Rezolvarea ecuaţiilor şi sistemelor de ecuaţii diferenţiale ordinare (II)

Metode multipas

Prof.dr.ing. Gabriela Ciuprina

Universitatea "Politehnica" Bucureşti, Facultatea de Inginerie Electrică

Suport didactic pentru disciplina *Metode numerice*, 2016-2017

Cuprins

Unipas vs. multipas

$$\frac{dx}{dt} = f(x,t) \quad \Rightarrow \quad x(t_B) - x(t_A) = \int_{t_A}^{t_B} f(x,t)\,dt$$

$$x(t_B) = x(t_A) + \int_{t_A}^{t_B} f(x,t)\,dt$$

Soluţia se va calcula numeric în punctele discrete $t_0 = t0, t_1, \ldots, t_n = T$

Unipas

$t_A = t_j \quad t_A = t_{j+1}$

$$x(t_{j+1}) = x(t_j) + \int_{t_j}^{t_{j+1}} f(x,t)\,dt$$

$$x_{j+1} = x_j + I \quad I \approx \int_{t_j}^{t_{j+1}} f(x,t)\,dt$$

$j = 0, n-1,$ x_0 este cunoscut;

Multipas

$t_A = t_j \quad t_A = t_{j+m}$

$$x(t_{j+m}) = x(t_j) + \int_{t_j}^{t_{j+m}} f(x,t)\,dt$$

$$x_{j+m} = x_j + I \quad I \approx \int_{t_j}^{t_{j+m}} f(x,t)\,dt$$

$j = 0, n-m,$ x_0 este cunoscut;

Unipas vs. multipas

Metode unipas - folosesc informaţii din intervalul $[t_j, t_{j+1}]$

- Metode θ

$$-x_j + x_{j+1} = h\left[\theta f(x_j, t_j) + (1-\theta) f(x_{j+1}, t_{j+1})\right]$$

  Dacă $\theta = 1$ - metoda este explicită (Euler explicit)
  Dacă $\theta \neq 1$ - metoda este implicită ($\theta = 0$ - Euler implicit, $\theta = 1/2$ - trapeze); necesită în general rezolvarea unei ecuaţii algebrice neliniare la fiecare pas de integrare.

- Metode Runge-Kutta

Unipas vs. multipas

Metode unipas - folosesc informaţii din intervalul $[t_j, t_{j+1}]$

- Metode θ

$$-x_j + x_{j+1} = h\left[\theta f(x_j, t_j) + (1-\theta) f(x_{j+1}, t_{j+1})\right]$$

- Metode Runge-Kutta

$$-x_j + x_{j+1} = I$$

unde $I \approx \int_{t_j}^{t_{j+1}} f(x,t)\,\mathrm{d}t$

$$I = h \sum_{i=1}^{\nu} b_i f(x(t_j + c_i h), t_j + c_i h)$$

unde $x(t_j + c_i h)$ nu sunt cunoscute şi sunt aproximate cu formule liniare ale unor valori calculate succesiv.

Unipas vs. multipas

Metode unipas - folosesc informaţii din intervalul $[t_j, t_{j+1}]$

- Metode θ

$$-x_j + x_{j+1} = h\left[\theta f(x_j, t_j) + (1-\theta) f(x_{j+1}, t_{j+1})\right]$$

- Metode Runge-Kutta - **explicite**

$$x_{j+1} = x_j + h\Phi, \quad j = 0, \ldots, n-1 \tag{1}$$

$$\Phi = \sum_{i=1}^{\nu} b_i K_i \tag{2}$$

$$K_1 = f(x_j, t_j) \tag{3}$$

$$K_i = f(x_j + h\sum_{p=1}^{i-1} a_{ip} K_p, t_j + c_i h) \quad i = 2, \ldots, \nu \tag{4}$$

