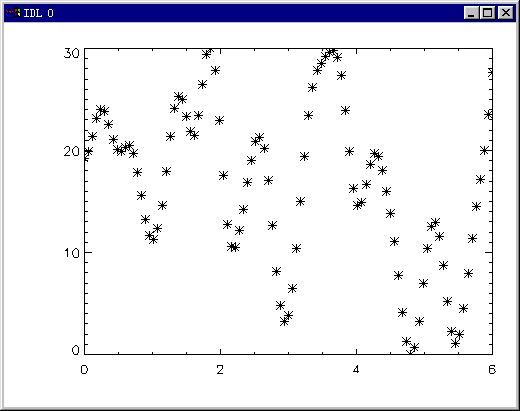
使用比正常值粗3倍的虚线来显示图形：

IDL>Plot, time, LineStyle=2, Thick=3

### 用符号代替线条显示数据

假如想用符号代替线条显示数据，就象LineStyle关键字一样，也存在类似的索引号供选择，以确定不同的线画图符号。表4给出了能通过PSym（绘图符号）关键字来选择的索引号。例如，可以通过设置PSym为2，用星号来绘图，如下：

IDL>plot, time, curve, Psym=2

 输出的图形应与图3中的图形相似。

**图3：用符号而不是线条来显示线画图。**

|  |  |
| --- | --- |
| 索引号 | 绘图符号 |
| 0 | 无符号，通过线条连接点 |
| 1 | 加号 |
| 2 | 星号 |
| 3 | 点 |
| 4 | 菱形 |
| 5 | 三角形 |
| 6 | 方形 |
| 7 | X |
| 8 | 用户自定义符号（用UserSym过程来定义） |
| 9 | 未用 |
| 10 | 直方图 |
| -PSym | 负值表示用线条连接相应的符号 |

**表4：这些符号索引号可以通过PSym关键字来引用以便在绘图中使用不同的符号。注意绘图符号为负值时表示用线条来连接相应的符号。**

### 用线条和符号来显示数据

赋予PSym关键字一个负值可以用线条将图形符号连接起来。

用实线与三角形符号绘出数据：

IDL>Plot, time, curve, PSym=-5

创建一个更大的符号，可用SymSize关键字。下面的语句画出的符号为正常的两倍。 IDL>Plot, time, curve, PSym=-5, SymSize=2.0

### 创建自己的图形符号

创建自己的图形符号：UserSym命令。在创建了一个特殊的图形符号之后，可通过设置PSym关键字为8来选择它。以下是一个创建五角星符号的例子。 x,y矢量定义五角星的顶点，它们的值为偏离原点（0，0）的位置。可以用UserSym命令通过设置关键字Fill创建一个填充的图形符号：

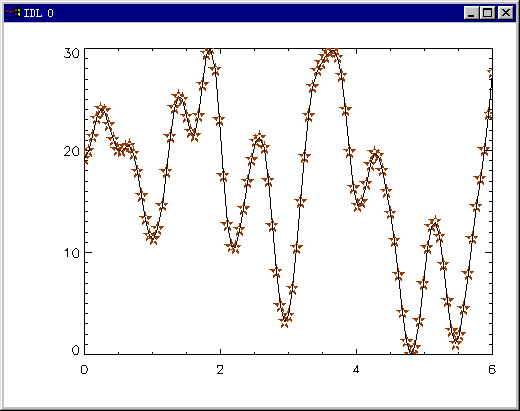
IDL>x=[0.0, 0.5, -0.8, 0.8, -0.5, 0.0]

IDL>Y=[1.0, -0.8, 0.3, 0.3, -0.8, 1.0]

IDL>TvLCT, 255, 255, 0, 150

IDL>UserSym, x, y, Color=150, /Fill

IDL>Plot, time, curve, PSym=-8, SymSize=2.0



**图4：用UserSym程序创建的符号来绘制的图。**

### 用不同的颜色绘制线画图

将颜色索引号1、2和3分别设置为碳灰，黄，绿色：

IDL>TvLCT, [70,255,0], [70,255,255], [70,0,0], 1

在碳灰背景下绘黄色图：

IDL>Plot, time, curve, Color=2, Background=1

如果只是想使线条成为不同的颜色,首先必须将NoData关键字打开来绘图,然后用OPlot命令覆盖该图。例如,在碳灰色背景上绘制黄色外框,数据用绿色显示:

IDL>Plot, time, curve, Color=2, Background=1, /NoData

IDL>OPlot, time, curve, Color=3

### 限定线画图的范围

仅绘出位于X轴上2至4之间的数据:

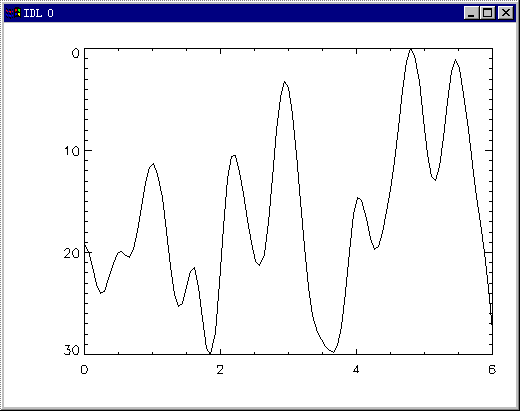
IDL>Plot, time, curve, XRange=[2, 4]

仅绘出Y值在10至20之间，X值在2至4之间的部分数据图形:

IDL>Plot, time, curve, YRange=[10, 20], XRange=[2, 4]

将Y轴的0点设置为图形的顶端，如下:

IDL>Plot, time, curve, YRange=[30, 0]

**图5:　将Y轴0点设置为图形顶端的图形**

### 改变线画图的风格

可以用[XYZ]Style这些关键字改变线画图的特性。

除去方框线，只留下X轴或Y轴:

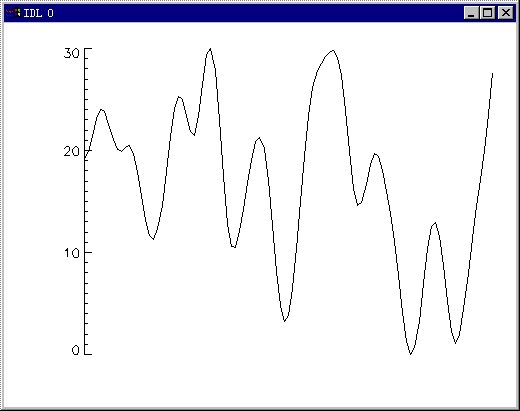
IDL>Plot, time, curve, XStyle=8, YStyle=8

|  |  |
| --- | --- |
| 值 | 对坐标轴的影响 |
| 1 | 精确的坐标轴范围 |
| 2 | 扩展坐标轴范围 |
| 4 | 不显示整个坐标轴 |
| 8 | 不显示外框（只画坐标轴） |
| 16 | 屏蔽Y轴起始值为0的设置（只有Y轴有此属性） |

**表5：[XYZ]Style关键字参数表，用于设置坐标轴的属性。注意：这些值可以累加从而设置坐标轴的多个而非单个属性**

仅用Y轴显示图形:

IDL>Plot, time, curve, XStyle=4, YStyle=8

**图6： 关闭X轴和方框只剩Y轴的线画图**

用Y轴和Y方向的网格线来显示同一幅图:

IDL>Plot, time, curve, XStyle=4, YTickLen=1, YGridStyle=1

让X轴显示精确的范围又隐藏方框线，可将两个参数值相加:

IDL>plot, time, curve, xstyle=8+1, xrange=[2, 5]

在线画图上创建网格线，可用TickLen关键字来完成：

IDL>Plot, time, curve, TickLen=1

创建向外的刻度标记:

IDL>Plot, time, curve, TickLen=-0.03

只在X轴上创建向外的刻度标记，键入：

IDL>Plot, time, curve, XTickLen=-0.03

在X轴上创建两个主要的刻度间隔，每个主要的刻度间隔内设置10个次要的刻度标记:

IDL>Plot, time, curve, XTicks=2, XMinor=10, XStyle=1

## 在线画图上绘出多种数据集

在同一套坐标轴内显示任意多套数据。OPlot命令就用于此目的。键入以下命令，输出结果应与图7相似:

IDL>Plot, curve

IDL>OPlot, curve/2.0, LineStyle=1

IDL>OPlot, curve/5.0, LineStyle=2

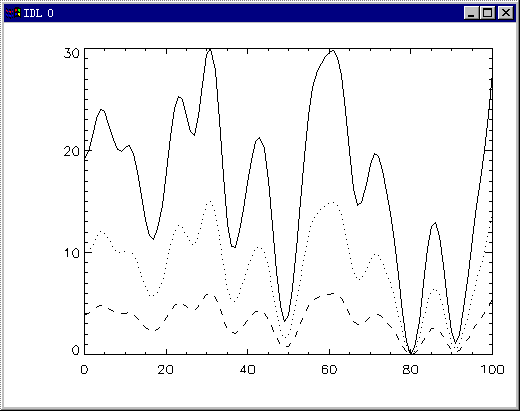
初始的Plot命令为以后的绘图建立数据比例(!X.S和!Y.S是比例参数)。或者说, !X.S和!Y.S系统变量告诉IDL如何在数据范围内取点以及如何将该点显示在设备坐标空间上。要确保初始图形有足够的轴长，以便包容以后绘制的所有图形，否则数据将被裁剪掉。可在第一个Plot命令中用XRange和YRange关键字来创建一个足够大的数据范围。为区别不同的数据集，可用不同的线型，不同的颜色，不同的图形符号等。Oplot命令接受很多被Plot命令接受的关键字。

IDL>TvLCT, [255, 255, 0], [0, 255, 255], [0, 0, 0], 1

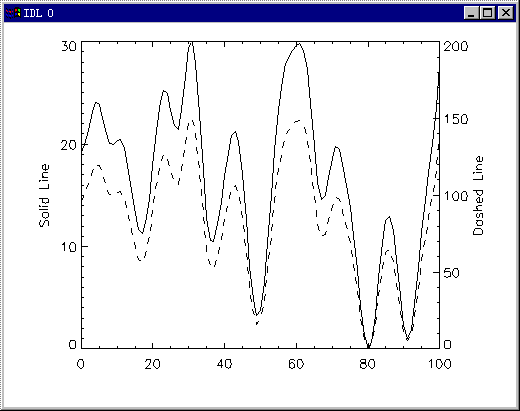
IDL>Plot, curve, /NoData

IDL>OPlot, curve, Color=1

IDL>OPlot, curve/2.0, Color=2

IDL>OPlot, curve/5.0, Color=3****

**图7：在同一个线画图上可以绘制无限多套数据集**

**图8：具有两个Y轴的线画图。第二轴是用Axis命令来定位的。一定要用Save关键字来将数据比例保存起来**

### 在多个轴的图上显示数据

在同一个线画图上显示两个或多个数据集，并用不同的y轴表示不同的数据集。使用Axis命令建立所需数量的坐标轴。使用Axis命令的关键是使用save关键字来存储正确的绘图比例参数（即存储在!X.S和!Y.S系统变量中的比例参数），以便后续图形的调用。

下面的例子在已绘出一幅图后，用带Save关键字的Axis命令建立第二个Y轴。OPlot命令中的曲线将调用通过Axis命令保存的比例因子，以确定其在图形中的位置

IDL>Plot, curve, YStyle=8, YTitle='Solid Line', $

Position=[0.15, 0.15, 0.85, 0.95]

IDL>Axis, YAxis=1, YRange=[0, Max(curve\*5+1)], /Save, $

YTitle='Dashed Line'

IDL>OPlot, curve\*5, LineStyle=2

Position关键字用来确定第一个图形在页面内的位置。

## 创建曲面图

在IDL中，任何二维的数据组都可以用Surface命令生成一个曲面图（经过自动消隐）。首先用LoadData命令打开 Elevation Data数据集：

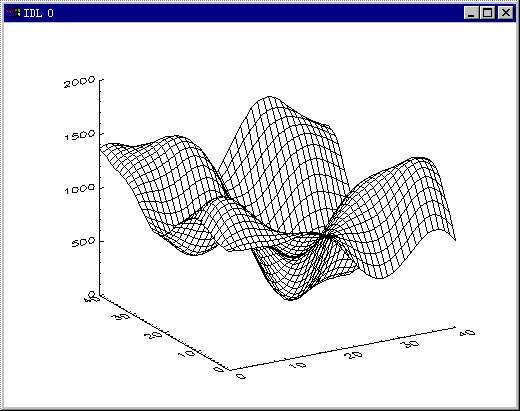
IDL>peak=LoadData(2)

