CCLRC
Daresbury Laboratory

Python for Scientific Programming

Paul Sherwood
CCLRC Daresbury Laboratory
p.sherwood@dl.ac.uk

Overview

- Introduction to the language
  - (thanks to David Beazley)
- Some important extension modules
  - free tools to extend the interpreter
  - extending and embedding with C and C++
- Chemistry projects in Python
  - some examples of science projects that use Python
- Our experiences
  - a GUI for quantum chemistry codes in Python
  - a users perspective

For URLs of the packages referred to in this talk, please see http://www.cse.clrc.ac.uk/qcg/python

CCLRC Daresbury Laboratory

Introduction to the Language

- An Interpreted Language….
  – Dynamic nature (typing, resolving etc)
  – New code can be entered in a running shell
  – Modules can be updated in a running interpreter
  – Silently compiles to intermediate bytecode

- Key Language Features
  – It has efficient high-level data structures
  – A simple but effective approach to object-oriented programming
  – Elegant syntax
  – Dynamic typing

CCLRC
Daresbury Laboratory

Language Overview

- Based on Slide series "An Introduction to Python by David M. Beazley

  Department of Computer Science
  University of Chicago
  beazley@cs.uchicago.edu

  O'Reilly Open Source Conference
  July 17, 2000

`http://innerpeace.org/download/pythonguideoffline.zip`

- Author of the "Python Essential Reference" in 1999 (New Riders Publishing).
- All of the material presented here can be found in that source

Python

What is it?
A freely available interpreted object-oriented scripting language.
Often compared to Tcl and Perl, but it has a much different flavor.
And a lot of people think it's pretty cool.

History
Developed by Guido van Rossum in early 1990's.
Named after Monty Python.
Influences include ABC, Modula-2, Lisp, C, and shell scripting.

Availability
http://www.python.org
It is included in most Linux distributions.
Versions are available for Unix, Windows, and Macintosh.
JPython. Python interpreter written in Java (http://www.jpython.org).

Starting Python

Chances are, Python is already installed on your machine...

```
unix % python
Python 1.5.2 (#1, Sep 19 1999, 16:29:25) [GCC 2.7.2.3] on linux2 Copyright 1991-
    1995 Stichting Mathematisch Centrum, Amsterdam
>>>
```

This starts the interpreter and allows you to type programs interactively.

On Windows and Macintosh

Python is launched as an application.

An interpreter window will appear and you will see the prompt.

IDLE

An integrated development environment for Python.

Available at www.python.org.

CCLRC
Daresbury Laboratory

Your First Program

Hello World

```
>>> print "Hello World“
Hello World
>>>
```

Well, that was easy enough.

Python as a calculator

```
>>> 3*4.5 + 5
18.5
 >>>
```

Basically, interactive mode is just a simple read-eval loop.

Something more complicated

```
>>> for i in range(0,10):
 ... print i
0
1
2
... etc ...
```

Programs and Files

Programs are generally placed in .py files like this

```
# helloworld.py
print "Hello World"
```

To run a file, give the filename as an argument to the interpreter

```
unix % python helloworld.py
Hello World
unix %
```

Or you can use the Unix #! trick

```
#!/usr/local/bin/python
print "Hello World"
```

Or you can just double-click on the program (Windows)

CCLRC Daresbury Laboratory

Program Termination

Program Execution

- Programs run until there are no more statements to execute.
- Usually this is signaled by EOF
- Can press Control-D (Unix) or Control-Z (Windows) to exit interactive interpreter

Forcing Termination

- Raising an exception:

```
>>> raise SystemExit
```

- Calling exit manually:

```
import sys
sys.exit(0)
```

CCLRC Daresbury Laboratory

Variables and Expressions

Expressions

- Standard mathematical operators work like other languages:

```
3 + 5
3 + (5*4)
3 ** 2
'Hello' + 'World'
```

Variable assignment

```
a = 4 << 3
b = a * 4.5
c = (a+b)/2.5
a = "Hello World"
```

- Variables are dynamically typed (No explicit typing, types may change during execution).
- Variables are just names for an object. Not tied to a memory location like in C.

Conditionals

if-else

```
# Compute maximum (z) of a and b
if a < b:
    z = b
else:
    z = a
```

The pass statement

```
if a < b:
    pass                  # Do nothing
else:
    z = a
```

Notes:

- Indentation used to denote bodies.
- pass used to denote an empty body.
- There is no '?:' operator.

Conditionals

elif statement

```
if a == '+':
    op = PLUS
elif a == '-':
    op = MINUS
elif a == '*':
    op = MULTIPLY
else:
    op = UNKNOWN
```

- Note: There is no switch statement.

Boolean expressions: and, or, not

```
if b >= a and b <= c:
    print "b is between a and c”
if not (b < a or b > c):
    print "b is still between a and c"
```

- Note: &&, ||, and ! are not used.