Unipas vs. multipas

Metode unipas - folosesc informaţii din intervalul $[t_j, t_{j+1}]$

- Metode θ

$$-x_j + x_{j+1} = h\left[\theta f(x_j, t_j) + (1-\theta) f(x_{j+1}, t_{j+1})\right]$$

- Metode Runge-Kutta - **implicite**

$$x_{j+1} = x_j + h\Phi, \quad j = 0, \ldots, n-1 \tag{5}$$

$$\Phi = \sum_{i=1}^{\nu} b_i K_i \tag{6}$$

$$K_1 = f(x_j, t_j) \tag{7}$$

$$K_i = f(x_j + h\sum_{p=1}^{\nu} a_{ip} K_p, t_j + c_i h) \quad i = 2, \ldots, \nu \tag{8}$$

Unipas vs. multipas

Metode multipas - folosesc informaţii din intervalul $[t_j, t_{j+m}]$
Metode liniare multipas[1]
Calculul unui pas nou x_{j+m} se face folosind relaţia

$$a_0 x_j + a_1 x_{j+1} + \cdots a_m x_{j+m} = h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_m f(x_{j+m}, t_{j+m})\right]$$

- a_i şi b_i sunt aleşi convenabil (convergenţă);
- $a_m = 1$;
- dacă $b_m = 0$ metoda este explicită;
- dacă $b_m \neq 0$ metoda este implicită;
- cazul $m = 1$, $a_0 = -1$ (din motive de convergenţă), $b_0 = \theta$, $b_1 = 1 - \theta \Rightarrow$ metode unipas de tip θ.

[1] Există şi metode neliniare multipas.

Formule de integrare numerică Newton-Cotes

Algoritmii multipas pentru rezolvarea ODE se bazează pe formulele de cuadratură Newton-Cotes (formule de integrare numerică scrise pentru reţele de discretizare uniformă).

1. Formule NC "închise" - folosesc inclusiv valorile în capete. Se folosesc în calculul integralelor definite şi în rezolvarea ODE cu metodele multipas implicite.
2. Formule NC "deschise" - nu folosesc valorile în capete. Se folosesc în rezolvare ODE cu metodele multipas explicite.

Formule Newton-Cotes închise

Grid uniform $x_0, x_1, \ldots, x_n$, pas h
$f_i = f(x_i)$
Formulele conţin f_0 şi f_n.

n (gradul polinomului)	Pasul h	Numele uzual al formulei	Formula	Eroarea locală
1	$x_1 - x_0$	trapezului	$\frac{h}{2}(f_0 + f_1)$	$O(h^3)$
2	$x_1 - x_0 = \frac{x_2 - x_0}{2}$	Simpson 1/3	$\frac{h}{3}(f_0 + 4f_1 + f_2)$	$O(h^5)$
3	$x - 1 - x_0 = \frac{x_3 - x_0}{3}$	Simpson 3/8	$\frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3)$	$O(h^5)$

Formule Newton-Cotes deschise

Grid uniform $x_0, x_1, \ldots, x_n$, pas h
$f_i = f(x_i)$
Formulele **nu** conţin f_0 şi f_n.

Gradul polinomului	Pasul h	Numele uzual al formulei	Formula	Eroarea locală
0	$x_1 - x_0 = \frac{x_2 - x_0}{2}$	regula dreptunghiului sau punctului din mijloc	$2hf_1$	$O(h^3)$
1	$x_1 - x_0 = \frac{x_3 - x_0}{3}$	trapezului	$\frac{3h}{2}(f_1 + f_2)$	$O(h^3)$
2	$x_1 - x_0 = \frac{x_4 - x_0}{4}$	regula lui Milne	$\frac{4h}{3}(2f_1 - f_2 + 2f_3)$	$O(h^5)$

Metode Milne explicite

$$x_{j+m} = x_j + I \quad I \approx \int_{t_j}^{t_{j+m}} f(x,t)\,\mathrm{d}t$$

$j = 0, n-m$, x_0 este cunoscut;

Integrala I se aproximează cu formule Newton-Cotes deschise.

