

## Solving PDEs with PGI CUDA Fortran

### Part 4: Initial value problems for ordinary differential equations

#### Outline

ODEs and initial conditions. Explicit and implicit Euler methods. Runge-Kutta methods. Multistep Adams' predictor-corrector and Gear's BDF methods. Example: Lorenz attractor.

#### Ordinary differential equations and initial conditions

1 ordinary differential equation for 1 unknown function  $y(x)$  of 1 variable  $x$

$$dy(x)/dx \equiv y'(x) = f(x, y(x))$$

For a unique solution, the **initial condition** is required

$$y(x_0) = y_0$$

A set of  $M$  ordinary differential equations for  $M$  unknown functions  $y_m(x)$  of 1 variable  $x$

$$dy_m(x)/dx = f_m(x, y_1(x), \dots, y_M(x)), \quad m = 1, \dots, M$$

or  $dY(x)/dx \equiv Y'(x) = F(x, Y(x))$

for vector  $Y$  of unknown functions  $y_m$  and vector  $F$  of right-hand-side functions  $f_m$

For a unique solution,  $M$  initial conditions are required

$$y_m(x_0) = y_{m0} \quad \text{or} \quad Y(x_0) = Y_0$$

These are **initial value problems (IVPs)** for **ordinary differential equations (ODEs)**.

Higher-order ODEs can be rewritten into a set of 1st-order ODEs (see, e.g., Numerical Recipes).

#### Discretization

x-grid and stepsize  $x_n, \quad h_n = x_{n+1} - x_n, \quad n = 0, 1, \dots$

numerical solution  $y_n \approx y(x_n)$

Numerical methods below are, for simplicity, formulated for 1 ODE and the constant stepsize  $h$ .

However, they all also work for  $y$  replaced by  $Y$  and  $h$  replaced by  $h_n$ .

#### Explicit Euler method

the left-hand 1st-order finite-difference scheme for the 1st derivative

$$y'(x_n) \approx (y(x_{n+1}) - y(x_n))/h$$

after substitution into the ODE, we get the approximate **Euler method**

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

it is an **explicit** formula as all terms on the right-hand side are known

**accuracy: 1st-order** method (corresponds to the truncated Taylor expansion with the 0th and 1st term only)

**stability:** consider a linear problem with constant coefficients

$$y'(x) = -cy, \quad y(0) = 1, \quad c > 0 \quad (\text{solution: } y(x) = e^{-cx})$$

$$\text{Euler method: } y_{n+1} = y_n + h(-cy_n) = (1 - ch)y_n$$

but for  $h > 2/c : |y_{n+1}| > |y_n|$ , or  $|y_n| \rightarrow \infty$ ; thus, there is a **stepsize limit** due to stability

pros: a simple explicit formula

cons: low accuracy  $\Rightarrow$  higher-order explicit methods

low stability  $\Rightarrow$  implicit methods

#### Implicit Euler method

the right-hand 1st-order finite-difference scheme for the 1st derivative

$$y'(x_{n+1}) \approx (y(x_{n+1}) - y(x_n))/h$$

after substitution into the ODE, we get another Euler method

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

it is an **implicit** formula as there are references to unknown  $y_{n+1}$  on the right-hand side

again, it is only **1-st order accurate**

as above, consider the problem

$$y'(x) = -cy, \quad y(0) = 1, \quad c > 0 \quad (\text{solution: } y(x) = e^{-cx})$$

the implicit formula:  $y_{n+1} = y_n + h(-cy_{n+1}) = y_n/(1 + ch)$

thus, implicit Euler method is stable for any (positive)  $h$ , it has an infinite region of **absolute stability**

**Semi-implicit Euler method** for  $Y' = F(Y)$

solving implicit Euler method by linearization of  $F(Y)$ , similarly as in the Newton method for root finding

$$Y_{n+1} = Y_n + hF(Y_{n+1}) \approx Y_n + h[F(Y_n) + (\partial F/\partial Y)_{Y_n} \cdot (Y_{n+1} - Y_n)]$$

$$Y_{n+1} = Y_n + h[I - h(\partial F/\partial Y)]^{-1} \cdot F(Y_n)$$

i.e., in each step,  $M \times M$  (Jacobian) matrix assembly and inversion is required

