Why (and Why Not) to Use Fortran

Instead of C++, Matlab, Python etc.

Nick Maclaren

University of Cambridge Computing Service

nmm1@cam.ac.uk, 01223 334761

June 2012

Domain of Interest

Fortran is used for scientific/numerical computing
And, nowadays, it is used only for such requirements

Still used for such tasks in embedded programming
Things like aircraft controllers, chemical plants

Compare with C++, Python, Matlab, S–Plus etc.
All of which are often used for scientific computing

Warning: personal opinions will abound in this talk
Based on factual reasons, but priorities differ
I favour reliability, portability and parallelism

Coverage

Will cover modern, standard Fortran and C++
Mainly the available C++03 and Fortran 2003
Mentioning latest (and greatest?) 2011 standards

Modern Fortran includes all of Fortran 77 as subset
C++ includes most of C, with some subtle differences

Versions of Python and Matlab less relevant
But essentially Python 2.6+ and Matlab 7+
For Python, will assume you also have numpy

Rationality and Irrationality

Choose a language because they already know it
Or because they are joining a group that uses it
Others need to modify an existing program written in it
Or easy to get programmers for short–term project

$\Rightarrow$ All are good, rational reasons

Some claim that Fortran is an obsolete language
Or that everyone should use C++, Python or whatever

$\Rightarrow$ Those are bad, irrational reasons

More Irrationality

Or that it's taught in computer science courses

$\Rightarrow$ That is also a bad, irrational reason

Why?

Any competent developer can learn a new language
Mentally inflexible people make bad programmers

And computer scientists aren't usually what you need
What you need is to write reliable, practical software

Other Languages

Far too many to enumerate, but mostly irrelevant
Will mention just a few, and relevance here

Let's avoid Excel, Basic and Pascal – please!
Also computer science and experimental ones

Ada, possibly – but I haven't looked at even Ada 95

C is a semi–portable, high–level assembler
Commonly used nowadays for system interfaces etc.

Executive Summary (1)

Any not mentioned is poor to horrible

$\Rightarrow$ This is from viewpoint of practical scientists
Unfair on C++ for skilled, disciplined programmers

Ease of use: Python and Matlab, then Fortran

Prototyping: Matlab, then Python, rarely others
Mainly because high–level and interpreted

Debuggability: Python, Matlab, then NAG Fortran

Executive Summary (2)

Portability: Fortran the best, by a mile

And over–elaborate C++ is by far the worst

Software engineering: Fortran best, then Python

Reasons are complicated, but several of them

Performance: Fortran or C++ (no overall difference)

Python and Matlab IF good toolbox exists

Actually programming them is always slow

Parallelism: Fortran best for shared memory

Little to choose for distributed memory

Executive Summary (3)

Array handling: Fortran best, then Matlab and Python
    Less clear for sparse matrices, or unusual ones

Text handling: Python best, then C++, then Fortran
    Python definitely best for regular expressions

‘Computer science’: C++, then Python and Fortran
    Includes data networks (a.k.a. graph structures)

‘System interfaces’: Python, then C++, then Fortran
    Includes writing multi–program applications

There’s no universally best language, nor ever will be

Matlab, Mathematica, S-Plus etc.

High–level, domain–specific packages
From 1960s in statistics and engineering domains
Usually interactive, but better ones are programmable
$\Rightarrow$ All are largely interpreted languages

Will describe only Matlab, but comments are general
Mathematica is for similar types of application
Genstat, S–Plus, R are for statistical programming
Most domains have at least one, often several

Octave and R are free, others need licences
Some can be expensive, especially Matlab toolboxes

Matlab

Originally a simple language for matrix arithmetic
Can now do most numerical scientific calculations

Very heavily used for scientific/numerical computing
Not very well documented or numerically robust
Quality still better than most open–source code

Matlab has lots of specialist toolboxes
Generally, you need at least some, but cost builds up
High–level (e.g. array operations) is fairly efficient

Octave is a GNU application, very Matlab–like

Matlab Benefits

Can be easier to use than the others if

- you don't know any of the languages, or
- it or a toolbox matches your requirement, or
- you just want to do some prototyping, or
- you don't need immense efficiency

Some of benefits with a Fortran or C++ library!
For example, NAG, Netlib, and many others
$\Rightarrow$ And often get better efficiency, too

Matlab always worth considering for one–off code
E.g. useful for checking results of other code!

