SCAT

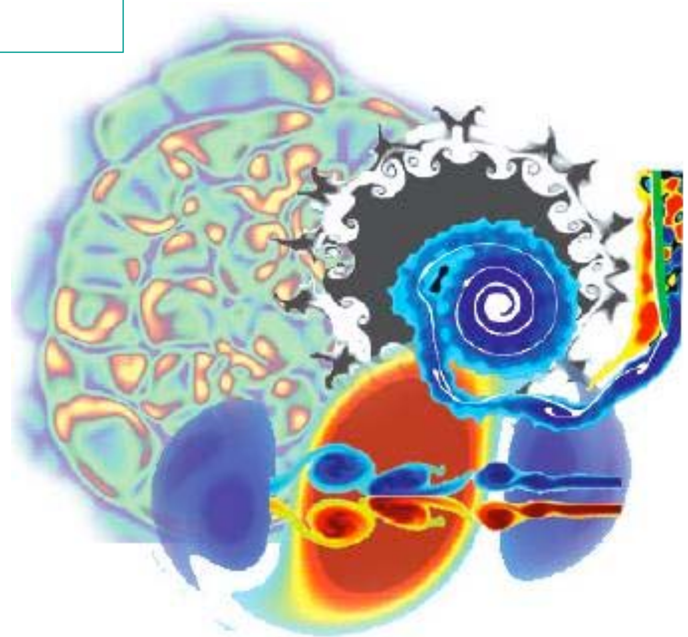
SCIENTIFIC COMPUTING ADVANCED TRAINING

Introduction to Scientific Computing

Two-lecture series for post-graduates,

Dr. Lorena Barba

University of Bristol



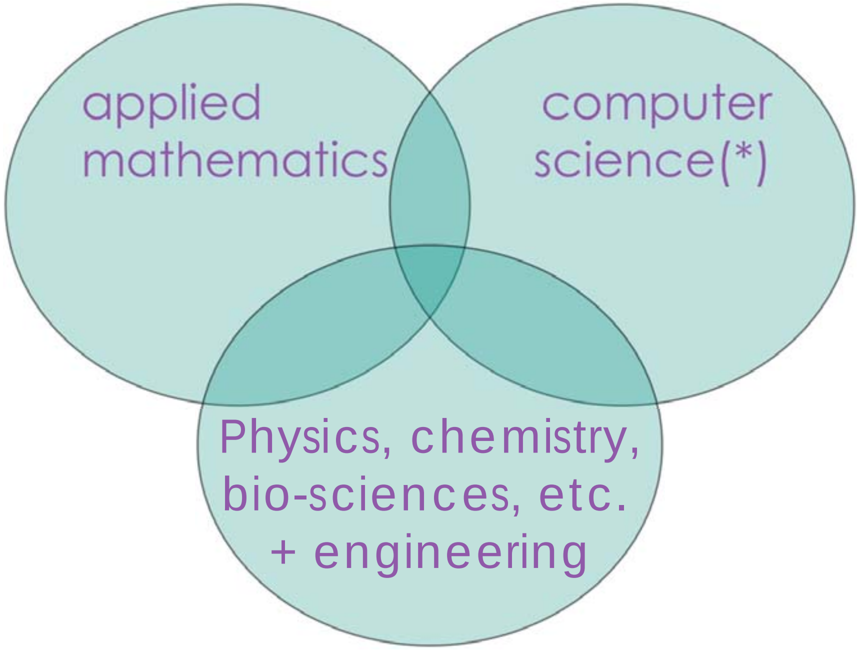
22 and 30 May 2006

Post-graduate lectures
Department of Mathematics

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

What is Scientific Computing?

