MA3457/CS4033:

Numerical Methods for Calculus and Differential Equations

Course Materials

P A R T   V

B’14
2014-2015

The Shooting Method – MATLAB Implementation

Function **BVP_shooting**

- specifies BCs,
- calls the function **RK4_sys** (in which *the Runge-Kutta method of order 4 adjusted to the system of ODE* is implemented) and gets the solution for the system from there, and
- calculates the linear combination of the solutions for two IVP and plots the results

Function **RK4_sys**

- is similar to **RK4**, but it *relies on MATLAB ability to treat all variables as vectors*

Function **fivp_sys**

- specifies the DE to deal with – consists of the RHSs of the system of four ODEs.

The Shooting Method – MATLAB Scripts (1)

LIBRARY OF MATLAB PROCEDURES

`BVP_shooting`

Shooting method for linear BVP – specifies BC and calles **`RK4_sys`**; to be used with **`fivp_sys`**

```
% Linear shooting method for an ODE specified in fivp_sys
%
clear all
ya = 2; yb = 5/3; z0 = [ ya 0 0 1 ];
a = 0; b = 1; tspan = [ 0 1 ];
[ x z ] = RK4_sys('fivp_sys', tspan, z0, 10)
[ n m ] = size(z)
y(1:n,1) = z(1:n,1) + (yb - z(n,1))*z(1:n,3)./z(n,3)
plot(x, y)
```

The Shooting Method – MATLAB Scripts (2)

LIBRARY OF MATLAB PROCEDURES

RK4_sys

Solves systems of ODEs by the Runge-Kutta method of order 4

```
function [x, u] = RK4_sys(f, tspan, u0, n)
%
% The procedure solves a system of ODE-IVP using the Runge-Kutta
% method of order 4.  In function f(x, u):
%   input:  column vector x and row vector u
%   output: column vector of values for u'
%
% Interval of interest: tspan = [a, b]
%
a = tspan(1); b = tspan(2); h = (b-a)/n;
x = (a+h : h : b)';
k1 = h*feval(f, a, u0)';
k2 = h*feval(f, a+h/2, u0+k1/2)';
k3 = h*feval(f, a+h/2, u0+k2/2)';
k4 = h*feval(f, a+h, u0+k3)';
u(1, :) = u0 + k1/6 + k2/3 + k3/3 + k4/6;
%
for i = 1 : n-1
    k1 = h*feval(f, x(i), u(i, :))';
    k2 = h*feval(f, x(i)+h/2, u(i, :)+k1/2)';
    k3 = h*feval(f, x(i)+h/2, u(i, :)+k2/2)';
    k4 = h*feval(f, x(i)+h, u(i, :)+k3)';
    u(i+1, :) = u(i, :) + k1/6 +k2/3 +k3/3 +k4/6;
end
%
x = [a
     x]
u = [u0
     u];
```

The Shooting Method – MATLAB Scripts (3)

LIBRARY OF MATLAB PROCEDURES

`fivp_sys`

Specifies the ODE for solving by the linear shooting method; to be used with **`BVP_shooting`**

