*Dr Barry Wardell*

# ACM40290 Numerical Algorithms

UCD School of Mathematics and Statistics

Semester 1 2017/18

# Contact Details

- Dr Barry Wardell
  UCD School of Mathematics and Statistics
- Room S1.76 Science Centre South
- Ext. 2543
- barry.wardell@ucd.ie

# ACM40290: Numerical Algorithms

- Schedule
  - 2 lectures per week:
    11AM Mondays (J109 Arts), 12PM Wednesdays (**232 SCN** Science North)
  - 1 tutorial slot, 12PM Thursdays (H1.51 Science Hub). Will only run approx every second week, starting after the first assignment is due.
  - Office hours: on demand - email/ask in class to arrange time
  - Notes available on Blackboard
- Assessment
  - 6 assignments - 30% - not all equally weighted as some are longer than others
  - Final exam - 2 hours - 70%

# Important Orientation Information

- Applications for Extenuating Circumstances
  - These refer to very grave issues that occasionally arise such as
    - Serious illness, hospitalisation, an accident
    - Family bereavement (parent, sibling)
    - Ongoing serious personal or emotional circumstances.
  - Extenuating Circumstances **do not** cover events which are **foreseen** (e.g. 21st party, Debs ball, wedding etc)
- Minor Circumstances (absent for a few days)
  - These situations should be handled locally by making direct contact with the lecturer/relevant School. Extenuating Circumstances do **NOT** apply in these cases.

# Important Orientation Information

- Missing a Lecture, Lab session or Tutorial
  - Contact lecturer about making it up.
- Late submission of Coursework
  - Where coursework is submitted late due to unanticipated exceptional or extenuating circumstances, students should follow procedures under the **Policy on Late Submission of Coursework**: http://www.ucd.ie/registry/academicsecretariat/docs/latesub_po.pdf. **An application for Extenuating Circumstances is not appropriate in this case.**

# Important Orientation Information

- Plagiarism
  - Plagiarism is a **serious academic offence**. While plagiarism may be easy to commit unintentionally, it is defined by the act not the intention.
  - All students are responsible for being familiar with the University's policy statement on plagiarism and are encouraged, if in doubt, to seek guidance from an academic member of staff.
  - The University encourages students to adopt good academic practice by maintaining academic integrity in the presentation of all academic work.
  - For more detailed information see: http://www.ucd.ie/governance/resources/policypage-plagiarismpolicy/

| Grade | Low | High |
|---|---|---|
| A+ | 90.00 | 100 |
| A | 80.00 | 89.99 |
| A- | 70.00 | 79.99 |
| B+ | 66.67 | 69.99 |
| B | 63.33 | 66.66 |
| B- | 60.00 | 63.32 |
| C+ | 56.67 | 59.99 |
| C | 53.33 | 56.66 |
| C- | 50.00 | 53.32 |
| D+ | 46.67 | 49.99 |
| D | 43.33 | 46.66 |
| D- | 40.00 | 43.32 |
| E+ | 36.67 | 39.99 |
| E | 33.33 | 36.66 |
| E- | 30.00 | 33.32 |
| F+ | 26.67 | 29.99 |
| F (FM) | 23.33 | 26.66 |
| F- | 20.00 | 23.32 |
| G+ | 16.67 | 19.99 |
| G | 13.33 | 16.66 |
| G- | 0.02 | 13.32 |
| NG | - | 0.01 |

# Survey

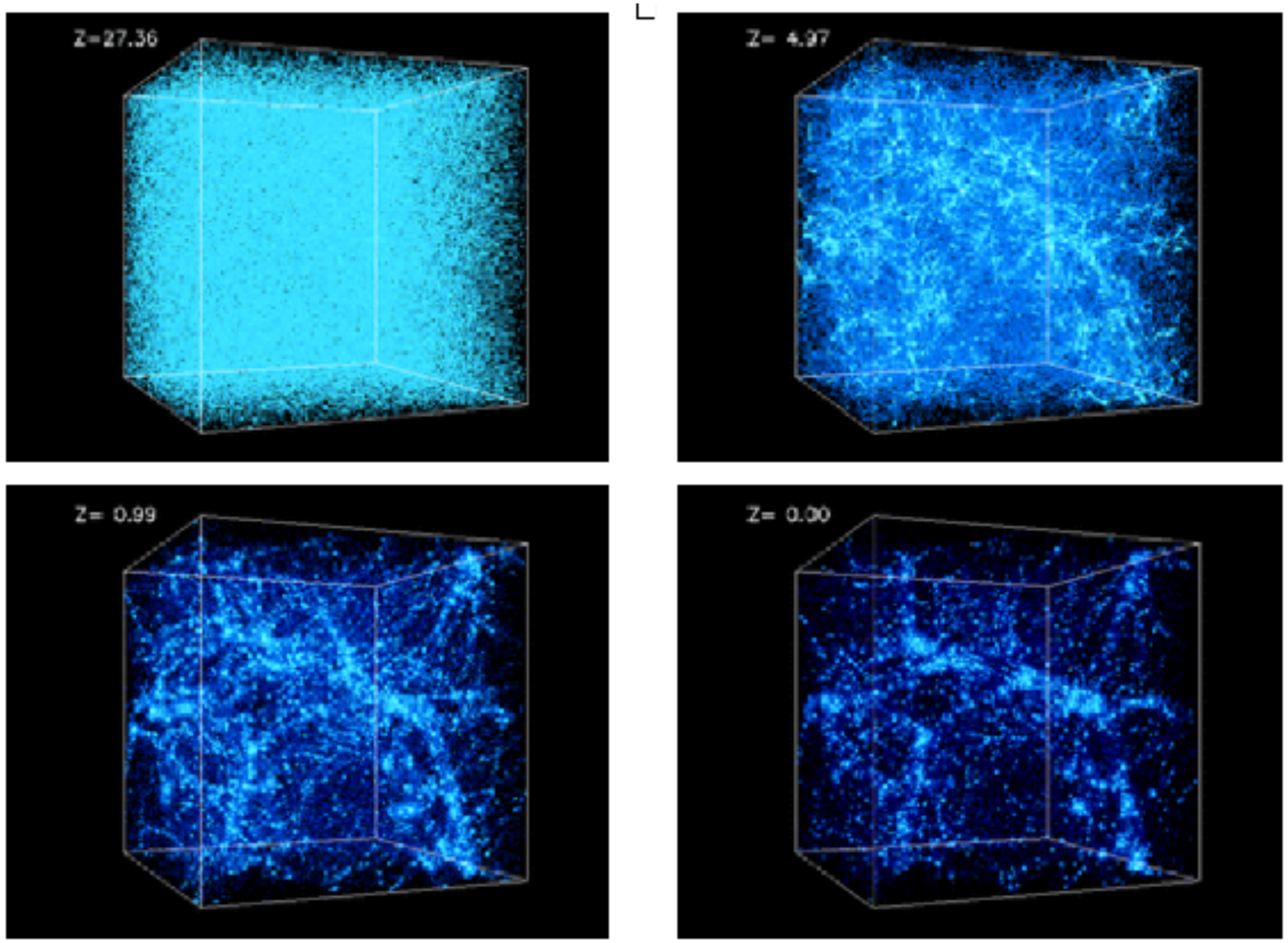
- Who has:
  - Taken a course in scientific computing?
  - Taken a course in numerical algorithms?
  - Used MATLAB? Octave? Python?
  - A background in Mathematics? Computer Science? Engineering? Physics?