通过键入Help命令，可以发现这是一个41\*41的浮点数组：

IDL>Help, peak

这个数组可以用一个命令使之视面图：

IDL>Surface, peak, CharSize=1.5

输出结果应与图9相似。

**图9：利用高程数据生成简单的曲面图。**

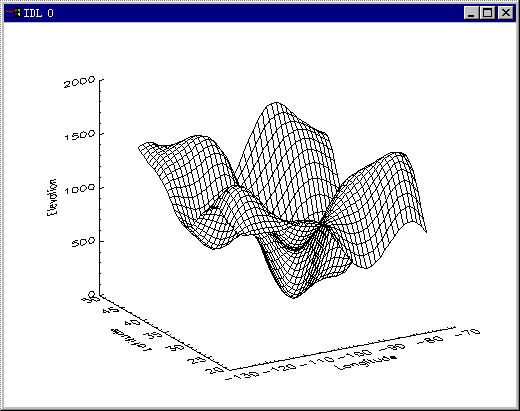
注意，如果仅用单个数组作为变量调用Surface命令，它将把该数组作为其元素个数（此例在X和Y方向都为41）的函数来绘图。可以规定X和Y轴的数值，以便显示的图形具有实际意义。例如，X和Y轴的数值可以是经纬度坐标。这里，使纬度范围为从24度到48度，经度范围为-122度到-72度：

IDL>lat=FIndGen(41)\*(24./40)+24

IDL>LON=FIndGen(41)\*50.0/41-122

IDL>Surface, peak, lon, lat, XTitle='Longitude', $

YTitle='Latitude', ZTitle='Elevation', CharSize=1.5

**图10：一个具有实际意义坐标值的曲面图。**

以上命令中的lon和lat参数是单调递增并且是规则的。

可以键入以下命令模拟随机分布的经度点：

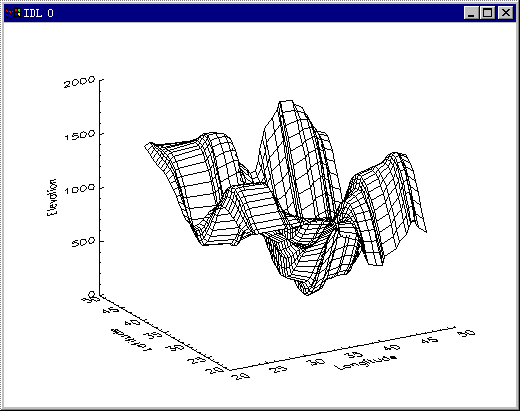
IDL>seed=-1L

IDL>newlon=RandomU(seed, 41)\*41

IDL>newlon=newlon[Sort(newlon)]\*(24./40)+24

IDL>Surface, peak, newlon, lat, XTitle='Longitude', $

YTitle='Latitude', ZTitle='Elevation', CharSize=1.5

**图11：同样的曲面图，但其X矢量具有不规则的空间分布**

## 定制曲面图

有70多个不同的关键字可以用来修饰曲面图。实际上，许多关键字在Plot命令中已经学过。例如在上面的代码中，就使用了相同的标题关键字对曲面图的轴进行标记。然而要注意，当用Title关键字时，所添加的标题被旋转了，从而保证标题总是位于曲面图的XY平面内。键入：

IDL>surface, peak, lon, lat, XTitle='Longitude', $

YTitle='Latitude', Title='Mt.Elbert', Charsize=1.5

使图形标题位于与显示面平行的平面内，就必须用Surface命令绘制曲面图，用XYOutS命令显示标题：

IDL>Surface, peak, lon, lat, Xtitle=’Longitude’, $

Ytitle=’Latitude’, Charsize=1.5

IDL> XYOutS, 0.5, 0.90, /Normal, Size=2.0, Align=0.5, $

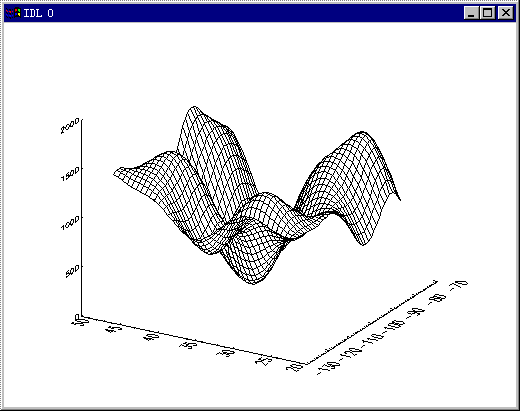
‘Mt.Elbert’

### 旋转曲面图

在观察曲面图时可能希望能旋转一个角度。曲面图可以用Ax关键字使其绕X轴或用Az关键字使其绕X轴旋转。当从轴上的正值向原点观察时，曲面图以逆时针方向，按某个角度值旋转。当Az和Ax关键字被忽略时其默认值是30度。

使曲面图绕Z轴旋转60度，绕X轴旋转35度：

IDL> Surface, peak, lon, lat, Az=60, Ax=35, Charsize=1.5

**图12：用Az和Ax关键字使曲面图旋转**

### 为曲面赋色

在碳灰色背景上创建一个黄色的曲面图，可键入：

IDL> TvLCT, [70, 255, 0], [70, 255, 255], [70, 0, 0], 1

IDL> Surface, peak, Color=2, Background=1

使曲面图的底面的颜色不同于顶面，比如说绿色，可以使用Bottom关键字来实现：

IDL>Surface, peak, Color=2, Background=1, Bottom=3

将轴以不同的颜色显示，比如绿色，而不是曲面，必须键入两个命令。第一个命令使用NoData关键字，只将轴绘出。第二个命令是在关闭轴线后绘出曲面本身。查看第31页的表5，了解[XYZ]Style关键字的参数值及其含意：

IDL>Surface, peak, Color=3, /NoData

IDL>Surface, peak, /NoErase, Color=2, Bottom=1,XStyle=4, YStyle=4, ZStyle=4

用不同的颜色画出曲面的格网线也是有可能的，而不同的颜色代表不同的数据。比如，可用第二个数据集覆盖第一个，第二个数据集含有对第一个数据集的格网进行着色后的信息。

为了说明如何工作，可打开一个名为Snow Pack的数据集，并用以下这些命令将此数据作为一个曲面显示。注意，Snow Pack数据集的大小与peak数据集一样，都是41\*41浮点数组：

IDL>snow=LoadData(3)

IDL>Help, snow

IDL>Surface, snow

通过用snow的变量值对peak变量的格网着色，将snow变量中的数据覆盖到peak变量中数据的上面。首先，用LoadCT命令装载色彩表内的一些颜色。实际的阴影处理是通过shades关键字完成的，如下：

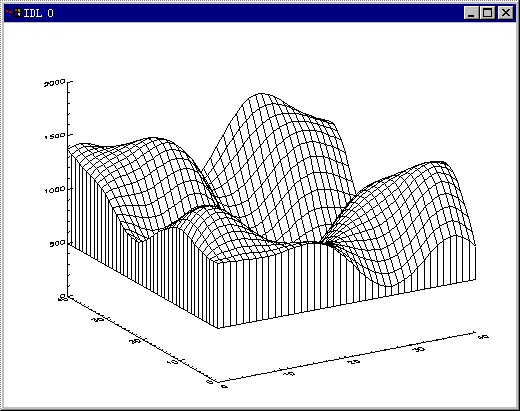
IDL>LoadCT, 5

IDL>Surface, peak, Shades=BytScl(snow, Top=!D.Table\_Size-1)

注意，必须用BytScl命令将snow数据集调整为所使用IDL时的色彩数。如果调整失败，只能看到一套坐标轴，而看不到曲面显示。这是因为，数据必须调整到曲面阴影处理时所需的0到255的范围。

### 修改曲面图外观

显示一个带边缘的曲面图，使用Skirt关键字来指定边缘该画到何处：

 IDL>Surface, peak, Skirt=0

IDL>Surface, peak, Skirt=500, Az=60

**图13：带边缘的曲面图**

仅绘出水平线，获得一种层叠线形图：

IDL>Surface, peak, /Horizontal

可以通过关键字只显示曲面的底面或顶面：

IDL>Surface, peak, /Upper\_Only

IDL>Surface, peak, /Lower\_Only

只显示曲面本身，而不需要轴线：

IDL>Surface, peak, XStyle=4, YStyle=4, ZStyle=4

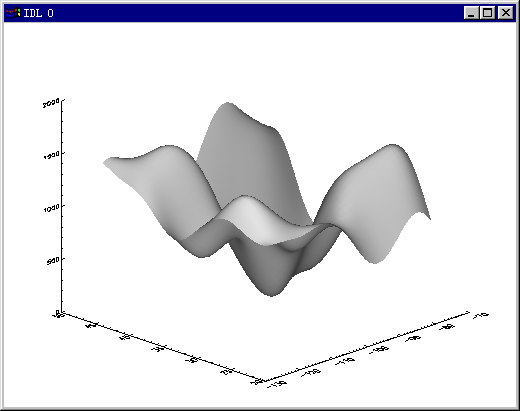
## 创建阴影曲面图

使用Gouraud光源阴影算法创建阴影曲面图：

IDL>Shade\_Surf, peak

Shade\_Surf命令接受大多数被Surface命令接受的关键字。

旋转阴影曲面：

IDL>Shade\_Surf, peak, lon, lat, Az=45, Ax=30

**图14：　用Gouraud光源阴影算法生成的阴影曲面图**

### 改变阴影处理参数

用Set\_Shading命令可以改变Shade\_Surf命令所使用的阴影处理参数。

将光源的光线的方向从平行Z轴的默认值[0，0，1]改变为平行X轴的方向[1，0，0]：

IDL>Set\_Shading, Light=[1, 0, 0]

IDL>Shade\_Surf, peak

也可以从色彩表中挑选哪种颜色索引号用作阴影处理。

把红色色普表（色普表3）装载到颜色索引号100到199之中，并将之用于阴影处理：

IDL>LoadCT, 3, NColors=100, Bottom=100

IDL>Set\_Shading, Values=[100, 199]

IDL>Shade\_Surf, peak

注意将光源位置和颜色参数恢复原值，否则可能会造成混乱。

IDL>LoadCT,5

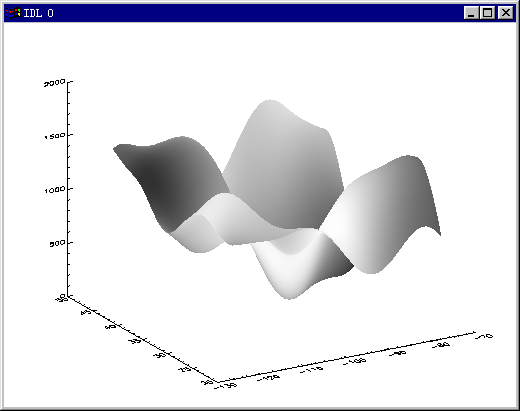
IDL>Set\_Shading, Light=[0,0,1], Value=[0,!D,Table\_Size-1]

### 用其它数据集为阴影处理提供参数

其它数据集也可以为阴影处理时的各数据点提供颜色值。正如前述，可以用Shades关键字为曲面上各点指定颜色索引号。每个像素点的阴影处理都是根据该点周围数据值通过插值求出。例如，下面是一个用snow变量生成的阴影曲面图：

IDL>Shade\_Surf, snow

现在用这个数据集来对最初的高程数据集进行阴影处理:

IDL>Shade\_Surf,peak,lon,lat, Shades=BytScl(snow, Top=!D.Table\_Size)

**图15：用snow数据集对peak数据进行阴影处理**

如果要求根据数据点的高程值来对曲面进行阴影处理，可对数据集本身进行字节比例缩放:

IDL>Shade\_Surf, peak, Shades=BytScl(peak, Top=!D.Table\_Size)

将另一数据集覆盖在曲面图上是一种给数据升维的方法。例如，可将一组数据集覆盖在一个三维曲面图上，就可以直观的获得四维的信息。如果同时让两组数据集合随时间活动起来,就可以直观的获得五维信息。

将原始曲面覆盖在经过阴影处理的曲面图上，通过结合使用Shade\_Surf命令和Surace命令:

IDL>Shade\_Surf, peak

IDL>Surface, peak, /NoErase

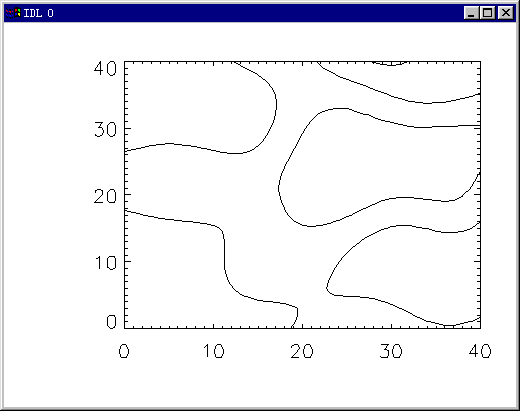
## 创建等值线图

在IDL中，任意二维数组都可以用一个Contour命令显示为等值线图。如果已经在这次IDL运行中定义了peak变量，可直接使用该变量。如果没有定义,可以使用LoadData命令来载入Elevation Data中的数据集。

IDL>peak=LoadData(2)

IDL>Help, peak

将这个数据集显示为等值线图（图16）:

 IDL>Contour, peak, CharSize=1.5

**图16：一个基本的等值线图，注意X、Y轴的标记代表该数组中的元素个数**

注意，如果仅用单个二维数组作为参数调用Contour命令，它将把该数组作为其元素个数（此例在X和Y方向都为41）的函数来绘图。如前述所用Surface命令一样，可以指定X 轴和Y轴的数值，以便使其具有实际意义。可以像前述一样使用经度和纬度矢量

IDL>lat=FIndGen(41)\*(24./40)+24

IDL>lon=FindGen(41)\*50.0/40-122

IDL>Contour, peak, lon, lat, XTitle='Longitude', $

YTitle='Latitude'

注意轴自动缩放了。首先，等值线没有延伸到等值线图的边缘，其次，可以发现轴上的标记与lon矢量和lat矢量的最小值和最大值不同。

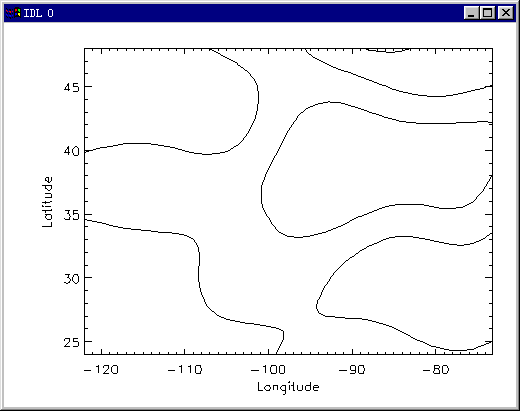
IDL>Print, Min(lon), Max(lon)

IDL>Print, Min(lat), Max(lat)

为了防止轴的自动缩放，可以设置XStyle和YStyle关键字：

IDL>Contour, peak, lon, lat, XTitle='Longitude', $

YTitle='Latitude', XStyle=1, YStyle=1



**图17：具有实际数量意义的等值线图**

在早期的IDL版本中，Contour 命令使用所说的单元画法来计算并绘画数据的等值线。在这种方法中，等值线图是从图底画到图顶。这种方法是有效的，但是它不允许选项，比如标注等高线。而单元跟踪法被用来完整地画出围绕等值线图的每一条等值线。这需要较长的时间，但可以允许对等值线作更多的控制。例如，等值线可以断开用于等值线的标注。这种单元跟踪法可以用Follow关键字来调用：

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, /Follow

从IDL5版本开始，Contour命令一般都使用单元跟踪法来绘制等值线图。所以，Follow关键字已经过时了。但该关键字仍然被使用，是因为它对自动标注其它每条等值线的有益作用。选择等值线数目缺省情况下，IDL选择6条匀称的等值线间隔（即有5条等值线）绘制等值线图。但是，可以用几种不同的方法改变缺省值。例如，可以用Nlevels关键字告诉IDL需要绘制多少条等值线。IDL将计算出等间隔的等值线间隔数。例如，要绘制具有12条等间隔的等值线图，可键入：

IDL＞Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Follow, $

Nlevels=12



**图18：这是将等值线级别设置为12的等值线图。注意，每隔一条等值线都会标注一条，这是使用Fellow关键字的一种副作用。**

不幸的是，IDL只计算出少于12条的间隔线。显然，NLevels关键字的值在IDL中只能作为等值线选择算法中的一个“建议”。

因此，大多数IDL程序员都是自己计算等值线数目。例如，能精确规定哪条等值线应该画，并用Levels关键字传给Contour命令，而不是用NLevels关键字，如下所示：

IDL>vals=[200, 300, 600, 750, 800, 900, 1200, 1500]

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals

要选择12个间距相等的等值线间隔，可编写如下代码：

IDL>nlevels=12

IDL>step=(Max(peak)-Min(peak))/nlevels

IDL>vals=Indgen(nlevels)\*step+Min(peak)

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals

可以C\_Labels关键字精确的指定哪一根等值线应该标注，如果某元素的值是1（或更精确，只要是正数），相应的等值线就给予标注；如果某元素的值是0，相应的等值线就不予标注。如果某条等值线没有元素值与之对应时，那么这条等值线就不标注。

例如，要标注第一，第三，第六和第七条等值线：

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals, C\_Labels=[1, 0, 1, 0, 0, 1, 1, 0]

要标注所有的等值线，可以使用Replicate命令来将1复制所需要的次数：

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, /Follow, $

Levels=vals, C\_Labels=Replicate(1, nlevels)

## 修改等值线图

等值线图可用与Plot命令和Surface命令中相同的关键字进行修改。但是仍然还有许多仅适用于Contour命令的关键字。它们中的大部分经常用于修改等值线本身。

用坐标轴标题注释等值线图：

IDL> contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Follow,

Xtitle=’Longitude’, Ytitle=’Latitude’, $

Charsize=1.5, Title=’Study Area 13F89’, Nlevels=10

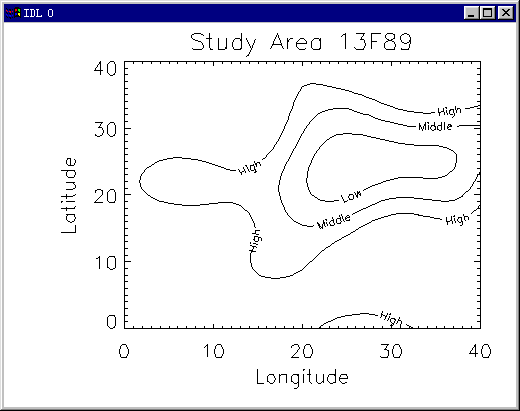
也可以用C\_Annotation关键字在等值线上标注释。可以用字符串标记每一条等值线：

IDL> contour, peak, Xstyle=1, Ystyle=1, /Follow, $

Xtitle=’Longitude’,Ytitle=’Latitude’,Charsize=1.5, $

Title=’Study Area 13F89’,

C\_annotation=[‘Low’,’Middle’,’High’],Levels=[200, 500, 800]



**图19： 等值线可以用自己提供的文本标识**

### 改变等值线图的外观

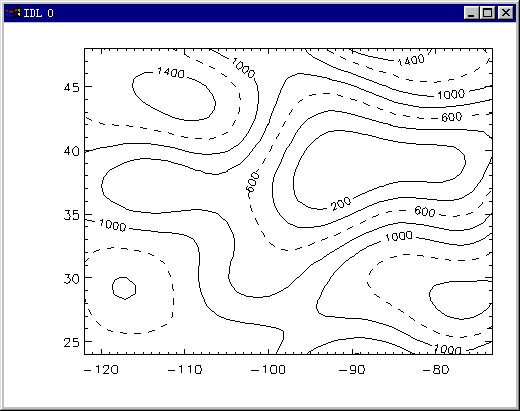
　　修改等值线图的外观有许多方法。这里有一些例子。能改变的特性之一是等值线的线型（见表3列出的可选用的线型值）。例如，为了使等值线成为虚线的线型，键入：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

/Follow, C\_LineStyle=2

假如需要隔二条等值线有一条虚线，可以用C\_LineStyle 关键字指定一个线型索引矢量，假如等值线数比索引号多，那么这些索引号将被循环使用或被重复使用。键入 ：

IDL> contour, peak, lon, lat, XStyle=1, Ystyle=1, /Follow, $

Nlevels=9, C\_LineStyle=[0, 0, 2].

**图20：可以修改等值线图的许多方面。这是隔二条等值线有一条虚线的等值线图。**

使等值线具有双倍的宽度：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

Nlevels=12, 　C\_Thick=2,　 /Follow

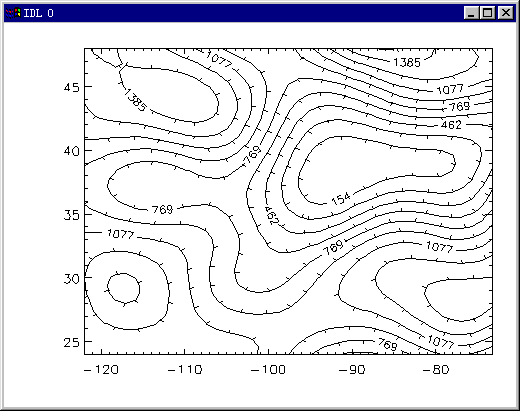
通过指定一个线宽矢量，可以间隔修改等值线的宽度：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

Nlevels=12, 　C\_Thick=[1, 2],　 /Follow

通过修改等值线图,可以很容易地看到等值线图的下坡方向：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

/ Follow, Nlevels=12, /Downhill

**图21：用Downhill来显示等值线图的下坡方向。**

### 给等值线图赋色

　　以碳灰颜色为背景的黄色等值线图：

IDL> TvLCT, [70, 255], [70, 255], [70, 0], 1

IDL> Contour, peak, lon, lat, XStyle=1, YStyle=1, $

NLevels=10, Color=2, Background=1, /Follow

将上图的等值线变为绿色 ：

IDL> TvLCT, 0, 255, 0, 3

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Follow, $

Nlevels=10, Color=2, Background=1, C\_Colors=3

关键字C\_Color也可以被表达为色彩表索引号的矢量，并以循环的方式绘制等值线。也可以使用Tek\_Color命令为等值线创建或者装入颜色，如下：

IDL> Tek\_Color

IDL> TvLCT, [70, 255], [70, 255], [70, 0], 1

IDL>Contour, peak, lon, lat, XStyle=1, YStyle=1, $

Nlevels=10, Color=2, Background=1, $

C\_Colors=IndGen(10)+2, /Follow

可以用C\_Colors关键字使每间隔二条等值线有一条蓝色的等值线，其余的等值线为绿色：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

NLevels=12, Color=2, Background=1, $

C\_Colors=[3, 3, 4], /Follow

## 创建填充的等值线图

　　创建一张填充的等值线图，使用关键字Fill。首先，装入12种颜色于色彩表中作为填充颜色。色彩索引号由关键字C\_Colors给出：

IDL> LoadCT, 0

IDL> LoadCT, 4, Ncolors=12, Bottom=1

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Fill, $

NLevels=12, /Follow, C\_Colors=Indgen(12)+1

用这种方法填充颜色还是存在许多问题，在等值线图有一个以背景颜色填充的”洞”。假如将背景色与图形颜色交换一下，就可以看得更清楚一些

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Fill, $

NLevels=12, /Follow, C\_Colors=Indgen(12)+1, $

Background=!P.Color, Color=!P.Background

“洞”产生的原因是由于IDL用第一种颜色填充了第一和第二条等值线间的空间。用第一种填充颜色去填充第零条(或背景)和第一条等值线之间的空间似乎更合理。但是，要使IDL这样做，不得不给定自己的等值线数目，并用关键字Levels传送给Contour命令。通常可用下述代码实现：

IDL> step = (Max(peak) – Min(peak)) / 12.0

IDL> clevels = IndGen(12)\*step + Min (peak)

现在，就得到了正确的等值线填充颜色。

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Fill, $

Levels=clevels, /Follow, C\_Colors=Indgen(12)+1, $

Background=!P.Color, Color=!P.Background

将填充的等值线图放在地图投影上，使用Cell\_Fill关键字也是个好主意。

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

Levels=clevels, C\_Colors=Indgen(12)+1, /Cell\_Fill

单元填充算法有时会破坏等值线图的坐标轴。可以通过不带数据的等值线图的重新绘制来修复：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