### Runge-Kutta methods

- more accurate, higher-order methods for integrating ODEs
- approximate the Taylor expansion by averaging the appropriately chosen  $dy/dx$  along  $y(x)$  between  $x_n$  and  $x_{n+1}$
- **RK1**: the explicit Euler method, the simplest RK method
 
$$y_{n+1} = y_n + hk_1$$

$$k_1 = f(x_n, y_n)$$
 for  $f(x)$  independent of  $y$ , it is equivalent to the rectangle quadrature rule
- **RK2**: the 2nd-order RK method (midpoint method)
 
$$y_{n+1} = y_n + hk_2$$

$$k_1 = f(x_n, y_n), \quad k_2 = f(x_n + h/2, y_n + hk_1/2)$$
 for  $f(x)$  independent of  $y$ , it is equivalent to the trapezoidal quadrature rule
- **RK4**: the most popular, **4th-order RK method**

$$y_{n+1} = y_n + h(k_1 + 2k_2 + 2k_3 + k_4)/6$$

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + h/2, y_n + hk_1/2)$$

$$k_3 = f(x_n + h/2, y_n + hk_2/2)$$

$$k_4 = f(x_n + h, y_n + hk_3)$$
 for  $f(x)$  independent of  $y$ , it is equivalent to the Simpson's quadrature rule
- a **general p-step (explicit) RK method**

$$y_{n+1} = y_n + h \sum_{i=1}^p c_i k_i$$

$$k_1 = f(x_n, y_n)$$

$$k_i = f(x_n + \alpha_i h, y_n + h \sum_{j=1}^{i-1} \beta_{ij} k_j), \quad i = 2, \dots, p$$
- stepsize control (different  $h$  in each step): guess a step size, or make a few runs with step doubling, or use RK methods with adaptive stepsize control (e.g., Numerical Recipes, Chapter 16.2)
- pros: higher accuracy, better stability (not as much as in implicit variants) than explicit Euler
- implicit RK methods available (more stable than the explicit methods, but not infinitely stable)

### Multistep methods

A general **linear multistep method**

$$y_{n+1} = \sum_{i=1}^p a_i y_{n+1-i} + h \sum_{i=0}^p b_i f_{n+1-i} \quad \text{with } a_p \text{ or } b_p \text{ nonzero}$$

the methods are **explicit** and  $p$ -step for  $b_0=0$  and **implicit** and  $(p+1)$ -step otherwise

**Adams' family** of multistep methods

- based on polynomial approximation of  $f(x,y(x))$  between  $x_{n+1-p}$  and  $x_n$  ( $x_{n+1}$  for implicit methods) and analytical integration of the approximating polynomial between  $x_n$  and  $x_{n+1}$ 

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} N_p(x) dx = y_n + h \sum_{i=0}^p b_i f_{n+1-i}$$
- the order (i.e., the coincidence with the truncated Taylor expansion) is  $p$  for explicit and  $p+1$  for implicit methods
- the explicit  $p=1$  method is the explicit Euler, the implicit  $p=0$  method is the implicit Euler

Coefficients  $a_i$  and  $b_i$  of Adams' methods

**explicit (Adams-Bashforth) methods**

all p:	$a_1 = 1, \text{ other } a_i = 0$					
i:	0	1	2	3	4	
p=0:	$b_i$	–				
p=1:	$b_i$	0	1			
p=2:	$2b_i$	0	3	–1		
p=3:	$12b_i$	0	23	–16	5	
p=4:	$24b_i$	0	55	–59	37	–9

e.g., explicit p=2:  $y_{n+1} = y_n + h(3f_n - f_{n-1})/2$

**implicit (Adams-Moulton) methods**

all p:	$a_1 = 1, \text{ other } a_i = 0$					
i:	0	1	2	3	4	
p=0:	$b_i$	1				
p=1:	$2b_i$	1	1			
p=2:	$12b_i$	5	8	–1		
p=3:	$24b_i$	9	19	–5	1	
p=4:	$720b_i$	251	646	–264	106	–19

implicit p=1:  $y_{n+1} = y_n + h(f_{n+1} + f_n)/2$

The **predictor-corrector algorithm**: conventional application of Adams' methods

step P (predictor): applies an explicit Adams' formula of a given order,  $y_{\text{new}} = y_{n+1}[\text{explicit}](x_{n+1})$

step E (evaluation): updates  $f_{n+1} = f(x_{n+1}, y_{\text{new}})$

step C (corrector): applies an implicit Adams' formula of the same order,  $y_{\text{new}} = y_{n+1}[\text{implicit}](x_{n+1})$

steps E and C can be repeated: variants **PEC, PECE, P(EC)<sup>2</sup>E**

i.e., a predictor extrapolates  $f$  into  $x_{n+1}$ , a corrector makes use of this value for polynomial interpolation

- initialization of multistep methods by their lower-order relatives or by RK methods
- the predictor-corrector algorithm is essentially explicit, its stability is therefore worse than that of the corrector
- adaptive stepsize control is laborious