# Scientific Computing

- Computation is now recognised as the “third pillar” of science (along with theory and experiment).
- Why?
  - **Computation allows us to explore theoretical/mathematical models when those models can’t be solved analytically.**
  - This is **usually** the case for real-world problems!
  - E.g. Navier–Stokes equations model fluid flow, but exact solutions only exist in a few simple cases.
  - Advances in algorithms and hardware over the past ~50 years have steadily increased prominence of scientific computing.

# Scientific Computing

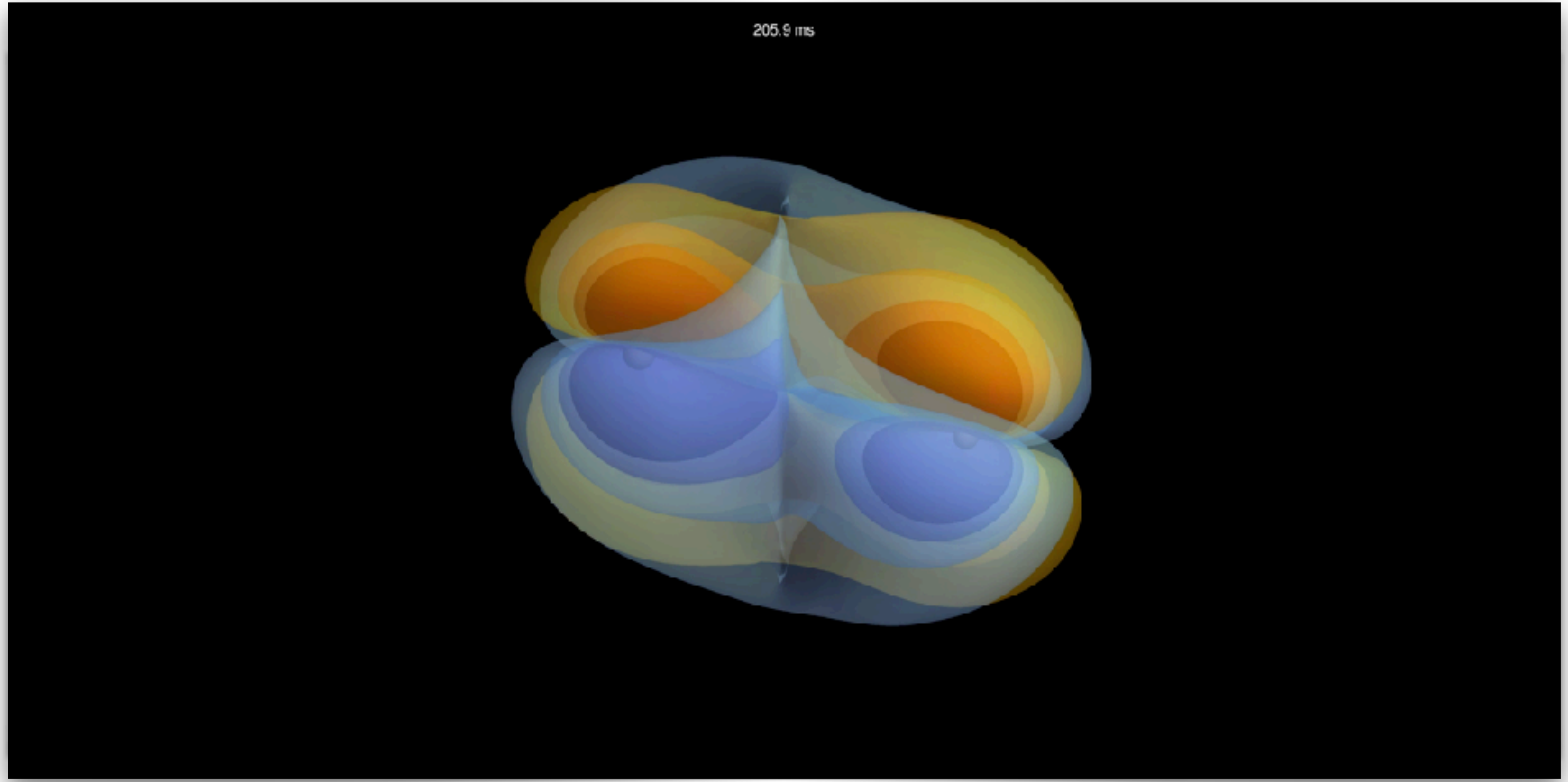
- Computation is now very prominent in many different branches of science.
- For example…




*cosmicweb.uchicago.edu*

# Scientific Computing: Cosmology

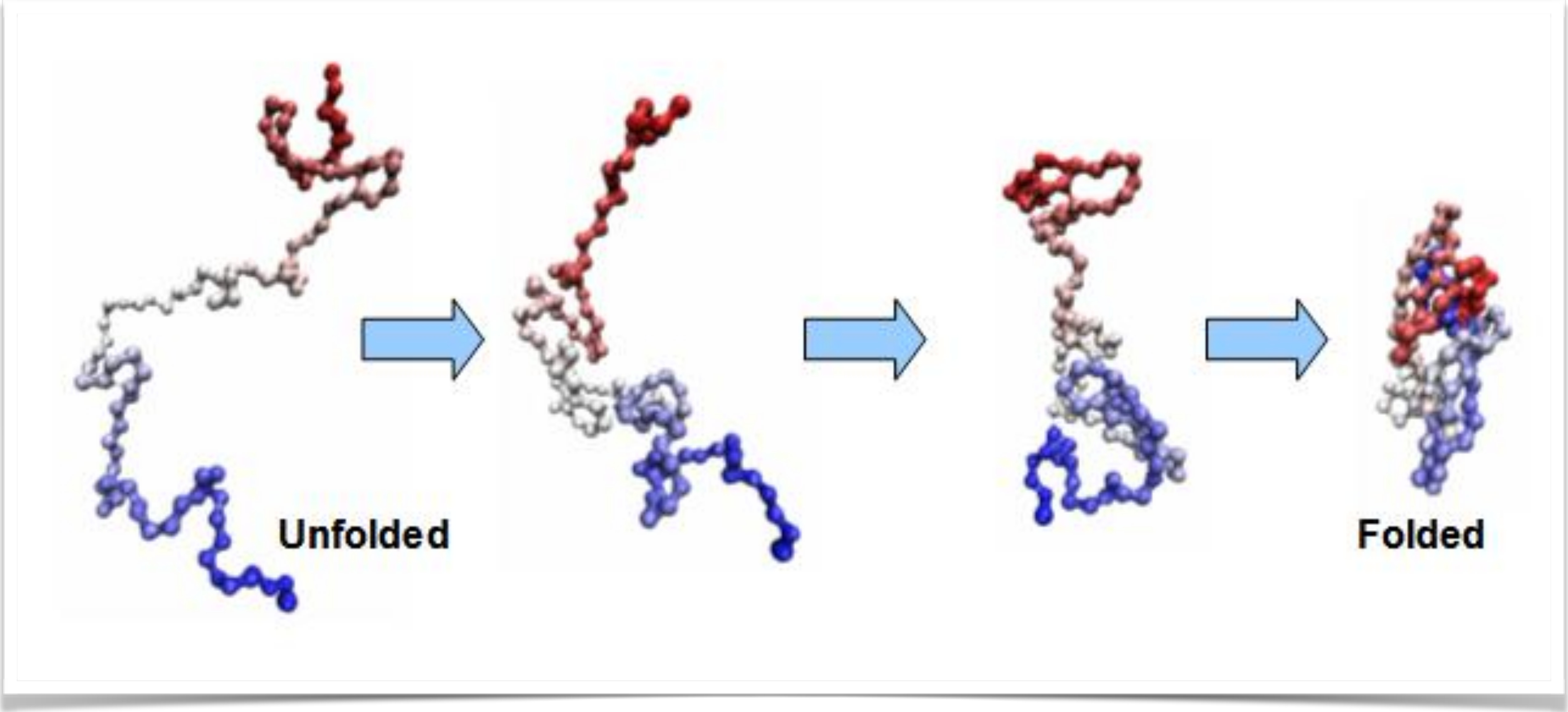
Cosmological simulations allow researchers to test theories of galaxy formation




