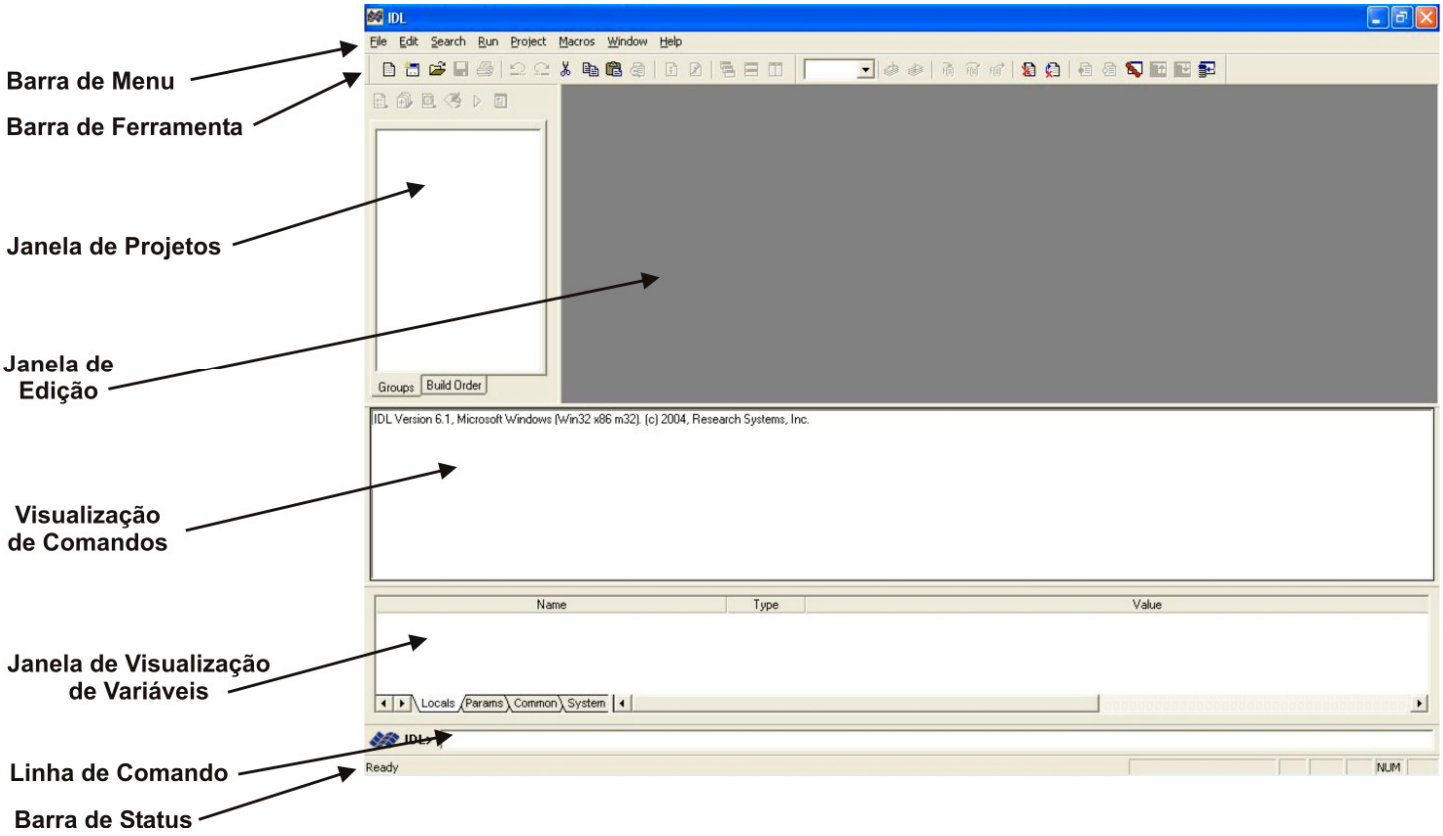
Organização do Guia

Esta é uma tentativa de facilitar a compreensão deste manual de IDL. Foi utilizado um estilo de manual típico, como segue abaixo.

Área	Formatação Guia
Código do Programa IDL	A fonte em minúsculo é usada neste manual para os programas de IDL. Códigos escritos em um programa do arquivo ou em uma mudança de código existente é escrito em **negrito**.
Código da Linha de Comando	O código a ser digitado pelo usuário na Linha de Comando do IDL é apresentado com **IDL**>.
Unidades do programa	Quando referenciado no texto, os programas em IDL ou escritos pelo usuário estarão em maiúsculo e em itálico; ex: *PLOT*, *HELP*, *PRINT*.
Arquivos	Arquivos e diretórios do computador são listados em negrito; ex: **cindex.pro, training**.
Variáveis e Tipo de Variáveis	Quando referenciadas no texto, as variáveis e variáveis de sistema serão definidas com o itálico; ex: *pstate*, *!pi*.
Palavras chaves	Quando referenciado no texto, palavras chaves e palavras chaves reservadas estarão em maiúsculo; ex: XSIZE, TITLE.
Comentários	Os Comentários são marcados com ponto e vírgula (;)
Espaço	Endentação e linhas vazias são usadas para ajustar deliberadamente o código fora das seções relacionadas.
Continuação da Linha	O sinal do dólar $ no final da linha indica que a instrução atual continua na próxima linha. O sinal do dólar $ pode aparecer e qualquer lugar exceto dentro de uma palavra ou entre o nome de uma função e o primeiro parênteses aberto. É permitido um número ilimitado de continuação de linhas.
Componentes do IDLDE	Os componentes do IDLDE estarão em negrito e itálico; ex: ***Barra de Menu***, ***Linha de Comando***.

Ambiente de Desenvolvimento do IDL

O ambiente de desenvolvimento do IDL (IDLDE) é uma interface gráfica que fornece ao usuário ferramentas para edição e depuração de erros para o IDL.



Uma breve descrição da funcionalidade dos componentes do IDLDE estão listados na tabela abaixo.

Componentes IDLDE	Funcionalidade
Barra de Menu	Menus para aberturas, edição, compilação e execução de programas em IDL.
Barra de Ferramentas	Controles gráficos com a funcionalidade similar da ***Barra de Menu***.
Janela de Projetos	Uma ferramenta para facilitar o agrupamento dos programas e arquivos de dados do IDL.
Janela de Edição	Onde os programas de IDL são escritos e editados.
Visualização de Comandos	Usado pelo IDL para retornar ao usuário informações sobre execuções.
Janela de Visualização de Variáveis	Apresenta todas as variáveis que estão atualmente abertas na sessão do IDL.
Linha de Comando	O lugar onde são inseridos comandos que não fazem parte da Janela de Edição.
Barra de Status	Indica o status atual do IDLDE.

Observações do IDL em plataformas com base UNIX

Nas plataformas com base UNIX, o IDLDE pode ser executado digitando
$ idlde

em uma janela de comando. Uma versão de linha de comando do IDL pode ser executada digitando
$ idl

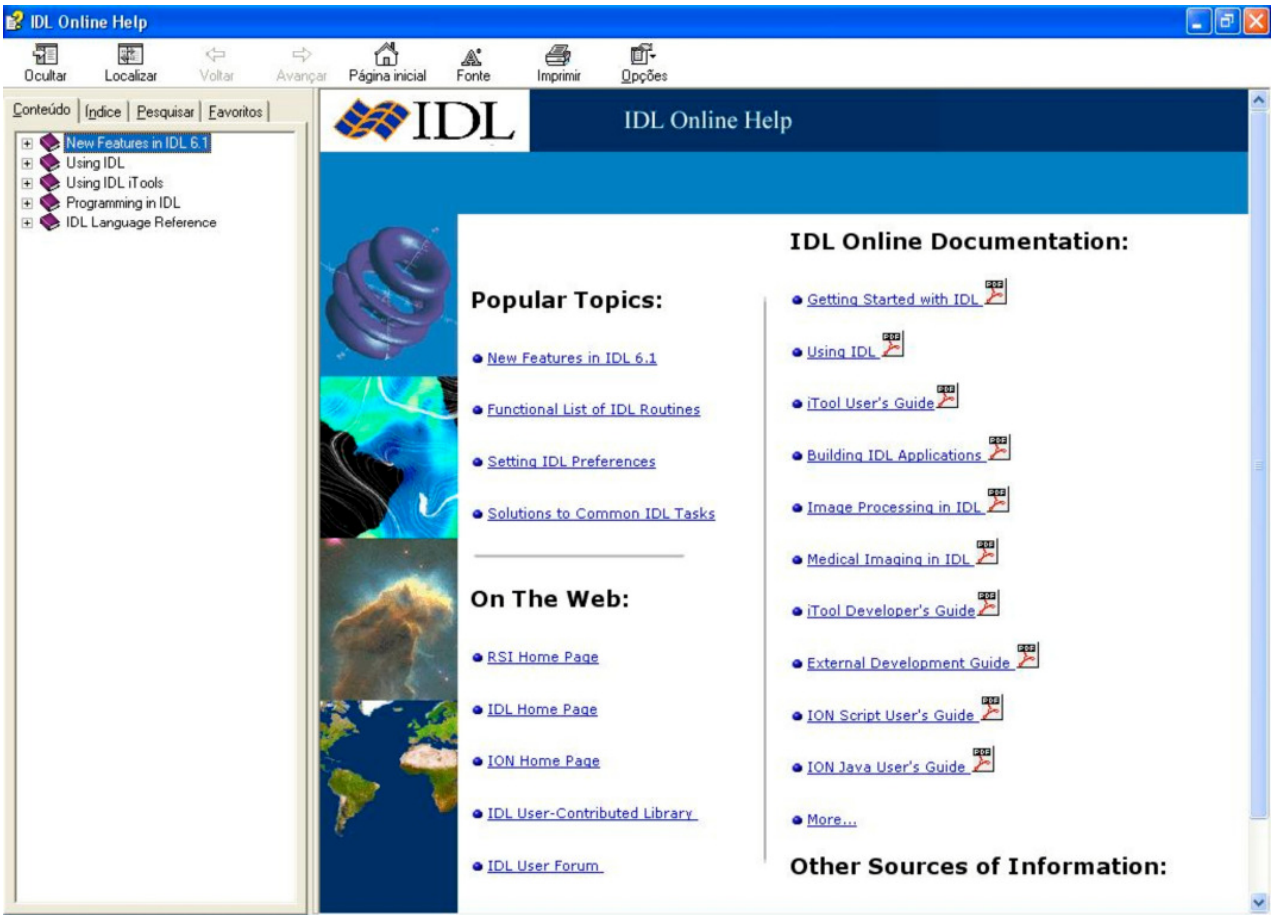
com a versão de linha de comando, você pode usar seu editor favorito para criar e ou editar programas do IDL.

IDL Online Help

O IDL é equipado com um sistema de documentação extensivo, que é denominado IDL Online Help. Com essa ferramenta, você terá uma documentação detalhada de todas as rotinas do IDL também de suas palavras-chave, facilitando assim a forma de aprendizado desta linguagem, por outro lado o IDL Online Help é todo em inglês.

Para executar esta ferramenta basta você digitar um ponto de interrogação na ***Linha de Comando*** do IDLDE.

IDL> ?



ou selecionar **Help > Contents...** na ***Barra de Menu*** do IDLDE.
Nas plataformas com base UNIX, a ferramenta pode ser iniciada digitando em uma janela de comando:

```
$ idlhelp
```

Na página inicial do IDL Online Help temos links para arquivos em PDF contendo mais informações de um determinado assunto se precisar-mos, ao meu ver esta é uma das ferramentas mais úteis do IDL para quem está começando a trabalhar com ele.

Comandos de Execução

Comandos de execução são instruções utilizadas para compilar, executar, executar passo a passo e parar os procedimentos, funções e programas principais.

Comando	Descrição
.compile	Compila os módulos do programa
.continue	Continua a execução de um programa que havia sido interrompido devido a algum erro, uma instrução de parada, ou uma interrupção pelo teclado.
.edit	Abre os arquivos na ***Janela de Edição*** do IDLDE.
.go	Executa um programa previamente compilado.
.out	Continua a execução do programa até que o módulo atual retorne ao seu chamador.
.reset_session	Reinicia sistema de memória do IDL, removendo variáveis, funções e procedimentos compilados do usuário.
.return	Continua a execução até uma instrução de retorno ser encontrada.
.run	Compila procedimentos, funções e programa principal do IDL, executa programa; se o arquivo for chamado sem um nome, ele pode ser usado para editar, compilar e rodar um programa de nível principal na ***Linha de Comando***.
.rnew	Similar ao .run, exceto por todas as variáveis do usuário serem apagadas antes do novo programa principal ser executado.
.skip	Pula sobre um especifico número de instruções no programa atual.
.step	Executa um especifico número de instruções no programa atual, até que pare.
.stepover	Chama outra unidade de programa.
.trace	Similar ao .continue, mas ele indica cada linha do código antes dele ser executado.

Tipos de Programas

IDL suporta três tipos de programas: principal, procedimentos e funções. Procedimentos e funções são a chave para modelar a programação estruturada. Encorajamos você a escrever procedimentos e funções ao invés de escrever tudo no programa principal.
Todos programas precisam ser compilados antes de serem executados. Compilar é o ato de interpretar as instruções do código fonte para um arquivo de código byte armazenado na memória. O código byte na memória que é executado quando você executa o programa.

Principal (Main Program)

Os programas principais estão incorporados a ***linha de comando*** do IDL, e são úteis quando você tem alguns comandos que você quer executar sem ter que criar um arquivo separado contendo-os. Programas principais não são explicitamente nomeados; consistem em uma série de instruções que não são precedidas por um cabeçalho como no procedimento ou na função. Entretanto, requerem uma instrução END. Por não ter um cabeçalho, o programa não pode ser chamado por outras rotinas e não pode ser passado como argumento. Quando o IDL encontra um programa principal como o resultado de um comando executivo .RUN, compila-o em um programa especial nomeado $MAIN$ e executa-o imediatamente. Mais tarde, ele poderá ser executado novamente usando o comando executivo .GO.

O exemplo a seguir cria um programa principal simples usando a ***linha de comando*** do IDL:

1. Na ***linha de comando*** do IDL, digite:
```
X=2
```
2. Digite .RUN na ***linha de comando*** do IDL. A ***linha de comando*** muda de IDL> para - .
3. Digite:
```
X = X * 2
PRINT, X
END
```
4. Esta instrução retorna o valor de X após a multiplicação, as instruções são compiladas e executadas pelo programa especial $MAIN$:
```
% Compiled module: $MAIN$.
        4
```
5. Digite .GO na ***linha de comando***. O Programa principal será executado novamente, retornando o novo valor do X:
```
8
```

Procedimentos e Funções

Procedimentos e funções contem módulos que dividem grandes tarefas em pequenas, mais manejáveis. Programas modulares simplificam a correção de erros e a manutenção, pois eles podem ser usados novamente, eles diminuem a quantidade de código requerida por novas aplicações.

Novos procedimentos e funções podem ser escritos no IDL e serem chamados da mesma maneira que os procedimentos ou funções definidas pelo sistema. Quando um procedimento ou uma função acaba, é executada uma instrução *RETURN*. Funções sempre retornam um resultado explicito. Um procedimento é chamado por uma

instrução de chamada de procedimento, enquanto a função é chamada por uma referência da função.

Procedimento

O procedimento já está contido no programa do IDL. Um procedimento começa com a instrução de declaração do procedimento, que consiste em uma palavra-chave reservada PRO, depois o nome do procedimento, e qualquer instrução para o mesmo. Um procedimento estará terminado com uma instrução de END.

Um exemplo de um procedimento é apresentado logo abaixo, no arquivo que chamamos de **procedimento.pro**, que foi criado da seguinte maneira:

1º Você deve abrir o IDLDE
2º File -> New -> Editor ou Ctrl+N
3º Digite o código abaixo

```
Pro procedimento
        Cd, current = c
        Print, c
End
```

4º Salve o arquivo com o nome que desejar (exemplo: procedimento)

Veja que este procedimento retorna o diretório padrão do IDL.
Compile o procedimento usando apenas o comando de execução .COMPILE (se ele estiver no diretório do IDL, caso contrário faça da mesma forma que foi feito no programa principal):

```
IDL> .compile procedimento.pro
```

Execute o procedimento chamando pelo nome:

```
IDL> procedimento
C:\RSI\IDL61
```

Se o arquivo que contem este programa não estiver no diretório do IDL, poderia ser executado especificando o caminho até o arquivo após o comando de execução .COMPILE . Por exemplo, **procedimento.pro** pode estar armazenado no diretório **C:\temp**, com isso fora do diretório do IDL, o comando de execução .COMPILE pode ser:

```
IDL> .compile  "C:\temp\procedimento.pro"
```

Função

Uma função outro programa que está contido no IDL; entretanto, uma função retorna informações. Uma função começa com a instrução de declaração da função, que consiste na palavra-chave reservada FUNCTION, o nome da função, e todos os seus parâmetros. O corpo da função, incluindo por ultimo a instrução *RETURN*. Uma função é terminada com a instrução END.

Um exemplo de uma função é apresentado logo abaixo, no arquivo que chamamos de **funcao.pro**, que foi criado para este exemplo (siga os passos que foram descritos no exemplo do procedimento, apenas substituindo o código).

```
Function funcao, f
        Return, f gt 1
End
```

Para compilar a função use o comando de execução .COMPILE

```
IDL> .compile  funcao
```

A sintaxe para chamar esta função é:

```
IDL> f = funcao (10)
IDL> print, f
        1
```

Da mesma forma que no procedimento, se o arquivo contendo a função não estiver no diretório do IDL, ele poderá ser compilado especificando todo o caminho até o arquivo após o comando de execução .COMPILE.