- Solution of scientific problems using computers
  - It is a multidisciplinary activity ➛
  - “third pillar of science”
    - Next to experiments and analysis
  - Now a necessary avenue for enquiry in almost all fields of science and engineering /



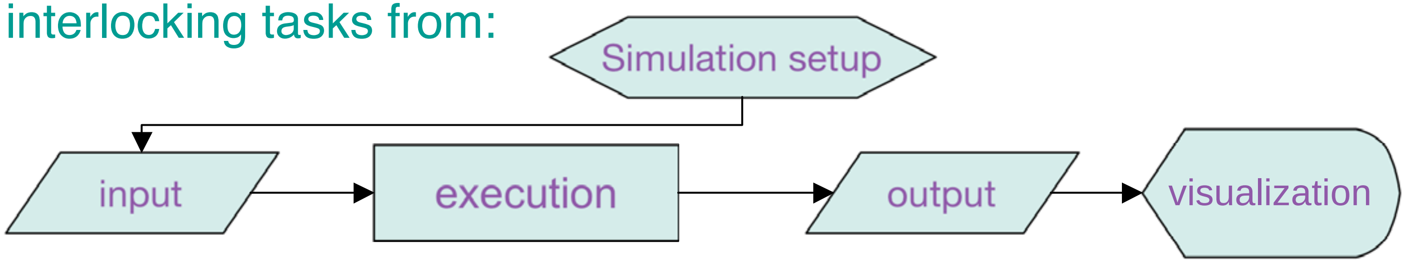
(*) meaning: numerical analysis, algorithm and software development, implementation, execution, profiling, optimizing…
"Computer science is no more about computers than astronomy is about telescopes." Edsger Dijkstra.

What is it NOT?

- All scientists use computers … but scientific computing is NOT:
  - Word processing, type-setting (LaTeX), publication production
  - Email, web browsing, file transfer (FTP, SSH), etc.
  - "Everyday" apps: PowerPoint, Excel, Illustrator, Photoshop

… however *essential* many of these tools may be to the scientist.

- It is NOT *just programming*
  - Workflow of computational science: sum total of all complex and interlocking tasks from:



… to scientific discovery.