*Barry Wardell*

# Scientific Computing: Astrophysics

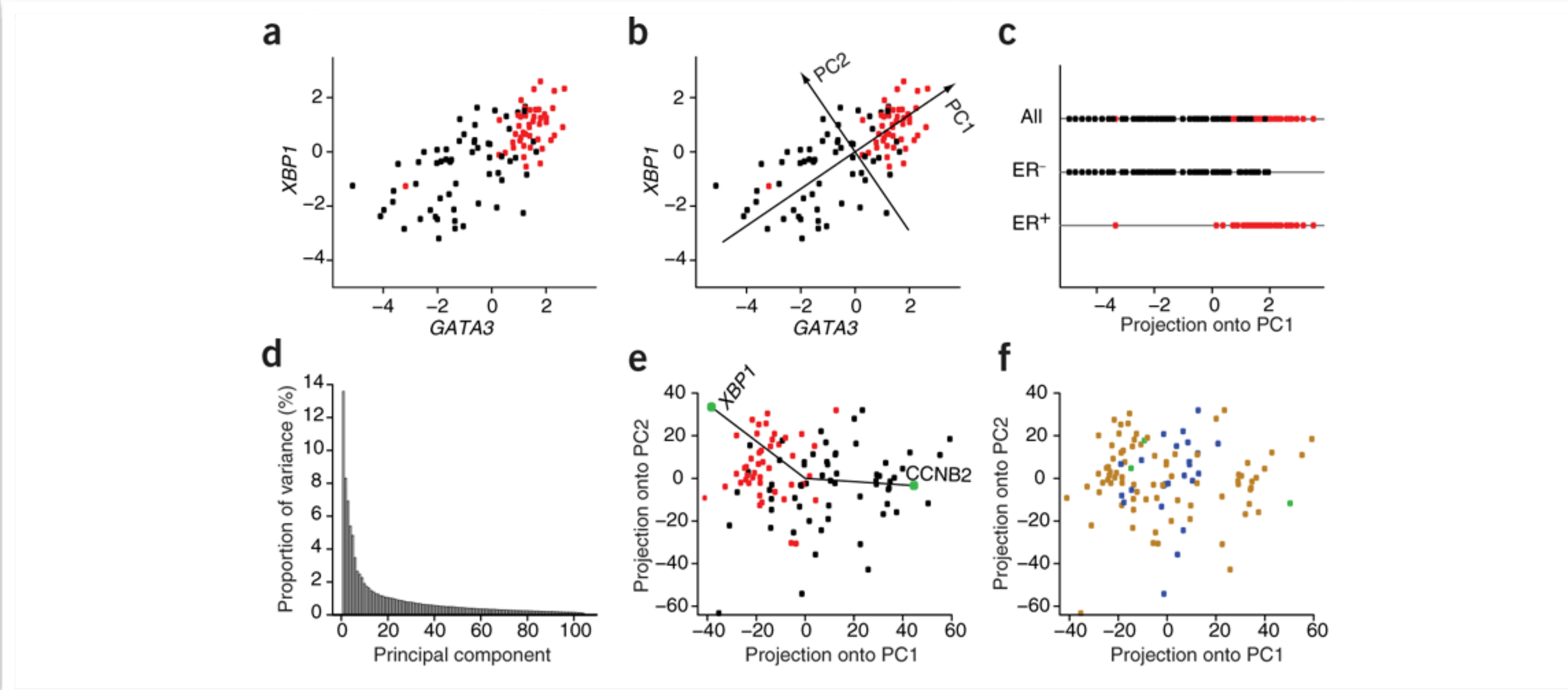
Simulations of the merger of a pair of black holes were crucial to the detection of gravitational waves by LIGO.