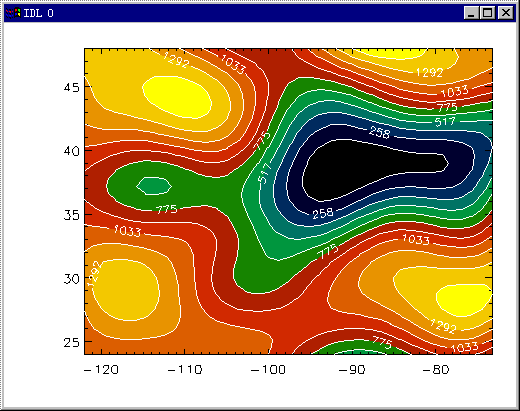
Levels=clevels, /NoData, /NoErase, /Follow

在已填充好颜色的等值线图上看到等值线。在IDL中用Overplot关键字实现：

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, $

Levels=clevels, /Fill, C\_Colors=IndGen(12)+1

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, /Follow, $

Levels=clevels, /overplot 

**图22：在已填充的等值线图上覆盖等值线**

　　注意，在等值线图上，Overplot关键字仅仅绘出等值线，而不绘出等值线图的坐标轴。 NoErase关键字则是绘出完整的等值线图，而不删除在屏幕上已显示的内容。

## 在显示窗口定位图形输出

　　IDL有几种在显示窗口中定位线画图、曲面图、等值线图和其它图形的方法（比如地图投影）。图形位置是指在显示窗口上被图的坐标轴框起来的部分中的位置。图形位置不包括坐标轴标识，坐标轴标题或者其它注释（见下面的图23）。图形区域是显示窗口的一部分，包括图形位置，也包括环绕图形位置的空间，用来注明坐标轴标识，坐标轴标题和图标题等。图形边界定义为在显示窗口内不包括图形位置的区域。

　　图形位置可以用!P.Position系统变量设置，或者用Position关键字对Plot,　Surface, Contour或其它IDL图形命令进行设置。整个图形区域可用!P.Region系统变量设置，或者通过!X, !Y和!Z系统变量的Region字段来设置单个坐标轴的区域。图形边界可以用[XYZ]Margin关键字来对Plot,　Surface,　Contour或IDL的其它图形命令进行设置，或者通过!X,　!Y和!Z系统变量的Margin字段来设置。

10

5

0 0 20 40 80

10

5

0 0 20 40 80

10

5

0 0 20 40 80

图形边缘

图形区域

图形位置

　　在缺省值情况下，IDL是在将图形输出到显示窗口的时候设置图形边界的。但是，正如所看到的，这并不是最好的选择。有时，使用图形定位来定位图形显示会更好，尤其是，当在一个显示窗口中显示多个命令的输出结果时。

**图23： 图形位置是被坐标轴包围起来的区域。图形区域与图形位置类似，但它还包括图形标题和其它注释的区域。图形边缘正好与图形位置相反。图形边缘由字符的单位确定，而图形位置和图形区域是由归一化的坐标单位确定。**

### 设置图形边缘

图形边缘可以用图形命令中的[XYZ]Margin关键字设置，或者通过!X,!Y和 !Z系统变量的Margin字段来设置。关于图形边缘的特殊地方在于根据字符尺寸来确定的单位。X方向的边缘是用两元素矢量来设置的，这两个元素分别规定左右的偏移量。Y方向的边缘用同样的方法确定底部和顶部的偏移量。默认边缘值是X轴方向为10和3，Y轴方向为4和2。

查看当前字符尺寸的设备坐标值或像素坐标值：

IDL> Print, !D.X\_Ch\_Size, !D.Y\_Ch\_Size

　　例如，在苹果机（Macintosh）中，默认的字符尺寸在X方向上为6个像素，在Y方向上为9个像素。因此，一张等值线图的边缘就被确定为图形的左边为60个像素（6\*10），右边为18个像素（6\*3）。如果CharacterSize关键字在 Contour命令中设置为2，那么将会出现图形的左边边缘为120个像素，而图形的右边边缘为36个像素。

例如，为了将图形四周边缘都改变为3个缺省的字符宽度：

IDL> Plot, time, curve, Xmargin=[3, 3], Ymargin=[3, 3]

如果同时改变字符尺寸，图形将出现非常大的差异。因为图形边界是由字符的尺寸确定的：

IDL> !X. Margin = [3, 3]

IDL> !Y. Margin = [3, 3]

IDL> Contour, peak, CharacterSize=2.5

IDL. Contour, peak, CharacterSize=1.5

确保将图形边界已恢复为默认值：

IDL> !X.Margin = [10, 3]

IDL> !Y.Margin = [4, 2]

### 设置图形位置

　　设置图形位置需要设置一个四个元素的矢量，该矢量依次给定图形在显示窗口中的左下角和右上角坐标[X0，Y0，X1，Y1]。这些坐标值通常为归一化的值，其范围在0至1之间（如：0常常代表显示窗口的左边或者底部，1常常代表显示窗口的右边或者顶部。）

　　 将图形输出结果在显示窗口的上半部分显示，可以设置!P.Position系统变量并显示图形：

IDL> !P.Position = [0.1, 0.5, 0.9, 0.9]

IDL> Plot, time, curve

将!P.Position系统变量复位，以便后面的图形输出能正常地显示在窗口中：

IDL> !p.position = 0

　　假如仅想给一张图形显示定位，可以用图形命令的Position关键字规定一个图形位置。如果要在整个显示窗口的左半部分显示等值线图：

IDL> Contour, peak, Position=[0.1, 0.1, 0.5, 0.9]

注意，Position关键字可以用来在相同的显示窗口输入多幅图形。只要确保在输入第二幅图形和所有的后续图形时，使用NoErase关键字。这可防止在显示图形时删除前面已显示的图形。对于所有的图形输出命令来说，这是一项默认特性，但是TV和TVScl命令是例外。

在一张等值线图上加入一条线图：

IDL> Plot, time, curve, Position=[0.1, 0.55, 0.95, 0.95]

IDL> Contour, peak, Position=[0.1, 0.1, 0.95, 0.45], /NoErase

### 设置图形区域

图形区域与图形位置一样，都是由归一化坐标值来确定的。同样可以通过设置！P.Region系统变量来指定。

在显示窗口上方三分之二的部分区域中显示一幅图形：

IDL> !P.Region = [0.1, 0.33, 0.9, 0.9]

IDL> Plot, time, curve

将!P.Region系统变量复位，以便后续图形能正常地在窗口内显示：

IDL> !P,Region = 0

### 创建多个图形

　　通过使用图形位置和图形区域系统变量以及上面所讨论的关键字可以在一个显示窗口中定位多个图形（只要绘制第二个和后续的图形时使用了NoErase关键字）。但是使用!P.Multi系统变量在显示窗口内创建多个图形更加容易。!P.Multi由以下五个元素的矢量定义。

!P.Multi[0] !P.Multi的第一个元素包括剩下的要在显示窗口或者PostScript页上绘制的图形数目。这有点不直观，以下就可以看到它是如何使用的。通常设置为0，意思是，没有剩下要在显示窗口输出的图形。接下来的图形命令将删除显示的图形，并且开始绘制新的多个图形中的第一个。

!P.Multi[1] 此元素规定了该页上图形的列数

!P.Multi[2] 此元素规定了该页上图形的行数

!P.Multi[3] 此元素规定了在Z方向上叠加的图形数目（仅适用已经建立了三维坐标系的情况下）

!P.Multi[4] 此元素规定了是先按行显示图形（!P.Multi[4]=0），还是先按列显示图形（!P.Multi[4]=1）。

将!P.Multi参数设置为按两行两列在显示窗口内显示四幅图形，并且，先按列显示图形：

IDL> !P.Multi = [0, 2, 2, 0, 1]

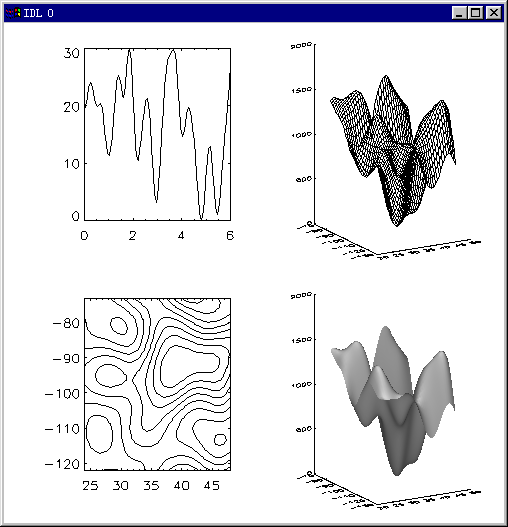
显示图形时，每个图形占据窗口的四分之一位置：

IDL> window, Xsize=500, Ysize=500

IDL> Plot, time, curve, LineStyle=0

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, Nlevels=10

IDL> Surface, peak, lon, lat

IDL> shade\_Surf, peak, lon, lat

**图24： 在单个显示窗口内可以绘制多幅图形**

#### 给多幅图形的图留下标题空间

为刚刚创建好的四个图形的总图加上一个标题，应为标题留出空间：

IDL> !P.Multi = [0, 2, 2, 0, 1]

IDL> !Y.Omargin = [2, 4]

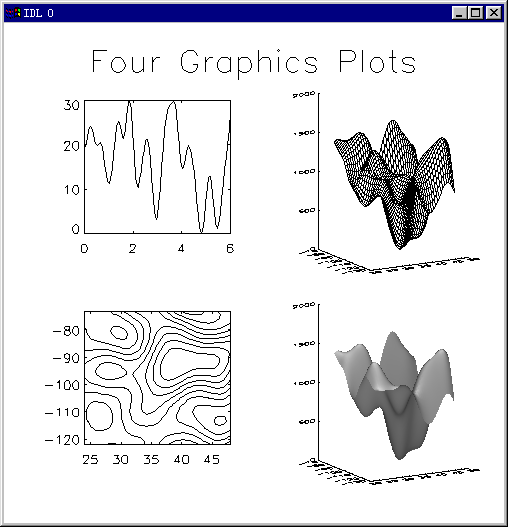
IDL> Plot, time, curve, LineStyle=0

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, Nlevels=10

IDL> Surface, peak, lon, lat

IDL> Shade\_Surf, peak, lon, lat

IDL> XYOuts, 0.5, 0.9, /Normal, ‘Four Graphics Plots’, $

Alignment=0.5, Charsize=2.5

**图25： 使用关键字!Y.OMargin在多幅图形的上方留出4个字符高度的空间来放标题**

#### 使用!P.Multi变量创建不对称的排列

　　曲面图与阴影图一上一下地排列显示在显示窗口的左边，而显示窗口的右边是一张用同样数据生成的等值线图：

IDL> !P.Multi = [0, 2, 2, 0, 1]

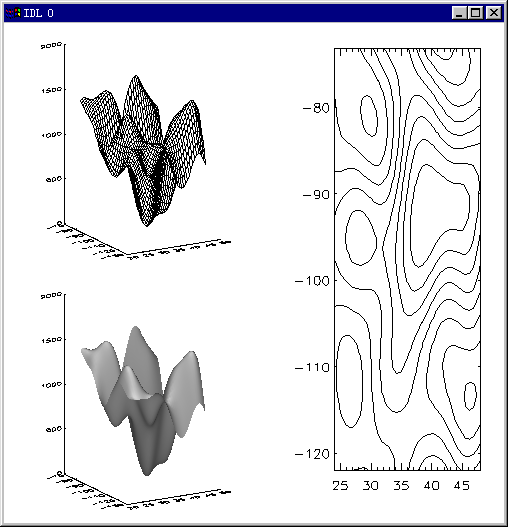
IDL> !Y.OMargin=[0,0]

IDL> Surface, peak, lon, lat

IDL> Shade\_Surf, peak, lon, lat

IDL> !P.Multi = [1, 2, 1, 0, 0]

IDL> Contour, peak, lon, lat, Xstyle=1, Ystyle=1, Nlevels=10

　　第一个!P.Multi命令设置了一个二列二行的排列形式，第一、第二张图已制好。第二个 !P.Multi命令设置了一个二列一行的排列形式。但要注意 !P.Multi[0]被设置为1。结果是等值线图进入了显示窗口的第二个位置而不是第一个

**图26： 可以使用!P.Multi在显示窗口定位图形的不对称排列**

注意：与PLOT和CONTOUR命令不一样，TV命令与!P.Multi一起使用无效。

如果设置了MULTI关键字，!P.Multi系统变量就有效：

IDL> image = LoadData(7)

IDL> !P.Multi=[0, 2, 2]

