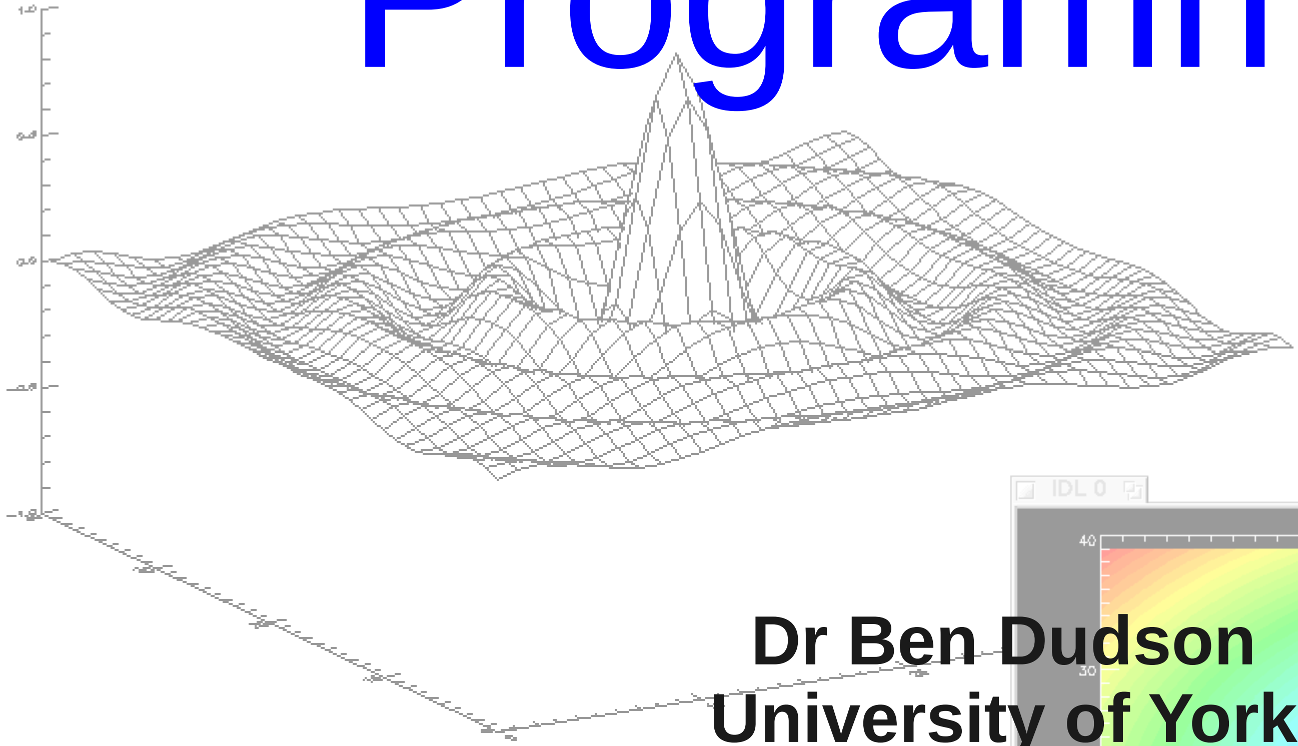
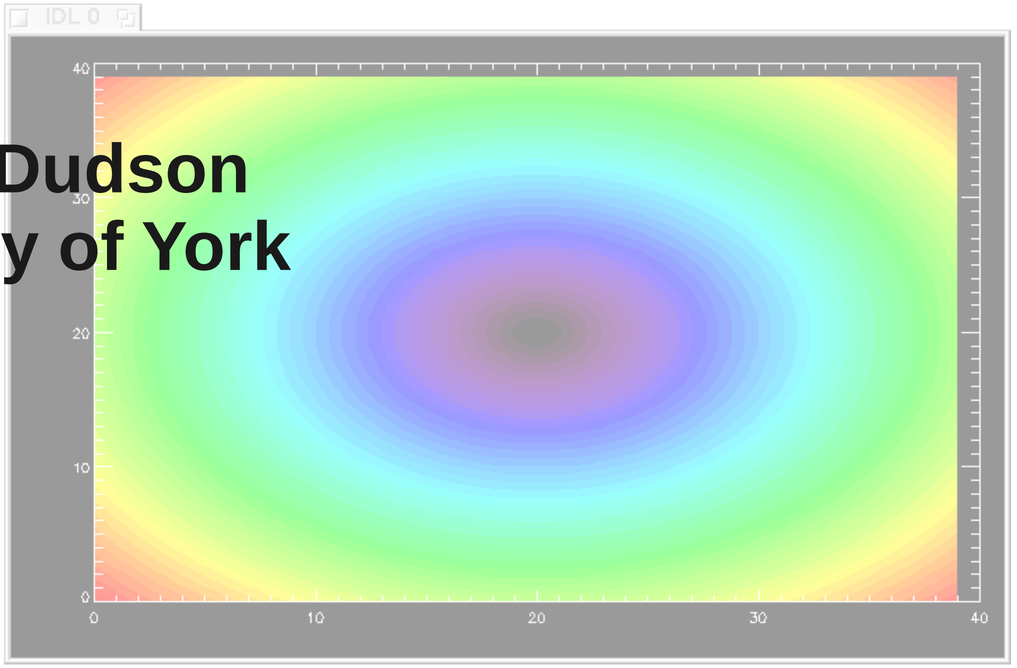
Programming



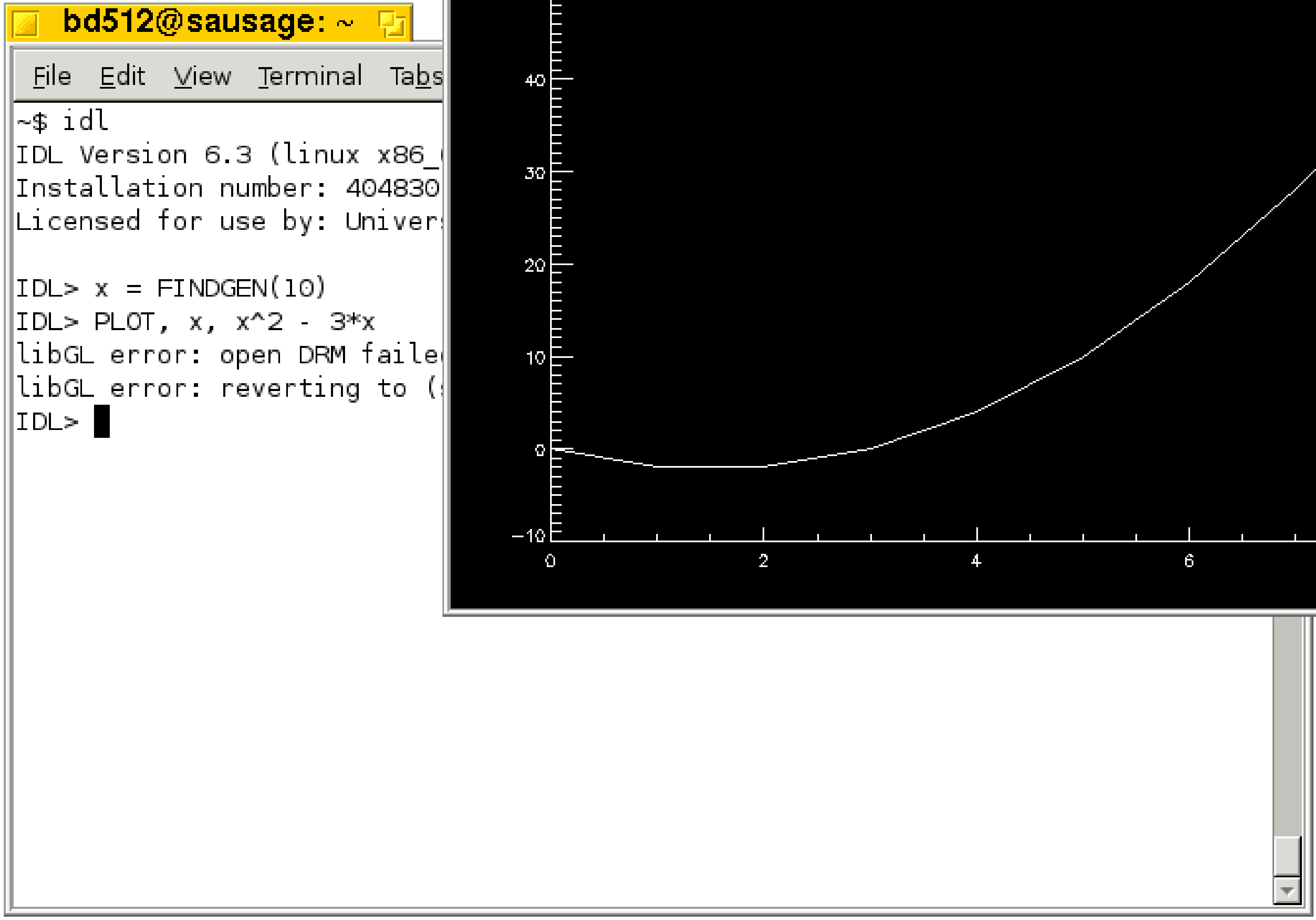


Dr Ben Dudson
University of York

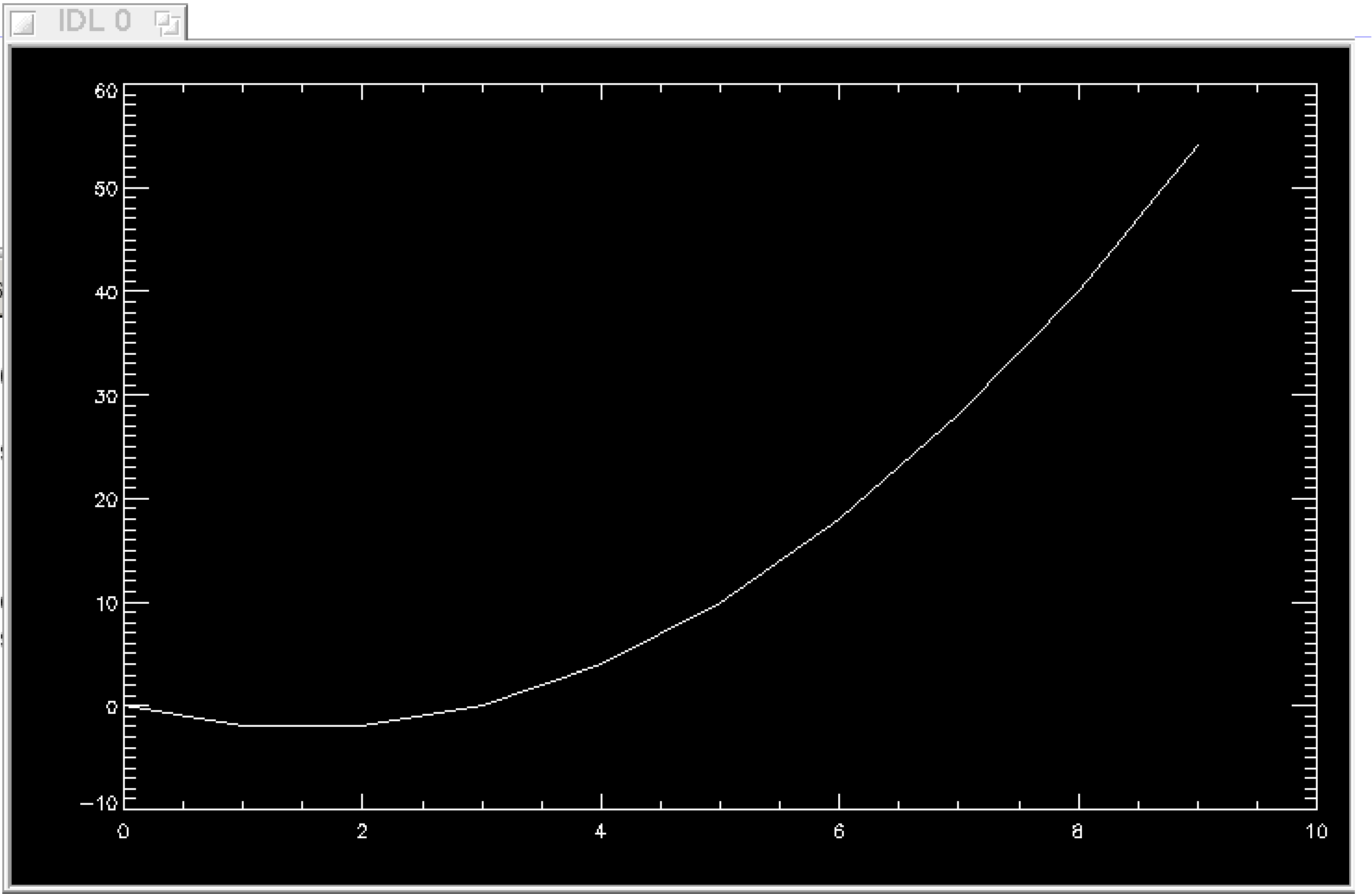
Outline

- Last lecture covered the basics of programming and IDL
- This lecture will cover
  - More advanced IDL and plotting
  - Fortran and C++
  - Programming techniques
- After this you can work on the IDL exercises, and get started with Fortran or C/C++

Plotting data



```
~$ idl
IDL Version 6.3 (linux x86_(
Installation number: 404830
Licensed for use by: Univers

IDL> x = FINDGEN(10)
IDL> PLOT, x, x^2 - 3*x
libGL error: open DRM failed
libGL error: reverting to (s
IDL>
```



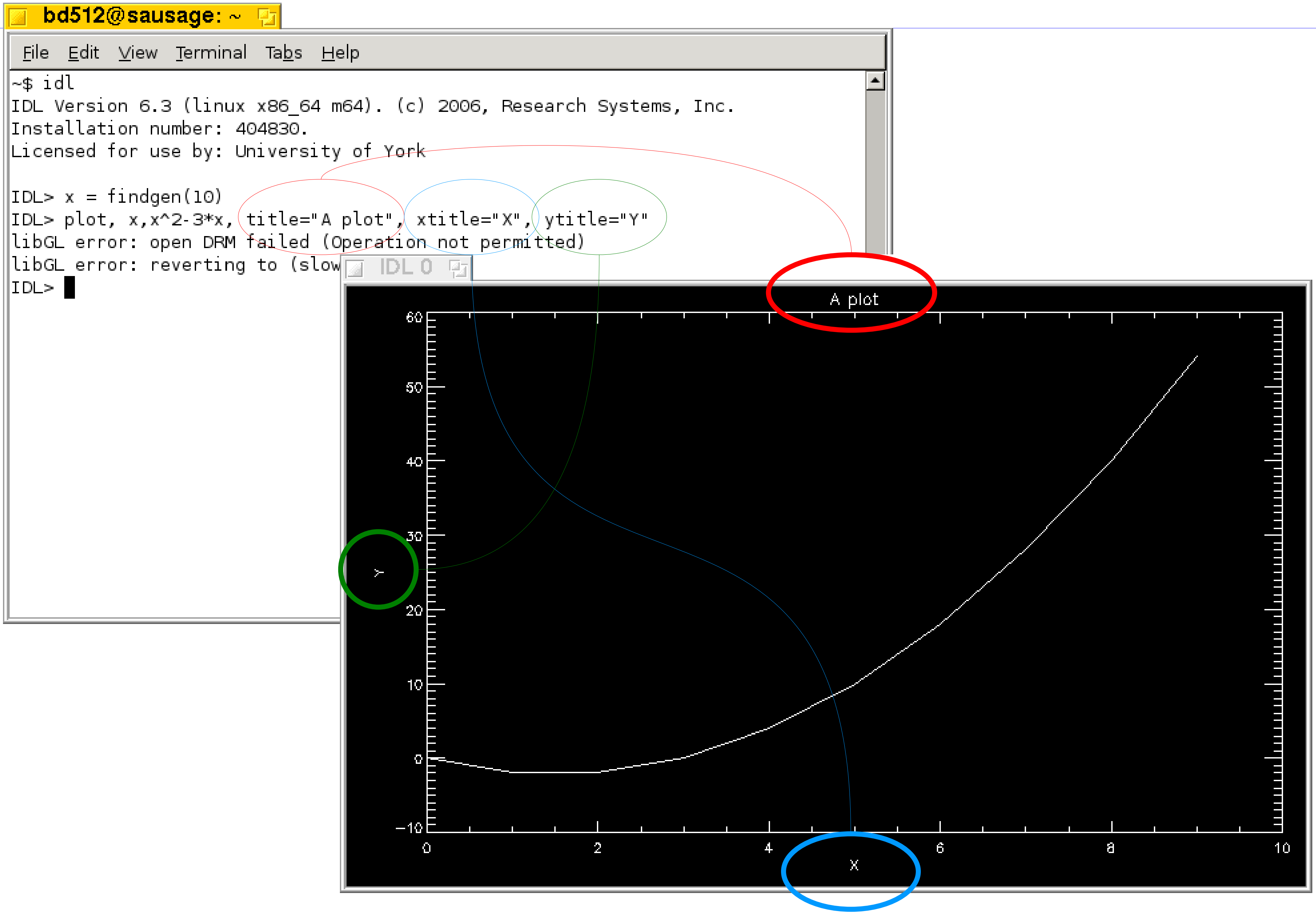
Optional arguments

- IDL commands usually have some mandatory arguments
- Often also have many optional parameters, called keywords

- Allows a command to have a “simple” form, which can be made more complicated as needed

- An example is the PLOT command, which has lots of keywords...