*cnx.org*

# Scientific Computing: Biology

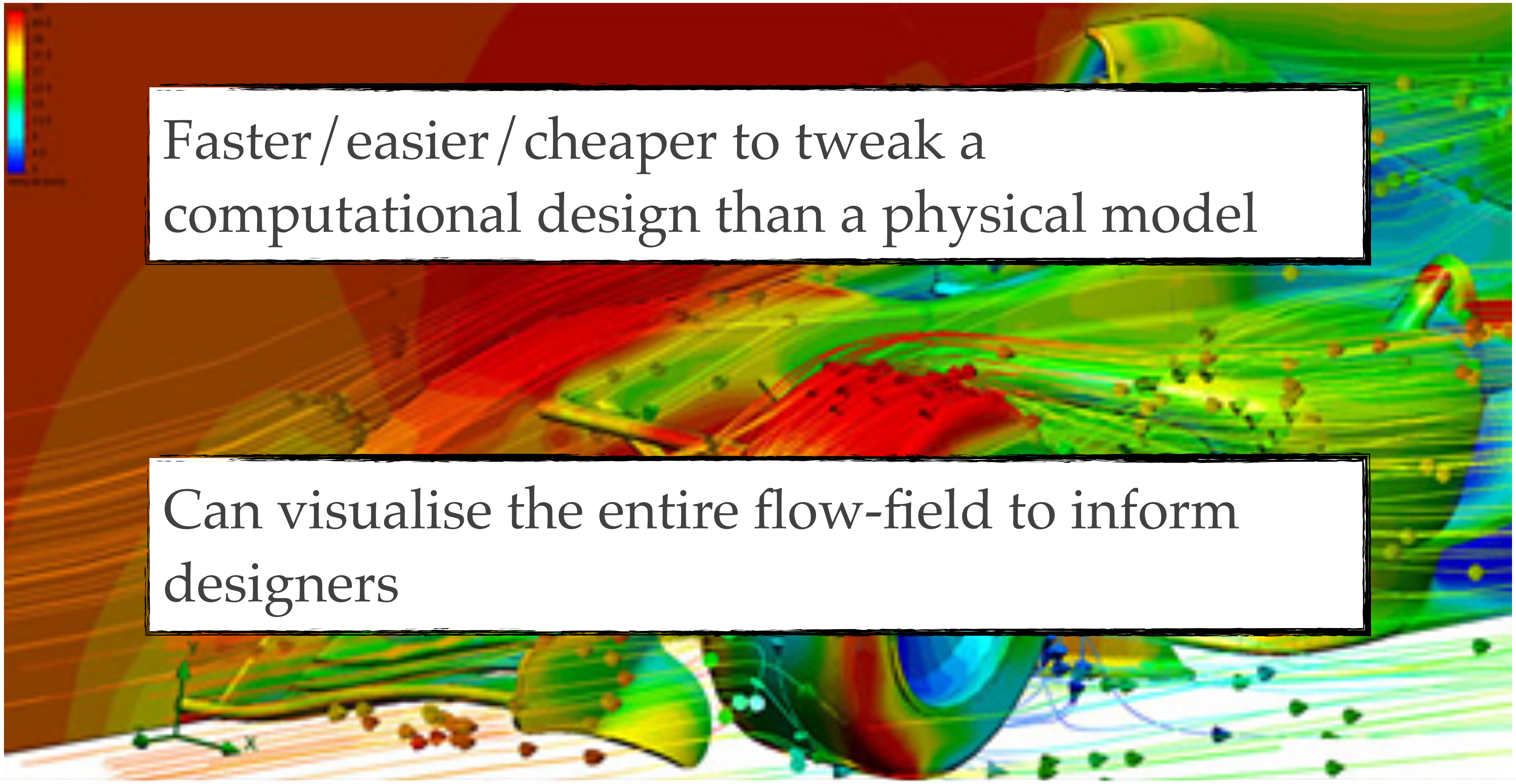
Scientific computing is now crucial in molecular biology, e.g. protein folding




*Nature Biotechnology, 2008*

# Scientific Computing: Biology

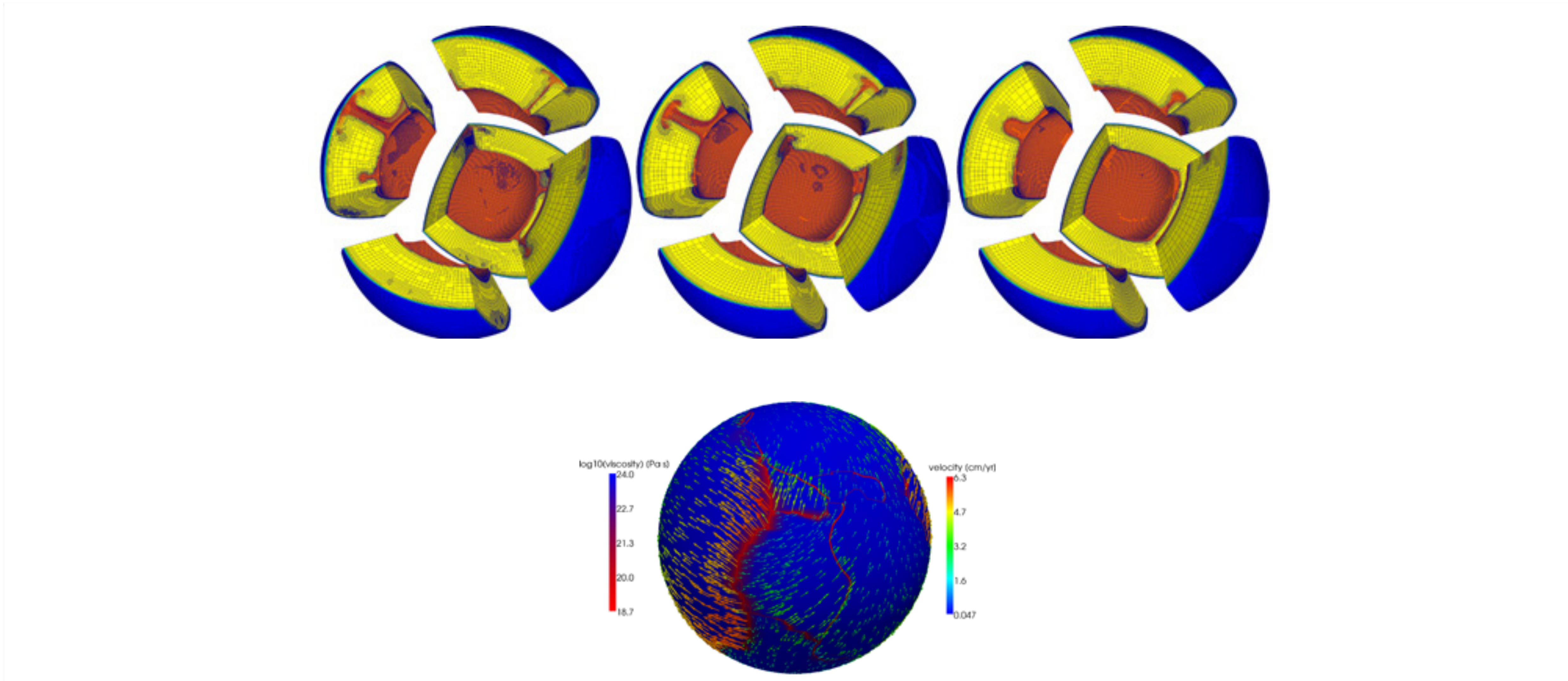
Or statistical analysis of gene expression




*www.mentor.com*

# Scientific Computing: Computational Fluid Dynamics

Wind-tunnel studies are being replaced and / or complemented by CFD simulations




*www.tacc.utexas.edu*

# Scientific Computing: Geophysics

- In geophysics we only have data on (or near) the Earth's surface.
- Computational simulations allow us to test models of the interior.