Python

A very simple, high–level interpreted language
Started in computer science, and inclined that way
Much easier and better engineered than most
It traps most user errors, including numeric ones

Almost all of its functionality is in library modules
Huge numbers of very useful ones, as standard
Best for scripting, text munging, system interfaces
Scientific programming really needs numpy

I don’t know Ruby, but reported as Python–like
Reported to be a bit cleaner and somewhat slower

Numpy/Scipy

numpy is extensions for scientific programming
Also provides facilities to help calling Fortran
scipy goes a lot further – a bit like Matlab

numpy less conventional than Fortran or Matlab
Not much harder to use than Matlab, but different
Documentation is confusing, though better than C++

Code used to be very poor, but seems better now
Unclear whether numerically robust or how reliable
High–level (e.g. array operations) is fairly efficient

Python Benefits

$\Rightarrow$ Essentially the same as Matlab!

Big difference is if you do a lot of non–numeric coding
Then it's much easier to use Python instead

Reminder: often easier if

- you don't know any of the languages, or
- a module matches your requirement, or
- you just want to do some prototyping, or
- you don't need immense efficiency

Python always worth considering for one–off code

C++

Originally to move C programmers to a higher level
Designed for functionality more than error prevention
Not really very good for scientific programming

Language is very complicated, and hard to learn well
Most people follow recipes – often different ones

Still has C's ‘‘high–level assembler’’ principles
Significant advantages and serious disadvantages

$\Rightarrow$ You can do almost anything you want to
You can bypass all checking if you try, just as in C

C++ Standard Library

Real problems are with library, because of design
Its specification and diagnostics are often baffling
Templates are C are compile–time polymorphism
But very unconstrained – mistakes cause chaos

Standard library is large, but not all that powerful
E.g. 4 classes for vectors; none for n–D arrays
Often have to extend library classes, unnecessarily
Use LAPACK, FFTW, MPI etc. just as for Fortran

Almost all C++ uses an extra major class library
Current dogma is you should always do this

Some Class Libraries

- Boost is a library that provides a lot of classes

Fair checking, but little scientific programming

- CERN ROOT has a hotch–potch of scientific tools

Documentation is both inadequate and erroneous

- CGAL is for computational geometry

And so on …

Often very complicated and idiosyncratic

On most desktop systems, but highly non–portable

Can be nasty for HPC or in long term

OK if they do what you want – but choose carefully

C++ Benefits

Can be easier to use than the others if

- you need your own data structures, or
- you need assembler level coding, or
- there is a suitable library, or
- you need high efficiency, or
- you need to mix in a lot of C

$\Rightarrow$ Main reason is that people think they know it

Can do the same with Fortran, but more tediously
I can't recommend C++ as a first serious language
Much harder to learn well – though not than C!

Fortran

One of 3 remaining original high–level languages
Very strange to people used to C–derived languages

Fortran 90 much higher–level and more modern
Older code still works (even most of Fortran 66)

Standard is about 1/3 size of C++ and much simpler
Standard much most explicit and least ambiguous

$\Rightarrow$ Comparable in power to C++ – just very different

Don't design Fortran and C++ applications same way

Fortran Benefits

Can be easier to use than the others if

- you need to code in parallel, or
- you need serious portability, or
- you are using matrices, or
- you need high efficiency

Can do matrices with Matlab and Python
But operations on elements very slow if using them
C++ depends on library and what you need to do

I teach Fortran scientific programming in 3 days
Not everything, but all many/most programmers need

Running out of Time

Will just skim through various areas
Would be only half–way through if not!