Adding titles



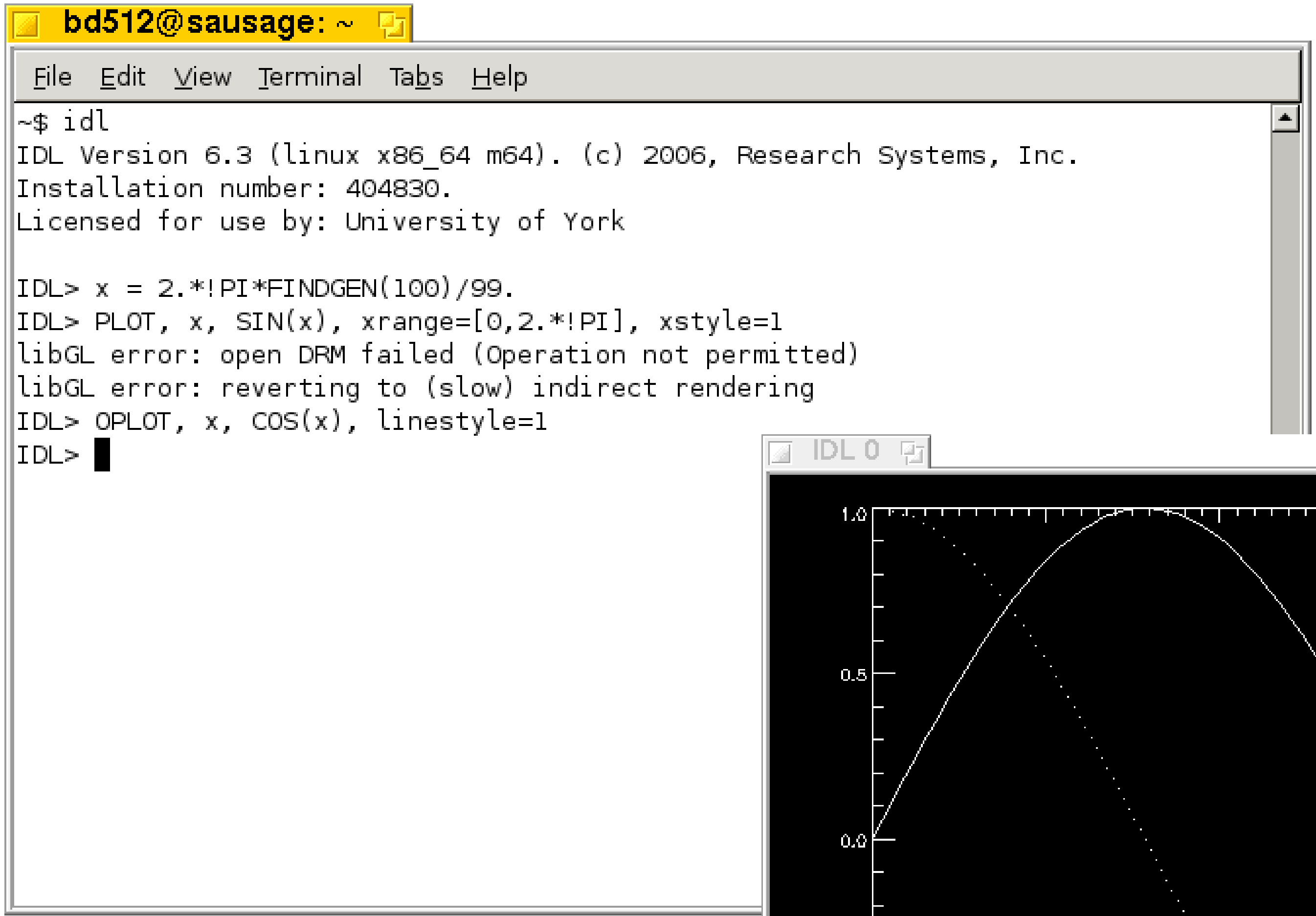
```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Research Systems, Inc.
Installation number: 404830.
Licensed for use by: University of York

IDL> x = findgen(10)
IDL> plot, x,x^2-3*x, title="A plot", xtitle="X", ytitle="Y"
libGL error: open DRM failed (Operation not permitted)
libGL error: reverting to (slow
IDL>
```

Plot options

- Plot has many different options, but some of the most useful are:

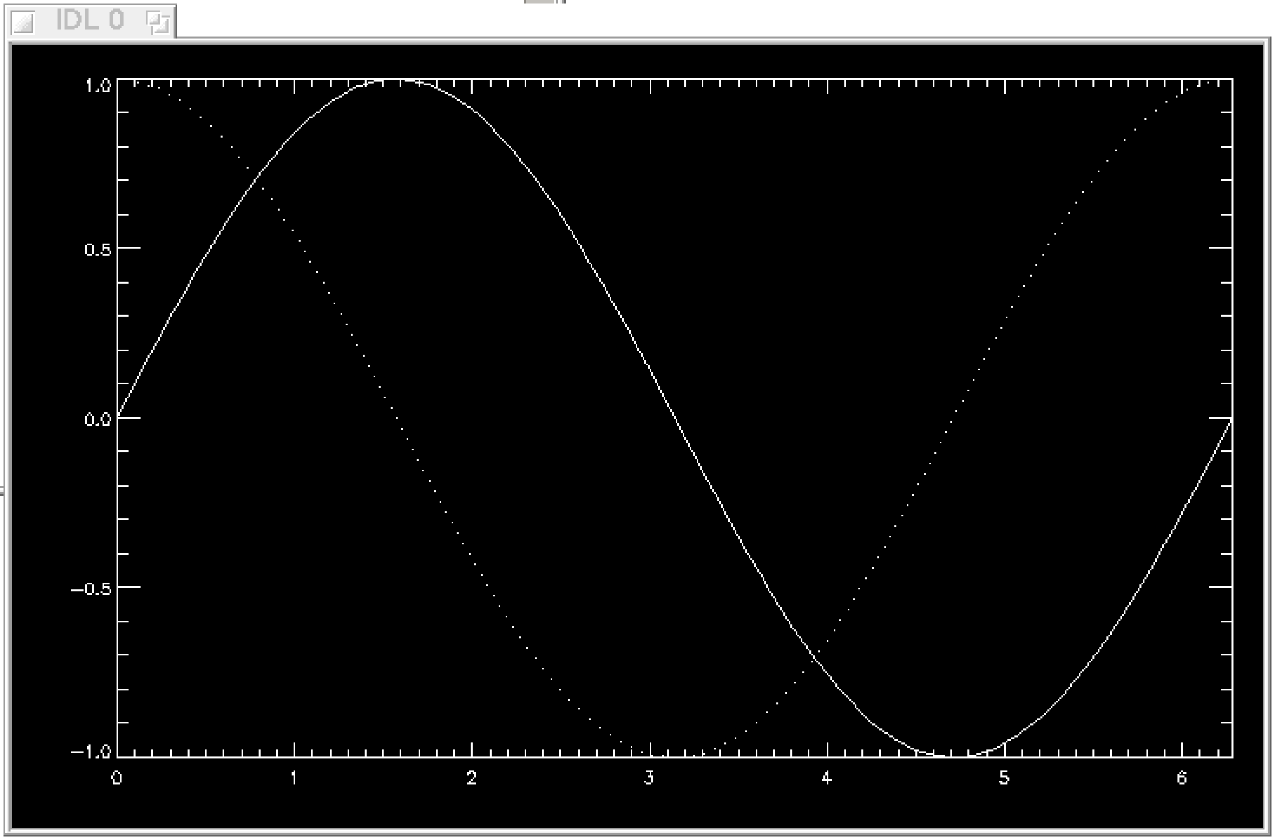
Title=”Some title”	Set plot title
Xtitle=”X axis label”	
Ytitle=”Y axis label”	
Linestyle=<number>	Change the line style (solid, dashed, dotted etc.)
Psym=<number>	Plot symbols not lines
Charsize=<number>	Font size for labels
Xrange=[min, max]	X-axis range
Yrange=[min, max]	Y-axis range

OPLOT

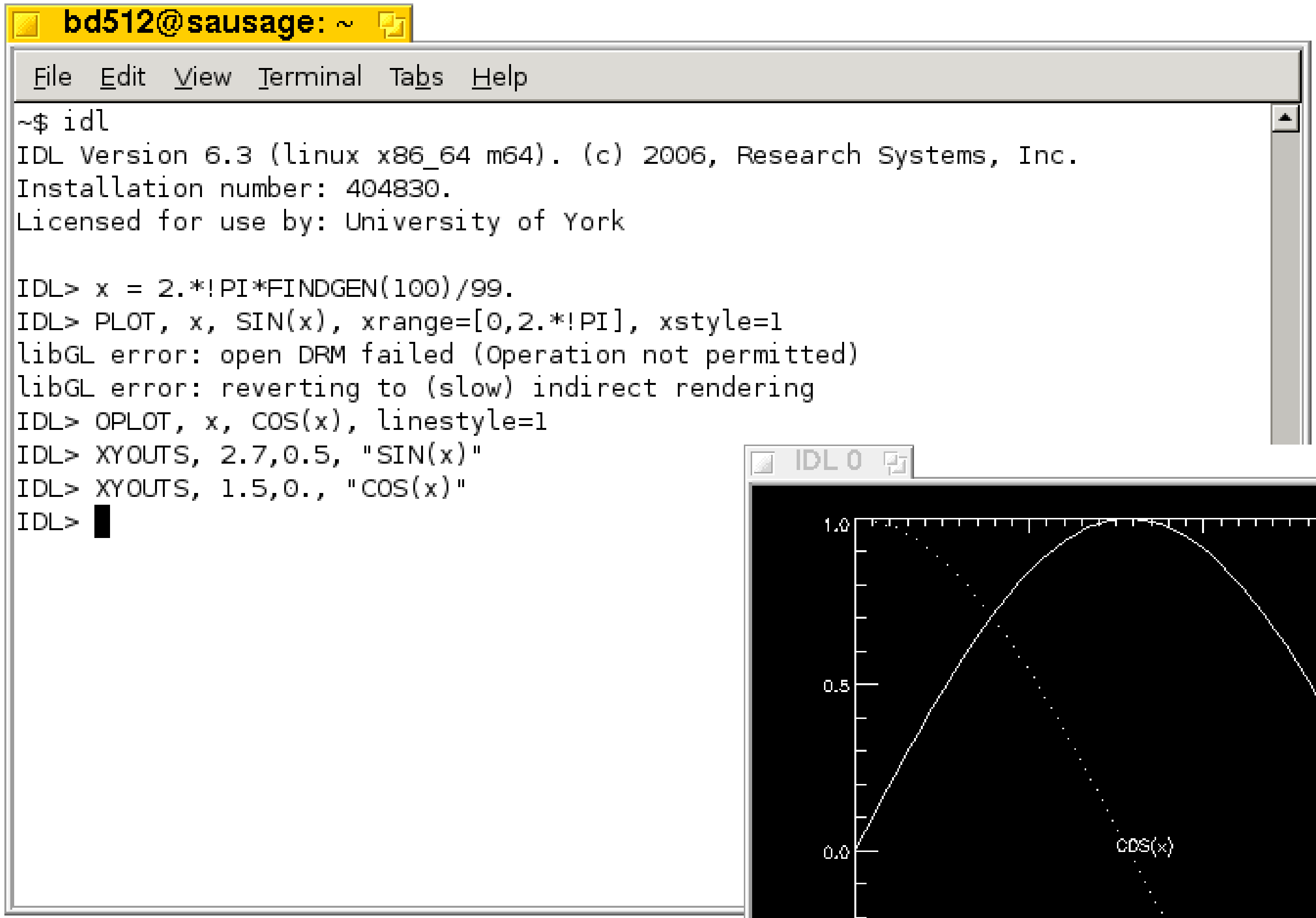


```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Research Systems, Inc.
Installation number: 404830.
Licensed for use by: University of York

IDL> x = 2.*!PI*FINDGEN(100)/99.
IDL> PLOT, x, SIN(x), xrange=[0,2.*!PI], xstyle=1
libGL error: open DRM failed (Operation not permitted)
libGL error: reverting to (slow) indirect rendering
IDL> OPLOT, x, COS(x), linestyle=1
IDL> 
```

