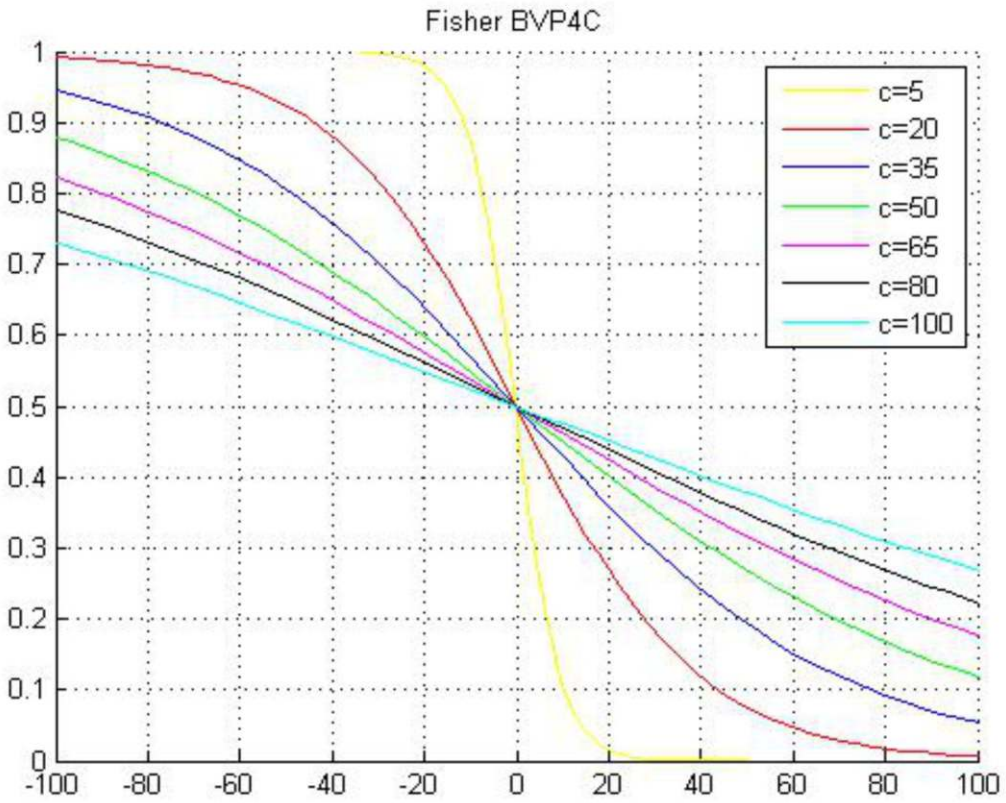
DANIEL N. POP

ASPECTS CONCERNING SOME NUMERICAL METHODS FOR APPROXIMATE SOLUTION OF TWO-POINT BOUNDARY VALUE PROBLEMS



Presa Universitară Clujeană

DANIEL N. POP

ASPECTS CONCERNING SOME NUMERICAL METHODS FOR APPROXIMATE SOLUTION OF TWO-POINT BOUNDARY VALUE PROBLEMS

Referenţi ştiinţifici:

Prof. univ. dr. Damian Trif

Conf. univ. dr. Radu T. Trîmbiţaş

ISBN 978-973-595-878-7

Universitatea Babeş-Bolyai
Presa Universitară Clujeană
Director: Codruţa Săcelean
Str. Hasdeu nr. 51
400371 Cluj-Napoca, România
Tel./fax: (+40)-264-597.401
E-mail: editura@editura.ubbcluj.ro
http://www.editura.ubbcluj.ro/

DANIEL N. POP

ASPECTS CONCERNING SOME NUMERICAL METHODS FOR APPROXIMATE SOLUTION OF TWO-POINT BOUNDARY VALUE PROBLEMS

PRESA UNIVERSITARĂ CLUJEANĂ /
CLUJ UNIVERSITY PRESS
2015

Dedication: To my wife, **Ligia**

Contents

List of Figures

Introduction

The purpose of this work is to determinate the approximate solutions of boundary value problems with conditions inside the interval $(0, 1)$ using collocation method with global *B-spline functions* of degree k (order $k + 1$), orthogonal polynomials *Tchebychev* and combined methods with *B-spline functions* or *C.C method* and *Runge-Kutta* methods. Also using the solver `BVP4C` we compare this methods for boundary values problems with boundary conditions at infinity.

This work is structured in six chapters and an appendix:

- *Chapter 1- Main results,*
- *Chapter 2-Mesh Selection,*
- *Chapter 3- Linear form,*
- *Chapter 4- Nonlinear form,*
- *Chapter 5- Boundary Conditions,*
- *Chapter 6 - MATLAB* [1] *and MAPLE* [2] *Codes.*
- *Appendix A1 -Errors and Floating Point Arithmetic*

In the first Chapter, we present main results on existence and uniqueness of two-point boundary value problems related to linear or nonlinear differential equations, basic issues

[1] MATLAB is a trademark of Mathworks Inc., Natick M.A. United States of America

[2] MAPLE is a trademark Waterloo Maple Inc., Ontario Canada

on polynomial interpolation, Newton's method, and collocation methods. It also present a series of known results in the theory of differential equations to be used as numerical examples studied by us in subsequent chapters.

The purpose of *the second Chapter* is to discuss the practical selection of such a mesh, with the objective of achieving a sufficiently accurate solution as inexpensively as possible.

In the *third Chapter*, we study both problems with oscillatory and non-oscillatory solutions and numerical examples in this chapter are of great technical domains: theory of waves and environmental protection (pollutant transport in rivers), flame propagation in nuclear reactors and reaction diffusion equation . Under certain conditions imposed using arbitrary grid we determined a global approximation method for the oscillatory solutions based on cubic B-spline functions, in which the internal memory used and run time are smaller than if we use a global *C.C. (Tchebychev Collocation)* method. In the case of existence of singularities at the ends of interval $(0, 1)$, we determine an approximate solution using a global method based on *B-spline functions* of degree k and *Runge-Kutta* methods and compare this results with those obtained using the solver `BVP4C.`

We dedicated *Chapter 4* to study the linear and nonlinear problems with unique solution, and to determined the approximate solution using four methods:

1. A global collocation method with B-spline functions of degree k,
2. A collocation method based on combined *B-spline functions* of degree k and *Runge-Kutta* method,
3. A combined method based on collocation methods and *Runge-Kutta* method or *C.C (Tchebychev Collocation)* method,
4. The solver BVP4C.

Chapter 5 is dedicated to treat the BVP problems with Boundary Conditions at Infinity , that model various phenomena in physics, chemistry, biology, medicine.

The development of mathematical software `MATLAB` and `MAPLE` diminished the importance of rounding errors without diminishing the importance of algorithms speed of convergence and that's why in *Chapter 6* we gave codes in `MATLAB` and `MAPLE` for examples used in paper, also we present codes for some problems using the solver `BVP4C.`

In *Appendix A1* we recall some fundamentals results concerning computation errors evaluation.

The writing of this book has benefited enormously from a lot of discussion with prof. dr. Ion Păvăloiu, prof .dr. Ioana Chioreanu from " Babeş-Bolyai " University Cluj-Napoca and prof. dr. Eugen Drăghici from " Lucian Blaga " University Sibiu. We appreciate the help provided by many experts who have commented on portions of this work, prof.dr. Damian Trif , conf. dr. Radu T.Trîmbiţaş from " Babeş-Bolyai " University Cluj-Napoca, and prof.dr. Alexandru I. Lupaş from " Lucian Blaga " University Sibiu have been especially helpful.

I wish to dedicate this book to my wife Ligia Pop.

Daniel N.Pop
Lucian Blaga University of Sibiu
Herman Oberth Computers and Electrical engineering
Department
Bd-ul. Victoriei street nr: 10
Romania
E-mail:popdaniel31@yahoo.com

Chapter 1

Basic Notions and Main Results

1.1 Conditions of existence and uniqueness for two point boundary value problems in compact interval $[a, b]$.

The problem studied in this work has the form (PVPN)

$$\begin{cases} y'' = -f(x, y), & x \in [0, 1] \\ y(a) = \alpha, & \\ y(b) = \beta, \quad a, b \in (0, 1),\ b < a & \end{cases} \tag{1.1}$$

where $f(x, y) \in C([0, 1] \times \mathbb{R})$, $a, b, \alpha, \beta \in \mathbb{R}$.

We also consider the problem (BVPN)

$$\begin{cases} y''(x) + f(x, y) = 0, & x \in [a, b] \\ y(a) = \alpha, & \\ y(b) = \beta. & \end{cases} \tag{1.2a}$$

The problem (BVPN) is equivalent to an integro-differential equation of *Fredholm* type

$$y(x) = \frac{x - a}{b - a}\beta + \frac{b - x}{b - a}\alpha + \int_a^b G(x, s) f(s, \varphi(s)) ds,$$

where G (x, s) is *Green*'s function given by:

$$G(x,s) = \begin{cases} \frac{(s-a)(b-x)}{b-a}, \text{ if } a \leq s \leq x \leq b \\ \frac{(x-a)(b-s)}{b-a}, \text{ if } a \leq x \leq s \leq b \end{cases}.$$

Definition 1 *Let* $[a,b] \subset R$, *finite interval and a function* $f : [a,b] \times R \to R$, f *satisfies a* Lipshitz *condition in variable* y, *if there exist a real constant* $K \in R$ *so that:*

$$\begin{aligned} |f(x,y_1) - f(x,y_2)| &\leq K\ |y_1 - y_2| \\ \forall (x,y_1),(x,y_2) &\in \Omega, \end{aligned} \tag{1.3}$$

and $\Omega \subset [a,b] \times R$ *is a domain.*

Theorem 2 ***(Picard) [51, page 33]****If* $f : [a,b] \times R \to R$ *satisfies the conditions*

1. f *is continuous in a domain* $\Omega \subset [a,b] \times R$,

2. f *satisfies a Lipschitz condition of the form (1.3),*

3. $(b-a) \leq h$, *where* $h > 0$ *is small enough,*

4. *if in addition* h *verifies the relationship:*

$$\frac{1}{2} K \cdot h^2 + 4 \cdot h \leq 1,$$

then there exist the unique solution of the problem (BVPN) on the interval $[a,b]$ *of length* h.

Next we give the following theorem which offers a sufficient condition of uniqueness of the solution of the problem (BVPN).

Theorem 3 ***(R.D Russel, L.F Shampine) [61, pag 10]*** *Suppose that* $y(x)$ *is a solution of the boundary value problem (BVPN), that the functions*

$$f(x,z) \text{ and } \frac{\partial f(x,z)}{\partial y}$$

are defined and continuous for $a \leq x \leq b$, *and* $|z - y| \leq \delta,\ \delta > 0$, *and the homogeneous equation* $y''(x) = 0$ *subject to the homogeneous boundary conditions* $y(a) = y(b) = 0$ *has only the trivial solution. If the linear homogeneous equation*

$$z''(x) + \frac{\partial f(x,y)}{\partial y} z(x) = 0,$$

has only trivial solution, then this is sufficient to guarantee that there exists a $\sigma > 0$, *such* $y(x)$ *is the unique solution of the problem (BVPN) in the sphere*

$$\{w : \|w - y''\| \leq \sigma\}.$$

Also we we recall the following result.

Theorem 4 *[35, pp: 112-113] Suppose that*

$$D = \{a \leq x \leq b, -\infty < y < \infty\},$$

and $f(x, y)$ *is continuous on* D. *If* f *satisfies a* Lipschitz *condition on* D *in the variable* y, *then the initial value problem (IVP)*

$$\begin{cases} y' = z, \\ z' = -f(x,y),\ a \leq x \leq b \\ y(a) = \alpha, \\ y'(a) = \gamma, \end{cases},$$

has a unique solution $y(x)$ *for* $a \leq x \leq b$.

If the problem (BVPN) has the unique solution, the requirement $y(x) \in C^1[0,1]$ ensure the existence and the uniqueness of the solution of the problem (PVPN).

We will consider the cases:

1. f linear, i.e $f(x, y) = q(x)y(x) - r(x)$, where q, $r \in C[0, 1]$.

2. f nonlinear (y' missing) .

Remark 5 *In the following we use the notations: for linear boundary value problem (BVPL), (PVPL) and (BVPN), (PVPN) if f is nonlinear.*

In linear case occurs:

Theorem 6 ***([17, pag 520], [47, pag 286])*** *Let the (BVPL) problem, and in addition:*

1. $q(x)$, $r(x) \in C[a,b]$,
2. $q(x) \geq 0$ *pe* $[a,b]$,

then the (BVPL) problem has unique solution.

Theorem 7 ***[1, pag 461]*** *The (BVPL) problem has unique solution, iff homogeneous problem (with $r = 0,\ \alpha = \beta = 0$) has only trivial solution.*

The question is how we determine an approximate solutions of the problem (PVPL), methods are considered from one of the following classes: discrete methods and collocation methods.

Since in linear case we have studied the oscillatory and non-oscillatory solutions, we recall some notions and basic results consisting in disconjugate criteria.

Definition 8 *[60, pag 79] Let the differential equation*

$$-y''(x) + q(x)y(x) = 0, \tag{1.4}$$

where $q \in C([a,b]$. A solution of differential equation (1.4) is called oscillatory if it has more than two zeros on the interval $[a,b]$.

Remark 9 *According to* Sturm*'s theorem [60, pag 76] that, if the differential equation (1.4) has an oscillatory solution, then all they are oscillatory solutions.*

Theorem 10 ***(Lyapunov)*** *[38, pag 14] The differential equation*

$$-y''(x) + q(x)y(x) = 0,$$

has oscillatory solutions on $[a, b]$, *if*

$$(b-a)\int_a^b |q(x)|\,dx > 4, \tag{1.5}$$

and non-oscillatory solutions, if

$$(b-a)\int_a^b |q(x)|\,dx \leq 4.$$

1.2 Newton's method

The best-known and most popular method for solving nonlinear equations is Newton's method. Let $f(x)$ be a real-valued scalar function and let x^* be a root, i. e.,

$$f(x^*) = 0 \tag{1.6}$$

Having chosen an initial approximation so, we compute :

$$x_{m+1} := x_m - \frac{f(x_m)}{f\,'(x_m)} \quad , \; m = 0, 1, 2, . \tag{1.7}$$

Newton's method arises from linearizing $f(x)$, i. e., replacing $f(x)$ by a linear approximation $l(x)$ consisting of the first two terms in its *Taylor expansion* about x_m, and solving $l(x_m) = 0$. Thus,

$$f(x) \approx l(x) := f(x_m) + f\,'(x_m)(x - x_m)$$

and x_m in (1.7) is the root of $l(x)$.

One can show that if $|x_0 - x^*|$ is small enough and f satisfies some (weak) conditions, this process gives second-order convergence i.e.,

$$|x_{m+1} - x^*| = \mathbb{O}(\,|x_m - x^*|)^2,$$

The same idea can be used to solve a system of equations

$$f(x^*) = 0 \tag{1.8}$$

where

$$f(x) = (f\,_1(x), ..., f_n(x))^T, \; x = (x_1..., x_m)^T.$$

Expanding $f(x)$ around an approximation $x_m$of $x*$ gives the linear expression

$$l(x) := f\,(x^m) + J(x^m)(x - x^m),$$

where

$$J(x) = \left(\frac{\partial f_m}{\partial x_j}\right) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix},$$

is the *Jacobian matrix* of $f(x)$ (also denoted by ∇f).

Thus, given an initial approximation x° , the generalization of (1.7) is

$$J(x^m)(x^{m+1} - x^m) = -f\ (x^m). \tag{1.9}$$

An important factor in assessing an iterative method is how quickly it converges. The *rate of convergence* of a convergent iterative method is the (largest) constant $p \geq 1$ such that for some constant $c > 0$

$$\left|x^{m+1=} - x^*\right| \leq c \cdot |x^m - x^*|^p . \tag{1.10}$$

If $p = 1$ then the convergence is said to be *linear* (and we must require $c < 1$ for convergence) and if $p = 2$, it is *quadratic*. If p > 1 then we can fairly safely use the approximation:

$$|x^* - x^m| = \left|x^{m+1} - x^m\right| =: e_{m+1} \tag{1.11}$$

and a practical estimate of the rate is given by :

$$p \approx \frac{\ln(\frac{e_{m+1}}{e_m})}{\ln(\frac{e_m}{e_{m-1}})}. \tag{1.12}$$

For *Newton's method,* as in the scalar case, one can show that if $|x^* - x^0$ is small enough and f satisfies suitable conditions, there is quadratic convergence (see the next Theorem).

Theorem 11 *[41] (*Newton-Kantorovich) *Assume that f(s) is continuously differentiable on a sphere $D = S(x^\circ, r)$ and that*

$$\begin{aligned} \left|J(x^0)^{-1} f\ (x^0)\right| &\leq \alpha, \\ \left\|J(x^0)^{-1}\right\| &\leq \beta, \\ \|J(x) - J(y)\| &\leq \gamma\,|x - y| \quad x, y \in D, \end{aligned} \tag{1.13}$$

hold. If $\sigma < \frac{1}{2}$ *and* $r \geq r_-$ *then*

(*i*) *There exists a unique root* x^* *in* $\overline{S}$ (x°,r_-),

(*ii*) *The Newton sequence of iterates starting from* s° *converges to* s^*.

If $\sigma < \frac{1}{2}$ *then*

(*iii*) s^* *is the unique root of* $f(x)$ *in* $S(x^\circ, \min(r, r+))$.

(*iv*)

$$|x^m - x^*| \leq (2\sigma)^{2^m} \frac{\alpha}{\sigma}, \ m = 0, 1, ...$$

Quadratic convergence, when present, makes for a very rapid process in an actual computation, because the number of accurate digits roughly doubles per iteration step.

Thus, *Newton's method* converges very rapidly when s° is sufficiently close to x^*.

It should be realized, though, that this is a local property. *Newton's method* may behave poorly when $|x^\circ - x^*|$ is not small or c is large. In fact, it may not converge at all.

Remark 12 *Further generalizations of these results are possible. For example, if the hypotheses of the theorem are satisfied except that errors are made in solving (1.9) (e.g., because a numerical method is used to approximate the solution, or simply because of roundoff errors), and if these errors are sufficiently small, then the conclusions of the theorem still hold, with the bounds suitably perturbed.*

1.2.1 Quasi-Newton methods

There are two main drawbacks to *Newton's method*. The first is the fact that the method is guaranteed to converge only if the initial guess is already "sufficiently close" to the solution (i.e., a local convergence result).