Arquivos de Lote

Um arquivo de lote contém uma ou mais instruções ou comandos do IDL. Cada linha do arquivo de lote é lida e executada antes de prosseguir para à linha seguinte. Os arquivos de lote não podem ser considerados exatamente um programa, pois diferente dos programas estes não são compilados apenas executados.

Um exemplo de um arquivo de lote é apresentado logo abaixo, no arquivo que chamamos de **lote.pro**, que foi criado para este exemplo (siga os passos que foram descritos no exemplo do procedimento, apenas substituindo o código).

```
print, systime()
procedimento
```

Veja que o exemplo acima alem de retornar a Data/Hora do sistema, também executa o procedimento visto no exemplo acima.

Para executar um arquivo de lote:

```
IDL> @lote
Wed Jun 22 16:25:40 2005
C:\RSI\IDL61
```

Se o arquivo **lote.pro** não estiver diretório do IDL, o caminho completo do arquivo deve ser especificado antes do símbolo @.

Parâmetros de posição e de palavra-chave

Parâmetros são usados para passar informações entre os programas. No IDL temos dois tipos de parâmetros: de posição e por palavra-chave. Embora os dois tipos podem ser usados para enviar ou receber informações, parâmetros de posições são tipicamente usado para requerer informações, enquanto palavras-chave são usadas para informações opcionais. Como os parâmetros são empregados, embora, seja uma escolha do programador.

A ordem dos parâmetros de posição em uma chamada à um programa importante. Por exemplo, se *x* e *y* são vetores, então:

```
IDL> x = findgen (20)
IDL>  y = x ^ 2
```

Então esta instrução *PLOT*

```
IDL> plot, x, y
```

É diferente desta outra

```
IDL> plot, y, x
```

Parâmetros de palavra-chave, por outro lado, podem ser listados em qualquer ordem. Por exemplo, as seguintes instruções de um procedimento tem o mesmo resultado:

```
IDL> plot, x, y, xtitle = 'Tempo', ytitle = 'Velocidade'
IDL> plot, x, y, ytitle = 'Velocidade', xtitle = 'Tempo'
```

Parâmetros de palavras-chave, podem ser abreviado. Isto é útil ao usar IDL interativamente pela ***Linha de Comando***, mas não é recomendado que se faça isto no código de um programa, pode confundir o próprio programador.
Palavras-chave boleanas podem ser determinadas com uma " / " antes da palavra-chave. Por exemplo, as seguintes instruções de um procedimento tem o mesmo resultado:

```
IDL> plot, x, y, /nodata
IDL> plot, x, y, nodata = 1
```

Passando por Parâmetros

Quase todos os programas em IDL usam parâmetros. Esta sessão descreve os dois mecanismos distintos para passagem de parâmetro.

Examine a diferença entre os argumentos usados nestas duas chamadas para o procedimento *PLOT*:

```
IDL> x = findgen (10)
IDL> plot, x                ; Primeira chamada
IDL> plot, findgen (10)     ; Segunda chamada
```

Ambas chamadas tem os mesmos dados e o mesmo resultado, uma linha retilínea indo de 0 até 9. A diferença encontra-se na maneira que os dados são passados para o *PLOT*: uma variável é passada na primeira chamada, uma expressão (o que retornar da chamada da função) na segunda chamada.

No IDL existe dois mecanismos de passagem de parâmetros: passagem por valor e passagem por referência. Passagem por valor significa que cada parâmetro pega uma copia do valor dos argumentos, de modo que as mudanças ao parâmetro não afetem os argumentos, mesmo se as variáveis tiverem o mesmo nome, e as mudanças serão perdidas quando retornar para o programa.

Com a passagem por referência, um argumento não será copiado para os parâmetros. Qualquer modificação do parâmetro no programa muda o argumento pois os dois referenciam-se a mesma memória.

Muitas rotinas do IDL utilizam parâmetros para passar informações de volta para quem as chamou. Pegue, como exemplo, esta chamada a função *WHERE*:

```
IDL> x = 0
IDL> index = where(x lt 5, count)
IDL> print, index, count
        0
      1
```

O segundo argumento no *WHERE* (count) é usado para retornar o número de vezes que a expressão condicional no primeiro argumento é compatível. Este argumento precisa ser passado por referência, ou as informações não serão retornadas.

As regras para determinar como um parâmetro deve ser passado no IDL são:

- Por valor: Expressões, incluindo sobrescrever os elementos de uma matriz ou vetor, campos de uma estrutura, variáveis de sistema e constantes.
- Por referência: Variáveis nomeadas, estruturas completas.

Atenção: A passagem de parâmetros de uma estrutura pode ser feita por referência apenas quando vamos passar toda a estrutura, caso a intenção seja a passagem de apenas algum campo desta estrutura será passado por valor.

O mecanismo de passagem por referência torna possível a informação de uma variável ser modificada quando ela é passada para o programa.

Chamada de Programas

O mecanismo de chamada de programas do IDL é uma seqüência de etapas que localizam, solvem e executam um procedimento ou função, quando o procedimento ou função é chamado por nome, da ***linha de comando*** ou dentro de um programa do IDL. Por exemplo, quando chamamos o procedimento *LOADCT* pela ***linha de comando***:

```
IDL> loadct, 5
% Compiled module: LOADCT.
% Compiled module: FILEPATH.
% Compiled module: PATH_SEP.
% LOADCT: Loading table STD GAMMA-II
```

O mecanismo chamado é usado para localizar, solver e executar *LOADCT*, carregando a tabela de cor número 5 na sessão do IDL.

O mecanismo de chamada consiste em quatro etapas.

1. Procura por uma rotina nomeada na tabela de rotinas do sistema do IDL. Se a rotina estiver listada na tabela do sistema, executa ela; senão, processa a próxima etapa.

2. Verifica se a rotina se encontra em um estado compilado na sessão atual do IDL. Isto pode ser diagnosticado usando o *HELP* com a palavra chave ROUTINE. Se o programa já estiver compilado, então roda ele. Se ele não estiver vai para a próxima etapa.

3. Procura por um arquivo ou um arquivo de sistema que tenha o mesmo nome base que a rotina chamada, com a extensão **.pro** ou **.sav**. A procura começa no diretório atual do IDL, então ele prossegue através dos diretórios listados na variável de sistema *!path*. Se o arquivo for encontrado, o IDL compila qualquer unidade de programa encontrada no arquivo, começando pelo topo, até que a rotina chamada seja encontrada. A rotina chamada é então compilada e executada. Se um arquivo com o mesmo nome da rotina chamada não for encontrado, ou se um arquivo com o mesmo for encontrado, porém ele não contiver a rotina chamada, então iremos para última etapa.

4. Emite uma mensagem de erro, informando que a função ou procedimento solicitado não pode ser achado.

No exemplo anterior foi chamado o procedimento *LOADCT*, na terceira etapa do mecanismo de chamada. *LOADCT* não é uma rotina do sistema, por isso não foi compilado mais cedo. Entretanto, o arquivo **loadct.pro** existe no subdiretório **lib**, do qual é parte do diretório de busca padrão do IDL. O IDL abre o arquivo, compila o procedimento *LOADCT* interiormente e executa ele.

Note que as rotinas *FILEPATH* e *PATH_SEP* foram executados também na chamada do *LOADCT*.

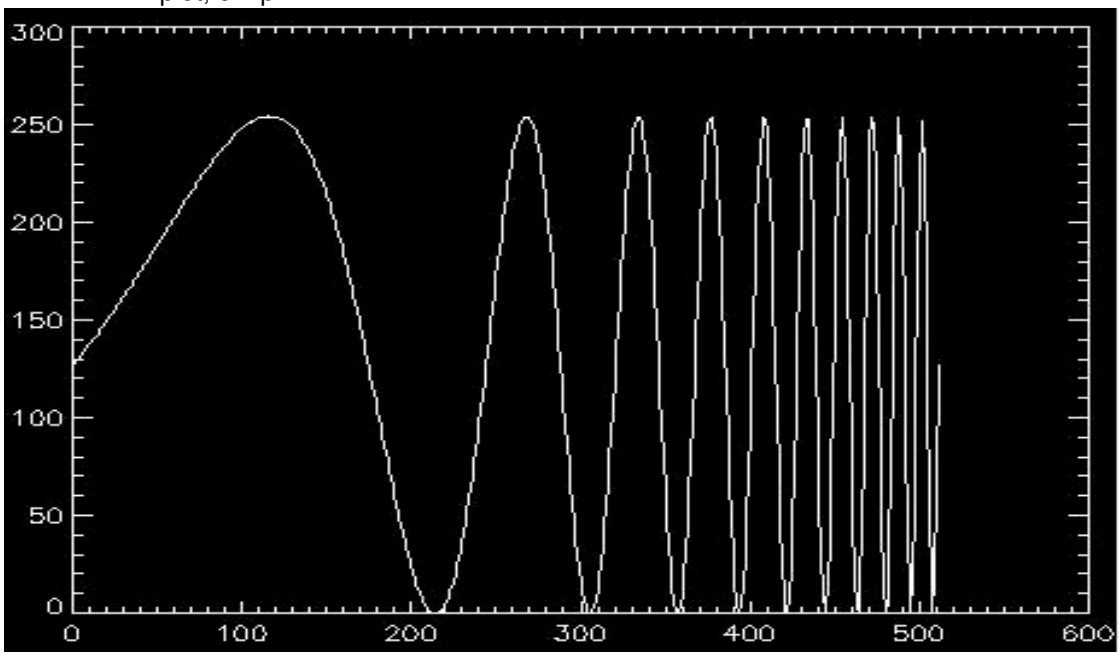
Gráfico em Linha

Visualização das tabelas de dados do IDL. Uma maneira de visualizar a seqüência de valores dos dados, como uma linha do tempo, para exibi-los como gráfico em linha use o procedimento *PLOT*.
Neste exemplo, nós iremos ler dados para uma nova variável *chirp* e exibi-los na tela como gráfico em linha.

```
IDL> filename = filepath('chirp.dat', subdir=['examples','data'])
IDL> chirp = read_binary(filename)
IDL> help, chirp
CHIRP    BYTE      = Array [512]
```

O valor dos dados foram lidos para um arquivo no IDL na forma de uma matriz de uma dimensão (também conhecido como vetor). Note que a matriz é do tipo byte.
O que há na variável *chirp* ? Com o procedimento *PRINT*, nós podemos visualizar os dados em forma de tabela, mas considerando que são 512 valores, pode ser mais fácil de compreender os dados visualizando graficamente. Exiba os dados na tela usando o procedimento *PLOT*.

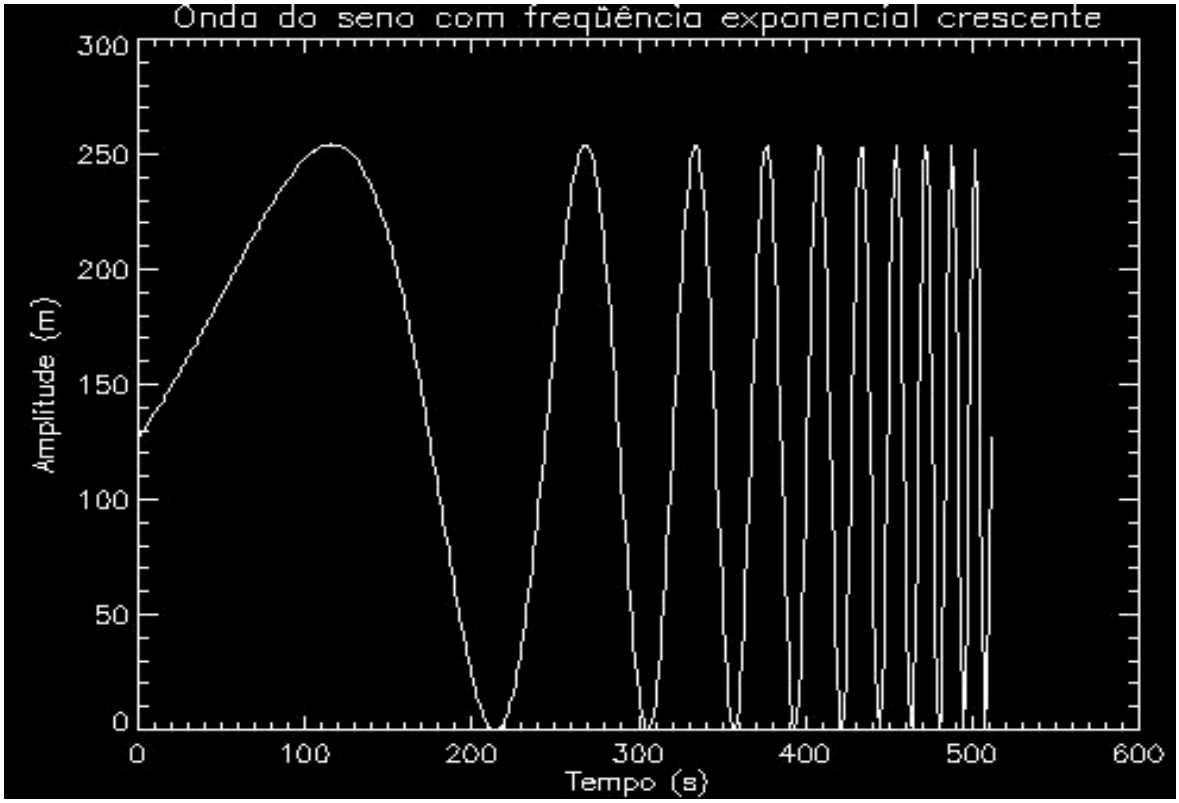
```
IDL> plot, chirp
```



Este é apenas um exemplo básico da funcionalidade do procedimento *PLOT* podemos explorar mais as suas possibilidades utilizando as palavras-chaves.
Podemos por exemplo descrever os títulos:

```
IDL> plot, chirp, xtitle = 'Tempo (s)', ytitle = 'Amplitude (m)', $
```

IDL> title = ‘Onda do seno com freqüência exponencial crescente’



Você poderá encontrar maiores especificações destas e de outras palavras-chave utilizando a ferramenta IDL Online Help.

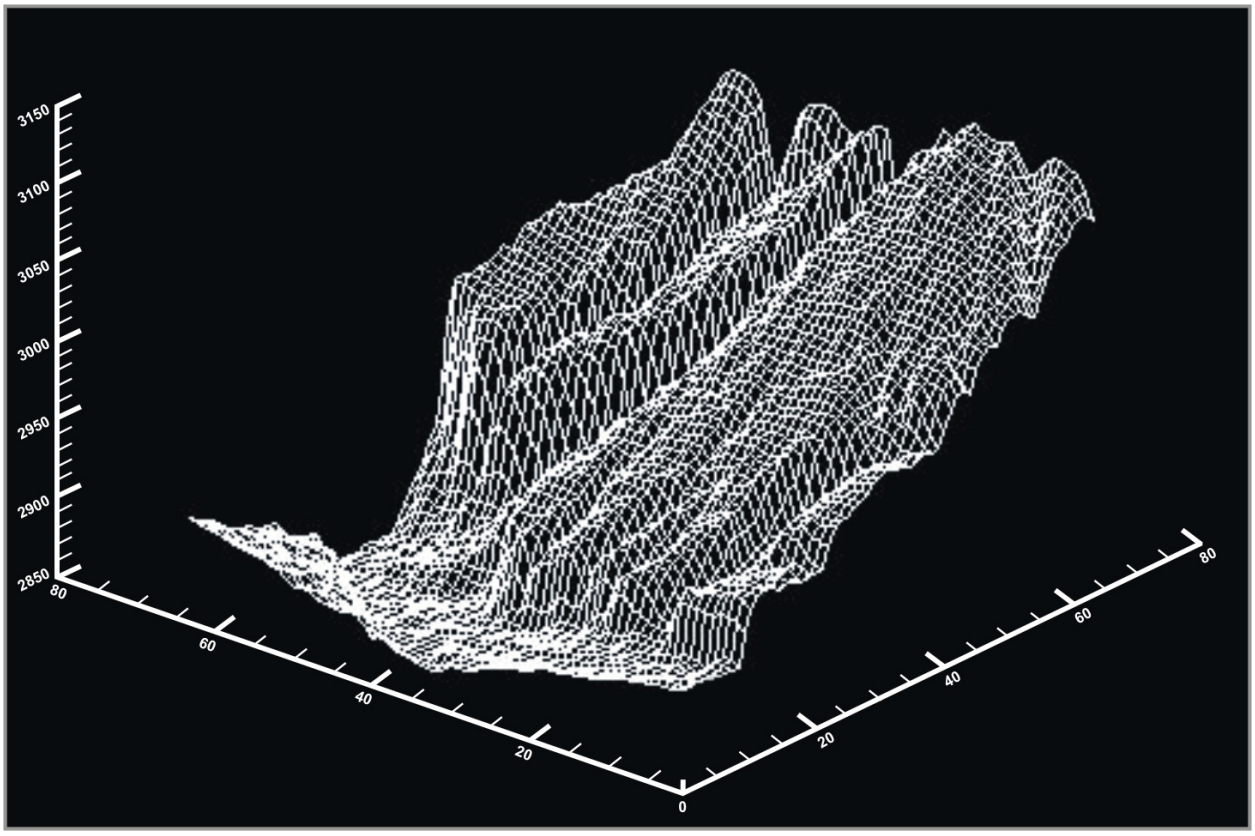
Gráfico de Superfície

Para exemplo de um gráfico de superfície usaremos o arquivo **lvdem.sav** (que pode ser encontrado no servidor de ftp da RSI (ftp://ftp.rsinc.com/pub/training/IDL_intro/), neste servidor entre no arquivo zipado intro.zip, entre na pasta IDL_training, e depois na pasta introduction, pegue o arquivo lvdem.sav e coloque ele dentro da pasta IDLXX, que esta dentro do diretório da RSI), como este arquivo é do tipo *.sav*, então ele precisa primeiro ser restaurado, dentro deste arquivo encontraremos a variável *lvdemdata* os dados contidos nesta variável representam valores de elevação obtidos pela USGS do Canyon Big Thompson em Colorado nos Estados Unidos.

```
IDL> restore, 'lvdem.sav'
IDL> help, lvedemdata
LVDEMDATA      INT       = Array [64, 64]
```

Esta variável tem duas dimensões, representando uma matriz de elementos de 64 x 64. Podemos ver os dados armazenados em uma matriz bidimensional como esta usando o procedimento *SURFACE*. Para fazer a rotação da *SURFACE* devemos utilizar as palavras-chave *AX* e *AZ*.

```
IDL> surface, lvdemdata, az=50, ax=40
```



O procedimento *SURFACE* retorna um emaranhado de fios representando os dados. Outra forma de visualizar esses dados seria usando o procedimento *SHADE_SURF* que apresenta na tela a mesma imagem porém de forma preenchida.