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

Building blocks of computational science

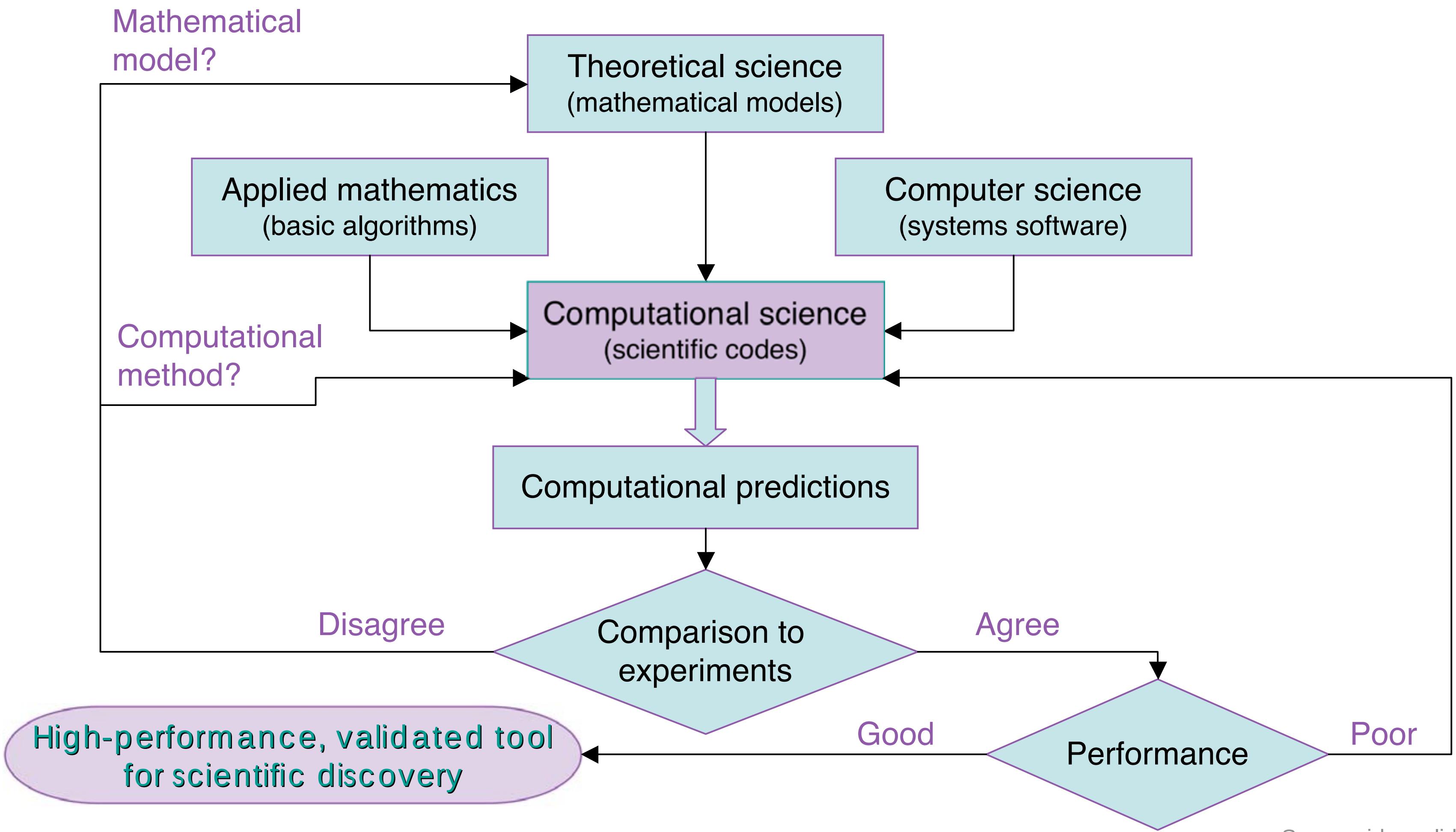
Computational Modelling and Simulation

- **Basic Theoretical Research**
  - lay the foundations for the mathematical description of the phenomena of interest
- **Scientific Code Development**
  - provide the mathematical algorithms, provide the basic computer software, cast equations into algorithms, make efficient use of HPC
- **Scientific Code Validation**
  - benchmark new software
  - compare results with experiments

HPC = High Performance Computing

Source: “Scientific Discovery through Advanced Computing”, Office of Science, US Dept. of Energy March 2000.

Flowchart of the process of scientific computing



Source: idem slide 3

  - In English, "verify", "validate", "confirm" are all synonyms
- As *technical terms* in CSE, they are different:

Verification → solving the equations right

Validation → solving the right equations

- In fact, a "code" cannot be validated, it is verified. A "calculation" can be validated, for a specific class of problems.
- Numerical errors *vs.* conceptual modeling errors
  - *e.g.* assumption of incompressibility in fluid dynamics → modeling

SCAT

SCIENTIFIC COMPUTING ADVANCED TRAINING

Verification

- = "Formal proof of program correctness" [Jay, 1984; IEEE Std. Dict.]
- Verification can and *should* be completed without appeal to physical experiments.
- *As a code builder, I can tell you:*
  - What equations my code solves
  - A theoretical order of convergence for my code
  - The observed order of convergence for a well-behaved problem
  - What grid refinement level was sufficient to attain asymptotic performance on those well-behaved problems
- As a code user,
  - Verification needs to be done again!
  - … for a specific *calculation.*

I cannot tell you what equations you need to solve for *your* problem

I cannot tell you what grid will be required for *your* problem

Verification: order of convergence

- Error of the discrete solution:

$$e = f(\Delta) - f^{exact} = C \cdot \Delta^p + h.o.t.$$

  - For an order-*p* method, and for a well-behaved problem, the error in the solution asymptotically will be proportional to Δ^p, where Δ is *some* measure of discretization (e.g., grid spacing).
    - *p = 2* implies a "second order" method.