Overlays a plot on top of the existing one.
Doesn't change the axes

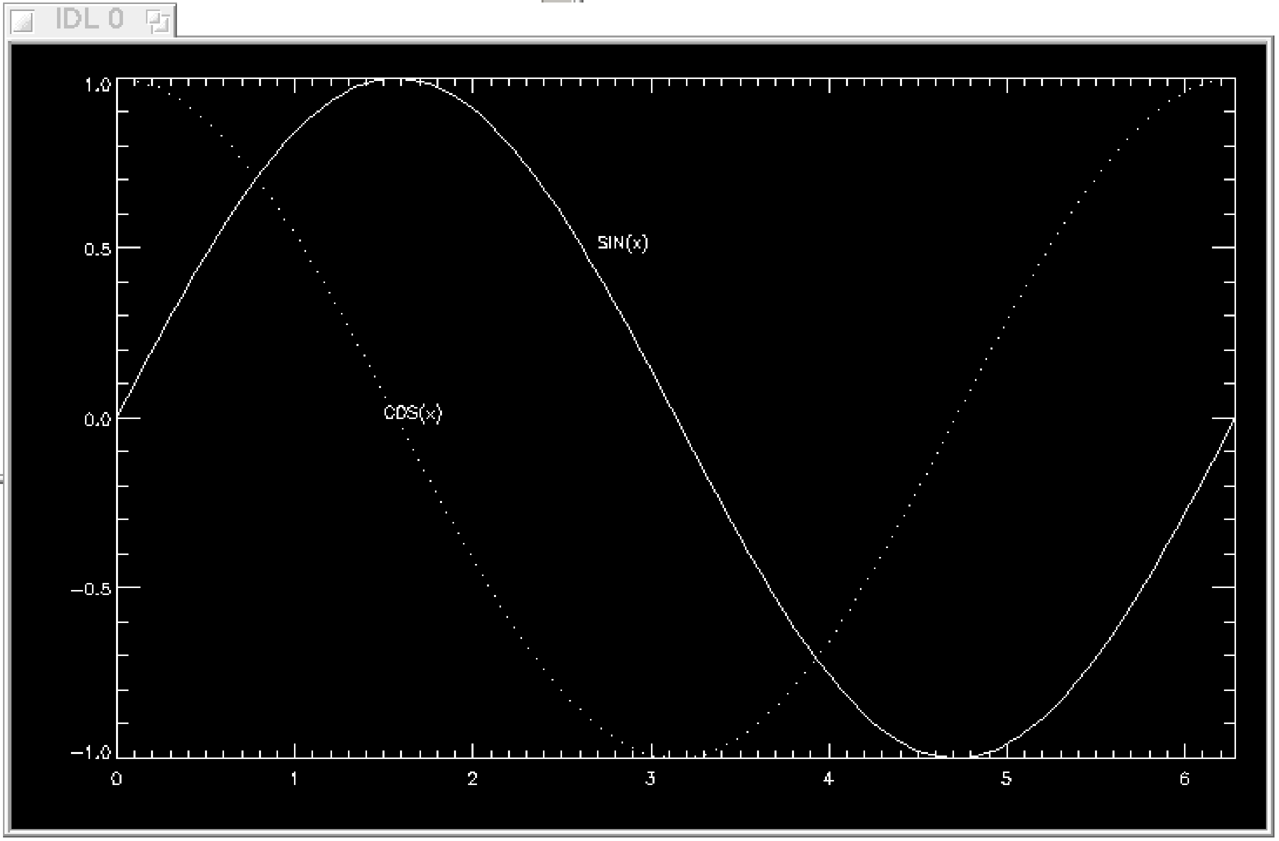


XYOUTS



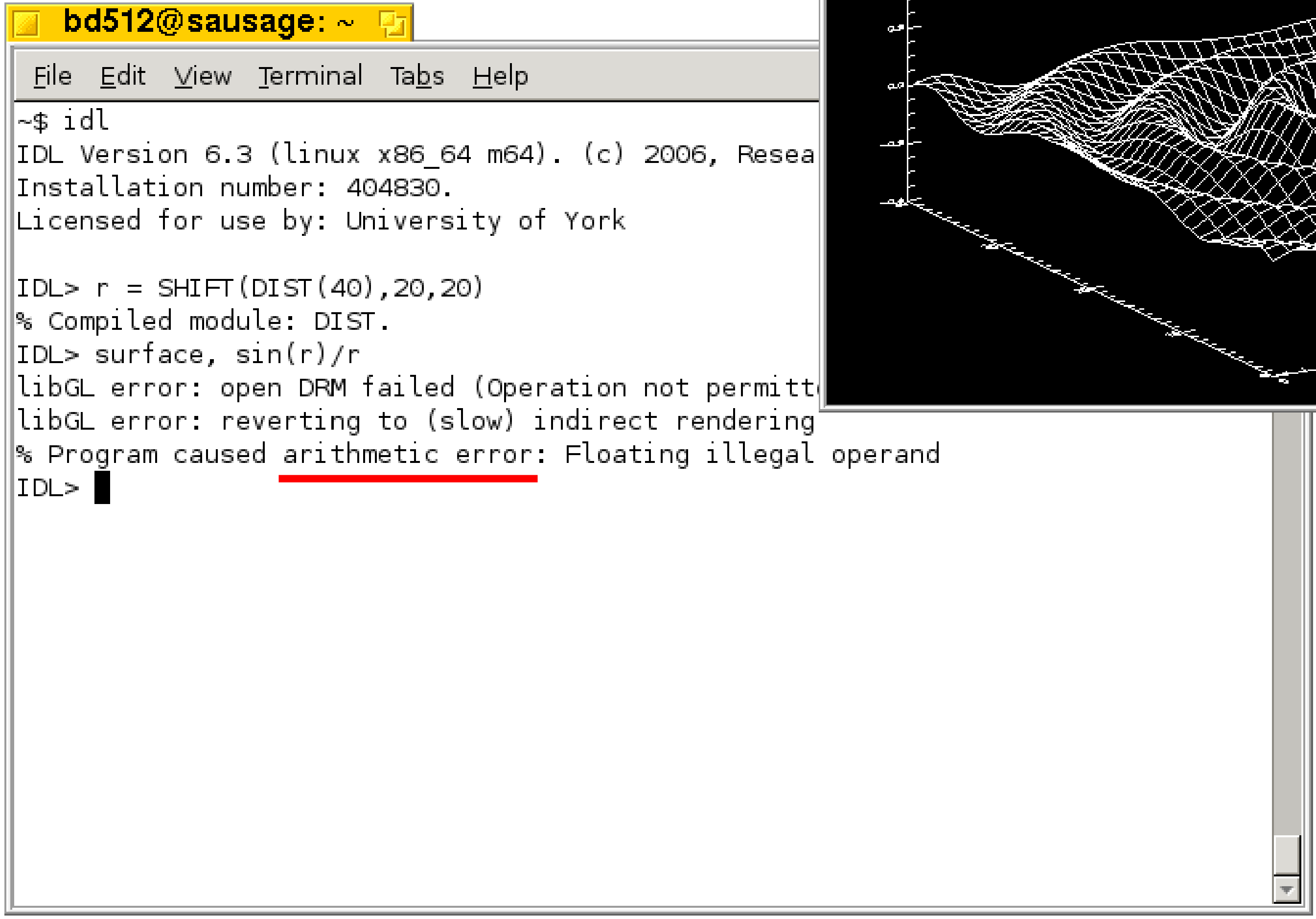
```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Research Systems, Inc.
Installation number: 404830.
Licensed for use by: University of York

IDL> x = 2.*!PI*FINDGEN(100)/99.
IDL> PLOT, x, SIN(x), xrange=[0,2.*!PI], xstyle=1
libGL error: open DRM failed (Operation not permitted)
libGL error: reverting to (slow) indirect rendering
IDL> OPLOT, x, COS(x), linestyle=1
IDL> XYOUTS, 2.7,0.5, "SIN(x)"
IDL> XYOUTS, 1.5,0., "COS(x)"
IDL>
```

Adds labels to a graph

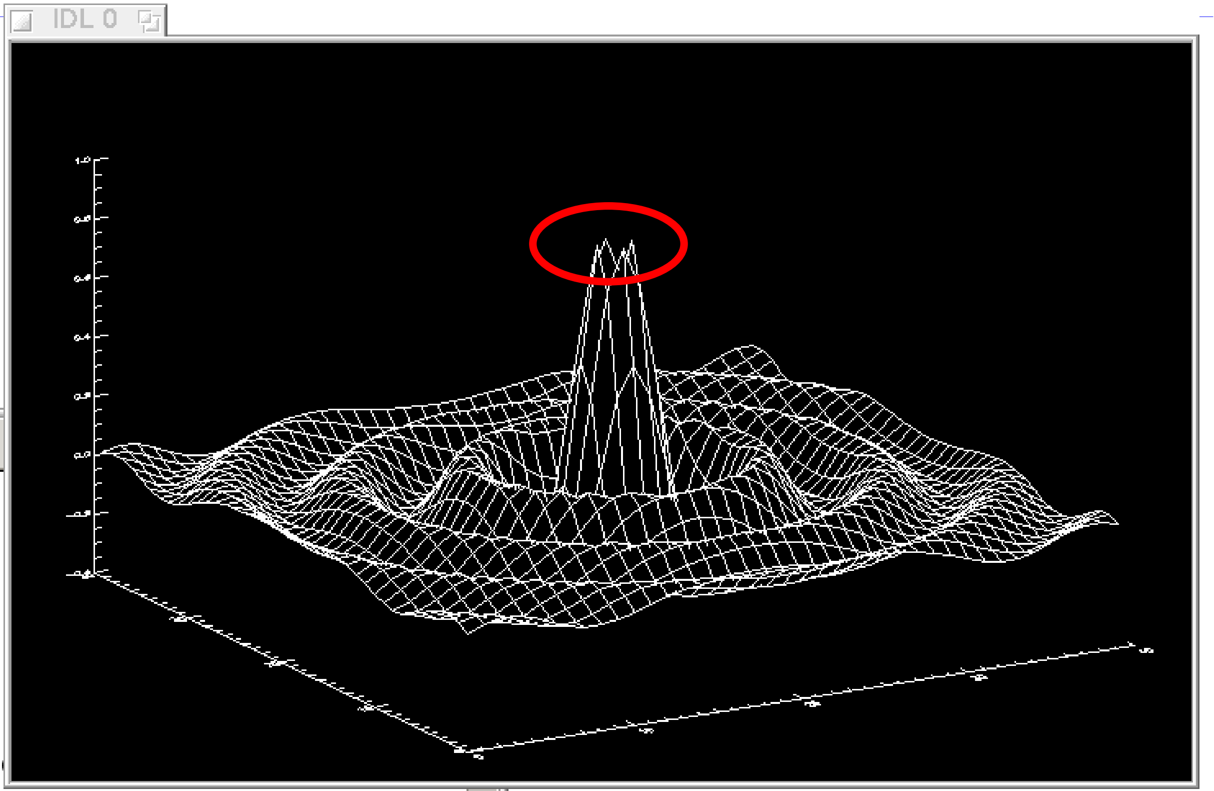


2D surface plots

SURFACE takes a 2D array, in this case a Sinc function

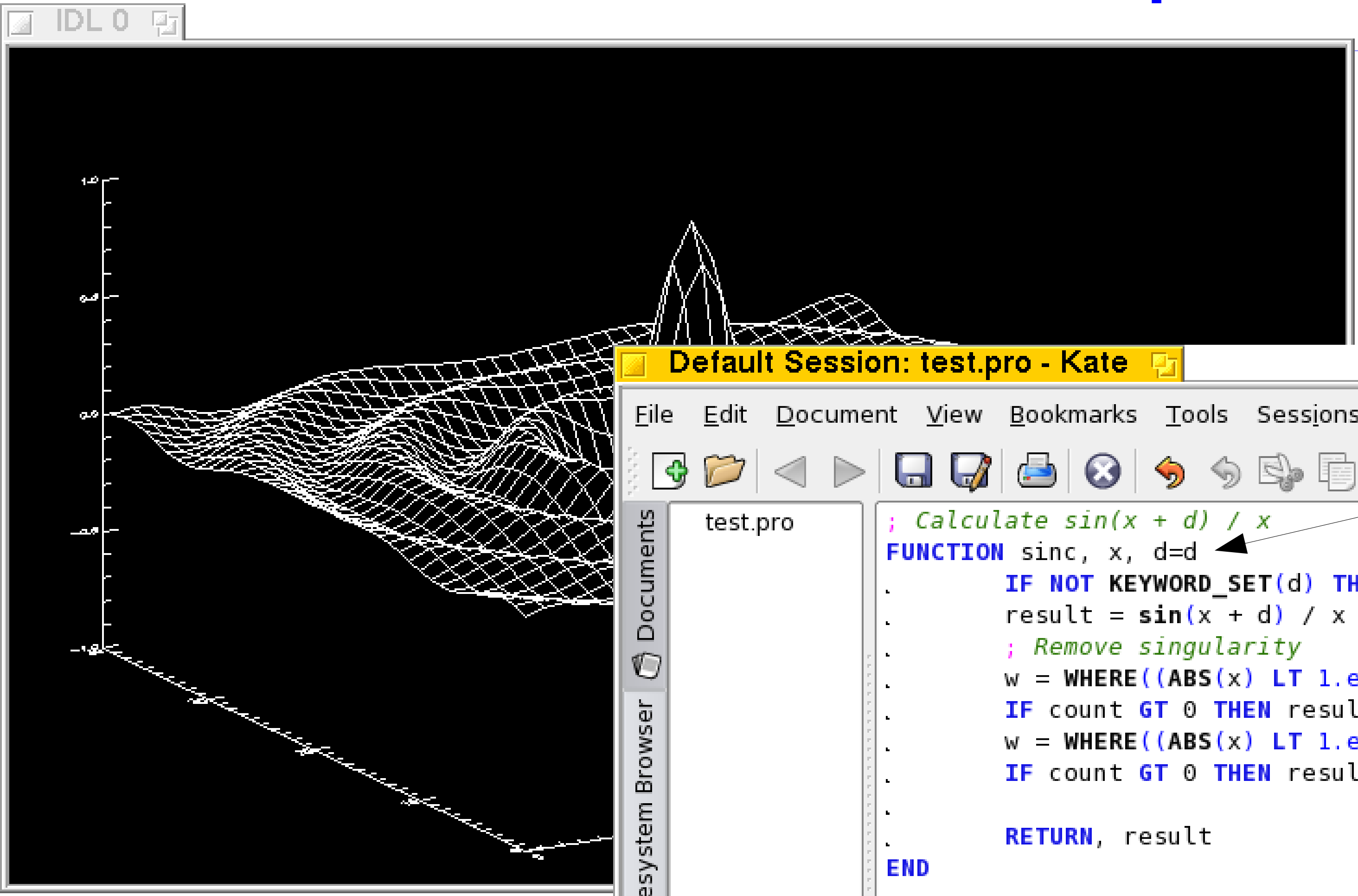


```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Resea
Installation number: 404830.
Licensed for use by: University of York

IDL> r = SHIFT(DIST(40),20,20)
% Compiled module: DIST.
IDL> surface, sin(r)/r
libGL error: open DRM failed (Operation not permitt
libGL error: reverting to (slow) indirect rendering
% Program caused arithmetic error: Floating illegal operand
IDL>
```

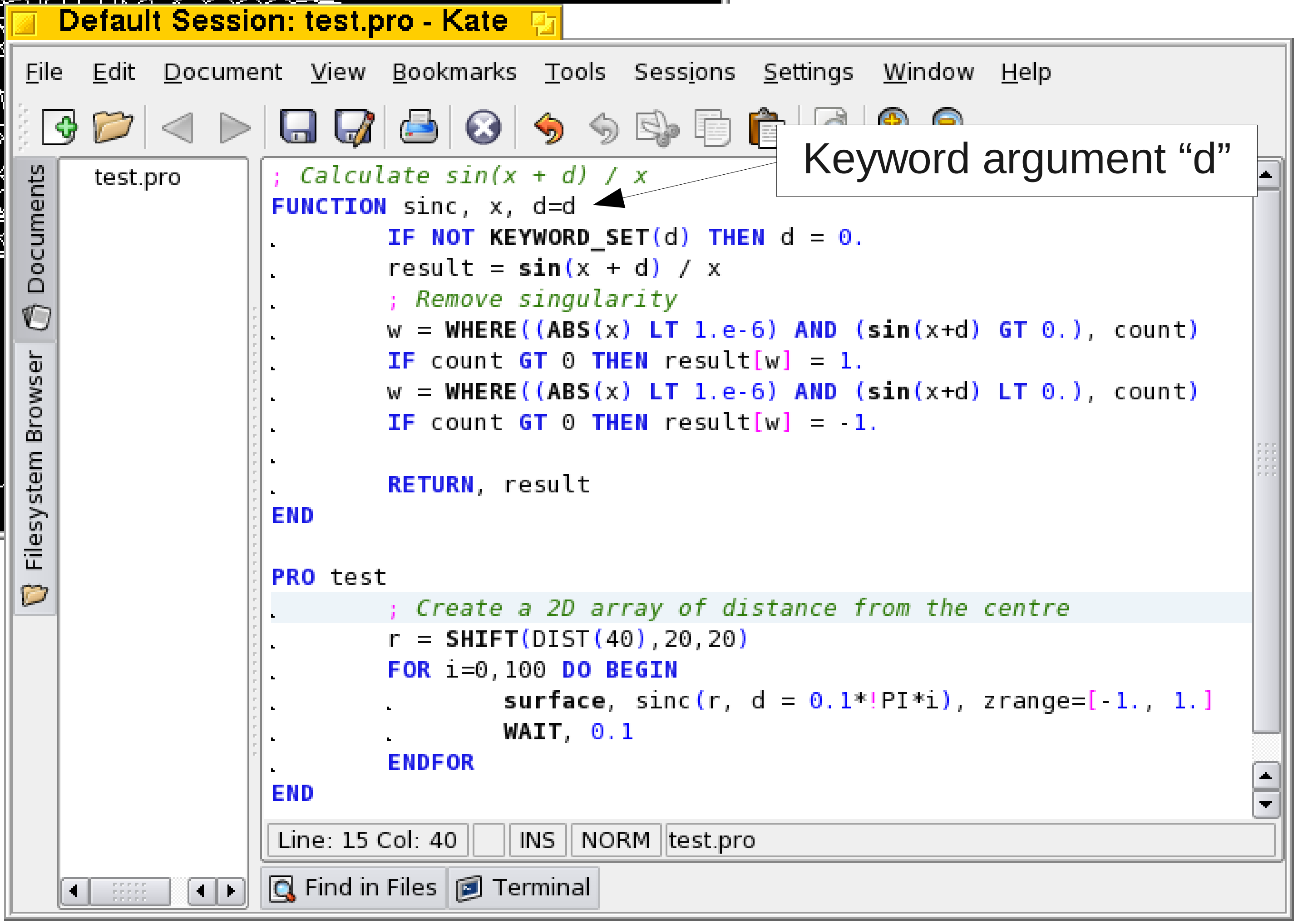


Arithmetic error caused by divide-by-zero in the centre of the plot

2D surface plots



Animation of moving sinc function
Removed singularity


Default Session: test.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
test.pro
Line: 15 Col: 40
INS
NORM
test.pro
Find in Files
Terminal