The second drawback is the expense of the method: each iteration (1.9) requires the evaluation of the *Jacobian* $J(x^m)$ and the right hand side $f(x^m)$ and the factorization of $J(x^m)$. In many instances the first partial derivatives needed for $J(x^m)$ may be unavailable. Even if they are, when $J(x^m)$ is a dense matrix, each iteration requires roughly n^2+n scalar

function evaluations and $\mathbb{O}(n^3)$ arithmetic operations to solve the linear system (1.9). This provides motivation to find cheaper ways to perform the iteration.

Quasi-Newton methods approximate $J(x^{m+1})$ by modifying $J(x^m)$ in a simple way that allows iterates to be computed cheaply. Probably the simplest way to do this is to hold the Jacobian fixed, e.g., to use $J(x^m)$ as the approximation to $J(x^{m+1})$ when calculating x^{m+2}, and repeat this process for several iterations. Another way is to use so called *Broyden updates*, whereby an approximation $\widehat{J}_{m+l}$ to $J(x^{m+1})$ and a factorization of $\widehat{J}_{m+l}$ are obtained from the previous approximate *Jacobian* $\widehat{J}_m$ by performing only $\mathbb{O}(n^2)$ arithmetic operations and evaluating only $f(x^m)$ and $f(x^{m+1})$.

Specifically, if

$$w^m = f(x^{m+1}) - f(x^m)$$

then with $\Delta x^m := x^{m+1} - x^m$

$$\widehat{J}_{m+1} = \widehat{J}_m + \frac{1}{|\Delta x^m|_2^2} \Delta x^m (w^m - \widehat{J}_m \Delta x^m)^T \tag{1.14}$$

Given, for example, a *QU factorization* of $\widehat{J}_m$ it is possible to find the corresponding factorization of $\widehat{J}_{m+1}$ in $\mathbb{O}(n^2)$ arithmetic operations before finding x^{m+2}.

Alternatively, if

$$H_m = \widehat{J}_m^{-1}, \; H_{m+1} = \widehat{J}_{m+1}^{-1},$$

then this inverse can be directly calculated from

$$H_{m+1} = H_m + \frac{(H_m \Delta x^m)(\Delta x^m - H_m w^m)^T}{(\Delta x^m)^T H_m w^m}. \tag{1.15}$$

With this approach, *Broyden's* method can thus be implemented in the form

$$x^{m+1} = x^m - H_m f(x^m) \tag{1.16}$$

This requires only n scalar function evaluations and $\mathbb{O}(n^2)$ arithmetic operations per iteration although (1.14) has preferable stability properties (which in turn are inferior to those of *Newton's method*). The *Broyden update* is sometimes called `rank-1` update. Indeed, the matrix $J_{m+1} - J_m$ is of rank 1.

A price paid for using *quasi-Newton methods* such as (1.14) is that the convergence rate drops from quadratic to superlinear; i. e.,

$$|x^{m+1} - x^*| \ \leq \ \gamma_m |x^m - x^*| \tag{1.17}$$

where $\lim_{m\to\infty} \ \gamma_m = 0$. A property of superlinear convergent iterates is that

$$\lim_{m\to\infty} \frac{|x^{m+1} - x^m|}{|x^m - x^*|} = 1$$

so it is reasonable to use distance between successive iterates as a measure of error, as in (1.11). Many quasi-Newton methods such as (1.16) suffer from the more serious disadvantage that the updates do not preserve matrix sparsity, making them impractical in many applications (including for most methods for solving BVPs).

1.2.2 Quasilinearization

While *Newton's method* has so far been considered only for solving a system of nonlinear equations (1.8) (with a solution $s* \in R^n$), it is also possible to apply it in a direct way to a nonlinear operator equation, which in our case is a nonlinear BVP.Historically, this has been called *quasilinearization*, a name which of course refers to the fact that a nonlinear equation is in some way linearized. We also use this name, since it allows us to conveniently distinguish it from the case where *Newton's method* is applied to a discrete system of nonlinear equations.

To see how quasilinearization is done, it is desirable to express BVPs in operator notation. We define the nonlinear differential operator

$$Ny(x) := y'(x) - f(x, y(x)), \ a < x < b, \tag{1.18}$$

and the boundary conditions

$$Gy := g(y(a), y(b)), \tag{1.19}$$

and consider the BVP:

$$\begin{aligned} Ny &= 0,\ a < x < b && \text{(1.20a)} \\ Gy &= 0 && \text{(1.20b)} \end{aligned}$$

The linearization about a known function $y^m(x)$ now has the form

$$\begin{aligned} Ny(x) &= Ny^m(x) + N\,'(y^m)(y(x) - y^m(x)),\ a < x < b \\ Gy &= Gy^m + G\,'(y^m)(y - y^m). \end{aligned}$$

Setting the right-hand sides to 0 and solving for y^{m+1}, we obtain

$$N\,'(y^m)(y^{m+1}(x) - y^m(x)) = -Ny^m(x),\ a < x < b \tag{1.21}$$

$$G\,'(y^m)(y^{m+1} - y^m) = -Gy^m. \tag{1.22}$$

The operators $N\,'(y)$ and $G\,'(y)$ are called *Frechet derivatives* of Ny and Gy respectively, but it is not our intention to discuss the functional analysis explaining what properties they have. Instead, we just explain how one does the quasilinearization in practice. If D denotes differentiation, i.e., $Dy := y\prime(x)$, then formally differentiating (1.19) with respect to y gives a differential operator $N\,'(y)$ which depends on the function y where the linearization takes place and operates on any function $w \in C^{(1)}[a, b]$

$$N\,'(y)w(x) = [D - \frac{\partial f}{\partial y}(x, y(x))]w(x). \tag{1.23}$$

Also, differentiating (1.20a) with respect to $y(a)$ and $y(b)$, we have :

$$G\,'(y)w = \frac{\partial g(y(a), y(b))}{\partial y(a)}w(a) + \frac{\partial g(y(a), y(b))}{\partial y(b)}w(b). \tag{1.24}$$

Substituting (1.23,1.24) into (1.22,1.21), we get the linear BVP for $w(x)$

$$w\,'(x) - A(x)w(x) = -(Dy^m(x) - f(x, y^m(x))),\ a < x < b \tag{1.25}$$

$$B_a w(a) + B_b w(b) = -g(y^m(a), y^m(b)), \tag{1.26}$$

where

$$A(x) = \frac{\partial f}{\partial y}(x, y^m(x)), \tag{1.27}$$

$$B_a = \frac{\partial g}{\partial y(a)}(y^m(a), y^m(b)),$$

$$B_b = \frac{\partial g}{\partial y(b)}(y^m(a), y^m(b)).$$

Thus (1.25,1.26) and

$$y^{m+1}(x) = y^m(x) + w(x) \tag{1.28}$$

comprise one step of *Newton's method* applied to the nonlinear BVP (1.20a, 1.20b). We see that this quasilinearization amounts to solving a nonlinear BVP by an iterative method where a sequence of linear BVPs (1.25), (1.28) is solved, $m = 0, 1, ...$ with $y^\circ(x)$ a given initial solution guess.

In fact, we can rewrite (1.25) as

$$L[y^m]w = -Ny^m, \tag{1.29}$$

$$B[y^m]w = -Gy^m, \tag{1.30}$$

where we define the linear operators like

$$L[y^m](\cdot) := [D - A(x)](\cdot) \tag{1.31}$$

$$B[y^m](\cdot) := B_a(\cdot)\ _{x=a} + B_b(\cdot)\ _{x=b} \tag{1.32}$$

using (1.27).

The *Newton-Kantorovich Theorem* can be used to guarantee convergence of the *quasi-linearization method* under suitable assumptions. In this new context (1.13) holds for y° near y if the operator $[D - A(x)]$ is smooth and invertible at $y(x)$. This translates into invertibility of the variational problem

$$L[y]w = 0,\ B[y]w = 0 \tag{1.33}$$

i. e, one requires that $y(x)$ be isolated or that the unique solution to (1.33) is $w = 0$. The bound in (2.64b) reflects the effect that perturbations in the right-hand-sides of the two equations (1.13) have upon the solution. It turns out that the sensitivity of the nonlinear BVP near an isolated solution and the convergence properties of quasilinearization are directly related to the conditioning of the variational problem for the desired solution.

Given the nonlinear BVP

$$Nu(x) = u^{(n)}(x) - f(x, u, u\,', ..., u^{(n-1)}) = 0,\ a < x < b \tag{1.34}$$

$$g(u(a), ..., u^{(n-1)}(a), u(b), ..., u^{(n-1)}(b)) = 0, \tag{1.35}$$

we can write

$$y(x) = (y_1(x), ..., y_n(x)) := (u(x), ..., u^{(n-1)}(x)).$$

Corresponding to the steps in (1.23) through 1.35), one can show that quasilinearization takes the form

$$\begin{aligned} L[u^m]w(x) &= -Nu^m(x),\ a < x < b \\ B_a z(a) + B_b z(b) &= -g(y^m(a), y^m(b)), \\ y^{m+1}(x) &= y^m(x) + w(x), \end{aligned}$$

where

$$L[u]w(x) = w^{(n)}(x) - \sum_{i\ =1}^{n} \frac{\partial f(x, y)}{\partial y_1} w^{(i-1)}(x)$$

$$B_a := \frac{\partial g(u,v)}{\partial u}, B_b := \frac{\partial g(u,v)}{\partial v}$$

at

$$u = y^m(a),\ v = y^m(b),$$

and

$$z(x) := (w(x), ..., w^{(n-1)}(x)).$$

1.3 Polynomial Interpolation

1.3.1 Lagrange polynomials

Given a set of n points $x_1, x_2, ...x_n$ and function values $f(x_1), ...f(x_n)$, the polynomial interpolation problem is that of finding a polynomial $p(x)$ which satisfies

$$p(x_i) = f(x_i), i = 1, 2, ...n. \tag{1.36}$$

The points $[x_i]_{i=1}^n$are called *interpolation points* and $p(x)$ is said to interpolate $f(x)$ at these points. Interpolation polynomials are frequently used in data fitting; moreover, they play a central role in the development and analysis of numerical methods for solving differential equations.

Assume for now that the interpolation points $[x_i]_{i=1}^n$are distinct. Then there is a unique polynomial $p(x)$ of order n (degree $< n$) satisfying 1.36). A simple constructive proof that $p(x)$ exists involves writing $p(x)$ in its Lagrange form

$$p(x) = \sum_{i=1}^{n} f(x_i)L(x_i), \tag{1.37}$$

$$L(x_i) = \prod_{j=1, i\neq j}^{n} \frac{(x - x_j)}{(x_i - x_j)} \; 1 \leq i \leq n. \tag{1.38}$$

Since each $L_i(x)$ is a polynomial of degree $n-1$ which satisfies

$$L_i(x_j) = \delta_{ij}$$

the function $p\{x)$ is a polynomial of order n which satisfies (1.36). Uniqueness of the interpolation polynomial readily follows, since if $q(x)$ were another polynomial of order n (denoted $q \in P_n$) satisfying (1.36),$r(x) = p(x) - q(x)$ would be a polynomial of order n with zeros at $x_1, ...x_n$. This implies $r(x) = 0$.

Another obvious way to compute $p(x)$ is to write

$$p(x) = \sum_{j=1}^{n} a_j x^{j-1}$$

and to determine the unknown coefficients by satisfying (1.36). This results in the system of equations:

$$Va = f \tag{1.39}$$

where

$$V = (x_i^{j-1})_{i\ ,j\ =1}^{n},\ a = (a_1, ...a_n)^T\ ,\ f = (f\ (x_1), ...f(x_n))^T, \tag{1.40}$$

the matrix V is called is called the *Vandermonde matrix* and the i^{th} column of its inverse can be shown to consist of the coefficients of $l_i\ (x)$ in (1.38).

The Lagrange form (1.38) is convenient from a theoretical standpoint, but in practice it is expensive to evaluate and cumbersome to use when interpolation at additional points is desired. The Vandermonde system (1.38) can be ill-conditioned. This motivates developing a computationally more convenient form for $p(x)$.

Assume that the n distinct points $x_1, x_2, ...x_n$ lie in an interval $[a, b]$ and that the values of $f(x)$ at these n points are given. Furthermore, assume that $f(x)$ is sufficiently smooth, so whenever $f^{\ (i)}\ (x)$ is used we are assuming that $f\ (x) \in C^{(i)}[a, b]$.

We define divided differences of $f(x)$ at the points $x_1, x_2, ...x_n$ as follows:

The 0^{th} divided difference $f\ (x)$ at x_i is

$$f[x_i] := f\ (x_i),\ 1 \leq i \leq n$$

The k^{th} divided difference $f\ [x_i, x_{i+1}, ...x_{i+k}]$ at $x_i, x_{i+1}, ...x_{i+k}$ is recursively defined as [1]

$$f\ [x_i, x_{i+1}, ...x_{i+k}] := \frac{f\ [x_{i+1}, x_{i+2}, ...x_{i+k}] - f\ [x_i, x_{i+1}, ...x_{i+k-1}]}{x_{i+k} - x_i},\ 1 \leq i \leq n \tag{1.41}$$

A further useful property of divided differences is that

$$f\ [x_i, x_{i+1}, ...x_{i+k}] = \frac{f^{\ (k)}(\xi)}{k!}, \tag{1.42}$$

for some point ξ in $[a, b]$. After constructing a table of divided differences for $f\ (x)$ at the interpolation points, we express the interpolating polynomial directly in its *Newton*

form :

$$p(x) = \sum_{i=1}^{n} f\ [x_1, x_2, ...x_i] \prod_{j=1}^{i-1} (x - x_j).$$

The *Newton form* for $p(x)$ overcomes the two disadvantages of the *Lagrange form*. To evaluate $p(x)$ at a point $x = z$ nested multiplication (*Horner's algorithm*) can be used. The following algorithm produces $a_i = p(z)$.

Evaluating polynomial in *Newton's form*

$$a_{n+i} = 0$$

FOR $j = n, , ...1$ DO

$$a_j := f\ [x_1, ...x_j] + (z - x_j)a_{j+1}$$

If the polynomial of order $n+1$ interpolating $f(x)$ at $x_1, x_2, ...x_{n+1}$ is needed, we only have to extend the divided difference table to give $f\ [x_1, x_2, ...x_{n+1}]$ since

$$p(x) + f\ [x_1, x_2, ...x_{n+1}] \prod_{j=1}^{i-1} (x - x_j), \tag{1.43}$$

is desired polynomial.

The form of the error arising from interpolation is given in the following theorem.

Theorem 13 *[6, pag: 58]The unique polynomial $p(x)$ of order n interpolating $f(x)$ at $x_1, x_2, ...x_n$ has, for any $\overline{x} \in [a, b]$, the error term :*

$$f(\overline{x}) - p(\overline{x}) = \frac{f^{\ (n)}(\xi_{\overline{x}})}{n!} \prod_{j=1}^{n} (\overline{x} - x_j) \tag{1.44}$$

for some $\xi_{\overline{x}} \in [a, b]$.

If $f^{\ (n)}(x)$ is bounded by M_n on $[a, b]$, then from (1.44) the interpolation error is bounded by $M_n\ C_n\ h^n$, where $h = b - a$ and C_n is a constant which depends on n. The rate at which C_n actually grows with n depends upon how the interpolation points are selected. For example, for equally spaced points C_n grows like $e^{n/2}$, while for the so called *Chebychev* points C_n only grows linearly with n.

1.3.2 Osculatory interpolation

Frequently we wat to interpolate derivative values of $f(x)$ in addition to function values. This is called `osculatory interpolation.`It can be done simply when the `Newton form` of the interpolating polynomial is used. The definition of k^{th} divided difference of $f(x)$ is extended to the case $x_1 = x_2 = ... = x_{i+k}$ by defining

$$f\ [x_1, x_2, ...x_{i+k}] := \frac{f^{\ (k)}\ (x_i)}{k!} \tag{1.45}$$

The basic properties of divided differences can be shown to still hold. Note that if $x_1 = x_2 = ... = x_n$ the `Newton form (1.43)`is just the first terms of the `Taylor expansion` about x_i

$$p(x) = \sum_{i=0}^{n-1} \frac{f^{\ (i)}(x_i)}{i!}(x - x_l)^i,$$

wich is, of course, a polynomial matching the first n derivatives of $f\ (x)$ at x.

Theorem 14 *Given points $x_1, x_2, ...x_n$ in $[a, b]$ the polynomial*

$$p(x) = \sum_{i=1}^{n} f\ [x_1, x_2, ...x_i] \prod_{j=1}^{i-1}(x - x_j),$$

interpolates $f\ (x)$ in the following sense :

If the point z occurs l times among the points $\{x_i\}_{j=1}^n$ then

$$p^{(j)}(z) = f^{\ (j)}(z),\ j = 0, 1, ..., i - 1.$$

1.3.3 Piecewise polynomials

In this subsection we define a `spline` as any piecewise polynomial function.

Definition 15 *Given a partition $\overline{\Delta}$ of some interval $[a, b]$ as*

$$\overline{\Delta} := \{a = x_0 < x_1 < ... < x_n = b\}, \tag{1.46}$$

$\mathbb{P}_{k,\overline{\Delta}}$ is defined as a family of functions which on each subinterval of $\overline{\Delta}$ are polynomials of order k. Thus $s(x) \in \mathbb{P}_{k,\overline{\Delta}}$ if

$$s(x) := a_{i,0} + a_{i,1}(x - x_1) + ... + a_{i,k-1}(x - x_i)^{k-1} \tag{1.47}$$

on $[x_i, x_{i+1}]$ *for some constants* $\{a_{ij}\}_{j=0}^{k-1}, 1 \leq i \leq N$. *At a mesh point* x_i, *when* $s(x)$ *is not continuous, we shall arbitrary define* $s(x)$ *to be right continuous by settings:*

$$\begin{aligned} s(x_i) &= P_i(x_i) \\ s(b) &= P_N\ (b) \end{aligned}$$

The representation (1.47) is called the ***local monomial representation(splines)*** *for* $s(x)$.

Definition 16 *The subfamily of* $\mathbb{P}_{k,\overline{\Delta}}$ *consisting of splines* $s(x) \in C^{(m-1)}[a,b]$ *we denote by* $\mathbb{P}_{k,\overline{\Delta},m}$.

It may be desirable to construct a basis for $\mathbb{P}_{k,\overline{\Delta},m}$,i.e a convenient set of splines which span $\mathbb{P}_{k,\overline{\Delta},m}$. For special case $m \leq \frac{k}{2}$, `Hermite-type bases` are one possibility [6, pp: 60-61].

In this subsection we show briefly how to construct a more general alternative basis the `B-splines basis` which has historically been the most used general basis of $\mathbb{P}_{k,\overline{\Delta},m}$.