# What is Scientific Computing?

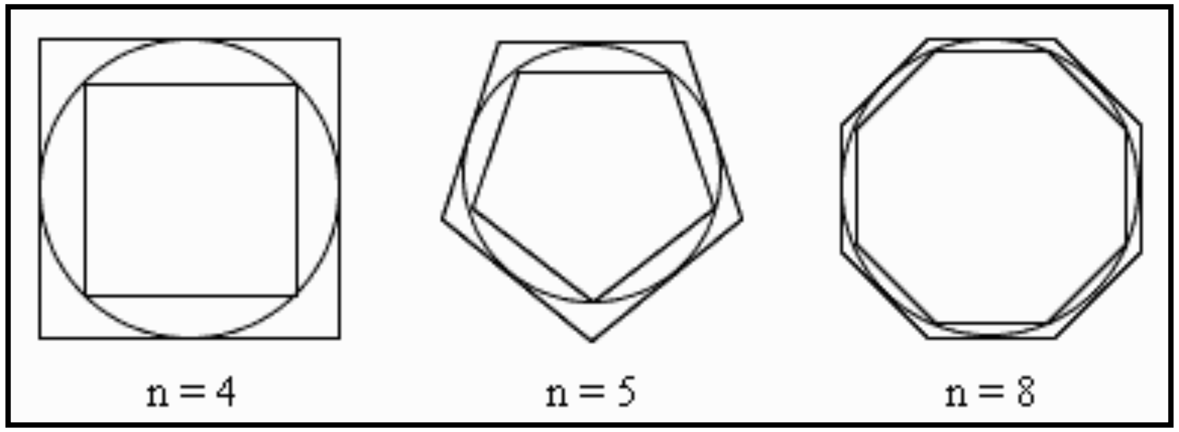
- Scientific Computing (SC) is closely related to Numerical Analysis (NA)

> "Numerical Analysis is the study of algorithms for the problems of continuous mathematics"
> Nick Trefethen, SIAM News, 1992.

- NA is the study of these algorithms, SC emphasises their application to practical problems
- Continuous mathematics $\Rightarrow$ algorithms involving real (or complex) numbers, as opposed to integers
- NA/SC is quite distinct from Computer Science, which usually focuses on discrete mathematics (graph theory, cryptography, ...)

# What is Scientific Computing?

- NA/SC have been important subjects for centuries! (Though the names we use today are relatively recent…)
- One of the earliest examples: Archimedes (287–212 BC) approximation of $\pi$ using n = 96 polygon




- Archimedes calculated that $3\,\frac{10}{71} < \pi < 3\,\frac{10}{70}$, an interval of 0.00201

# What is Scientific Computing

- Key Numerical Analysis ideas captured by Archimedes:
  - Approximate an infinite/continuous process (integration) by a finite/discrete process (polygon perimeter)
  - Error estimate ($3\frac{10}{71} < \pi < 3\frac{10}{70}$) is just as important as the approximation itself

# What is Scientific Computing

- We will encounter algorithms from many Great Mathematicians: Newton, Gauss, Euler, Lagrange, Fourier, Legendre, Chebyshev, …
- They were practitioners of scientific computing (using "hand calculations"), e.g. for astronomy, mechanics, optics,…
- And were very interested in accurate and efficient methods since hand calculations are so laborious

# Scientific Computing vs Numerical Analysis

- SC and NA are closely related, each field informs the other

| |
|---|
| We focus on knowledge required for you to be a responsible user of numerical methods for practical problems |

# Course Layout

- MATLAB programming:
  - data types and structures
  - arithmetic operations
  - functions
  - input and output
  - interface programming
  - graphics
  - implementation of numerical methods
- Introduction to numerical computing:
  - Approximation
  - finite floating point arithmetic
  - catastrophic cancellation
  - chopping and rounding error
  - discretisation error
  - convergence

# Course Layout

- Solution of nonlinear equations:
  - bisection method
  - secant method
  - Newton's method
  - fixed point iteration
  - Muller's method
- Numerical optimization:
  - Method of golden section search
  - Newton's method optimization

# Course Layout

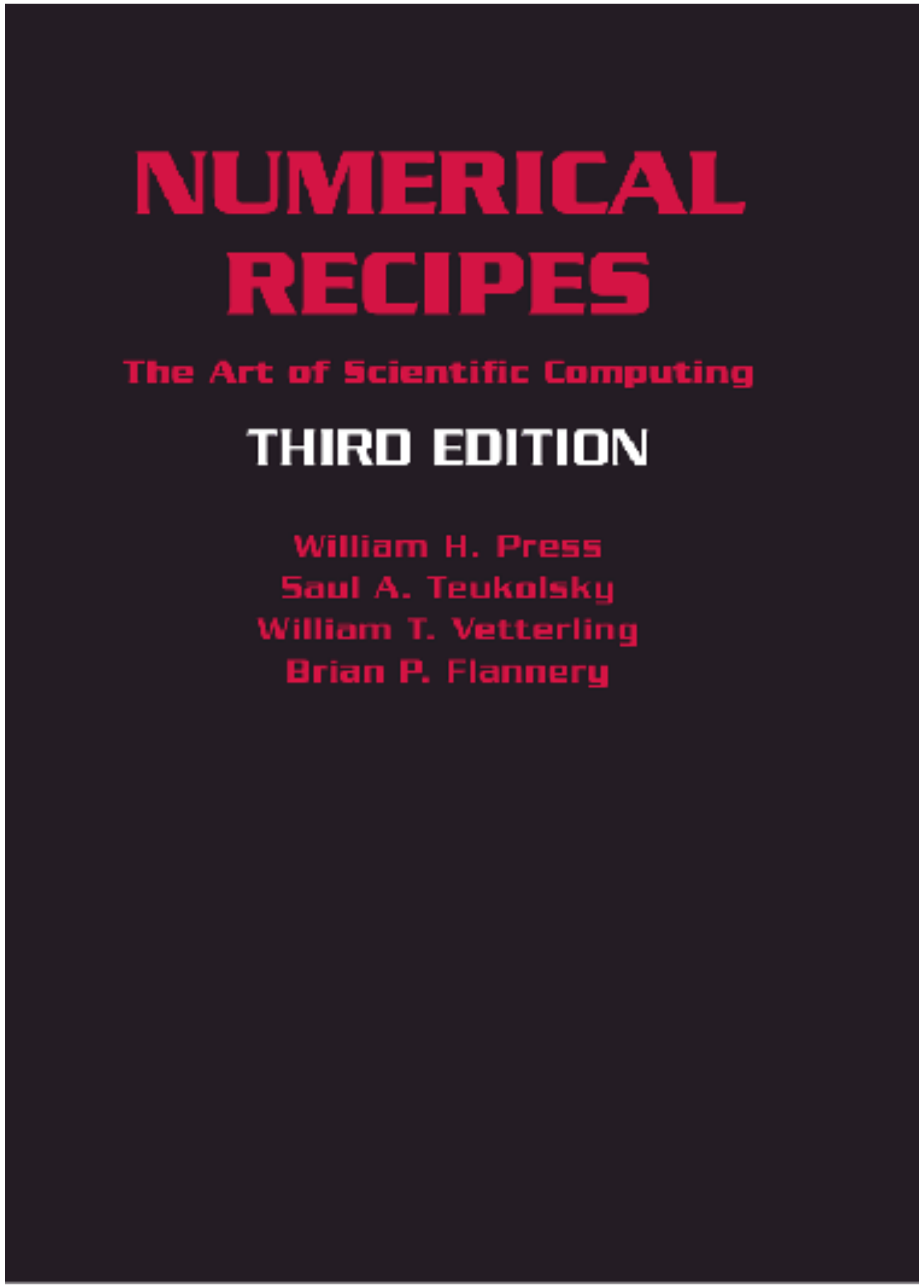
- Solutions of linear algebraic equations (i.e. matrix equations)
  - forwarding Gaussian elimination
  - pivoting
  - scaling
  - back substitution
  - LU-decomposition
  - norms and errors
  - condition numbers
  - iterations
  - Newton's method for systems
  - computer implementation
- Interpolation:
  - Lagrange interpolation
  - Newton interpolation
  - inverse interpolation.