IDL> shade_surf, lvedemdata, az=50, ax=40

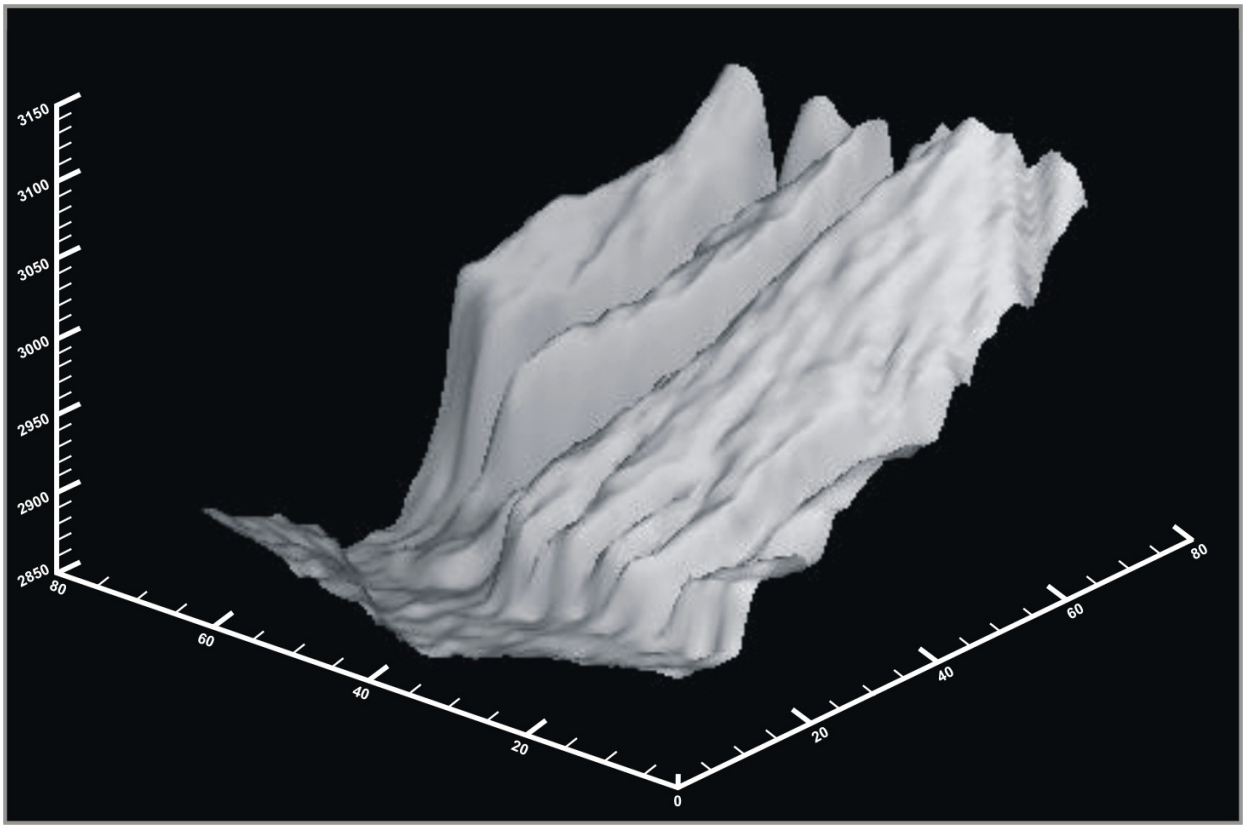
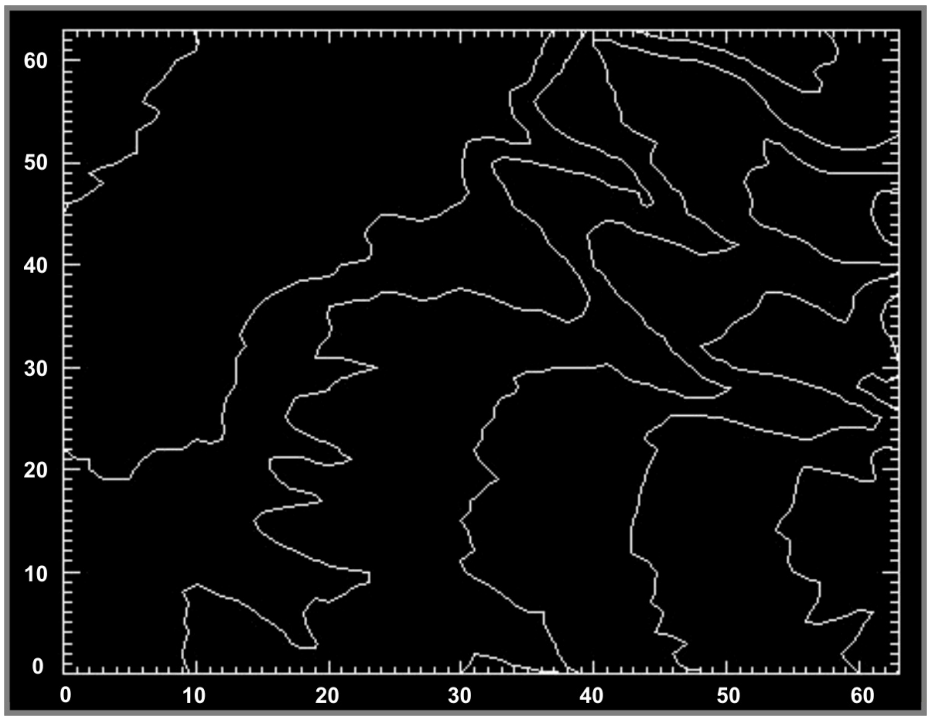


Gráfico de Contorno

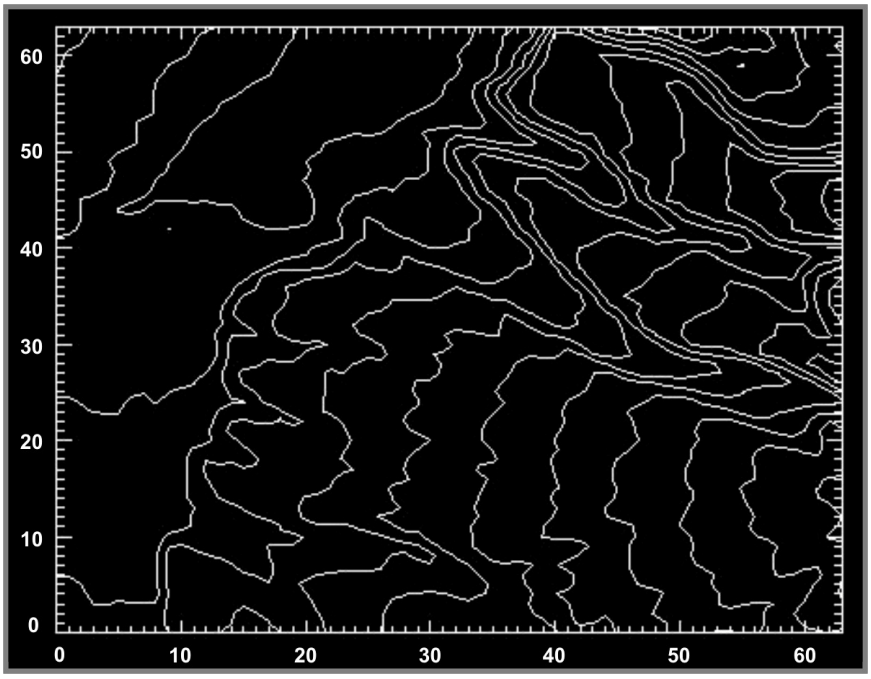
Mais uma vez como exemplo usarei a variável *lvdemdata* os dados desta variável podem ser mostrados na tela como um gráfico de contorno do IDL, utilizando o procedimento *CONTOUR* e as palavras-chave XSTYLE e YSTYLE para ajustar a localização no gráfico de contorno.

IDL> contour, lvdemdata, xstyle=1, ystyle=1



Por padrão, o procedimento *CONTOUR* tem 05 níveis de contornos e fica fora de ajuste por isso o uso das palavras-chave XSTYLE e YSTYLE, para personalizar os níveis de contorno do gráfico de contornos podemos usar a palavra-chave NLEVELS.

IDL> contour, lvdemdata, xstyle=1, ystyle=1, nlevels=12



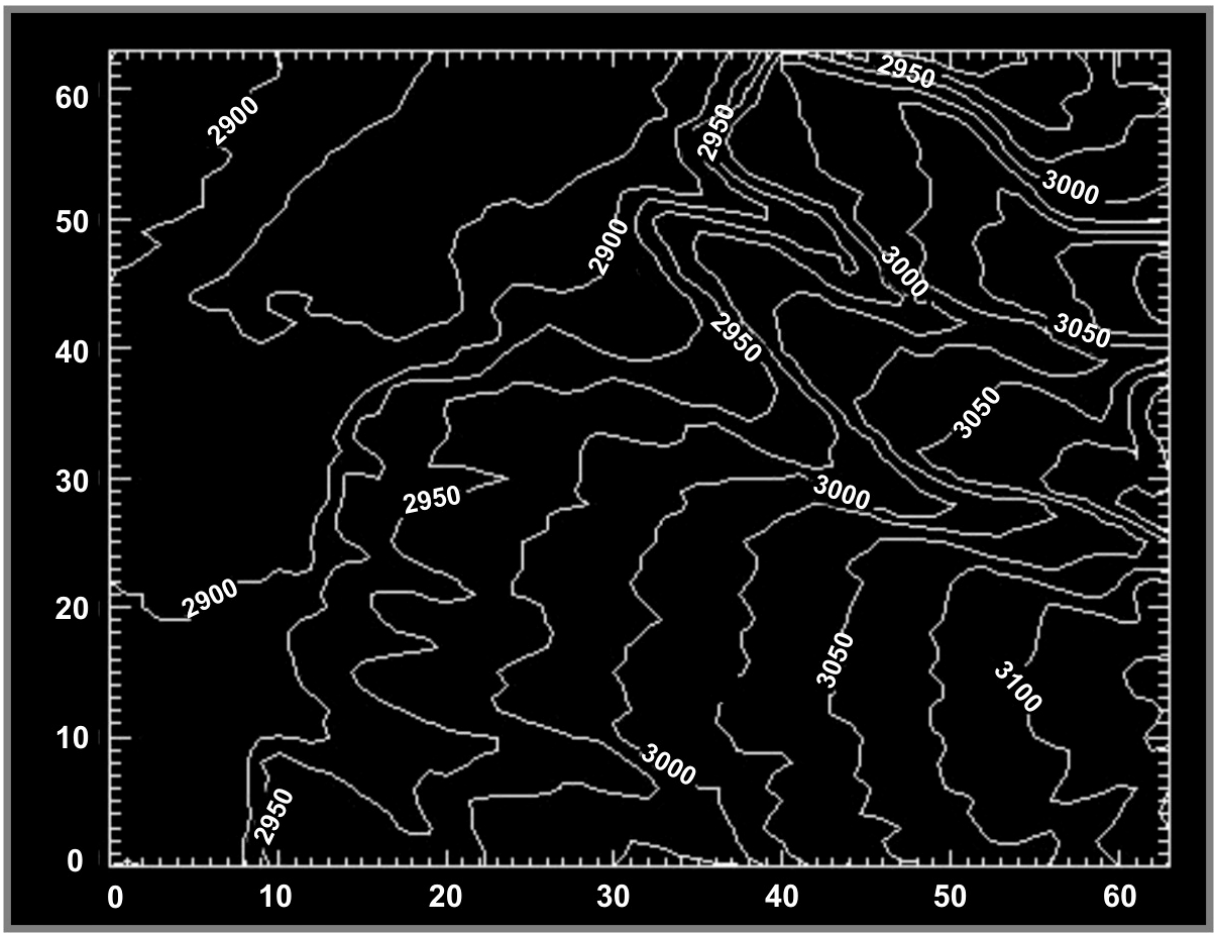
Permitir que o IDL escolha níveis de contorno é útil para conseguir uma idéia preliminar da série de dados, mas uma técnica melhor seria atribuir o nível de contorno, tendo o conhecimento das escalas dos dados em cada dimensão. Comece determinando o valor mínimo e o máximo dos dados com as funções *MIN* e *MAX*.

```
IDL> print, min(lvdemdata), max(lvdemdata)
        2853            3133
```

Com estes valores, defina o nível de contorno e aplique ele no gráfico através da palavra-chave LEVELS. Use a palavra-chave FOLLOW para mostrar na tela os valoras do contorno.

```
IDL> clevels = indgen(13) * 25 + 2850
IDL> print, clevels
2850 2875 2900 2925 2950 2975 3000 3025 3050 3075 3100 3125 3150

IDL> contour, lvdemdata, xstyle=1, ystyle=1, levels= clevels, /follow
```



Exposição e processamento de imagens

IDL pode ser usado para apresentar na tela matrizes como imagens. Os valores dos dados de uma matriz são transformados em uma escala cinza intensa ou colorida. Carregue a paleta com escala cinza e apresente a mesma imagem usando o procedimento *TVSCL*.

```
IDL> loadct, 0
IDL> tvscl, lvdemdata
```

A imagem resultante é realmente pequena. Isto é porque os elementos contidos na matriz *lvdemdata* são transformados para os pixels da tela do seu computador. A matriz é de 64 elementos em um lado, então a imagem resultante será de 64 pixels. Você pode utilizar o IDL para determinar a resolução da tela do seu computador com a função *GET_SCREEN_SIZE*:

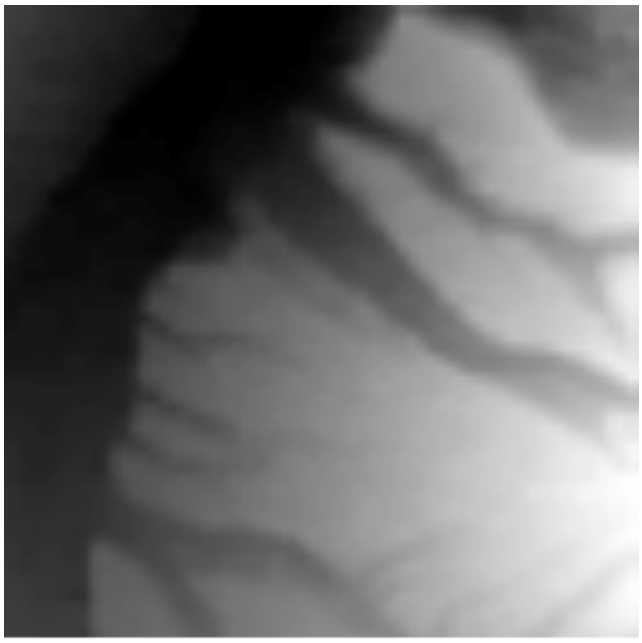
```
IDL> print, get_screen_size
      1280.00  1024.00
```

Redimensione *lvdemdata* interpolando ele para uma matriz de 256 x 256 elementos usando a função *REBIN*. Armazene o resultado em uma variável *dem*. Exiba o resultado na tela.

```
IDL> x = 256
IDL> y = 256
IDL> dem = rebin (lvdemdata, x, y)
IDL> tvscl, dem
```

Perceba que apesar da imagem estar maior você ainda fica com a janela maior do que a imagem porém podemos ajustar o tamanho da janela:

```
IDL> window, 0, xsize = x, ysize = y
IDL> tvscl, dem
```



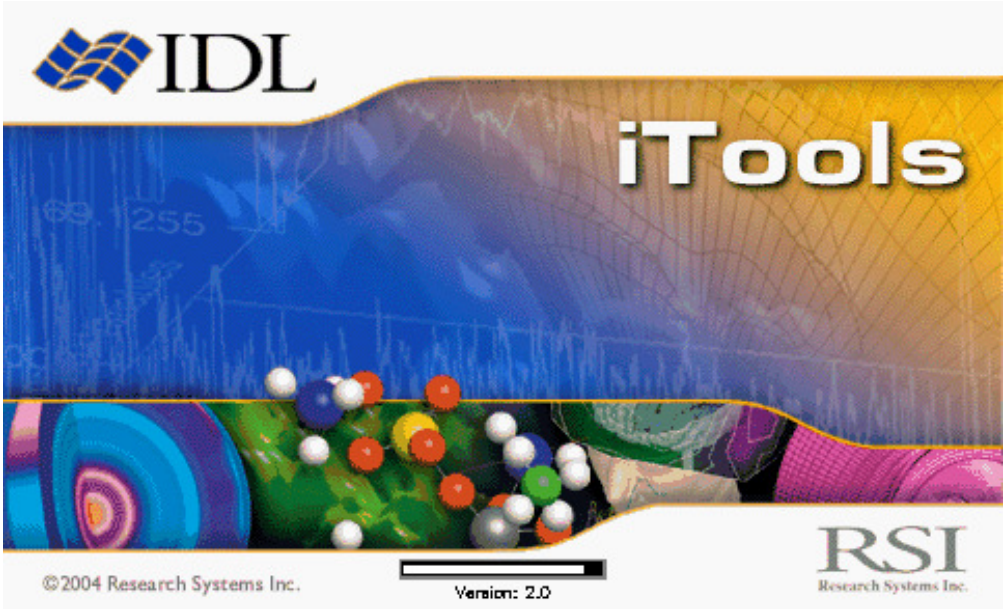
Os dados usados para exibir esta imagem são apenas números, o IDL pode confundir eles. Tente diferenciar a imagem usando a função *SOBEL*. A função *SOBEL* utiliza um destaque no contorno.

```
IDL> tvscl, sobel(dem)
```

