# Computational Methods for Physics

SHPC4001

Week 1

# Computational Methods for Physics

**Objectives of this unit :**

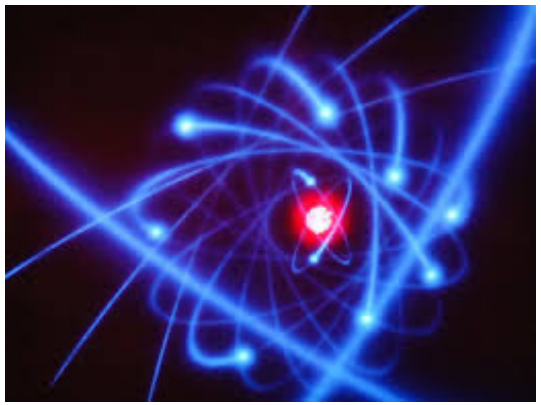
- Learn basic programming and associated computing skills;
- Learn about numerical approximations and error analysis;
- Learn to apply numerical methods to solve practical problems.

**Method of Assessment :**

100% on assignments

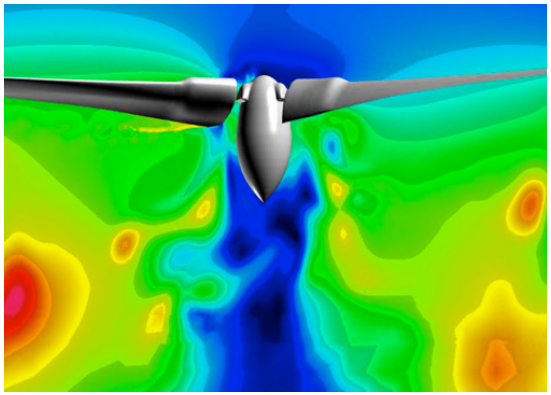
# Four components in scientific computation

1. **Theoretical model** uses mathematical equations to describe the world around us.



Schrödinger's equation

$$i\frac{\partial\psi(\mathbf{q},t)}{\partial t}=\left\{\sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\nabla_{r_i}^2+V(\mathbf{r}_i)\right)+\frac{e^2}{4\pi\varepsilon}\sum_{i>j=1}^{N}\frac{1}{\mathbf{r}_{ij}}\right\}\psi(\mathbf{q},t)$$

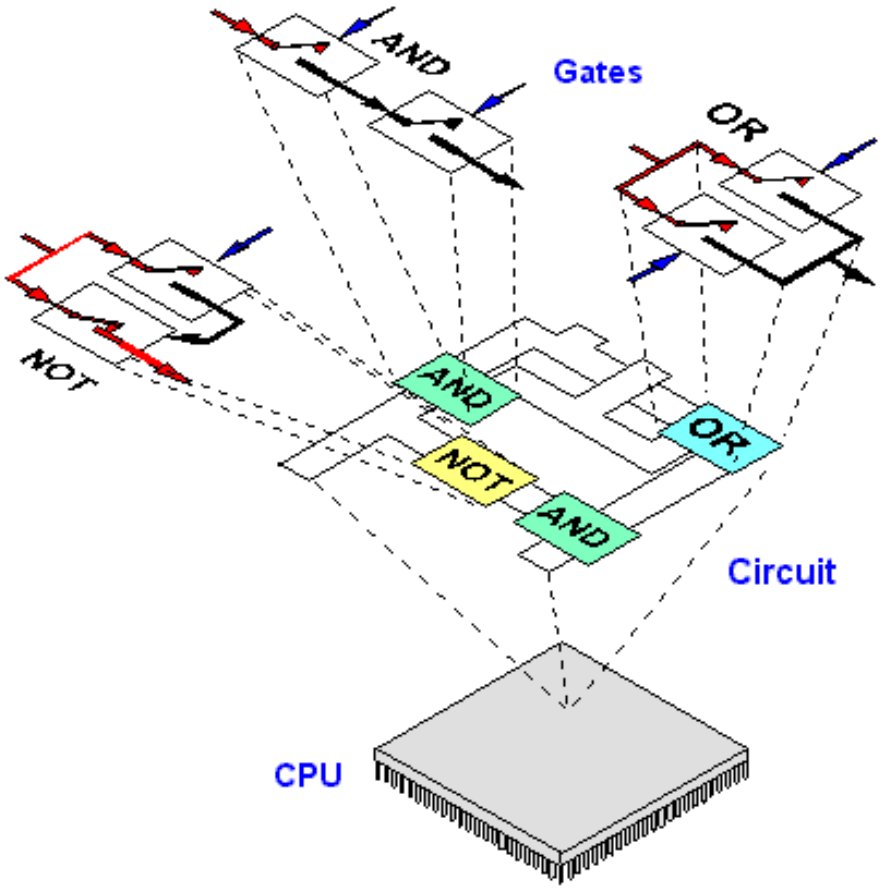


Navier-Stokes equation

$$\frac{\partial\mathbf{u}}{\partial t}=-(\mathbf{u}\cdot\nabla)\mathbf{u}+v\nabla^2\mathbf{u}-\frac{1}{d}\nabla p+\mathbf{f}$$