CCLRC Daresbury Laboratory

Basic Types (Numbers and Strings)

Numbers

```
a = 3                # Integer
b = 4.5              # Floating point
c = 517288833333L    # Long integer (arbitrary precision)
d = 4 + 3j           # Complex (imaginary) number
```

Strings

```
a = 'Hello'                     # Single quotes
b = "World"                     # Double quotes
c = "Bob said 'hey there.'"     # A mix of both
d = '''A triple quoted string can span multiple lines
like this'''
e = """Also works for double quotes"""
```

CCLRC Daresbury Laboratory

Basic Types (Lists)

Lists of arbitrary objects

```
a = [2, 3, 4]                 # A list of integers
b = [2, 7, 3.5, "Hello"]   # A mixed list
c = []                          # An empty list
d = [2, [a,b]]               # A list containing a list
e = a + b                     # Join two lists
```

List manipulation

```
x = a[1]                      # Get 2nd element (0 is first)
y = b[1:3]                    # Return a sublist
z = d[1][0][2]               # Nested lists
b[0] = 42                     # Change an element
```

List methods

```
a.append("foo")             # Append an element
a.insert(1,"bar")           # Insert an element
len(a)                          # Length of the list del a[2] # Delete an element
```

CCLRC Daresbury Laboratory

Basic Types (Tuples)

Tuples

```
f = (2,3,4,5)                    # A tuple of integers
g = (1,)                         # A one item tuple
h = (2, [3,4], (10,11,12))       # A tuple containing mixed objects
```

Tuple Manipulation

```
x = f[1]                         # Element access.
x = 3 y = f[1:3]                 # Slices.
y = (3,4) z = h[1][1]            # Nesting. z = 4
```

Comments

- Tuples are like lists, but size is fixed at time of creation.
- Can't replace members (said to be "immutable")
- Why have tuples at all? This is actually a point of much discussion.

CCLRC Daresbury Laboratory

Basic Types (Dictionaries)

Dictionaries (Associative Arrays)

```
a = { }                              # An empty dictionary
b = { 'x': 3,
      'y': 4 }
c = { 'uid': 105, 'login': 'beazley', 'name' : 'David Beazley' }
```

Dictionary Access

```
u = c['uid']                         # Get an element
c['shell'] = "/bin/sh"               # Set an element
if c.has_key("directory"):           # Check for presence of an member
    d = c['directory']
else:
    d = None
d = c.get("directory",None)          # Same thing, more compact
k = c.keys()                         # Get all keys as a list
```

Loops

The while statement

```
while a < b:
    # Do something
    a = a + 1
```

The for statement (loops over members of a sequence)

```
for i in [3, 4, 10, 25]:
    print i
# Print characters one at a time
for c in "Hello World":
    print c
# Loop over a range of numbers
for i in range(0,100):
    print i
```

CCLRC Daresbury Laboratory

Functions

The def statement

```
# Return the remainder of a/b
def remainder(a,b):
    q = a/b
    r = a - q*b
    return r
# Now use it
a = remainder(42,5)     # a = 2
```

Returning multiple values (a common use of tuples)

```
def divide(a,b):
    q = a/b
    r = a - q*b
    return q,r
x,y = divide(42,5)     # x = 8, y = 2
```

CCLRC Daresbury Laboratory

Classes

The class statement

```
class Account:
    def __init__(self, initial):
        self.balance = initial
    def deposit(self, amt):
        self.balance = self.balance + amt
    def withdraw(self,amt):
        self.balance = self.balance – amt
    def getbalance(self):
        return self.balance
```

Using a class

```
a = Account(1000.00)
a.deposit(550.23)
a.deposit(100)
a.withdraw(50)
print a.getbalance()
```

Exceptions

The try statement

```
try:
    f = open("foo")
except IOError:
    print "Couldn't open 'foo'. Sorry."
```

The raise statement

```
def factorial(n):
    if n < 0:
        raise ValueError,"Expected non-negative number"
    if (n <= 1):
        return 1
    else:
        return n*factorial(n-1)
```

Exceptions

Uncaught exceptions

```
>>> factorial(-1)
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in factorial
ValueError: Expected non-negative number
```

CCLRC
Daresbury Laboratory

Files

The open() function

```
f = open("foo","w")       # Open a file for writing
g = open("bar","r")       # Open a file for reading
```

Reading and writing data

```
f.write("Hello World")
data = g.read()           # Read all data
line = g.readline()       # Read a single line
lines = g.readlines()     # Read data as a list of lines
```

Formatted I/O

Use the % operator for strings (works like C printf)

```
for i in range(0,10):
    f.write("2 times %d = %d\n" % (i, 2*i))
```

CCLRC Daresbury Laboratory

Modules

Large programs can be broken into modules

```
# numbers.py
def divide(a,b):
    q = a/b r = a - q*b
    return q,r
def gcd(x,y):
    g = y
    while x > 0:
        g = x
        x = y % x
        y = g
    return g
```

The import statement

```
import numbers
x,y = numbers.divide(42,5)
n = numbers.gcd(7291823, 5683)
```

- import creates a namespace and executes a file

Python Library

Python is packaged with a large library of standard modules
String processing
Operating system interfaces
Networking
Threads
GUI
Database
Language services
Security.