$m = 2 \Rightarrow$ NC cu 1 punct interioar

$$x_{j+2} = x_j + 2hf(x_{j+1}, t_{j+1})$$

Metode Milne explicite

$$x_{j+m} = x_j + I \quad I \approx \int_{t_j}^{t_{j+m}} f(x,t)\,\mathrm{d}t$$

$j = \overline{0, n-m}$, x_0 este cunoscut;

Integrala I se aproximează cu formule Newton-Cotes deschise.

$m = 3 \Rightarrow$ NC cu 2 puncte interioar

$$x_{j+3} = x_j + \frac{3h}{2}(f(x_{j+1}, t_{j+1} + f(x_{j+2}, t_{j+2})$$

Metode Milne explicite

$$x_{j+m} = x_j + I \quad I \approx \int_{t_j}^{t_{j+m}} f(x,t)\,\mathrm{d}t$$

$j = \overline{0, n-m}$, x_0 este cunoscut;
Integrala I se aproximează cu formule Newton-Cotes deschise.
$m = 4 \Rightarrow$ NC cu 3 puncte interioare

$$x_{j+4} = x_j + \frac{4h}{3}(2f(x_{j+1}, t_{j+1}) - f(x_{j+2}, t_{j+2}) + 2f(x_{j+3}, t_{j+3}) + O(h^5))$$

Adams-Bashforth

Formula generală a metodelor liniare multipas:

$$a_0 x_j + a_1 x_{j+1} + \cdots a_m x_{j+m} = h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_m f(x_{j+m}, t_{j+m})\right]$$

$\Rightarrow x_{j+m}$

unde $a_m = 1$

Familia Adams-Bashfort:

- $a_{m-1} = -1$
- $a_i = 0, \forall i < m - 1$
- $b_m = 0$ (metodă explicită)

$$x_{j+m} = x_{j+m-1} + h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_{m-1} f(x_{j+m-1}, t_{j+m-1})\right]$$

$$x_{j+m} = x_{j+m-1} + \sum_{i=0}^{m-1} b_i f(x_{j+i}, t_{j+i})$$

Adams-Bashforth

Familia Adams-Bashfort:

$$x_{j+m} = x_{j+m-1} + \sum_{i=0}^{m-1} b_i f(x_{j+i}, t_{j+i})$$

b_i se determină astfel încât metoda are ordinul m

Metodă, ordin	b_0	b_1	b_2	b_3
AB cu 1 pas, ordin 1	1			
AB cu 2 paşi, ordin 2	3/2	-1/2		
AB cu 3 paşi, ordin 3	23/12	-16/12	5/12	
AB cu 4 paşi, ordin 4	55/24	-59/24	37/24	-9/24

AB cu un pas este Euler explicit.

Metode Milne implicite

$$x_{j+m} = x_j + I \quad I \approx \int_{t_j}^{t_{j+m}} f(x,t)\,\mathrm{d}t$$

$j = \overline{0, n-m}$, $\quad x_0$ este cunoscut;

Integrala I se aproximează cu formule Newton-Cotes închise.

Exemplu: $m = 2 \Rightarrow$ NC cu 3 puncte

$$x_{j+2} = x_j + \frac{h}{3}(f(x_j, t_j + 4f(x_{j+1}, t_{j+1} + f(x_{j+2}, t_{j+2})$$

Ecuaţie neliniară $\Rightarrow$ are nevoie de o estimare iniţială pentru x_{j+2}. Se poate face cu o formulă Milne explicită.

Evaluarea primelor puncte se face cu metode unipas.