# Four components in scientific computation

**2. Numerical Method** or algorithm or "recipe" provides a mechanism for a computer to accomplish a given computation.




**Boolean logic : NOT, OR, AND**

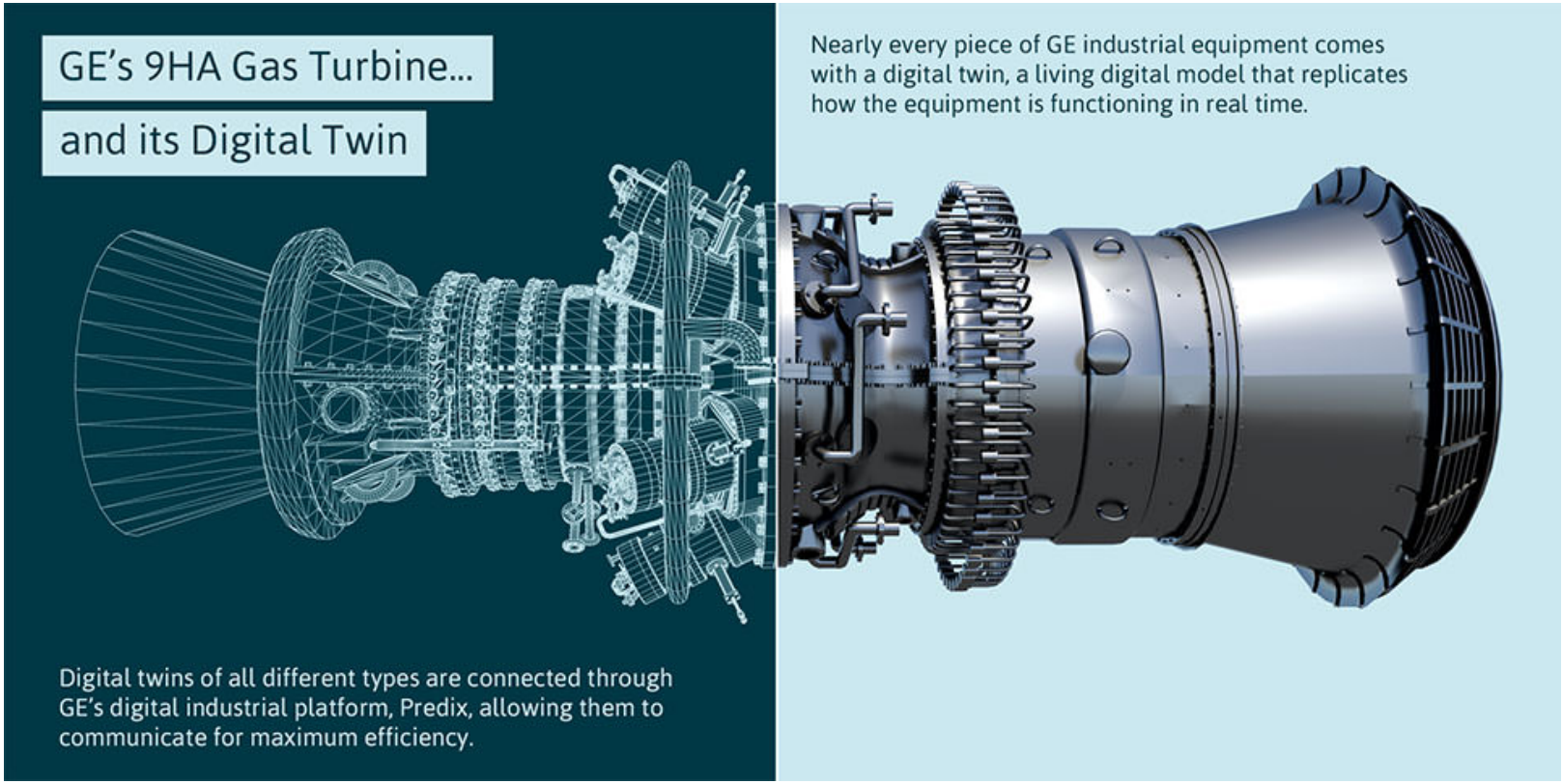
**With these simple operations, one can implement any numerical computation.**

**e.g. a single-bit adder**

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| + 0 | + 1 | + 0 | + 1 |
| 00 | 01 | 01 | 10 |
| AND | | | XOR |

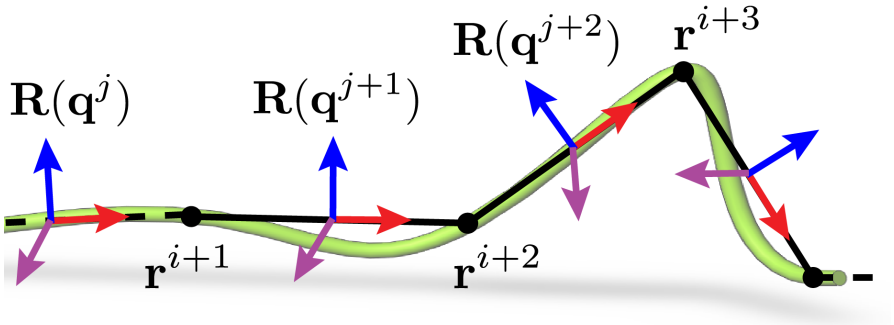
# Four components in scientific computation