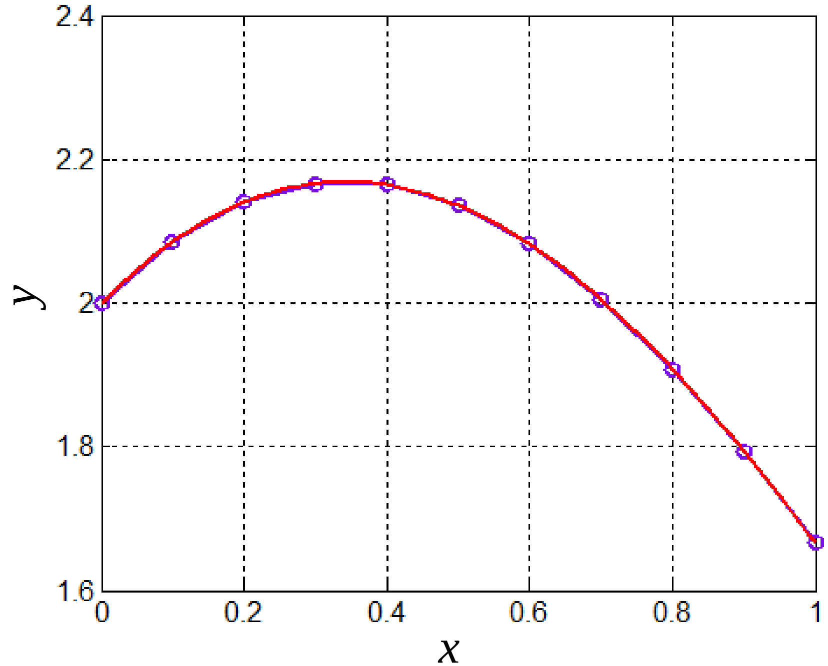
```
% RHS of system of four 1st-order ODE-IVP
%
function f_i_s=fivp_sys(x, z)
f_i_s = [z(2)
        z(2).*2.*x./(x.^2 + 1) - z(1).*2/(x.^2 + 1) + x.^2 + 1
        z(4)
        z(4).*2.*x./(x.^2 + 1) - z(3).*2/(x.^2 +1)];
```

Shooting Method for Linear BVPs (1)

BVP:

ODE: $$y'' = \frac{2x}{x^2+1}y' - \frac{2}{x^2+1}y + x^2 + 1$$

Dirichlet BCs: $y(0) = 2,\ y(1) = 5/3$



$y = x^4/6 - 3x^2/2 + x + 2$ – exact solution

shooting method

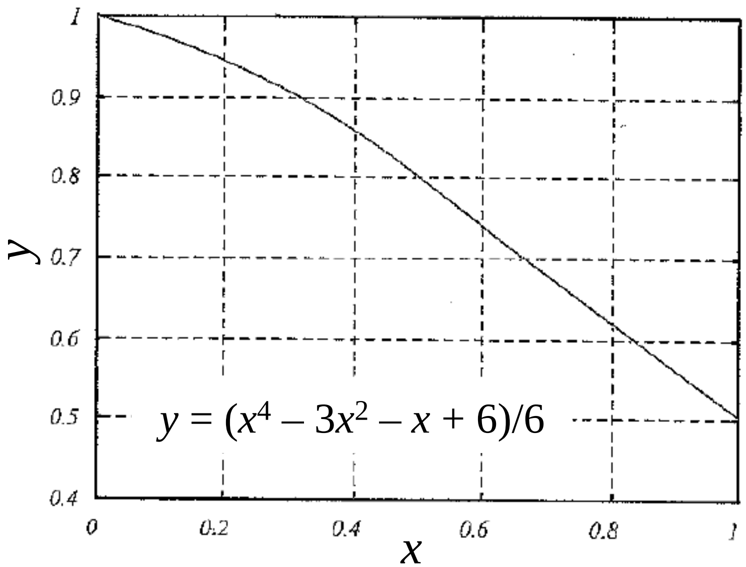
Shooting Method for Linear BVPs (2)

BVP:

ODE: $$y'' = \frac{2x}{x^2+1}y' - \frac{2}{x^2+1}y + x^2 + 1$$

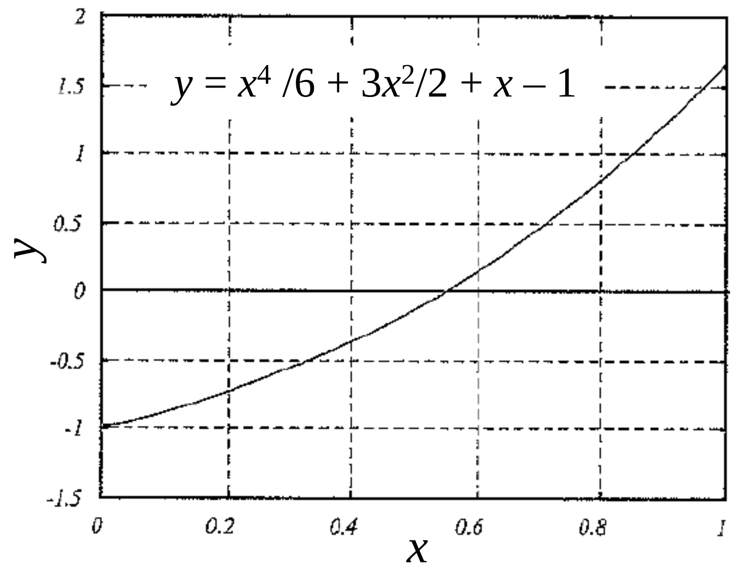
General/Mixed BCs at $x = b$:

$$y(0) = 1,\ y'(1) + y(1) = 0$$



General/Mixed BCs at $x = a$ and $x = b$:

$$y'(0) + y(0) = 0;\ y'(1) - y(1) = 3$$

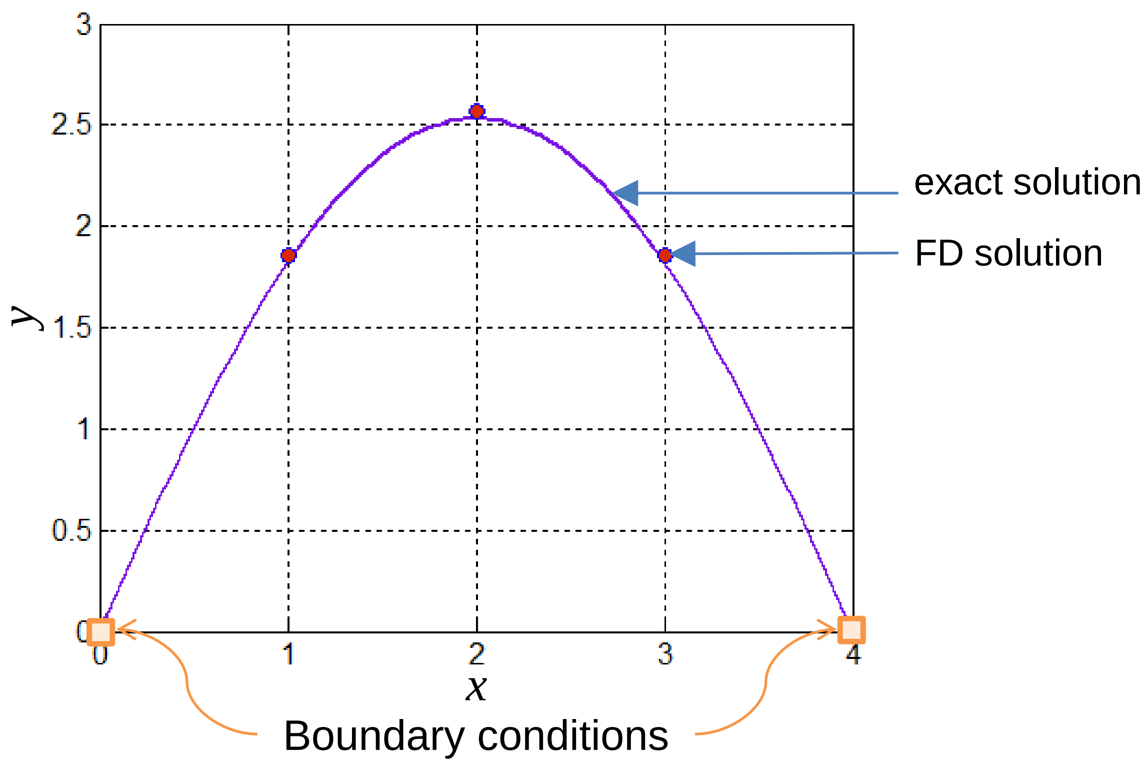


Exact solutions and numerical solutions (by the Shooting Method) are indistinguishable!