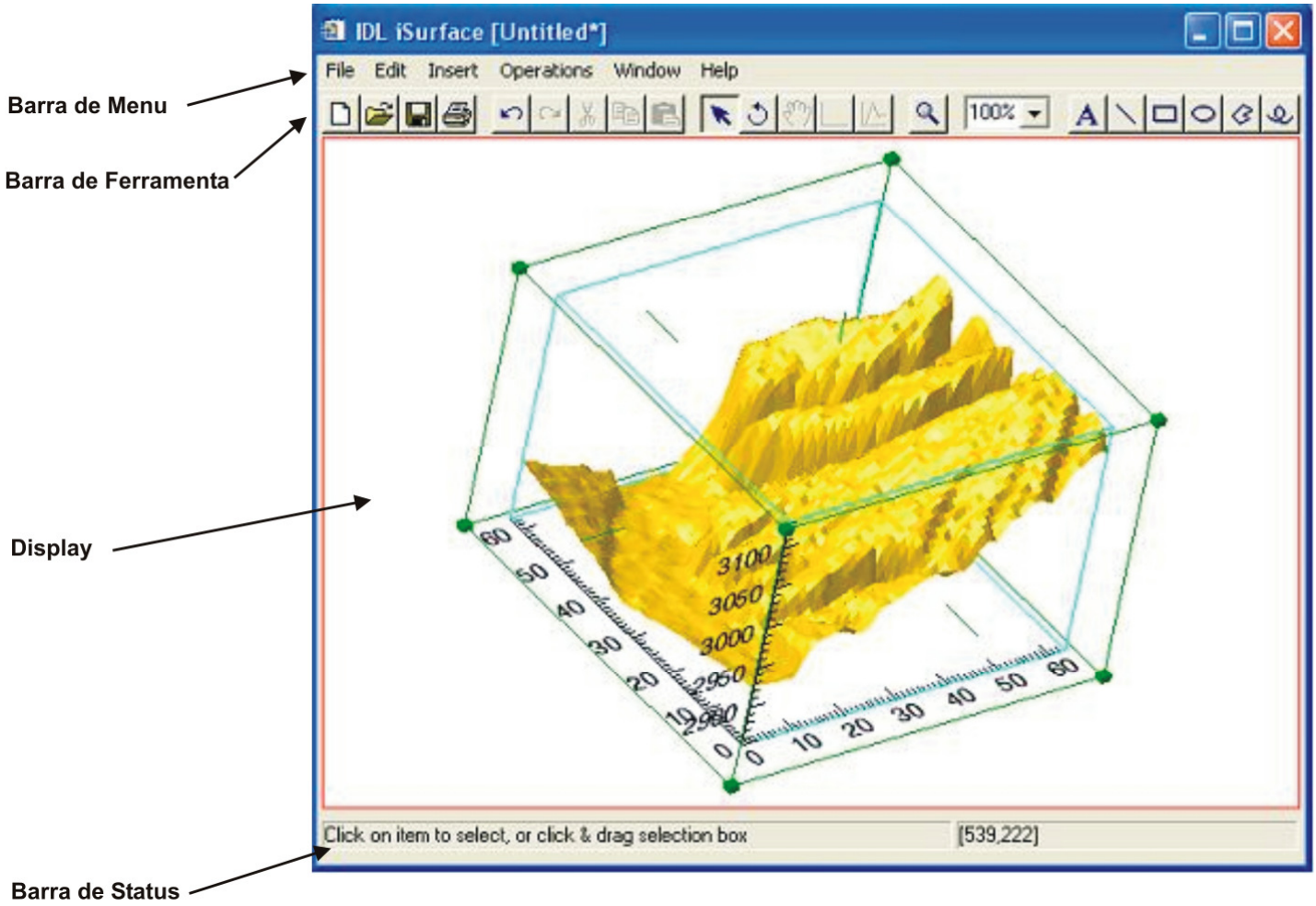
Ferramentas Inteligentes do IDL (iTools)

As ferramentas inteligentes do IDL (iTools), foram introduzidas a partir da versão 6.0, fornecendo um novo significado à visualização interativa dos dados.
Olhe a mesma imagem utilizada nos exemplos anteriores com a ferramenta iSurface.

IDL> isurface, lvdemdata



Após você digitar o comando iSurface na ***Linha de Comando*** aparecerá uma imagem como a imagem acima, que significa que o IDL iTools está carregando. Depois dele terminar de carregar esta imagem irá sumir aparecendo a imagem do Big Thompson Canyon que será exibida de uma forma semelhante a função *SHADE_SURF*. Você poderá movimentar a imagem, girá-la, mudar a sua cor, adicionar algumas anotações, gerar um gráfico de linha de uma determinada parte da sua escolha, você terá muitas possibilidades, cabe você explora-las.



Há ainda diversas outras ferramentas do iTools similares com os procedimentos utilizados nos exemplos anteriores, como iPlot, iContour, iImage, iMap, iVolume. A figura acima mostra a imagem com o iSurface, ela tem indicadores apresentando alguns componentes do iTools que serão descritos logo abaixo.

A Barra de Menu é utilizada para controlar diversos aspectos do iTool, incluindo importar ou exportar dados, inserir uma nova visualização ou um novo gráfico de elementos para uma visualização existente, executar operações nos dados indicados pelo iTools e capturar ou ajustar as propriedades de visualização que são usadas no iTools.

A Barra de Ferramentas apresenta botões com a mesma funcionalidade que alguns dos itens da Barra de Menu. Se movermos o mouse sobre os ícones teremos uma descrição das suas funcionalidades.

Os dados serão visualizados no Display. O teclado e o mouse serão importantes na interação dos dados visualizados.

A barra de status mostra informação sobre o estado atual do iTool, o mais interessante é sobre o que está ocorrendo no momento com o iTools. O painel direito da Barra de Status mostra a localização do cursor, se ele estiver sobre a visualização, apresentará os dados das coordenadas de forma tridimensional, por outro lado se estiver fora da visualização, apresentará os dados das coordenadas de forma bidimensional.

Quando você terminar de utilizar o iTools utilize o procedimento *ITRESET*, para encerrar o seu uso.

```
IDL> itreset
```

Introdução para Variáveis

Nomes Validos

Os nomes das variáveis precisam começar com uma letra ou um sobrescrito e podem ter até 128 letras, dígitos, sobrescritos, ou sinais de dólar. O tamanho das variáveis (irá depender do tipo) é limitado pelo computador e o sistema operacional que você está utilizando, e não pelo IDL. Os nomes de variáveis são únicos. O IDL converte todos os caracteres alfabéticos para maiúsculo internamente.

Tipos de Variáveis

Existem 16 tipos de variáveis no IDL, sete inteiras, duas flutuantes, duas complexas, uma do tipo texto, dois do tipo abstrato (objetos e ponteiros), e indefinido. As estruturas são consideradas seu próprio tipo, enquanto as matrizes são do mesmo tipo dos seus elementos.

Tipo	Tamanho (bits)	Criação por Escala	Função de Conversão
byte	8	a = 5B	*BYTE*
integer	16	b = 0S ; b = 0	*FIX*
unsigned integer	16	c = 0U	*UINT*
long integer	32	d = 0L	*LONG*
unsigned long integer	32	e = 0UL	*ULONG*
64-bit integer	64	f = 0LL	*LONG64*
unsigned 64-bit integer	64	g = 0ULL	*ULONG64*
float	32	h = 0.0	*FLOAT*
double	64	i = 0.0D	*DOUBLE*
complex	64	j = complex(1.0, 0.0)	*COMPLEX*
double complex	128	k = dcomplex(1.0, 0.0)	*DCOMPLEX*
string		l = 'hello' l = "hello"	*STRING/ STRTRIM*
pointer	32	m = ptr_new()	
object	32	n = obj_new()	
undefined structure	8		

Criação de Variáveis

No IDL as variáveis não precisam ser declaradas. O tipo das variáveis é determinado após o uso. Antes da criação, uma variável é do tipo *undefined*; não é permitido você utilizar uma variável *undefined* em uma expressão.

```
IDL> help, var
VAR             UNDEFINED = <Undefined>
IDL> var  = 0L
IDL> help, var
VAR             LONG = 0
```

Existe três formas de criar uma variável de forma que não seja *undefined*, são elas: Escalar, Matricial (ou Vetorial) e por Estruturas.

Escalar

As variáveis escalares são inicializadas de forma que a variável seja de valor escalar (ou seja apenas um valor). Use a idéia passada na tabela acima na coluna "Criação por Escala" para criar um tipo de variável específica. Por exemplo:

```
IDL> image_pixel = 0B                          ; byte
IDL> loopcounter = 0U                          ; uint
IDL> n_elms = 0L                               ; long
IDL> fileoffset = 0ULL                         ; ulong64
IDL> variance = 0.0                            ; float
IDL> total = 0.0D                              ; double
IDL> complex_var = complex(0.0 , 0.0)          ; complex
IDL> name = 'Hello World'                      ; string
IDL> new_image_pixel = image_pixel             ; byte
IDL> pixel_value = image_pixel + 0.0           ; byte + float  =  float
```

O nome da variável do lado esquerdo do sinal de igual absorverá o tipo do valor, variável, ou expressão do lado direito do sinal.

Matricial e Vetorial

O IDL é uma linguagem baseada em matrizes sejam elas unidimensionais (vetores) ou bidimensionais.

Criação

Tipo	Função de Criação	Índice que gera a Função
Byte	*BYTARR*	BINDGEN
Integer	*INTARR*	INDGEN
Unsigned integer	*UINTARR*	UINDGEN
Long integer	*LONARR*	LINDGEN
Unsigned long integer	*ULONARR*	ULINDGEN
64-bit integer	*LONG64ARR*	L64INDGEN
Unsigned 64-bit integer	*ULONG64ARR*	UL64INDGEN
Float	*FLTARR*	FINDGEN
Double	*DBLARR*	DINDGEN
Complex	*COMPLEXARR*	CINDGEN
Double complex	*DCOMPLEXARR*	DCINDGEN
String	*STRARR*	SINDGEN
Pointer	*PTRARR*	
Object	*OBJARR*	
Undefined		
Structure	*REPLICATE*	

Vetores e matrizes com diferentes tipos de variáveis e dados podem ser iniciados com rotinas do IDL como as listadas na tabela acima.

```
IDL> vec = fltarr (15)
IDL> vec[5] = 4.56
IDL> vec[13]= 1234.333
```

Matrizes e vetores no IDL começam com 0; consequentemente o vetor *vec* está subscrito de 0 até 14.

Vetores e matrizes também podem ser criados atribuindo à variável certo número de valores passados por colchetes.

```
IDL> temp = [12, 82, 97, 23, 0, 78]
```

Esta instrução cria um vetor de seis inteiros. Uma matriz de múltipla dimensão pode ser criada colocando valores dentro de vários colchetes dentro de dois colchetes, observe:

```
IDL> id4x4 = [ [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1] ]
IDL> help, id4x4
ID4x4     INT        = Array [4, 4]
```

A variável *id4x4* tornou-se uma matriz 4 X 4 de inteiros. Considere a instrução abaixo:

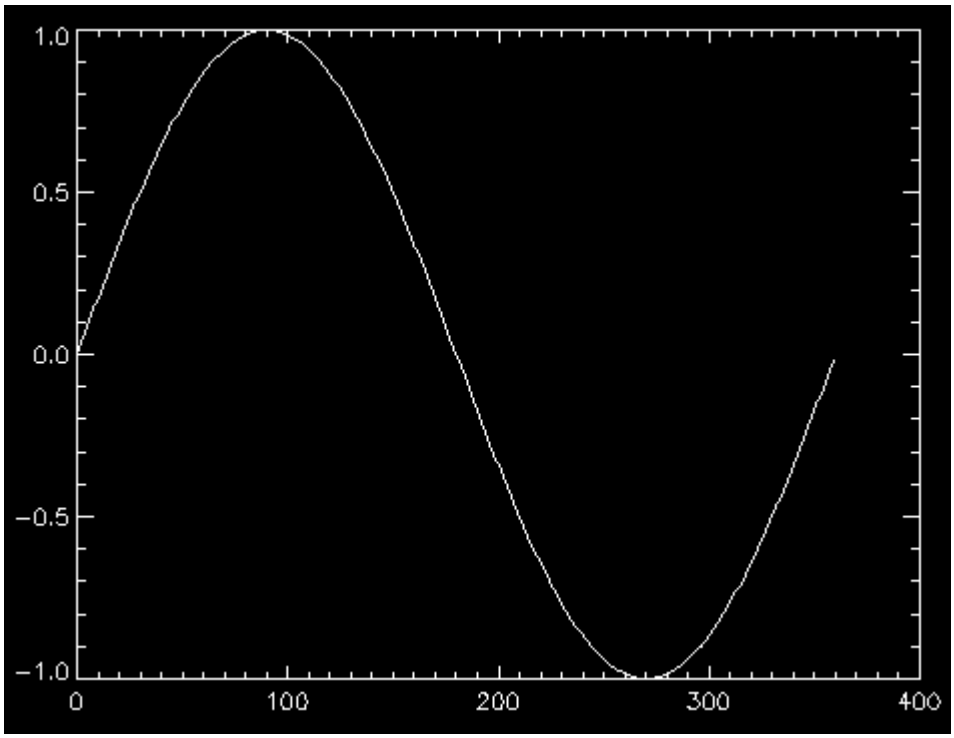
IDL> exemplo_arr = [3.4, 6.7D, 45U, 90L]

Esta instrução cria um vetor com diferentes tipos de variável. O IDL não apresenta problemas diante disso, ele executa uma conversão pelo tipo mais significativo. O tipo mais significativo nesta instrução é o *double* (6.7D). Todos os elementos do vetor são convertidos para *double*.

Operação de matrizes e vetores

Quando executamos uma operação com matrizes ou vetores no IDL, eles são executados em todos os elementos que estão dentro desta matriz ou vetor. Esta poderosa característica elimina a necessidade da utilização do laço para executar uma operação com cada um dos elementos.

IDL> x = findgen (360) * !dtor
IDL> sincurve = sin(x)
IDL> plot, sincurve



A primeira instrução multiplica os números 0.0, 1.0, 2.0, ...359.0 pela variável de sistemas *!dtor* fazendo a transformação de grau para radiano constante, então a segunda instrução calcula o seno de todos os elementos resultados pela transformação de graus em radianos, e a terceira linha de comando como já vimos anteriormente nesse guia exibe, um gráfico em linha do calculo final da matriz. Isto beneficia o IDL, eliminando o tempo consumido para controlar os laços, que executam as operações em todos os elementos do vetor. Use a vantagem desta possibilidade o máximo que poder - é mais rápido e mais fácil de ler.

Matrizes Multidimensionais

As matrizes no IDL podem ter até oito dimensões. No caso da matriz bidimensional do IDL a subscrição é especificada normalmente como [coluna, linha].

```
IDL> multi = lindgen (4, 6)        ; quatro colunas, seis linhas
IDL> print, multi

      0       1       2       3
      4       5       6       7
      8       9       10      11
      12      13      14      15
      16      17      18      19
      20      21      22      23
```

O método de subscrição do IDL é por coluna. Internamente as matrizes são dispostas no formato de linha (os elementos do *multi* são numerados na ordem em que eles são armazenados na memória). A subscrição em uma matriz bidimensional pode ser feita utilizando a notação [coluna, linha].

```
IDL> print, multi [1, 0], multi [3, 5]
1       23
```

Todas as escalas de operadores podem ser utilizadas nas matrizes multidimensionais.

```
IDL> print, multi [*, 4]    ; a quinta linha
16      17      18      19

IDL> print, multi [2, *]    ; a terceira coluna

      2
      6
      10
      14
      18
      22

IDL> print, multi [2: 3, 1: 2]    ; uma quadra – a 3º e a 4º coluna e a 2º e 3º linha
      6       7
      10      11
```

Multiplicação de Matrizes

O operador # calcula o produto de uma matriz, multiplicando as colunas da primeira matriz pelas linhas da segunda. A segunda matriz deve ter o mesmo número de colunas que a primeira tem de linhas. A matriz resultante terá o número de colunas da primeira matriz e o mesmo número de linhas da segunda matriz.

Por exemplo:

```
IDL> a = indgen(3,2)            ; matriz 3 X 2, na sintaxe do IDL.
IDL> print, a
      0    1    2
      3    4    5
IDL> b = indgen(2,3)            ; matriz 2 X 3.
IDL> print, b
```

```
        0     1
        2     3
        4     5
IDL> print, a # b
        3        4        5
        9        14       19
        15       24       33
```

Os cálculos são feitos desta forma:

A 0,0 . B 0,0 + A 0,1 . B 1,0	A 1,0 . B 0,0 + A 1,1 . B 1,0	A 2,0 . B 0,0 + A 2,1 . B 1,0
A 0,0 . B 0,1 + A 0,1 . B 1,1	A 1,0 . B 0,1 + A 1,1 . B 1,1	A 2,0 . B 0,1 + A 2,1 . B 1,1
A 0,0 . B 0,2 + A 0,1 . B 1,2	A 1,0 . B 0,2 + A 1,1 . B 1,2	A 2,0 . B 0,2 + A 2,1 . B 1,2

Usando os valores atuais:

(0).(0) + (3).(1)	(1).(0) + (4).(1)	(2).(0) + (5).(1)
(0).(2) + (3).(3)	(1).(2) + (4).(3)	(2).(2) + (5).(3)
(0).(4) + (3).(5)	(1).(4) + (4).(5)	(2).(4) + (5).(5)

O operador ## calcula o produto de uma matriz, multiplicando as linhas da primeira matriz pelas colunas da segunda. A segunda matriz deve ter o mesmo número de linhas que a primeira tem de colunas. A matriz resultante terá o número de linhas da primeira matriz e o mesmo número de colunas da segunda matriz.