And there are many third party modules
XML
Numeric Processing
Plotting/Graphics
etc.

All of these are accessed using 'import'

```
import string
a = string.split(x)
```

CCLRC
Daresbury Laboratory

Summary so far ….

You have seen about 90% of what you need to know
Python is a relatively simple language.
Most novices can pick it up and start doing things right away.
The code is readable.
When in doubt, experiment interactively.

… for more of David Beazley's slides, see the web pages (link at end).

Standards and Portability

- Not subject to any standardisation effort, it is essentially a single implementation
  - i.e. python is defined by an open-source program, written in C, which can be ported to a wide range of platforms.
  - Jython is the main exception.. A Python interpreter which runs in a Java VM
- In practice
  - How portable is the interpreter?
    - It can easily be downloaded for Windows, Linux, Mac OSX
      - **Some issues with some modules, e.g. TkInter on Mac OSX**
    - or compiled from Source
  - Wide user base is comforting
  - Main portability issues will be around the extensions

CCLRC Daresbury Laboratory

Extension Packages

- The range of freely downloadable modules is one of the strengths of Python
- Usually adding a module to your distribution is relatively painless
  - code is dynamically loaded from the interpreter, no relinking of interpreter needed
  - standardised approach to compiling and/or installing as part of your python installation (distutils module provides setup.py)
  - binary distributions are usually available (.rpm under linux, .exe installers in windows)

CCLRC Daresbury Laboratory

Extension Packages

- Numerical Python
  - adds an multidimensional array datatype, with access to fast routines (BLAS) for matrix operations
- Scientific Python
  - basic geometry (vectors, tensors, transformations, vector and tensor fields), quaternions, automatic derivatives, (linear) interpolation, polynomials, elementary statistics, nonlinear least-squares fits, unit calculations, Fortran-compatible text formatting, 3D visualization via VRML, and two Tk widgets for simple line plots and 3D wireframe models. Interfaces to the netCDF, MPI, and to BSPlib.
- SciPy
  - includes modules for graphics and plotting, optimization, integration, special functions, signal and image processing, genetic algorithms, ODE solvers, and others

CCLRC Daresbury Laboratory

Extension Packages

- GUI toolkits
  - Tkinter
    - python bindings to Tk toolkit, shipped with python and used by python's own IDE (IDLE)
    - still some problems here on MacOS/X
  - Pmw (Python MegaWidgets)
    - more complex widgets based on Tkinter
  - wxPython
  - pyQT
  - pyGTK
- Also consider….
  - Anygui
    - write once, run with any toolkit

Extension Packages

- 3D Visualisation
  - pyOpenGL
    - low level 3D primitives
  - Visual Python (now vpython)
    - low level
  - VTK
    - large and powerful visualisation toolkit
    - can be tricky to build from source
- Graph plotting
  - matplotlib
    - pure python library with matlib-like approach
  - Pmw/BLT
    - BLT is a Tk extension, Pmw provides bindings
  - R Bindings
    - general purpose statistics language with plotting tools

CCLRC Daresbury Laboratory

Extension Packages

- Web
  - Zope is a web server written in Python
  - Python can be used as a CGI language
    - install mod_python into apache to avoid start-up costs of each script

- Grid and e-Science
  - Python tools for XML
    - pyXML
    - 4suite package (recommended)

  - Python COG kit for globus
    - client side tools

Tools

- Windows
  - I regard Mark Hammond’s PythonWin is essential
    - good handling of windows processes
    - access to MFC, COM etc
    - convenient way to move data from scientific applications to Excel and similar windows software

- Wrapping - automated generation of python commands from libraries and their header files
  - SWIG - general purpose tool
  - SIP - specialised for Python and C++
  - VTK - incorporates internal wrapping code for its C++ classes

CCLRC Daresbury Laboratory

Development Environments

- IDLE
  - Python’s native IDE
- Emacs python mode
  - My choice
  - useful tools to handle code indentation (important)
  - ctrl-C ctrl-C executes the buffer
- Other Shells available
  - PythonWin
  - PyCrust
  - Ipython
- There is also an outlining editor: Leo

CCLRC Daresbury Laboratory

Extending and Embedding

- Python is a C program and has a well developed and well-documented API for
  - Extending Python
    - writing your own functions, classes etc in C, C++ etc
      - **needed to overcome limitations of interpreter performance**
  - Embedding Python
    - simplest case, just call python functions from within your code (e.g. to take advantage of extension modules)
    - more generally
      - **provide a number of extensions to the interpreter**
      - **embed python as a command line interpreter for your application**