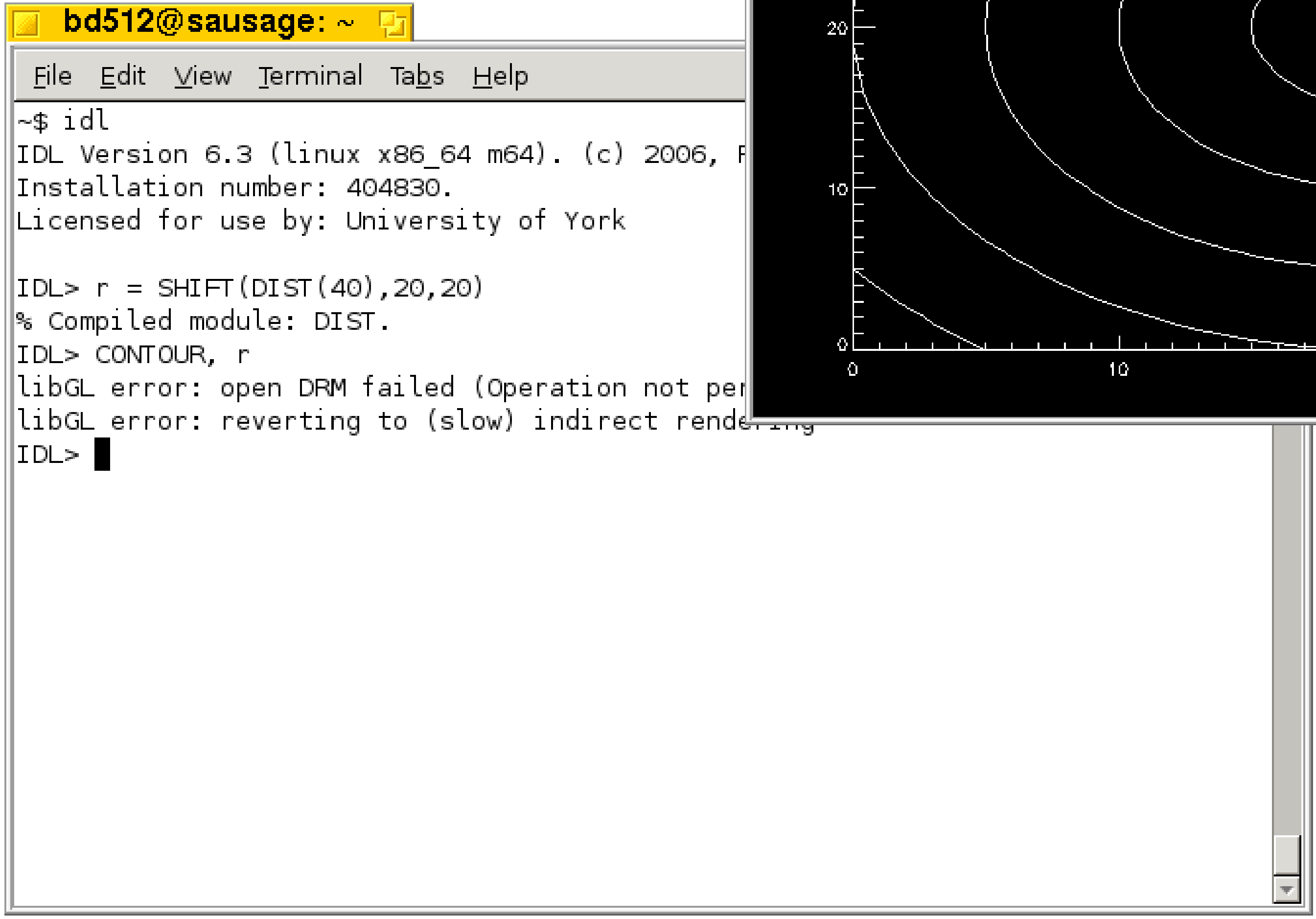
```
; Calculate sin(x + d) / x
FUNCTION sinc, x, d=d
        IF NOT KEYWORD_SET(d) THEN d = 0.
        result = sin(x + d) / x
        ; Remove singularity
        w = WHERE((ABS(x) LT 1.e-6) AND (sin(x+d) GT 0.), count)
        IF count GT 0 THEN result[w] = 1.
        w = WHERE((ABS(x) LT 1.e-6) AND (sin(x+d) LT 0.), count)
        IF count GT 0 THEN result[w] = -1.

        RETURN, result
END

PRO test
        ; Create a 2D array of distance from the centre
        r = SHIFT(DIST(40),20,20)
        FOR i=0,100 DO BEGIN
                surface, sinc(r, d = 0.1*!PI*i), zrange=[-1., 1.]
                WAIT, 0.1
        ENDFOR
END
```

Keyword argument “d”

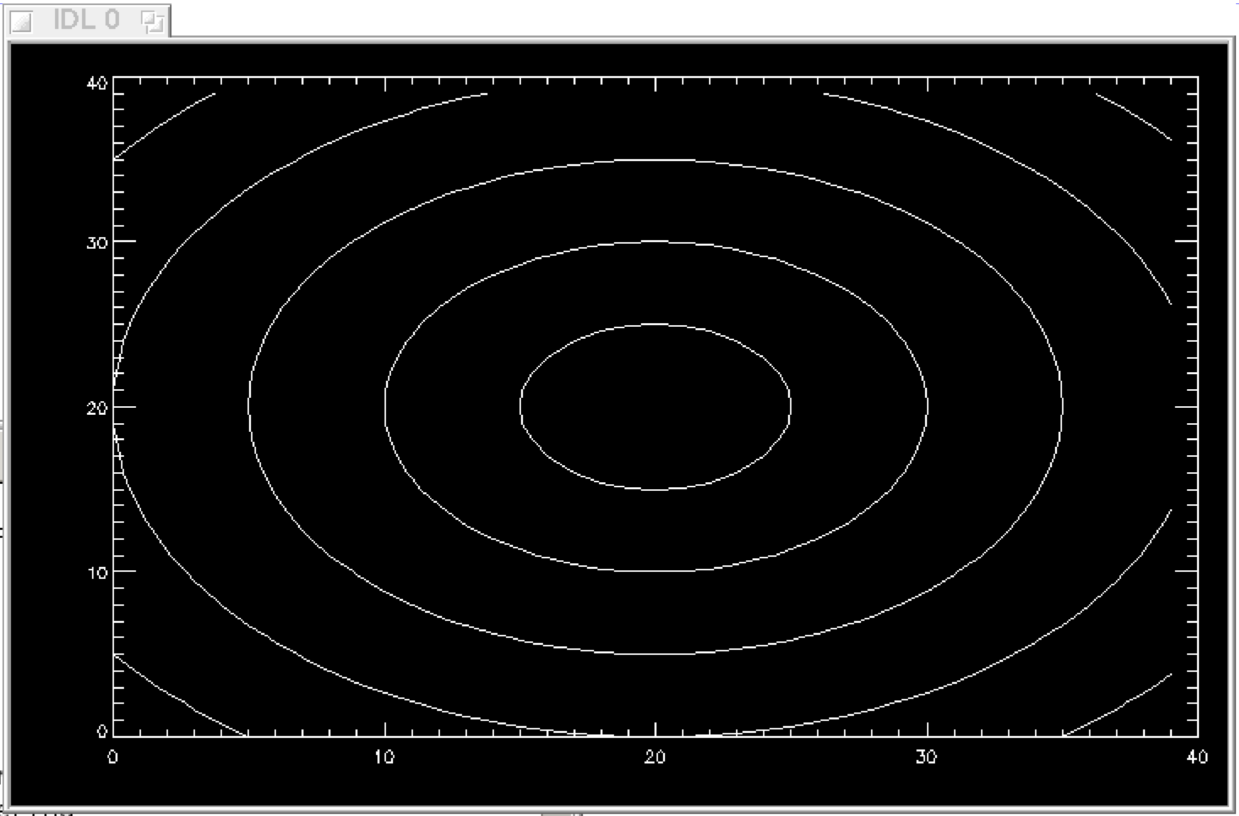
ZRANGE keyword to SURFACE forces the range to be the same for all plots

2D contour plots

Simplest way to use contour

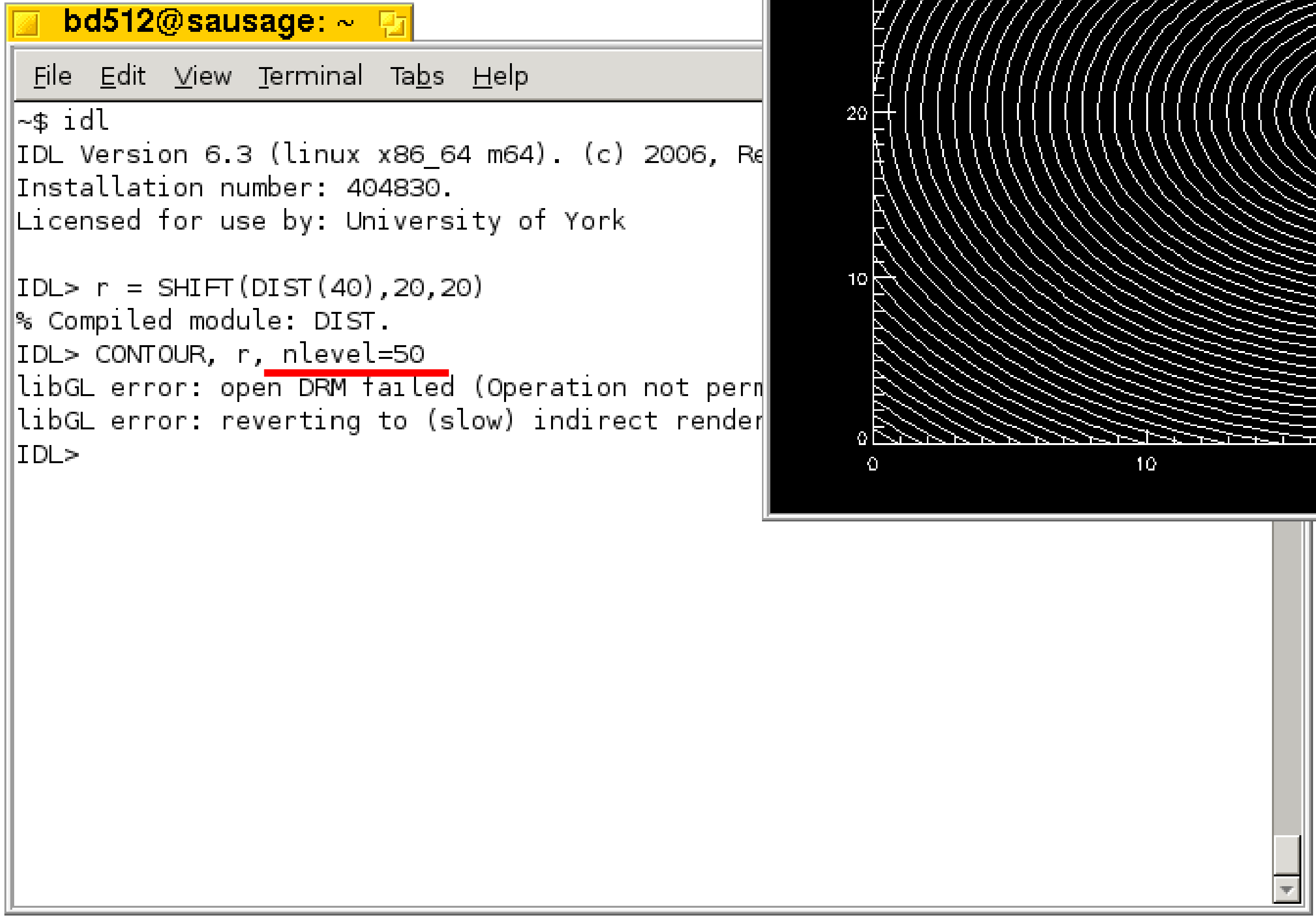


```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, R
Installation number: 404830.
Licensed for use by: University of York

IDL> r = SHIFT(DIST(40),20,20)
% Compiled module: DIST.
IDL> CONTOUR, r
libGL error: open DRM failed (Operation not per
libGL error: reverting to (slow) indirect rendering
IDL>
```