Por exemplo:

```
IDL> print, a ## b
        10       13
        28       40
```

Os cálculos são feitos desta forma:

A 0,0 . B 0,0 + A 1,0 . B 0,1 + A 2,0 . B 0,2	A 0,0 . B 1,0 + A 1,0 . B 1,1 + A 2,0 . B 1,2
A 0,1 . B 0,0 + A 1,1 . B 0,1 + A 2,1 . B 0,2	A 0,1 . B 1,0 + A 1,1 . B 1,1 + A 2,1 . B 1,2

Usando os valores atuais:

(0).(0) + (1).(2) + (2).(4)	(0).(1) + (1).(3) + (2).(5)
(3).(0) + (4).(2) + (5).(4)	(3).(1) + (4).(3) + (5).(5)

A função *WHERE*

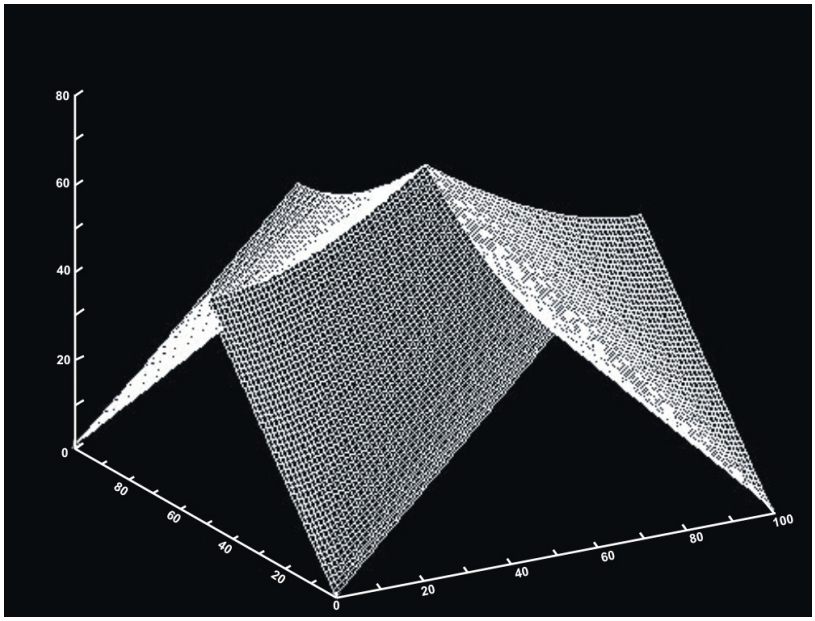
A função *WHERE* avalia uma expressão e retorna um índice unidimensional de cada elemento em um vetor para todas as expressões verdadeiras (ele retorna a posição, não os valores dos dados). Se a expressão for falsa, o valor de retorno é −1. Por exemplo, iremos alterar todos os elementos entre 0.4 e 0.5 de uma matriz para 1.0:

```
IDL> x = findgen (360) *!dtor
IDL> data = sin(x)
IDL> indices = where ( data gt 0.4 and data lt 0.5, count)
IDL> print, count
12
IDL> data [indices] = 1.0
```

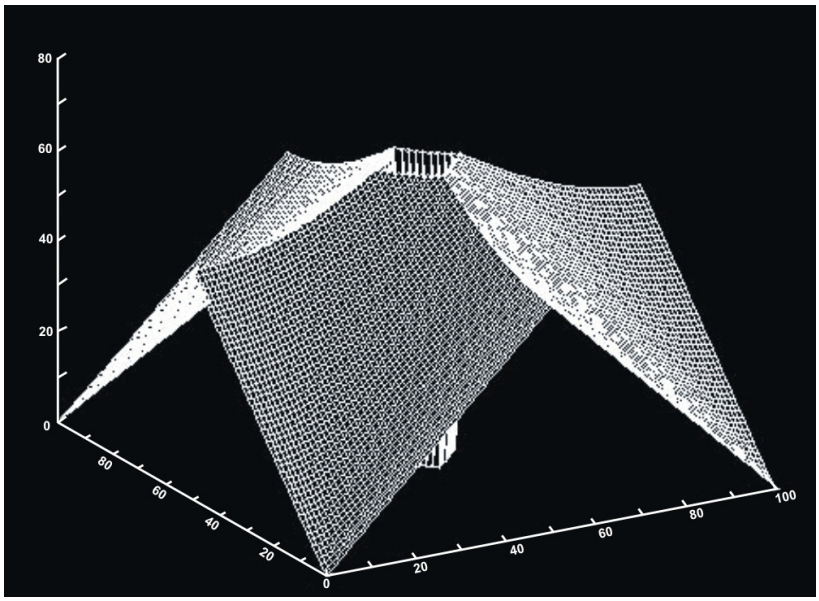
Observe que na terceira linha, onde se encontra a função *WHERE* há um argumento chamado count, este argumento cria uma variável com o nome que você decidir (no exemplo ela é chamada de *count*), esta variável irá retornar o número total de elementos encontradas pela função.

Uma matriz multidimensional também pode utilizar a função *WHERE*. Por exemplo,

```
IDL> d = dist (100)
IDL> surface, d
```




```
IDL> multi_ind = where (d gt 65, count)
IDL> print, count
     145
IDL> d [multi_ind] = 0.0
IDL> surface, d
```



Note que os comandos são praticamente iguais, aos de uma matriz unidimensional.

Estruturas

Uma estrutura no IDL, são diversas variáveis com tipos diferentes. Existem dois tipos de variáveis de estrutura no IDL: estruturas nomeadas e estruturas anônimas.

Estruturas Nomeadas

Uma estrutura nomeada é a estrutura que armazena o layout da estrutura sob um único nome no IDL. Os componentes de uma estrutura nomeada são o nome, os campos e os valores. As chaves são usadas exclusivamente para criação das estruturas.

```
IDL> p1 = { point, x: 1.0, y: 3.0, color: bytarr(3) }
```

A instrução define o nome da estrutura, *point*, este contém três campos, *x*, *y*, e *color*. A estrutura é armazenada na variável *p1*. O tipo de variável dos campos de uma estrutura é determinado pelo valor utilizado no campo. No caso da estrutura *point*, os campos *x* e o *y* são flutuantes e o *color* é um vetor de bytes. Para ter mais informações sobre a variáveis da estrutura, use a palavra-chave STRUCTURE para a rotina *HELP*.

```
IDL> help, p1, /structure
** Structure POINT, 3 tags, length=12, data length=11:
X                FLOAT        1.00000
Y                FLOAT        3.00000
COLOR  BYTE      Array[3]
```

Os campos de uma estrutura podem ser inicializados por variáveis. O tipo da variável determinará o tipo dos campos. Observe o exemplo abaixo.

```
IDL> a = 0UL
IDL> b = 0.9
IDL> c = 23.4D
IDL> my1 = { mystructure, f1: a, f2: b, f3: c }
```

A estrutura nomeada *mystruct* contém três tipos de campos: uma *unsigned long*, uma *float* e uma *double*.

Uma vez que uma estrutura nomeada esteja definida no IDL, os campos não podem mudar o tamanho ou o tipo, nem podem ser adicionados ou removido da estrutura. As definições de uma estrutura nomeada ficam armazenadas na memória do IDL até que a sessão seja encerrada ou reiniciada.
Para criar um novo exemplo da estrutura nomeada, para atribuir a estrutura para uma variável, use chaves e entre elas insira o nome das estruturas nomeadas.

```
IDL> p2 = { point }
IDL> my2 = { mystruct }
```

A variável *p2* contém uma cópia fiel da estrutura *point*. Quando uma nova estrutura nomeada é criada desta maneira, todos os campos serão zero ou nulos (*strings*, *pointers*, *objects*), os valores utilizados para definir a estrutura não serão carregados.

```
IDL> help, p2, /struture
** Structure POINT, 3 tags, length=12, data length=11:
```

```
X          FLOAT      0.000000
Y          FLOAT      0.000000
COLOR        BYTE     Array[3]
```

Acessando campos das estruturas

Para acessar um campo de uma estrutura, use o nome da estrutura com o nome do campo,

```
IDL> print, p2.x
      0.000000
IDL> p2.y = 9.5
IDL> p2.color = [255, 0, 255]
IDL> my2.f1 = 44L
```

Os campos das estruturas podem ser acessados por números.

```
IDL> print, p2. (0)                    ; da mesma forma que p2.x
      0.000000
IDL> p2. (1) = 9.5                     ; da mesma forma que p2.y
IDL> p2. (2) = [255, 0, 255]           ; da mesma forma que p2.color
IDL> my2.(0) = 44L                     ; da mesma forma que my2.f1
```

Estruturas Anônimas

Estruturas anônimas não tem um nome associado com elas.

```
IDL> an1 = { lat : 0.0, lon: 0.0, ht: 0L, pitch: 0.0 }
IDL> an1.lat = 12.4
IDL> an1.pitch = 45.67
```

Veja que não tem nenhum nome associado a ela, uma nova estrutura anônima pode ser definida adicionando ou removendo campos.

```
IDL> an1 = { lat : 0.0, lon: 0.0, ht: 0L, pitch: 0.0, attitude: 0U }
IDL> an2 = an1
IDL> an2.pitch = 79.0D
```

Trabalhando com Estruturas

Matrizes ou Vetores de Estruturas

Para criar uma matriz ou vetor de uma estrutura use a função *REPLICATE*. Esta função pode ser usada para copiar dados de qualquer tipo mas é usado freqüentemente para replicar matrizes ou vetores de estrutura. A matriz ou vetor criada pode ser replicada pelo nome da estrutura ou pelas suas variáveis.

```
IDL> ptarr = replicate ( { point }, 5 )
IDL> ptarr[0].x = 5.6
IDL> ptarr[0].y = 9.3
IDL> print, ptarr[0].x
        5.60000
```

```
IDL> print, ptarr[1].x
        0.000000
```

Cada campo da matriz ou vetor de estrutura pode ser tratado como se fosse uma estrutura normal.

```
IDL> ptarr.x = 6.7
IDL> print, ptarr[0].x
        6.70000
IDL> print, ptarr[1].x
        6.700000
```

A instrução acima fez com que todas as posições do vetor de estruturas *ptarr* do campo *x* passasse a ser 6.7.

```
IDL> ptarr.y = findgen ( 5 )
IDL> print, ptarr.y
        0.000000    1.00000    2.00000    3.00000    4.00000
```

A instrução acima mostra uma maneira de inicializar um vetor ou matriz com uma seqüência de números que começa em 0.000000.

Tipos e Tamanhos de Campos de uma Estrutura

Quando uma estrutura anônima ou nomeada é criada, o tipo e tamanho dos campos são definidos e não podem mais ser alterados. Quando você insere um valor no campo da estrutura, o IDL converte o valor ou variável para o mesmo tipo do campo.

```
IDL> anon = { nome : 'João', idade : 34, altura : 1.70, categoria : fltarr(10) }
IDL> anon.idade = 45.5                  ; o valor é convertido para integer
IDL> anon.altura = 70                   ; o valor é convertido para float
IDL> anon.categoria[9] = 78.54D         ; o valor é convertido para float
IDL> anon.categoria = fltarr (8)        ; tudo bem mas o campo ainda contém 10 valores
IDL> anon.categoria = intarr (10)       ; tudo bem mas o campo ainda é fltarr
IDL> help, anon, /structure
** Structure <10e8918>, 4 tags, length=60, data length=58, refs=1:
    NOME        STRING 'João'
    IDADE       INT          45
    ALTURA      FLOAT   70.0000
    CATEGORIA   FLOAT   Array[10]
IDL> anon.categoria = fltarr (15)       ; ERRO! Na tentativa de redimensionar o campo
```

O IDL é muito flexível com conversões, mas quando tratamos de um campo de matriz ou vetor ele não pode ser expandido.

Funções Variadas de uma Estrutura

Os nomes dos campos de uma estrutura podem ser retornados na forma de um vetor de *strings* usando a função *TAG_NAMES*.

```
IDL> fnames = tag_names (p2)
IDL> print, fnames
        X Y COLOR
```

A função *TAG_NAMES* pode ser usada também para retornar o nome da estrutura.

```
IDL> sname = tag_names (p2, /structure_name)
IDL> print, sname
        POINT
```

Para saber o número de campos em uma estrutura use a função *N_TAGS*.

```
IDL> ncampos = n_tags(p2)

IDL> print, ncampos
     3
```

A função *N_TAGS* pode também devolver o tamanho em bytes de uma estrutura.

```
IDL> slen = n_tags (p2, /length)
IDL> print, slen
     12
```

Trabalhando com Variáveis

A Exceção do Tipo *Byte*

A exceção para a regra, de que não se pode converter caracteres alfabéticos para números, é o tipo *byte*. Quando o IDL converte uma *string* de caracteres para *byte*, o resultado é um vetor de *byte* com o mesmo comprimento da *string*. Inversamente, uma matriz ou vetor de bytes convertido para uma *string*, resulta em uma *string* contendo os valores dos caracteres ASCII.

```
IDL> str = 'Hello World'
IDL> bstr = byte(str)
IDL> help, bstr
        BSTR        BYTE     = Array[11]
IDL> bstr [0] = bstr [0] + 6
IDL> print, string(bstr )
        Nello World
```

Estes comandos convertem uma *string* em um vetor de *bytes* e manipulação do primeiro *byte*. Adicionando seis no primeiro *byte* mudando o 'H' para 'N'. Convertendo os *bytes* de volta para *string* o resultado é 'NELLO WORLD'.

Variáveis Dinâmicas

O tipo de dados e a estrutura organizacional de uma variável muda dinamicamente. Por exemplo, esta é a forma errada de inicializar os elementos de uma matriz ou vetor de inteiros com o valor 1:

```
IDL> array = intarr(10, 12)
IDL> print, array
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0    0
IDL> array = 1
IDL> print, array
        1
```

No terceiro comando veja que é trocada a estrutura da variável nomeada *array* (que até então é uma matriz de 10 colunas e 12 linhas preenchida com zeros), ela passa a ser uma variável do tipo *integer* com valor 1.

Uma forma melhor de inicializar uma matriz ou vetor de inteiros com 1 é:

```
IDL> array = intarr( 10, 12)
IDL> array = array + 1
IDL> print, array
        1    1    1    1    1    1    1    1    1    1
```

```
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
```

Neste caso o segundo comando adiciona o valor 1 para todos elementos da variável nomeada *array*.

Os tipos de variáveis podem ser alterados dinamicamente.

```
IDL> variavel = 4
IDL> help, variavel
        VARIAVEL      INT     =      4
IDL> variavel = 4.0
IDL> help, variavel
        VARIAVEL      FLOAT    =      4.00000
IDL> variavel = '4.0'
IDL> help, variavel
        VARIAVEL      STRING    = '4.0'
```

Veja que a variável com nome de *variavel* muda de *integer*, para *float* e depois para *string*.

Usando o Tipo de Variáveis Errado

Tipos de variáveis dinâmicas são muito eficientes, mas podem se transformar em um problema se não tomarmos cuidado. Um erro muito comum que ocorre, é especificar uma divisão com resultado inteiro, quando temos uma divisão com resultado flutuante. Por exemplo:

```
IDL> valor = 16 / 5
IDL> print, valor
        3
```

A expressão acima resulta em um valor inteiro (3). Se você espera que o valor retornado seja 3.2, você precisaria rescrever esta expressão, utilizando um ponto (.) após o número cinco (5):

```
IDL> valor = 16 / 5.
IDL> print, valor
        3.20000
```

Variáveis de Sistema

O IDL tem uma classe especial de variáveis predefinidas que estão disponíveis para todas as unidades do programa. Estas variáveis são chamadas variáveis de sistema. Elas são identificadas por um ponto de exclamação antes do seu nome.

Algumas variáveis de sistema comuns, que fornecem apenas a informação de leitura.

Variáveis de Sistema	Descrição
!dtor	O fator da conversão dos graus para aos radianos
!radeg	O fator da conversão dos radianos para aos graus
!pi	Valor do pi com precisão simples
!dpi	Valor do pi com dupla precisão
!error_state	Uma variável do tipo estrutura contendo informações da ultima mensagem de erro.
!version	Dá informações sobre a versão em uso do IDL.

Gráficos Diretos de Saída

O *!d*, *!p* e o *!x*, *!y* e *!z* são variáveis de sistema usadas para fornecer informações sobre o dispositivo atual de gráfico direto. Estas variáveis de sistema são estruturas com campos múltiplos para identificar informações relacionadas ao gráfico.

A variável de sistema *!d* fornece informações sobre o atual dispositivo exposto. Alguns campos da variável de sistema *!d* estão listados na tabela abaixo.

Campos de *!d*	Descrição
name	O nome do atual dispositivo gráfico.
n_colors	O número de cores disponíveis no dispositivo gráfico.
table_size	O número índice da tabela de cor atual.
x_size	A largura da atual janela gráfica, em pixels.
y_size	A altura da atual janela gráfica, em pixels.
Window	O Índice da janela ativa dos gráficos.