**2. Numerical Method** or algorithm or "recipe" provides a mechanism for a computer to accomplish a given computation.



# Four components in scientific computation

**2. Numerical Method** or algorithm or "recipe" provides a mechanism for a computer to accomplish a given computation.




# Four components in scientific computation

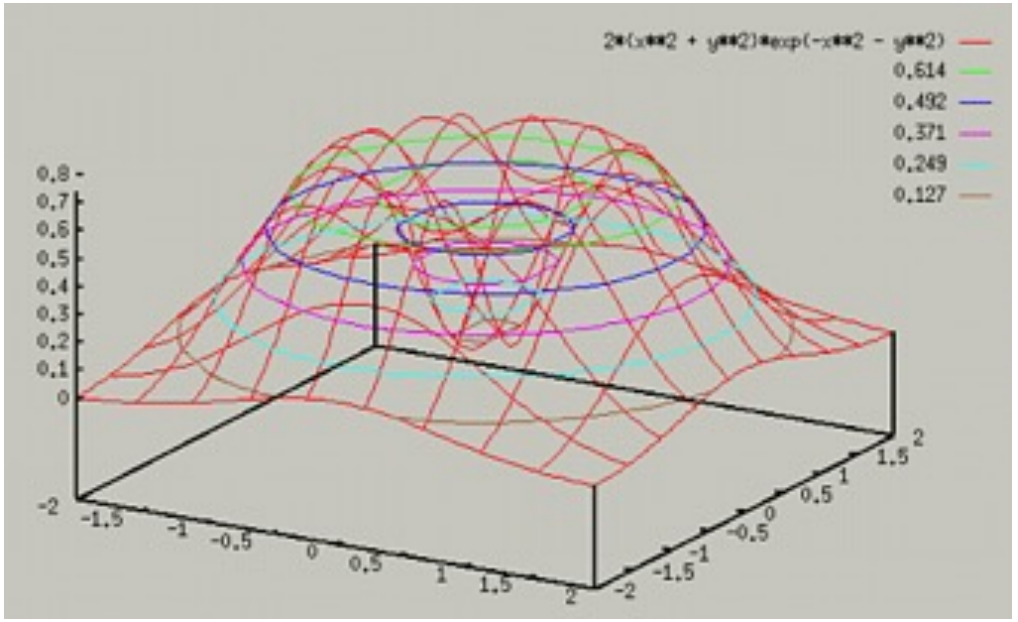
**3. Programming** is to transcribe the numerical algorithm into a set of instructions that the computer can understand.

```
program perfect
implicit none
integer::i,l=0,d=1,m,n,s=0,r,di=1

!taking the number input
write(*,*)"Enter the number"
read(*,*)i

!calculating length of the number
m=i
do
        if(d==0)exit
        d=m/10
        l=l+1
        m=d
enddo
!calculating the sum of the digits raised
!to the power the legth
n=i
do
        if(di==0)exit
        di=n/10
        r=mod(n,10)
        s=s+(r**l)
        n=di
enddo

!checking if it is plus perfect or not
if(s==i)then
        write(*,*)"It is a plus perfect number"
else
        write(*,*)"It is not a plus perfect number"
endif
end program
```

# Four components in scientific computation

## 4. Data Visualization and Interpretation

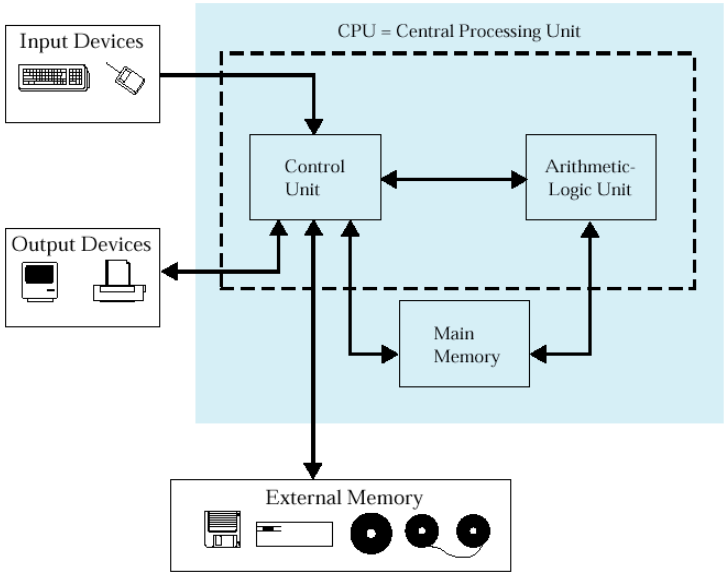
```
program perfect
implicit none
integer::i,l=0,d=1,m,n,s=0,r,di=1

!taking the number input
write(*,*)"Enter the number"
read(*,*)i

!calculating length of the number
m=i
do
        if(d==0)exit
        d=m/10
        l=l+1
        m=d
enddo
!calculating the sum of the digits raised
!to the power the legth
n=i
do
        if(di==0)exit
        di=n/10
        r=mod(n,10)
        s=s+(r**l)
        n=di
enddo

!checking if it is plus perfect or not
if(s==i)then
        write(*,*)"It is a plus perfect number"
else
        write(*,*)"It is not a plus perfect number"
endif
end program
```



# Computer Hardware

- **Computer Structure & Operation**


Edric with the 36-core 'QUISA Workstation' and the SpinQ 'Gemini' educational quantum computer

