Computational Physics 2

Administrative

Lecturer: me

Lectures: Thursdays 12noon - 1pm
Labs: Thursdays 2pm - 4pm

Mark distribution: Cont.assessment
(= quizes + assignments) $\longrightarrow$ 30%
Exam $\longrightarrow$ 70%

MP468P Project: (dual-department students) Oct–May

Overview of lecture slides 00

1. Content/overview of module
2. The unix/linux command line
3. Programming language(s) — choice of python
4. Changing landscape of computational physics
5. Things not covered
6. The practice of scientific computing
7. You should...

Computational Physics 2: Content

Module topics

- Random numbers and stochastic processes
- Monte Carlo methods
- Linear algebra
  (Linear sets of equations, matrix decompositions, eigenvalues)
- Minimisation / Optimisation
- Partial differential equations (PDEs)
  + ODE boundary value problems
- "Soft skills": unix/linux command-line, python

Computational Physics 2: Specialties

This module is a bit different...

- Components of $\left\{ \begin{array}{l} \text{Math (linear algebra, PDE's)} \\ \text{Statistics (random processes, probability)} \\ \text{Computer Science (Algorithms)} \end{array} \right.$
- 'Lab' work
- May turn out to be the most useful subject for your future
- The most 'modern' module.
  (Numerics with python — less than 20 years old)
- Some aspects will become outdated in a few years.
  (Programming tools, workflow, etc. Not the principles.)

The command line

Why unix systems, why command line?

- Serious computing generally done on unix/linux machines
- Serious computing usually done remotely on multiple machines/cores
  - at high-performance computing facilities (e.g., ICHEC in Ireland)
  - wherever you have access to multiple cpu's or gpu's
  - Difficult to work remotely without command-line knowledge!

The command line

Common commands

- Basic: ls, cd, cp, mv, rm, mkdir, less, grep, diff, cat, top, ps, kill
- Slightly less basic:
  ssh, find, awk, tar, sort, gzip, unzip, chmod, chown, tail, head

Combinations

- Piping the results of one command (program) into another:
  ls | sort -r   ls -l | sort -g -k 5   cat *.tex | grep -i perturbative
- Redirection: output of one command sent to a file:
  cat file1.txt file2.txt > outfile.txt

Computer languages other than python

Should you learn other programming languages?

C? julia? Fortran? Mathematica? matlab/octave? maple?

SageMath? R? html? java? javascript? C++? Go?

lisp? php? perl? bash? awk & sed? Ruby? C#?

Pascal? COBOL? Basic? Assembly language? POV-Ray?

Computer languages

Low-level vs High-level

- Low-level $\rightarrow$ closer to machine;
  programmer implements many details;
  speed and control at the expense of programmer time.
- High-level $\rightarrow$ closer to human; $\approx$ scripting languages
  programmer uses libraries; pre-defined data structures
- Nowadays, low-level $\approx$ compiled; high-level $\approx$ interpreted.

Low-level languages:
(low to high)

Machine language
Assembly language
C / Fortran

High-level languages:

python / julia / matlab
Mathematica / Maple
awk / bash / perl

Using python for computational physics

Python: the best programming language ever (?)

- Widely used — lots of information easily available
- Easy to learn
  interpreted not compiled, don't have to worry about variable types
- Libraries available for many common (and specialized) tasks.
  - Most relevant for us: numpy, scipy, matplotlib
- Speed does not matter nowadays for many tasks.
  - Many tasks done by external, non-python libraries.
    Example: matrix eigensolvers call 'lapack' library.
- Counting starts at 0 instead of 1, like a proper computer language.