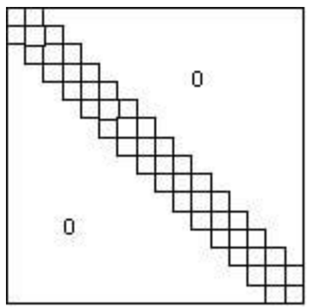
Finite-Difference Method for Linear BVPs

BVP:
- ODE: $y'' = y + x(x-4)$
- Dirichlet BCs: $y(0) = y(4) = 0$

FD Solution with n = 4



Tridiagonal Matrix



$$\begin{bmatrix} \mathbf{B}_1 & \mathbf{C}_1 & & & \cdots & & 0 \\ \mathbf{A}_2 & \mathbf{B}_2 & \mathbf{C}_2 & & & & \\ & \ddots & \ddots & \ddots & & & \vdots \\ & & \mathbf{A}_k & \mathbf{B}_k & \mathbf{C}_k & & \\ \vdots & & & \ddots & \ddots & \ddots & \\ & & & & \mathbf{A}_{n-1} & \mathbf{B}_{n-1} & \mathbf{C}_{n-1} \\ 0 & & \cdots & & & \mathbf{A}_n & \mathbf{B}_n \end{bmatrix}$$

<u>Size of the matrix</u>: (n-1)x(n-1)

<u>Number of terms in each line</u>: 3

WHY?

- <u>*In order to cover the whole interval with the Central Difference Formula*</u> (CDF) <u>index *i* runs from 1 to *n*-1</u>, where *n* is a number of subdivisions
- 3 is the number of points in which the CDF approximate a 2nd order derivative.

Finite Difference Method – MATLAB Scripts (1)

LIBRARY OF MATLAB PROCEDURES

BVP_FD

Solves a BVP $y'' = p(x)y' + q(x)y + r(x)$ with a Dirichlet boundary condition on the interval $[a, b]$ by the finite-difference method (including solving a tridiagonal system using the Thomas method) (to be used with **Trid_Thomas**).

```
% Numerical solution of a linear ODE-BVP by the FD method.
% Equation:  y_xx = p(x) y_x + q(x) y + r(x)
% Interval:  aa <= x <= bb
% Boundary conditions:  y(aa) = ya, y(bb) = yb
%
% ******* Problem definition
% DE: y_xx = 2y_x - 2y + 0
aa = 0; bb = 3; n = 300;
% Define p(x), q(x), r(x)
p = 2*ones(1, n-1); q = -2*ones(1, n-1); r = zeros(1, n-1);
% Boundary conditions
ya = 0.1; yb = 0.1*exp(3)*cos(3);
%
% Parameters and gridpoints x(1), ..., x(n-1)
h = (bb - aa)/n; h2 = h/2; hh = h*h;
x = linspace(aa+h, bb, n);
%
% Upper diagonal (a), diagonal (d), lower diagonal (b)
a = zeros(1, n-1); b = a;
a(1:n-2) = 1 - p(1,1:n-2)*h2; d = -(2 + hh*q);
b(2:n-1) = 1 + p(1,2:n-1)*h2;
%
% Right hand side (c)
c(1) = hh*r(1) - (1 + p(1)*h2)*ya;
c(2:n-2) = hh*r(2:n-2);
c(n-1) = hh*r(n-1) - (1 - p(n-1)*h2)*yb;
%
% Solution of tridiagonal system
y = Trid_Thomas(a, d, b, c);
xx = [aa x]; yy = [ya y yb];
out = [xx' yy']; disp(out)
%
plot(xx, yy, '-'), grid on, hold on
%
% Graph exact solution (if known)
% plot(xx, 0.1*exp(xx).*cos(xx))
% hold off
```

Finite Difference Method – MATLAB Scripts (2)

LIBRARY OF MATLAB PROCEDURES

Trid_Thomas

Solves a tridiagonal system of linear equations using the Thomas method.

```
function x = Trid_Thomas(a, d, b, r)
%
% The procedure solves matrix equation Ax = b
% where A is a tridiagonal matrix
%
% Input:
%   a - upper diagonal of matrix A, a(n) = 0
%   d - diagonal of matrix A
%   b - lower diagonal of matrix A, b(1) = 0
%   r - right-hand side of equation
%
n = length(d);
a(1) = a(1)/d(1);
r(1) = r(1)/d(1);
%
for i = 2 : n-1
  denom = d(i) - b(i)*a(i-1);
  if (denom == 0), error ('Zero in denominator'), end
  a(i) = a(i)/denom;
  r(i) = (r(i) - b(i)*r(i-1))/denom;
end
%
r(n) = (r(n) - b(n)*r(n-1))/(d(n) - b(n)*a(n-1));
x(n) = r(n);
for i = n-1: -1 : 1
  x(i) = r(i) - a(i)*x(i+1);
end
```