Definition 17 *The* ***truncated power*** *function* $(t)_+^r$ $(r > 0) \in C^{(r-1)}(-\infty, \infty)$ *is the function defined by*

$$(t)_+^r := (\max[t,0])^r = \begin{cases} t^r\ ,\ if\quad t > 0 \\ 0,\ if\, t \leq 0 \end{cases} . \tag{1.48}$$

Let $\{t_j\}_{j\ =1}^{J+k}$, $J := (N-1)(k-m)+k$, be a sequence of nondecreasing points such that

$$t_1 \leq t_2 \leq ... \leq t_k = x_1; x_2, x_3, ...x_N$$

are each repeated $(k-m)$ times and

$$x_{N+1} = t_{j+1} \leq t_{j+2} \leq ... \leq t_{j+k}$$

For $1 \leq j \leq J$ define $M_j(x)$ by

$$M_j(x) := g_s[t_j, ...t_{j+k}]$$

where

$$g_s(t) := (x-t)_+^{k-1}.$$

The `B-spline of order k` on $t_j, ...t_{j+k}$ is

$$B_{j,k}(x) := (t_{j+k} - t_j)(-1)^k M_j(x), 1 \leq j \leq J. \tag{1.49}$$

The set $\{B_{j,k}(x)\}_{j=1}^J$ forms a basis for $\mathbb{P}_{k,\overline{\Delta},m}$.This basis has a number of attractive features. For example, $B_{j,k}(x)$ is positive on (t_j, t_{j+k}) and zero elsewhere; i.e., $B_{j,k}(x)$ has local support (t_j, t_{j+k}), so computations using a `B-spline` basis generally lead to linear systems of equations with banded matrices. Evaluation of k th-order `B -splines` and their derivatives can be done by generating lower order ones first, utilizing a relation expressing p th-order B -splines in terms of $(p-l)$st-order ones. Specifically, the following algorithm can be used to evaluate the k nonzero it th-order B -splines $B_{j,k}(x)$ at

$x \in (t_j, t_{j+1})$.

Algorithm for B-spline evaluation

$B_{j,1}(x) = 1$

FOR $l = 1, ..., k-l$ DO

$B_{j-l,1+1}(x) = 0$

FOR $i = 1, ..., l$ DO

$M_{j+i-l,l}(x) = B_{j+i-l,l}/(t_{j+l} - t_{j+i-l})$

$B_{j+i-l-1,l+1}(x) = B_{j+i-l-1,l+1}(x) + (t_{j+l} - x)M_{j+i-l}(x)$

$B_{j+i-l-1,l+1}(x) = (x - t_{j+l-1})M_{j+i-l,l}(x)$

If $s(x) = \sum_{r=1}^{J} a_{r,i+1}B_{r,k}(x)$, then its derivates can be found for $x \in (t_j, t_{j+k})$ from

$$s^{(j)}(x) = \sum_{i=j-k+i-1}^{J} a_{l,i+1}B_{l,k-i}(x), \tag{1.50}$$

where

$$a_{l,i+1} := \begin{cases} a_l \text{ , if } i = 0 \\ (k-i)\frac{a_{l,j} - a_{l-1,j}}{t_{k+j-l} - t_l} \text{ , if } i > 0 \end{cases}.$$

Spline interpolation

Consider the problem of interpolating a function $f\ (x)$ at $J = (N - \backslash)(k - m) + k$ points $z_1 < z_2 <...< z_j$ in $[a, b]$ with a spline function $s(x) \in \mathbb{P}_{k,\overline{\Delta},m}$. Given a B -spline basis for $\mathbb{P}_{k,\overline{\Delta},m}$ this problem is equivalent to solving the linear system of equations

$$s(z_j) = \sum_{i\ =1}^{J} B_{i,k}(z_j)a_i = f\ (z_j),\ 1 \leq j \leq J \tag{1.51}$$

for the unknown coefficients $\{a_i\}_{i=1}^{J}$ It can be shown that the matrix $\{B_{i,k}(z_j)\}_{i,j=1}^{J}$ is nonsingular iff $B_{i,k}(z_j) \neq 0$ for $1 < j < J$. From the local support property of B-splines, the matrix has bandwidth at most $2k - 1$ whenever the interpolation problem has a unique solution. In fact, the matrix is totally positive (i. e., all minors have strictly positive determinants), which implies that Gaussian elimination without pivoting can be used to solve (1.51). Note that using a local representation like (1.47) instead to find $s(x)$ would require forming a system of equations involving the J interpolation conditions and the

$(N - l)m$ continuity conditions at $\{x_i\}_{i=2}^{N}$.

The form of the error for spline approximation is what one might expect from the comments following.

Theorem 18 *[6, pag 62] If* $f(x) \in C^{(k)}[a, b]$ *then there is a spline* $\widehat{s}(x) \in \mathbb{P}_{k,\overline{\Delta},m}$ *satisfying*

$$\max_{a\leq\ x\leq\ b} |f(x) - \widehat{s}(x)| := \|f - \widehat{s}\| \leq C_k h^k \left\|f^{(k)}\right\|, \tag{1.52}$$

for some constants C_k *independent of* $f(x)$, $h = \max_{1\leq\ i\ \leq\ N}(x_{i+1} - x_i)$.

Corollary 19 *Under conditions , a spline approximation is determined locally e.g each B-spline coefficients* a_i *only depends upon* $f(x)$ *on* $[t_{i+1-k}, t_{i+k}]$ *and (1.52) can be strengthened to*

$$\max_{t_i\ \leq\ x\leq\ t\ _{i+1}} |f(x) - \widehat{s}(x)| := \|f - \widehat{s}\|_{[t_i,t_{i+1}]} \leq \widehat{C_k}\widehat{h}^k \left\|f^{(k)}\right\|_{[t_i,t_{i+1}]}$$

where $\widehat{h}_i = \max_{1\leq\ i\ \leq\ N}\ (t_{i+1+k} - t_{i+k})$ *and* $\widehat{C_k}$ *is a constant independent of* $\overline{\Delta}$ *and* $f(x)$.

This form for error motivates selection of a mesh which `equidistributing` or `distributes equally` the local errors $\left\|f^{(k)}\right\|_{[t_i,t_{i+1}]}$.

In fact if

$$\overline{\Delta}^* : a = \tau_1 < \tau_2 < ... < \tau_{N+1} = b,$$

is chosen such that

$$\int_{\tau_j}^{\tau_{j+1}} \left|f^{(k)}(x)\right|^{1/k} dx = \frac{1}{N}\int_a^b \left|f^{(k)}(x)\right|^{1/k} dx,\ 1 \le j \le N$$

then there exist a function $s^*(x) \in \mathbb{P}_{k,\overline{\Delta},m}$ satisfying

$$\|f - s^*\| \le \frac{C^k}{N^k}[\int_a^b \left|f^{(k)}(x)\right|^{1/k} dx]^k.$$

1.4 Numerical methods with nonuniform mesh

1.4.1 Discrete methods

The discrete methods are one step methods and multistep methods. Since in following chapters we use *Runge-Kutta* methods for initial value problem `(IVP)`, we recall the following results.

One step *Runge-Kutta* scheme

A discrete numerical one step approximation solution $\{y_i : i = 0, 1, ...n\}$ can be calculated on nonuniform mesh Δ of the interval $[a, b]$

$$\overline{\Delta} := \{a = x_0 < x_1 < ... < x_n = b\}, \tag{1.53}$$

defined so that

$$y_i \approx y(x_i),$$

where $y(x) \in C^1[a, b]$ is unique solution of (BVPN) problem. We denote the stepsize by

$$h_i = x_{i+1} - x_i \ ; \ i = 0, 1, 2, ..., n-1. \tag{1.54}$$

If

$$h_0 = ... = h_{n-1} = \frac{(b-a)}{n},$$

we have uniform mesh.

In the case of nonuniform mesh we denote by:

$$h := \max |x_{i+1} - x_i| \ , \ i = 0, 1, 2, ..., n-1, \tag{1.55}$$

$$\underline{h} := \min |x_{i+1} - x_i| \ , \ i = 0, 1, 2, ..., n-1. \tag{1.56}$$

We construct the following collocation points like in ([64], [65])

$$\xi_{ij} = x_i + h_i \rho_j; \ 1 \leq j \leq k, \ 0 \leq i \leq n-1, \tag{1.57}$$

on each subinterval $[x_i, x_{i+1}],\ i = 0, 1, ..., N-1$, where

$$0 < \rho_1 < \rho_2 < ... < \rho_k < 1, \tag{1.58}$$

are the roots of k-th *Legendre* polynomial (see [5] for more details).

Definition 20 *[6, pag 69] The general form of a $q-$stage* Runge-Kutta *method is*

$$\begin{cases} y_0(a) = \alpha, \\ y_{i+1} = y_i + h_i \sum\limits_{j=1}^{k} \beta_j f_{ij};\ 0 \le i \le n-1 \end{cases}, \tag{1.59}$$

where

$$f_{ij} = f(\xi_{ij}, y_i + h_i \sum_{l=1}^{k} \alpha_{jl} f_{il});\ 1 \le j \le k.$$

The *Runge-Kutta* method (1.59) is called **explicit** if

$$\sum_{l=1}^{k} \beta_l = 1;\ \sum_{l=1}^{k} \alpha_{jl} = \rho_j;\ 1 \le j \le k, \tag{1.60}$$

where ρ_j are given by (1.58), it is called **implicit** otherwise. **Explicit** schemes have the advantage that the computation of new approximation y_{i+1} is straightforward. For **implicit** schemes we need to solve a nonlinear equation (if f is nonlinear). If :

$$\left| h \frac{\partial f}{\partial y} \right| \|A\| < 1,$$

where the matrix A is define as [6, page: 113], then this can be done by a simple fixed-point iteration and otherwise by the more expensive *Newton* method. Note that an implicit scheme always requires to predict a starting value for the iteration.

Theorem 21 *[6, pag 220] If $f(x, y) \in C^{(p+1)}([a, b] \times \Omega),\ \Omega \subset \mathbb{R}$ a* **Runge-Kutta** *scheme it is accurate of order p for a (BVPN) problem. Hence*

$$|y_i - y(x_i)| = \mathbb{O}(h^p),\ 0 \le i \le n-1. \tag{1.61}$$

Also at collocation points

$$\xi_{ij},\ i = 0, 1, ...n,\ j = 1, 2, ..., k,$$

it holds

$$\left|y_{ij} - y(\xi_{ij})\right| = \mathbb{O}(h^p) + \mathbb{O}(h_i^{k+1}); 1 \leq j \leq k,\ 0 \leq i \leq n-1,$$

where $h_i = x_{i+1} - x_i,\ h = \max\limits_{0 \leq\ i \leq\ n-1} \{h_i\}$.

1.4.2 Collocation methods

Let Δ be a arbitrary mesh of the interval $[a, b]$ given by (1.53) and we construct the collocation points like in (1.57).

One renumbers the collocation points, such that the first is $\xi_0 := a,\ \xi_1 := x_0 + h_0\rho_0$, and the last $\xi_{N-1} := x_{n-1} + h_{n-1}\rho_k,\ \xi_N = b$. So the grid Δ becomes

$$\overline{\Delta} := \{a = \xi_0 < \xi_1 < ... < \xi_N = b\}, \tag{1.62}$$

where $N = nk + 2$. We denote by $P_{k,\overline{\Delta},s}$, the set of functions $s(x) \in C^2[a, b]$ which have the following proper ties for any $i = 0, 1, 2, ..., N-1$

1. $s(x) = s_i(x),\ x \in [\xi_i, \xi_{i+1}]$,
2. $s_i(x)$ are polynomial of degree k.

We wish to find an approximate solution of the (BVPN) problem, having the following form

$$u_{\overline{\Delta}}(x) = \sum_{j=0}^{N} c_j \phi_j(x), \tag{1.63}$$

where $\phi_j(x) \in \mathbb{P}_{k,\overline{\Delta},s},\ j = 0, 1, 2, ...N$ are linearly independent functions, and c_j are real parameters. These parameters are determined so that $u_{\overline{\Delta}}(x)$ satisfy the differential equation (1.2a) in $(N-1)$ points

$$\xi_i \in \overline{\Delta},\ i = 1, 2, ..., N-1,$$

and satisfies the boundary conditions

$$u_{\overline{\Delta}}(a) = \alpha,\ u_{\overline{\Delta}}(b) = \beta.$$

We impose the condition as for every $j = 0, 1, 2, ...n,\ \phi_j(x)$ and any linear combination of these are $C^1[a, b]$ class (like the exact solution of (BVPN) problem).

In this paper we use for $\{\phi_j(x),\ j = 0, 1, 2, ...N\ \}$ the following functions:

1. B-splines of order $k + 1$ (degree k),

2. *Chebychev* orthogonal polynomials of first kind.

B-splines of order k+1 (degree k)

For reason of efficiency, stability, flexibility in order and continuity, we choose B-splines as basis functions. Efficient algorithms for calculating with are given by *deBoor (*[14], [15]) and *Riesler* [59, page 18]. Let the grid $\overline{\Delta}$ given by (1.62) for $x \in [a, b]$ and $0 \leq i \leq N - k - 1$, we define **B-splines functions of order k+1 (degree k)** so:

$$\begin{cases} B_{i,0}(x) = \begin{cases} 1 & \text{if } \ \xi_i \leq x < \xi_{i+1},\ i = 0, 1, ..., N-1 \\ 0 & \text{otherwise} \end{cases} \\ B_{i,k}(x) = \frac{x-\xi_{\ i}}{\xi_{i+k}-\xi_i} B_{i,k-1}(x) + \frac{\xi_{i+k+1}-x}{\xi_{i+k+1-}-\xi_{i+1}} B_{i+1,k-1}(x), \\ \qquad \text{if } \ \xi_i < \xi_{i+k} \ \text{ and } \ \xi_{i+1} < \xi_{i+k+1}. \end{cases} \tag{1.64}$$

We recall some properties of **B-splines functions of order k+1 (degree k)**:

Proposition 22 *[59, teorema 1.4.3, pag. 25] With the above notations:*

1. $B_{i,k}(x)$ *is a polynomial of degree* k,

2. $B_{i,k}(x) = 0$ *for* $x \notin \left[\xi_i, \xi_{i+k+1}\right]$,

3. $B_{i,k}(x) > 0$ *for* $x \in (\xi_i, \xi_{i+k+1})$,

4. $B_{i,k}(x)$ *form the basis in* $\mathbb{P}_{k,\overline{\Delta},s}$.

Definition 23 *[6, pag 218] Given a grid* $\overline{\Delta}$ *of the interval* $[a, b]$ *(1.62), a collocation approximate solution in k stages given by (1.63) is continuous piecewise polynomial function*

$$u_{\overline{\Delta}}(x) = \sum_{i=0}^{n} c_i B_{i,k}(x), \tag{1.65}$$

which reduces to a polynomial of degree at most k (order $k+1$) on each mesh subinterval and satisfies (1.2a) in grid $\overline{\Delta}$ and boundary conditions.

Since in this work we used combined methods: collocation method with B-splines functions and *Runge-Kutta* methods we recall some results [6, pp: 210-219].

Theorem 24 *[6, **Theorem 5.73, page 219**] Given a mesh $\overline{\Delta}$ of the interval $[a,b]$ given of (1.62), there is only one step explicit Runge-Kutta method with k stages (1.59) equivalent to the collocation method (1.65), (i.e. they have same order of convergence). Moreover*

$$u_{\Delta}(x_i) = y_i,\ u_{\Delta}(\xi_{ij}) = y_{ij},\ 0 \leq i \leq n\ , 1 \leq j \leq k,$$

where $x_i \in \Delta$, and ξ_{ij} are collocation points given by (1.57).

If $y(x)$ is the exact solution of (BVPL) problem and $u_{\overline{\Delta}}(x)$ is approximate collocation solution in k stages we denote by

$$e_{\overline{\Delta}}(x) := u_{\overline{\Delta}}(x) - y(x). \tag{1.66}$$

Theorem 25 *[6, **Theorem 5.75**, page 219] The $k-$stage collocation approximate solution exist for boundary nonlinear value problems (BVPN) and it is obtained in a stable way. Moreover, the error and its derivatives satisfy for every $i = 0, 1, ..., n$*

$$\max_{x\ \ i \leq x \leq x_{i+1}} |e_{\Delta}(x)| = \mathbb{O}(h^k),$$

$$\max_{x\ \ i \leq x \leq x_{i+1}} \left| e_{\Delta}^{(j)}(x) \right| = \mathbb{O}(h^{k+1-j}) \cdot \theta_i^{j-1},\ 1 \leq j \leq k,$$

where

$$\theta_i := \frac{h}{h_i},\ h_i = x_{i+1} - x_i,\ h = \max_{0 \leq\ i \leq\ n-1} \{h_i\},$$

and $e_{\Delta}^{(j)}(x)$ is derivative of order j.

Collocation spectral methods

1. Tchebychev polynomials-General properties. In their book [27], *Fox* and *Parker* collected the underlying principles of the *Tchebychev* theory. The polynomials whose properties and applications will be discussed were introduced more than a century ago by the Russian mathematician *P.L. Chebychev* (1821-1894). Their importance for numerical analysis was rediscovered around the middle of the 20−th century by *C.Lanczos* [43] .

Definition 26 *The polynomials* $T_k(x),\ k \in \mathbb{N}$ *defined by:*

$$T_k(x) = \cos(k \arccos(x)),\ -1 \leq x \leq 1,\ k = 0, 1, ...N, \tag{1.67}$$

are called the Tchebychev *polynomials of the first kind.*

Definition 27 *The polynomials* $U_k(x),\ k \in \mathbb{N}$ *defined by*

$$U_k(x) = \frac{\sin((k+1) \cdot \arccos(x))}{(k+1) \cdot \sqrt{1-x^2}}, x \in [-1, 1]\,;\ k = 0, 1, ...N, \tag{1.68}$$

are called the Tchebychev *polynomials of the second kind.*

Proposition 28 *(***Orthogonality***) [29, page: 6] The polynomials* $T_k(x),\ k \in \mathbb{N}$ *are orthogonal i.e,*

$$(T_n, T_m)_w = \frac{\pi}{2} \cdot c_n^1 \cdot \delta_{n,m};\ n, m \in \mathbb{N}, \tag{1.69}$$

relative to ***weight function***

$$w : I = [-1, 1] - > \mathbb{R}, \quad w\ (x) := \frac{1}{\sqrt[2]{1-x^2}} \tag{1.70}$$

where $\delta_{n,m}$ *stands for the* Kronecker *delta symbol and, throughout in this work, the coefficients are defined by*

$$c_n^1 := \begin{cases} 0, & \text{if } n < 0 \\ 2, & \text{if } n = 0 \\ 1, & \text{if } n \geq 1 \end{cases}. \tag{1.71}$$

Definition 29 *[44, pp: 5-6] We say that the functions :*

$$\phi_0, ..., \phi_N : [a, b] \to \mathbb{R},$$

form a Chebychev *system (or T-system) of order N on $[a, b]$, if any nonzero linear combination of these functions*

$$\sum_{k=0}^{N} \alpha_k \phi_k, \ \left(\sum_{j=0}^{N} \alpha_j^2 > 0 \right),$$

has at most N zeros on $[a, b]$.