# Course Layout

- Numerical Integration
  - finite differences
  - Newton cotes rules
  - trapezoidal rule
  - Simpson's rule
  - extrapolation
  - Gaussian quadrature.
- Numerical solution of ordinary differential equations
  - Euler's method
  - Runge-Kutta method
  - multi-step methods
  - predictor-corrector methods
  - rates of convergence
  - global errors
  - algebraic and shooting methods for boundary value problems

# Textbooks and Other Resources

- Very good resource for understanding algorithms, finding a starting point to point you in the right direction for solving a particular problem.
- Don't use the code/implementations; only average quality and importantly licensing is very restrictive ⇒ very difficult to share/reuse code.



# Textbooks and Other Resources


1. Hager, William W, *Applied Numerical Linear Algebra*, Prentice-Hall, 1988

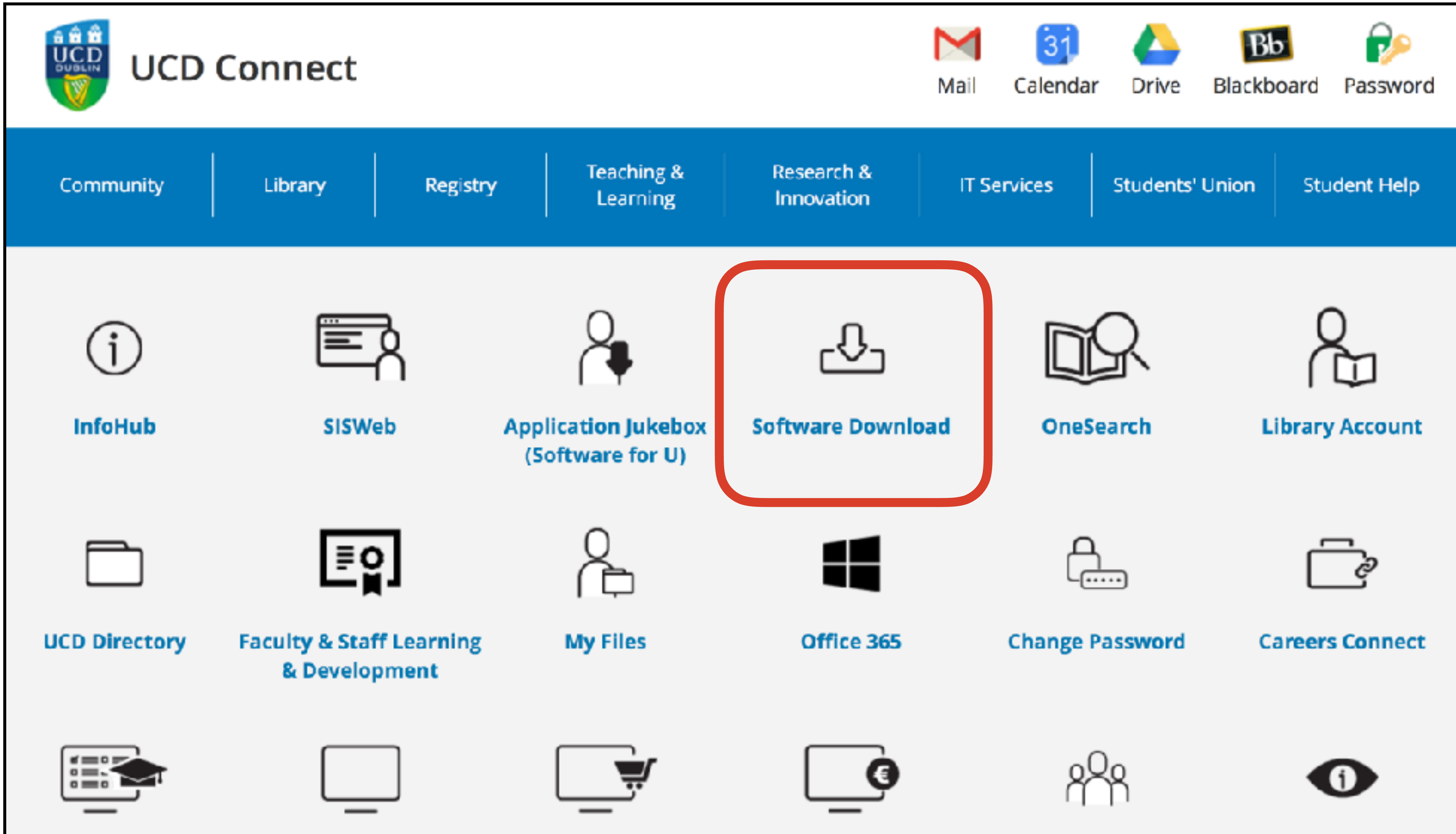
2. Forsythe, G., Malcolm, M., and Moler, C., *Computer Methods Mathematical Computations*, Prentice-Hall, 1977

3. Kahaner, D., Moler, C, and Nash, S., *Numerical Methods and Software*, Prentice-Hall, 1989

# Textbooks and Other Resources

- Trefethen's Essays (http://people.maths.ox.ac.uk/trefethen/essays.html). Short and not very technical.
  - Maxims about Numerical Mathematics, Computers, Science, and Life (SIAM News, Jan/Feb 1998)
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/maxims.html
  - Numerical Analysis (Princeton Companion to Mathematics, to appear), Oxford University, March 2006
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/NAessay.pdf
  - The Definition of Numerical Analysis
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/defn.ps.gz
  - Predictions for Scientific Computing 50 years from now.
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/future.ps.gz
  - Who Invented the Great Numerical Algorithms.
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/inventorstalk.pdf
  - Ten Digit Algorithms — "Ten digits, five seconds, and just one page".
    http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/tda.html

# Getting MATLAB



# Module Focus

- Solve the right problem $\Rightarrow$ make sure your model is right
- Solve the problem right $\Rightarrow$ use the best methods
- Don’t reinvent the wheel $\Rightarrow$ be aware of the existence of good software and know how to use it
- Be skeptical about your answer $\Rightarrow$ understand how to assess accuracy and reliability of numerical results