IDL> FOR j=0, 3 DO TVImage, image, /Multi

　　确保已经将!P.Multi复位，以便在一页上显示单个图形。像许多系统变量一样， !P.Multi可以通过设置!P.Multi=0重新设置为它的默认值。

IDL> !P.Multi = 0

## 给图形显示添加文本

　　图形注释和其它文本可以通过许多方式添加到图形显示上。最通常的方法是通过图形显示命令的关键字。被添加的文本可以三种字体“风格”中的任意一种形式出现

|  |  |
| --- | --- |
| **!P.Font** | **字体选择** |
| -1 | 矢量字体（也叫软字体或Hershey字体） |
| 0 | 硬字体 |
| 1 | TureType轮廓字体 |

**表6： 字体“风格”可以通过设置!P.Font系统变量或者Font关键字为适当值来加以选择。矢量字体是直接图形命令的缺省字体，它们有不依赖于平台的优点**

设置!P.Font系统参数为-1或者在图形输出命令上设置Font关键字为-1，选择矢量字体：

IDL> Plot, time, curve, Font=-1, Xtitle=’Time’, $

Ytitle=’Signal’, Title=’Experiment 35F3a’

用默认的Helvetica TrueType字体的外形来绘制图形，须设置Font关键字为1：

IDL> Plot, time, curve, Font=1, Xtitle=’Time’, $

Ytitle=’Signal’, Title=’Experiment 35F3a’

TrueType字体可以用Device命令通过Set\_Font和TT\_Font关键字来选择：

IDL> Device, Set-Font=’Courier’, /TT\_Font

IDL> Plot, time, curve, Font=1, Xtitle=’Time’, $

Ytitle=’Signal’, Title=’Experiment 35F3a’

帮助

IDL> ? fonts

　　硬字体通过设置!P.Font系统变量或 Font关键字为0来加以选择。通常情况下，硬字体并不用于图形显示中，而是在当内容被输出到硬拷贝输出设备时使用，例如 PostScript打印机。直到最近的IDL版本，硬字体都不能很好地在三维空间内旋转。因此，在使用类似于Surface等三维命令时，一般都不使用硬字体。

IDL> Plot, time, curve, Font=0, Xtitle=’Time’, $

Xtitle=’Signal’, Title=’Experiment 35F3a’

### 找出可用字体的名称

可以用以下Device命令找出可用的硬字体名：

IDL> Device, Font=’\*’, Get\_FontNames=fontnames

IDL> For j=0, N\_Elements(fontnames)-1 DO Print, fontnames[j]

　　使用TT＿Font关键字，TureType字体名称可用类似的方法找出。TT＿Font关键字用来选择系统上可用的TureType字体。

IDL> Device, FONT=’\*’, Get\_FontNames=fontnames, /TT-Font

IDL> For j=0, N\_Elements(fontnames)-1 DO Print, fontnames[j]

|  |  |  |  |
| --- | --- | --- | --- |
| 数值 | 描述 | 数值 | 描述 |
| !3 | Simplex Roman | !12 | Simplex Script |
| !4 | Simplex Greek | !13 | Complex Script |
| !5 | Duplex Roman | !14 | Gothic Italian |
| !6 | Complex Roman | !15 | Gothic German |
| !7 | Complex Greek | !16 | Cyrillic |
| !8 | Complex Italian | !17 | Triplex Roman |
| !9 | Math Font | !18 | Triplex Italian |
| !10 | Special Characters | !20 | Miscellaneous |
| !11 | Gothic English | !X | 回到刚进入时的字体 |

**表7：Hershey字体和其相应的在IDL中用于选择各字体的索引号**

### 用XYOutS命令添加文本

　　在IDL 中一个非常重要的命令是XYOutS命令（“在XY 给定的位置，输出一个字符串”）。这个命令用来在窗口的特定位置放入一个文本字符串。（XYOutS 的第一个位置参数是X的位置，第二个位置参数是Y的位置）。例如，给线画图加上一个较大的标题，键入如下命令：

IDL> Plot, time, curve, Position=[0.15, 0.15, 0.95, 0.85]

IDL> XYOutS, 0.5, 32, ‘Results: Expe riment 35F3a’, Size=2.0

注意，是用数据坐标来给定X和Y的位置，同时Y坐标在图形边界之外。在默认情况下，XYOutS过程使用数据坐标系统。选用适当的关键字，设备坐标系统和归一化坐标系统也可使用。（数据坐标系统自然地由其自身描述。设备坐标有时称为像素坐标，设备坐标系统经常和图像一起使用。归一化的坐标系统在每个方向从0到1。当需要用独立于设备输出图形时，经常使用归一化坐标。）

可以像如下使用归一化坐标把标题加在线画图上

IDL> Plot, time, curve, Position=[0.15, 0.15, 0.95, 0.85]

IDL> XYOutS, 0.2, 0.92, ‘Results: Experiment 35F3a’, $

Size=2.0, /Normal

### 用矢量字体使用XYOut

XYOutS命令可用于矢量字体，TureType字体或硬字体，只需按上述的那样，简单的设置Font关键字值。在直接图形命令中，矢量字体系统使用最频繁。

矢量字体或Hershey字体的主要优点是它们的平台独立性，并且可在三维空间中缩放和旋转。可以用Triplex Roman字体输出上图的标题：

IDL> Plot, time, curve, Position=[0.15, 0.15, 0.95, 0.85]

IDL> XYOutS, 0.2, 0.92, ‘!17Results: Experiment 35F3a!X’, $

Size=2.0, /Normal

Triplex Roman字体由!17转义序列来选定。标题串末端的!X将使字体转变为Simplex Roman字体，而Simplex Roman字体是在变为Triplex Roman字体前所使用的字体。这个转变步骤是非常重要的。否则，默认设置将变为Triplex Roman，并且所有接下来的串标记都将使用Triplex Roman字体。

使用 Greek字符集作为X轴的标题，并且按下面输出下图的标题：

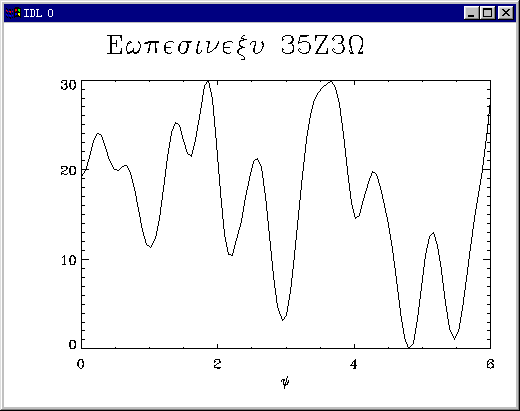
IDL> Plot, time, curve, Xtitle=’!17w’, $

Position=[0.15, 0.15, 0.95, 0.85]

IDL> XYOutS, 0.2, 0.92, ‘Experiment 35F3X’, size=2.0, /Normal

　　可以在图27中看到结果。即使没有规定该图的标题是什么字符集，标题也是用Greek字符集输出的。恢复为默认项Simplex Roman的唯一办法是使用 Simplex Roman字体输出另一个字符串：

IDL> XYOutS, 0.5, 0.5, ‘!3Junk’, /Normal, CharSize=-1

注意，在上面的代码中CharSize关键字的使用。当这个关键字值为-1时，字符串被隐藏，不在窗口显示。

**图27当选择一种Hershey字体要注意，否则可能为用户提供一个看上去象希腊字母的标题**

### 排列文本

　　可以用XYOutS命令的Alignment关键字通过相对位置来定位文本。当Alignment的值为0时，字符串居左排列（这是默认值）；当Alignment的值为1时，字符串居右排列；当Alignment的值为0.5时，将根据X和Y值所定义的位置居中排列。例如：

IDL> Window, Xsize=300, Ysize=250

IDL> XYOutS, 150, 55, ‘Research’, Alignment=0.0, $

/Device, CharSIZE=2.0

IDL> XYOutS, 150, 110, ‘Research’, Alignment=.5, $

/Device, CharSIZE=2.0

IDL> XYOutS, 150, 170, ‘Research’, Alignment=1.0, $

/Device, CharSize=2.0

IDL> Plots, [0.5,0.5], [1.0,0.0], /Normal

### 删除文本

用XYOutS书写的文本有时可以通过用背景颜色书写同样的文本来删除。Color关键字与!P.Background系统变量一起使用可以达到这个目的。需要指出的是，这仅仅在文本只是写在背景上没有任何东西的情况下奏效。通常还有别的更有效的方法来删除注释。

用背景颜色删除注释：

IDL> window, Xsize=300, Ysize=250

IDL> XYOutS, 150, 110, ‘Research’, Alignment=0.50, $

/Device, CharSize=2.0

IDL> XYOutS, 150, 110, ‘Research’ Alignment=0.50, $

/Device, CharSize=2.0, Color=!P.Background

### 改变文本的方向

　　用XYOutS命令输出的文本可以通过Orientation关键字相对于水平方向上的角度来定向。Orientation关键字可确定文本基线从水平基线开始旋转的度数：

IDL> Window,Xsize=300, Ysize=250

IDL> XYOutS, 150, 110, ‘Research’, Alignment=0.5, $

/Device, CharSize=2.0, Orientation=45

IDL> XYOutS, 150,180, ‘Research’, Alignment=0.50, $

/Device, CharSize=2.0, Orientation=-45

## 给图形显示添加线和符号

　　给图形添加注释的另一种有效程序是PlotS命令，它是用来在图形显示上添加符号或线条。PlotS命令可在二维或三维空间中使用。

　　用PlotS程序画线，只需提供含有X和Y坐标的矢量，矢量中的X、Y值是需要连接的点的X、Y坐标值。例如，从点（0，15）到点（6，15）在线画图上画一条基线：

IDL> Window, XSize=500, YSize=400

IDL> Plot, time, curve

IDL> PlotS, [0,6], [15,15], LineStyle=2

输出结果应与图28相似。

　　PlotS程序可以用来在任何位置标上符号。下面是在曲线上每五个点处标注一个菱形符号的实例。

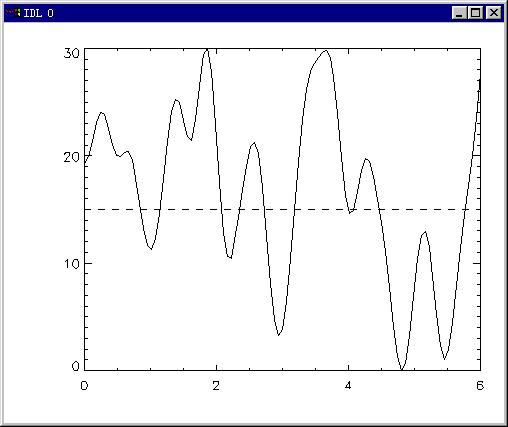
IDL> TvLCT, [70, 255, 0], [70, 255, 250], [70, 0, 0], 1

IDL> Plot, time, curve, Background=1, Color=2

IDL> index = IndGen(20)\*5

IDL> Plots, time[index], curve[index], Psym=4, $

Color=3, SymSize=2

**图28：用PlotS 命令画一条虚线跨过图形的中部**

　　PlotS命令也可以用来在图上重要信息的周围画出一个方框。通过PlotS命令与其它图形命令组合，如XYOutS命令，可以有效地注释图形显示。例：

IDL> TvLCT, [70, 255,0], [70,255,255], [70, 0, 0],1

IDL> Plot, time, curve, Background=1, Color=2

IDL> box\_x\_coords = [0.4, 0.4, 0.6, 0.6, 0.4]

IDL> box\_x\_coords = [0.4, 0.6, 0.6, 0.4,0.4]

IDL> PlotS, box\_x\_coords, box\_y\_coords, Color=3, /Normal

IDL> XYOutS, 0.5, 0.3, ‘Critical Zone’, Color=3, Size=2, $

Alignment = 0.5, /Normal

　　注意，可以使用XYOutS和PlotS命令为图形显示创建图例。

## 图形显示添加色彩

　　另一种有效的标注图形显示的方法是使用颜色。Polyfill命令是一个低级的图形显示命令，它可用特殊的颜色或图案填充任意形状的多边形（无论是在二维或还是在三维环境中定义的）。例如，使用Polyfill命令用红颜色填充上面线画图中方框：