- Verification of code:
  - Evaluate the error using an *analytic* solution, or
  - Perhaps use the Method of Manufactured Solutions
  - Monitor the error as the "grid" is systematically refined
    - Grid-refinement study, or grid convergence study.
    - Asymptotic range of convergence: $C = e/\Delta^p$ should become constant

Grid convergence study

- Error: $$e = f(\Delta) - f^{exact} = C \cdot \Delta^p + h.o.t.$$
- Neglecting h.o.t.'s and taking logarithm:
$$\log(e) \approx \log(C) + p \log(\Delta)$$
  - The order of convergence can be obtained from the slope of the curve $\log(e)\ vs.\ \log(\Delta)$
  - More direct evaluation of p: from three solutions using a constant grid-refinement ratio, r :
$$p = \log\left(\frac{f_3 - f_2}{f_2 - f_1}\right) / \log(r)$$
- Obtain the "observed order of convergence"

Verification of calculations

- Verification of calculations → Error *estimation*.
- In order to estimate errors with a systematic grid refinement (or coarsening), we need to know the convergence rate, p.
- Richardson extrapolation:

$$f_{exact} \approx f_1 + \frac{f_1 - f_2}{r^p - 1}$$

- The refinement does not have to be an integer: $r = \Delta_2 / \Delta_1$
- Fractional error on the fine grid: $\epsilon_1 = \frac{f_1 - f_{exact}}{f_{exact}}$

Source: many papers by P. J. Roache.

Validation

- Validation has highest priority to scientists and engineers because "nature" is the final jury.
  - But experimental data is not absolute (and sometimes does not agree with *other* experiments)
- Validation is *ongoing* (as experiments are improved, etc.)
- Careful with "false invalidation"!

"No one believes numerical results, except the author of the calculation. Everyone believes the experimental results, except the one who performed the experiment.

Basics of all Scientific Computing: Numerical Analysis

Most scientific computing deals with the same types of problems: ordinary and partial differential equations, systems of linear equations, vector and matrix operations, interpolation of functions, etc.

Do not try to reinvent the wheel!

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

- dot product
- cross product
- vector norm
- scalar multiplication
- sum row elements
- get maximum/minimum
- vector-matrix multiply
- matrix product
- matrix transpose
- matrix norm
- etc.

SCIENTIFIC COMPUTING ADVANCED TRAINING

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

given

$$(x_i, y_i),\ i = 1 \cdots n$$

Find a reasonable function $f(x)$ such that,

$$f(x_i) = y_i$$

- Simplest: line segments
- polynomial interpolation
- Lagrange interpolation
- Chebyshev polynomials
- Rational polynomials
- Fourier interpolation
- cubic-splines
- B-splines
- surface interpolation

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

$$\left(\frac{\partial f}{\partial x}\right)_i \approx \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$

$$\left(\frac{\partial f}{\partial x}\right)_i \approx \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

$$\left(\frac{\partial f}{\partial x}\right)_i \approx \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}}$$

- Forward difference (FD)
- Backward difference (BD)
- Central difference (CD)

Plus, methods for higher derivatives,mixed derivatives, enforcing BC's

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

• Find approximate value of

$$\int_a^b f(x)dx$$

given $f(x_i),\ x_i$

- Rectangle rule
- Trapezoidal rule
- Newton-Cotes formulas
- Romberg integration
- Gauss quadrature

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

Linear systems:

- Find a vector $x \in \mathbb{R}^n$ so that, $Ax = b$ where, $b \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$
- Gaussian elimination: $O(n^3/3)$ operations
- Pivoting
- Iterative methods: Jacobi, Gauss-Seidel, Successive-Over-Relaxation (SOR), Conjugate Gradient, and many more…

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

Fourier transform:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt$$

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{-2\pi i f t} df$$

- Discretely sampled data:

$$h_k \equiv h(t_k), \qquad t_k \equiv k\Delta,$$
$$k = 0, 1, 2, \ldots, N-1$$

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt$$
$$\approx \sum_{k=0}^{N-1} h_k \, e^{2\pi i f_n t_k} \Delta$$

- Aliasing
- Slow FT: $O(N^2)$
- Fast FT: $O(N \log_2 N)$

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

- Find one or more solution vectors $x \in \mathbb{R}^n$ such that: $f(x) = 0$

  where, $f : \mathbb{R}^n \to \mathbb{R}^n$
- Newton's method

$$x_{k+1} = x_k - [f'(x_k)]^{-1} f(x_k)$$

x_0 Initial guess.