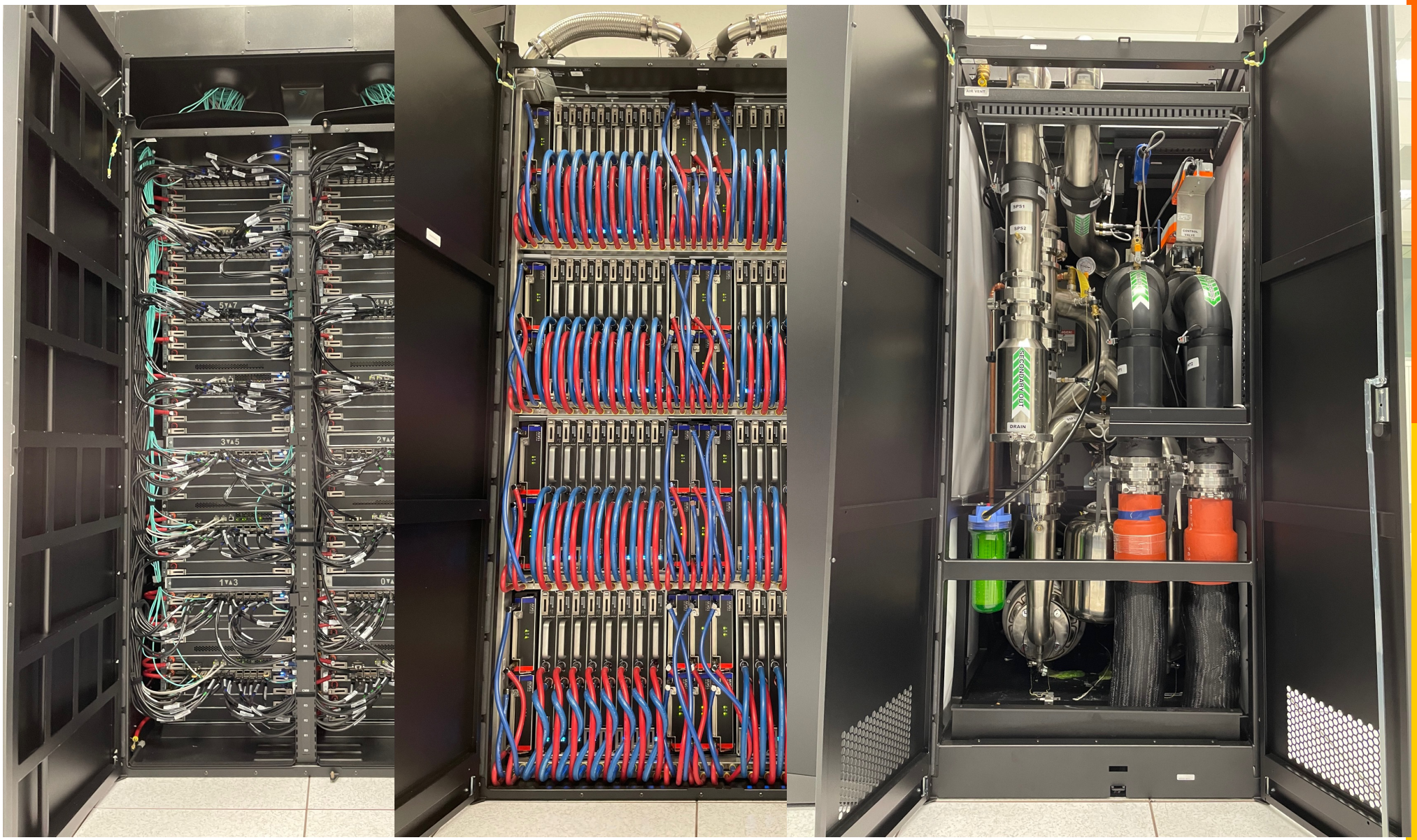




**Computing Power**: 69 teraFLOPS, from 208 dual-socket compute nodes
**Memory**: 13 terabytes (64 gigabytes per compute node, at 1,600 MHz)
**Interconnect**: Cray Aries interconnect, at 72 gigabits/sec per node


**Computing Power**: Setonix has 217,088 AMD compute cores across 1792 nodes, 750 next-generation AMD GPUs, 548 TB of CPU and GPU RAM, connected by HPE's Slingshot interconnect at 100Gb/sec, offering over 50 petaFLOPS of compute power. $70 million to build.


HPE Cray
HPE Cray
setonix
BJ


SECONDARY OUT
DRAIN

# Computer Software
## (a series of instructions to the CPU)

**Minimum software components :**

1. Operating system
   - PCs & Macs : Windows, MacOS, UNIX
   - workstations and supercomputers : UNIX
2. Editor
   - PCs & Macs : WORD, BBEdit, SimpleText, TextWrangler
     Workstations and supercomputers : vi, xemacs, gedit, nano
3. Compiler
   - programming languages:
     Basic, Pascal, Fortran, C, C++, Java, Python
   - software packages: Matlab, Maple, Mathematica, Gaussian, OpemFoam

# Use *Mathematica* to solve an equation

In[7]:=
```
f[x_] := x^3 - 2.2 x
g[x_] := f[f[f[f[x]]]]
NSolve[g[x] == 0]
```

Out[9]=
```
{{x → -1.98531}, {x → -1.96668}, {x → -1.95864}, {x → -1.93217}, {x → -1.89926},
 {x → -1.87202}, {x → -1.76915}, {x → -1.74189}, {x → -1.66218}, {x → -1.6155},
 {x → -1.52784}, {x → -1.47283}, {x → -1.42161}, {x → -1.38713}, {x → -1.26677},
 {x → -1.25821}, {x → -1.22893}, {x → -1.13763}, {x → -1.12422}, {x → -1.09335},
 {x → -1.01724}, {x → -1.00927}, {x → -0.909106 - 0.385894 i}, {x → -0.908467 + 0.385522 i},
 {x → -0.894127 - 0.444901 i}, {x → -0.894019 + 0.444868 i}, {x → -0.891533 - 0.429527 i},
 {x → -0.891421 + 0.429678 i}, {x → -0.873124 - 0.295019 i}, {x → -0.87311 + 0.29501 i},
 {x → -0.861912 - 0.392291 i}, {x → -0.861898 + 0.392306 i}, {x → -0.69238 - 0.155711 i},
 {x → -0.69238 + 0.155711 i}, {x → -0.412161 - 0.171546 i}, {x → -0.412161 + 0.171546 i},
 {x → -0.402666 + 0.242365 i}, {x → -0.402666 - 0.242365 i}, {x → -0.188682 - 0.0816936 i},
 {x → -0.188682 + 0.0816936 i}, {x → 0.}, {x → 0.188682 + 0.0816936 i}, {x → 0.188682 - 0.0816936 i},
 {x → 0.402666 - 0.242365 i}, {x → 0.402666 + 0.242365 i}, {x → 0.412161 - 0.171546 i},
 {x → 0.412161 + 0.171546 i}, {x → 0.69238 + 0.155711 i}, {x → 0.69238 - 0.155711 i},
 {x → 0.861919 - 0.392289 i}, {x → 0.861923 + 0.392286 i}, {x → 0.873108 - 0.29501 i},
 {x → 0.873111 + 0.295037 i}, {x → 0.891526 - 0.429514 i}, {x → 0.891587 + 0.429678 i},
 {x → 0.894045 - 0.444794 i}, {x → 0.894171 + 0.444881 i}, {x → 0.908048 - 0.385964 i},
 {x → 0.908331 + 0.38543 i}, {x → 1.01826}, {x → 1.01842}, {x → 1.08593}, {x → 1.09041},
 {x → 1.16656}, {x → 1.22893}, {x → 1.22976}, {x → 1.26484}, {x → 1.35076}, {x → 1.41217},
 {x → 1.42111}, {x → 1.54172}, {x → 1.58777}, {x → 1.6695}, {x → 1.72632}, {x → 1.77891},
 {x → 1.86767}, {x → 1.89827}, {x → 1.92647}, {x → 1.95243}, {x → 1.97739}, {x → 1.97771}}
```