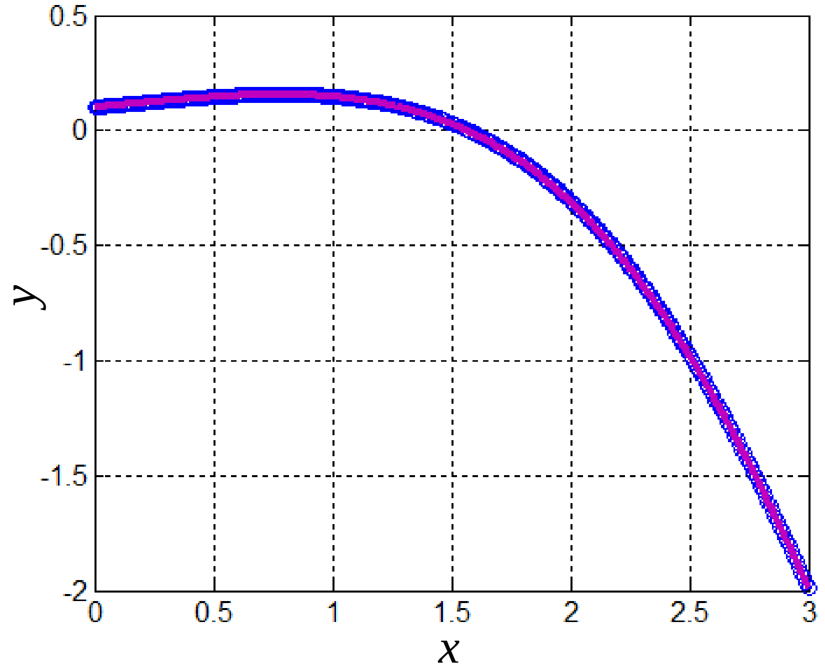
Finite-Difference Method for Linear BVPs (2)