Adams-Moulton

Formula generală a metodelor liniare multipas:

$$a_0 x_j + a_1 x_{j+1} + \cdots a_m x_{j+m} = h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_m f(x_{j+m}, t_{j+m})\right]$$

$\Rightarrow x_{j+m}$ unde $a_m = 1$

Familia Adams-Moulton:

- $a_{m-1} = -1$
- $a_i = 0, \forall i < m-1$ item $b_m \neq 0$ (metodă implicită)

$$x_{j+m} = x_{j+m-1} + h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_m f(x_{j+m}, t_{j+m})\right]$$

$$x_{j+m} = x_{j+m-1} + \sum_{i=0}^{m} b_i f(x_{j+i}, t_{j+i})$$

Adams-Moulton

Prin eliminarea restricţiei $b_m = 0$ de la AB, metodele devin implicite, şi o metodă cu m paşi poate ajunge la ordinul $m + 1$.

Metodă, ordin	b_0	b_1	b_2	b_3
AM cu 1 pas, ordin 1	0	1		
AM cu 1 pas, ordin 2	1/2	1/2		
AM cu 2 paşi, ordin 3	15/12	8/12	-1/12	
AM cu 3 paşi, ordin 4	9/24	19/24	-5/24	1/24

AM cu un pas este Euler implicit (ordinul 1) sau metoda trapezelor (ordinul 2).

BDF (Gear)

Formula generală a metodelor liniare multipas:

$$a_0 x_j + a_1 x_{j+1} + \cdots a_m x_{j+m} = h\left[b_0 f(x_j, t_j) + b_1 f(x_{j+1}, t_{j+1}) + \cdots b_m f(x_{j+m}, t_{j+m})\right]$$

$\Rightarrow x_{j+m}$

Dacă $b_i = 0 \quad \forall i < m$ şi notând $b_m = \beta$ - metodele se numesc **BDF**[2]

$$\sum_{i=0}^{m-1} a_i x_{j+i} + x_{j+m} = h\beta f(x_{j+m}, t_{j+m}) \quad \Rightarrow x_{j+m}$$

unde $h = t_{i+m} - t_{i+m-1}$

[2] *Backward Differentiation Formula*

BDF

Coeficienţii undei metode BDF de ordin m se determină pornind de la polinomul Lagrange de ordin m, notat $p_{i,m}(t)$ care trece prin punctele $(t_i, x_i), \ldots, (t_{i+m}, x_{i+m})$.

$$x'(t_{i+m}) \approx p'(t_{i+m})$$

şi

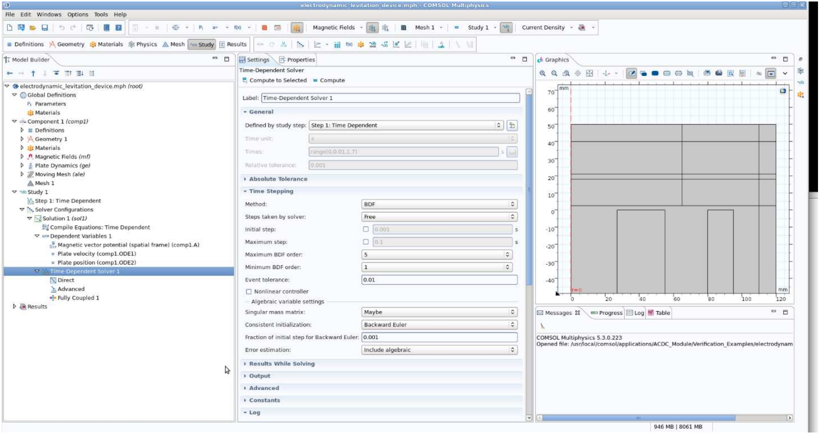
$$x'(t_{i+m}) = f(x(t_{i+m}), t_{i+m})$$