In[10]:=
```
g[-1.9853128147769314`]
```

Out[10]=
$$-5.6046 \times 10^{13}$$

# Why Fortran?

- Fortran was developed for FORmula TRANslation, and it is very good at that.
- Fortran is designed to help develop codes quickly and easily, hiding some of the complications of other languages
- The maturity of optimising compilers provide benefit that other languages cannot yet comp
- Many physics computer programs, are written in Fortran (see, e.g. http://www.cpc.cs.qub.ac.uk/cpc).

# A Comparison of C, Fortran and Java for Numerical Computation

J.V. Ashby
(Rutherford Appleton Laboratory)

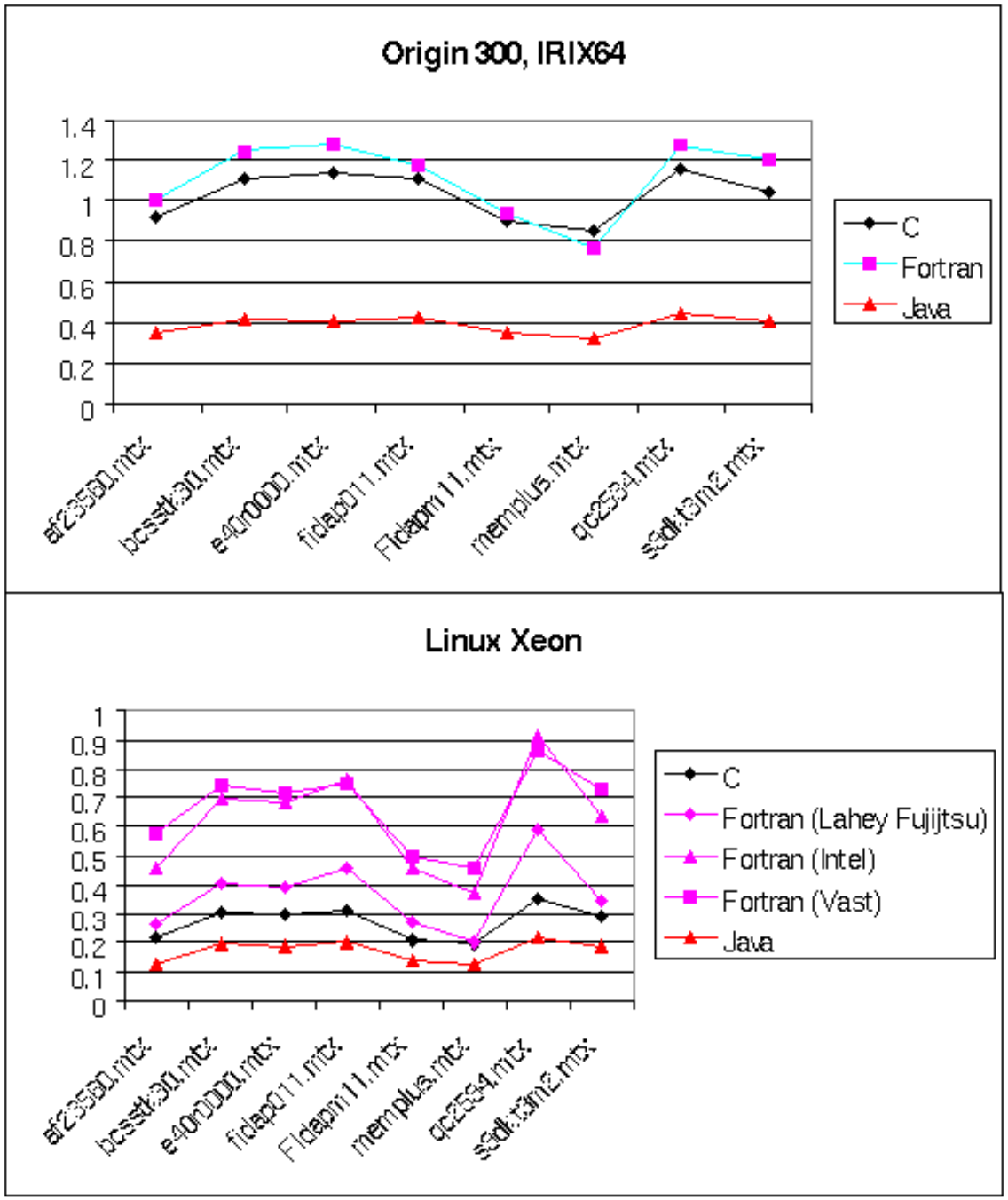



Figure 1: Normalised speeds for sparse matrix multiply in various languages on various machines.