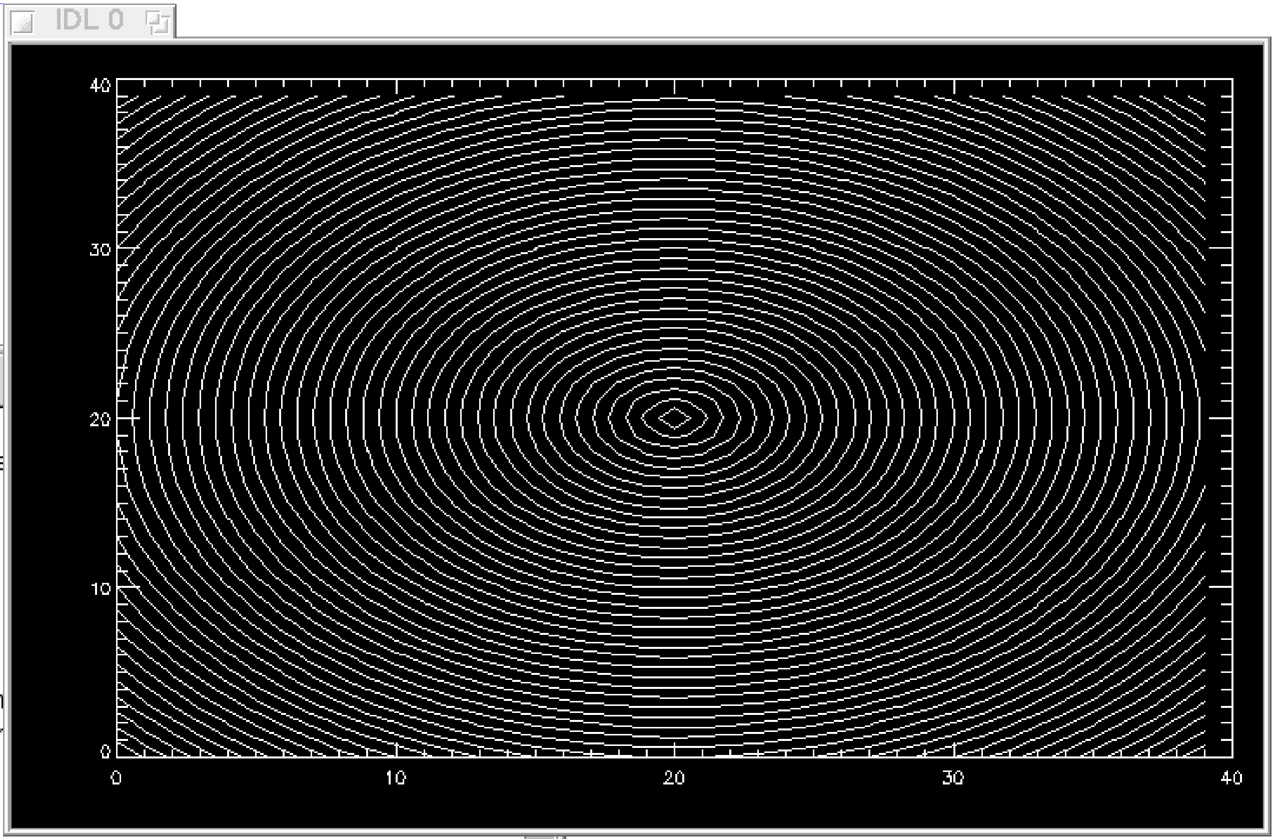


2D contour plots

Nlevel keyword sets the number of levels

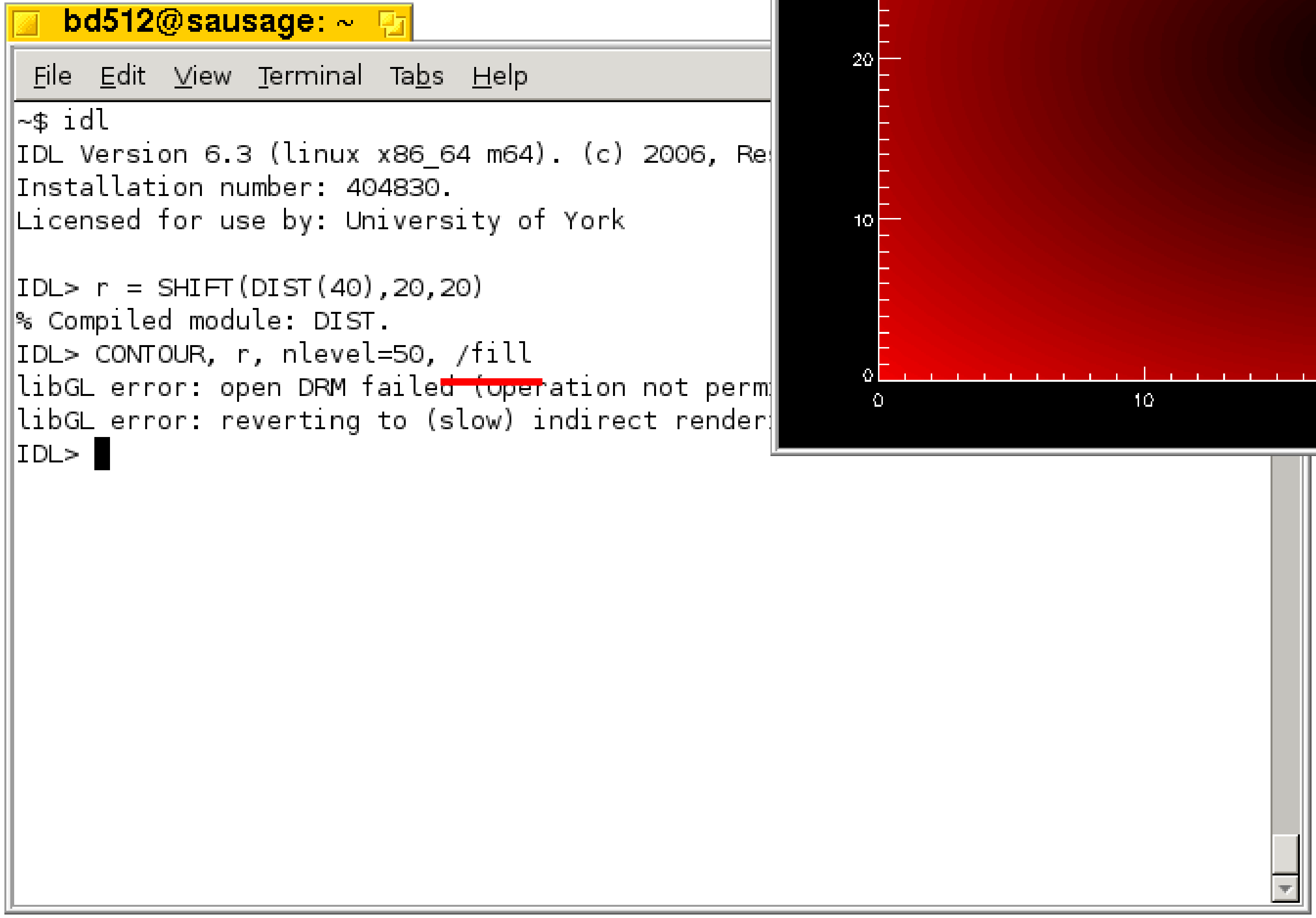


```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Re
Installation number: 404830.
Licensed for use by: University of York

IDL> r = SHIFT(DIST(40),20,20)
% Compiled module: DIST.
IDL> CONTOUR, r, nlevel=50
libGL error: open DRM failed (Operation not perm
libGL error: reverting to (slow) indirect render
IDL>
```

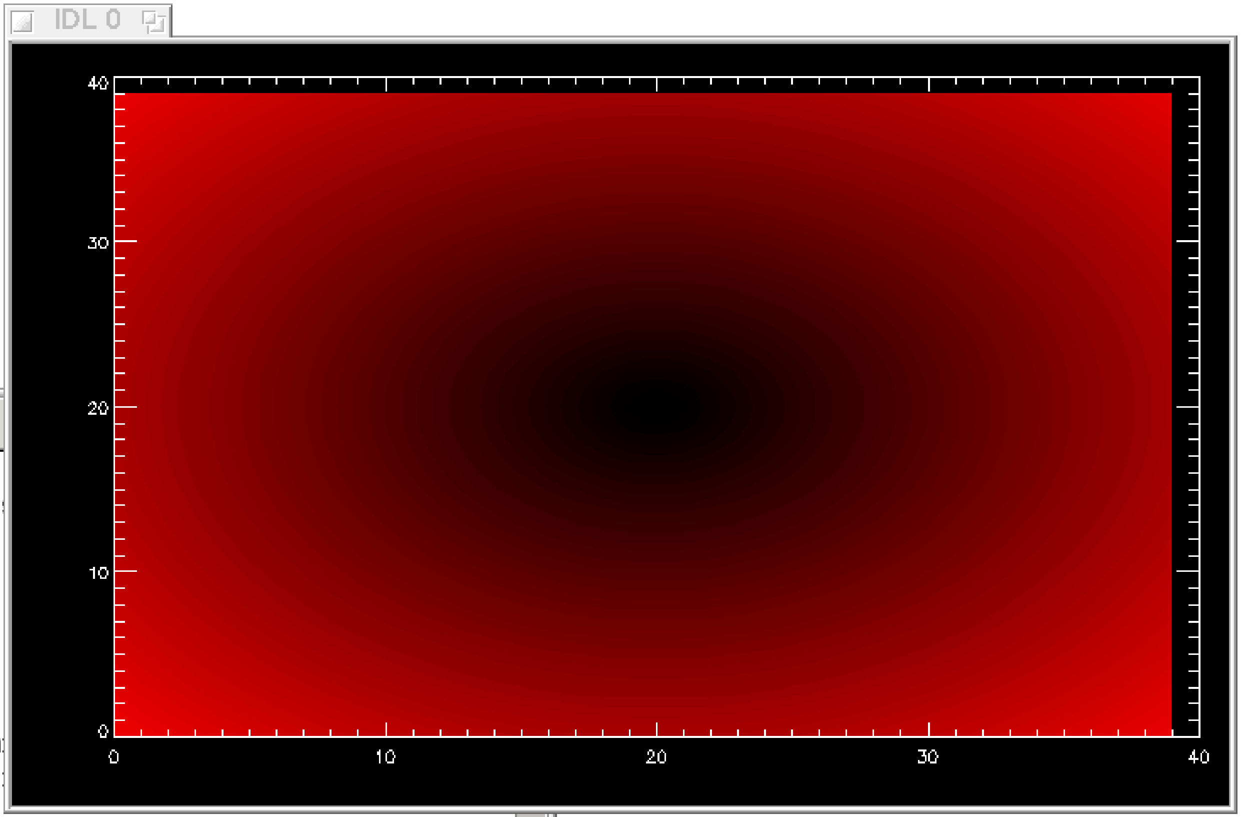


2D contour plots

/fill keyword switch creates a filled contour plot



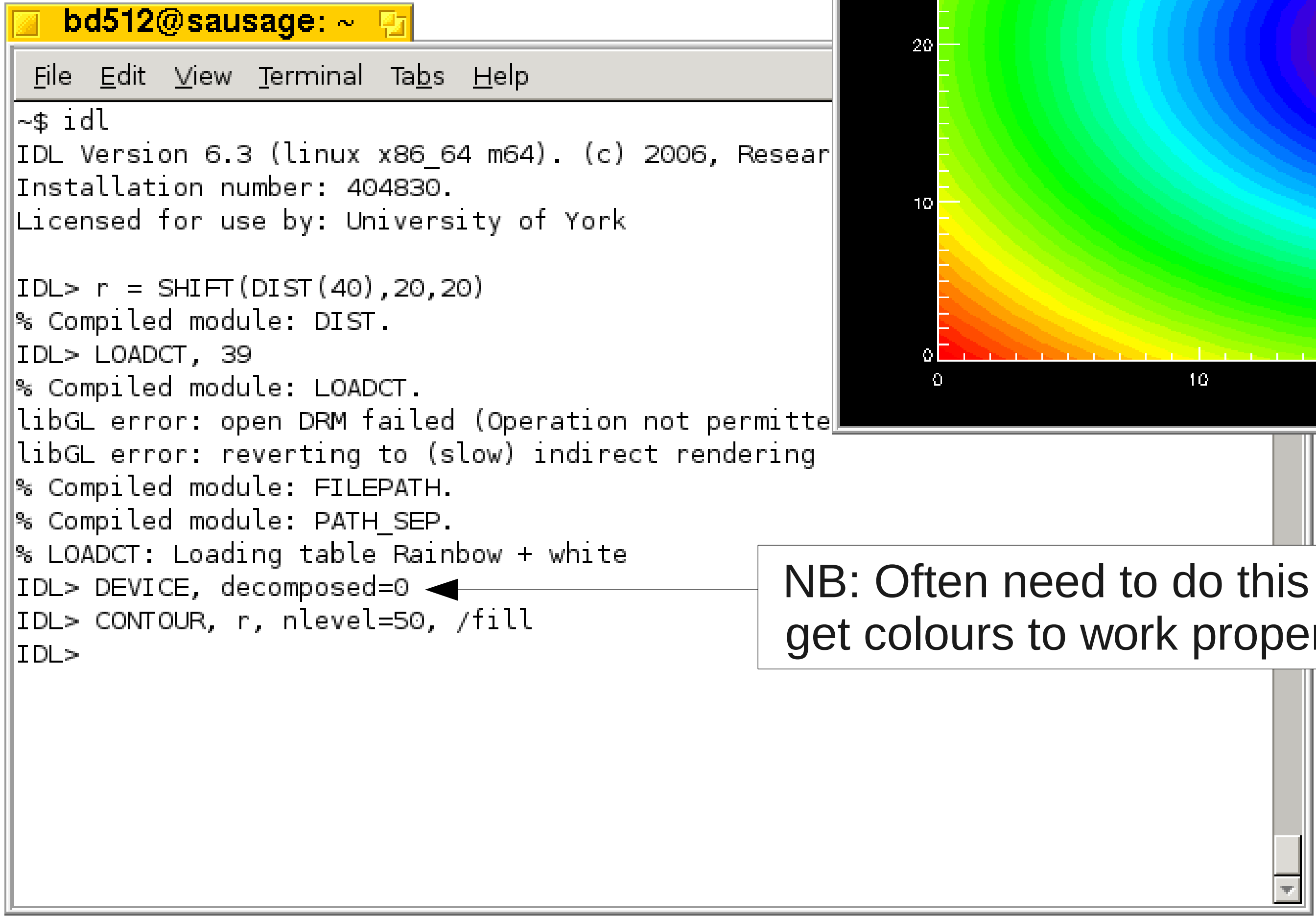
```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Re
Installation number: 404830.
Licensed for use by: University of York

IDL> r = SHIFT(DIST(40),20,20)
% Compiled module: DIST.
IDL> CONTOUR, r, nlevel=50, /fill
libGL error: open DRM failed (operation not perm
libGL error: reverting to (slow) indirect render
IDL>
```



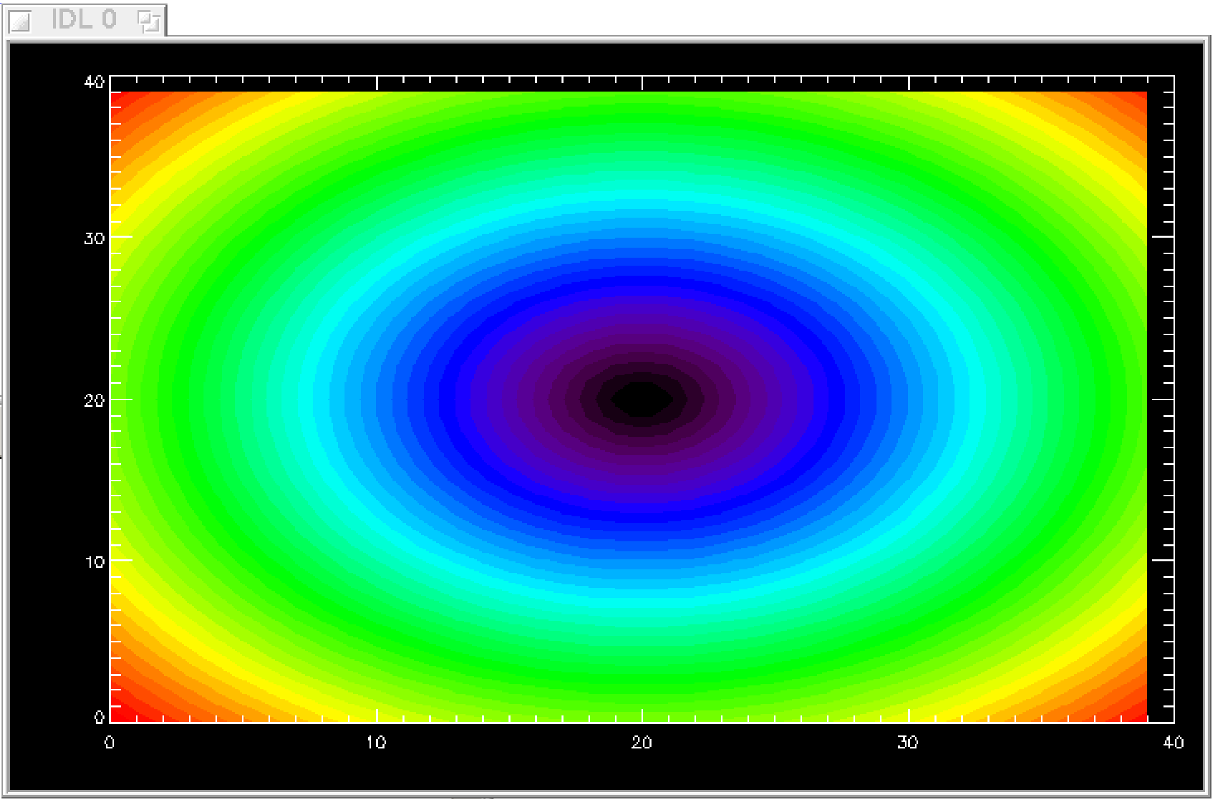
Default colors are not very nice...