# Example 1

- Famous equation: pendulum equation

$$\boxed{\frac{d^2\theta}{dt^2} + \sin\theta = 0}$$

- Subject to initial conditions (at $t = 0$)

$$\boxed{\theta = \theta_0 \qquad \frac{d\theta}{dt} = 0}$$

# Example 1

1. The traditional Mathematical Approach

Simplify the problem

$$\boxed{\frac{d^2\theta}{dt^2} + \theta = 0}$$

Solve analytically

$$\boxed{\theta(t) = A \sin t + B \cos t}$$

Enforce initial conditions

$$\boxed{A = 0 \qquad B = \theta_0}$$

Works well for small swings, but bad for large swings

# Example 1

2. The traditional Numerical Approach

Rewrite as the first-order system

$$\boxed{\frac{d\theta}{dt} = V} \qquad \boxed{\frac{dV}{dt} = -\sin\theta}$$

Divide time into equal steps $t_n = n\Delta t$ and approximate $\theta(t)$, $V(t)$ by

$$\boxed{\theta_n \approx \theta(t_n), V_n \approx V(t_n)}$$

Now discretise the system

$$\boxed{\frac{V_{n+1} - V_n}{\Delta t} = -\sin(\theta_n) \qquad \frac{\theta_{n+1} - \theta_n}{\Delta t} = V_n}$$

Write the code and the values of $\theta_n$ and $V_n$ spiral out to infinity. We know that solutions are periodic.