BVP: $y'' = 2y' - 2y,\ \ y(0) = 0.1,\ \ y(3) = 0.1e^3\cos(3)$

```
>> BVP_FD
         0    0.1000
    0.0100    0.1010
    0.0200    0.1020
    0.0300    0.1030
    0.0400    0.1040
    0.0500    0.1050
    0.0600    0.1060
    0.0700    0.1070
    0.0800    0.1080
    0.0900    0.1090
    0.1000    0.1100
    0.1100    0.1110
    0.1200    0.1119
    0.1300    0.1129
    0.1400    0.1139
    0.1500    0.1149
    0.1600    0.1159
    0.1700    0.1168
    0.1800    0.1178
    0.1900    0.1188
    0.2000    0.1197
    0.2100    0.1207
    0.2200    0.1216
    0.2300    0.1226
    0.2400    0.1235
    0.2500    0.1244
    0.2600    0.1254
    0.2700    0.1263
    0.2800    0.1272
    0.2900    0.1281
    0.3000    0.1290
    0.3100    0.1299
    0.3200    0.1308
    0.3300    0.1316
    0.3400    0.1325
    0.3500    0.1333
    0.3600    0.1342
    0.3700    0.1350
    0.3800    0.1358
    0.3900    0.1367
    0.4000    0.1375
    0.4100    0.1382
    0.4200    0.1390
    0.4300    0.1398
    0.4400    0.1405
    0.4500    0.1413
    0.4600    0.1420
    0.4700    0.1427
    0.4800    0.1434
    0.4900    0.1441
    0.5000    0.1448
    0.5100    0.1454
    0.5200    0.1460
    0.5300    0.1467
    0.5400    0.1473
    0.5500    0.1478
    0.5600    0.1484
    0.5700    0.1489
    0.5800    0.1495
    0.5900    0.1500
    0.6000    0.1505
    0.6100    0.1509
    0.6200    0.1514
    0.6300    0.1518
    0.6400    0.1522
    0.6500    0.1526
    0.6600    0.1529
    0.6700    0.1533
    0.6800    0.1536
    0.6900    0.1539
    0.7000    0.1541
    0.7100    0.1544
    0.7200    0.1546
    0.7300    0.1547
    0.7400    0.1549
    0.7500    0.1550
    0.7600    0.1551
    0.7700    0.1552
    0.7800    0.1552
    0.7900    0.1552
    0.8000    0.1552
    0.8100    0.1551
    0.8200    0.1550
    0.8300    0.1549
    0.8400    0.1547
    0.8500    0.1545
    0.8600    0.1543
    0.8700    0.1541
    0.8800    0.1538
    0.8900    0.1534
    0.9000    0.1530
    0.9100    0.1526
    0.9200    0.1522
    0.9300    0.1517
    0.9400    0.1511
    0.9500    0.1506
    0.9600    0.1499
    0.9700    0.1493
    0.9800    0.1486
    0.9900    0.1478
    1.0000    0.1470
    2.6600   -1.2667
    2.6700   -1.2860
    2.6800   -1.3055
    2.6900   -1.3252
    2.7000   -1.3449
    2.7100   -1.3648
    2.7200   -1.3848
    2.7300   -1.4049
    2.7400   -1.4252
    2.7500   -1.4456
    2.7600   -1.4661
    2.7700   -1.4867
    2.7800   -1.5074
    2.7900   -1.5283
    2.8000   -1.5492
    2.8100   -1.5703
    2.8200   -1.5915
    2.8300   -1.6127
    2.8400   -1.6341
    2.8500   -1.6556
    2.8600   -1.6772
    2.8700   -1.6989
    2.8800   -1.7207
    2.8900   -1.7425
    2.9000   -1.7645
    2.9100   -1.7865
    2.9200   -1.8087
    2.9300   -1.8309
    2.9400   -1.8532
    2.9500   -1.8756
    2.9600   -1.8980
    2.9700   -1.9205
    2.9800   -1.9431
    2.9900   -1.9657
    3.0000   -1.9885
```

Finite-Difference Method for Linear BVPs (3)

BVP: $y'' = 2y' - 2y, \;\; y(0) = 0.1, \;\; y(3) = 0.1e^3 \cos(3)$



$y = 0.1e^x \cos x$ – exact solution

FD method

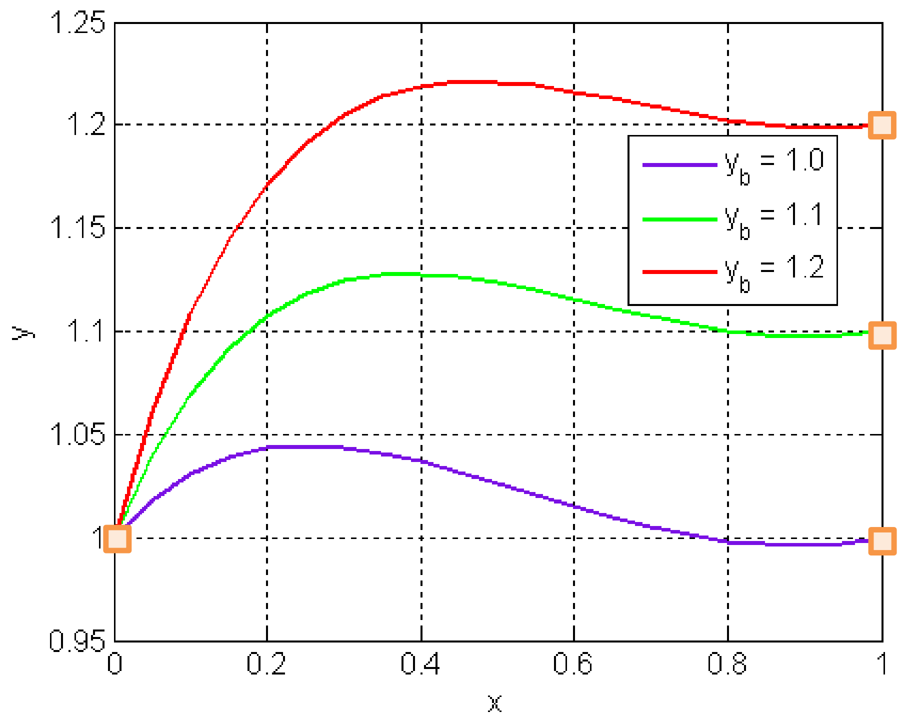
Finite-Difference Method for Linear BVPs (4)