CCLRC Daresbury Laboratory

Native Code Extension Modules

- (Example taken from the standard Python docs).
- Let's create an extension module called "spam" and let's say we want to create a Python interface to the C library function system().
- This function takes a null-terminated character string as argument and returns an integer. We want this function to be callable from Python as follows:

```
>>> import spam
>>> status = spam.system("ls -l")
```

Source File Structure

- Begin by creating a file spammodule.c.
- The first line of our file pulls in the Python API

```
#include <Python.h>
```

- All user-visible symbols defined by Python.h have a prefix of "Py" or "PY", except those defined in standard header files.
- "Python.h" includes a few standard header files: <stdio.h>, <string.h>, <errno.h>, and <stdlib.h

C Code

- This will be called when the Python expression "spam.system(string)"is evaluated

```
static PyObject *
spam_system(self, args)
    PyObject *self;
    PyObject *args;
{
    char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    return Py_BuildValue("i", sts);
}
```

Initialisation

- To declare to Python, provide a method table:

```
static PyMethodDef SpamMethods[] = {
    ...
    {"system",  spam_system, METH_VARARGS,
     "Execute a shell command."},
    ...
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
```

- provide an initialisation function named “init”+module

```
void
initspam(void)
{
    (void) Py_InitModule("spam", SpamMethods);
}
```

CCLRC Daresbury Laboratory

Distribution and Installation Tools

- Distribution and Installation
  - required tools (distutils) are now part of standard python
    - can compile from source in situ and install .so files, and can create binary distributions and RPMs
    - provide a file setup.py:

```
from distutils.core import setup, Extension
module1 = Extension(‘spam',
                    sources = [’spammodule.c'])
setup (name = 'PackageName',
       version = '1.0',
       description = 'This is a demo package',
       ext_modules = [module1])
```

    - run “python setup.py build”
  - Tools for wrapping up python interpreter + scripts into a single executable are available (py2exe)

CCLRC Daresbury Laboratory

Scientific Python Applications

- MMTK (Hinsen)
  - Includes force field modelling and MD
  - mostly python, with some C code for compute intensive parts
  - visualisation through interfaces to VMD, VRML etc (Scientific Python)

- Molecular building and visualisation
  - PyMOL (Delano)
    - C core, python and Tkinter control
  - Chimera (UCSF)
    - C++ core
  - PMV and MGLTools (Sanner, Scripps)
    - interpreted language Python as the environment for independent and re-usable components for structural bioinformatics

CCLRC
Daresbury Laboratory

Scientific Python Applications

- NWChem (PNNL)
  - quantum chemistry package with embedded python scripting
- CAMPOS/ASE
  - ab initio atomistic simulations and visualizations
  - a number of modules controlled by python interpreter
- PyQuante (Muller)
  - quantum chemistry programs scripted in python
  - some C code for integrals etc

CCLRC Daresbury Laboratory

Case Study: the CCP1GUI project

- Motivation
  - Simplify and consolidate the use of a number of chemistry codes.
  - Make it easier to get started with a particular code.
  - Particularly needed for for teaching purposes.
  - Requirement for a simplified environment for constructing and viewing molecules.
  - Need to be able to visualise the complex results of quantum mechanical calculations.
  - Program should be free so no barriers to its widespread use.
  - Need a single tool that can be made to to run on a variety of hardware/operating system platforms.

How is it being developed?

- Why we chose python
  - Free – pre-installed on many operating systems
  - Concise and easy, should help others pick it up easily
  - Heavily object-oriented – simplifies developing new interfaces based on reuse of existing code
  - Interpreted language – speeds development and prototyping
  - Integrates well with C/C++ to take advantage of compiled code if needed later
- Why VTK for visualisation
  - Free – large community of users/developers.
  - Used in many scientific fields, so a wide range of capabilites.
  - Ported to most operating systems/hardware platforms
  - Automatic wrapping for Python/Java/Tcl.

