Lecture 1, Sci. Comp. for DPhil Students II

Nick Trefethen, Tuesday 21.01.20

Today

- IV. ODEs and nonlinear dynamics
- IV.1 ODE IVPs
- IV.2 Runge-Kutta and multistep formulas
- IV.3 IVP codes in MATLAB and Simulink
- IV.4 IVP solutions in Chebfun

Handouts

- fact sheet (revised for this term)
- Assignment 1, due next Tuesday at 10:00
- anonymous functions, `m26_vanderpol.m`, `m26chebfun.m`
- Shampine & Reichelt: first 2 pages of MATLAB ODE Suite article

  ----

Before beginning, we have some fun with a Christmas present I got from my daughter Emma, who works at Google: `websitefordad.com`.

–

Welcome back to the second term of Scientific Computing for DPhil students.

Last term was focussed on linear algebra and optimization. This term we look at ODEs, PDEs, and nonlinear dynamics.

I believe that nonlinearity was the most important theme in the mathematics of the second half of the 20th century. It was computers that led to this.

Is anyone here new to the course? (That's fine, though people who turn in assignments for grading have to have done part 1 last term.)

Go over the course fact sheet. Note the new URL for this term: `https://courses.maths.ox.ac.uk/node/45433`. As you know, I recommend you read the online lecture notes after each lecture.

IV. ODEs and nonlinear dynamics

IV.1 ODE IVPs

Freely available textbook: *Exploring ODEs*, by T., Birkisson, and Driscoll, SIAM, 2018. See under Books at my web page.

ODE = ordinary differential equation (i.e., just 1 independent variable)

IVP = initial-value problem

First-order IVP in standard form:

$$u' = f(t,u), \quad t > 0 \quad [u = u(t)],$$

$$u(0) = u_0 \ \text{ (initial data)}$$

Despite appearances, this form is very general.
One reason is that it may describe a **system** of N ODEs:

$$u(t) = N\text{-vector (for each } t), \qquad u_0 = N\text{-vector.}$$

Another reason is that it may be equivalent to a higher-order system.
E.G., consider the simple linear **harmonic oscillator**

$$w'' = -w, \qquad w(0) = 1, \ w'(0) = 0.$$

Define the 2-vector u by $u^{(1)} = w$, $u^{(2)} = w'$. Then we have equivalently the 1st-order system of two linear equations

$$u^{(1)\prime} = u^{(2)}, \quad u^{(1)}(0) = 1,$$
$$u^{(2)\prime} = -u^{(1)}, \quad u^{(2)}(0) = 0,$$

or if written in matrix form, which we can do since the problem is linear,

$$\begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix}, \qquad \begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

In this ODE the variable t does not appear explicitly.
An ODE with that property is said to be **autonomous**.

Larger example: the sun and moon and nine planets. Each has three coordinates of position and three of velocity. All together, that's an autonomous ODE IVP of dimension $N = 66$. (Or 60, if Pluto isn't a planet.)

Some simple examples are linear and can be solved analytically.
Most nonlinear ODEs, however, cannot be solved analytically.
We need numerics.

Nonlinear example: van der Pol equation

$$w'' + C(w^2 - 1)w' + w = 0, \quad C > 0 \text{ fixed.}$$

For $|w| > 1$, a damped oscillator. For $|w| < 1$, negatively damped.
Big solutions decay; small solutions grow.
As $t \to \infty$ we have convergence to a **limit cycle** of size $O(1)$.

Equivalent first-order system:

$$u^{(1)\prime} = u^{(2)}, \quad u^{(2)\prime} = -u^{(1)} - C[(u^{(1)})^2 - 1]u^{(2)}.$$

[anonymous functions; `m26_vanderpol.m`]

IV.2 Runge-Kutta and multistep formulas

ODEs have been solved numerically for a long time.