Remark 30 *Using the results, [44, page: 8] it follows that the set of* Chebychev *orthogonal polynomials $T_k(x),\ k = 0, 1, 2, ...N,$ form a* Chebychev *system of order N on $[-1, 1]$ and moreover for every function:*

$$u(x) \in L_w^2(I) \ ,$$

where $L_w^2(I)$ is the set of square integrable functions on I relative to weight w, can be developed in a Chebychev *series*

$$u(x) = \sum_{k=0}^{\infty} \overset{*}{u} \cdot T_k(x), \tag{1.72}$$

where

$$\overset{*}{u} = \frac{2}{\pi c_k^1} (u, T_k)_w,$$

and c_k^1 are given by the relationship (1.71).

Spectral methods for o.d.e

The spectral methods are particulary attractive due to the following approximation properties. The distance between the exact solution $y(x)$ of the (BVPN) problem and its spectral approximation $u_N(x)$ is of order $\frac{1}{N^s}$ [29, **Remark 27,** page: 29] i.e,

$$\|y(x) - u_N(x)\| \leq \frac{C}{N^s},$$

where the exponent s depends only on the regularity (smoothness) of the exact solution $y(x)$. Moreover, if $y(x)$ is infinitely derivable, the above distance vanishes faster than any power of $1/N$, and this means spectral accuracy.

Let's consider the partition of the interval $[0, 1]$ so

$$0 = \xi_0 < \xi_1 < ... < \xi_N = 1,$$

where

$$\xi_j = \frac{-\cos\frac{\pi j}{N} + 1}{2},\ j = 0, 1, ..., N.$$

We observe that the nodes are symmetrically located around $1/2$ if N is even.

This nodes are called *Gauss-Lobatto -Chebychev* points (shortly *G.L.C)*. In this work we try to find the approximate solution $u_N(x)$ of the exact solution $y(x)$ of the (PVPN) or (PVPL) problem so:

$$u_N(x) = \sum_{k=0}^{N} u_j l_k(x), N \in \mathbb{N},$$

where $u_j := u_N(\xi_j)$ which satisfies (1.2a) in collocation points $\xi_j,\ j = 1, 2, ...,\ N-1$ and boundary conditions:

$$u(\xi_0) = \alpha,\ u(\xi_N) = \beta.$$

1.4.3 Examples of technical problems

1. Reaction diffusion equation (linear form)

In the paper [46][52] *P.A.Domenico, F.W. Schwartz,* described the propagation of heat from chemical reactions through a bar of given length L, the axis Ox is chosen along the bar, which occupies the interval $[0, L]$. We consider the problem of non-stationary heat transfer in the bar, and if $L = 1$ we take the heat source as follows

$$s(x) = 10e^{-5x\ ^2}.$$

The problem may change to the advection-diffusion equation on introducing the advection speed $U(x)$, constant or variable and considering the case of stationary and apply to the transport of pollutants in rivers, where $U(x)$ is the local speed of transport.

In this paper we consider the case $U(x) = m(constant)$. We treated the case $m = 4$, given a differential equation whose solution is non-oscillatory

$$-z'' + 4z'(x) = 10e^{(-5z\ ^2)};\ 0 < z < 1 \tag{1.73}$$

Using a substitution of the form [60, page: 77]

$$z(x) := \varphi(x)y(x),$$

where

$$\varphi(x) = e^{\ \frac{1}{2}\int_{x_0}^{x} p(s)ds}\ , \varphi(x) \neq 0,$$

differential equation (1.73) can be transformed as follows

$$-y'' + 4y(x) = 10e^{-5x\ ^2-2x}; 0 < x < 1. \tag{1.74}$$

2. Burden and Faires problem [17, problem 5, page: 561] This problem appears in the resistance structure calculation of certain construction

$$\begin{cases} -y'' - y = x, & x \in (0,1) \\ y\left(\frac{1}{4}\right) = \frac{\sin\frac{1}{4}}{\sin 1} - \frac{1}{4},\ y\left(\frac{1}{2}\right) = \frac{\sin\frac{1}{2}}{\sin 1} - \frac{1}{2} \end{cases} .. \tag{1.75}$$

3. Frequency domain equation for the vibrating string. This problem is considered in the paper of *Greengard and Rokhlin* [32, page: 443]:

$$\begin{cases} y''(x) + (k^2+5)y(x) = 5\sin(kx),\ 0 < x < 1 \\ y(c) = \sin(kc),\ y(d) = \sin(kd),\ 0 < c < d < 1 \end{cases} , \tag{1.76}$$

The exact solution is: u(x)=sin(px). Here we solve a problem using B-splines functions and Tchebychev Collocation method of type which arises when dealing with a frequency domain equation for vibrating string and we compare the costs (Run-times and Internal Memory). In order to demonstrate the performance of B-splines method on large scale oscillatory cases, we solved the problem for p ranging from 9 to 630 and n between 10

and 1200. We observed that reliable results can only be expected under the assumption of scale resolution, is small, i.e that $p/n < 1$. In this situation the error in approximating the exact solution

$$u(x) = sin(px)$$

is of order $O(h^p)$. We treated the case: $c = Pi/54$, $d = Pi/12$, $k = 9$.

4. Bratu's problem ([29], [69], [70]). Spontaneous combustion occurs in the model

$$\begin{cases} -y'' + \lambda e^u = 0, \ 0 < x < 1, \\ y(0) = y(1) = 0, \end{cases} \tag{1.77}$$

where $\lambda > 0$ is a parameter. Substituting a function of the form:

$$y(x) = -2\log(\cosh((x - 0.5)\frac{\theta}{2})/\cosh(\frac{\theta}{4}),$$

which satisfies the boundary conditions, into differential equation, we find that $y(x)$ is a solution if:

$$\theta = \sqrt{2\lambda}\sinh\left(\frac{\theta}{4}\right). \tag{1.78}$$

This nonlinear algebraic equation for θ has two, one or no solutions when $\lambda < \lambda_c, \lambda = \lambda_c$ or $\lambda > \lambda_c$, respectively. The critical value λ_c satisfies

$$1 = \frac{1}{4}\sqrt[2]{2\lambda_c}\sinh(\frac{\theta}{4}).$$

We treated the case $\lambda = \lambda_c = 1$.

A diagram of bifurcation for this problem is also available in the monograph of *U.Ascher, B.Mattheij and R.D.Russel* [6, page: 491].

5. The average temperature in a reaction-diffusion process [29, page: 45]

Consider the BVP problem

$$\begin{cases} u'' + u^p = 0, 0 < x < L \\ u(0) = u(L) = 0. \end{cases}. \tag{1.79}$$

It represents the steady state of the reaction-diffusion system. *Ph.Korman [42]* invokes a phase-plane analysis to observe that the positive solution $\overline{u}(x)$ is unique $\overline{u}(x) > 0$ for $x \in (0, L)$ and for any L this solution satisfies

$$\int_0^L \overline{u}(x)dx = \frac{\pi}{\sqrt{2}} \tag{1.80}$$

Also he have proved that the solution $\overline{u}(x)$ satisfies

$$\int_0^L \overline{u}(x)dx = \sqrt{2(p+1)}u_{\max}^{\frac{3-p}{2}}(F(1) - F(0))$$

where $F(v)$ is the primitive function of

$$v/\sqrt{1 - v^{p+1}},\ v = \frac{u}{u_{\max}}.$$

and the $u_{\max}$ is the maximum value of u on $[0, L]$.The independence on the domain condition (1.80) could be interpreted in the sense of *Levine [45]* where "large domains are less dissipative" than small domains.

D.Trif, C.Gheorghiu [30] prove that, for $p = 3$, the boundary value problem (1.79) has a unique positive solution and solve it using spectral *Galerkin* methods, convergence rate being $\mathbb{O}(10^{-8})$.The authors use Newton quasilinearization method, taking as starting solution the function

$$u_0 := (1 - x^2)^2;$$

further they demonstrates that this iterative process is convergent.

Remark 31 *For different values of parameter p the problem (1.79) may have a unique positive solution or multiple oscillatory solutions. Bifurcation diagram of this problem is found in the work [6, pag 491].*

6. The charge density in atoms of high atomic number. The Thomas-Fermi equation,

$$y'' = x^{-1/2}y^{3/2}, \tag{1.81}$$

is to be solved with boundary conditions

$$y(0) = 1, \quad y(\infty) = 0. \tag{1.82}$$

This (BVP) arises in a semiclassical description of the charge density in atoms of high atomic number. There are difficulties at both end points; these difficulties are discussed at length in *Davis* (1962) and in *Bender and Orszag* (1999). *Davis [24]* discusses series solutions for

$$y(x) \text{ as } x \to 0.$$

It is clear that there are fractional powers in the series. That is because, with $y(0) = 1$, ODE requires that

$$y(x) \sim x^{-1/2} \text{ as } x \to 0,$$

and hence there be a term $\frac{4}{3}x^{3/2}$ in series for $y(x)$. Of course, there must also be lower-order terms so as to satisfy the boundary condition at $x = 0$. *Bender & Orszag* discuss the asymptotic behavior of $y(x)$ as $x \to 0$. Verify that trying a solution of the form

$$y''(x) \sim ax^{\alpha},$$

yields

$$y_0(x) = 144x^{-3},$$

as an exact solution of the ODE that satisfies the boundary condition at infinity.

7. **Models for a transport, reaction and dissipation of pollutants in rivers.**

One model study by *Ames and Lohner* (1981) [2], gives rise to a system of three first-order PDE in one space variable x and time t. By looking for traveling wave solutions, that depend only the variable

$$z = x - t$$

they reduce PDE, to ODE

$$f'' = \beta g f \tag{1.83}$$
$$g'' = -\beta g h$$
$$h'' = \lambda \beta g h$$

Here f represents a pollutant, g bacteria, and h carbon the physical parameters λ and β are constants. After showing that the equations for g and h imply that

$$g(z) = E - \frac{h(z)}{\lambda} \tag{1.84}$$

where E is a given value for $g(\infty)$. They reduce the system of ODE so

$$h'' = \lambda\beta(E - \frac{h}{\lambda})h, \tag{1.85a}$$
$$f'' = \beta(E - \frac{h}{\lambda})f. \tag{1.85b}$$

Since:

$$h(\infty) = 0 \tag{1.86}$$

the equation (1.85a) can be approximated for large z by:

$$h'' = E\lambda\beta h$$

For $\beta = \lambda = 10$ and $E = 1$, this equation becomes

$$h'' = 100h \tag{1.87}$$

with solution

$$h(z) = Ae^{10z} + Be^{-10z}$$

Because $h(0) = 1$, $h(\infty) = 0$ implies $A = 0$ and $B = 1$, then the approximate solution $h(z)$ of the problem: (1.87) with conditions (1.86) is asymptotically multiple of e^{-10z}.

With $h(z) = e^{-10z}$, $\beta = \lambda = 10,\ E = 1$ the equation (1.85b) becomes

$$f'' = 10\ (1 - \frac{e^{-10z}}{10})\ f, \tag{1.88}$$

subject to boundary conditions

$$f(0) = 1,\ f(\infty) = 0. \tag{1.89}$$

Also for large z the equation (1.88) can be approximate by

$$f\ '' = 10f.$$

Solving this approximating ODE we find saturation behavior in transient area (an interval bounded by that starts from 0) and not the behavior of the asymptotically area where $h(z) \simeq 0,\ f(z) \simeq\ 0$.

We present two approximation methods for exact solutions of the problems (1.87) and (1.88) with conditions (1.86),(1.89). We divide the interval $[0, \infty)$ in two subintervals like: a transient area $[0, 10]$ and the asymptotically area $(10, \infty)$,where $z = 10$ is the positive solution of equation

$$r^2 - 100 = 0.$$

8. Flame propagation in nuclear reactors A reaction diffusion equation looks like the heat equation with a function $f(u)$ added on:

$$u_t = \Delta u + f(u) \tag{1.90}$$

Such equations appear in the sciences as models of diverse physical, chemical and biological phenomena. Since f may be nonlinear, explicit solutions cannot usually found. Wheareas the linear wave equation propagates arbitrary solutions at fixed speed, reaction diffusion equations may single out certain wave forms and allow only these to propagate without distortion. a typical problem for an equation of this kind investigates the existence, form and stability of these *traveling waves.* Such a solution can be written as

$$u(x, t) = U(z),\ z = x - ct \tag{1.91}$$

where c is the wave speed. The existence of traveling waves in chemical reactions was first observed and studied by *Luther* at the beginning of twenty century. *Fischer's* equation was one-dimensional and had a specific "logistic" reaction term

$$u_t = Du_{xx} + ru(1 - \frac{u}{K})$$

Fischer propose this equation as a model of diffusion of a species in a 1D habitat: D is the diffusion constant, r is the growth rate of the species, and K is the carrying capacity. A dimensionless version of the equation take the form:

$$u_t = u_{xx} + u(1 - u) \tag{1.92}$$

Since *1937* has also been used to study flame propagation in nuclear reactors. The solution to (1.92) depends on the initial data, $u_0(x)$.*Kolmogorov* proved that $u_0(x)$ is monotonic and continuous with

$$u_0(x) = \begin{cases} 1, & x < a \\ 0, & x > b \end{cases} \quad -\infty < a < b < \infty,$$

then the solution evolves into a traveling wave with speed $c = 2$. We can begin to understand the behavior of the *FKPP* equation by noting that since the right-hand side of (1.92)is zero $u = 0$ or $u = 1$

two solutions are obvious: $u(x,t) = 0$ or $u(x,t) = 1$ for all x, t.More generally it is clear that for initial data $u_0(x) = C$, the wave will remain flat for all t with $u(x,t) = C(t)$. Moreover the sign of the term $u - u^2$ such the positive solutions will be repelled from 0 and attracted to 1. Now the real interest in (1.92) lies in the behavior when both $u \simeq 0$ and $u \simeq 1$ are present: how does the wave get from one of these value to other and how does the wave front connecting the two move with time ?

The generally answer is that regions with $u \simeq 1$ tend grow,eating up adjacent region with $u \simeq 0$ and the wave may travel in either direction.In this book we solve this problem with conditions at infinity

$$u" + cu' + u(1 - u) = 0 \tag{1.93}$$

and the solution is to satisfy the boundary conditions

$$u(-\infty) = 1,\ u(\infty) = 0 \tag{1.94}$$

For "large" c the solution $u(\xi)$ is approximated by outer solution $u_0(\xi)$ that is the solution of ODE

$$u_0' + u_0(1 - u_0) = 0$$

Generally such a solution cannot satisfy both boundary conditions, but in this instance the solution

$$u_0(\xi) = \frac{1}{1 + e^{\xi}}$$

does.

Returning to the original independent variable, we have the outer solution

$$U(z) \approx \frac{1}{1 + e^{z/c}}.$$

9. Keller problem

In [40, Section 6.2] *Keller* discusses the numerical solution of a model of the steady concentration of a substrate in an enzyme-catalyzed reaction with *Michaelis-Menten* kinetics. A spherical region is considered and the PDE is reduced to an ODE by symmetry.

The equation

$$y\text{ "} + 2\frac{y'}{x} = \frac{y}{\varepsilon(y + k)} \tag{1.95}$$

involves two physical parameters, ε and k. The boundary conditions are $y(1) = 1$, and the symmetry conditions $y'(0) = 0$.

In this book we solve the problem using `BVP4c`,for the parameter values

$$\begin{aligned} \varepsilon &= 0.1 \\ k &= 0.1 \end{aligned}$$

We approximate $y(x)$ on the interval $[0, d]$ with a few terms of a *Taylor* series expansion and solve the BVP numerically on $[d, 1]$.Two terms of a *Taylor* series for $y(x)$ and

one for $y'(x)$ are satisfactory when

$$d = 0.001$$

We will need to communicate d and the parameters ε and k to our functions for evaluating the ODEs and boundary conditions.

Remark 32 *If the reader do not set any options will need to use* `[ ]` *as a placeholder in this argument that precedes unknown parameters in the call list of* `bvp4c`.

Chapter 2

Mesh Selection

All numerical methods for solving BVPs on a computer involve at some stage discretization of differential equations on a mesh. Choosing a good mesh is essential if a method is to be efficient for problems with solutions having narrow regions of rapid change. Essential parts of this chapter have been taken from ([6], [28],[33],[35]).

Indeed recall the importance of a good initial guess means a satisfactory initial guess to start the discrete nonlinear BVP, where a good initial guess means a satisfactory initial solution profile, discretized on an appropriate mesh. Since the computational effort for a given basic discretization method increases with number of mesh points we want the coursed possible mesh which still yields an approximate solution with an adequate error.

This means, in particular that the mesh must be sufficiently fine there exists a solution to the discretized system and that the discrete BVP adequately represents the continuous BVP.The choice of a mesh should generally be based on two considerations :

1. controlling discretization error,

2. preserving stability.

We shall loosely to the former as *accuracy considerations*

Consider the nonlinear BVP:

$$y' = f(x, y),\ a < x < y \tag{2.1}$$

and boundary conditions:

$$g(y(a), y(b)) = 0, \tag{2.2}$$

where as usual we assume that there is an isolated solution $y(x)$that we wish to compute and that f and g are sufficiently smooth. Suppose that a basic method is used which, for a mesh n on $[a, b]$, gives a corresponding discrete or continuous approximate solution $y_n(x)$.

2.1 Mesh Selection Problem

Given a BVP (2.1) and an error tolerance TOL, find a mesh

$$\Delta : a = x_1 < x_2 < ... < x_N < x_{N+1} = b \tag{2.3}$$

with

$$h = \max_{1 \le i \le N} h_i, h_i = x_{i+1} - x_i \tag{2.4}$$

such that N is "small" and the error in $y_n(x)$ as an approximation to $y(x)$ is less than TOL (in an appropriate norm, measuring absolute or relative errors).

The notion of a "small" N is certainly more qualitative than quantitative. We can attempt to make this precise in at least two ways.

One is to look for an optimal mesh, i. e., a mesh which yields a minimal mesh size N for a given error tolerance TOL. In practice, however, the exact solution of the resulting optimization problem turns out to be too expensive, if not impossible. Thus we settle for a quest for good meshes, i. e., meshes with N not much larger than the optimal one. To make the notion of a good mesh more quantitative, we could consider a sequence of tolerances and meshes and require that, as $TOL- > 0$, the mesh size $N = N(TOL)$ satisfies

$$N(TOL) < CN^*(TOL)$$

with $N^*(TOL)$ the optimal mesh size for this error tolerance and $C = 1$ a constant independent of TOL.