Using python for computational physics

Python: a terrible choice of programming language

- Widely used — lots of junk information and incompetent users
- Slow. Very, very slow. Crawling slow.
  interpreted not compiled
- Sometimes speed actually matters.
  - E.g., Monte Carlo calculations.
  - To speed up critical parts of your code, you might have to write those parts in C/Fortran. $\longrightarrow$ two-language problem
- Designed originally for computer people, not for physicists or for numerical work. We are secondary citizens in the python world. Sometimes this shows :-(
- Counting starts at 0 instead of 1, an insult to people who deal with matrices and vectors.

Alternatives to python

Matlab/octave

- Pros: Designed for numerical work. Just-In-Time compiler makes matlab faster than python. Octave freely available. Packages less chaotic than for python.
- Cons: Matlab needs expensive license. Octave slower.
  Not a proper programming language.

C/ C++

- Pros: Fast if coded correctly.
- Cons: Have to learn a (much) more complicated language than python. Compilation necessary — development cycle slower. Memory allocations by hand. Not as many convenient predefined data structures. Using libraries is a more involved process.

Alternatives to python, continued

Fortran

- Pros: Designed explicitly for numerical work. Fast if coded correctly.
- Cons: Compilation cycle. Not used much outside numerics.

julia

- Pros: Designed explicitly for numerical work.
  Aims to solve the two-language problem — aims to be fast to develop and fast to run. Aims to overcome deficits of python.
- Cons: Still new, and changing/growing.
  E.g., libraries currently even more chaotic than python.

Alternatives to python, continued further

Mathematica/ Maple

- Pros: Combination of numerical and symbolic capabilities.
- Cons: Not free or open-source. Expensive license.
  Not general-purpose programming languages.

R

- Pros: Great for statistics. Great graphics package.
- Cons: Slow. Not as suitable for non-statistical tasks.

Changing landscape of computational physics

Algorithms and principles

- Mostly stable, but some things change
- Example: gradient descent

Changing landscape...

Programming practices (and fashions)

- Rapid change — be warned (and be prepared)
- python was considered unacceptably slow for numerics, until $\sim$2005.
- double precision was considered unacceptably slow for numerics.
- integer division, different in python2 and python3.
- GPU usage increasingly unavoidable. :-(
- For scientific usage, python might be replaced by julia soon(ish).

Things not covered in MP468C

- Many, many aspects of numerical analysis!!
  Graph algorithms, advanced data structures, adaptive numerical integration, extrapolation, finite element methods, ...
- Serious applications of computers in physics
  Quantum Monte Carlo, molecular dynamics, density functional theory,....
- Statistical data analysis, Machine learning
- Parallel computing
- GPU computing
- Cloud computing
- Other programming languages/paradigms: Matlab/octave, mathematica, julia, C, ...
- Many python features: objects/classes, sympy, making packages,...

The practice of computational physics

Do's and dont's

- Don't guess what a command/package does. Look it up.
  E.g., If you use np.arange(2,10), first read its doc.

- Looking up documentation: use reliable sources.

- Start coding first, think later?

  Please please please don't!!!

- First calculate by hand (on paper) whatever is needed.
  When possible: write out what you need to code as pesudocode or as an algorithm.

Practice — writing out algorithms

Example (from wikipedia page on Metropolis-Hastings)

1. Initialise
   1. Pick an initial state x_0.
   2. Set $t = 0$.
2. Iterate
   1. *Generate* a random candidate state x' according to $g(x' \mid x_t)$.
   2. *Calculate* the acceptance probability $A(x', x_t) = \min\left(1, \frac{P(x')}{P(x_t)} \frac{g(x_t \mid x')}{g(x' \mid x_t)}\right)$.
   3. *Accept or reject*:
      1. generate a uniform random number $u \in [0, 1]$;
      2. if $u \le A(x', x_t)$, then *accept* the new state and set $x_{t+1} = x'$;
      3. if $u > A(x', x_t)$, then *reject* the new state, and copy the old state forward $x_{t+1} = x_t$.
   4. *Increment*: set $t = t + 1$.