BDF

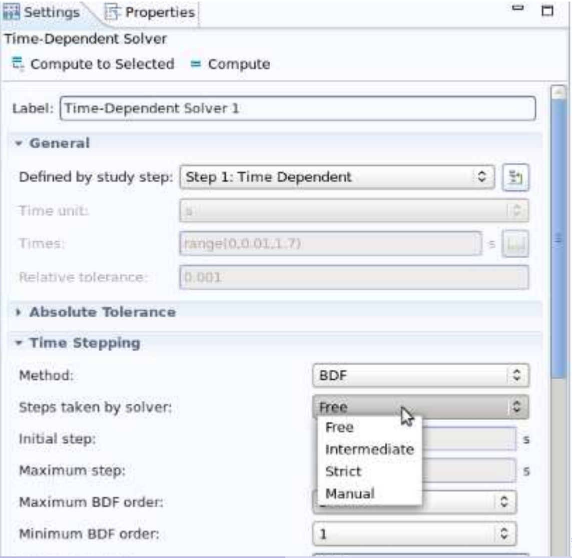
Metodă, ordin	β	a_0	a_1	a_2	a_3
BDF cu 1 pas, ordin 1	1	-1	1		
BDF cu 2 paşi, ordin 2	2/3	1/3	-4/3	1	
BDF cu 3 paşi, ordin 3	6/11	-2/11	9/11	-18/11	1

BDF cu un pas este Euler implicit (ordinul 1).

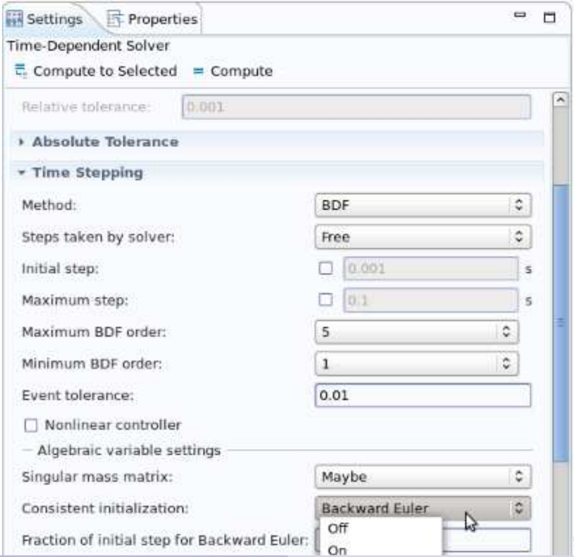
COMSOL


electrodynamic_levitation_device.mph - COMSOL Multiphysics
File Edit Windows Options Tools Help
Model Builder
Settings
Properties
Time-Dependent Solver
Compute to Selected
Compute
Label: Time-Dependent Solver 1
General
Defined by study step: Step 1: Time Dependent
Absolute Tolerance
Time Stepping
Method: BDF
Steps taken by solver: Free
Initial step:
Maximum step:
Maximum BDF order: 5
Minimum BDF order: 1
Event tolerance: 0.01
Nonlinear controller
Algebraic variable settings
Singular mass matrix: Maybe
Consistent initialization: Backward Euler
Fraction of initial step for Backward Euler: 0.001
Error estimation: Include algebraic
Results While Solving
Output
Advanced
Constants
Log
Graphics
Messages
Progress
Log
Table
COMSOL Multiphysics 5.3.0.223
946 MB | 8061 MB

COMSOL


Settings
Properties
Time-Dependent Solver
Compute to Selected
Compute
Label:
Time-Dependent Solver 1
General
Defined by study step:
Step 1: Time Dependent
Time unit:
Times:
range(0,0.01,1.7)
Relative tolerance:
0.001
Absolute Tolerance
Time Stepping
Method:
BDF
Steps taken by solver:
Free
Free
Intermediate
Strict
Manual
Initial step:
Maximum step:
Maximum BDF order:
Minimum BDF order:
1

COMSOL


Settings
Properties
Time-Dependent Solver
Compute to Selected
Compute
Relative tolerance:
0.001
Absolute Tolerance
Time Stepping
Method:
BDF
Steps taken by solver:
Free
Initial step:
0.001
s
Maximum step:
0.1
s
Maximum BDF order:
5
Minimum BDF order:
1
Event tolerance:
0.01
Nonlinear controller
Algebraic variable settings
Singular mass matrix:
Maybe
Consistent initialization:
Backward Euler
Off
On
Fraction of initial step for Backward Euler:

Matlab (non-stiff)

https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.