A dual formulation for the mesh selection problem is often found useful. This formulation requires, given a mesh size N, to find a mesh which minimizes the error in some norm.

For a given BVP and numerical method, the appropriateness of the discretized system is directly determined by the mesh. Generally speaking, the mesh must be fine in regions where the desired solution changes rapidly but can be relatively coarse elsewhere, provided that the resulting scheme is sufficiently stable. A suggestive example is presented in book [6, pag:360].

When attempting to analyze mesh selection, it is easy to see what we want: the role of mesh selection is to obtain a global solution to (2.1) as cheaply as possible for a given accuracy. Unfortunately, the mesh selection analysis is faced with an inevitable dilemma: **A strategy is generally more successful if it utilizes the special properties of the particular numerical method being used, namely its error form.** However, this error form is normally based upon an asymptotic analysis, and the very fact that mesh selection is important at all is often because one does not yet have a good numerical approximation. Consider mesh selection for the limiting case $h \to 0$, or even $N \to \infty$. For when $N \to \infty$, all quasiuniform meshes (i. e., all meshes satisfying)

$$h \leq \overline{K} \cdot h_{\min}, \ h_{\min} := \min_{1 \leq i \leq N} h_i, \tag{2.5}$$

for some fixed constant $\overline{K}$ yield an $\mathbb{O}(N^{-1})$ error when a difference scheme of order s is used.

In general, we distinguish between two types of mesh selection:

1. a priori and

2. adaptive.

The first type is usually relevant in specialized circumstances. For instance, one may have some information (perhaps obtained by a preliminary mathematical analysis on the

shape of the desired solution for a given BVP. Such an initial solution profile can then be used to produce an initial mesh, like an equidistributing mesh.

In most of this chapter we restrict consideration to adaptive mesh selection, where the mesh and the approximate solution are repeatedly updated until prescribed error criteria are deemed satisfied.

Although our characterization is somewhat artificial, we divide adaptive mesh selection methods into two basic groups:

1. direct and

2. transformation methods.

The direct methods have generally been the most useful in applications to date, and in opinion of *Ury Ascher* they are the most important.

2.1.1 Error equidistributing and monitoring

Given a mesh Δ as in (2.2) and an approximate solution $y_n(x)$, we denote by e_i, some measure of the error on the i^{th} subinterval of the mesh,$[x_i, x_{i_{+1}}]$. This may be an

absolute or a relative error, or a combination of both. For instance, we may set

$$e := |y_i - y(x_i)|\,,\ 1 \le i \le N+1 \tag{2.6}$$

when a simple finite difference scheme is used on π, or

$$e_i := \max_{x_i \le x \le x_{i+1}} |y_\pi x) - y(x)| =: ||y_\pi - y||_i,\ 1 \le i \le N \tag{2.7}$$

if the approximate solution is defined continuously. Throughout this chapter we use the max (or *sup*) function norm.

The error e_i, depends on the subinterval $[x_i$,$x_{i+1}]$ and generally increases as h_i increases. It will turn out to be convenient to consider a corresponding error measure d_i, which only

varies linearly with h_i; i. e., we write

$$d_i = h_i \Phi_i, \tag{2.8}$$

with Φ_i, independent of h_i.

For example, if the discretization method is a difference scheme of order s then the results in [6, Section 5.5] tell us (under sometimes unrealistic assumptions) that

$$|y_i - y(x_i)| = C h_i^s \, |y_i^s(x_i)| + \mathbb{O}(h_i^{s+1}) + o(h^s)$$

so an expression like 2.8 can be written

$$\Phi_i \sim C^{1/s} \left| y^{(s)}(x_i) \right|^{1/s}.$$

This expression for Φ_i, does depend on the mesh, but not in a crucial way, and we ignore this dependence when obtaining next results.

Given N, we can try to choose the mesh πso that $\max_{1 \leq \, i \leq \, N} \ |e_i|$ is minimized. A related, simpler choice is to minimize

$$\max \ |d_i|$$

instead. Using this, we obtain a minimax problem with only one constraint

$$\min(|d| : \sum_{i \, = \, 1}^{N} h_i = b - a. \tag{2.9}$$

The solution of the trivial optimization problem (2.9) is to make all d_i, equal to the same constant λ, which yields

$$\begin{aligned} h_i &= \lambda / \Phi_i, 1 \leq i \leq N, \\ \lambda &= (b-a) / \sum_{j \, = 1}^{N} \Phi_j^{-1}. \end{aligned} \tag{2.10}$$

We now generalize the above and consider a smooth function Φ instead of just a set of discrete values on π.

Definition 33 *[6, pag:363] A function* $\Phi(x, y)$ *is called a* monitor function *if it has contonous partial derivable on*

$$\{(x, v) : v \in S_\rho(y(x)), a \leq x \leq b$$

for some sphere S_ρ *of radius* $\rho > 0$ *about* $y(x)$*, and if*

$$\Phi(x, y(x)) \geq \delta > 0,\ a \leq x \leq b \tag{2.11}$$

fore some $\delta > 0$*. A mesh* π *is equidistributing with respect to a monitor function* $\Phi(x, y(x))$ *on* $[a, b]$ *if for some constant* λ

$$\int_{x_i}^{x_{i+1}} \Phi(x, y(x))dx = \lambda,\ 1 \leq i \leq N., \tag{2.12}$$

It follows that:

$$\lambda = \frac{\theta}{N}, \tag{2.13}$$

where:

$$\theta = \int_a^b \Phi(x, y(x))dx \tag{2.14}$$

Also the mesh π is equidistributing with to a mesh function

$$d = (d_1(y)..., d_N(y)^T$$

if,

$$|d_i(y)| = const, 1 \leq i \leq N \tag{2.15}$$

A sequence of meshes $[\pi_N]_{N\,=\,N_0}^{\infty}$ is asymptotically equidistributing (as ,eq.) with respect to $\Phi(x, y(x))$ if

$$\int_{x_i}^{x_{i+1}} \Phi(x, y(x))dx \equiv \lambda(1 + \mathbb{O}(h)),\ 1 \leq i \leq N \tag{2.16}$$

and asymptotically equidistributing with respect to d if

$$|d_i(y)| = const(1 + \mathbb{O}(h),\ 1 \leq i \leq N. \tag{2.17}$$

For convenience, we shall just refer to a mesh π as being as.eq., or to an as.eq. mesh, and implicitly assume that N is sufficiently large that an approximate solution in fact exists for an appropriate unspecified sequence of meshes. We are now prepared to investigate various ways to choose monitor functions and select as.eq. meshes.

2.1.2 Direct Methods

A mesh selection method (or rather, a solution method) from the class considered here may be described as follows:

Outline: A direct method Given a discretization scheme and an initial solution profile

REPEAT

- Determine a mesh π from the current solution
- Solve the BVP (2.1) for $y_n(x)$on the new mesh

UNTIL error tolerance is satisfied.

Normally one starts from a given initial (coarse) mesh and first solves (2.1) on it.

The main question is, given an approximate solution $y_n(x)$on a current mesh π, how is a new, refined mesh π^* determined? In view of the preceding section, we would know how to proceed if a **monitor function** try were available: We would determine an as.eq. mesh if with respect to $\Phi(x, y(x))$, where N and the constant λ in (2.13) are determined by using the desired error tolerance. For a monitor function we now consider two choices:

1. one arising from equidistributing the local truncation error and
2. valid for collocation methods, using a localized form of the global error.

Suppose that the discretization method used to solve (2.1) gives an approximate solution $y_n(x)$ which satisfies the error bound

$$\|y_n - y\| \leq K \max |\tau_i(y)| \tag{2.18}$$

. Under normal circumstances, the local truncation error of a difference scheme of order s satisfies

$$\tau_i(y) = h_i^s T(x_i) + \mathbb{O}(h_i^p),\ p > s \tag{2.19}$$

where $T(x)$ is continues function involving high derivatives of $y(x)$. It is then natural to choose the *monitor function*

$$\Phi(x, y(x)) := |T(x)|^{1/s} \tag{2.20}$$

The theoretical significance of this is given in

Theorem 34 *[6, pag:365] Suppose that an as.eq. mesh n with respect to* $|T(x)|^{1/s}$ *is determined, i. e.,*

$$\int_{x_i}^{x_{i+1}} |T(x)|^{1/s}\, dx = \lambda(1 + \mathbb{O}(h)),\ 1 \leq i \leq N \tag{2.21}$$

where:

$$\begin{aligned} \lambda &= \frac{\theta}{N}, \\ \theta &= \int_a^b |T(x)|^{1/s}\, dx. \end{aligned}$$

Then the corresponding discretization method with error satisfying (2.18) and (2.19) gives:

$$\|y_n - y\| \leq K\lambda^s(1 + \mathbb{O}(h) + \mathbb{O}(h^p) \tag{2.22}$$

B.Matheij [6] prove:

Corollary 35 *If*

$$M_L = \{x : |T(x)|^{1/s} \geq L\},$$

with L chosen such that

$$\int_{M_L} |T(x)|^{1/s}\, dx = \frac{1}{2}\int_a^b |T(x)|^{1/s}\, dx = \frac{\theta}{2} \tag{2.23}$$

then the error satisfies

$$\|y_n - y\| \leq K(\frac{2\mu(M_L)}{N})^s \|T\| (1 + \mathbb{O}(h)) + \mathbb{O}(h^p), \tag{2.24}$$

where $\mu(M_L)$ is the measure of M_L.

Remark 36 *For a BVP whose solutiony(x) has a boundary or interior layer (i. e., a small subinterval where the solution varies fast), if (2.19) holds and $T(x)$ involves some derivative(s) of $y(x)$ then M_L is often an interval of width proportional to the layer width, say ϵ. In such a case, the first term in the bound (2.24) can be independent of the layer width, and the error from an as.eq. mesh can be essentially independent of ϵ.*

Remark 37 *It is not always obvious how $T(x)$ can be approximated in practice. If an approximation to $y(x)$ is available, then its derivatives or divided differences (i. e.derivatives of an interpolant, if it is only a mesh function) can normally be used to approximate the derivatives appearing in $T(x)$. This is discussed in detail for collocation in the next section.*

2.2 A Mesh Strategy for collocation

From the last section we have a general mesh selection approach in which we alternately solve a BVP using a discretization method and adjust the mesh by using a monitor function which itself involves the current approximate solution. Here we choose the discretization method to be spline collocation, which can be viewed as a family of high-order finite difference schemes as described in last section. The BVP is a higher- order equation, unlike that in the previous section, but the viewpoint taken there easily generalizes, since the determination of the mesh and the solution of the unmodified BVP are done separately. For the monitor function we can, of course, choose the local truncation error as

described in last section, but a robust algorithm is obtained by instead using the leading term of the error away from the mesh points ([64][65]). While the resulting strategy is far from being the only reasonable one, it has proven very successful in practice and is the one used in the collocation code [10].

2.2.1 A practical mesh selection algorithm

We use the spline collocation method for solving the BVP

$$Lu = u^{(m)} = \sum_{i=1}^{m} c_l(x)u^{(l-1)} = q(x),\ a < x < b \tag{2.25}$$

$$B_1 y(a) + B_2 y(b) = \beta, \tag{2.26}$$

where

$$y(x) := (u(x), u'(x), ...u^{(m-1)}(x))^T$$

The collocation solution $u_n(x)$ is determined, for a given mesh π and for k given collocation points on each mesh subinterval, as that piecewise polynomial function of order $k+m$ which has $m-1$

continuous derivatives {i. e.,$u_n(x) \in \mathbb{P}_{k+m} \cap\ C^{m-1}[a,b]$ and satisfies the differential (2.25) equation at the collocation points and the BC (2.26).

Then for $x_i < x < x_{i+1}$:

$$u(x) - u_n(x) = h_i^{k+m} u^{(k+m)}(x_i) P(\frac{x - x_i}{h_i})(1 + \mathbb{O}(h_i)) + \mathbb{O}(h^p), \tag{2.27}$$

where

$$P(\xi) = \frac{1}{k!(m-1)!} \int_0^{\xi} (t-\xi)^{m-1} \prod_{l=1}^{k} (t - \rho_l) dt. \tag{2.28}$$

For the Gauss points $|\rho_i|_1^k$, which we shall use here, $p = 2k$. Recall that we have the local representation 2.27 for the global error, because the scheme is actually superconvergent for certain solution and derivative values.

Thus, corresponding to (2.18) we have

$$\|u_\pi - u\|_i := \max_{x_i \le x \le x_{i+1}} |u_n(x) - u(x)| \le C_i h_i^{k+m} \left\|u^{(k+m)}\right\|_i + \mathbb{O}(h^{2k})$$

where $C_i = \widehat{C}(1 + \mathbb{O}(h_j))$,

$$\widehat{C} := \max_{0 \le \xi \le 1} |P(\xi)| .$$

A natural choice for a monitor function is therefore

$$\left|u^{(k+m)}(x)\right|^{1/(k+m)}, \text{ for } x_i \le x < x_{i+1} \tag{2.29}$$

The exact solution $u(x)$ and hence also $u^{(k+m)}(x)$ are unknown, and one cannot directly replace$u(x)$ by $u_\pi(x)$ in (2.29) because $u_\pi.^{(k+m)} = 0$.However, suppose that

$$v(x) \in \mathbb{P}_{2,\pi} \cap \ C[a,b]$$

is the piecewise linear function which interpolates $u_\pi^{(k+m-1)}(x)$ at the subinterval midpoints; i. e.

$$v(x_{i+1/2}) = u_\pi^{(k+m-1)}(x_{i+1/2})$$

$l < i < N$, and define the monitor function like:

$$\Phi(x, v(x)) := |v'(x)|^{1/(k+m)} \tag{2.30}$$

Since $P^{(k+m-1)}(1/2) = 0$ implies:

$$v(x_{i+1/2}) = u_\pi^{(k+m-1)}(x_{i+1/2}) = u^{(k+m-1)}(x_{i+1/2}) + \mathbb{O}(h_i^2), \ 1 \le i \le N$$

and hence $v'(x) = u^{(k+m)}(x)(1 + \mathbb{O}(h))$;i.e is an $\mathbb{O}(h)$ approximation to the monitor function in (2.29). Moreover $v'(x)$ is a piecewise constant function, so

$$\theta := \int_a^b |v'(x)|^{1/(k+m)} \, dx \tag{2.31}$$

is easy to compute. Finding an as.eq mesh (2.3) such that

$$\int_{x_i}^{x_{i+1}} |v'(x)|^{1/(k+m)} \, dx = \frac{\theta}{N}, 1 \le i \le N$$

is also easy since

$$t(x) := \frac{1}{\theta}\int_a^x |v'(\xi)|^{1/(k+m)}\, d\xi \tag{2.32}$$

a piecewise linear. Thus if we know x_i then finding x_{i+1} such that

$$t(x_{i+1}) = \frac{1}{N} \tag{2.33}$$

amounts to inverse interpolation for this simple function. From the last theorem , if π^* is the new mesh determined from (2.31) then the new spline collocation solution $u_{\pi^*}(x)$

satisfies :

$$\|u_{\pi^*} - u\| \leq \widehat{C}(\frac{\theta}{N})^{k+m}((1 + \mathbb{O}(h)) + \mathbb{O}(h^{2k}) \tag{2.34}$$

A new mesh size N such that the collocation solution on the corresponding as.eq. mesh has uniform error less than tolerance TOL, can be predicted from (9.20) by choosing :

$$N = \theta(\frac{\widehat{C}}{N})^{1/(k+m)} \tag{2.35}$$

Here $\widehat{C}$ is known from (2.27). If TOL is a relative tolerance, then a trivial modification to (2.35) is made using $\|u_\pi\|$.

2.2.2 Algorithm for collocation with mesh selection

Input: A BVP (2.25) + (2.26), a collocation method with $k, m, \widehat{C}$, an error tolerance TOL, and an initial mesh π.

Output: an approximate solution $u_\pi\ (x)$

1. Set **FLAG** := false

2. **REPEAT**

2.1 Solve the collocation equations for $u_\pi(x)$.

2.2 Form the piecewise linear interpolant $v(x)$ to $u_\pi^{(k+m-1)}(x)$ using values at $x_{i+1/2}(1 \leq i \leq N)$

2.3 Calculate θ from (2.31)

2.4 IF $\widehat{C}((\frac{\widehat{\theta}}{N})^{(k+m)} < TOL$ THEN FLAG := true

2.5 ELSE

Find a new N from 2.35.

Solve for a new mesh π from (2.32) +(2.33)

UNTIL FLAG = true (or iteration limit exceeded)

To implement a practical mesh selection strategy, additional bells and whistles are needed to ensure that the strategy does not go agree. *Carl de Boor [13]* propose package **COLSYS**, for a scalar **BVP**. Also he suggest for robustness, errors are estimated for mesh selection and for termination of computation in two different ways. The rough error estimation for mesh selection has already been described. For termination (and to provide error estimates for intermediate meshes), an extrapolation principle is used. specifically, when two collocation solutions $u_{\pi_1}(x)$ and $u_{\pi_2}(x)$ are computed on meshes π_1 and π_2, where π_2 is π_1 "doubled" (i. e.,each subinterval is halved), the error on π_2 is estimated from (2.27) by

$$u(x) - u_{\pi_2}(x) = \frac{1}{1 - 2^{k+m}}(u_{\pi_1}(x) - u_{\pi_2}(x)) \tag{2.36}$$

Given an approximate solution $u_\pi(x)$, the quantities

$$r_1 = \max_{1 \leq i \leq N} h_i(\frac{\widehat{C}\,\|v'\|}{TOL})^{1/(k+m)},$$

$$r_1 = \sum_{i\,=\,1}^{N} h_i(\frac{\widehat{C}\,\|v'\|}{TOL})^{1/(k+m)},$$

$$r_3 = \frac{r_2}{N},$$

are computed. The ratio r_1/r_3 gives some idea of the gain to be achieved by redistribution. Specifically, if it is large then the maximum subinterval error estimate is significantly larger than the average one, and the mesh is not well-distributed. The code checks to see if :

$$\frac{r_1}{r_2} \leq 2 \tag{2.37}$$

- If (2.37) is satisfied, then the mesh is considered to be sufficiently equidistributing to not warrant construction of an entirely new set of points $[x_i]$, and the mesh is

"doubled," i.e.,

$$\pi^* = [x_1, x_{3/2}, x_2, ..., x_N, x_{N+1/2}, x_{N+1}]$$

This facilitates error estimation with (2.36).