Colour tables

LOADCT loads a numbered color table (here 39)



```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Resear
Installation number: 404830.
Licensed for use by: University of York

IDL> r = SHIFT(DIST(40),20,20)
% Compiled module: DIST.
IDL> LOADCT, 39
% Compiled module: LOADCT.
libGL error: open DRM failed (Operation not permitte
libGL error: reverting to (slow) indirect rendering
% Compiled module: FILEPATH.
% Compiled module: PATH_SEP.
% LOADCT: Loading table Rainbow + white
IDL> DEVICE, decomposed=0
IDL> CONTOUR, r, nlevel=50, /fill
IDL>
```



NB: Often need to do this to get colours to work properly

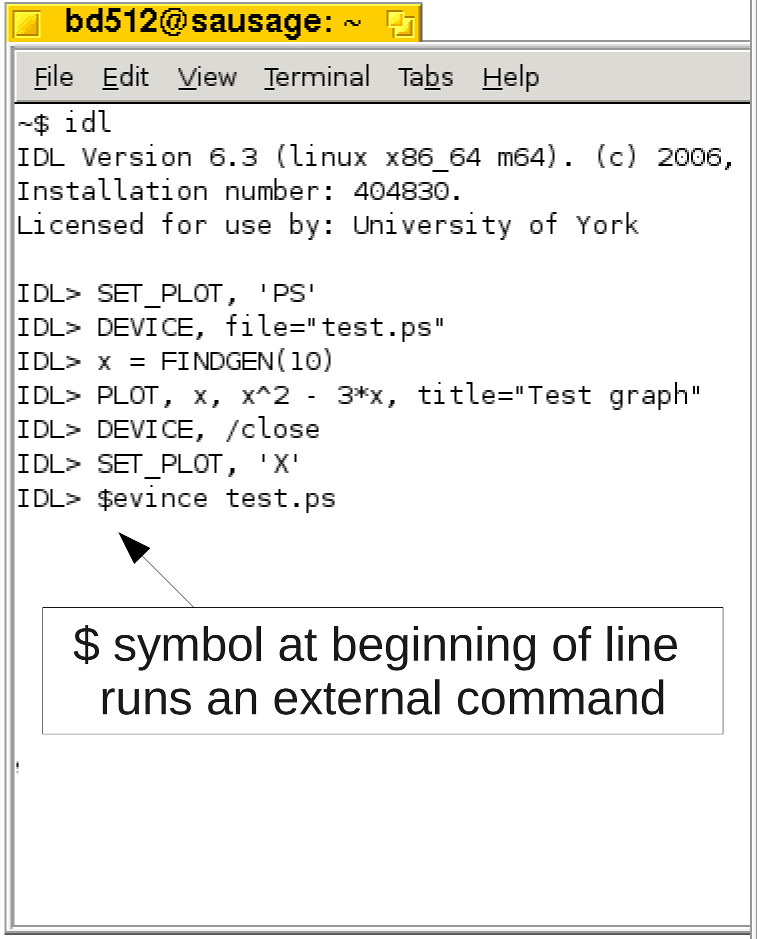
Saving your plots

At some point you will probably want to save a graph for putting into a presentation or paper.

The best way to do this is to save as a PostScript

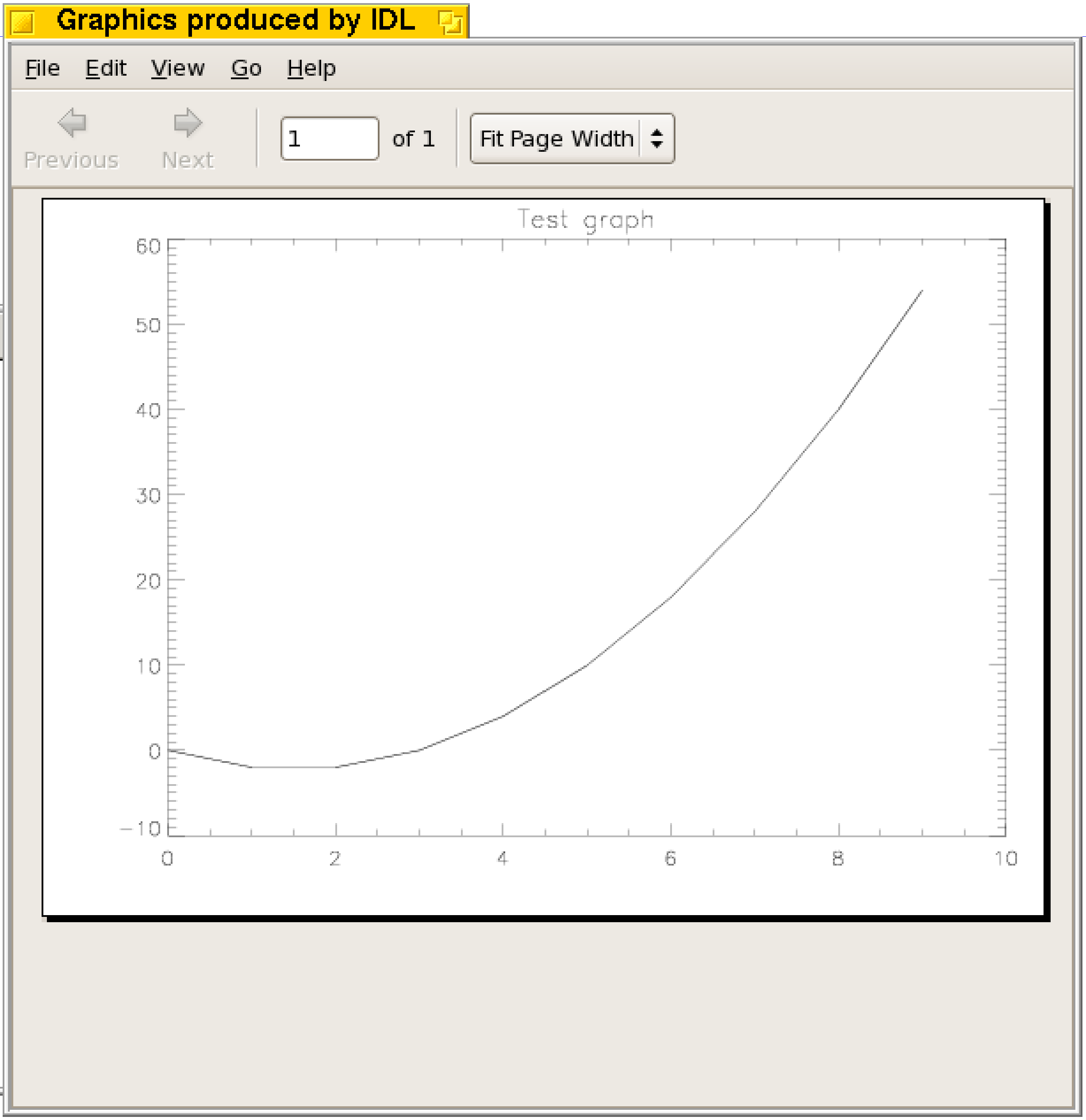
```
IDL> SET_PLOT, 'PS'

IDL> DEVICE, file=' myresult.ps'

… Plotting commands here …

IDL> DEVICE, /close

IDL> SET_PLOT, 'X'
```

Saving your plots



```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006,
Installation number: 404830.
Licensed for use by: University of York

IDL> SET_PLOT, 'PS'
IDL> DEVICE, file="test.ps"
IDL> x = FINDGEN(10)
IDL> PLOT, x, x^2 - 3*x, title="Test graph"
IDL> DEVICE, /close
IDL> SET_PLOT, 'X'
IDL> $evince test.ps
```

$ symbol at beginning of line runs an external command



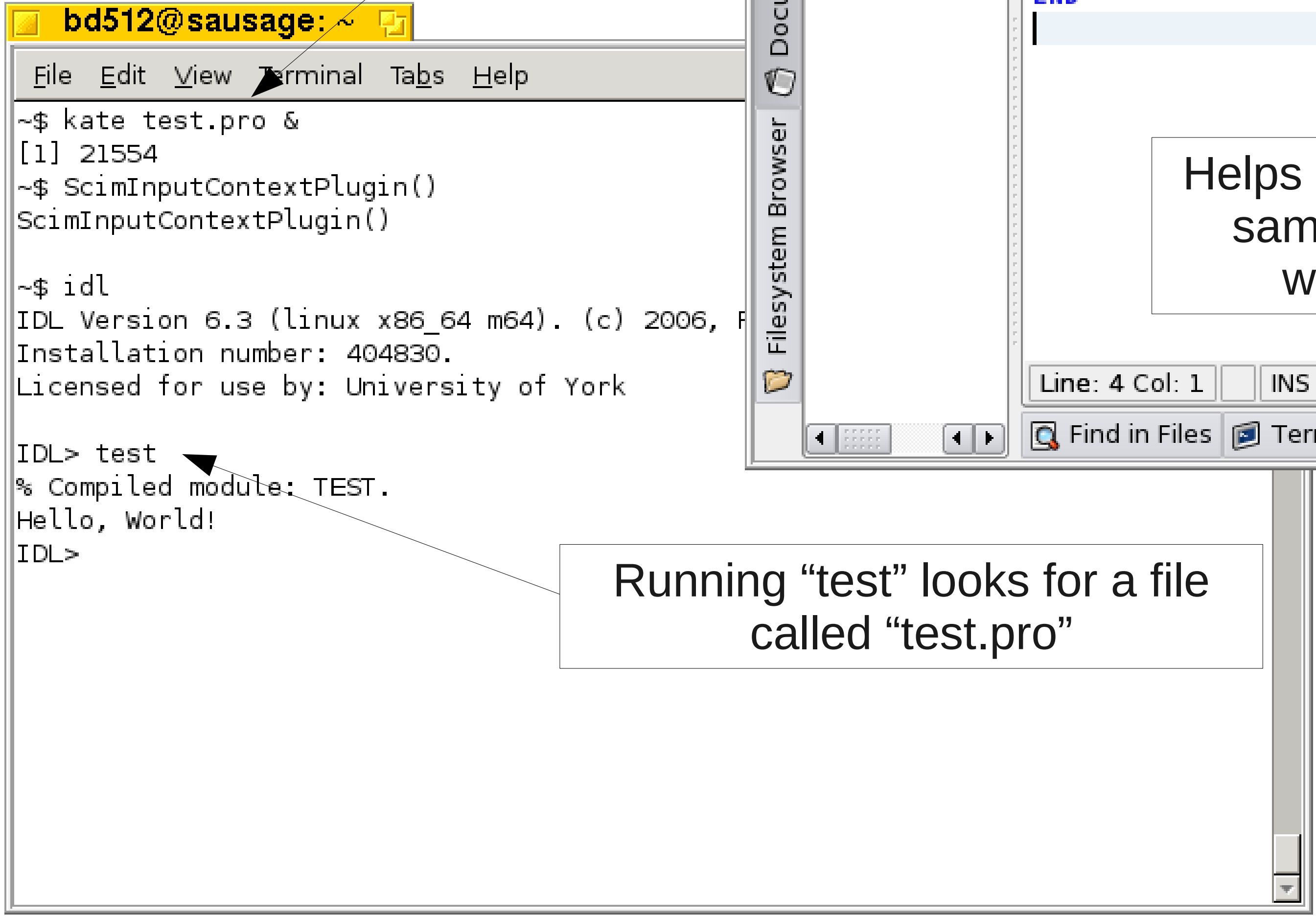
Automating IDL

- Typing commands into IDL is useful to get a quick result, but is tedious if you need more complicated programs, or to do things several times
- But IDL is more than a glorified calculator...
- You can write commands to a text file to create quite complicated codes to analyse data and plot results

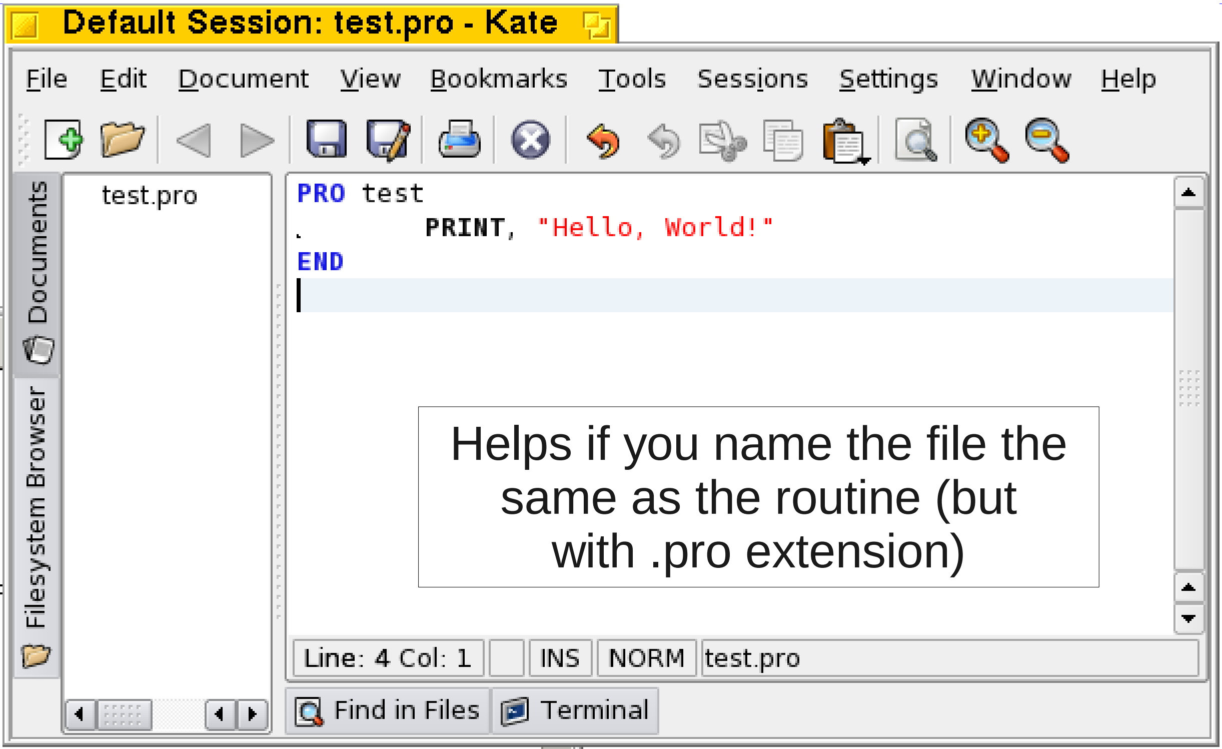
```
PRO test

  ...

  Commands here

  ...

END
```

Procedures and Functions

Create a file “`test.pro`”



```
~$ kate test.pro &
[1] 21554
~$ ScimInputContextPlugin()
ScimInputContextPlugin()

~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, R
Installation number: 404830.
Licensed for use by: University of York

IDL> test
% Compiled module: TEST.
Hello, World!
IDL>
```


Default Session: test.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
test.pro
Line: 4 Col: 1
INS
NORM
test.pro
Find in Files
Terminal