- Minimization:

$$\min \{f(x)\}$$

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

$$\frac{dy}{dt} = f(y,t)$$

$$y_0 = y(t_0)$$

- Forward Euler

$$y_{n+1} = y_n + hf(y_n, t_n)$$

- Backward Euler
- Trapezoidal, Crank-Nicholson
- Leapfrog (multi-step)
- Runge-Kutta schemes
- Adams-Bashford

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools →
  - Mean, variance, skewness
  - Linear correlation
  - Random number generation
  - Monte-Carlo
- *Understanding errors, convergence, stability!*

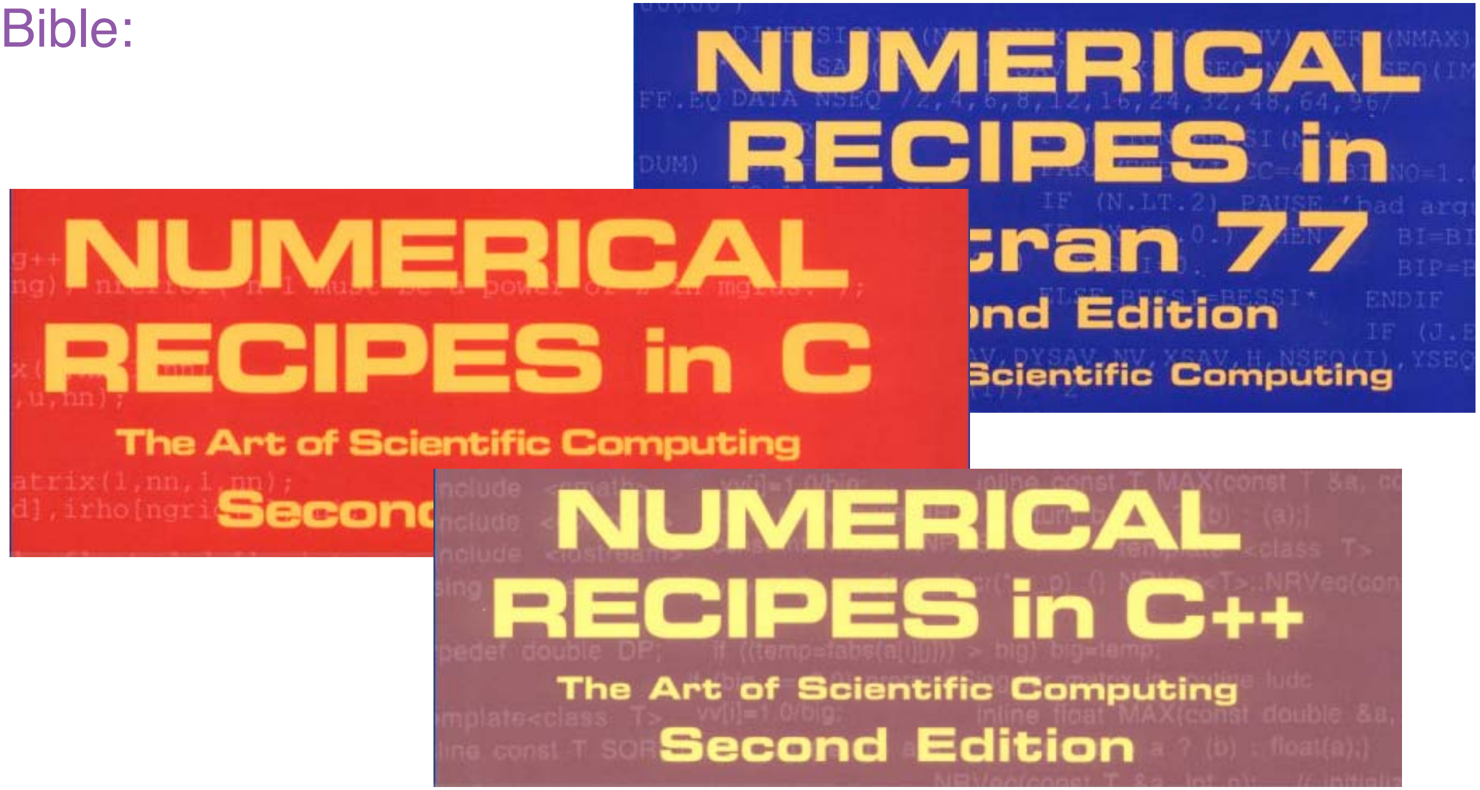
Basic numerical toolbox

- Vector and matrix operations
- Function interpolation
- Finite difference approximations to derivatives
- Integration of functions
- Solving systems of linear equations
- Discrete Fourier transforms
- Nonlinear equations and optimization
- Time stepping methods for ODE's.
- Stochastic tools
- *Understanding errors, convergence, stability!*

- Modelling
- Round-off error
- Truncation error
- Consistency
- bad initial guess!
- ill-conditioning
- bad mesh … etc.

How to learn to implement numerical methods?

- The Bible:



http://www.nr.com/

Which programming language to use?

- C++ versus Fortran:
  - C++ has attractive features for scientific computing:
    - Templates for generic programming
    - Operator overloading for expressiveness
    - Object-orientation for abstraction and code re-use
  - Why Fortran? Fortran is 30 years old!
    - Advantage: lots and lots of “legacy” code
  - Until the early 90s, Fortran was a lot faster for number crunching