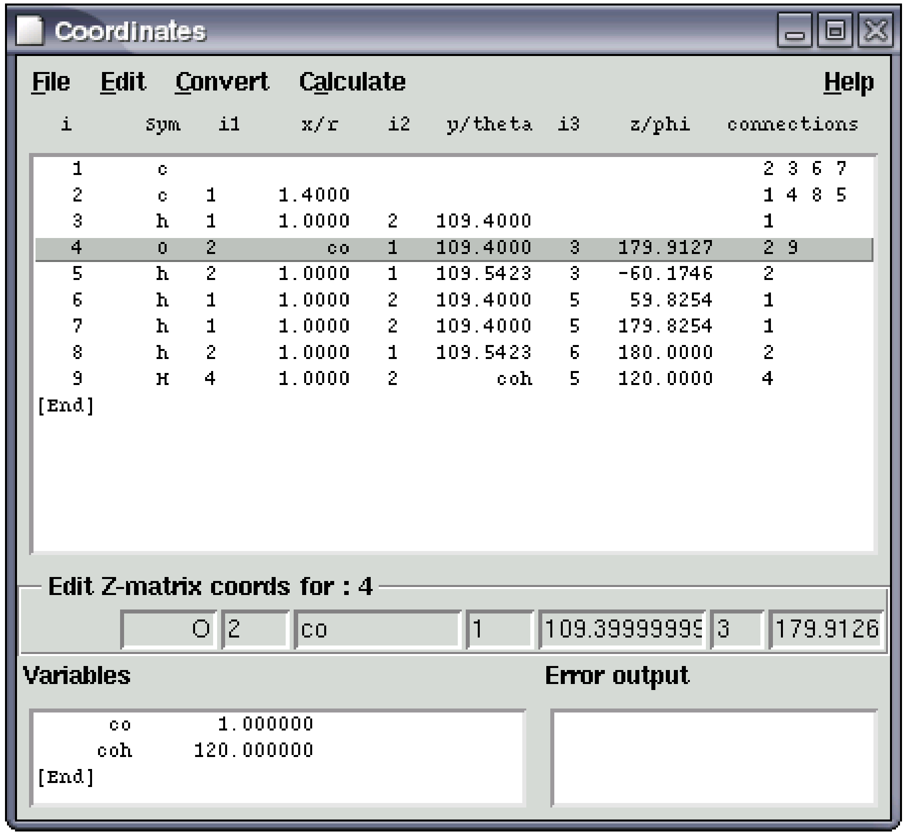
CCLRC Daresbury Laboratory

Current Capabilities

- Interfaces to GAMESS-UK, ChemShell (QM/MM) and MOPAC.
- Dalton under development, Molpro planned.
- Powerful molecule builder
  - point-and-click and internal coordinate editing
- Supports reading and writing in a variety of file formats (xyz, internal coordinates, PDB, Xmol, XML, CHARMM, ChemShell, Gaussian, GAMESS-UK…)
- Variety of visualisation options

CCP1GUI Molecule Builder

- Versatile molecule-constructing environment:
  - Simple point-and-click operations for many functions.
  - Commonly used molecular fragments added at the click of a button to quickly build up complex molecules.
  - Highly-featured Z-matrix editor for Cartesian, internal and mixed coordinates
  - Can convert between the different representations.
  - Select and set the variables for a geometry optimisation.



```
 i     sym   i1     x/r     i2   y/theta   i3    z/phi    connections

 1      c                                                  2 3 6 7
 2      c    1    1.4000                                   1 4 8 5
 3      h    1    1.0000    2   109.4000                   1
 4      o    2        co    1   109.4000    3   179.9127   2 9
 5      h    2    1.0000    1   109.5423    3   -60.1746   2
 6      h    1    1.0000    2   109.4000    5    59.8254   1
 7      h    1    1.0000    2   109.4000    5   179.8254   1
 8      h    2    1.0000    1   109.5423    6   180.0000   2
 9      H    4    1.0000    2        coh    5   120.0000   4
[End]
```

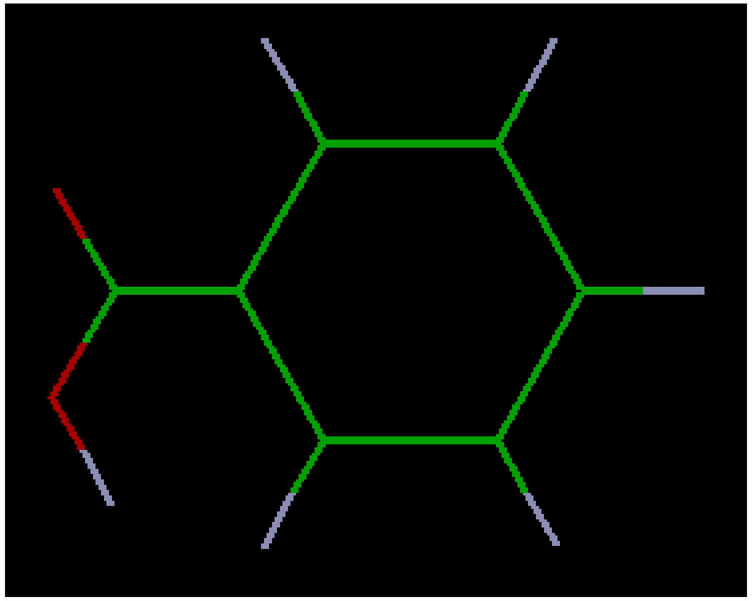
```
O  2  co  1  109.39999999  3  179.9126
```