FortranFacesFuture.pdf

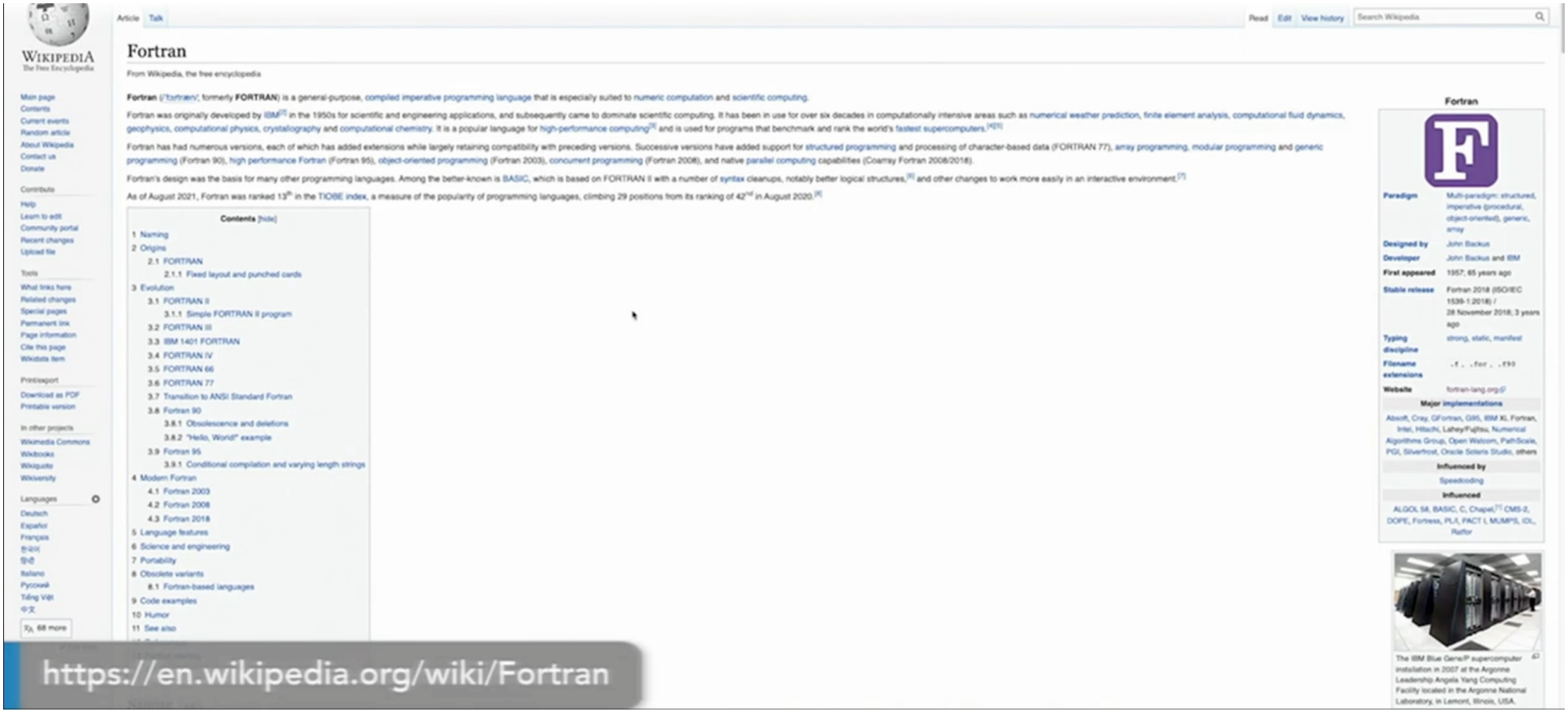
Despite the wealth of off-the-shelf software packages, there often comes a time in scientific research when they do not do quite what is needed. It is then usually much simpler for a researcher to write the necessary software than for a software expert to understand the scientific requirements. And, true to the design goals of John Backus and his team back in the mid-1950s, Fortran is still one of the easiest languages for a scientist or engineer to learn. ■

There are many reasons why physicists still choose Fortran over other languages

Or not at all


linkedin.com/learning/introduction-to-fortran
Learning
Browse
What do you want to learn toda
Home
Solutions for:
Introduction to Fortran
Beginner • 2h 28m • Released: May 25, 2022
4.6 (47) • 14,193 learners
Start my free month
Buy this course

# Introduction to Fortran

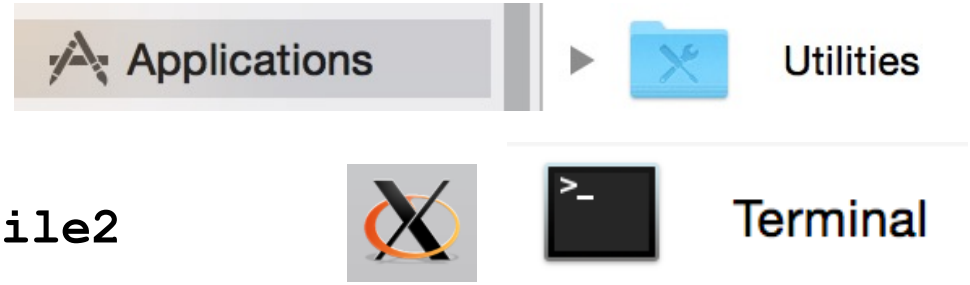



By: Brad Richardson, May 2022

- login into your account
- learn Unix commands
- learn Gnuplot
- intro to Fortran
- bouncing ball

# Basic commands

**(teams/general) UNIX Tutorial for Beginners.pdf**




**`ls`** lists the files of the current directory

**`cd <directory>`** change directory

**`pwd`** shows present working directory

**`cp <file1> <file2>`** copies **`file1`** to **`file2`**

(**`e.g. cp ~jwang1/cp/bounce.dat b.dat`**)