    - Benchmarks showed C++ much slower, between 20% to 10 times!
  - Performance of C++ has increased markedly!
    - Better optimizing C++ compilers
    - New library techniques

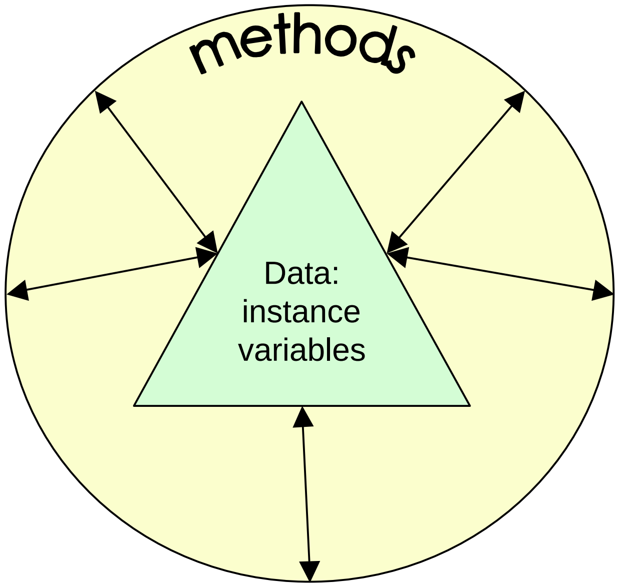
⇒ Really *think* about it before settling for good ol’ Fortran!

Source: “Scientific Computing: C++ versus Fortran”, T. Veldhuizen, Dr. Dobb’s Journal (Nov. 1977)

What is Object-Orientation?

- Essential properties for object-oriented software:
  1. Data encapsulation / abstraction
     - Encapsulation: each object hides its internal structure from the rest of the system. “Hiding the implementation” → an object, once fully tested, is guaranteed to work ever after.
  2. Class hierarchy and inheritance
     - **Class**: description of all properties of all **objects** of the same **type**. Properties can be structural (static) or behavioral (dynamic).
       - Static properties are described by instance **variables**. Dynamic properties are described by **methods**.
     - Inheritance: ability to derive the properties of an object from those of another (the superclass)
  3. Polymorphism
     - Ability to manipulate objects from different classes, not necessarily related by inheritance through a **common set of methods**.

Objects -- encapsulation



State → Private

Behavior → Public

- the user need never peek inside the object
- messages define the interface to the object

OOP : code and data are merged into a single indivisible thing — an object.