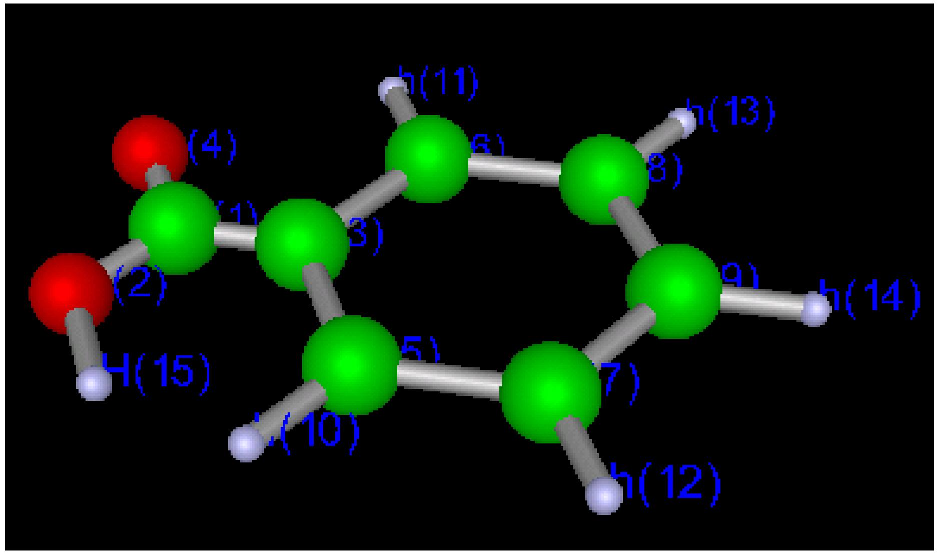
```
      co        1.000000
     coh      120.000000
[End]
```

CCLRC Daresbury Laboratory

Visualising Molecules

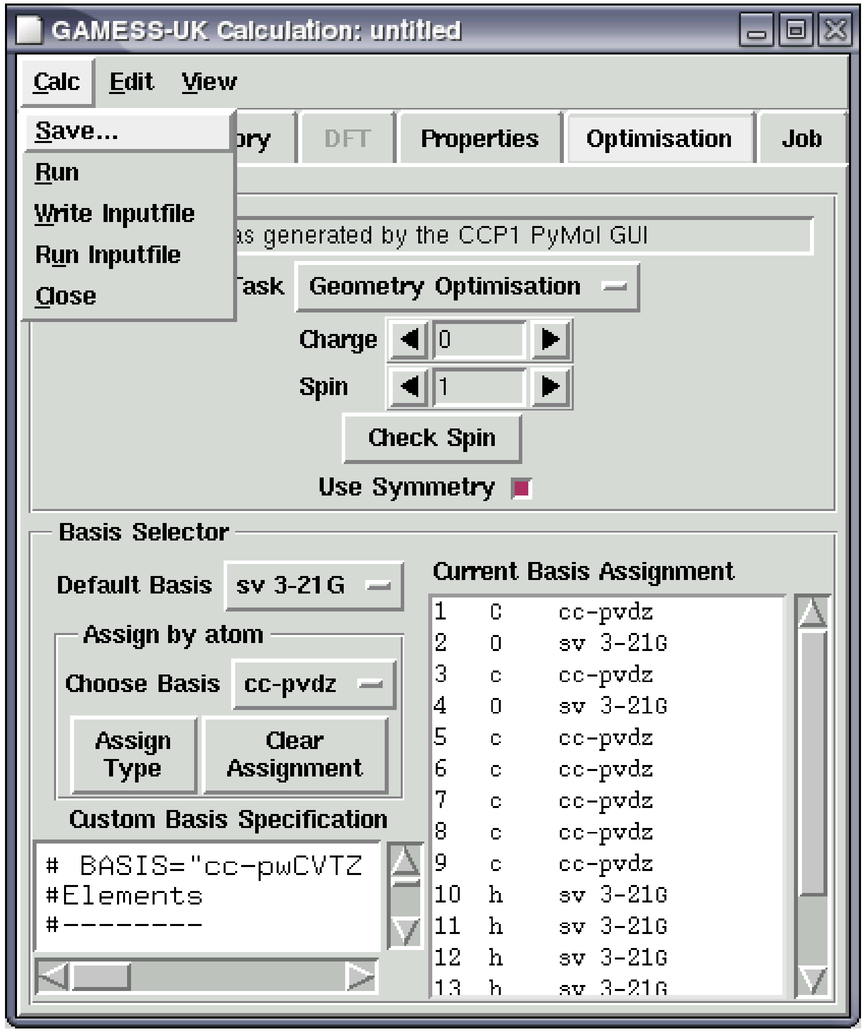
- Wireframe representation.
- “Ball and Stick” models.
- Contacts between nonbonded atoms.
- Extend repeat units.
- …





Driving GAMESS-UK

- Set up and run most basic GAMESS-UK runtypes.
- Specification of the atomic basis sets.
- Control of SCF convergence.
- Set functional/grid/Coulomb fitting for DFT calculations.
- Calculate a variety of molcular properties.
- Control of Geometry optimisations/transition state searches.
- Specify where the job is run, which files are saved, etc.

