Agenda

What is IDL?
How to run IDL
Simple IDL Commands
Arrays in IDL
Data I/O
Basic Plotting
Built-in Analysis Functions
Mapping

Example
Scientific Computing Today

Traditional programming languages: Basic, Pascal, FORTRAN, C, C++, Java

Scripting languages: sh, csh, bash, Tcl, Perl

Numeric/graphics: IDL, Matlab, Octave

Symbolic: Maple, Mathematica, MathCad

Dataflow: LabView, Simulink, AVS, DX, OpenDX
What is IDL?

Interactive Data Language
What is IDL?

**Interactive Data Language**

Proprietary software distributed by Research Systems, Inc. of Boulder, CO now a division of Kodak.
What is IDL?

Interactive Data Language

Proprietary software distributed by Research Systems, Inc. of Boulder, CO now a division of Kodak.

Grew out of programs written for analysis of data from NASA missions such as Mariner and the International Ultraviolet Explorer.
What is IDL?

**Interactive Data Language**

Proprietary software distributed by Research Systems, Inc. of Boulder, CO now a division of Kodak.

Grew out of programs written for analysis of data from NASA missions such as Mariner and the International Ultraviolet Explorer.

Oriented toward use by scientists and engineers in the **analysis** and **visualization** of multi-dimensional data sets.
What is IDL?

Interactive Data Language

Proprietary software distributed by Research Systems, Inc. of Boulder, CO now a division of Kodak.

Grew out of programs written for analysis of data from NASA missions such as Mariner and the International Ultraviolet Explorer.

Oriented toward use by scientists and engineers in the analysis and visualization of multi-dimensional data sets.

Platform Independent: Unix, linux, Windows, Macintosh
What is IDL?

Interactive Data Language

Offers all the power, adaptability, and programmability of high level languages like FORTRAN, C, and C++. 
What is IDL?

Interactive Data Language

Offers all the **power**, **adaptability**, and **programmability** of high level languages like FORTRAN, C, and C++.

But it adds two capabilities which are essential for modern data analysis: **interactivity** and **graphics** display.
What is IDL?

- User-written .pro files
- User-written functions in C or Fortran
- Numerical and graphical .pro files
- User interface
  Core numerics and graphics
Ways to Run IDL

Interactively

Development Environment (DE)
Command Line

Batch mode
Ways to Run IDL

Batch mode

```
prompt> idl filename
```

where `filename` contains a list of IDL commands

commands are interpreted as if they were entered at the command line

No multiline statements such as

```
begin
...
end
```
Ways to Run IDL

Command Line

prompt> idl

IDL> command

IDL>

each command entered is interpreted and executed and the prompt is returned
Ways to Run IDL

Development Environment

\texttt{prompt> idlde}

or

select from desktop, panel, toolbar, or launcher
Graphical User Interface (DE)
Graphical User Interface (DE)
Graphical User Interface (DE)
Graphical User Interface (DE)
Graphical User Interface (DE)
Graphical User Interface (DE)
Graphical User Interface (DE)

IDL Development Environment

% Compiled module: PLT_PHASE.
% Compiled module: PLT_PHR.
% Compiled module: PLT_BOTH.
% Compiled module: CNV_MDHMS_SEC.
% Compiled module: CNVTIME.
% Compiled module: CNV_SEC_MDHMS.
IDL> a = 5.0
IDL> b = 1.01
IDL> c = a^2 * b

Name | Type      | Value  
-----|-----------|--------
A    | FLOAT     | 5      
B    | FLOAT     | 3.14159
C    | FLOAT     | 78.5398

Locals | Params | Commons | System

variables
Graphical User Interface (DE)
Using IDL as a calculator

IDL> a = 3 ; assignment statement
    ; semicolon is used for comments

IDL> print, a + 4 ; built-in print command
    7

IDL> print, a + 4. ; variables dynamically typed
    7.000000 ; take on highest precision