Matlab (non-stiff) https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

- Single-step

ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair.

ode23 is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness.

- Multi-step

ode113 is a variable-step, variable-order Adams-Bashforth-Moulton PECE solver of orders 1 to 13. The highest order used appears to be 12, however, a formula of order 13 is used to form the error estimate and the function does local extrapolation to advance the integration at order 13. It may be more efficient than ode45 at stringent tolerances or if the ODE function is particularly expensive to evaluate.

Matlab (stiff) https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

ode15s	Stiff	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s		Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
ode23t		Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb		Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude

Matlab (stiff) https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

- Single-step

ode23s is based on a modified Rosenbrock formula of order 2. Because it is a single-step solver, it may be more efficient than ode15s at solving problems that permit crude tolerances or problems with solutions that change rapidly. It can solve some kinds of stiff problems for which ode15s is not effective. The ode23s solver evaluates the Jacobian during each step of the integration, so supplying it with the Jacobian matrix is critical to its reliability and efficiency.

ode23t is an implementation of the trapezoidal rule using a "free" interpolant. This solver is preferred over ode15s if the problem is only moderately stiff and you need a solution without numerical damping. ode23t also can solve differential algebraic equations (DAEs)

- Multi-step ode15s, ode23tb

Matlab (stiff) https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

- Single-step ode23s, ode23t
- Multi-step

ode15s is a variable-step, variable-order (VSVO) solver based on the numerical differentiation formulas (NDFs) of orders 1 to 5. Optionally, it can use the backward differentiation formulas (BDFs, also known as Gearş method) that are usually less efficient. Like ode113, ode15s is a multistep solver. Use ode15s if ode45 fails or is very inefficient and you suspect that the problem is stiff, or when solving a differential-algebraic equation (DAE).

ode23tb is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a trapezoidal rule step as its first stage and a backward differentiation formula of order two as its second stage. By construction, the same iteration matrix is used in evaluating both stages. Like ode23s and ode23t, this solver may be more efficient than ode15s for problems with crude tolerances.

Matlab (fully-implicit) https://ch.mathworks.com/help/matlab/math/choose-an-ode-solver.html

ode15i	Fully implicit	Low	Use ode15i for fully implicit problems $f(t,y,y')=0$ and for differential algebraic equations (DAEs) of index 1.

- Multi-step

ode15i is a variable-step, variable-order (VSVO) solver based on the backward differentiation formulas (BDFs) of orders 1 to 5. ode15i is designed to be used with fully implicit differential equations and index-1 differential algebraic equations (DAEs). The helper function decic computes consistent initial conditions that are suitable to be used with ode15i.

Referinţe

- [Cheney08] W.Cheney, D.Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole publishing Company,2000. (Capitolele 10 şi 11)

  Disponibilă la http://www.physics.brocku.ca/Courses/5P10/References/cheneykincaid.pdf

- [Press02] W.H.Press, S.A.Teukolsky, W.T. etterling, B.P. Flannery, *Numerical Recipies in C*, 2002. (Capitolul 16)

  Disponibilă la https://www2.units.it/ipl/students_area/imm2/files/Numerical_Recipes.pdf

- [Strang&Moler15] *Introduction to Differential Equations and the MATLAB ODE Suite* - Open Courseware at MIT

  Disponibil la http://ocw.mit.edu/RES-18-009F15 sau https://www.youtube.com/watch?v=ZvL88xqYSak

- [Trefethen18] N.Trefethen, *Exploring ODEs*, SIAM 2018.

  Disponibil la https://people.maths.ox.ac.uk/trefethen/books.html