- If (2.37) is not satisfied, then r_2 predicts the number of points needed to satisfy the tolerance [cf. (2.35)]. Normally

$$N^* = \min[N, \max(N, r_2)]$$

is used instead of r_2, mainly to prevent jumping to incorrect conclusions early in the mesh selection process. Then the new mesh if is chosen using (2.32). However, this strategy is not followed and the mesh is doubled instead if one of the following holds:

(i) N is less than it was for the mesh previous to π,

(**ii**) the current N has been used for three consecutive meshes, or

(**iii**) there have been three consecutive equidistributing and doubling (i. e., $N/2$ and N mesh points have been used three consecutive times). These precautions prevent cycling or, again, prematurely

jumping to conclusions about what a desirable mesh is, by insuring that N increases gradually. Also, when the choice

$$N^* = \frac{r_2}{2} < N$$

is made, then doubling, it is to be hoped, will provide a solution (and error estimate) satisfying the requested tolerance.

The user also has the option to (i) insert an initial mesh, (ii) insert fixed points (points required to be included in all subsequent meshes), and (iii) turn off the mesh selection so that only mesh doubling is used. These can be particularly useful features if one has special information about the solution, such as the location of boundary layers.

2.2.3 Transformation methods

With transformation methods, a change of variables is applied to the given BVP (2.1) to produce a transformed problem which is, we hope, more amenable to effective numerical computation. Thus, a coordinate transformation $x(t)$ of the independent variable is sought such that as a function of t the solution $y(x(t))$ of (2.1) looks smoother, i. e., varies slowly. We shall see how the direct methods of the previous sections can be viewed as particular ways of achieving such a coordinate transformation (in particular see (2.32)]. Indeed, *the monitor function* is used to provide a description of suitable coordinate transformations.

Definition 38 *A coordinate transformation* $t(x), t : [a, b] -> [0, 1]$, *is called* admissible *if it satisfies*

$$\frac{dt}{dx} = \frac{1}{\theta}\Phi(x, y(x)),\ a < x < b \tag{2.38}$$

$$t(a) = 0,\ t(b) = 1, \tag{2.39}$$

where $\Phi(x, v(x))$ *is* monitor function *and* θ *is given by (2.14).*

Explicit method

If some knowledge of the solution $y(x)$ is available, then an explicit coordinate transformation $x(t)$ can often be chosen to appropriately conform to this solution's behavior and to furthermore satisfy

$$\begin{aligned} \frac{dx}{dt} &> 0,\ 0 < t < 1 \\ x(0) &= a,\ x(1) = b. \end{aligned}$$

The BVP (2.1) can then be rewritten as

$$\widehat{y}'(t) = \widehat{f}(t, \widehat{y}(t))x'(t),\ 0 < t < 1 \tag{2.40a}$$

$$g(\widehat{y}(0), \widehat{y}(1)) = 0,$$

where $\widehat{y}(t) := y(x(t))$ and $\widehat{f}(t, \widehat{y}(t) := f(x, y(x))$.We must to choose $x\{t)$ in such a way that $\widehat{y}(t)$ will be smooth and well-behaved, so that the transformed BVP will be "easy" to solve. Since this frequently involves smoothing out regions of rapid variation that $y(x)$ may have, $x(t)$or its inverse $t(x) = x^{-1}$ (t) is often called a stretched coordinate. Ideally, the coordinate stretching should be sufficiently successful that the transformed BVP (2.40) could be efficiently solved on a uniform mesh (using our basic numerical method).

Implicit method

Obvious disadvantages of explicit coordinate transformations are that a priori knowledge of the behavior of $y(x)$ is needed and that choosing an appropriate $x(t)$ is at best very complicated if components of $y(x)$ are badly behaved in several regions of $[a, b]$. An alternative is to build an implicit mesh selection strategy into the problem statement. There are many approaches to choosing an admissible coordinate transformation $y(x)$, or $t(x)$, for which $\widehat{y}(t)$ is well-behaved. By "well-behaved" we mean here that an equally spaced mesh in the variable t is satisfactory for solving the transformed problem for $y(t)$ efficiently. Suppose we have decided upon the choice of the *monitor function* $\Phi(x, y(x))$. We can use the conditions (2.40a) for an admissible coordinate transformation to augment the original BVP (2.1). The requirement of an equidistributing mesh of size N, i. e., a mesh satisfying (2.12), can be simply expressed in terms of $t(x)$ using by

$$t(x_{i+1}) - t(x_i) = \frac{1}{N}(\frac{\lambda}{\theta}),\ 1 \leq i \leq N-1 \tag{2.41}$$

The mesh selection strategy can thus be accomplished if we solve the BVP (2.1, 2.38) subject to (2.41) with 6 an undetermined constant . This, however, can be written in an equivalent form as follows: Write (2.38) in terms of t as the independent variable, add a

trivial ODE for the constant 9, and add boundary conditions to obtain the system

$$\begin{aligned} \widehat{y}'(t) &= \widehat{f}(t, \widehat{y}(t))x'(t) \\ x'(t) &= \theta \cdot \widehat{\Phi}(t, \widehat{y}(t))^{-1}. \\ \theta'(t) &= 0,\ 0 < t < 1 \\ g(\widehat{y}(0), \widehat{y}(1)) &= 0, \\ x(0) &= a,\ x(1) = b, \end{aligned} \tag{2.42}$$

where $\widehat{\Phi}(t, \widehat{y}(t)) := \Phi(x, y(x)$. It is also nonlinear even if the original BVP is linear. A nice feature of the BVP (2.42) is that when it is solved, the constraints (2.41) are simply satisfied by letting

$$x_i = x(\frac{i-1}{N}), 1 \leq i \leq N+1 \tag{2.43}$$

In practice, of course, the BVP (2.42) must be solved approximately. It is to be hoped, though, that the solution is smooth and can thus be computed on the uniform mesh $\{0, l/N, ..., 1\}$, whereby approximate values of

$$y(x_i) = \widehat{y}((i-1)/N)$$

with x_i, defined in (2.43) are obtained. In general, a monitor function $\varphi(x, y(x))$ is chosen as some measure of the change in the behavior of $y(x)$ which is also reflected in the error behavior for the numerical solution. One common choice is the *arc length monitor function*,

$$\varphi(x, y(x)) = \sqrt{1 + |y'(x)|_2^2} = \sqrt{1 + |f(x, y(x))|_2^2} \tag{2.44}$$

The advantages of the *arc length monitor* in this context are that it does not involve high derivatives of $y(x)$ and that it is a smooth function of its arguments.

This is much more important here than in the previous subsections, because the monitor function is one of the BVP unknowns to be actually computed.

Chapter 3

Cubic B-spline collocation method with uniform mesh

3.1 Mathematical support

The purpose of this section is to approximate the solution of the following problem:

$$\begin{aligned} Ly(x) &= r(x), 0 \leq x \leq 1 \\ y(a) &= A,\ y(b) = B, \\ 0 &< a < b < 1, \end{aligned} \tag{3.1}$$

where

$$Ly(x) := -\frac{d}{dx}(\frac{dy}{dx}) + q(x)y(x), 0 \leq x \leq 1. \tag{3.2}$$

and $q(x), r(x) \in C(0,1), q(x) < 0, y(x) \in C^2(0,1)$, using cubic B-splines collocation method. In fall back we assume that the function $r(x), q(x)$ are such that the problem (3.1) has a unique solution. To describe the basic method in this and later sections we choose a uniform mesh of the given interval $(0,1)$ like

.

$$\Delta := 0 = x_0 < x_1 < ... < x_{n+1} = 1, x_j = j \cdot h,\ j = 0, 1, ..., n+1,\ h = \frac{1}{n+1}. \tag{3.3}$$

We introduce the domain of definition of the operator L, defined by (3.2), as

$$D_B(L) := \{u \in C^2(0,1) | Lu \in C^2(0,1) \text{ and } u(a) = A, u(b) = B \ \},$$

and suppose that $D_B(L)$ is dense in $C^2(0,1)$ such that

$$L : D_B(L) \subset C^2(0,1) \to C^2(0,1).$$

We also define

$$U := \{u \in D_B(L) : u|[x_i, x_{i+1}] \in \pi_3, u\prime \in C[x_j, x_{j+1}]\},$$

where π_3, is the set of polynomials of degree at most three. We use the following notations

$$q_j := q(x_j), r_j := r(x_j).$$

Obviously, $dim\ U = n + 2$. Since $U \subset C^2(0,1)$ (see [59]) there exists in U a basis consisting of cubic B-splines functions:

$$\{s_0, s_1, ...s_{n+1}\}$$

and moreover ([72, pag 69])

$$s_i(x) = S(\frac{x}{h} - i),\ i = 0, 1, 2, ...n + 1 \tag{3.4}$$

$$s\ "_i(x) = \frac{1}{h^2} S"(\frac{x}{h} - i), i = 0, 1, 2, ...n + 1 \tag{3.5}$$

where

$$S(x) = \begin{cases} 0, \text{ if } x \in \mathbb{R} \setminus [-2, 2] \\ (2-x)^3 - 4(1-x)^3 - 6x^3 + 4(1+x)^3 \text{ , if } x \in [-2, -1] \\ (2-x)^3 - 4(1-x)^3 - 6x^3, \text{ if } x \in [-1, 0] \\ (2-x)^3 - 4(1-x)^3, \text{if } x \in [0, 1] \\ (2-x)^3, \text{ if } x \in [1, 2] \end{cases}.$$

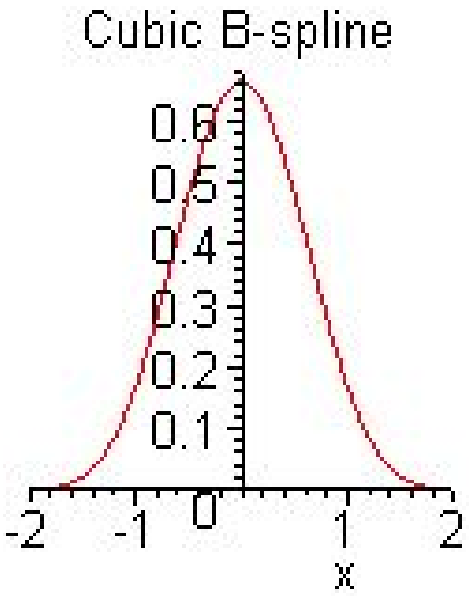


Figure 3.1: Cubic-Spline

The graph is depicted in figure 3.1:

We observe that

$$S(0) = \frac{1}{6};\ S(1) = S(-1) = \frac{1}{24}; S(1) = S(-1) = \frac{1}{4}S(0) \tag{3.6}$$

$$S^{"}(x) = \begin{cases} 0, \text{if } x \in \mathbb{R} \setminus [-2, 2] \\ x + 2, \text{ if } -2 \leq x \leq -1 \\ -3x - 2, \text{ if } -1 \leq x \leq 0 \\ 3x - 2, \text{ if } 0 < x \leq 1 \\ 2 - x, \text{ if } 1 \leq x \leq 2 \end{cases}$$

We also see

$$S\,"(0) = -2; S\,"(-1) = S\,"(1) = 1; S\,"(1) = S\,"(-1) = -\frac{1}{2}S\,"(0) \tag{3.7}$$

Using Orthogonal Spline Collocation Methods an approximate solution $s_y(x) \in U$ has

the form

$$s_y(x) = \sum_{i=0}^{n+1} c_i \cdot s_i(x),\ c_i \in \mathbb{R}, i = 0, 1, ..., n+1$$

Moreover, from (3.4)

$$s_y(x) = \sum_{i=0}^{n+1} c_i \cdot S(\frac{x}{h} - i),\ c_i \in \mathbb{R}, i = 0, 1, ..., n+1$$

Lemma 39 *For any $i = 0, 1, 2, ..., n, n+1,\ j = 1, 2, ...n$ the following relations hold*

$$s_i(x_j) = \begin{cases} \frac{1}{24},\ if\ \ i = j-1, i = j+1 \\ \frac{1}{6}, if\ i = j \\ 0, otherwise \end{cases} \tag{3.8}$$

$$s_i^{"}(x_j) = -\frac{1}{2 \cdot h^2} \begin{cases} -\frac{1}{2}, if\ i = j-1, i = j+1 \\ 1, if\ i = j \\ 0, otherwise \end{cases} \tag{3.9}$$

Proof. Since $s_i(x_j) \neq 0$, only for $i = j-1, j, j+1$,using (3.5) we obtain

$$\begin{aligned} \text{if } i &= j-1 \Rightarrow s_i(x_j) = S(j-i) = S(1) = \frac{1}{24} \\ \text{if } i &= j \ \Rightarrow s_i(x_j) = S(j-i) = S(0) = \frac{1}{6} \\ \text{if } i &= j+1 \Rightarrow s_i(x_j) = S(j-i) = S(-1) = \frac{1}{24} \end{aligned}$$

Also $s_i"(x_j) \neq 0$, only for $i = j-1, j, j+1$,using (3.5) we obtain

$$\begin{aligned} \text{if } i &= j-1 \Rightarrow s"_i(x_j) = \frac{1}{4h^2} \\ \text{if } i &= j \ \Rightarrow s"_i(x_j) = -\frac{1}{2h^2} \\ \text{if } i &= j+1 \Rightarrow s"_i(x_j) = \frac{1}{4h^2} \end{aligned}$$

■

Lemma 40 *For any $x_0 \in [x_j, x_{j+1}], j = 1, 2, ..., n-1$ the following inequalities hold*

$$0 < \lim_{h\to 0} s_{j+2}(x_0) = \lim_{h\to 0} s_{j-1}(x_0) < \frac{1}{24} \tag{3.10}$$

$$\frac{1}{24} < \lim_{h\to 0} s_{j+1}(x_0) = \lim_{h\to 0} s_j(x_0) < \frac{1}{8} \tag{3.11}$$

Proof. Let $\lambda(h) := s_{j+2}(x_0); \eta(h) := s_{j+1}(x_0); \rho(h) := s_j(x_0); \varphi(h) = s_{j-1}(x_0)$.In ([72, pag 72]) one shows that

$$\begin{aligned} s_{j+2}(x_0) &= S(\frac{x}{h} - j - 2) = \\ &= \frac{1}{24}\begin{cases} [(j+4)h - x]^3 - 4[(j+3)h - x]^3 + 6[(j+2)h - x]^3 - 4[(j+1)h - x]^3 \\ \text{if } h \le x \le (j+1)h \\ [(j+4)h - x]^3 - 4[(j+3)h - x]^3 + 6[(j+2)h - x]^3 \\ \text{if } (j+1)h \le x \le (j+2)h \\ [(j+4)h - x]^3 - 4[(j+3)h - x]^3 \\ \text{if } (j+2)h \le x \le (j+3)h \\ [(j+4)h - x]^3 \\ \text{if } (j+3)h \le x \le (j+4)h \\ 0, \text{ otherwise} \end{cases} . \end{aligned}$$

Because $x_0 \in [x_j, x_{j+1}]$ and $x_j = jh$ it follows $x_0 \in [jh, (j+1)h]$. Then

$$0 < \lim_{h\to 0} \lambda(h) = \frac{x_0^3}{24}$$

Since $0 < x_0 < 1$, we obtain the following relations on $\lambda, \varphi, \eta, \rho$

$$\begin{aligned} 0 &< \lim_{h\to 0} \lambda(h) < \frac{1}{24}, \\ 0 &< \lim_{h\to 0} \varphi(h) = \frac{1}{24} \lim_{h\to 0} \{[(j+1)h - x_0]^3 - 4[(jh - x_0]^3 + 6[(j-1)h - x_0]^3\} = \frac{x_0^3}{24} \end{aligned}$$

and:

$$0 < \lim_{h\to 0} \varphi(h) < \frac{1}{24}$$

also:

$$\begin{aligned} 0 &< \lim_{h\to 0}\eta(h) = \frac{1}{24}\lim_{h\to 0}\{[(j+3)h - x_0]^3 - 4[jh - x_0]^3\} == \frac{x_0^3}{8} \\ \frac{1}{24} &< \lim_{h\to 0}\eta(h) < \frac{1}{8} \end{aligned}$$

and:

$$0 < \lim_{h\to 0}\rho(h) = \frac{1}{24}\lim_{h\to 0}\{[(j+2)h - x_0]^3 - 4[(j+1)h - x_0]^3\} = \frac{x_0^3}{8}$$

Finally $0 < x_0 < 1$,implies

$$\frac{1}{24} < \lim_{h\to 0}\rho(h) < \frac{1}{8}$$

■

Lemma 41 *For $h \to 0$, it holds*

$$\begin{aligned} 0 &< \varphi(h) < \eta(h) < \frac{1}{8}, \\ 0 &< \lambda(h) < \eta(h) < \frac{1}{8} \\ 0 &< \varphi(h) < \rho(h) < \frac{1}{8} \\ 0 &< \lambda(h) < \rho(h) < \frac{1}{8}, \end{aligned} \tag{3.12}$$

$$\frac{1}{12} < \varphi(h) + \eta(h) + \lambda(h) + \rho(h) < \frac{1}{3}. \tag{3.13}$$

Proof. The relation (3.12) and (3.13) are immediate consequences of the previous Lemma.

■

We show that

$$s_y(x) = \sum_{i=0}^{n+1} c_i S(\frac{x}{h} - i), \tag{3.14}$$

can be determinate to be an approximate solution of the problem (3.1). We impose the conditions:

$$Ls_y(x) = r(x),\ 0 < x < 1 \tag{3.15}$$

$$s_y(a) = A, s_y(b) = B,\ 0 < a < b < 1. \tag{3.16}$$

Theorem 42 *(Daniel N.Pop) If the problem (3.1) has a unique solution, then there exits and it is unique a function $s_y(x)$ which verifies (3.15) and (3.16) on mesh points*

$$x_j = jh, j = 1, 2, 3, ...n.$$

Proof. We suppose that $a \in [x_j, x_{j+1}]$. Since $x_j \neq 0$, only for $x_{j-2} < x < x_{j+2}$

then: $S(\frac{a}{h} - i) \neq 0$, only for $i = j-1, j, j+1, j+2$ and

$$A = c_{j-1}S(\frac{a}{h} - j + 1) + c_jS(\frac{a}{h} - j) + c_{j+1}S(\frac{a}{h} - j - 1) + c_{j+2}S(\frac{a}{h} - j - 2).$$

Let $\alpha := S(\frac{a}{h} - j + 1); \beta := S(\frac{a}{h} - j); \gamma := S(\frac{a}{h} - j - 1); \delta := S(\frac{a}{h} - j - 2)$ then

$$c_{j-1} \cdot \alpha + c_j \cdot \beta + c_{j+1} \cdot \gamma + c_{j+2} \cdot \delta = A \tag{3.17}$$

Since $a < b$, then $b \in [x_{j+m}, x_{j+m+1}], j = 1, 2, ..., n, m = 1, 2, ...n - j$ and $S(\frac{b}{h} - i) \neq 0$ only for $i = j + m - 1, j + m, j + m + 1, j + m + 2$.