**`mv <file1> <file2>`** change filenames

**`rm <file>`** removes a file

**`mkdir <directory>`** creates a directory

**`rmdir <directory>`** removes a directory

**`man <command>`** online manual

**`man -k <keyword>`** search for the specified string in *all* man pages.

**`vi <filename>`** a sophisticated text editor

**`nano or pico or jpico <filename>`** more novice friendly text editors

**`more <filename>`** display a file a page at a time

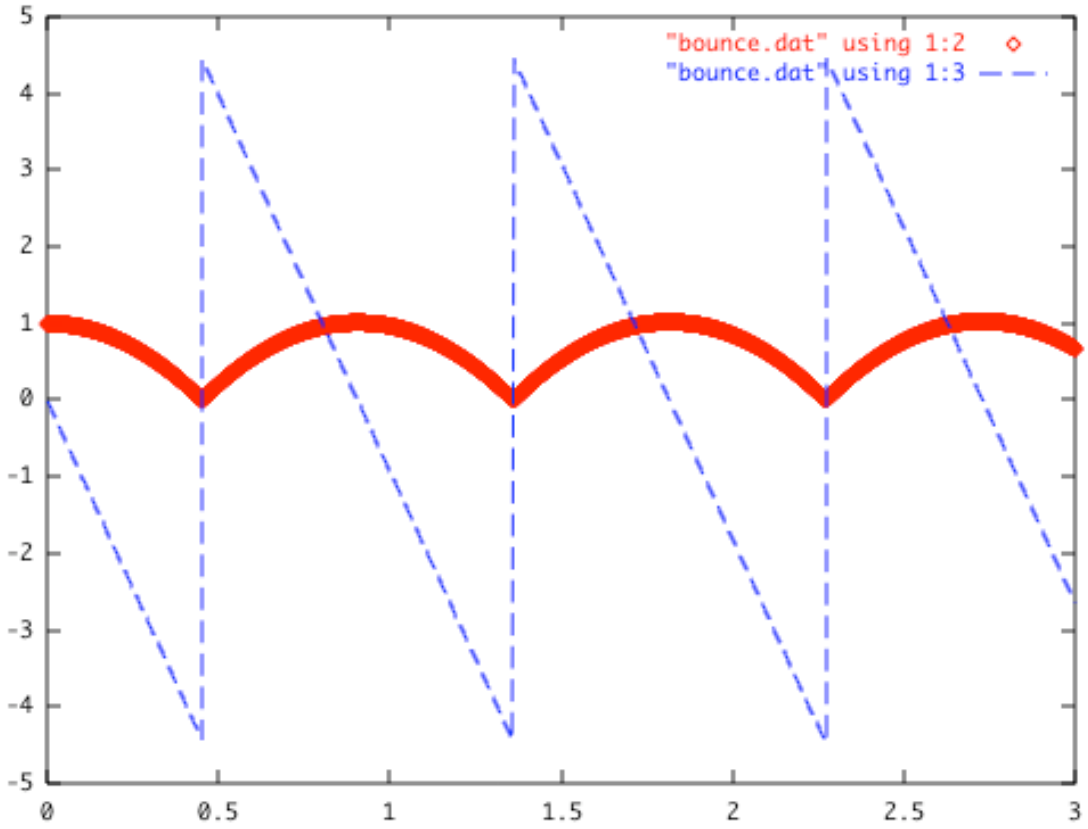
**`tail <filename>`** display a file a page at a time

**`grep <word> <filename>`** searches files for specified words

**`logout`** exit

# Data Visualization

**GNUPLOT** is a free, command-driven, interactive, function and data plotting program. Gnuplot can be run under DOS, Windows, Macintosh OS, VMS, Linux, and many others.



# Gnuplot

(teams/general) GnuplotSummary.pdf, GnuplotTutorial.pdf

- **gnuplot**

```
gnuplot> plot sin(x)
gnuplot> splot sin(x)*cos(y)
gnuplot> plot sin(x) title 'Sine', tan(x) title 'Tangent'
gnuplot> plot "force.dat" using 1:2
gnuplot> plot "force.dat" using 1:2 title 'data A', \
              "force.dat" using 1:3 title 'data B'
gnuplot> set yrange [20:500]
gnuplot> plot "force.dat" using 1:2 title 'data A' with lines, \
              "force.dat" using 1:3 title 'data B' w linespoints
gnuplot> exit
```

copy (teams/general)force.dat

# Introduction to Fortran Programming

```
program first

! This is a most basic Fortran program

  implicit none

  write(*,*), "Test!"

end program first
```

```
> copy (teams/general)firstprogram.f90
> gfortran firstprogram.f90 -o firstprogram
> ./firstprogram
```

Why do we need
implicit none ?

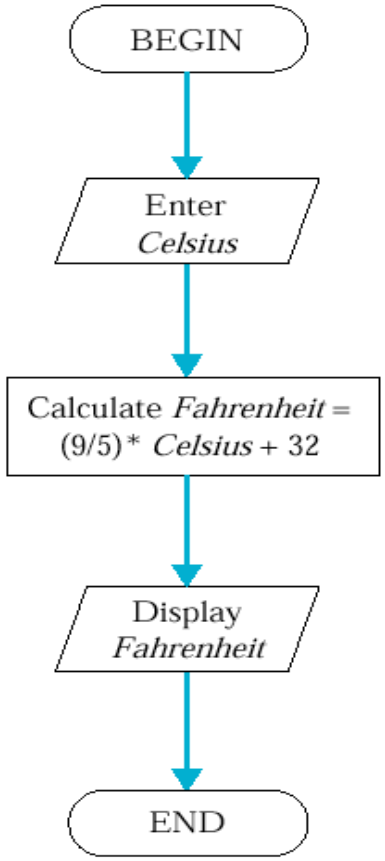
```
program second

implicit none
real:: a1, a2, b
!      initialise variables
a1=0.0
a2=0.0
!      set a1=1.0 and a2=2.0
a1=1.0
a2=2.0
!      compute b=a1+a2
b=a1+a2
write(*,*)b
end program second
```

Fortran assumes ...
i-n integer
a-h o-z real
This turns that off

```
> copy (teams/general)secondprogram.f90
> gfortran secondprogram.f90 -o secondprogram
> ./secondprogram
```

Correct answer ?