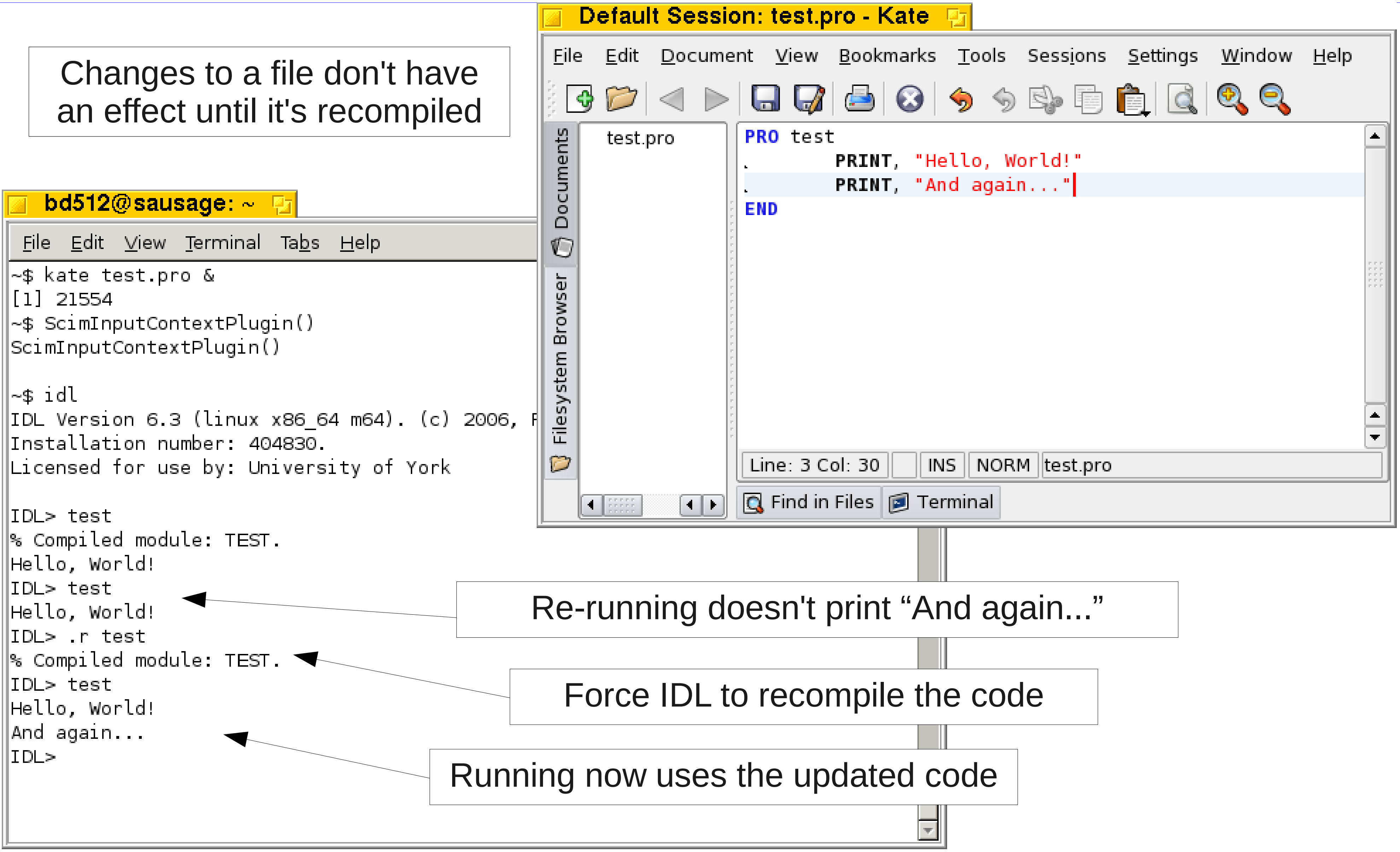
```
PRO test
        PRINT, "Hello, World!"
END
```

Helps if you name the file the same as the routine (but with .pro extension)

Running “test” looks for a file called “test.pro”

Remember to re-compile

Changes to a file don't have an effect until it's recompiled


Default Session: test.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
test.pro
Line: 3 Col: 30
INS
NORM
test.pro
Find in Files
Terminal
bd512@sausage: ~
File Edit View Terminal Tabs Help

```
PRO test
        PRINT, "Hello, World!"
        PRINT, "And again..."
END
```

```
~$ kate test.pro &
[1] 21554
~$ ScimInputContextPlugin()
ScimInputContextPlugin()

~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, R
Installation number: 404830.
Licensed for use by: University of York

IDL> test
% Compiled module: TEST.
Hello, World!
IDL> test
Hello, World!
IDL> .r test
% Compiled module: TEST.
IDL> test
Hello, World!
And again...
IDL>
```

Re-running doesn't print "And again..."

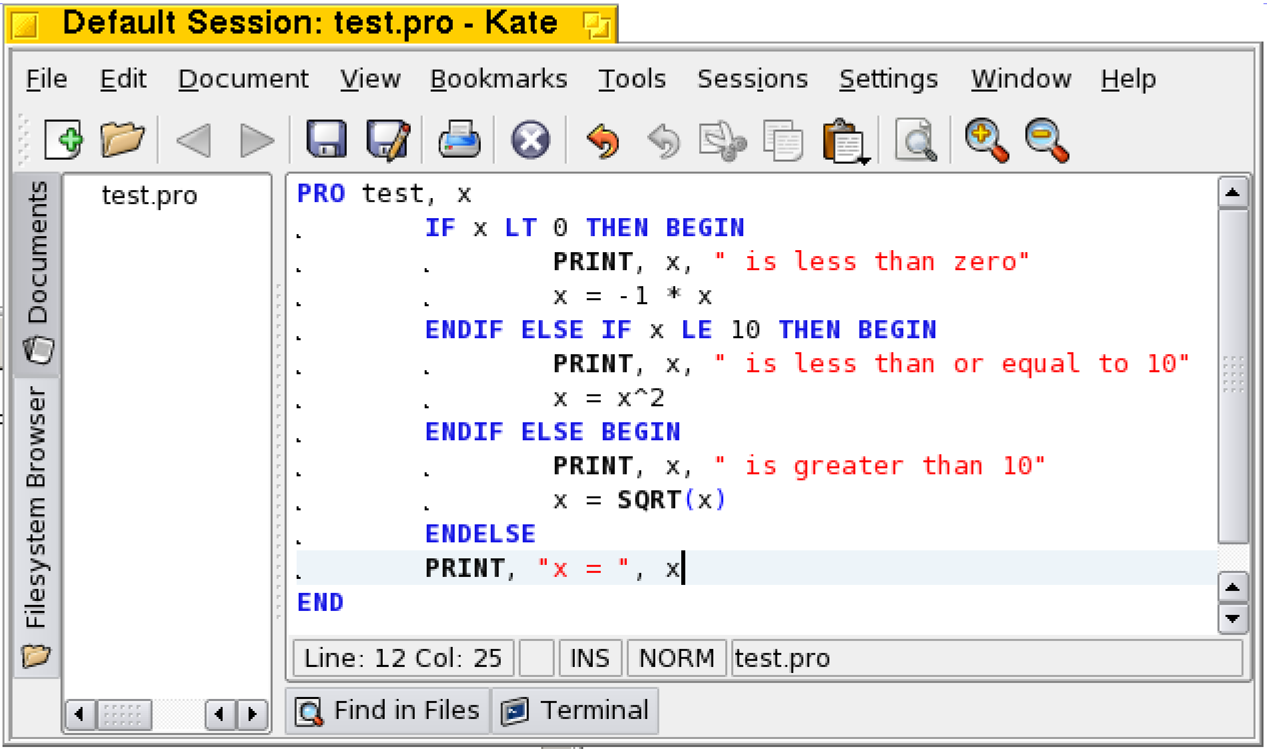
Force IDL to recompile the code

Running now uses the updated code

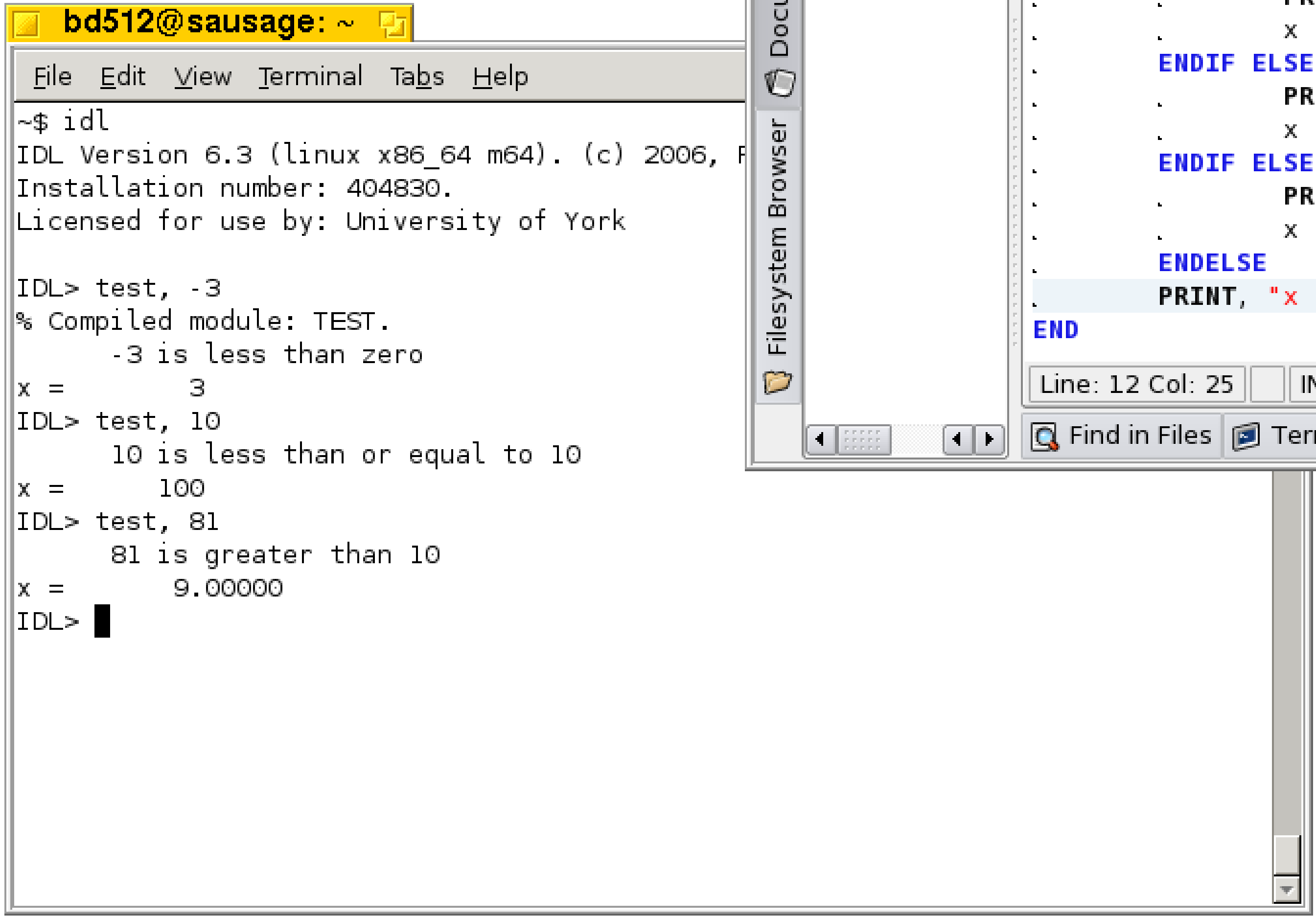
One of the little quirks that IDL has...

Conditionals II

IF statements can have several clauses, optionally ending in an 'ELSE' section


Default Session: test.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
test.pro
Line: 12 Col: 25
INS
NORM
test.pro
Find in Files
Terminal

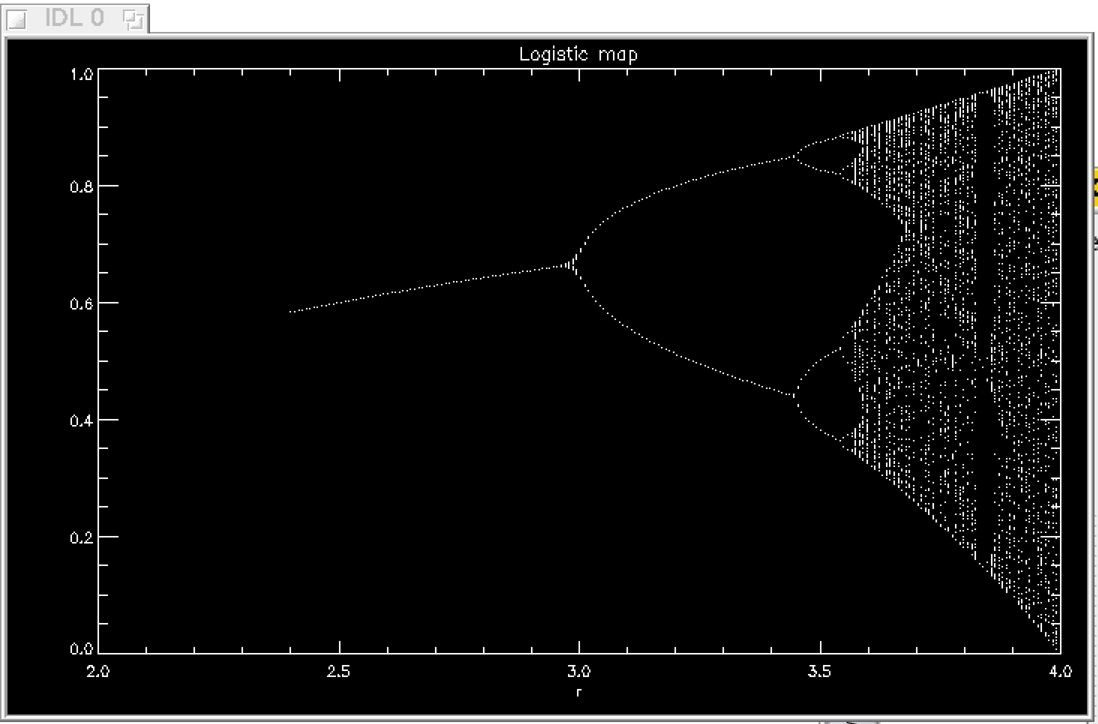
```
PRO test, x
        IF x LT 0 THEN BEGIN
                PRINT, x, " is less than zero"
                x = -1 * x
        ENDIF ELSE IF x LE 10 THEN BEGIN
                PRINT, x, " is less than or equal to 10"
                x = x^2
        ENDIF ELSE BEGIN
                PRINT, x, " is greater than 10"
                x = SQRT(x)
        ENDELSE
        PRINT, "x = ", x
END
```