A variável de sistema *!p* permite que você altere configurações padrões de um gráfico em linha, gráfico de contorno e um gráfico de superfície. Alguns dos campos da variável de sistema *!p* estão na tabela abaixo.

Campos de *!p*	Descrição
color	O índice de cor do gráfico.
charsize	O tamanho do caracter para o rótulo.
linestyle	O estilo de linha utilizado no gráfico.
multi	Especifica o número de gráficos em uma página.
position	Especifica a posição do gráfico em uma página.
psym	O estilo de símbolo utilizado no gráfico.
subtitle	Uma palavra (frase) utilizado como subtítulo, colocado abaixo do eixo X.
title	Uma palavra (frase) especificando o titulo do gráfico.

O *!x*, *!y* e *!z* são variáveis de sistema que permitem a você alterar as configurações padrões dos eixos de um gráfico. Alguns exemplos destes campos, estão listados na tabela abaixo.

Campos de *!x*, *!y* e *!z*	**Descrição**
minor	O número de pequenos intervalos dentro de grandes intervalos.
range	Marca a distância mínima e máxima dos eixos.
tickname	Colocar uma matriz ou vetor (geralmente do tipo *string*) de anotações para cada intervalo maior.
ticks	O número de intervalos constantes no eixo.
title	O título dos eixos.

Operadores

Operadores são elementos de programação utilizados para combinar valores, variáveis e expressões em uma expressão ou instrução mais complexa. A tabela abaixo mostra os operadores que o IDL tem suporte, sua precedência, e mostra alguns exemplos de uso destes operadores.

Prioridade	Operadores	Descrição	Exemplos
Primeiro	( )	Parênteses para agrupar.	IDL> print, 3*2+1, 3*(2+1) *7 9*
	[ ]	Colchetes para concatenar matrizes ou vetores.	IDL> a = indgen (3) IDL> print, [a, -17] 0 1 2 -17
Segundo	( )	Parênteses em uma chamada de função.	IDL> print, indgen (3)*2 0 2 4
	[ ]	Colchetes para sobrescrever matrizes ou vetores.	IDL> a = indgen (5) IDL> print, a[3]*2 6
	.	Referência a estruturas.	IDL> s = {val :5} IDL> print, s.val*2 10
Terceiro	*	Referência a ponteiros.	IDL> p = ptr_new(1) IDL> print, *p 1
	++	Incrementar.	IDL> a = 5 IDL> print, a++, a, ++a 5 7 7
	--	Decrementar.	IDL> print, a--, a, --a 7 5 5
	^	Exponenciação.	IDL> print, a^2, a^(-2) 25.0000 0.0400000
Quarto	# e ##	Multiplicação de matrizes.	(veja logo a frente “Multiplicação de Matrizes”)
	*	Multiplicação.	IDL> print, 5*2 10
	/	Divisão.	IDL> print, 5/2, 5/2. 2 2.50000
	mod	Módulo.	IDL> print, 5mod2 1
Quinto	+	Adição / Concatenação de *strings*.	IDL> print, 5+2 7 IDL> print, 'Dave'+' Stern' Dave Stern
	-	Subtração / Negação unária.	IDL> print, 5-2, -5 3 -5
	<	Mínimo.	IDL> print, 5 < 2 2
	>	Máximo.	IDL> print, 5 > 2 5
	not	Complemento Bit a Bit.	IDL> print, not 5 -6
Sexto	eq	Igual á.	IDL> print, 5eq2, 2.0eq2 0 1
	ne	Diferente de.	IDL> print, 5ne2, 2.0ne2 1 0
	le	Igual ou menor que:	IDL> print, 5 le 2, 2.0 le 2 0 1
	lt	Menor que:	IDL> print, 5 lt 2, 2.0 lt 2 0 0

	ge	Maior ou igual que:	IDL> print, 5 ge 2, 2.0ge2 1 1
	gt	Maior que:	IDL> print, 5 gt 2, 2.0gt2 1 0
Sétimo	and	E Bit a Bit.	IDL> print, 5 and 6 4
	or	OU Bit a Bit.	IDL> print, 5 or 6 7
	xor	XOR Bit a Bit.	IDL> print, 5 xor 6 3
Oitavo	&&	E Lógico.	IDL> print, 5 && 6 1
	||	OU Lógico.	IDL> print, 5 || 6 1
	~	Negação Lógica.	IDL> print, ~5 0

Quando operadores do mesmo nível de precedência são usados sem serem agrupados em uma expressão, eles são avaliados da esquerda para direita. Por exmplo:

```
IDL> print, 5 mod 2 * 7
        7
```

Operação de Matrizes ou Vetores

Os operadores do IDL podem ser usados com matrizes ou vetores da mesma forma que são usados na escalar.

Por exemplo:

```
IDL> print, indgen(5) mod 2
        0     1     0     1     0
```

Aqui, o operador modulo (MOD) é aplicado em cada elemento retornado da função *INDGEN*.

```
IDL> print, indgen(5) > 2
        2     2     2     3     4
IDL> print, indgen(5) gt 2
        0     0     0     1     1
```

Neste caso, os operadores são aplicados em cada elemento do vetor.

Operadores Compostos

O IDL começou a ter suporte a estes operadores a partir da versão 6.0 . Os operadores válidos estão listados na tabela abaixo.

*=	<=	eq=	mod=
##=	^=	ge=	#=
xor=	gt=	-=	or=
le=	+=	and=	lt=
/=	>=	ne=	

Estes operadores compostos podem ser combinados com outros operadores. Uma instrução como essa

```
a op = expressão
```

é equivalente á

```
a = temporário (a) op (expressão)
```

onde *op* é um operador do IDL podendo ser combinado para formar um dos operadores compostos da lista acima, e *expressão* é qualquer expressão do IDL, o resultado será o mesmo que da primeira instrução.

Por exemplo:

```
IDL> a = 3
IDL> a += 5 * !pi
IDL> print, a
        18.7080
```

O valor final é o valor inicial mais 5 vezes π.

Instruções de Comando

Estas instruções são logicamente equivalentes a instruções de comando de outras linguagens de programação. Elas freqüentemente tem uma sintaxe similar, ou até idêntica.

Instruções Compostas

Por padrão, no IDL uma instrução de comando é uma única etapa da execução. Diversas etapas de execuções podem resultar em uma instrução de comando quando ela estiver agrupada por BEGIN-END. Isto é chamado de uma instrução composta. Por exemplo, esta instrução de comando IF–THEN.

```
If (x lt 0) then print, x
```

Não precisa de BEGIN-END, enquanto esta instrução

```
If (x lt 0 ) then begin
        print, x
        y = -x
endif
```

precisa.

Neste exemplo, a instrução END toma a forma de ENDIF. END é o suficiente para fechar qualquer BEGIN, mas usando o ENDIF deixa o código mais fácil de se entender, o que é desejável. Cada instrução de comando no IDL tem a sua versão de END. Note que o BEGIN e o END são palavras chaves reservadas, então você não pode usa-las como variáveis ou nome de programas.

Instruções Condicionais

Instruções condicionais são usadas para retornar diferentes resultados para diferentes entradas.

IF-THEN-ELSE

A instrução IF é usada para executar uma condição de uma instrução ou de um bloco de instruções.
Exemplo:

```
x = randomu (seed, 1) – 0.5
if ( x gt 0 ) then print, x
if ( x gt 0 ) then y = x else y = -x
```

ou desta forma

```
x = randomu (seed, 1) – 0.5
if ( x gt 0 ) then begin
        print, x
        y = x
endif else begin
        y = -x
```

```
endelse
```

Os parênteses em torno da condição não são necessários, mas eles deixam o código mais fácil de ser compreendido.

CASE

A instrução CASE é usada para selecionar uma instrução para que ela seja executada, dependendo o valor do seletor de expressão do CASE.

```
case pedacodetorta of
        0: begin
                torta = 'maca'
                topping = 'sorvete'
           end
        1: torta = 'abobora'
        else: torta = 'cherry'
endcase
```

A clausula ELSE é opcional, mas note que é preciso um sinal de dois pontos, diferente da clausula ELSE na instrução IF. Se a expressão resultar em um valor que não está especifico no CASE, e não existir o ELSE, resultara em um erro.

SWITCH

A instrução SWITCH é usada para selecionar uma ou mais instruções para execução, dependendo o valor do seletor de expressão do SWITCH. O SWITCH difere-se do CASE nisso, se uma combinação for encontrada os itens subsequentes da lista serão executados, até que um ENDSWITCH ou um BREAK seja encontrado.

CASE	**SWITCH**
x = 2 case x of 1: print, 'one' 2: print, 'two' 3: print, 'three' 4: print, 'four' endcase	*x = 2* switch x of 1: print, 'one' 2: print, 'two' 3: print, 'three' 4: print, 'four' *endswitch*
IDL Mostra: two	IDL Mostra: two three four

```
x = randomn (seed)
switch 1 of
x lt 0: begin
        print, 'x é menor que zero'
        break
       end
x gt 0: print, 'x é maior que zero'
x eq 0: print, 'x é igual a zero'
else:
endswitch
```

Note que se o *x* é maior que zero, as duas últimas instruções do SWITCH foram executadas.

Instrução de Repetição

Uma tarefa muito comum na programação é executar uma instrução, ou várias instruções, múltiplas vezes.

FOR-DO

A instrução de repetição FOR, executa uma instrução ou um bloco de instruções por um número específico de vezes.

```
for j = 0, 9 do print, j
```

É possível especificar a magnitude do contador da repetição, assim como a sua direção. Por exemplo, no código a seguir o contador da repetição está decrementando em dois com cada interação com a repetição.

```
sequencia = findgen(21)
for j=20, 0, -2 do begin
        print, j, sequencia[j], sequencia[j] mod 4
endfor
```

WHILE-DO

A instrução de repetição WHILE, executa uma instrução ou um bloco de instruções pelo tempo em que a condição teste for verdadeira. A condição é checada no ponto de entrada da repetição.

```
previous = 1 & current = 1
while (current lt 100) do begin
        next = current + previous
        previous = current
        current = next
endwhile
```

Este código gera a seqüência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, ...

REPEAT-UNTIL

A instrução de repetição REPEAT é similar ao WHILE, exceto pela condição de teste da repetição ser no ponto de saída.

```
repeat begin
        n++
endrep until (n gt 20)
```

Instruções de Controle de Fluxo

A instrução de controle de fluxo pode ser usada para modificar o comportamento de instruções condicionais e interativas. A instrução de controle de fluxo permite que você saia de uma repetição, e possa retornar na próxima interação com a repetição.

BREAK

A instrução BREAK fornece uma forma imediata de sair de um FOR, WHILE, REPEAT, CASE, ou SWITH.
Exemplo:

```
FOR i = 0,n-1 DO BEGIN
        IF (array[i] EQ 5) THEN BREAK
ENDFOR
PRINT, i
```

CONTINUE

A instrução CONTINUE fornece uma forma imediata de começar a próxima repetição incluindo as instruções de repetição FOR, WHILE, ou REPEAT. Visto que a instrução BREAK sai do laço de repetição, a instrução CONTINUE sai apenas do atual laço de repetição, passando direto para a próxima repetição no mesmo laço.
Exemplo:

```
FOR I=1,10 DO BEGIN
        IF (I AND 1) THEN CONTINUE
        PRINT, I
ENDFOR
```

A Definição de Verdadeiro (true) e Falso (false)

O que é verdadeiro e o que não é, para os diferentes tipos de dados do IDL.

Variável ou tipo de expressão	Instruções Verdadeiras
Inteiros (*byte*, *integer*, *long*)	Valores impares são verdadeiros, valores pares são falsos.
Flutuantes (*float*, *double*, *complex*)	Valores diferentes de zero são verdadeiros, valores iguais a zero são falsos; a parte imaginária de um número complexo é ignorada.
String	Comprimento diferente de zero é verdadeiro, uma *string* nula é falsa.
Variáveis de Memória Heap (*pointers*, *objects*)	Variáveis de memória com dados válidos são verdadeiras, variáveis de memória com dados nulos são falsos.

Em uma unidade de programa, se o LOGICAL_PREDICATE for ativado pela instrução COMPILE_OPT, todos os valores iguais a zero ou nulos serão falsos, ou se não forem nulos ou diferentes de zero serão verdadeiros.

Para maiores informações sobre a instrução COMPILE_OPT, olhe no IDL Online Help.

Rotinas de Manipulação de Arquivos

No IDL existem muitas rotinas de manipulação de arquivos. Uma lista destas rotinas, com uma explicação da sua finalidade, está na tabela abaixo.

Rotinas	Finalidade
FILEPATH	Constrói uma *string* contendo o caminho absoluto do arquivo.
FILE_BASENAME	Esta função retorna o nome base do arquivo. O nome base é o segmento mais à direta no final do caminho até o arquivo.
FILE_DIRNAME	Esta função retorna o nome do diretório do caminho até o arquivo. O nome do diretório é todo o caminho até o arquivo exceto o segmento mais à direita.
FILE_EXPAND_PATH	Qualificação total do caminho do diretório e até o arquivo.
FILE_CHMOD	Permite que você altere as permissões de acesso ao arquivo.
FILE_COPY	Nos permite copiar arquivos, e ou diretórios dos arquivos, para uma nova localização.
FILE_DELETE	Apaga arquivos ou esvazia diretórios.
FILE_INFO	Esta função retorna informações sobre o arquivo, incluindo seu nome, tamanho, permissões, acessos e tempos de modificações.
FILE_LINES	Informa o número de linhas de texto contidas no arquivo.
FILE_LINK	Este procedimento cria links para arquivos UNIX, regulares e simbólicos.
FILE_MKDIR	Cria um novo diretório nos arquivos de sistema.
FILE_MOVE	Move arquivos, ou diretórios de arquivos, para uma nova localização.
FILE_READLINK	Retorna o caminho apontado para o Link simbólico do UNIX.
FILE_SAME	Esta função é usada para determinar se dois arquivos com nomes diferentes referem-se ao mesmo arquivo fundamentado.
FILE_SEARCH	Encontra os arquivos cujos os nomes combinam com a *string* especificada.
FILE_TEST	De o caminho até o arquivo, que esta rotina retorna 1 se o arquivo existir, senão ele retorna 0.
FILE_WHICH	Procura por um arquivo específico no caminho de busca do diretório.
FSTAT	Relata informações sobre um arquivo aberto.
DIALOG_PICKFILE	Esta rotina nos permite selecionar interativamente um arquivo, ou diversos arquivos, ou um diretório.
EOF	Esta função uma unidade específica de um arquivo para a condição de fim do arquivo (end-of-file).
FLUSH	Este procedimento força todas as saída protegidas nas unidades especificadas do arquivo a ser escrito.
PATH_SEP	Retorna um delimitador apropriado do caminho de um arquivo para o atual sistema de operação.
COPY_LUN	Copia os dados entre dois arquivos abertos.
POINT_LUN	Marca ou obtém a posição atual do ponteiro de um arquivo em um arquivo especificado.
SKIP_LUN	Lê dados de um arquivo aberto e move o arquivo de

	ponteiro.
TRUNCATE_LUN	Este procedimento divide os conteúdos de um arquivo (que deve estar aberto em modo de escrita) na posição atual do ponteiro de arquivo.
SOCKET	Abre um socket TCP/IP para o cliente como uma unidade de arquivo do IDL.

Abaixo estão alguns exemplos de uso das rotinas da tabela acima.

Usando o *FILE_WhICH* para localizar o arquivo contendo o procedimento *LOADCT*.

```
IDL> file = file_which('loadct.pro')
IDL> print, file
        C:\RSI\IDL61\lib\loadct.pro
```

Extraindo o nome base e o nome do diretório deste arquivo com o *FILE_BASENAME* e o *FILE_DIRNAME*.

```
IDL> print, file_basename(file)
        loadct.pro
IDL> print, file_dirname(file)
C:\RSI\IDL61\lib
```

Use o *FILE_LINES* para contar o número de linhas que temos no arquivo **loadct.pro**.

```
IDL> print, file_lines(file)
   185
```

Exiba o diretório atual com *FILE_EXPAND_PATH*.

```
IDL> print, file_expand_path('.')
        C:\RSI\IDL61
```

Exiba o nome dos arquivos começados com a letra a no subdirectório **lib** do IDL.

```
IDL> str = !dir + path_sep() + 'lib' + path_sep() + 'a*'
IDL> print, file_basename(file_search(str, /fold_case))
        a_correlate.pro adapt_hist_equal.pro amoeba.pro annotate.pro array_indices.pro arrow.pro
        ascii_template.pro
```

Veja o IDL Online Help para maiores informações das rotinas listadas na tabela acima.

Rotinas de Alto Nível para Arquivos

Rotinas de alto nível para arquivos são fáceis de serem usadas, mas lhe dão menos controle sobre a descrição de dados em um arquivo no formato ASCII ou binário.

Texto Simples e Arquivos Binários

A tabela abaixo nos apresenta quatro rotinas designadas para simplificar o processo de leitura de textos simples e de arquivos binários para o IDL.

Rotinas	Finalidade
ASCII_TEMPLATE	Um programa de UI (interface para o usuário) que pode ser utilizado para descrever o formato de um arquivo de texto. Retornando uma variável de estrutura que possa ser usada pelo *READ_ASCII* para ler o arquivo.
READ_ASCII	Lê os dados de um arquivo de texto para uma variável de estrutura. O formato deste arquivo pode ser especificado com palavras-chave ou pelo *ASCII_TEMPLATE*.
BINARY_TEMPLATE	Como o *ASCII_TEMPLATE*, um programa de UI que pode ser usado para descrever o conteúdo de um arquivo binário. Retornando uma variável de estrutura que possa ser utilizada pelo *READ_BINARY* para ler o arquivo.
READ_BINARY	Lê os dados de um arquivo binário para uma estrutura ou uma matriz. A organização deste arquivo pode ser especificada por palavras-chave ou pelo *BINARY_TEMPLATE*.