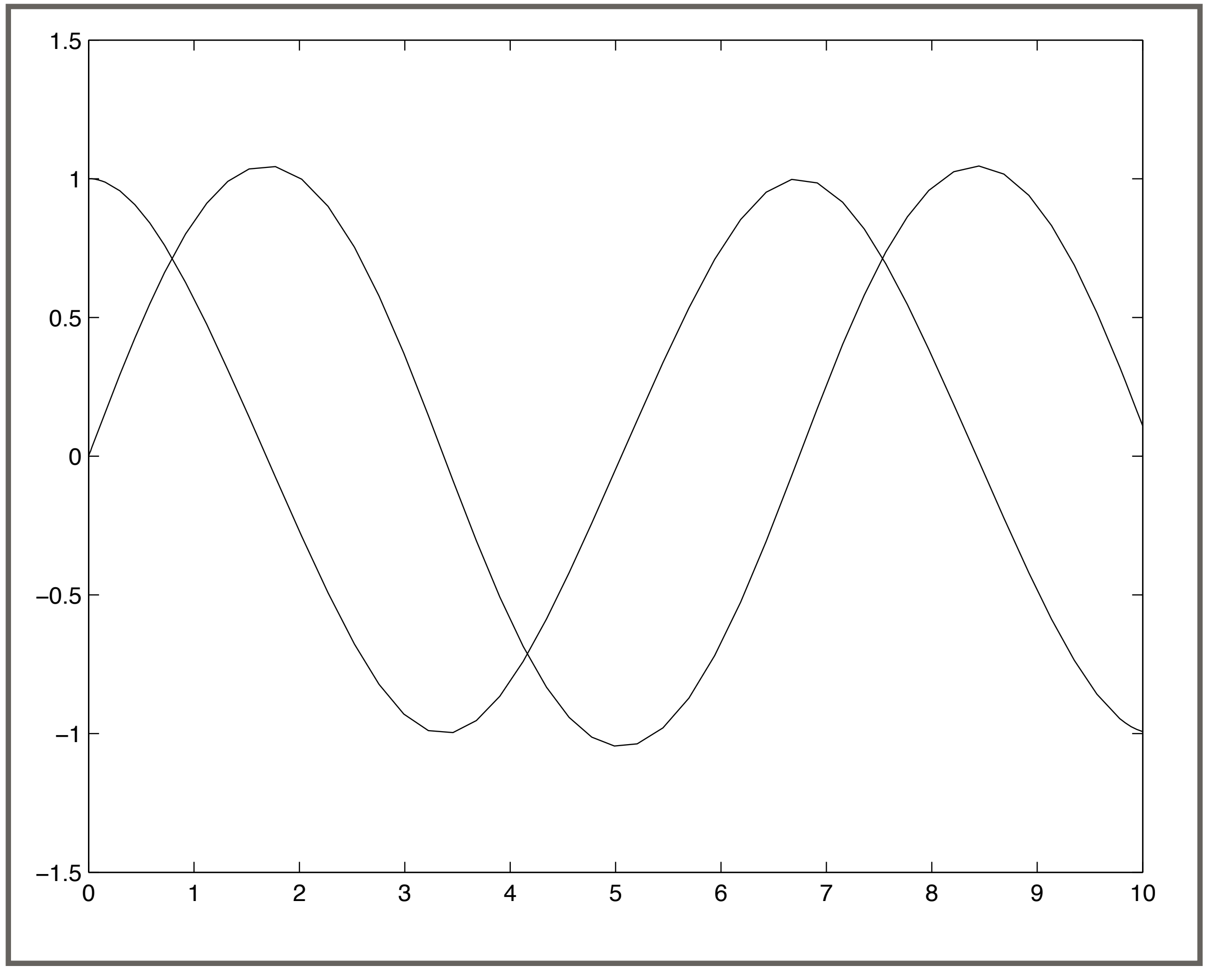
# Example 1

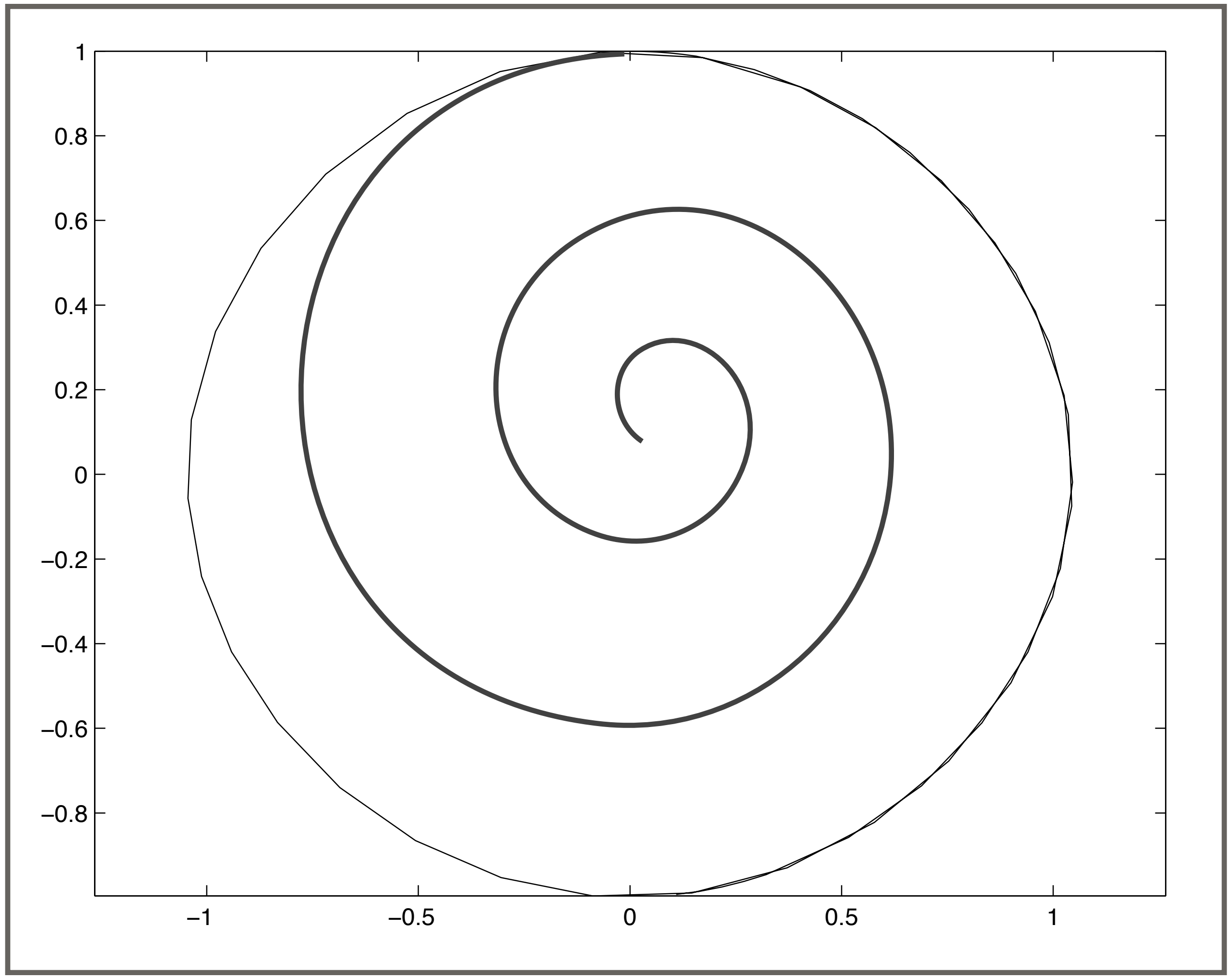
3. Software approach

Use the MATLAB routine ode45 to solve the ODE over the time interval of interest. This uses a Runge-Kutta routine with a Dormand-Price error estimator. You can specify the error tolerance in advance.

# Example 1

3. Software approach



# Example 1

4. Best approach - use Geometry

MATLAB built-in algorithms are not always the best option. Over a long period of time the solutions from MATLAB will drift.
Use the Stormer-Verlet method which is given by

$$\boxed{\theta_{n+1/2} = \theta_n + \frac{\Delta t}{2} V_n}$$

$$\boxed{V_{n+1} = V_n - \Delta t \sin(\theta_{n+1/2})}$$

$$\boxed{\theta_{n+1} = \theta_{n+1/2} + \frac{\Delta t}{2} V_{n+1}}$$

# Example 1

## 4. Best approach - use Geometry

*Acta Numerica* (2003), pp. 399–450
DOI: 10.1017/S0962492902000144

© Cambridge University Press, 2003
Printed in the United Kingdom

**Geometric numerical integration illustrated by the Störmer–Verlet method**

Ernst Hairer
*Section de Mathématiques,*
*Université de Genève, Switzerland*
*E-mail:* `Ernst.Hairer@math.unige.ch`

Christian Lubich
*Mathematisches Institut,*
*Universität Tübingen, Germany*
*E-mail:* `Lubich@na.uni-tuebingen.de`

Gerhard Wanner
*Section de Mathématiques,*
*Université de Genève, Switzerland*
*E-mail:* `Gerhard.Wanner@math.unige.ch`

# Example 2

Find the length of a vector $x = (x_1, x_2, \ldots, x_n)$

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

**algorithm** `Length` $(x, n)$

---

$sum := 0$
**for** $i := 1$ **to** $n$ **do**
  $sum := sum + x[i] \times x[i]$
**endfor**
**return** $\sqrt{sum}$
**endalg**

# Example 2

```
function y = Length(x,n)
     sum = 0;
     for i = 1:n
        sum = sum + x(i)*x(i);
     end
     y = sqrt(sum);
```

# Example 2

**Lesson: Don't write your own software unless you really have to.**

- Any $x \sim 10^{200}$ will lead to overflow
- Also danger of underflow
  Eg. $n = 10{,}000$; $x_i = 10^{-200} \Rightarrow |\mathbf{x}| = 10^{-200}$
  Code returns 0 ($x^{-400}$ set to zero)
- For **half** of possible machine numbers, code returns either overflow or underflow.

# Example 2

"A portable fortran program to find the Euclidean norm of a vector", Blue (1978)

if $n = 0$, set $\| x \| = 0$ and return.<br>
if $n < 0$, set an error flag and stop.<br>
if $n > N$, set an error flag and stop.<br>
$a_{\text{sml}} = 0;\ a_{\text{med}} = 0;\ a_{\text{big}} = 0$<br>
for $i = 1$ through $n$<br>
    if $|x_i| > B$, $a_{\text{big}} \leftarrow a_{\text{big}} + (x_i/S)^2$<br>
    else if $|x_i| < b$, $a_{\text{sml}} \leftarrow a_{\text{sml}} + (x_i/s)^2$<br>
    else $a_{\text{med}} \leftarrow a_{\text{med}} + x_i^2$<br>
if $a_{\text{big}}$ is nonzero<br>
    if $a_{\text{big}}^{1/2} > R/S$, $\| x \| > R$ and overflow would occur. Set $\| x \| = R$, set an error flag, and return.<br>
    if $a_{\text{med}}$ is nonzero<br>
        $y_{\min} = \min(a_{\text{med}}^{1/2}, S a_{\text{big}}^{1/2})$<br>
        $y_{\max} = \max(a_{\text{med}}^{1/2}, S a_{\text{big}}^{1/2})$<br>
    else set $\| x \| = S a_{\text{big}}^{1/2}$ and return<br>
else if $a_{\text{sml}}$ is nonzero<br>
    if $a_{\text{med}}$ is nonzero<br>
        $y_{\min} = \min(a_{\text{med}}^{1/2}, s a_{\text{sml}}^{1/2})$<br>
        $y_{\max} = \max(a_{\text{med}}^{1/2}, s a_{\text{sml}}^{1/2})$<br>
    else set $\| x \| = s a_{\text{sml}}^{1/2}$ and return<br>
else set $\| x \| = a_{\text{med}}^{1/2}$ and return.<br>
if $y_{\min} < \epsilon^{1/2} y_{\max}$, set $\| x \| = y_{\max}$.<br>
else set $\| x \| = y_{\max}(1 + (y_{\min}/y_{\max})^2)^{1/2}$

# Example 2

- Open source software:
  - GNU Scientific Library
  - LAPACK
  - Boost
  - Armadillo
  - BLAS
  - FFTW
  - Numpy, scipy, sympy
  - Octave
  - Netlib
- Commercial software:
  - MATLAB
  - Mathematica
  - Maple
  - Intel Math Kernel Library
  - NAG

But, **NEVER** blindly trust numerical results from any software, whether it is open source, commercial, or you own custom code.