- Low–level numeric coding not a problem

Specialist libraries easiest from Fortran and C++

Software Engineering

- Fortran has by far the best specification

Largely explicit, complete and unambiguous
Needed for portability, reliability and debuggability
⇒ Testing tells you only what this compiler does

- Fortran and Python both have modules

Collect related data, functions and interfaces together
A key feature for good software engineering

- Python and C++ have exceptions, in theory

Mainly useful for resource recovery and similar
Matlab's are undefined and Fortran has none

Error Detection

- Static error detection only in Fortran and C++

The C++ library is the main problematic area
Python or Matlab are dynamically checked

- Dynamic error detection is main problem

Python and NAG Fortran are good, then Matlab
Most Fortrans and all C++s are poor or bad
Some C++ libraries trap most of the simple errors

- Python and Matlab catch all 'SIGSEGVs'

NAG Fortran traps about as much as those two
In Python and Matlab some become logic errors

Optimisation/Efficiency

- Similar when using high–level libraries/modules

At low–level, C++ and Fortran much faster

- Fortran is much more optimisable than C++

C++ must inline across multiple files
Most libraries do it by fiendishly complex templates
Serious problem for portability and reliability

- For most array–based programs, Fortran is fastest

For pointer–based or character, usually C++
Difference usually marginal – may need recoding

Parallelism (1)

- For shared memory, easiest to call SMP library

Possible in all of them, for some algorithms
If you need to code your own, answer is Fortran

- For GPUs, the situation is very murky

There are modules for Python and Matlab
Or can program using CUDA or OpenAcc
From all of C++, Fortran and Python

No time to describe threading – but not advised
Data races cause rare, unrepeatable wrong answers
Scientific programs often suffer very badly from this

Parallelism (2)

- For distributed memory, usually call MPI

Possible in all, easiest in Fortran and C++

- Fortran 2008 has coarrays – a PGAS model

Will they take off? Your guess is as good as mine

- Python 2.6 introduced the multiprocessor module

It's a bit like MPI, but with a different objective

Data Structures

- For arrays, Fortran then Matlab and numpy

numpy arrays as good as Matlab, but different
For sparse or non–rectangular, Matlab may be best

- All have simple structures – with Matlab weakest

- C++ and Python have lists (a.k.a. chains)

All except Fortran have maps (a.k.a. directories)
Anything else needs pointers – can be a bit tedious

Pointers

C++ pointers are very low level and dangerous
Fortran's are very different and higher level
Python's are implicit (in use counts of references)
Matlab is similar, but very unlike normal pointers

Comparing their pointer support is like comparing
apples, blackberries, bananas and acorns ...

Coding pointer–based algorithms easiest in C++
Doing that is tedious but easy in Fortran
$\Rightarrow$ I really cannot recommend Matlab for them

Classes, Object Orientation etc.

- Not much to choose – basic to C++ and Python

But Fortran 2003 and Matlab have them, too
Matlab least flexible, but adequate

- Claim that O–O is always better is pure dogma

Not heavily used or wanted in scientific programming
Little sense for most matrix algebra, for example

- Polymorphism basic to Python and easy

Next easiest in Fortran, but patchily implemented
Heavily used in C++, but with quite a lot of gotchas
Not really relevant to Matlab, or available

Calling Fortran 77, C etc.

- Little problem from C++ or Fortran

C mistakes in Python and Matlab are evil

- Complicated data structures are for experts only

Also mixing Python, Matlab, real C++, real Fortran 90

- System interfaces are nowadays defined in C

Python has most as standard library modules
Other languages call C, but usually not a problem
Risk of conflict with run–time system or parallelism

$\Rightarrow$ But here be dragons!

I/O Facilities

- All truly horrible, but Matlab is worst

Defects wildly different, often misunderstood
Often use another language to do data conversion
Python best for munging text data

- Fortran and C++ I/O are like chalk and cheese

C's I/O seems easy, but is solid with gotchas
Fortran still very restrictive for free–format input
And pretty well every detail is like that ….

- I/O error detection best in Python and Fortran

C++ is worst, because it inherits so much from C