IDL> print, a * 1e-9 ; scientific notation
    3.000000e-09

Note: variable names are case insensitive (A = a)
## Data Types

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Byte</td>
<td>nB</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
<td>n</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>3</td>
<td>Longword</td>
<td>nL</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>4</td>
<td>Floating Point</td>
<td>n.n</td>
<td>32-bit (+10³⁸)</td>
</tr>
<tr>
<td>5</td>
<td>Double-Precision</td>
<td>n.nD</td>
<td>64-bit (+10³⁸)</td>
</tr>
<tr>
<td>6</td>
<td>Complex</td>
<td>COMPLEX(n.n,n.n)</td>
<td>real/imaginary pairs</td>
</tr>
<tr>
<td>7</td>
<td>String</td>
<td>'saaa'</td>
<td>0-32k in length</td>
</tr>
<tr>
<td>8</td>
<td>Struct</td>
<td></td>
<td>User-defined Structure</td>
</tr>
<tr>
<td>9</td>
<td>D-P Complex</td>
<td>DCOMPLEX(n.n,n.n)</td>
<td>real/imaginary pairs double-precision</td>
</tr>
<tr>
<td>10</td>
<td>Pointer</td>
<td></td>
<td>Pointer</td>
</tr>
<tr>
<td>11</td>
<td>ObjRef</td>
<td></td>
<td>Object Reference</td>
</tr>
<tr>
<td>12</td>
<td>Unsigned Int</td>
<td>nU</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>13</td>
<td>Unsigned Long</td>
<td>nUL</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>14</td>
<td>64-bit Long</td>
<td>LONG64(n)</td>
<td>64-bit signed longword</td>
</tr>
<tr>
<td>15</td>
<td>64-bit Unsigned Long</td>
<td>ULONG64(n)</td>
<td>64-bit unsigned longword</td>
</tr>
</tbody>
</table>
Unary and Compound Operators

IDL> a = 3 ; assignment statement

IDL> a += 5 ; C compound operators
IDL> a = a + 5 ; equivalent but rather tedious

IDL> print, a
8

IDL> a++ ; C unary operators
IDL> print, a
8

IDL> print, a-- ; careful...
8

IDL> print, a
7
Arrays in IDL

Array-oriented language

operations are performed on arrays

Any data type (but not mixed)

Dynamically sized (and resized)
Vectors in IDL

IDL> vec = [3, 5, 1, 4, 5] ; assignment statement

IDL> print, vec ; 5 element vector
            3      5      1      4      5
            ↑      ↑      ↑      ↑      ↑

IDL> vec[3] = 9 ; square brackets to access

IDL> print, vec[3] ; elements of array
            9
Matrices in IDL

IDL> mat = [[3, 5, 1, 4, 5],$  ; $ is line continuation
IDL>       [8, 3, 2, 9, 1]]

IDL> print, mat  ; 5-column 2-row array
    3  5  1  4  5
   8  3  2  9  1

IDL> help, mat  ; useful command...
MAT   INT     = Array[5, 2]

IDL> print, mat[3,1]  ; access elements of matrix
9

column  row
Matrices in IDL

IDL> print, mat[3,1] ; access elements of matrix
9

column  row

IDL> print, mat[5*1+3] ; equivalent...
9

column row format...

M by N
Matrices in IDL

<table>
<thead>
<tr>
<th>column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M by N</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

; n<sup>th</sup> row: n*M  m<sup>th</sup> column: m
1*5 + 3 = 8

mat[3,1] = mat[5*1+3]
Declaring Matrices

IDL> mat = [[3,5,1,4,5],[8,3,2,9,1]] ; explicitly
IDL> mat = fltarr(5,2) ; all zeros
IDL> arr = dblarr(8,5,2) ; other types

P by M by N array

arr[6,2,1] = arr[n*P*M + m*P + p]

40 + 16 + 6
Filling Up Arrays

IDL> darr = dblarr(8,5,2) ; all zeros
IDL> iarr = indgen(8,5,2) ; sequential

IDL> print, iarr

0  1  2  3  4  5  6  7
  8  9 10 11 12 13 14 15
 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31
 32 33 34 35 36 37 38 39
 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55
 56 57 58 59 60 61 62 63
 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79

also: bindgen(), lindgen(), findgen(), dindgen(),
cindgen(), dcindgen(), uindgen(), ulindgen(),
sindgen()
Accessing Array Elements

IDL> \text{mat} = [[3,5,1,4,5],[8,3,2,9,1]] \; ; \text{5 by 2}

IDL> \text{print, mat[3,*]} \; ; \text{print column m=3}

\begin{array}{c}
4 \\
9 \\
\end{array}

IDL> \text{print, mat[2:4,0]} \; ; \text{print columns 2-4}

\begin{array}{ccc}
1 & 4 & 5 \\
\end{array}

IDL> q = [3,2,0] \; ; \text{another intarr}

IDL> \text{print, mat[q,0]} \; ; \text{index with another}

\begin{array}{ccc}
4 & 1 & 3 \\
\end{array} 

; \text{array or indices...}
Searching Arrays

IDL> mat = [[3, 5, 1, 4, 5], [8, 3, 2, 9, 1]] ; 5 by 2

IDL> q = where(mat le 2, nq) ; find all elements in mat that are <= 2

IDL> print, q ; indices of the array
   2     7     9

IDL> print, mat[q] ; values of the array
   1     2     1

IDL> print, nq ; optional, but good to check...
   3

also: eq, ge, gt, lt
Array Size

IDL> mat = $\begin{array}{ccc}
3 & 5 & 1 \\
4 & 5 & 8 \\
1 & 2 & 9 \\
4 & 5 & 1 \\
\end{array}$ ; 5 by 2