BVP:

- ODE: $\dfrac{d^2y}{dx^2} + 4\dfrac{dy}{dx} + y = 2x^2$
- Dirichlet BCs: $0 \le x \le 1,\ y_a = 1, y_b = \begin{cases} 1.0 \\ 1.1 \\ 1.2 \end{cases}$

Problem Definition in BVP_FD

```
aa = 0; bb = 1; n = 20;  % n is chosen to discretize the interval
p = (-4)*ones(1, n-1); q = (-1)*ones(1, n-1); r = 2*x.^2;
ya = 1; yb = 1.2;  % or 1.1, or 1.0
```



BVPs – Errors

Shooting Method – ***the error is the same as for the method in the respective IVP*** (because this method solves the BVP by reformulating it as a series of IVPs).

FD Method – the error is determined by ***the order of accuracy of the numerical scheme*** used:

- *On the one hand*, the method depends on ***the truncation errors of the approximation formulas used for the derivatives***
- *On the other hand*, different n can be applied to discretize the interval, i.e., there are round-off errors in discretization of the boundary conditions

So, *the accuracy is determined by the* ***larger of the two errors***.

BVPs – Stability

Compared to IVPs: *the instability is associated with error that* ***grows as the solution progress from the initial point*** *to the final one*.

BVPs: *the growth of numerical error with the solution progress is limited by the rightmost BC***.** *Additional factors make stability an important issue*:

1) The ***DE itself may be unstable*** to small perturbations in the BCs.
2) ***Multiple valid solutions*** to the DE may exist for different rightmost BCs.
3) With the FD method, a BVP is converted to a matrix equation, so ***the solution is determined everywhere simultaneously***, thus *the notion of marching forward in time (or marching from left to right) is not relevant*. Stability of solving a BVP by FDs rests on ***stability of the scheme used to solve the resulting set of equations***.

NB: there is *no simple recipe about how do deal with instability*; all elements – DEs, related BCs, solvers of the system of equations, etc. – ***should be investigated and appropriate measures should be taken*** to ensure stability of a numerical solution.

Course Resume

Objectives

I. The primary goal **was** to introduce you to a *wide range of numerical algorithms related to problems in Calculus and Differential Equations, review their fundamental principles, and illustrate their applications.*

II. You **now** should be able **to apply numerical procedures *to solve applied problems*** and, when applying the algorithms to practical scenarios**, control their performance.**

Main Topics

- Interpolation
- Approximation
- Numerical Integration
- Initial Value Problems
- Boundary Value Problems

Project No. 1: *Interpolation*	Project No. 2: *Approximation*	Project No. 3: *Numerical Integration*	Project No. 4: *Initial Value Problem*