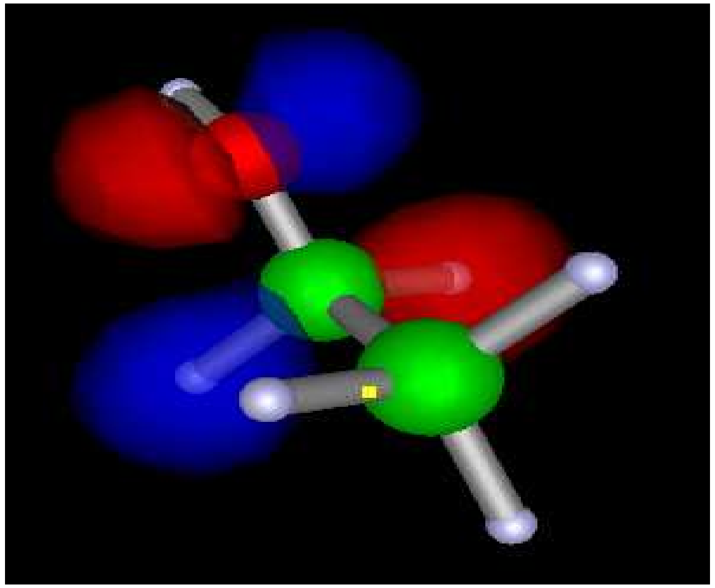


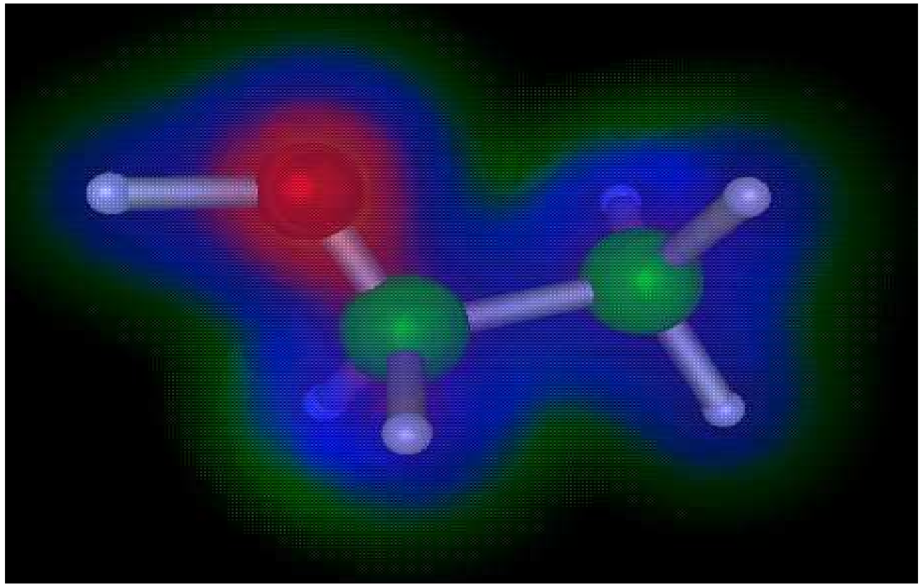
```
#  BASIS="cc-pwCVTZ
#Elements
#--------
```

```
1    C    cc-pvdz
2    O    sv 3-21G
3    c    cc-pvdz
4    O    sv 3-21G
5    c    cc-pvdz
6    c    cc-pvdz
7    c    cc-pvdz
8    c    cc-pvdz
9    c    cc-pvdz
10   h    sv 3-21G
11   h    sv 3-21G
12   h    sv 3-21G
13   h    sv 3-21G
```

Visualising Calculation Results

- Animate vibrational frequencies.
- Create a movie from the steps in a geometry optimisation pathway.
- Visualise scalar data.
  - Surfaces, grids, cut slices, volume rendering – can all be overlaid.

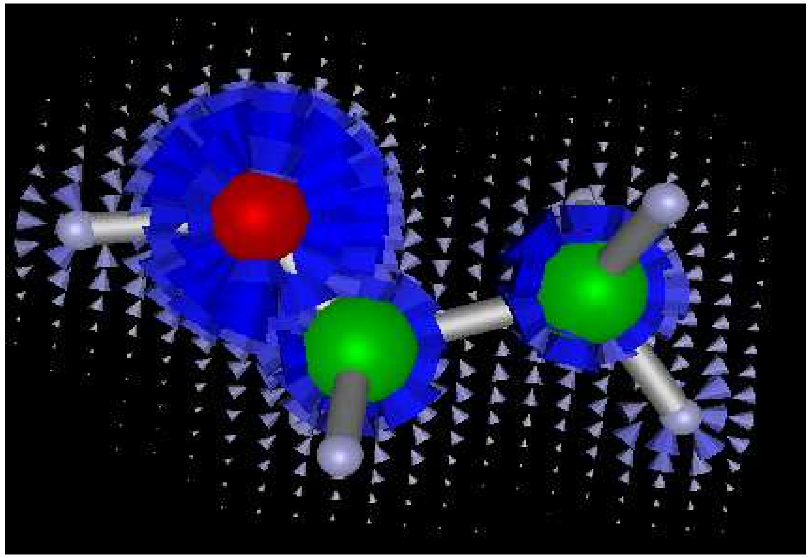

Transparent HOMO


Volume-rendered charge density.