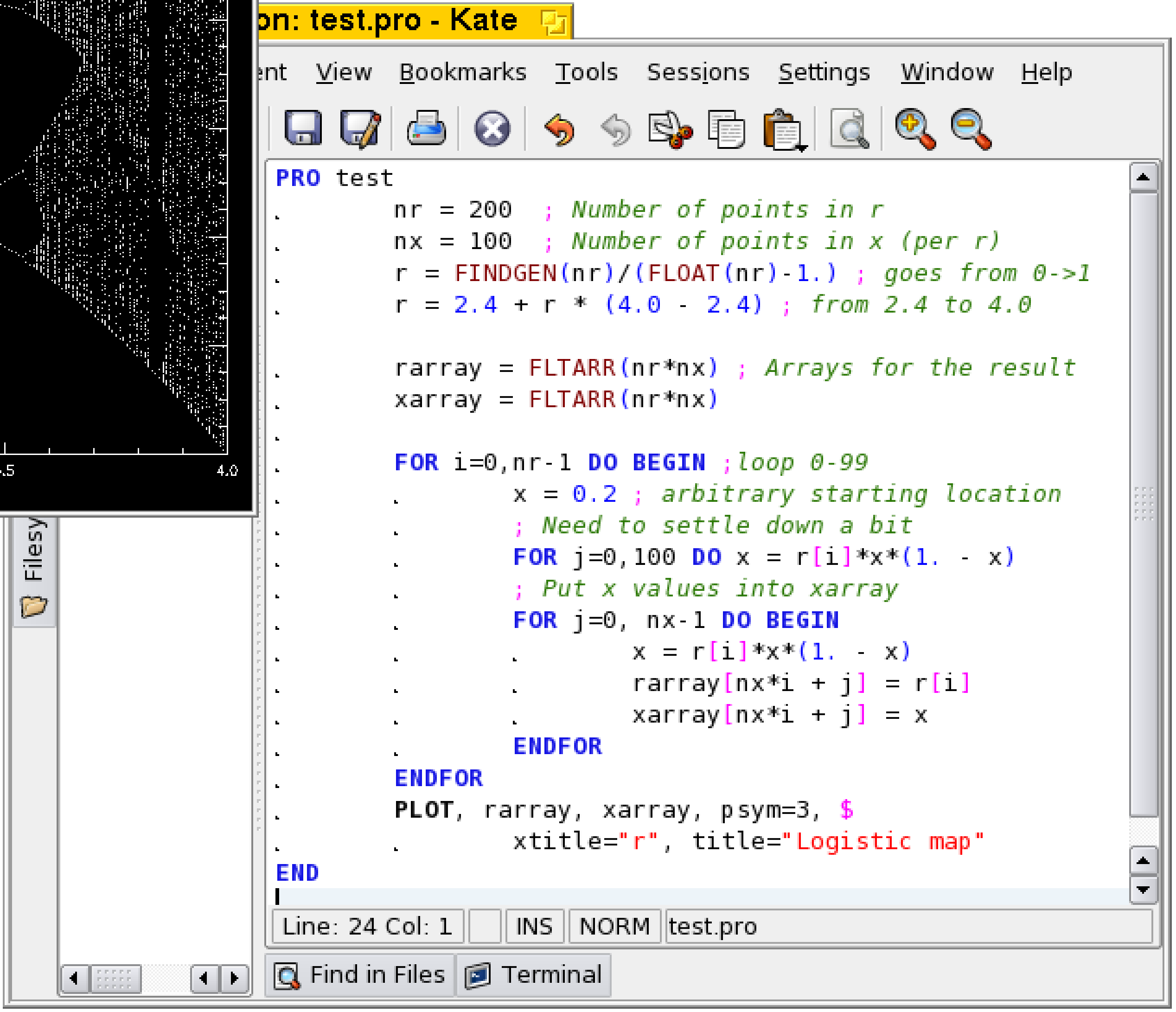


```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, R
Installation number: 404830.
Licensed for use by: University of York

IDL> test, -3
% Compiled module: TEST.
      -3 is less than zero
x =        3
IDL> test, 10
      10 is less than or equal to 10
x =      100
IDL> test, 81
      81 is greater than 10
x =       9.00000
IDL>
```

Loops

Using several loops to solve the Logistic chaotic map

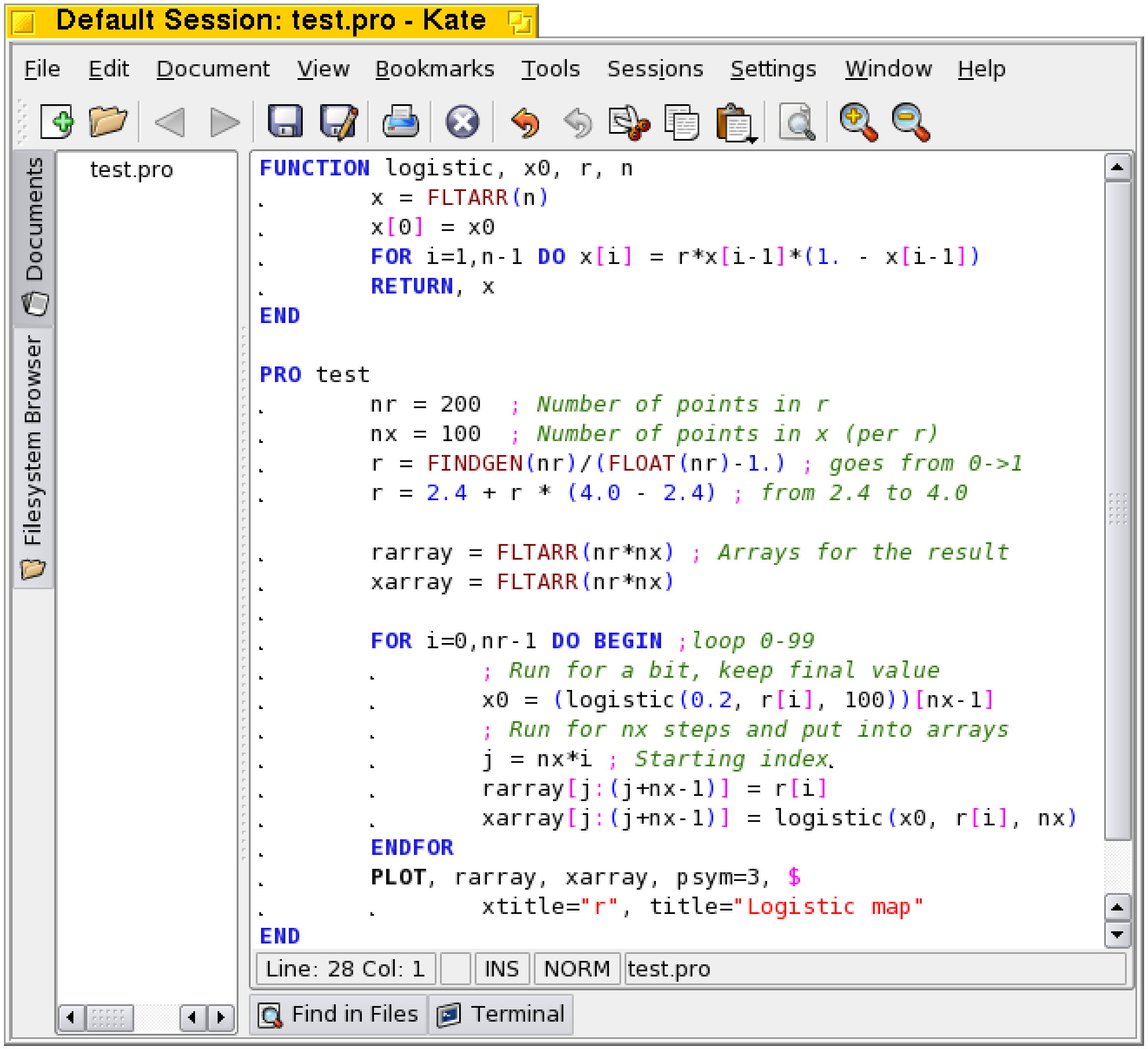




```
PRO test
        nr = 200   ; Number of points in r
        nx = 100   ; Number of points in x (per r)
        r = FINDGEN(nr)/(FLOAT(nr)-1.) ; goes from 0->1
        r = 2.4 + r * (4.0 - 2.4) ; from 2.4 to 4.0

        rarray = FLTARR(nr*nx) ; Arrays for the result
        xarray = FLTARR(nr*nx)

        FOR i=0,nr-1 DO BEGIN ;loop 0-99
                x = 0.2 ; arbitrary starting location
                ; Need to settle down a bit
                FOR j=0,100 DO x = r[i]*x*(1. - x)
                ; Put x values into xarray
                FOR j=0, nx-1 DO BEGIN
                        x = r[i]*x*(1. - x)
                        rarray[nx*i + j] = r[i]
                        xarray[nx*i + j] = x
                ENDFOR
        ENDFOR
        PLOT, rarray, xarray, psym=3, $
                xtitle="r", title="Logistic map"
END
```

Breaking up your code

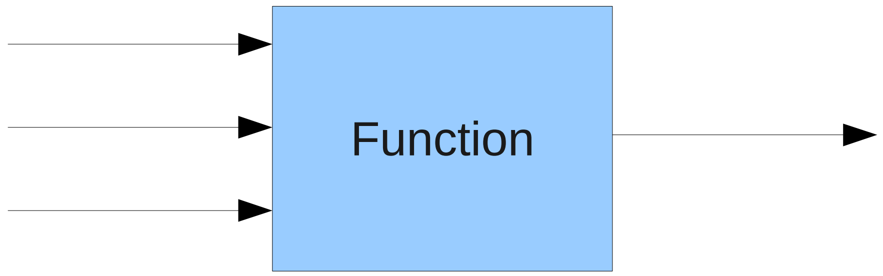
If you are doing something several times in a code, it's usually good to create a separate function


Default Session: test.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
test.pro
Line: 28 Col: 1
INS
NORM
test.pro
Find in Files
Terminal

```
FUNCTION logistic, x0, r, n
        x = FLTARR(n)
        x[0] = x0
        FOR i=1,n-1 DO x[i] = r*x[i-1]*(1. - x[i-1])
        RETURN, x
END

PRO test
        nr = 200  ; Number of points in r
        nx = 100  ; Number of points in x (per r)
        r = FINDGEN(nr)/(FLOAT(nr)-1.) ; goes from 0->1
        r = 2.4 + r * (4.0 - 2.4) ; from 2.4 to 4.0

        rarray = FLTARR(nr*nx) ; Arrays for the result
        xarray = FLTARR(nr*nx)

        FOR i=0,nr-1 DO BEGIN ;loop 0-99
                ; Run for a bit, keep final value
                x0 = (logistic(0.2, r[i], 100))[nx-1]
                ; Run for nx steps and put into arrays
                j = nx*i ; Starting index.
                rarray[j:(j+nx-1)] = r[i]
                xarray[j:(j+nx-1)] = logistic(x0, r[i], nx)
        ENDFOR
        PLOT, rarray, xarray, psym=3, $
                xtitle="r", title="Logistic map"
END
```

Using functions

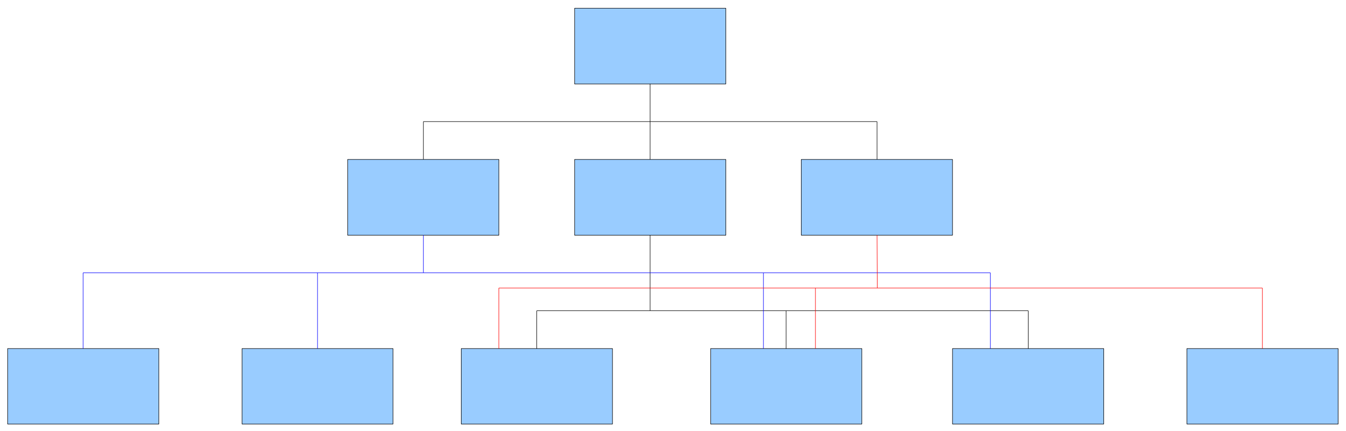
- To make your code understandable, break your problem up into smaller problems and solve separately
- If a function gets beyond ~50 lines long then try to split
- Each function should have a well defined set of inputs and outputs. The aim is to hide the details of what's inside



- A good start is to separate the parts of your code which deal with the user (input and output) from the parts which perform calculations
- Think about which bits are specific to your problem, and what parts are more general and can be used elsewhere

Principles of programming

- Computers are very very stupid. They have absolutely no common sense, and will do precisely what you tell them to do.
- Details matter. Programming languages are designed to remove ambiguity and redundancy, so you can't be vague.
- Abstraction is vital. Large programs have millions of lines of code, and trying to understand all this at once is impossible. The solution is to build up large programs in layers, each one using the layer below and hiding details from the layer above.



FORTRAN

- FORmula TRANslator was developed in the 1950s
- Designed for scientific users and number-crunching
- FORTRAN 77 was the standard scientific language for a long time
- FORTRAN 90, 95 and 2003 added extra features such as objects

```
PROGRAM HELLO_WORLD
 IMPLICIT NONE
 PRINT *, "Hello, World!"
END PROGRAM
```

C / C++

- C was developed in the 1970s as a systems programming language (UNIX was written in it, used for essentially all modern operating systems)
- C++ developed in 1980. A superset of C which includes features such as objects and templates (STL) *.
- Has directly influenced many popular languages: C++, Java, JavaScript, PHP, Perl, C#, Objective-C, ...

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

* Some features not compatible from C to C++

FORTRAN vs. C/C++

- Lots of scientific software written in FORTRAN
- Modern versions include Object-Oriented extensions
- Automates many details for the programmer
- Tends to have highest performance for number crunching

- C/C++ family of languages (e.g. Java, JavaScript, C#, ...) widely used outside academia
- Good compilers, debugging tools etc. freely available
- Allows fine control over memory access (pointers)
- C++ becoming more common for scientific use

Writing good code

- Write comments as you go. Helps others work out what your code is doing, and yourself when you come back to it.
- Fail quickly. Test your code as early as possible. Don't just write a huge code then expect it to work first time: You will spend much longer fixing bugs than writing code.
- Write less code. Number of bugs tends to increase faster than linearly with lines of code. Don't repeat yourself, and split big codes into smaller pieces which can be reused.
- Don't re-invent the wheel. Unless you have a really good reason, use library routines rather than write your own.

- Takes lots of practice. Look at others' code, and see what works and what doesn't

Weekly problems

- Due on Fridays most weeks this term
- Problems given in programming handbook
- Email your answers to me at bd512@york.ac.uk by 5:00pm
- **FORTRAN.** I will compile your code using gfortran:
```
$ gfortran -o test yourfile
$ ./test
```
- **C/C++**. Compile using g++, so you can use C++ features.
```
$ g++ -o test yourfile
$ ./test
```
- Marks and comments will be emailed back to you by the following friday

http://www-users.york.ac.uk/~bd512/teaching.shtml
for course information and links

Weekly problems

Programming mark scheme

Category	Criteria	Percentage
Results	•Code compiles and runs •Produces correct output •Figures well presented	40%
Structure	•Code is readable, has logical layout •Appropriate use of functions •Use of comments	40%
Efficiency	•Code is concise but clear •Solves the problem quickly	20%

See the programming handout

http://www-users.york.ac.uk/~bd512/teaching.shtml
for course information and links

Resources

Course information, links

http://www-users.york.ac.uk/~bd512/teaching.shtml

IDL introduction on the wiki

http://wilson5/mediawiki/index.php/IDL

Richard Martin's IDL slides (linked from IDL wiki page)

More tutorials, exercises linked to from my Links page

See
http://www.dfanning.com
for IDL tips and tricks