Abra o arquivo **ascii.txt** no subdiretório (do diretório de instalação do IDL) **examples**/**data** com um editor de texto a sua escolha. Note que existem quatro linhas de cabeçalho, seguidas por uma linha em branco, depois 7 colunas e 15 linhas com uma vírgula delimitando cada dado. Abaixo temos um exemplo de como usar o *READ_ASCII* para ler as informações deste arquivo para o IDL.

```
IDL> x = filepath('ascii.txt', subdir = ['examples', 'data'])
IDL> dados = read_ascii (x, data_start=5, header=cabeçalho)
IDL> help, dados, /structures
** Structure <10a0a30>, 1 tags, length=420, data length=420, refs=1:
   FIELD1          FLOAT     Array[7, 15]
```

Os dados do arquivo foram lidos para uma variável de estrutura contendo um campo: ponteiro para uma matriz flutuante de 7X15. Extraia uma coluna desta matriz.

```
IDL> elevacao = dados.(0) [2,*]
IDL> print, elevacao [0:5]
        399.000      692.000      1003.00      1333.00      811.000      90.0000
```

Tente ler este arquivo usando um modelo (template) de definição com o *ASCII_TEMPLATE*.
ASCII_TEMPLATE fornece uma interface gráfica com um processo de três etapas descrevendo o conteúdo de um arquivo. *ASCII_TEMPLATE* retorna uma variável de estrutura definindo um modelo reutilizável.

```
IDL> x_template = ascii_template(x)
```

No IDL Online Help na página do *ASCII_TAMPLATE* tem alguns exemplos mais detalhados do seu uso, e também captações da tela (screenshoots). Use o *READ_ASCII* com o modelo que você criou para ler o conteúdo contido no arquivo **ascii.txt**.

```
IDL> dados = read_ascii (x, template=x_template)
IDL> wind_speed = dados.(5)
IDL> print, wind_speed [0:5]
10        8        10        0        8        10
```

O arquivo **convec.dat** no subdiretório **exemplos/data** é um exemplo de arquivo binário. Leia o seu conteúdo para a variável *binario* do IDL com o *READ_BINARY*.

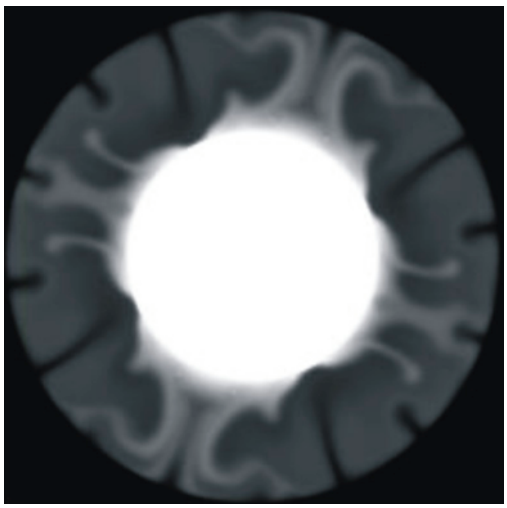
```
IDL> y = filepath('convec.dat', subdir = ['examples', 'data'])
IDL> binario = read_binary(y)
IDL> help, binario
BINARIO        BYTE      = Array[61504]
```

O arquivo foi lido, mas precisamos saber o que pode ser feito com estes dados. No arquivo **index.txt** no subdiretório **exemplos/data**, podemos ver que este arquivo contem a descrição de diversos arquivos incluindo o arquivo **convec.dat** que é uma matriz de *bytes* de 248 X 248 elementos representando um modelo de um manto de condução de calor. Esta informação suplementar é necessária para compreender o conteúdo do arquivo. Tendo esta informação, use a palavra-chave DATA_DIMS para o *READ_BINARY* ler o conteúdo do arquivo.

```
IDL> binario = read_binary(y, data_dims=[248,248])
IDL> help, binario
BINARIO        BYTE      = Array[248, 248]
```

Veja os dados como imagem com o comando *TV*.

```
IDL> tv, binario
```



Maiores informações sobre as rotinas que foram listadas na tabela anterior podem ser encontradas no IDL Online Help.

Rotinas de Baixo Nível para Arquivos

Esta seção descreve o uso do grupo de rotinas *OPEN*, *CLOSE*, *READ*, e *WRITE*. O uso destas rotinas de baixo nível para arquivos requer um pouco mais de conhecimento do IDL, tipos de arquivos e operações de sistemas, mas como um usuário, você terá um grande controle de como esta sendo executada a entrada/saída atual do arquivo.

Ao utilizar rotinas de baixo nível para arquivos do IDL, ajuda ter em mente algumas simples etapas para executar arquivos de entrada ou saída.

1. Encontre o arquivo no sistema de arquivos. Isto envolve tipicamente especificar o caminho de um arquivo e armazená-lo em uma variável, por exemplo, *FILEPATH*, *FILE_SEARCH* ou *DIALOG_PICKFILE*.

2. Defina o formato dos dados para o conteúdo do arquivo. Decidir como o IDL deve representar os dados em um arquivo é geralmente a parte mais difícil de usar rotinas de baixo nível.

3. Abra o arquivo para leitura, escrita ou atualização.

4. Executa arquivos de operações (queryng, reading, writing, positioning) dento do arquivo.

5. Feche o arquivo.

Abrindo e Fechando Arquivos

A tabela abaixo lista as rotinas, para abrir e fechar arquivos do IDL.

Rotinas	Finalidade
OPENR	Abre um arquivo para leitura.
OPENW	Abre um arquivo para escrita.
OPENU	Abre um arquivo para atualização (leitura e/ou escrita).
CLOSE	Fecha um arquivo ou uma lista de arquivos.
FREE_LUN	Fecha arquivos e limpa as unidades de arquivo.

Abaixo temos um exemplo da sintaxe usada para abrir e fechar um arquivo.

```
IDL> a = filepath('ascii.txt', subdir = ['examples','data'])
IDL> openr, 1, a
```

O procedimento *OPENR* é usado para abrir um arquivo em modo de leitura. O valor 1 é número da unidade lógica para o arquivo. Use a palavra-chave FILES para o comando *HELP* e veja quais arquivos estão abertos na sua sessão do IDL.

```
IDL> help, /files
Unit   Attributes                    Name
  1     Read                         C:\RSI\IDL61\examples\data\ascii.txt
```

Feche o arquivo com o procedimento *CLOSE*.

```
IDL> close, 1
```

Números de Unidades Lógicas

Um número de unidade lógica (LUN) é um número simples associado a um arquivo do IDL. Todos arquivos abertos são determinados por um LUN quando são abertos, e todas rotinas de entrada/saída deste arquivo serão referidas a este número. Vários arquivos podem ser abertos simultaneamente no IDL, cada um com um número de unidade lógica diferente.

Três números de unidades lógicas são reservados para uso do sistema operacional.

0, stdin : Este LUN representa o caminho padrão de entrada, que é geralmente o teclado.
-1, stdout : Este LUN representa a caminho padrão da saída, que é geralmente a tela do terminal.
-2, stderr : Este LUN representa a caminho padrão de erro, que é geralmente a tela do terminal.

Há 128 números de unidade lógicas disponíveis ao usuário.

1 – 99	Especifico diretamente a rotina *OPEN*.
100 – 128	Especifico para o *GET_LUN*, ou a palavra-chave GET_LUN para a rotina *OPEN*.

Um exemplo da especificação direta de um LUN é dado acima. Quando um LUN na escala de 1-99 é atribuído a um arquivo, não pode ser atribuído novamente até que o arquivo esteja fechado, a sessão atual do IDL seja reiniciada (reset), ou quando você sai do IDL.

O procedimento *GET_LUN*, ou a palavra-chave GET_LUN para o procedimento *OPEN*, pode ser usado deixando que o IDL especifique um LUN na escala de 100 – 128. Isto é particularmente usado para prevenir conflitos com outros LUNs já em uso.

Por exemplo, abra o arquivo **convec.dat** para leitura, deixe que o IDL escolha o LUN.

```
IDL> b = filepath('convec.dat', subdir=['examples','data'])
IDL> openr, lun, b, /get_lun
IDL> help, lun
LUN          LONG     =        100
```

Veja que quando utilizamos a palavra-chave GET_LUN no exemplo acima a variável *lun* recebe o valor 100.
Verifique que o arquivo está aberto com o procedimento *HELP*.

```
IDL> help, /files
Unit   Attributes                    Name
100          Read, Reserved
C:\RSI\IDL61\examples\data\convec.dat
```

Uma unidade de arquivo alocado com *GET_LUN* é liberado com o procedimento *FREE_LUN*. O procedimento *FREE_LUN* fecha o arquivo e limpa o LUN.

```
IDL> free_lun, lun
```

Rotinas do ENVI

Neste capítulo serão apresentadas algumas rotinas do ENVI, que são executadas através do IDL.
Existem atualmente mais de 150 rotinas de processamento do ENVI disponíveis, que abrangem virtualmente todas as funcionalidades fornecidas no programa. O processamento de cada rotina é feita da mesma forma que nos procedimentos ou funções do IDL, e é usada da mesma forma que qualquer outra rotina do IDL.

Palavras-chave Comuns para Rotinas de Processamento do ENVI

Quando é adquirida maior experiência em processamentos de rotinas do ENVI poderá ser observado que há diversas palavras-chave que são comuns para cada rotina. Estas palavras-chave fazem o controle básico de entrada e saída do arquivo que será processado.

FID

O ID do arquivo (File ID) é um *long integer*. O FID é fornecido ao programador como uma variável nomeada por uma das diversas rotinas do ENVI que podem ser usados para usar para abrir ou selecionar um arquivo. Todo o processado de um arquivo usando uma rotina do ENVI é realizado consultando seu FID. Entretanto, se você estiver trabalhando com o arquivo diretamente no IDL, o FID não é equivalente a um número de unidade lógico.
Se houver algum problema para abrir ou acessar um arquivo, o FID irá receber –1. FDIs com valor –1 são inválidos e não podem mais serem processados. Um programador de IDL que utiliza as rotinas do ENVI, deve sempre checar a possibilidade do FID, R_FID, e M_FID serem inválidos. Quando um arquivo de ID inválido é encontrado, você geralmente retorna apenas para a atual rotina em processo.

R_FID e M_FID

As rotinas do ENVI que resultam em novas imagens terão um R_FID, ou a palavra-chave correspondente ao FID de retorno. Se os resultados forem somente salvos na memória, o único método para acessá-los será ativando a palavra-chave R_FID. Rotinas que permitem o uso de máscara devem ter um M_FID, ou uma palavra-chave equivalente ao M_FID para especificar que o arquivo use uma máscara de banda "mask band".

DIMS

A palavra-chave das dimensões (DIMS) é uma matriz ou vetor de inteiros longos (*long integer array*) com 5 elementos, que define um subconjunto (subset) regional (de um arquivo ou uma matriz ou vetor) que será usado no processamento. Quase sempre que você especificar um FID, você terá também que especificar qual subconjunto regional deste arquivo será usado (mesmo com um arquivo completo, sem um subconjunto regional, para ser processado).
Deve ser salientado que no IDL os índices são começados em zero, como comentado anteriormente neste guia.

DIMS[0] = um ponteiro para abrir uma região de interesse, usada apenas nos casos onde um subconjunto regional é definido pela região de interesse (ROI), caso contrário ele receberá –1.

DIMS [1] = o número que começa a coluna (X) (uma subscrição de uma matriz ou vetor na base zero do IDL).

DIMS [2] = o número que termina a coluna (X).

DIMS [3] = o número que começa a linha (Y).

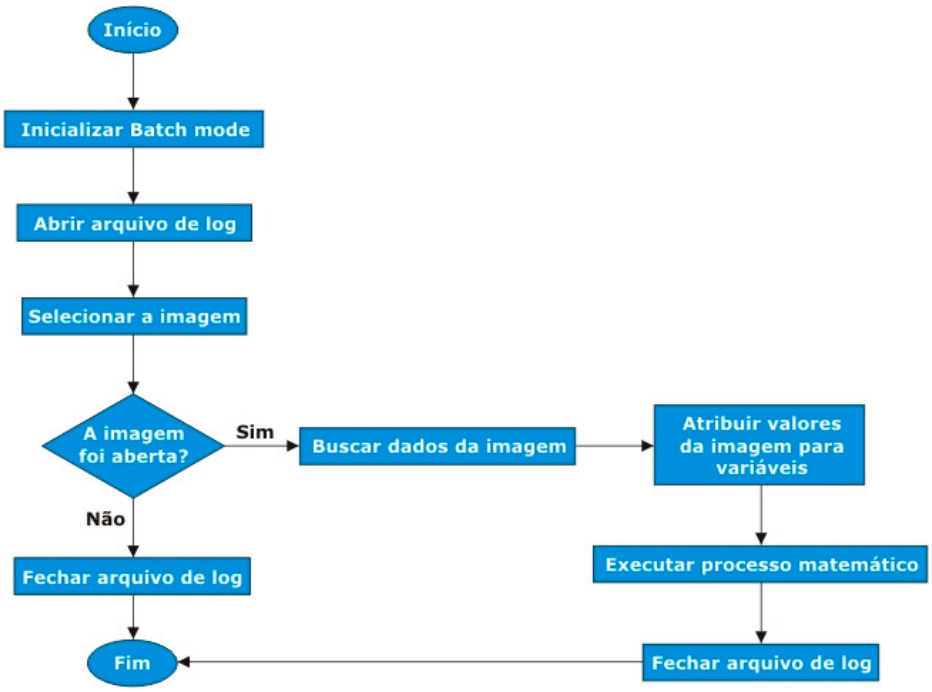
DIMS [4] = o número que termina a linha (Y).

POS

A palavra-chave POS define a posição da(s) banda(s) a serem usada(s) para o processamento, é uma matriz de inteiros longos com o seu comprimento variável, com tantos elementos quanto bandas. Os arquivos que são processados pelo ENVI podem ter bandas múltiplas, um subconjunto espectral (spectral subset) das bandas que serão usados para o processamento específico, pelo vetor de POS. As bandas começam especificamente em zero (Band 1 = 0, Band 2 = 1, Band 3 = 2, etc.). Por exemplo, para processar apenas a 4º e a 5º Banda de um arquivo de bandas múltiplas (multiband), POS = [3,4]. Muitas rotinas do ENVI permitem apenas o processamento de uma banda (*ENVI_GET_DATA*), caso esteja usando uma destas rotinas você poderá utilizar laços de repetição (Loop), para processar diversas bandas.

Exemplos de Rotinas do ENVI para uso no IDL

Abaixo pode ser visualizado um exemplo simples do uso de algumas rotinas do ENVI (que serão descritas após o exemplo), sendo utilizadas para processar uma operação máxima entre as bandas de uma imagem (no exemplo abaixo é utilizada a imagem **can_tmr.img**, que esta localizado por padrão no diretório **RSI\IDL61\products\ENVI41\data**).



```
1   Pro Exemplo_ENVI
2      ;
3      ; Restaura a base de arquivos do ENVI.
4      ;
5      envi, /restore_base_save_files
6      ;
7      ; Inicializa o ENVI em modo "batch" .
8      ;
9      envi_batch_init, log_file='exemplo.txt'
10     ;
11     ; Abre um arquivo de imagem.
12     ;
13     envi_select, fid=fid
14     ;
15     ; Teste para ver se o arquivo realmente foi aberto.
16     ;
17     if (fid eq -1) then begin
18        envi_batch_exit
19        return
20     endif
21     ;
22     ; Atribui valores da imagem para variáveis, que serão
23     ; utilizadas na função math_doit, como palavras-chave.
24     ;
25     envi_file_query, fid, ns=ns, nl=nl
26     t_fid = [fid,fid,fid]
27     dims = [-1, 0, ns-1, 0, nl-1]
28     pos = [0,1,2]
29     exp = '((b1 > b2) > b3)'
30     out_name = 'exemplo'
31     ;
32     ; Executa o processo matemático entre as bandas.
33     ;
34     envi_doit, 'math_doit', fid=t_fid, pos=pos, dims=dims, exp=exp, out_name=out_name
35     ;
36     ; Sair do ENVI
37     ;
38     envi_batch_exit
39  end
```

O exemplo acima executa uma operação com operadores de máximo entre três bandas de uma imagem, criando uma nova imagem e salvando-a no diretório raiz do IDL (por padrão **RSI\IDL6X**).
Na linha 10 do exemplo acima, é criado um arquivo chamado **exemplo.txt** no diretório raiz do IDL, este arquivo irá receber todas advertências e avisos de erros, que ocorrerem durante a execução do programa.
Na linha 13 é selecionado um arquivo de imagem, no qual desejamos que seja processado, este arquivo deve ser de um tipo reconhecido automaticamente pelo ENVI (TIF, JPEG, BMP, etc), caso contrário você terá que informar os parâmetros de entrada deste arquivo.
Na linha 17, temos uma instrução de condição (*IF*), verificando se realmente o arquivo foi aberto, se a imagem não foi aberta será perguntado se o IDL deve ser encerrado (linha 18), caso a resposta seja negativa ele irá executar o procedimento *RETURN* (linha 19).

Na linha 25 a rotina *ENVI_FILE_QUERY* através da palavra-chave FID, busca o número total de colunas (NS), e o número total de linhas (NL), que são passadas para uma variável *dim* (linha 27), a palavra-chave EXP da rotina *MATH_DOIT* receberá esta variável (linha 34).
Na linha 29 temos a variável *exp* recebendo uma operação de máximo, que será usada para atribuir valor a palavra-chave EXP da rotina *MATH_DOIT* (linha 34).
Para fazer o download deste exemplo acesse o link abaixo:
http://www.sulsoft.com.br/idl/guiadireto/exemplos/exemplo_envi.pro .