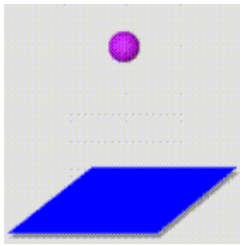
# Converts Celsius to Fahrenheit




```
program temperature

implicit none
real :: DegC, DegF

Write(*,*)"Please type in temp in Celsius"
Read(*,*)DegC

DegF = (9./5.)*DegC + 32.

Write(*,*)"This equals to”,DegF,"Fahrenheit"

end program temperature
```

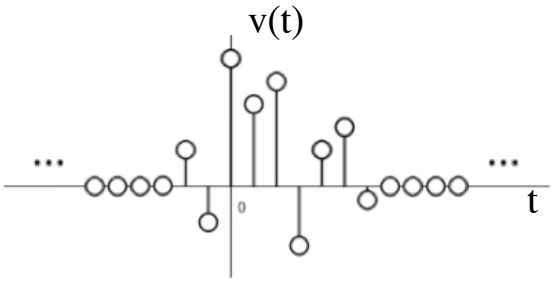
```
> copy (teams/general)temperature.f90
> gfortran temperature.f90 -o temperature
> ./temperature
```

# Fortran



$$\frac{dv}{dt} = F/m = -g$$

$$\frac{dx}{dt} = v$$

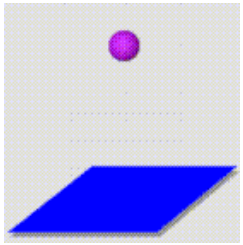



$$\frac{v(t+\Delta t) - v(t)}{\Delta t} \approx -g$$

$$\frac{x(t+\Delta t) - x(t)}{\Delta t} \approx v(t)$$

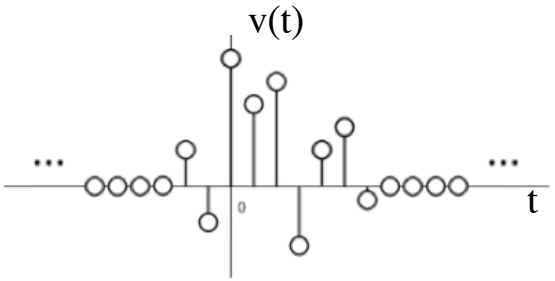
```
program bouncing_balls

! Use the FD method to compute the trajectory of a bouncing ball
! assuming perfect reflection at the surface x = 0. Use SI units.

integer :: steps
real :: x, v, g, t, dt

x = 1.0          ! initial height of the ball
v = 0.0          ! initial velocity of the ball
g = 9.8          ! gravitational acceleration
t = 0.0          ! initial time
dt = 0.01        ! size of time step

open(10,file='bounce.dat')    ! open data file

do steps = 1, 300     ! loop for 300 timesteps

  t = t + dt
  x = x + v*dt
  v = v - g*dt

  if(x.le.0) then                 ! reflect the motion of the ball
    x = -x                        ! when it hits the surface x=0
    v = -v
  endif

  write(10,'(3f10.6)') t, x, v  ! write out data at each time step

enddo

end program bouncing_balls
```

C



$$\frac{dv}{dt} = F/m = -g$$

$$\frac{dx}{dt} = v$$




$$\frac{v(t+\Delta t) - v(t)}{\Delta t} \approx -g$$

$$\frac{x(t+\Delta t) - x(t)}{\Delta t} \approx v(t)$$

```
#include <stdio.h>          /* standard input-output header */

/* Use the FD method to compute the trajectory of a bouncing ball  */
/* assuming perfect reflection at the surface x = 0. Use SI units. */

int main(void)
{   int steps;
    float x,v,t,g,dt;
    FILE* output_file;

    x = 1.0;   /* initial height of the ball  */
    v = 0.0;   /* initial velocity of the ball */
    g = 9.8;   /* gravitational acceleration */
    t = 0.0;   /* initial time */
    dt = 0.01; /* size of time step */

    output_file = fopen("bounce.dat","w");  /* open data file */

    for(steps=1;steps<=300;steps++)   /* loop for 300 timesteps */
    {   t += dt;      /*shorthand equivalent to t = t+dt */
        x += v*dt;
        v -= g*dt;
        if(x < 0)     /* reflect the motion of the ball */
        {             /* when it hits the surface x=0 */
            x = -x;
            v = -v;
        }

        fprintf(output_file,"%f %.7f %f \n",t,x,v);
            /* write out data at each time step */
    }
     fclose(output_file);
}
```

# Assignment 1 (due next Wednesday midnight)
# Submit via LMS

**Exercise 1**: Copy **`(teams/general)bouncing.f90`** to your directory, compile it to produce an executable file **`(gfortran bouncing.f90 -o bbf)`** and run **`(./bbf)`**. Plot and interpret your results. Are your numerical results physically correct? If not, can you identify a **systematic error** (not a round-off error) in the algorithm and then fix the problem?

**Exercise 2**: Copy **`(teams/general)bouncing.c`** to your directory, compile it to produce an executable file **`(cc bouncing.c -o bbc)`** and run **`(./bbc)`**. Compare with your Fortran results.

**Exercise 3**: Change time step **`dt`** in the code (either Fortran or C). Keep the same total evolution time. Explain the changes in the results.

**Exercise 4**: Change the initial velocity and position of the falling ball in the code. Plot and interpret your results.

**Exercise 5**: Consider inelastic collisions with the table (e.g. the ball loses 10% of its speed after each collision). Plot and interpret your results.

# Remember …

- Plan ahead and start early!!
- Save your work often!!
- Do only ONE change at a time and make sure it works before the next change.
- Document your code!!
- Learn to deal with computer problems, such as lost data due to computer crash or scratched drives, misuse of syntax, compiling errors, and all sorts of bugs in your codes!!

**Your assignment (wherever applicable) should include:**

- Introduction to the physical problem addressed.
- Discussion of the numerical method and algorithm used (include a flow-chart if necessary; include references).
- Details of code validation. Test cases.
- All necessary data, tables or graphical output.
- A clear summary of the results in terms of the solution to the physical problem addressed. Answers to all questions.
- Reference if you used any.
- Codes, which must be adequately commented.

**All in one PDF document, except the codes.**