Writing out algorithms

Example from Higham, *Accuracy & Stability of Numerical Algorithms*

Power method
for finding eigenvalues

% Choose a starting vector x.
while not converged
$\quad x := Ax$
$\quad x := x/\|x\|_\infty$
end

Writing out algorithms

Another example from Higham, *Accuracy & Stability...*

Computing the QR decomposition of an $n \times n$ matrix A, using a Gram-Schmidt-like method.

$$
\begin{array}{l}
a_k^{(1)} = a_k,\ k = 1\colon n \\
\text{for } k = 1\colon n \\
\quad r_{kk} = \|a_k^{(k)}\|_2 \\
\quad q_k = a_k^{(k)}/r_{kk} \\
\quad \text{for } j = k+1\colon n \\
\qquad r_{kj} = q_k^T a_j^{(k)} \\
\qquad a_j^{(k+1)} = a_j^{(k)} - r_{kj} q_k \\
\quad \text{end} \\
\text{end}
\end{array}
$$

Writing out algorithms

Example from Kreyszig, *Advanced Engineering Mathematics*

ALGORITHM RUNGE–KUTTA (f, x_0, y_0, h, N).

This algorithm computes the solution of the initial value problem $y' = f(x, y)$, $y(x_0) = y_0$ at equidistant points

(9) $$x_1 = x_0 + h, x_2 = x_0 + 2h, \cdots, x_N = x_0 + Nh;$$

here f is such that this problem has a unique solution on the interval $[x_0, x_N]$ (see Sec. 1.7).

INPUT: Function f, initial values x_0, y_0, step size h, number of steps N

OUTPUT: Approximation y_{n+1} to the solution $y(x_{n+1})$ at $x_{n+1} = x_0 + (n + 1)h$, where $n = 0, 1, \cdots, N - 1$

For $n = 0, 1, \cdots, N - 1$ do:

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1)$$
$$k_3 = hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$x_{n+1} = x_n + h$$
$$y_{n+1} = y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

OUTPUT x_{n+1}, y_{n+1}

End

Stop

End RUNGE–KUTTA

Writing out algorithms

Another example from Kreyszig, *Adv. Eng. Math.*

Table 20.2 Gauss–Seidel Iteration

ALGORITHM GAUSS–SEIDEL ($\mathbf{A}, \mathbf{b}, \mathbf{x}^{(0)}, \epsilon, N$)

This algorithm computes a solution $\mathbf{x}$ of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ given an initial approximation $\mathbf{x}^{(0)}$, where $\mathbf{A} = [a_{jk}]$ is an $n \times n$ matrix with $a_{jj} \neq 0, j = 1, \cdots, n$.

INPUT: $\mathbf{A}, \mathbf{b}$, initial approximation $\mathbf{x}^{(0)}$, tolerance $\epsilon > 0$, maximum number of iterations N

OUTPUT: Approximate solution $\mathbf{x}^{(m)} = [x_j^{(m)}]$ or failure message that $\mathbf{x}^{(N)}$ does not satisfy the tolerance condition

For $m = 0, \cdots, N-1$, do:

For $j = 1, \cdots, n$, do:

1 $$x_j^{(m+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{k=1}^{j-1} a_{jk} x_k^{(m+1)} - \sum_{k=j+1}^{n} a_{jk} x_k^{(m)} \right)$$

End

2 If $\max_j |x_j^{(m+1)} - x_j^{(m)}| < \epsilon\, |x_j^{(m+1)}|$ then OUTPUT $\mathbf{x}^{(m+1)}$. Stop

[*Procedure completed successfully*]

End

OUTPUT: "No solution satisfying the tolerance condition obtained after N iteration steps." Stop

[*Procedure completed unsuccessfully*]

End GAUSS–SEIDEL

How you can help (yourself and me)

Would help if you...

- Keep learning python and numpy intricacies —
  read sections of the official documentation
  (or a good book) as bedtime reading
- Install a linux/unix shell (a bash shell) on your own machine.