IDL> print, n_elements(mat[0,*]) ; vector 2

IDL> print, n_elements(mat) ; total number for 10 ; multi-dimensional

IDL> print, size(mat) ; more information
\[
\begin{array}{ccc}
2 & 5 & 2 \\
\end{array}
\]

# dimensions 1\textsuperscript{st} 2\textsuperscript{nd} data type total elements

elements in x dimension
Array Calculations

IDL> n = 100 ; declare num

IDL> x = findgen(n+1)*2*!pi/n ; x = \([0,2\pi]\) in steps of \(2\pi/100\)

IDL> y = cos(x) ; element-by-element assignment

IDL> y = print, y[0:5] ; assignment

1.00000 0.998027 0.992115 0.982287 0.968583 0.951057

No looping necessary
cleaner looking code
faster execution

Note: \(!pi\) is a system variable

Also: system structures such as \(!p, !x, !y, etc.\)

e.g. \(!p.multi = [0, ncol, nrow, nz, order]\)
Array Calculations

Using C

```c
#define N 101
int n;
float pi=3.1415926;
float x[N], y[N];

dx = 2*pi/N;
for(n=0; n<N; n++) {
    x[n] = n*dx;
    y[n] = cos(x[n]);
}
```
Array Calculations

Using C

```c
#define N 101
int n;
float pi=3.1415926;
float x[N], y[N];

dx = 2*pi/N;
for (n=0; n<N; n++) {
    x[n] = n*dx;
    y[n] = cos(x[n]);
}
```

Using IDL (looping)

```idl
n = 101
dx = 2*!pi/n
for i=0, n-1 do begin
    x[n] = n*dx
    y[n] = cos(x[n])
endfor
end
```
Array Calculations

Using C

```c
#define N 101
int n;
float pi=3.1415926;
float x[N], y[N];

dx = 2*pi/N;
for(n=0;n<N;n++) {
    x[n] = n*dx;
    y[n] = cos(x[n]);
}
```

Using IDL (looping)

```idl
n = 101
dx = 2*!pi/n
for i=0,n-1 do begin
    x[n] = n*dx
    y[n] = cos(x[n])
endfor
end
```

Using IDL (array)

```idl
n = 101
x = findgen(n)*2*!pi/(n-1)
y = cos(x)
```
Array Calculations

Using C

#define N 101
int n;
float pi=3.1415926;
float x[N], y[N];

dx = 2*pi/N;
for (n=0;n<N;n++) {
x[n] = n*dx;
y[n] = cos(x[n]);
}

Using IDL (looping)

n = 101
dx = 2*pi/n
for i=0,n-1 do begin
    x[n] = n*dx
    y[n] = cos(x[n])
endfor
end

Using IDL (array)

n = 101
x = findgen(n)*2*pi/(n-1)
y = cos(x)

Matlab: looping

23

array

21

n times slower than C

Minicourse: Introduction to IDL
Executive Commands
(or Dot Commands)

file loop.pro

n = 101
dx = 2*!pi/n
for i=0,n-1 do begin
  x[n] = n*dx
  y[n] = cos(x[n])
endfor
end

to execute commands in loop.pro

IDL> .run loop.pro

Other executive commands: (can only be used at the IDL command line)
.compile, .edit,
.run, .go, .continue, .out, .return, .run, .skip, .step, .stepover, .trace
Reading/Writing Files in IDL

Read in data from file:

```
openr, readf
```

Interactively with the DE

Write data to a file:

```
openw, writef
```
Reading Data from a File

IDL> filename = 'ssn.dat'
IDL> nlines = 3072
IDL> record = {time:0.0, ssn:0.0}
IDL> data = replicate(record, nlines)
IDL> openr, ifile, filename, /get_lun
IDL> reafdf, ifile, data
IDL> free_lun, ifile
IDL> print, data[0:9].time