Objects $\Rightarrow$ maintainability

Inheritance $\Rightarrow$ reuse

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

Example: vectors and matrices

- Perhaps the most fundamental module needed by any scientific code are classes for storing and manipulating arrays of numbers.
  - Neither C nor Fortran provide good abstractions for arrays
    - In C, you can't use arrays as a first-class object
    - Fortran lacs the capabilities for dynamic resizing of arrays
    - Neither language lets you express operations on arrays in a high-level fashion -- you must use explicit loops to do even basic computations
  - Even if you use no classes beyond a set of vector, matrix and array types, a good array class library can make the transition from C or Fortran worthwhile
- Polymorphism:
  - a function will behave differently depending on the type of object invoking it:
  - This code : `w = x + y + z;`
    might represent summation of three vectors, or matrices, or scalars

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

Shortcut to numerical bliss

- MATLAB
  - Example: solving a linear system of equations $Ax = b$
  - in MATLAB, use the backslash operator:

```
x = A\b;
```

  - … and get the answer!
  - … get a warning if the matrix is nearly singular
  - … get a least squares solution if an exact solution does not exist
- But … for challenging real-life problems… one can quickly run out of memory or cpu … so,

Use MATLAB for algorithm development in smaller problems, then, once the algorithm works, translate to C/C++ with MPI

Software packages for scientific computing

- *“Matlab is too expensive!”*
  - Use Octave instead … it’s free!
    http://www.gnu.org/software/octave/
- *“Matlab is too haaaard!”*
  - First, read the “Help”, but if you still have troubles … ask here: comp.soft-sys.matlab
- Other useful packages:
  - Scilab: http://www.scilab.org/
    comp.soft-sys.math.scilab

Numerical Libraries

Before writing one line of code, find out what libraries are available to help with your problem!

Again: don't reinvent the wheel!

SCAT
SCIENTIFIC COMPUTING ADVANCED TRAINING

NETLIB

- Began in 1985 for cost-effective, timely distribution of freely available, high-quality mathematical software.
- Collection has grown to include other software: networking tools, tools for visualization of multi-processor performance data; technical reports and papers, information about conferences… and more!
- http://www.netlib.org/
- Traditional numerical analysis areas:
  – Linear systems, eigenvalue problems, quadrature, nonlinear equations, differential equations, optimization
- BLAS (Basic Linear Algebra Subprograms)
  – High quality "building block" routines for performing basic vector and matrix operations. Level 1 BLAS do vector-vector operations, Level 2 BLAS do matrix-vector operations, and Level 3 BLAS do matrix-matrix operations.
  – Because the BLAS are efficient, portable, and widely available, they're commonly used in the development of high quality linear algebra software, LINPACK and LAPACK for example.

SCIENTIFIC COMPUTING ADVANCED TRAINING

More numerical libraries

- ScaLAPACK
  - a subset of LAPACK routines redesigned for distributed memory parallel computers.
- ATLAS
    - Automatically Tuned Linear Algebra Software
  - to provide portably optimal linear algebra software. The current version provides a complete BLAS interface for both C and Fortran77) and a very small subset of LAPACK.
- List of free Linear Algebra software : http://tinyurl.com/jhtfm

Modern Algorithms

The development of new numerical algorithms is crucial, and leverages huge hardware investments.

Top 10 Algorithms of the 20th Century

- 1946: The Monte Carlo method.
- 1947: Simplex Method for Linear Programming.
- 1950: Krylov Subspace Iteration Method.
- 1951: The Decompositional Approach to Matrix Computations.
- 1957: The Fortran Compiler.
- 1959: QR Algorithm for Computing Eigenvalues.
- 1962: Quicksort Algorithms for Sorting.
- 1965: Fast Fourier Transform.
- 1977: Integer Relation Detection.
- 1987: Fast Multipole Method.

Dongarra & Sullivan, IEEE Comput. Sci. Eng., Vol. 2(1):22--23 (2000).