Also

$$\begin{aligned} B &= s_y(b) = \\ &= c_{j+m-1}S(\frac{b}{h} - j - m + 1) + c_{j+m}S(\frac{b}{h} - j - m) + c_{j+m+1}S(\frac{b}{h} - j - m - 1) + \\ &\quad + c_{j+m+1}S(\frac{b}{h} - j - m - 2). \end{aligned}$$

Let $\mu := S(\frac{b}{h} - j - m + 1); \varepsilon := S(\frac{b}{h} - j - m); \tau := S(\frac{b}{h} - j - m - 1); \xi := S(\frac{b}{h} - j - m - 2)$ then

$$c_{j+m-1} \cdot \mu + c_{j+m} \cdot \varepsilon + c_{j+m+1} \cdot \tau + c_{j+m+2} \cdot \xi = B \tag{3.18}$$

We impose the conditions

$$Ls_y(x_j) = r_j; j = 1, 2, ...n.$$

Using (3.4) and (3.5) we have

$$Ls_y(x_j) = -\sum_{i=0}^{n+1} c_i[\frac{1}{h^2}S"(j - i) - q_jS(j - i)] = r_j; j = 1, 2, ...n.$$

Because $s_i(x_j) \neq 0$ and $s"_i(x_j) \neq 0$ only for $i = j-1, j, j+1$, using **Lemma 11** we obtain

$$\begin{aligned} Ls_y(x_j) &= -\frac{1}{h^2}S"(0)[-\frac{1}{2}c_{j-1} + c_j - \frac{1}{2}c_{j+1}] + \\ +q_jS(0)[\frac{1}{4}c_{j-1} + c_j + \frac{1}{4}c_{j+1}] &= r_j; j = 1, 2, ..., n. \end{aligned}$$

Relations (3.6) and (3.7) yield

$$Ls_y(x_j) = \frac{1}{4}c_{j-1}(\frac{q_j}{6} - \frac{1}{h^2}) + \frac{1}{2}c_j(\frac{q_j}{3} + \frac{1}{h^2}) + c_{j+1}(\frac{q_j}{6} - \frac{1}{h^2}) = r_j, j = 1, 2, ..n. \quad (3.19)$$

Because $q(x) < 0$, for any $0 < x < 1$, then

$$\frac{1}{4}(\frac{q_j}{6} - \frac{1}{h^2}) < 0,\ j = 0, 1, ..., n+1$$

we may divide the relation (3.19) by

$$\frac{1}{4}(\frac{q_j}{6} - \frac{1}{h^2})$$

and let

$$\begin{aligned} p_j &: = \frac{24 \cdot h^2}{h^2 \cdot q_j - 6}r_j; \\ t_j &: = \frac{4(h^2 \cdot q_j + 3)}{h^2 q_j - 6}, \end{aligned}$$

then the relation (3.19) has the form

$$c_{j-1} + t_j \cdot c_j + c_{j+1} = p_j. \quad (3.20)$$

We also observe that

$$\lim_{h \to 0} t_j = -2. \quad (3.21)$$

The relations (3.20), (3.18), (3.17) form a linear system of $(n+2)$ equations with $(n+2)$ unknowns $c_0, c_1, ..., c_{n+1}$:

$$\begin{cases} c_0 + t_1 c_1 + c_2 = p_0 \\ c_1 + t_2 c_2 + c_3 = p_1 \\ \dots \\ c_{j-1} + t_j c_j + c_{j+1} = p_j \\ \alpha c_{j-1} + \beta c_j + \gamma c_{j+1} + \delta c_{j+2} = A \\ c_j + t_{j+1} c_{j+1} + c_{j+2} = p_{j+1} \\ \dots \\ c_{j+m-1} + t_{j+m} c_{j+m} + c_{j+m+1} = p_{j+m} \\ \mu c_{j+m-1} + \varepsilon c_{j+m} + \tau c_{j+m+1} + \xi c_{j+m+2} = B \\ c_{j+m} + t_{j+m+1} c_{j+m+1} + c_{j+m+2} = p_{j+m+1} \\ \dots \\ c_{n-1} + t_n c_n + c_{n+1} = p_n \end{cases}$$

The system matrix is a band matrix with at most four nonzero diagonals. We denote this matrix with C and his determinant with $det\ C$.If we develop $det\ C$ after the columns $0, 1, 2, ..., j-2, j+m+3, ..., n, n+1$ we obtain:

$$\det\ C = \begin{vmatrix} 1 & t_j & 1 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ \alpha & \beta & \gamma & \delta & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & 1 & t_{j+1} & 1 & 0 & ... & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & 1 & t_{j+m} & 1 & 0 \\ ... & ... & ... & ... & ... & ... & \mu & \varepsilon & \tau & \xi \\ ... & ... & ... & ... & ... & ... & 0 & 1 & t_{j+m+1} & 1 \end{vmatrix}$$

If in **Lemma 12** we set $x_0 := a; x_0 := b$ then applying **Lemma 13** it follows that $\alpha, \beta, \gamma, \delta$ are nonzero. In $\det C$ using the properties of determinants, we obtain

$$\det\ C = \begin{vmatrix} c & d & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 1 & t_{j+1} & 1 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & 1 & t_{j+2} & 1 & 0 & ... & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & 1 & t_{j+m-1} & 1 & 0 \\ ... & ... & ... & ... & ... & ... & 0 & 1 & t_{j+m} & 1 \\ ... & ... & ... & ... & ... & ... & 0 & 0 & e & f \end{vmatrix}$$

where

$$\begin{aligned} c &: = \beta - \alpha t_j - \delta; \\ d &: = \gamma - \alpha - \delta t_{j+1}; \\ e &: = \varepsilon - \xi - \mu t_{j+m}; \\ f &: = \tau - \xi t_{j+m+1} - \mu \end{aligned}$$

From (3.21) and **Lemma 12** we have

$$\begin{aligned} \lim_{h\to 0} c &= \lim_{h\to 0}(\gamma - \alpha - \delta t_{j+1}) = \frac{a^3}{6}, \forall a \in (0,1), \\ \lim_{h\to 0} d &= \lim_{h\to 0}(\beta - \alpha t_j - \delta) = \frac{a^3}{6}, \forall a \in (0,1), \\ \lim_{h\to 0} e &= \lim_{h\to 0}(\varepsilon - \xi - \mu t_{j+m}) = \frac{b^3}{6}, \forall b \in (0,1), \\ \lim_{h\to 0} f &= \lim_{h\to 0}(\tau - \xi t_{j+m+1} - \mu) = \frac{b^3}{6}, \forall b \in (0,1). \end{aligned}$$

Let

$$\begin{aligned} x &: = \frac{\beta - \alpha t_j - \delta}{\gamma - \alpha - \delta t_{j+1}}, \\ z &: = \frac{\tau - \xi t_{j+m+1} - \mu}{e := \varepsilon - \xi - \mu t_{j+m}} \end{aligned}$$

then

$$\det\ C = c \cdot e \begin{vmatrix} x & 1 & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 1 & t_{j+1} & 1 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & 1 & t_{j+2} & 1 & 0 & ... & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & 1 & t_{j+m-1} & 1 & 0 \\ ... & ... & ... & ... & ... & ... & 0 & 1 & t_{j+m} & 1 \\ ... & ... & ... & ... & ... & ... & 0 & 0 & 1 & z \end{vmatrix}$$

But,

$$\lim_{h\to 0} x(h) = \lim_{h\to 0} z(h) = 1$$

and from (3.21) we have:

$$\lim_{h\to 0} |t_j(h)| = 2.$$

Let

$$D := \begin{bmatrix} x & 1 & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 1 & t_{j+1} & 1 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & 1 & t_{j+2} & 1 & 0 & ... & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & 1 & t_{j+m-1} & 1 & 0 \\ ... & ... & ... & ... & ... & ... & 0 & 1 & t_{j+m} & 1 \\ ... & ... & ... & ... & ... & ... & 0 & 0 & 1 & z \end{bmatrix}.$$

The matrix D is symmetrical, and for $h \to 0$ is diagonal dominant, therefore is non-singular. Then $\det\ D \neq 0, \det\ C \neq 0$ and the system has a unique solution. ■

3.2 Numerical Results

3.2.1 Example

We shall approximate the solution of following boundary value problem

$$\begin{aligned} -Z\,''(t) - 243Z(t) &= t; 0 < t < 1, \\ Z(\frac{\pi}{6}) &= \frac{\sin(\frac{3\sqrt[2]{3}\pi}{2}) - \frac{1}{6}\pi\sin(9\sqrt[2]{3})}{243\sin(9\sqrt[2]{3})} \\ Z(\frac{\pi}{4}) &= \frac{\sin(\frac{9\sqrt[2]{3}\pi}{4}) - \frac{1}{4}\pi\sin(9\sqrt[2]{3})}{243\sin(9\sqrt[2]{3})} \end{aligned} \tag{3.22}$$

The problem (3.22) has a unique solution

$$Z(t) = \frac{\sin(9\sqrt[2]{3}t) - t\sin(9\sqrt[2]{3})}{243\sin(9\sqrt[2]{3})}.$$

We used `Maple 14` to solve the problem exactly and to approximate the solution, the mesh considered was uniform, with

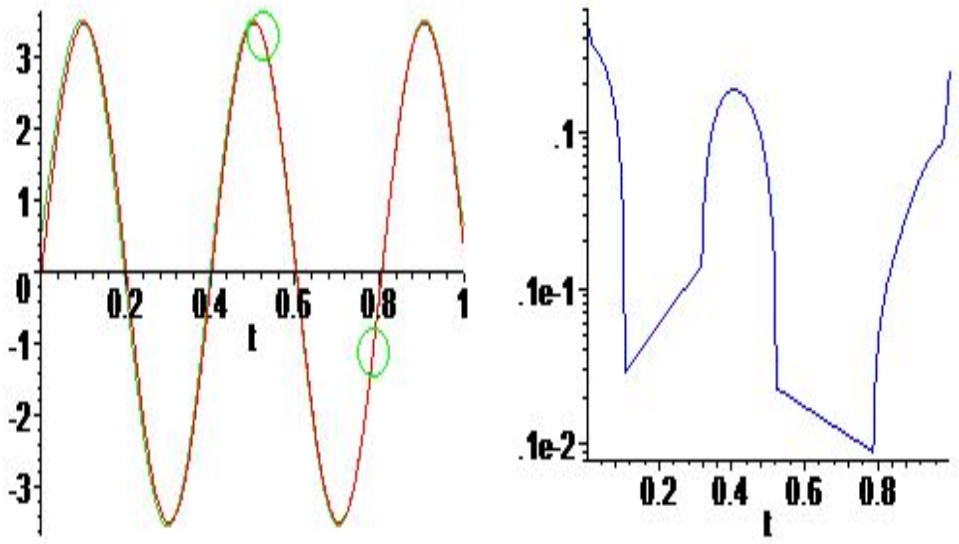
$$h = \frac{1}{52}$$

Since

$$\int_0^1 |q(t)dt| > 4$$

using unconjugated criteria given by Lyapunov (1893) the problem (3.22) has an oscillatory solution.

Both solution are represented on same graph in figure 3.2(a).The graph error in a semilogarithmic scale is given in figure 3.2(b).



(a) Exact and Approx solution

(b) Errors

Figure 3.2: Numerical example with uniform mesh

3.2.2 Coefficients of approximation

0.05875642949	0.01478815919	0.02952783767	0.04419585544
0.05875642949	0.07317381026	0.08741229213	0.10143662548
0.11521015320	0.12869854210	0.14186608740	0.15467756260
0.16709789740	0.17909216930	0.19062560250	0.20166362270
0.21217182700	0.22211602950	0.23146224290	0.23146224290
0.24017673290	0.24822599740	0.25557677910	0.26219610900
0.26805128590	0.27310991060	0.27733988500	0.28070942340
0.28318708480	0.28474176000	0.28534270740	0.28495953220
0.28356223750	0.28112120050	0.27760721550	0.27299148300
0.26724561130	0.26034167640	0.25225216060	0.24295003250
0.23240871200	0.22060210090	0.20750458460	0.19309105680
0.17733688270	0.16021798320	0.14171079360	0.12179225390
0.10043989140	0.07763175011	0.05334645858	0.02756317540

Chapter 4

Linear form of boundary values problems

4.1 Formulating the problem

We consider the linear boundary value problem with conditions inside the interval $[0,1]$ (PVPL)

$$\begin{cases} -y''(x)+q(x)y(x)=r(x), & x\in[0,1] \\ y(a)=\alpha,\ y(b)=\beta, & a,b\in(0,1), \end{cases}, \tag{4.1}$$

where $q(x),\ r(x)\in C[0,1];\ y\in C^1[0,1],\ a,b,\ \alpha,\beta\in\mathbb{R}$.

If the solution of two points boundary value problem (BVPL)

$$\begin{cases} -y''(x)+q(x)y(x)=r(x), & x\in[a,b] \\ y(a)=\alpha,\ y(b)=\beta, \end{cases}, \tag{4.2}$$

exists and it is unique, then the requirement $y(x)\in C^1[0,1]$ assures the existence and the uniqueness exact solution of the problem (4.2).

We present three approximation methods:

1. Global method with cubic B-spline function,

2. C.C method,

3. Combined method *Runge-Kutta* and B-spline functions of degree k.

4.2 Global method with cubic B-spline function

Consider the linear boundary value problem (PVPL) given by (4.1) and further consider that occurs the conditions of existence and uniqueness of exact solutions $y(x)$. We recall some properties of orthogonal *Legendre* [44] polynomials.

Definition 43 *We define the orthogonal* Legendre *polynomials as follows*

$$P_n(y) = \frac{1}{2^n n!}\frac{d^n}{dy^n}(y^2-1)^n; n = 0, 1, 2, ...$$

Proposition 44 *The roots of orthogonal* Legendre *polynomials are contained in* $(-1, 1)$.

The graph of *Legendre* polynomials of degree $k = 1, 2, 3, 4, 5$ is depicted in the figure 4.1

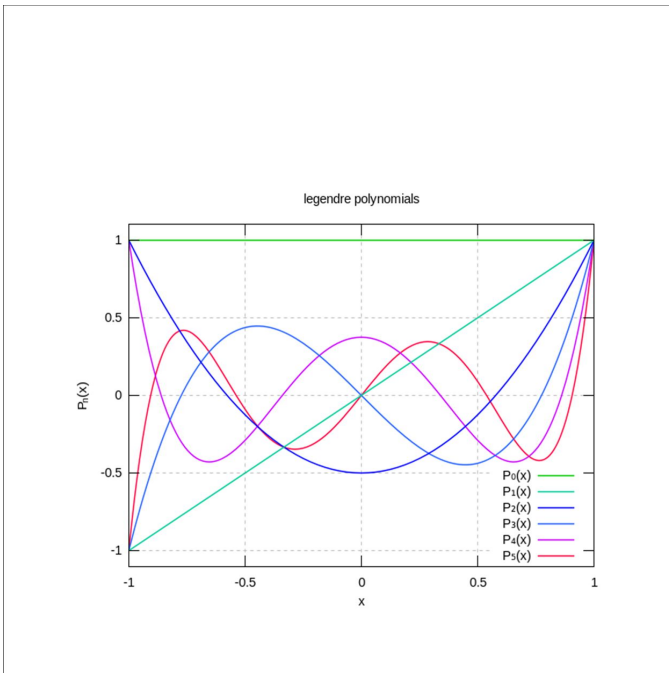


Figure 4.1: Legendre orthogonal polynomials

Using the affine transformation

$$y := 2x - 1$$

we obtain the orthogonal *Legendre* polynomials on $(0,1)$ having all roots within $(0,1)$

$$\bar{P}_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[x^n(x-1)^n];\ n = 0,1,2,...$$

4.2.1 Construction of collocation points and the base of cubic B-spline functions

We construct the partition of $[0,1]$ like in (1.53), the step-size (1.54) and collocation points (1.57).

Collocation points form the following matrix

$$\Upsilon = \begin{bmatrix} h_0\rho_1 & h_0\rho_2 & \dots & h_0\rho_k \\ x_1 + h_1\rho_1 & x_1 + h_1\rho_2 & \dots & x_1 + h_1\rho_k \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1} + h_{n-1}\rho_1 & x_{n-1} + h_{n-1}\rho_2 & \dots & x_{n-1} + h_{n-1}\rho_k \end{bmatrix}$$

We note that the points x_i of grid (1.53) are not collocation points, so you must insert their heads interval, i.e. 0 and 1, and inside a, b and so we get $nk+4$ points .

We renumber the collocation points such that

$$\xi_0 = 0,\ \xi_1 = h_0\rho_1, \xi_2 = h_0\rho_2, ...\xi_{N+1} = x_{n-1} + h_{n-1}\rho_k,\ \xi_{N+2} = 1,$$

where $N = nk+2$. Therefore, the partition of $[0,1]$ becomes

$$\Delta := \left\{0 = \xi_0 < \xi_1 < ... < a... < b... < \xi_{N+2} = 1\right\}.$$

We augment the above partition Δ [72] to form

$$\overline{\Delta} : \xi_{-2} < \xi_{-1} < \xi_0 = 0 < \xi_1 < ... < \xi_{N+2} = 1 < \xi_{N+3} < \xi_{N+4}, \tag{4.3}$$

where

$$\begin{aligned} \xi_l &= a; \quad \xi_{l+p} = b;\ 0 < l < N+1;\ 1 < l+p < N+2, \\ \xi_{-1} - \xi_{-2} &= \xi_0 - \xi_{-1} = \xi_1 - \xi_0, \\ \xi_{N+4} - \xi_{N+3} &= \xi_{N+3} - \xi_{N+2} = \xi_{N+2} - \xi_{N+1}. \end{aligned}$$

To simplify the presentation, we use the abbreviations

$$q_i := q(\xi_i)\ ;\ h_i := \xi_{i+1} - \xi_i;\ h := \max_{0\leq\ i\leq\ n} h_i;\ \underline{h} := \min_{0\leq\ i\leq\ n} h_i.$$

Definition 45 *For* $\forall\ x \in [0,1]$ *and* $\forall\ 0 \leq i \leq N+1$, *the cubic B-spline functions with the five knots :* ξ_i, ξ_{i+1}, ξ_{i+2}, ξ_{i+3}, ξ_{i+4} *are given by*

$$\begin{cases} B_{i,0}(x) = \begin{cases} 1 & if \qquad \xi_i \leq x < \xi_{i+1} \\ 0 & if \quad x \in [0,1] - [\xi_i, \xi_{i+1}) \end{cases} \\ B_{i,3}(x) = \frac{x-\xi_{\ i-2}}{h_{i-2}+h_{i-1}+h_i} B_{i,2}(x) + \frac{\xi_{\ i+2}-x}{h_{i+1}+h_i+h_{i-1}} B_{i+1,2}(x) \\ \qquad if\ \ \xi_i < \xi_{i+3} \quad and\ \xi_{i+1} < \xi_{i+4}\ . \end{cases} \tag{4.4}$$

4.2.2 The existence and uniqueness of approximate solution

We wish to find an approximate solution of the boundary value problem (PVPL) (4.1) in the following form

$$u_{\overline{\Delta}}(x) = \sum_{i=\ 0}^{N+2} c_i B_{i,3}(x), \tag{4.5}$$

where $B_{i,3}(x)$ is a cubic B-spline function with knots $\{\xi_{i+k}\}_{k\ =\ -2}^{2}$.