ENVI

Esta rotina é usada para restaurar a base de arquivos salvos do ENVI para o modo "batch" (restaura os arquivos do ENVI sem ativar o modo interativo), pode ser usada também para abrir o ENVI através do IDL. Para restaurar a base de arquivos, use a palavra-chave RESTORE_BASE_SAVE_FILES.

Sintaxe

ENVI [,/RESTORE_BASE_SAVE_FILES]

ENVI_BATCH_INIT

Use esta rotina para iniciar o ENVI em modo "batch", que é a modalidade sem a interface de menu do ENVI. Muitas das rotinas de processamento de imagens do ENVI que não requerem a interação do usuário foram projetados para rodar no modo "batch". Quando utilizamos a palavra-chave LOG_FILE, e atribuímos a um arquivo do tipo txt, todos os erros e avisos ocorridos durante a execução serão enviados para este arquivo.

Sintaxe

ENVI_BATCH_INIT [,PALAVRAS-CHAVE]

ENVI_SELECT

Este procedimento fornece a janela de diálogo padrão de seleção de arquivos do ENVI. Em seu formato padrão esta janela de diálogo permite a seleção de um arquivo ou uma banda com a opção para subconjunto regional e ou espectral. Entretanto, a janela de diálogo dispõe de diversas palavras-chave que permitem o controle preciso sobre todas opções durante a seleção dos dados.

Uma vez que a janela de diálogo esteja iniciada, o restante da sessão fica aguardando até que seja selecionado o botão OK ou CANCEL. Caso o botão selecionado seja o OK o valor retornado no FID é o id do dado selecionado. Mas se o botão selecionado for o CANCEL o valor retornado ao FID será –1. Este valor de FID é usado em diversas rotinas do ENVI e é um link com o arquivo selecionado.

Sintaxe

ENVI_SELECT [,PALAVRAS-CHAVE]

ENVI_BATCH_EXIT

Use esta rotina para sair do modo "batch" do ENVI, e para encerrar a sessão atual do ENVI.
Nas preferências do ENVI temos uma propriedade (**Exit IDL on Exit from ENVI**, localizado na guia Miscellaneous das preferências do ENVI) indicando que o IDL pode ser finalizado quando a sessão do ENVI é encerrada, por padrão esta propriedade esta marcada com **yes** (que significa que o IDL será encerrado junto ao ENVI), que pode ser alterada caso seja necessário. A rotina *ENVI_BATCH_EXIT* encerra a sessão que foi iniciado com *ENVI_BATCH_INIT*.

Sintaxe

ENVI_BATCH_EXIT [,PALAVRAS-CHAVE]

ENVI_FILE_QUERY

Esta rotina consulta as informações contidas no arquivo de cabeçalho da imagem. Caso a imagem não tenha um arquivo de cabeçalho (**.hdr**), será aberta a janela de edição de cabeçalho (Hearder Info).

Sintaxe

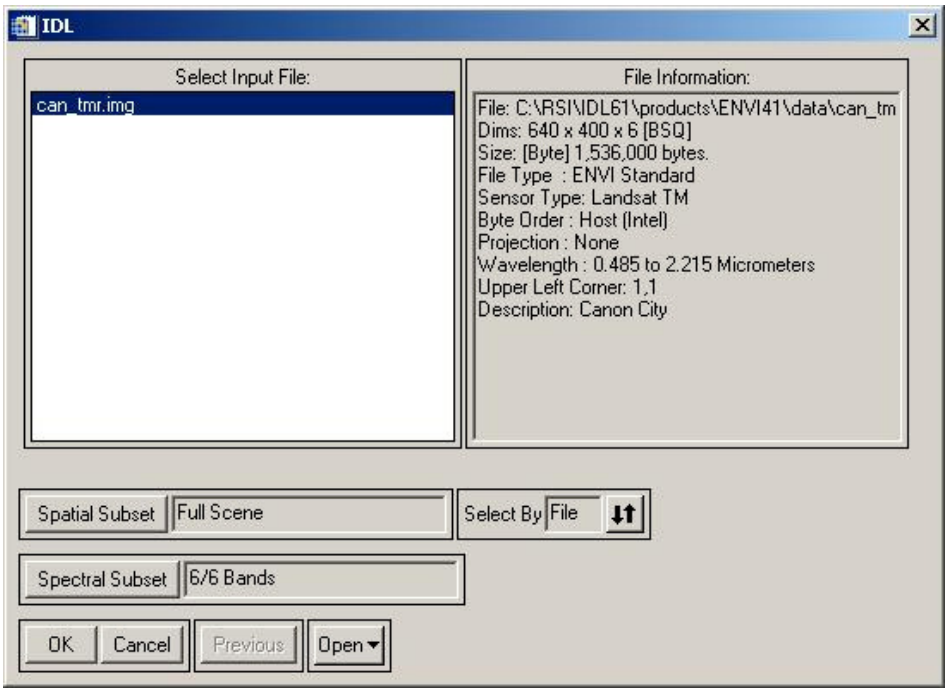
ENVI_FILE_QUERY [,PALAVRAS-CHAVE]

ENVI_DOIT

A rotina *ENVI_DOIT* é uma interface para todas as rotinas de processamento do ENVI (rotinas _DOIT). Cada _DOIT é apenas um argumento de texto do *ENVI_DOIT* com as palavras-chave após o _DOIT. Por exemplo, a chamada ENVI_CUBE_3D_DOIT é feita da seguinte forma: ENVI_DOIT, 'ENVI_CUBE_3D_DOIT'..... [palavras-chave do ENVI_CUBE_3D_DOIT].

Sintaxe

ENVI_DOIT, 'Nome da Rotina' [,PALAVRAS-CHAVE]



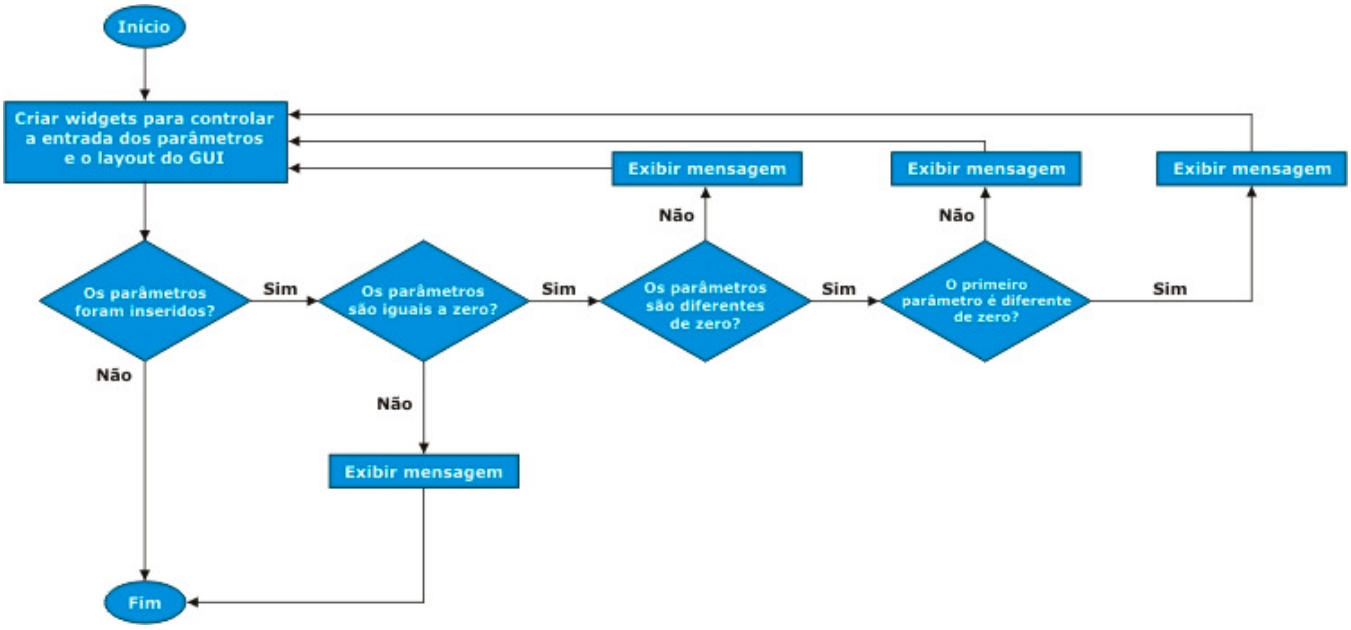
MATH_DOIT

Use esta rotina para executar operações matemáticas dos dados de uma imagem. No momento em que se utiliza operadores aritméticos em uma imagem que contenha bandas do tipo bit, deve se ter o conhecimento de que, se for feita uma soma de bandas do tipo bit (sem que uma função de conversão seja utilizada, como exemplo *FIX= integer*, *FLOAT= float*, *DOUBLE= double*), poderá ocorrer um overflow fazendo com que a expressão retorne um resultado diferente do que seria esperado.

Sintaxe

ENVI_DOIT, '*MATH_DOIT*' [,PALAVRAS-CHAVE]

Neste segundo exemplo temos outras rotinas do ENVI, criando uma janela de inserção de parâmetros e fazendo uma verificação se estes parâmetros são diferentes de 0 (zero).




```
1    pro Exemplo_2,ev
2
3       ;Instrução de Repetição (repeat) para controlar, quando será finalizado o procedimento.
4       repeat begin
5
6          ;Os comandos abaixo irão criar a janela de inserção de parâmetros.
7          base=widget_auto_base(title='Entre com os parâmetros')
8          base0=widget_base(base,/col)
9          base1=widget_base(base,/row)
10         base2=widget_base(base, /row)
11         base3=widget_base(base, /row, XSIZE=50,YSIZE=40)
12         label= widget_label(base0,value ='Parâmetros devem ser diferentes de 0', font="times*Bold" )
13         wparam_a=widget_param(base1,/auto,prompt='Parâmetro A:',uvalue='param_a', XSIZE=30)
14         wparam_b=widget_param(base2,/auto,prompt='Parâmetro B:',uvalue='param_b', XSIZE=30)
15         est= widget_label(base3,value =' ‘,  xsize = 74)
```

```
16          result = auto_wid_mng(base, column_base = base3)
17
18          ;Faz uma verificação vendo qual botão que foi clicado.
19          if (result.accept eq 1) then begin
20
21             ;Verifica se os parâmetros são diferentes de zero.
22             if (result.param_a ne 0 and result.param_b ne 0) then begin
23                texto = DIALOG_MESSAGE ("Os parâmetros digitados são diferentes de zero!" $
24                , /INFORMATION, TITLE="FIM DO EXEMPLO")
25                break
26             endif
27
28             ;Verifica se os dois parâmetros são iguais a zero.
29             if (result.param_a eq 0 && result.param_b eq 0) then begin
30                texto = DIALOG_MESSAGE ("Os dois parâmetros são iguais a zero, digite novamente" $
31                , /ERROR, TITLE="Os parâmetros devem ser diferentes de zero")
32                continue
33             Endif
34
35             ;Verifica se o parâmetro a é igual a zero.
36             If result.param_a eq 0  then begin
37                texto = DIALOG_MESSAGE ("O 1º parâmetro é zero, digite novamente",$
38                /ERROR, TITLE="Os parâmetros devem ser diferentes de zero")
39                continue
40             ;Verifica se o parâmetro b é igual a zero.
41             endif else begin
42                 texto = DIALOG_MESSAGE ("O 2º parâmetro é zero, digite novamente",$
43                 /ERROR, TITLE="Os parâmetros devem ser diferentes de zero")
44                 continue
45             endelse
46          endif
47       endrep until ((result.accept eq 0) || (result.param_a ne 0 && result.param_b ne 0))
48    end
```

Neste exemplo é utilizada algumas funções simples que trabalham com widgets, os valores retornados por estes widgets são sempre o widget ID do novo widget.
Na linha 7 do exemplo acima, a rotina *WIDGET_AUTO_BASE* cria o topo da janela de inserção de parâmetros.
Da linha 8 até a linha 11, é utilizado a rotina *WIDGET_BASE* para alterar o Layout da janela de inserção de parâmetros.
Na linha 12, temos a rotina *WIDGET_LABEL* que cria o texto que fica acima das caixas de entrada de parâmetros.
Na linha 13 e 14, é usada a rotina *WIDGET_PARAM* para exibir as caixas de texto onde serão inseridos os parâmetros.
Na linha 15, temos outro *WIDGET_LABEL* mas agora apenas alterando o Layout da janela, que será realmente chamada pela rotina *AUTO_WID_MNG* (linha 16), retornando para a variável *result* uma estrutura anônima.
Na linha 19, temos uma instrução de condição (IF), fazendo teste se o campo (*accept*) da estrutura (*result*) é igual a 1 (um), isto significa que os parâmetros foram digitados e foi clicado o botão OK.
Na linha 22, é feito um teste verificando se os campos (*param_a*, *param_b*), da estrutura (*result*) são diferentes de zero. Caso esta condição seja verdadeira, será enviada uma mensagem (linhas 23 e 24) dizendo que os parâmetros são diferentes de zero e será encerrado o exemplo.

Na linha 29, assim como nas linhas 36 e 41 é testado se os parâmetros inseridos são iguais a zero, se eles forem iguais a zero então deverá ser digitado um novo número. Para fazer o download deste exemplo acesse o link abaixo: http://www.sulsoft.com.br/idl/guiadireto/exemplos/Exemplo_2.pro.

WIDGET_AUTO_BASE

Normalmente quando se programa com widgets no IDL, todos as bases widget, incluindo a base de nível superior (ou TLB(Top Level Base)), são criados usando a função *WIDGET_BASE*. Entretanto, na programação com o ENVI, se você quiser criar hierarquias de widgets cujos eventos são automaticamente controlados pelo ENVI, você deve usar uma rotina especial do ENVI chamada *WIDGET_AUTO_BASE* para criar a TBL. Todas outras bases usadas na construção do GUI (interface gráfica ao usuário), deverão utilizar a função *WIDGET_BASE*.
As TBLs criadas usando o *WIDGET_AUTO_BASE* são automaticamente baseadas em colunas, centralizado, e modal (isto é, bloqueado). Ao contrário do *WIDGET_BASE*, que pode aceitar dezenas de palavras-chave diferentes que irão controlar a aparência da base, o *WIDGET_AUTO_BASE* aceita somente quatro palavras-chave: GROUP, TITLE, XBIG, YBIG. Estas palavras-chave permitem que o GUI possa ser prezo a um widget existente.

Sintaxe

resultado = *WIDGET_AUTO_BASE(*[,PALAVRAS-CHAVE])

WIDGET_BASE

A função *WIDGET_BASE* é usada para criar uma base de widget. Estas bases de widget servem como recipientes para outros widgets. Pode ser usado também com o objetivo de ser TLB, neste caso não devera ser acrescentado o argumento *base*(uma variável contendo o valor do widget ID da base "pai") na rotina.
As bases de widgets (criados com a rotina *WIDGET_BASE*) tem um papel especialmente importante na criação de aplicações de IDL baseadas em widgets, porque controlam seu comportamento e a maneira com que as aplicações e seus componentes são criados, processados, e destruídos.

Sintaxe

resultado = *WIDGET_BASE*(*base*[,PALAVRAS-CHAVE])

WIDGET_LABEL

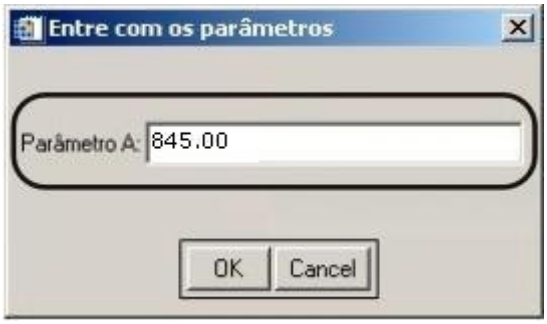
A função *WIDGET_LABEL* é usado para criar uma única linha de texto estático definido no momento em que a função é chamada. Se a palavra-chave VALUE não for inserida o texto exibido pela função será Label.

Sintaxe

resultado = *WIDGET_LABEL*(*base*[,PALAVRAS-CHAVE])

WIDGET_PARAM

A função *WIDGET_PARAM* é um widget usado para criar uma caixa de texto de entrada de parâmetros numéricos, como na imagem abaixo.



Se você estiver utilizando a função *AUTO_WID_MNG* no seu programa, você poderá utilizar a palavra-chave AUTO_MANAGE, para que você não precise criar um evento para controlar seu widget.

Sintaxe

resultado = *WIDGET_PARAM*(*base*[,PALAVRAS-CHAVE])

AUTO_WID_MNG

Use esta função para controlar automaticamente a manipulação de eventos compostos de widgets do ENVI, sem a necessidade de escrever um procedimento que faça esta manipulação. A função retorna uma estrutura anônima cujos nomes dos campos (tags) foram definidos pelo valor do usuário (uvalue) dos widgets que estão sendo controlados. A função *AUTO_WID_MNG* cria automaticamente os botões de OK e Cancel no widget a menos que a palavra-chave NO_BUTTONS seja "setada". Em todos os casos, se o botão OK for selecionado, o campo resul.accept (onde *result* é o nome da estrutura retornada pela função *AUTO_WID_MNG*) irá receber 1 (um). Por outro lado, se o botão Cancel for selecionado então result.accept receberá 0 (zero).

Sintaxe

resultado = *AUTO_WID_MNG* (*Base*[,PALAVRAS-CHAVE])

Para maiores informações sobre essas e outras rotinas do ENVI, você deve acessar os guias de programação do ENVI (refguide.pdf / progguide.pdf) que estão por padrão no diretório: \RSI\IDL6X\products\ENVI4X\help .