Course's MATLAB Procedures

Procedure:	Function:	Source:	Download:
Lagrange_coef	Computation of coefficients of the Lagrange interpolation polynomial	Class: Thur, Oct 30	Lagrange_coef.txt
Lagrange_Eval	Evaluation of the Lagrange polynomial at point $x = t$	Class: Thur, Oct 30	Lagrange_Eval.txt
Newton_coef	Evaluation of the Newton interpolation polynomial	Class: Fri, Oct 31	Newton_coef.txt
Newton_Eval	Evaluation of the Newton polynomial at point $x = t$	Class: Fri, Oct 31	Newton_Eval.txt
[Poly-Graph]	Visualization of the resulting interpolation polynomials	HW [C5]: Mon, Nov 3	-
[Chebyshev-Nodes]	Computation of abscissas of Chebyshev points	HW [C6]: Mon, Nov 3	-
interp1	MATLAB built-in interpolation function	HW [C7]: Mon, Nov 3	-
Spline_test	Script calling MATLAB "spline" function and plotting the result	Class: Thur, Nov 6	Spline_test.txt
FBC_Diff	Numerical differentiation with forward, backward, and central differences	HW [C10]: Fri, Nov 7	-
Lin_LS	Linear least squares approximation	Class: Tue, Nov 11	Lin_LS.txt
Quad_LS	Quadratic least squares approximation	Class: Thur, Nov 13	Quad_LS.txt
Cubic_LS	Cubic least squares approximation	HW [C15]: Thur, Nov 13	-
polyfit	MATLAB built-in approximation function	HW [C17]: Thur, Nov 13	-
[Legendre-Poly]	Visualization of the Legendre polynomials	HW [C18]: Fri, Nov 14	-
[Chebyshev-Poly]	Visualization of the Chebyshev polynomials	HW [C19]: Fri, Nov 14	-
Trap	Composite Trapezoid Rule	Class: Thur, Nov 20	Trap.txt
Simp	Composite Simpson's Rule	HW [C22]: Fri, Nov 21	-
Romb	Romberg integration	Class: Mon, Nov 24	Romb.txt
Gauss_quad	Integration using Gaussian Quadratures	Class: Tue, Nov 26	Gauss_quad.txt
Euler	Solution of IVP using Euler's method	Class: Thur, Dec 4	Euler.txt
Taylor_2	Solution of IVP using the 2nd order Taylor method	HW [C33]: Thur, Dec 4	-
RK2	Solution of IVP using the Runge-Kutta method of order 2	Class: Fri, Dec 5	RK2.txt
RK4	Solution of IVP using the Runge-Kutta method of order 4	Class: Fri, Dec 5	RK4.txt
RK4_sys	Solution of ODE-IVP system using the Runge-Kutta method of order 4	Class: Tue, Dec 9	RK4_sys.txt
fivp_sys	Definition of the right-hand sides in ODE-IVP systems	Class: Tue, Dec 9	fivp_sys.txt
BVP_shooting	Solution of BVP with the shooting method	Class: Tue, Dec 9	BVP_shooting.txt
BVP_shooting_b	Version of BVP_shooting for general boundary conditions at $x = b$	HW [C43]: Fri, Dec 14	-
BVP_FD	Solution of BVP with the finite-difference method	Class: Fri, Dec 14	BVP_FD.txt
Trid_Thomas	Solution of system of linear equations with tridiagoanl matrix	Class: Fri, Dec 14	Trid_Thomas.txt

General Purpose Numerical Software

Computer Algebra Systems

- MATLAB
- Mathematica
- Maple
- MathCAD
- etc.

Libraries of Algorithms/Codes

- International Mathematical and Statistical Libraries (***IMSL***)
- The Numerical Algorithms Group (***NAG***) package
- etc.

Final Exam – Template for the Report

MA3457/CS4033 **B'13**

FINAL EXAM ▼ PART 2

< TYPE YOUR NAME HERE >

1. (20 pts) The set of following data points is given:

```
x = [ 0.2   0.4   0.6   0.8   1.0 ];
y = [-9.6   6.0   10.0  9.5   8.4 ].
```

Determine the 4th order polynomial in the Lagrange form that passes through the points. Use the obtained polynomial to determine the interpolated value for $x = 0.5$.

< Here: present your solution – include MATLAB commands showing how you input the problem, major numerical output, graphs (if any), brief verbal comments, etc. >

2. (20 pts) The following are measurements of the rate coefficient, k, for the reaction $CH_4 + O \rightarrow CH_3 + OH$ at different temperatures T:

```
Temperature (K) = [ 595  623  761  849  989 1076 1146 1202 1382 1445 1562 ]
kx10^20 (m^3/s)  = [ 2.12 3.12 14.4 30.6 80.3 131  186  240  489  604  868 ].
```

Find the "best fit" linear, quadratic, and cubic functions and show their numerical values for all 11 points. Determine the total squared errors for all approximations. Plot them on one graph along with the data points.

< Here: present your solution – include MATLAB commands showing how you input the problem, major numerical output, graphs (if any), brief verbal comments, etc. >