Leonhard Euler, 1768
John Couch Adams, 1850s (unpublished)
  (predicted Neptune in 1845, at age 26; Leverrier found it in 1846)
  Cambridge Senior Wrangler, 1843
Francis Bashforth, 1883
  Cambridge Second Wrangler, 1843
Carl Runge, 1895
Karl Heun, 1900
Martin Wilhelm Kutta, 1901
Forest Ray Moulton, 1926 [an American – unusual in that era]
Richard von Mises, 1930

There are two main classes of methods:
Runge-Kutta = 1-step, **Adams** and others = multistep

Time discretization: $t_n = nk$, $k > 0$ fixed **time step**

```
  |-------|-------|-------|-------|-------|-------|--
 t =0    t =k    t =2k      ...
  0       1       2
```

Approximate $u(t_n)$ by v_n computed by finite differences.

Here are the first four **Adams-Bashforth** multistep formulas.
Notation: $f_n = f(t_n, v_n)$. The first is called the **(forward) Euler formula**.

$$v_{n+1} = v_n + kf_n, \quad \text{accuracy } O(k),$$

$$v_{n+1} = v_n + \frac{k}{2}(3f_n - f_{n-1}) \quad \text{accuracy } O(k^2),$$

$$v_{n+1} = v_n + \frac{k}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \quad \text{accuracy } O(k^3),$$

$$v_{n+1} = v_n + \frac{k}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \quad \text{accuracy } O(k^4).$$

This is an infinite sequence of formulas.
Drawback: they are tricky to start up because extra values are needed.

Some Runge-Kutta formulas:

"Modified Euler" $O(k^2)$

$a = kf(t_n, v_n)$

$b = kf(t_n + k/2, v_n + a/2)$

$v_{n+1} = v_n + b$

"Fourth-order Runge-Kutta" $O(k^4)$

$a = kf(t_n, v_n)$

$b = kf(t_n + k/2, v_n + a/2)$

$c = kf(t_n + k/2, v_n + b/2)$

$d = kf(t_n + k, v_n + c)$

$v_{n+1} = v_n + \frac{1}{6}(a + 2b + 2c + d)$

If you could take just one formula to a desert island, I recommend fourth-order Runge-Kutta.

Higher-order RK formulas get very complicated. It's not an infinite sequence in any simple sense. Beautiful theory based on trees (in the sense of graph theory) is due to John Butcher of the U. of Auckland.

For years there was a tendency for people on the continent to prefer RK, people in the UK and the USA to prefer multistep. Prof. Gerhard Wanner of the University of Geneva claims that this separation goes back to the fight about calculus between Leibniz and Newton!

IV.3 IVP codes in MATLAB and Simulink

ODE codes are among the must successful software products in all of scientific computing. The key is that they are **adaptive** – varying step size (always) and order (sometimes) automatically. More details later.

In Fortran, two classic collections of such codes are:
ODEPACK: `www.netlib.org/odepack/`
RKSUITE: `www.netlib.org/ode/rksuite/`

Matlab has eight codes:

`ode23` low-order RK
`ode45` higher-order RK

`ode113` variable-order multistep
`ode23s`, `ode15s`, `ode15i`, `ode23t`, `ode23tb` variants for stiff problems etc.

These differ in numerics but have the same syntax. They all began with

L. F. Shampine and M. W. Reichelt, The Matlab ODE suite, *SIAM J. Sci. Comput.* 18 (1997), 1-22.

[handout: first 2 pages]

Or one can use visual programming – Simulink. I'm not a fan of this for serious work, but it's fascinating.

In MATLAB, if you've got it installed: `demo simulink`

IV.4 IVP solutions in Chebfun

As you know, Chebfun is a project based at Oxford in which MATLAB's usual commands for vectors are overloaded by commands for functions defined on an interval $[a, b]$. The implementation involves polynomial (or piecewise polynomial) interpolants in Chebyshev points.

In Chebfun, you can solve ODEs with a backslash command.
The simplest way to do this is:

```
N = chebop(a,b)        % define the interval [a,b]
N.op = @(x,u) ...      % define the ODE, with diff(u,k) = kth derivative of u
N.bc = ...             % boundary conditions
```

See *Exploring ODEs* and Chapters 7 and 10 of the Chebfun Guide at `www.chebfun.org`. In *Exploring ODEs*, note in particular the collection of 100 template solutions in Appendix B.

`[m26chebfun.m]`