**Backward differentiation formulas (BDFs, Gear's method)**

- based on polynomial approximation of  $y(x)$  between  $x_{n+1-p}$  and  $x_{n+1}$  and analytical differentiation of the approximating polynomial at  $x_{n+1}$ 

$$y_{n+1} = \sum_{i=1}^p a_i y_{n+1-i} + h b_0 f_{n+1}$$
- implicit p-step methods of the order p; for p=1: implicit Euler method
- BDFs combined with the Newton method are known to have excellent stability (for  $p \leq 6$ )
- inevitable for **stiff problems** with two or more very different scales of the variable x on which the unknowns y are changing (stability conditions require to accommodate to the fastest scale, i.e., with small stepsize, while the process under study usually develops on the slowest scale, i.e., too many small steps would be necessary with explicit methods)

Coefficients  $a_i$  and  $b_i$  of Gear's methods

	i:	1	2	3	4	5	6	i:	0	> 0
p=1:	$a_i$	1						$b_i$	1	0
p=2:	$3a_i$	4	-1					$3b_i$	2	0
p=3:	$11a_i$	18	-9	2				$11b_i$	6	0
p=4:	$25a_i$	48	-36	16	-3			$25b_i$	12	0
p=5:	$137a_i$	300	-300	200	-75	12		$137b_i$	60	0
p=6:	$147a_i$	360	-450	400	-225	72	-10	$147b_i$	60	0

e.g., p=2:  $y_{n+1} = (4y_n - y_{n-1})/3 + 2f_{n+1}/3$

**On the crossroads**

**Euler methods** are extremely simple but inaccurate, too; the implicit variant is necessary when stability matters, i.e., when larger stepsize is required than a stability condition allows.

**Runge-Kutta explicit methods** are simple and fast enough both to code and to run, as each step requires just evaluation of an explicit formula, and for many problems, they are accurate enough. However, they are explicit and stepsize is limited.

**Predictor-corrector** implementation, including stepsize adaptivity, is rather an artwork, but it was done and can be reused from available packages. Still, stability is limited.

**Backward differentiation formulas**, as a multistep method, share many features with predictor-corrector methods, however, for their excellent stability, they are inevitable for stiff problems.

**And an example: Lorenz attractor**

- a problem of the **2D convection** in the atmosphere, mathematically simplified as much as possible
- a fully deterministic system with **chaotic behavior**
- a simplified problem: 3 ODEs for temporal evolution of **3 variables** (coefficients of eigenvalue expansions of the stream function and temperature anomalies) in the **3D (phase) space**

$$\begin{aligned} dA/d\tau &= P(B - A) \\ dB/d\tau &= rA - B - AC \\ dC/d\tau &= -bC + AB \end{aligned}$$

where A is a stream-function coefficient, B and C coefficients of temperature anomalies, P the Prandtl number,  $r=Ra/Ra_{CR}$  with Ra the Rayleigh number,  $b = 4/(1 + (2/\lambda)^2)$  corresponds to the size of a convection roll and  $\tau$  is nondimensionalized time

- parameters used by Lorenz (1963):  $P=10$ ,  $r=28$  and  $b=8/3$ , a sufficient time interval: 0..20
- temporal solutions roll around two fixed points (the **strange attractors**) along a lemniscate-shaped trajectory (like  $\infty$ )
- physically: the boundary layer of a convection cell grows, at some point it becomes unstable, convection resumes, either as a clockwise or counterclockwise roll: chaotic behavior in a deterministic system
- popular vizualization: A-B-C **phase portraits**

**Source codes**

- CPU: an arbitrary A-B-C point undergoes NT Runge-Kutta time steps, they are recorded and plotted
- CPU:  $NX \times NY \times NZ$  points, spread within a 3D cube, undergo NT time steps, each independent of others, only final positions of all points are recorded and plotted

- GPU: the previous case with one kernel
  - GPU: the previous case, now with a smaller kernel called repeatedly
- Goals: a massively parallel compute-bound kernel, SP/DP execution times, avoiding kernel execution timeout, stability limits of explicit schemes

### Links and references

#### Numerical methods

Ascher U. M. and Petzold L. R., Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, 1998

Press W. H. et al., **Numerical Recipes** in Fortran 77: The Art of Scientific Computing, Second Edition, Cambridge, 1992

Chapter 16.1: Runge-Kutta method

Chapter 16.2: Adaptive stepsize control for Runge-Kutta

Chapter 16.6: Stiff sets of equations

<http://www.nr.com>, PDFs available at <http://www.nrbook.com/a/bookfpdf.php>

#### Lorenz attractor

Schubert G. et al., Mantle Convection in the Earth and Planets, Cambridge, 2001, p. 332–337

[http://ebookey.org/Mantle-Convection-in-the-Earth-and-Planets\\_661884.html](http://ebookey.org/Mantle-Convection-in-the-Earth-and-Planets_661884.html)