IDL> TvLCT, 255, 0,0 ,4

IDL> Erase, Color=1

IDL> Polyfill, box\_x\_coords, box\_y\_coords, Color=4, /Normal

IDL> Plot, time, curve, Background=1, Color=2, /NoErase

IDL> PlotS, Box\_x\_coords, box\_y\_coords, Color=3, /Normal

IDL> XYOutS, 0.5, 0.3, ‘Critical Zone’, Color=3, Size=2, $

Alignment = 0.5, / Normal

　　颜色有时代表一个数据集的另外一维的特性。例如，可以二维圆形（或多边形）显示XY数据，而每个多边形的颜色就可表现出数据的某些附加特性，比如温度和人口密度等。

　　IDL没有构建圆的模块，但是可以编写这样一个功能模块。打开文本编辑器，键入代码来创建IDL的Circle功能。

FUNCTION CIRCLE, xcenter, ycenter, radius

Points = (2 \* ! PI / 99.0) \* FindGen(100)

x = xcenter + radius \* Cos(points)

y = ycenter + radius \* Sin(points)

RETURN, Transpose([x],[y])

END

　　组成圆周的X和Y值将以2\*100数组形式返回。可以将该数组输入到Polyfill命令中。以Circle.pro保存该程序，并通过键入如下命令进行编译：

IDL> .Compile circle

然后，创建随机分布的X和Y数据。（将Seed设置回初始状态，这样输出结果将与图29看上去相似）

IDL> seed = -3L

IDL> x = RandomU(seed,30)

IDL> y = RandomU(seed,30)

　　将Z值设为这些X值和Y值的函数：

IDL> z = (3 \* ( (x-0.5)^2) + 5\*((y-0.25)^2)) \* 1000

　　打开窗口绘制XY位置，可以看到这些数据以随机形式分布：

IDL> Window, Xsize=400, Ysize=350

IDL> Plot, x, y, Psym=4, Position=[0.15, 0.15, 0.75, 0.95],$

Xtitle=’X Locations’, Ytitle=’Y Locations’

　　以不同颜色的圆显示与XY位置相关的Z数据。需要加入一张颜色表，并且使Z数据缩放至可获得的颜色数的范围内：

IDL> LoadCT, 2

IDL> zcolors = Bytscl(z, Top=!D.Table\_Size-1)

在这个例子里使用的Circle程序有许多弱点。主要缺点是它并非总是生成圆。假如用数据坐标系统来给定圆的坐标，圆形可能将以椭圆的形式显示，主要取决于图形长宽比例以及其它影响因素。（要获得非常棒的圆，可以从NASA Goddard Astrophysics的IDL例库中下载TVCircle程序，可以用浏览器通过http://idlastro.gsfc.nasa.gov/homepage.html来找到该例库）。为避免Circle程序中的这种不足，可用Convert\_Coord命令将数据坐标转换为设备坐标。键入：

IDL> coords = Convert\_Coord (X, Y, /Data, /To\_Device)

IDL> x = coords(0,\*)

IDL> y = coords(1,\*)

　　最后使用Polyfill命令画出表示Z数据的彩色圆：

IDL> For j=0, 29 Do Polyfill, Circle(x(j), y(j), 10), $

/Fill, Color=zcolors(j), /Device

　　最好有一个色棒能够告知Z值和各种颜色的某些关系。可以用本书的Colorbar程序增加一个色棒，键入：

IDL> Colorbar, Position = [0.85, 0.15, 0.90, 0.95], $

Range=[Min(z), Max(z)], /Vertical, $

Format=’(I5)’, /Right, Title=’Z Values’

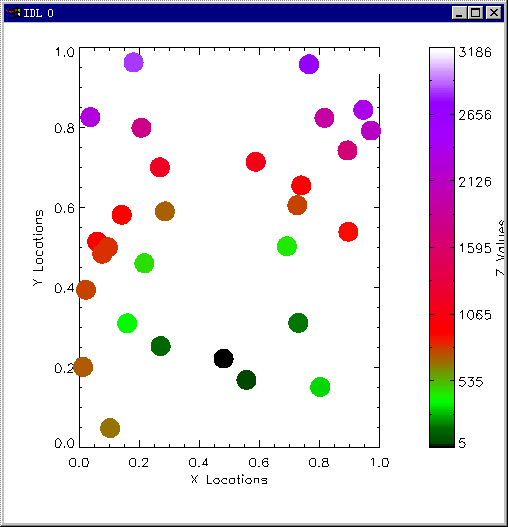
输出结果应与图29相似

图29：在二维图中圆的颜色代表了第三维信息

# 第三章 图像数据处理

## 图像处理

任何类型的二维数据集都可认为是一幅图像。但是要在一个8位的显示设备上显示图像数据，就必须将图像数据调整为 0～255之间的字节型数值。（在一个24-bit的显示设备上，24位图像的RGB值必须调整成字节型数值。）因为图像总是以字节型数值显示，所以图像总是以字节型数组来存储。但是无论图像是怎样存储的，图像总是由两个显示图像的IDL命令：TV和TVScl以字节型数值来显示。

用命令LoadData来打开图像数据集Ali and Dave。将要处理这两幅图像数据中的第二幅图像

IDL>image=LoadData(10)

IDL>image=image[\*,\*,1]

### 显示图像

可用TV和TVScl两个IDL命令中的任一个来显示图像。这两个命令几乎在各个方面都是一样的，包括能与之一起使用的关键字。仅仅在一个方面不同：TVScl将图像数据调整为与IDL运行时所用颜色数目相适应的字节型数值。例如：如果在使用IDL时用220种颜色，则在图像显示之前TVScl将图像数据调整为0～219之间的字节型数值。

另一方面，TV命令取图像数据本身的值，仅仅将其作为字节型数值传送到显示设备上。如果有必要，图像数据将被截断以符合字节型数值。如果图像数据不被调整到0～255之间，图像将很可能显示不正确。

注意，与Plot，Surface和Contour命令不同，TV和TVScl命令在显示图像之前不删除已显示的内容。一般情况下这个问题不大，但有时候也会产生一些麻烦。如果想要一个干净的显示窗口来显示图像数据，无论当前窗口上的显示内容是什么，都可用一个简单的命令Erase来删除。

IDL>Erase

刚才读取的IDL的图像数据集已经调整在0～255之间。

可以键入如下内容来查看：

IDL>Print,Max(image),Min(image)

但是，如果在一个8位显示设备上工作，可能没有全部使用在显示器上可用的256种颜色。如果需要了解正在使用多少种颜色：

IDL>Print,！D.Table\_Size

在一个8位显示器(这里指颜色表的大小)上,运行IDL时所用颜色的数目通常是在210～240之间，显然可用的颜色太少了。在一个24位的显示器上，可以获得1670000种颜色，但颜色表大小仍然是256。

打开一个显示窗口，装上灰度颜色表,　用TV命令显示图像:

IDL>Window,0,XSize=192,YSize=192

IDL>LoadCT,0

IDL>TV,image

用TV命令，数据没有经过调整就被传送到显示器中。图像上所有大于IDL运行时的颜色数目的像素值都被设为同样的值。比！D.Table\_Size-1值大的像素被以相同的颜色显示。（在这种情况下，看到的颜色是灰色明暗图。）

如果用TVScl命令显示图像，也许能看出差别。打开另一个窗口并将其移到第一个窗口的旁边。用TVScl命令显示图像：

IDL>window,1,XSize=192,YSize=192

IDL>TVScl,image

可看到两个图像的明暗程度不同。因为这幅图像数据最大值为238，所以差别是很微弱的。如果看不出差别，可先在0～255之间对数据进行调整：

IDL>west,0

IDL>image=Bytscl(image)

IDL>tv,image

IDL>west,1

IDL>tvscl,image

如果仍不能看到差别，可装入颜色表。Red Temperature颜色表可能起作用：

IDL>LoadCT,3

如果要了解TVScl作了些什么，可调整数据并用TV命令显示：

IDL>Window,2,XSize=192,YSize=192

IDL>scaled=Bytscl(image,Top=!D.Table\_Size-1)

IDL>TV,scaled

在窗口2中看到的图像应与窗口1中的图像一样。这就是所说的，TVScl将数据调整为与IDL运行时所用颜色数目相适应的字节型数值。

注意：如果在显示窗口的图像不是用red\_temperature颜色表显示的话，则可能是在一个16位或24位彩显上使用IDL。

确保关闭颜色分解器：

IDL>Device,Decomposed=0

IDL>TV,scaled

如果用的是一个16位或24位显示器，为了看到新的颜色生效，在改变颜色表后，需要重新运行每个图形命令。在一个16位或24位显示器上，颜色表中的颜色没有直接被索引或连接到显示器上的色彩表。何况颜色表是图像用来查找每个像素该使用哪种颜色的一种方法。而像素的颜色是直接表示的。

一般来说，如果不知道数据是否被调整过，很可能想用TVScl命令，因为这将给图像像素值以最大可能的对比度。但是如果颜色是重要的话，那么可能不会用TVScl命令。相反，将自己缩放图像数据，然后用TV命令来显示。

### 调整图像数据

假设正在测量大气压，并将测量数据在一色棒旁边以图像显示。可能想比较这个星期收集的图像数据和上个星期收集的图像数据。换句话说，想确定一种特定的颜色，比如红色，在这套数据中的红色和上个星期的数据中的红色表示相同的压力。

如果用TVScl命令显示这个星期和上个星期的图像数据，绝对不能保证特定的红色在两个数据组中能代表同一事情。

这些出入来自两个原因。第一，可能今天使用IDL时的颜色数目和上个星期使用IDL时不同。因为TVScl将图像数据调整到IDL运行时的颜色数目内，这可能会引起错误。第二，不能确保两组数据组间具有相同的数据范围。因而，用TVScl调整可能再次引起错误。

为解决这些问题，可用BytScl命令调整数据，并用TV命令显示。为确保IDL运行时所用的颜色数目不引起错误，可将数据调整到相同的颜色值内。并且，为确保数据集中数据的范围不引起错误，可以将数据调整到同样的数据范围。

可通过BytScl命令，应用关键字Top，Min和Max实现上述要求。例如，假设总是想以100种不同的灰度深浅或颜色深浅来显示数据，并且假设在任何数据集中希望最小数据值为15，而最大的有效值为245。可用如下BytScl命令实现：

IDL>scaledImage=BytScl(image,Min=15,Max=245,Top=99)

这个例子中， 数据调整之前在数据集中小于15的数值将设定为15。类似地，在数据调整之前，在数据集中任何大于245的数值将被设定为245。一旦数据被调整了，就可用TV命令显示。

IDL>TV,scaledImage

如果总是这样调整数据集（并且在IDL运行时总是有至少100个灰色级别或颜色数），那么上个星期的数据集就能直接与这个星期的数据集比较。一个特定的颜色，红色将总是表示一个特定的数据范围或压力。

#### 将图像调整到颜色表的不同部位

需要知道如何调整图像数据的另一个理由，是要能在使用8位显示器时，将数据调整到颜色表的不同部位。这使图像能用不同颜色显示出来，或者能将颜色表的特定色段用于特别的目的。例如，也许想将颜色表的一部分保留作为画图用的颜色。

注意：用24位彩显的一个很大的好处是能随时使用一个没有限制的颜色表。24位彩显的缺点是，在改变颜色表之后，为了看到新颜色生效，不得不重新运行图形命令（例如：TV命令）。

在大多数8位显示器上仅仅有一个物理颜色表，并且所有的IDL图形窗口都用它。但是通过操作颜色表可以让它看上去象是同时装入几个不同的颜色表。可以通过将不同的颜色表装入到一个物理颜色表的不同部位来实现这一点。也许实现这点的最简单的方式是在LoadCT或XLoadCT命令中用NColors和Bottom关键字。

例如，假设想用两个看上去不同的颜色表来显示同一幅图像。在用IDL打开一个图形窗口后，能通过测试系统变量！D.Table\_Size的值知道在IDL运行时颜色表中有多少种颜色。如果将这个数目一分为二，就知道每个图像该用多少种颜色：

IDL>half=!D.Table\_Size/2

为了在同一窗口用看上去不同的两个颜色表显示图像数据image，必须将图像数据调整为适应两个颜色空间范围的值。首先，用BytScl命令调整图像数据为适应第一个部分颜色表的值，生成一个新的图像image1：

IDL>image1=BytScl(image,Top=half-1)

现在，按如下做法将图像数据调整为适应第二个部分颜色表的值，生成第二个图像image2：

IDL>image2=BytScl(image,TOP=half-1)+Byte(half)