1749.05 1749.13 1749.21 1749.29 1749.38 1749.46
1749.54 1749.63 1749.71 1749.80

Note: must know number of lines ...
can do with spawn, 'wc -l ' + filename
or by looping while not eof(ifile) do begin
readf, ifile, record
data[i++] = record
endwhile
Reading Data from a File
**Minicourse: Introduction to IDL**

February 2005

---

IDL Development Environment

---

```
license/license.dat:/usr/local/src/license/*.lic

IDL> Import_Ascii
% Compiled module: ASCII_TEMPLATE.
% WIDGET_LABEL: Requested font does not exist: helvr14.
% WIDGET_TABLE: Requested font does not exist: courier*12.
```

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN_ASCII</td>
<td>STRUCT</td>
<td>{ &lt;Anonymous&gt; }</td>
</tr>
<tr>
<td>TIME</td>
<td>FLOAT</td>
<td>Array[3072]</td>
</tr>
<tr>
<td>SSN</td>
<td>FLOAT</td>
<td>Array[3072]</td>
</tr>
</tbody>
</table>

---

```
IDL> [ ]
```
Plotting Features in IDL

IDL has built-in 2D and 3D plotting capabilities.

IDL> plot, data.time, data.ssn
Plotting Features in IDL

Many options for making fancier plots.

IDL> `plot, xrange=[1980, 2005], xtitle='Year', ytitle='Monthly Observed Sun Spot Number'`
Plotting Features in IDL

IDL> sm_data = smooth(data.ssn, 6)

IDL> oplot, data.time, sm_data, color=200, thick=4

IDL> loadct, 39

IDL> oplot, data.time, data.ssn, color=30, thick=3

IDL> oplot, data.time, data.ssn, color=200, thick=4
Estimate the power spectrum (see Numerical Recipes)

IDL> \( H = \text{fft}(\text{data.ssn}) \); use the built-in \text{fft}

IDL> \( \text{nDat} = \text{n_elements(data.ssn)} \); number of data points

IDL> \( q = \text{indgen}(\text{nDat}/2-1)+1 \); negative frequencies

IDL> \( p = \text{nDat} - q \); positive frequencies

IDL> \( \text{PSD} = \text{fltarr}(\text{nDat}/2+1) \); declare PSD holder

IDL> \( \text{PSD}[0] = \text{abs}(H[0])^2 \); load the

IDL> \( \text{PSD}[q] = \text{abs}(H[q])^2 \times \text{abs}(H[p])^2 \);

IDL> \( \text{PSD}[\text{nDat}/2] = \text{abs}(H[\text{nDat}/2])^2 \);

IDL> \( \text{per} = \text{nDat} \times 1./12./\text{findgen}(\text{nDat}/2+1) \); in years

IDL> \text{plot, per, PSD, xrange=[0,20],/nodata, xtitle='Period (years)'}

IDL> \text{oplot, per, PSD, color=250, thick=2}; plot PSD
Discrete Math Functions

Correlation Analysis
Curve and Surface Fitting
Differentiation and Integration
Eigenvalues and Eigenvectors
Gridding and Interpolation
Linear Systems including LAPACK Routines
Multivariate Analysis
Nonlinear Equations
Optimization
Statistics and Probability
More Complicated Plots

October 1993 — June 2004

20% — 46022
15% — 21770
10% — 6830
5% — 1052

Possible Periods (%)
Mapping Features in IDL

IDL has built-in mapping capabilities

IDL> map_set, /orthographic,/grid, /continent,$
IDL> /noborder

orthographic map projection
basic view
Mapping Features in IDL

IDL> map_set, 70, -75, /orthographic,/grid, $
IDL> /continent,/noborder

orthographic map projection
rotated view
Mapping Features in IDL

IDL> map_set, 70, -75, /orthographic,/grid, $
IDL> /continent,/noborder, $
IDL> e_continents={fill:1, color:150}, $
IDL> e_horizon={fill:1, color:50}

orthographic map projection
rotated view
colored
Map Projections in IDL

- Stereographic
- Gnomonic
- Azimuthal
- Lambert's
- Hammer-Aitoff
- Sinusoidal
Data on Maps
Further Topics to Explore

Writing IDL Programs

Calling external functions (C and FORTRAN) from IDL

IDL Widgets

Plotting 2D and 3D

Object Graphics

Debugging

Data Structures
How to Get IDL Help

Online:

`prompt> idlhelp`

Help menu in the DE

Internet:

`http://www.dfanning.com/documents/tips.html`

Listserve:

IDL users at Dartmouth

IDL-USERS@LISTSERV.DARTMOUTH.EDU

Books:
Where is IDL at Dartmouth?

Download onto your own computer: ~25 licenses