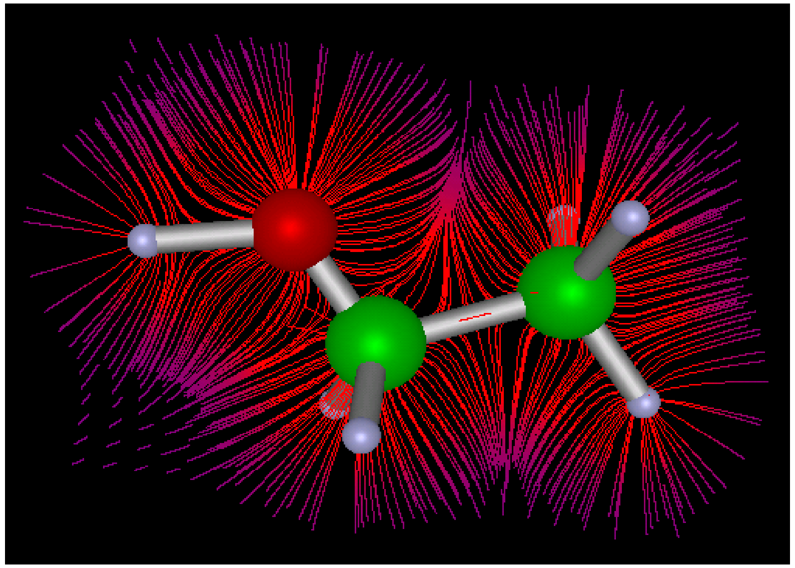
CCLRC Daresbury Laboratory

Visualising Calculation Results

- Currently developing the ability to view vector data (e.g. charge density gradient).
- View vectors as:
  - hedghog plots (lines with length/orientation describing the vector)
  - Glyphs (as above but using cones)
  - Streamlines (follow a particle as it travels through the vector field).



Glyphs



Streamlines

Interactivity and Customisation

- Initialisation
  - source a python file on startup
    - can add new modules, menus etc as well as modify all internal variables
- Interactive Shell
  - Python’s native IDE (IDLE) is a pure Python/Tkinter code
  - We adapted version 0.8 slightly (following approach as PMV from Scripps)
  - Provides useful dynamic extensibility:
    - can type commands into the shell window
    - can access and modify all the data structures in the GUI
    - can open a python source file and execute the contents, we are putting together a collection of samples

CCLRC Daresbury Laboratory

Experiences and Comments

- A big step forward compared to previous experience of scripting (Unix shell scripts and Tcl/Tk)
  - good range of data types
  - ease of incorporating extensions.
- Very few problems with portability
  - still some issues with Tk on MacOSX
- There are a lot of extensions, sometimes it can be a bit of work to satisfy all the requirements of a package
  - there are some useful downloads, python + popular extensions
    - ActiveState Python
      - **Linux, Solaris and Windows, incl expat, zlib**
    - Python Enthought Edition
      - **For Windows, includes VTK**

CCLRC Daresbury Laboratory

Experiences and Comments

- Indenting
  - Curious at first, but works well
  - It can be a nuisance to cut and paste between codes written with different conventions -
    - so choose a standard and stick to it (I use a 4 char indent following pythons own source)
    - see the python Style guide and try and follow its recommendations http://www.python.org/peps/pep-0008.html
- When editing modules, its handy to have a self-test clause at the end:

```
if __name__ == "__main__":
    o = MyObject()
    o.run()
```

  - In an IDE or emacs, executing the buffer while editing it makes for an easy development/test cycle

CCLRC Daresbury Laboratory

Python Futures

- Language is continuing to develop
  - we see no major language deficiencies at the moment
  - the simple code I write works with all versions….
- Most important change is the associated software base
  - Thriving community
  - Strong open source ethos
  - Python bindings for many toolkits appearing
  - quote from Andrew Dalke “well on the way to becoming the high-level programming language for computational chemistry”

Summary

- An easy, attractive language
- Well suited for GUI construction and high-level control of modular programs - as a “glue” language ...

  .. but can also be used to build complicated applications
  - Ideal for prototyping phase
  - Some C or C++ code may be needed as the application matures
- For URLs of the packages referred to in this talk, please see http://www.cse.clrc.ac.uk/qcg/python