按如下做法将两个已调整的图像肩并肩地放在同一个窗口

IDL>Window,XSize=192\*2,YSize=192

IDL>TV,image1

IDL>TV,image2,192,0

现在需要用一个灰度颜色表(颜色表索引号为0)将左边的图像显示出来。必须将那些灰度级颜色装入颜色表中被第一个图像数据占用的部分：

IDL>LoadCT,0,NColors=half,Bottom=0

如果用XLoadCT命令将颜色装入颜色表的第二部分，就能为右边的图像交互式地选择想要的任何颜色表：

IDL>XLoadCT,NColors=half,Bottom=half

恢复一个正常的颜色表：

IDL>LoadCT,0

#### 在24位显示器上用不同的颜色表显示图像

在16位或24位显示器上运行时，使用不同的颜色表和装入颜色并显示图像：

IDL>world=LoadData(7)

IDL>Window,1,Title=‘Gray Scale Image’

IDL>LoadCT,0

IDL>TV,world

IDL>Window,2,Title=‘Color Image’

IDL>LoadCT,5

IDL>TV,world

### 显示24位图像

真彩色（或24位）图像也能用TV命令显示。24位图像总是由一个3维数据集构成，它的3个维数中的一个值设为3。例如，数据集可以是一个m\*n\*3的数组，这种情况下，图像被认为是隔波段扫描（band-interleaved）；如果图像是m\*3\*n则被认为是隔行扫描（row-interleaved）；如果是3\*m\*n则被认为是隔像素扫描（pixel-interleaved）。

装载一幅24位图像，键入如下命令：

IDL>rose=LoadData(16)

这个数据组是一个按像素扫描的图像

IDL>Help,rose

　　　ROSE BYTE =Array[3,227,149]

要在一个8位显示器上显示一幅24位的图像，需要用关键字True来说明其用的是哪种扫描方式。True=1为隔像素扫描；True=2为隔行扫描；True=3为隔波段扫描。

IDL>Window,XSize=227,YSize=149

IDL>TV,rose,True=1 ;Pixel-interleaved

注意，24位图像在8位显示器上显示将表现为灰度级。要在这样的显示器上看到真彩色的图像，需要创建一幅2维图像以及伴随该24位图像或3维图像数据的红色、绿色、蓝色颜色表。这在IDL中可用命令Color\_Quan来实现。如果使用8位显示器，键入如下命令：

IDL>image2d=Color\_Quan(image24,1,r,g,b)

IDL>TVLCT,r,g,b

IDL>TV,image2d

#### 在24位显示器上显示24位图像

正确显示一幅24位图像，必须打开颜色分解器。这在大多数真彩模式下的工作站上自动实现的，但在真彩模式Windows下，IDL5.2版本却不能自动实现。确保以正确的图像颜色显示24位图像：

IDL>Device,Decomposed=1

IDL>TV,image24

#### 在24位显示器上显示8位图像

在24位显示器上，8位图像在显示时遍历了整个颜色表。换句话说，一个8位图像的像素值被作为一个索引号，该索引号为给定的像素查找特定的红色，绿色和蓝色。这意味着如果在使用IDL时改变了颜色表，必须重新显示该2维图像来看新的颜色是否生效。这是因为在24位显示器上颜色是在图像被显示时决定的，同时也因为正在用RGB颜色模式。并且特别要注意必须关上颜色分解器，否则将忽视颜色表矢量，并总是用灰度色彩来显示8位图像。如果用的是一个24位显示器，键入如下命令：

IDL>world=LoadData(7)

IDL>Window,XSize=360,YSize=360

IDL>LoadCT,5

IDL>Device,Decomposed=0

IDL>TV,world

　　为了以另一种颜色表显示图像，装入该颜色并重新运行TV命令，使图像像素值遍历整个颜色表矢量。注意当只运行LoadCT命令时，图像颜色不变。

IDL>LoadCT,3

IDL>TV,world

### 控制图像显示顺序

通常，当IDL显示一幅图像时，习惯上图像的第0列和第0行为图像的左下角。如果要将图像的第0列和第0行作为图像的左上角，可以通过设置系统变量！Order。默认时，！Order为0。如果希望将所有图像的左上角都显示在第0列和第0行，可设置！Order=1。

如果只是希望用第二种方式显示某幅图像，可在使用TV或TVScl命令时，用关键字Order设置。例如，可以同时观看两种显示方式：

IDL>Window,XSize=192\*2,YSize=192

IDL>TVScl,image,Order=0

IDL>TVScl,image,Order=1,192,0

　　可能从别人那得到一个图像数据文件，显示时倒过来了。这大多是因为创建数据文件的人在排放第0列和第0行时用了不同的习惯。将关键字Order的值反过来，看是否纠正了错误.

### 改变图像尺寸

IDL提供了两个改变图像大小的命令：Rebin和Congrid 。

Rebin的限制为新建图像的尺寸必须是原始图像尺寸的整数倍或整数比例。例如，变量image可以在X方向或Y方向上变化为192/2和192\*3个元素。但不应该是300或500个元素。图像大小也可以在一个方向减小，另一个方向增大。例如，可将变量image重新变化为384列和96行，键入如下命令。

IDL>Window,XSize=384,YSize=96

IDL>new=Rebin(image,384,96)

IDL>TVScl,new

输出图像应与图31类似。



**图31：用Rebin命令缩放的图像其大小必须与原始图像大小有整数倍关系。**

在默认情况下，当放大一幅图像时Rebin采用双线性插值，当缩小一幅图像时则采用最邻近平均法。如果关键字Sample被设定后，在两个方向上都可用最邻近采样法。双线性插值更为精确，但需要更多的计算时间。

IDL>Window,XSize=192/2,YSize=192/2

IDL>new=Rebin(image,96,96,/Sample)

IDL>TVScl,new

除了下面两个方面外，Congrid与Rebin是相似的。第一，在新图像中的列数和行数可以设为任意值。第二，在默认情况下，用的是最邻近采样法。如果想用双线性插值，必须设置关键字Interp：

IDL>Window,XSize=600,YSize=400

IDL>new=Congrid(image,600,400,/Interp)

IDL>TVScl,new

#### 在PostScript设备上改变图像大小

PostScript设备，其像素是可调节的（相对于固定像素的显示器来说），在调节图像尺寸时有所不同可不用Rebin或Congrid命令来改变图像的大小，而是用TV或TVScl命令通过关键字XSize和Ysize来改变图像的大小。

例如，将图像输出到一个PostScript文件时,如果想将图像的显示比例定为6：4的话，可用如下代码，而不是用Congrid命令将图像放大为600\*400。

IDL>thisDevice=！D.Name

IDL>Set\_Plot,'PS'

IDL>Device,XSize=6,YSize=4,/Inches

IDL>TVScl,image,XSize=6,YSize=4,/Inches

IDL>Set\_Plot,this Device

如果图像大小和位置是用下面的归一化坐标来表示的，可以写出真正的独立于显示设备的图像显示代码。

### 在显示窗口中定位图像

通常显示一幅图像时，IDL将图像的左下角放在窗口的左下角。但是可通过TV或TVScl命令的附加参数来将图像移动到显示窗口中的其它位置。

例如，如果给出第二个参数，它则被视为图像在窗口中的位置。图像位置由显示窗口的尺寸和图像的尺寸计算出来的。

　　位置可从显示器的左上角开始，一直到显示器的右下角。例如，在384＊384的显示窗口内，从显示器的左上角开始，对于192＊192的图像来说有四种位置：

IDL>Window,XSize=384,YSize=384

IDL>TVScl,image,0

IDL>TVScl,image,1

IDL>TVScl,image,2

IDL>TVScl,image,3

通过显式地指定图像左下角的象素位置来定位一幅图像是可以的。TV或TVScl命令中在图像数据名后给定两个附加参数即可以实现这点。例如，将一幅192＊192名为image的图像定位于刚刚创建的显示窗口中间：

IDL>Erase,Color=!D.Table\_Size-1

IDL>TVScl,image,96,96

这样，将图像的左下角放在像素点（96,96）处。当希望为附加图留下空间时，如色棒或其它的注释，这种定位图像的方法是很重要的。

例如，键入如下命令来在窗口的左边显示一个色棒，在窗口的右边显示图像。显示窗口将如图32所示。

IDL>Window,XSize=320,YSize=320

IDL>ncolors=！D.Table\_Size

IDL>TvLCT,255,255,0,ncolors-1

IDL>Erase,color=ncolors-1

IDL>colorbar=Replicate(1B,20)#BIndGen(256)

IDL>TV,BytScl(colorbar,Top=ncolors-2),32,36

IDL>TV,BytScl(image,Top=ncolors-2),92,64



**图32：此图像用颜色棒来显示其颜色梯度**

#### 用归一化的坐标来定位图像

设想将图像放入一个任意的窗口内，并占满其比如80%的空间。相对于归一化坐标来说，可将图像在窗口的位置表达为：

position=[0.1,0.1,0.9,0.9]

显示窗口中可视部分的大小由系统变量！D.X\_VSize和!D.Y\_Vsize以设备或像素单元来给定。

通过像素坐标，可以按如下方法计算出图像所需的尺寸和在输出窗口起始的位置：

xsize=(position[2]-position[0])\*！D.X\_VSize

ysize=(position[3]-position[1])\*！D.Y\_VSize

xstart=position[0]\*！D.X\_VSize

ystart=position[1]\*！D.Y\_VSize

将图像输出到显示设备和输出到PostScript文件的唯一区别是如何确定图像的尺寸。可以编写如下代码来显示图像:

IF !D.Name EQ 'PS' THEN $

TV, image, XSize=xsize, YSize=ysize, xstart, ystart $

ELSE $

TV, Congrid(image, xsize, ysize), xstart, ystart

无论是将图像输出到显示器还是输出到PostScript文件中, 上述代码都起作用。但这样做时图像的横纵比例不能得到保证。可以让图像适合窗口的形状。

TVImage用关键字Position来定位图像和确定图像大小，。如果希望TVImage程序能完好地保持显示图像的横纵比例，可以使用关键字Keep\_Aspect\_Ratio。

可以用TVImage重新生成色棒位于图像左边的上述图像：

IDL>Erase,color=ncolors-1

IDL>barPosition=[32,32,52,292]/320.0

IDL>imagePosition=[92, 64, 284, 256]/320.0

IDL>colorbar=Replicate(1B, 20)#BIndGen(256)

IDL>TVImage,BytScl (colorbar,Top=ncolors-2),$

Position=barPosition

IDL>TVImage,BytScl (image,Top=ncolors-2),$

Position=imagePosition

图像可以在任何尺寸的窗口或PostScript文件中以及显示器上显示，而且在显示窗口中增加其它图形成为可能。例如，可以在色棒和图像周围放置外框或标记：

IDL>TvLCT,255,255,255,ncolors-1

IDL>Plot,[0,！D.Table\_Size,YRange=[0,！D.Table\_Size],$

/NoData,Color=0,Position=barPosition,XTicks=1,$

/NoErase, XStyle=1, YStyle=1, XTickFormat='(A1)'?$

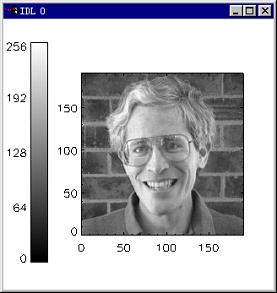
YTick=4

IDL>Plot, IndGen(192), IndGen(192), /NoData, $

Position=imagePosition, /NoErase, $

XStyle=1, YStyle=1, Color=0

输出结果应如图33所示。



**图33：用命令TVImage不仅允许使用独立于设备的方法定位图像而且容易使用其它图形命令**

### 从显示器中读取图像

将图形显示读到一个图像变量中以便处理甚至拷贝输出，可以用TVRD命令将IDL图形窗口的内容读到一个2维IDL字节型数组中。

在一个8位显示器上读取整个图形窗口：

IDL>Window,XSize=250,YSize=250

IDL>TVScl,image

IDL>new\_image=TVRD()

IDL>Help,new\_image

#### 在24位显示器上截屏

如果在16位或24位显示器上运行IDL，不能像上面那样使用TVRD命令。16位或24位显示器有3个颜色通道。如果像上面不用任何参数来使用TVRD命令，那么得到的2维数组的每个像素值为该像素三个通道中的最大像素值。除非装载一个灰度色彩表（这样，每个通道有同样的值）。在24位显示器上截屏，可在命令TVRD中设置关键字True。

IDL>new\_image=TVRD(True=1)

　　得到的是一个24位图像，而不是一个2维8位图像。显示该图像：

IDL>Help,new\_image

IDL>Erase

IDL>TV,new\_image,True=1

#### 读取显示图像的一部分

如果只想读显示窗口的某一部分，可指定想要的部分窗口的左下角的像素坐标和要读取的列和行的数目。指定矩形区域：

IDL>new\_image=TVRD(40,30,110,130)

在IDL中，从显示器读取的2维数组可以像处理其它图像那样进行。显示：

IDL>Erase

IDL>TV,new\_image

## IDL中基本的图像处理

### 直方图均衡化

如果观察图像中的像素值分布，往往会发现分布趋向集中在一个狭窄的数值范围内。实际上，图像有一个非常窄的动态颜色范围。如果像素分布开，以致使像素值的每个子范围都与这些像素值一样拥有数目大约相同的像素，则该图像的信息内容就有可能增加。将像素分布到整个颜色范围的过程叫做直方图均衡化。

例如，用LoadData命令打开数据集CT Scan Thoracic Cavity。这是一幅CT扫描图像，该图像具有一个狭窄的动态颜色范围。

IDL>scan=LoadData(5)

要看变量scan的像素值分布的柱状图：

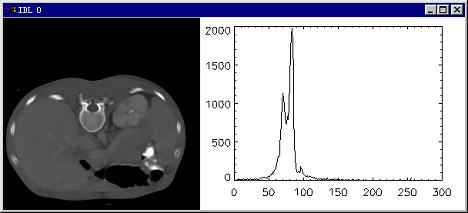
IDL>LoadCT,0

IDL>Window,0,XSize=600,YSize=250

IDL>TV,scan

IDL>Plot,Histogram(scan),/NoErase,Max\_Value=5000,$

Position=[0.5,0.15,0.95, 0.95]

****

**图34：正常图像具有狭窄的像素值分布。这里的像素值集中在50-10之间。**

可看到大多数像素值落在50～100之间。将像素分布至整个颜色范围内，使得每种颜色值都有大致相同的像素个数：

IDL>equalized=Hist\_Equal(scan)

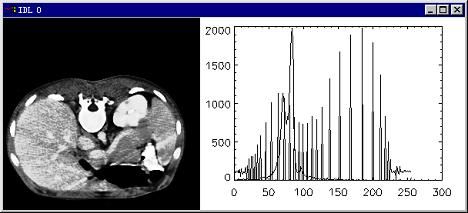
查看新的像素分布柱状图 和 HISTOGRAM-EQUALIZED图像：

IDL>Window, 1, XSize=600, YSize=250

IDL>TV, equalized

IDL>Plot, Histogram(equalized), Max\_Value=5000, $

Position=[0.5, 0.15, 0.95, 0.95], /NoErase

****

**图35：一幅直方图均衡化后的图像。象素分布扩展到了整个颜色范围。**

### 平滑图像

可以通过将每个像素值与它周围相邻像素值进行平均来平滑图像。均值平滑由IDL中的功能函数Smooth完成的，它是在给定的奇数宽度的范围内实现等加权值平滑。例如，如果周围是3\*3宽度，那么每个像素由它和它的周围八个像素值的平均值代替。

比较一幅没有经过平滑处理的图像和经过5\*5方盒的均值平滑处理后的图像：

IDL>Window,0,XSize=192\*3,YSize=192

IDL>TV,image,0,0

IDL>smoothed=Smooth(image,5,/Edge\_Truncate)

IDL>TV, smoothed, 192, 0

注意，与命令Smooth一起使用的关键字Edge\_Truncate。该关键字可复制图像边缘附近的像素，以便实现整幅图像的平滑。如果不使用该关键字，图像边缘附近的像素仅仅是简单复制，而没有平滑。

图像平滑被应用在一种称作晕光蒙片的图像处理技术中。这种技术可用作定位图像上的棱边或者是像素值突然变化的地方。从未平滑的图像中减去平滑的图像：

IDL>TV, ((image-smoothed)+255)/2.0, 2\*192, 0



**图36：左边为原始图像，中间为平滑处理过的图像，右边为经晕光蒙片处理后的图像。**

用Smooth命令，赋给相邻的像素值相等的权值来计算平均值。有时会出现模糊图像。另一种方式是用卷积的过程来平滑图像。这种技术中，一个方形内核和图像一起参与卷积计算。例如，在3\*3的情况下，Smooth命令使用的内核为：

1 1 1

1 1 1

1 1 1

如果给予中心像素值更大的权值，而它周围像素值的权值小一些，图像就不会那么模糊了。可以创建如下的一个核心：

1 2 1

2 8 2

1 2 1

通过Convol命令用上述内核对图像进行卷积处理：

IDL>kernel=[[1,2,1], [2,8,2], [1,2,1]]

IDL>TV, image, 0, 0

IDL>TV, Smooth(image, 3, /Edge\_Truncate), 192, 0

IDL>TV, Convol(image, kernel, Total(kernel), $

/Edge\_Truncate), 2\*192, 0

可以创建任意大小的内核。如下是一个典型高斯分布的5\*5内核：

1 2 3 2 1

2 7 11 7 2

3 11 17 11 3

2 7 11 7 2

1 2 3 2 1

可将上述内核应用于图像处理：

IDL>kernel=[[1,2,3,2,1], [2,7,11,7,2], [3,11,17,11,3], $

[2,7,11,7,2], [1,2,3,2,1]]

IDL>TV, Convol(image, kernel, Total(kernel), $

/Edge\_Truncate), 192\*2, 0

#### 消除图像噪声

噪声的一般表现形式是黑白点相间噪声，其中一些随机的像素有极端的像素值。首先创建一幅噪声图像。用以前的图像将10%的像素转换为黑白点相间噪声：

IDL>noisy=image

IDL>points=RandomU(seed, 1800)\*192\*192

IDL>noisy(points)=255

IDL>points=RandomU(seed, 1800)\*192\*192

IDL>noisy(points)=0

在原始图像的旁边创建一个窗口并显示噪声图像:

IDL>Window,XSize=192\*3,YSize=192

IDL>TV,image,0,0

IDL>TV, noisy, 192, 0

IDL中的Median命令是从图像上消除黑白点相间噪声的很好选择。Median命令与Smooth命令类似。不同之处是Median命令计算相邻像素的中间值,而不是平均值。这就有两个重要作用。第一，它能删除图像中的极端值。第二，它不使那些尺寸比邻域范围大的图像棱边或特征变模糊：

IDL>TV, Median(noisy, 3), 2\*192, 0

图形显示应如图37所示。



**图37：左边为原始图像，中间为噪声图像，右边为用中值滤波器平滑处理后的噪声图像。**

### 增强图像棱边

一个图像可以锐化或通过微分来增强图像棱边。IDL提供了两个棱边增强函数：Roberts和Sobel。还有一些其它方法也可用来增强图像棱边。例如，可以用拉普拉斯算子来和图像做卷积：

1 1 　　1

1 -7 1

1 1　　 1

因为改进了图像棱边的对比度，这也常常被称为拉普拉斯(Laplacian）锐化操作：

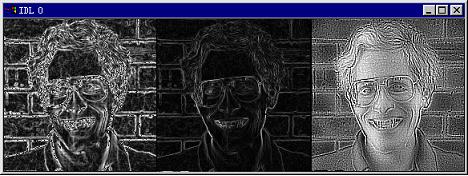
IDL>TV, Sobel(image), 0

IDL>TV, Roberts(image), 1

IDL>kernel=[[1,1,1], [1,-7,1], [1,1,1]]

IDL>TV, Convol(image, kernel), 2

图形显示应如图38所示。



**图38：三种增强图像棱边的方式。左边用的是Sobel方法。中间用的是Roberts方法。右边是用拉普拉斯算子对图像做卷积。**

### 图像的频域滤波

频域滤波是常规的图像和信号处理技术。它可以用来平滑处理图像，锐化图像，降低图像的模糊程度，和恢复图像。

频率域滤波有如下三个基本步骤：

1. 用快速傅里叶变换（FFT）将图像从空间域转变为频域。

2. 将转换后的图像乘以一个频率滤波器。

3. 将滤波后的图像返回为空间域。

这些步骤在IDL中是用快速傅里叶变换（FFT函数）完成的。（如果命令FFT的第二个定位参数为-1，则图像由空间域转变为频域。如果参数为1，则图像将相反转化）。频率滤波命令的一般形式如下：

filtered\_image=FFT(FFT(image, -1)\*filter, 1)

在这种情况中，image既可是一维矢量，也可以是一幅二维图像。滤波器是用来滤波图像中某些特定频率的一维矢量或二维数组。下面将详细介绍。

#### 创建图像滤波器

在IDL中用基于数组的操作和函数很容易创建图像数字滤波器。许多普通的滤波器利用了所谓的频率图像或欧氏距离图的优点。一幅二维图像的欧氏距离图是一个与图像有同样大小的数组。距离图的每个像素被赋给一个值，这个值等于它到二维数组最近的角的距离。在IDL中的Dist命令用作创建欧氏距离图或频率图像：

IDL>Surface,Dist(40)

在频域滤波中所用的滤波器一般为Butterworth频率滤波器。如下的方程给出一个低过Butterworth频率滤波器一般形式：

filter=1/[1+C(R/R0)2n]

其中，常量C等于1.0或0.414[这个值将滤波器的幅度在R=R0时定义为50%或1/Sqrt(2)],R为频率图像，R0为给定的滤波器截止频率（实际中由像素宽度代替），n是滤波器的阶数，通常为1。

高过Butterworth滤波器由如方程给出：

filter=1/[1+C(R0/ R)2n]

要将频域滤波器应用到图像中，可用命令LoadData来打开图像Earth Mantle Convection。这是一个248\*248的二维数组。

IDL>convec=LoadData(11)

用如下命令来打开一个窗口，装入颜色表Standard Gramma II，并在左上角显示原始图像：

IDL>Window,0,XSize=248\*2,YSize=248\*2

IDL>LoadCT,5

IDL>TV,convec,0,248

在频域滤波中的第一步是用函数FFT将图像由空间域转换为频率域：

IDL>freqDomainImage=FFT(convec,-1)

通常，低频项代表一般的图像形状，高频项对图像增加细节。浏览频率域的图像通常是没有意义的，但有时对观察频域图像的功率谱有用。

功率谱是一幅频域图像中不同组成部分的幅度图。与源点（通常代表图像的中心）不同的距离代表不同的频率，相对源点不同的方向代表在原图像特征的不同方向。每个位置的功率表明该频率的大小和以在图像中的方向。功率谱对分离图像中的周期性结构或噪声是特别有用的。功率谱的幅度通常用对数座标表示，因为功率从某个频率到下一频率的变化非常大。

计算出这幅对流图像的功率谱，并将其显示在原始图的相邻位置上：

IDL>power=Shift(Alog(Abs(freqDomainImage)),124,124)

IDL>TV,power,248,248

功率谱中的对称性表明这个图像上在越来越多的频率中包含了许多周期性结构。输出应该类似于图39）。这个练习的目的是过滤掉图像中较高的频率。

接下来的一步是用频率滤波器转换图像。ButterWorth低通滤波器用于滤出图像内的高频成分。这些高频成分为图像提供详细信息，所以最终的结果是完成图像的平滑处理。创建低通频率滤波器可键入：

IDL>filter = 1.0 / (1.0D + Dist(248)/15.0)^2

注意，截止频率宽度是15个像素。这是足以删除高频的一半。在图39中功率谱可以看到。

使用该频率滤波器，再将图像由频率域转换回空间域，最后显示滤波后的图像。键入：

IDL>filtered = FFT(freqDomainImge \* filter, 1)

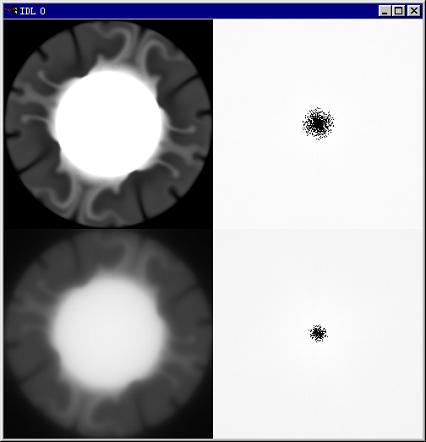
IDL>TV, filtered, 0, 0

为了让自己看到滤掉高频成分的图像，可以显示滤波后图像的功率谱，并在其旁显示滤波后的图像，键入：

IDL>filteredFreqImg = FFT(filtered, -1)

IDL>power = shift(Alog(Abs(filteredFreqImg)), 124, 124)

IDL>TV, power, 248,0

****

**图39：频域滤波器的图解。在图的上半部分是没有滤波的图像，左边是它的功率谱。在图的下半部分是滤波后的图像，相邻的左边是它的功率谱。注意，大约一半高频成分在滤波后的图像中已经被消除，消除了很多图像细节信息，也平滑了图像。**