We impose the conditions:

(c1) The approximate solution $u_{\overline{\Delta}}(x)$ given by (4.5) verifies the differential equation of (4.1) at

$$\overline{\Delta} := \{\xi_j\ ,\ j = 1, ..., N+1,\ j \neq l,\ j \neq l+p\}.$$

(**c2**) The approximate solution $u_{\overline{\Delta}}(x)$ (4.5) verifies

$$u_{\overline{\Delta}}(\xi_l) = \alpha,\ u_{\overline{\Delta}}(\xi_{l+p}) = \beta.$$

We recall that $a = \xi_l$ and $b = \xi_{l+p}$.

Conditions **(c1)** and **(c2)** yield to a linear system with $N+3$ equations and $N+3$ unknows $c_i,\ i = 0, ..., N+2$:

$$A \cdot c = \gamma. \tag{4.6}$$

We denote by

$$f_i(x) := B''_{i,3}(x) - q(x)B_{i,3}(x), i = 0, 1, ...N,$$

then

$$A = \begin{bmatrix} f_i(\xi_j);\ i \in \{0, 1, 2..., N+2\} \quad j \in \{1, 2, ...N+1\} \backslash \{l.l+p\} \\ B_{i,3}(\xi_l); i = l-1, l, l+1 \\ B_{i,3}(\xi_{l+p}); i = l+p-1, l+p, l+p+1 \end{bmatrix}. \tag{4.7}$$

The system matrix A is tridiagonal with 3 nonzero elements on each row. The right hand side of (4.6) is

$$\gamma = [r(\xi_1), ...r(\xi_{l-1}), \gamma_1, r\left(\xi_{l+1}\right), ...r(\xi_{l+p-1}), \gamma_2, r(\xi_{l+p+1}), ...r(\xi_{n+2})].$$

Lemma 46 *[59, pag 23]For every $l > 0,\ x \in [0,1],\quad B_{i,l}(x) \in C^1[0,1]$ occurs*

$$B'_{i,l}(x) = l\ [\frac{B_{i,l-1}(x)}{\xi_{i+l} - \xi_i} - \frac{B_{i+1,l-1}(x)}{\xi_{i+l+1} - \xi_{i+1}}]. \tag{4.8}$$

Lemma 47 ***(Daniel N.Pop [53])*** *For $\forall\ x \in [0,1]$ and $B_{i,3}(x) \in C^2[0,1]\ 0 \leq i \leq N+2$ then*

$$\begin{aligned} B''_{i,3}(x)\ &= 3![\frac{B_{i,1}(x)}{(h_i + h_{i+1} + h_{i+2})(h_{i+1} + h_i)} - \\ &- \frac{B_{i+1,1}(x)(h_{i+2} + 2h_{i+1} + 2h_{i+2} + h_{i+3})}{(h_{i+2} + h_{i+1})(h_{i+2} + h_{i+1} + h_i)(h_{i+3} + h_{i+2} + h_{i+1})} + \\ &+ \frac{B_{i+2,1}(x)}{(h_{i+3} + h_{i+2} + h_{i+1})(h_{i+2} + h_{i+3})}]. \end{aligned} \tag{4.9}$$

Proof. For $l = 3$ we get from (4.8)

$$B'_{i,3}(x) = 3[\frac{B_{i,2}(x)}{h_{i+2} + h_{i+1} + h_i} - \frac{B_{i+1,2}(x)}{(h_{i+3} + h_{i+2} + h_{i+1}}],$$

then

$$B''_{i,3}(x) = 3\left[\frac{B'_{i,2}(x)}{h_{i+2} + h_{i+1} + h_i} - \frac{B'_{i+1,2}(x)}{h_{i+3} + h_{i+2} + h_{i+1}}\right]. \tag{4.10}$$

Using again (4.8) for $l = 2$ it results

$$B\ '_{i,2}(x) = 2\left[\frac{B_{i,1}(x)}{h_{i+1} + h_i} - \frac{B_{i+1,1}(x)}{h_{i+2} + h_{i+1}}\right], \tag{4.11}$$

$$B'_{i+1,2}(x) = 2\left[\frac{B_{i+1,1}(x)}{h_{i+2}+h_{i+1}} - \frac{B_{i+2,1}(x)}{h_{i+3}+h_{i+2}}\right]. \tag{4.12}$$

Replacing (4.11) and (4.12) in (4.10) we get (4.9). ■

Theorem 48 ***(Daniel N.Pop)[53]*** *If for every $x \in [0,1]$*

$$q(x) < K \leq 0, \tag{4.13}$$

then the matrix A given by (4.7) is nonsingular.

Proof. In the following we denote by

$$f_i(x) := B''_{i,3}(x) - K \cdot B_{i,3}(x),\ \forall\ i = 0, 1, ...N+2. \tag{4.14}$$

Since $B_{i,3}(x) > 0$ for every $x \in (\xi_{j+1}, \xi_{j+3})$ [59, pp: 18-20] resulting

$$\begin{aligned} B_{i,3}(\xi_{j+1}) &> 0,\ B_{i,3}(\xi_{j+2}) > 0,\ B_{i,3}(\xi_{j+3}) > 0. \\ B_{i,3}(\xi_{l+p}) &> 0; i = l+p-1, l+p, l+p+1. \end{aligned} \tag{4.15}$$

From (4.4) and (4.9) we obtained the following inequalities

$$\begin{aligned} B_{i,3}(\xi_{j+2}) &> B_{i,3}(\xi_{j+1}) + B_{i,3}(\xi_{j+3}),\ j = -2, -1, 0, 1, ...N, \\ B''_{i,3}(\xi_{j+1}) &= \frac{3!}{(h_{j+2}+h_{j+1}+h_j)(h_{j+1}+h_j)} > 0, \\ B''_{i,3}(\xi_j) &= \frac{3!(h_j + 2h_{j+1} + 2h_{j+2} + h_{j+3)}}{(h_{j+2}+h_{j+1}+h_j)(h_{j+2}+h_{j+1})(h_{j+3}+h_{j+2}+h_{j+1})} > 0, \\ B''_{i,3}(\xi_{j+3}) &= \frac{3!}{(h_{j+3}+h_{j+2}+h_{j+1})(h_{j+2}+h_{j+3})} > 0, \\ B''_{i,3}(\xi_{j+2}) &> B''_{i,3}(\xi_{j+1}) + B''_{i,3}(\xi_{j+3}),\ j = -2, -1, 0, 1, ...N. \end{aligned} \tag{4.16}$$

Using again (4.14) and (4.16) we get

$$f_i(\xi_{j+2}) > f_i(\xi_{j+1}) + f_i(\xi_{j+3}),\ \forall j = -2, -1, 0, 1, ...N.$$

From the above relationship we deduce that collocation matrix A given by (4.7) is strictly diagonally dominant, so nonsingular and $\det A \neq 0$.

So the coefficients c_i from the relation (4.5) can be determined uniquely. ■

To solve the system (4.6) we use *Crout*'s factorization algorithm for linear tridiagonal systems [17, page: 353], this algorithm requires only $(5N+11)$ multiplication/division and $(3N+6)$ addition / substraction and therefore has considerable computational advantages compared with methods, where the matrix are not tridiagonal and irreducible, especially for large values of N.

Conclusion 49 *In terms of costs (time U.C and internal memory), effectiveness of this method with cubic B-spline methods in comparison with a situation arises in pseudo-spectral oscillatory solutions, as this that collocation matrix is dense.*

4.2.3 Error study

Below we expose some considerations on the study of the error.

Theorem 50 *[53] If $y(x) \in C^1[0,1]$ is the exact solution of boundary value problem (PVPL), then there exits a unique cubic B-spline function*

$$u_{\overline{\Delta}}(x) \in S(\overline{\Delta}),$$

where

$S\left(\overline{\Delta}\right) = \{p(x) \in C^2[0,1]/p(x)$ *is a polynomial function of degree three for each subintervals*: $[\xi_{i-2},\ \xi_{i+2}];\ 0 \leq i \leq N+2\}$,*which verifies*

$$\max_{\xi_i \leq x \leq \xi_{i+4}} |y(x) - u_{\overline{\Delta}}(x))| \leq K \cdot H^2 \cdot \left\|y^{(2)}\right\|_{[\xi_i, \xi_{i+4}]},\ i = -2,-1,0,1,...,N, \tag{4.17}$$

where

$$H := \max\{h_i, h_{i+1}, h_{i+2}, h_{i+3}\},$$

and K is real constant independent of $\overline{\Delta}$ given by (4.3).

Proof. Using the results of *U. Ascher* [5] and *C.deBoor* [15], convergence of the method presented in the previous paragraph takes place for every division

$$\xi_j \in \overline{\Delta},\ j \in \{-2,-1,0,1,...,N+4\} - \{l, l+p\}$$

and more

$$\left|y^{(\upsilon)}(\xi_j) - u^{(\upsilon)}_{\overline{\Delta}}(\xi_j)\right| = \mathbb{O}(H^{2k}), \forall\ 0 \leq \upsilon \leq 2,\ \forall\ k \geq 2, \tag{4.18}$$

where υ means order derivative index and the index k is the degree of *Legendre* polynomial.

More in points $\xi_l = a,\ \xi_{l+p} = b,$ according [6, page 222] occurs

$$\begin{aligned} |y(\xi_l) - u_{\overline{\Delta}}(\xi_l)|_{[\xi_{l-2},\xi_{l+2}]} &\leq K_1 \cdot H^2 \cdot \left\|y^{(2)}\right\|_{[\xi_{l-2},\xi_{l+2}]}, \\ \left|y(\xi_{l+p}) - u_{\overline{\Delta}}(\xi_{l+p})\right|_{[\xi_{l+p-2},\xi_{l+p+2}]} &\leq K_2 \cdot H^2 \cdot \left\|y^{(2)}\right\|_{[\xi_{l+p-2},\xi_{l+p+2}]}, \end{aligned} \tag{4.19}$$

where K_1, K_2 are real constants independent of division $\overline{\Delta}$ given by (4.3) and $y(x) \in C^1[0,1]$ the exact solution of the bylocal linear problem (PVPL). Using the relations (4.18) and (4.19), that the proposed method is convergent with order of convergence $\mathbb{O}(H^2)$. ■

4.2.4 Numerical results

In the works ([54], [55]) we prove the effectiveness of the method presented in above paragraph for the case $q(x) < K \leq 0,$ by two examples: one with oscillatory solution and other with non-oscillatory solution. For implementation of costs (run times and internal memory) we used a pair of commands `Maple 13` ([3, page: 236], [34]) `profile-show-profile`, and for error control the command `plots[log-plot]`.

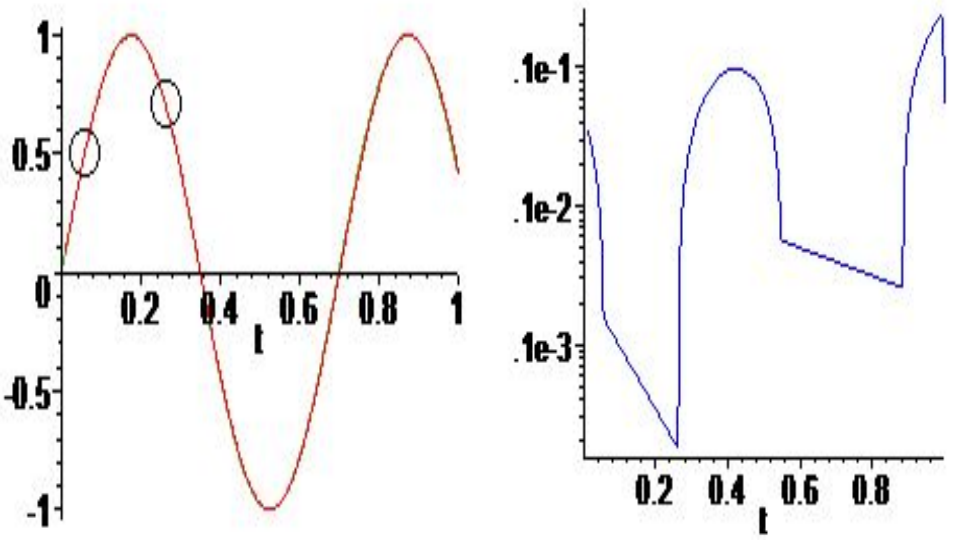
To make a comparative study of computational costs and errors we took two different divisions of the interval $[0,1]$ so : $N \in \{10, 36\}$.

1. Greengard and Rokhlin problem (oscillatory solution) We solve the bylocal problem (1.76) for $k = 9$ using *cubic B-spline* functions on two meshes $\overline{\Delta}$ of the interval $(0,1)$ given by (4.3). We obtain the following results

N	Run times	Internal memory	Tolerance
10	2.955	64944668	$0.5e^{-1}$
36	12.668	132688208	e^{-2}

.

For $N = 36$ we represented approximate solution in Figure 4.2(a) and errors in logarithmic scale in Figure 4.2(b)



(a) Approximate solution (b) Errors in logarithmic scale

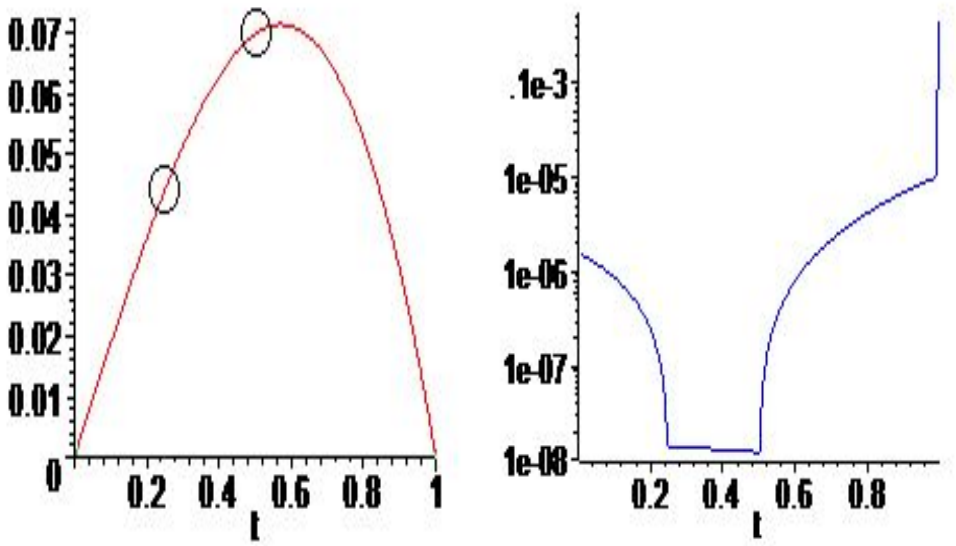
Figure 4.2: Greengard-Rokhlin problem, non-uniform mesh, Cubic B-spline

2. Burden and Faires problem(non-oscillatory solution) We solve the two-points boundary value linear problem (1.75) using *cubic B-spline* functions on two meshes $\overline{\Delta}$ of the interval $(0, 1)$ given by (4.3). We obtain the following results

N	Run times	Internal memory	Tolerance
10	2.724	68700548	$0.1e^{-25}$
36	11818	235934564	$0.1e^{-40}$

.

For $N = 36$ we represented approximate solutions in Figure (4.3(a)) and errors in logarithmic scale in Figure (4.3(b)).



(a) Approximate solution (b) Errors in logarithmic scale

Figure 4.3: Burden-Faires problem, non-uniform mesh, Cubic B-spline

4.3 Pseudo-spectral (C.C) method

Formulating the problem

We consider the linear boundary value problem (PVPL) with conditions inside the interval $[0,1]$. This problem will be solved numerically by dividing them into three two-points boundary value problems as follows

- (BVPL1)

$$\begin{cases} -y\,''(x)+q(x)y(x)=r(x),\ a\le x\le b, \\ y(a)=\alpha,\ y(b)=\beta \\ a,b,\alpha,\beta\in\mathbb{R}, \end{cases}, \tag{4.20}$$

- (BVPL2)

$$\begin{cases} -y\,''(x)+q(x)y(x)=r(x),\ 0\le x\le a, \\ y(a)=\alpha,\ y'(a)=\gamma,\ a,\alpha,\gamma\in\mathbb{R}, \end{cases}, \tag{4.21}$$

- (BVPL3)

$$\begin{cases} -y\,''(x)+q(x)y(x)=r(x),\ b\le x\le 1, \\ y(b)=\beta,\ y'(b)=\delta,\ b,\beta,\delta\in\mathbb{R}. \end{cases} \tag{4.22}$$

We determined the approximate solutions of the problems (4.20), (4.21) and (4.22) using the $C.C$ method [29, page: 36-38].

Choosing the mesh of the interval $[0,1]$ and define the *Tchebychev'* polynomials of first kind on $[a,b]$.

Definition 51 *If* $T_k(x)=\cos(k\arccos(x))$ $k=0,1,2,..N$ *are the* Tchebychev' *polynomials* of first kind *on* $[-1,1]$, *we define the* Tchebychev' *polynomials* of first kind *on* $[a,b]$ *so*

$$T_k(\frac{(b-a)x+(a+b)}{2}) := \cos(k \arccos \frac{(b-a)x+(a+b)}{2}) \tag{4.23}$$

We choose to solve numerically the bylocal linear problem (PVPL) the following mesh of the interval $[0,1]$

1. on $[0,a]$

$$x_j^1 = \frac{a}{2}(-\cos\frac{\pi j}{N}+1),\ j=0,1,...,N, \tag{4.24}$$

2. on $[a,b]$

$$x_j^2 = \frac{(b-a)\cos\frac{\pi j}{N}+a+b)}{2},\ j=0,1,...,N, \tag{4.25}$$

3. and on $[b,1]$

$$x_j^3 = \frac{(b-1)\cos\frac{\pi j}{N}+1+b)}{2},\ j=0,1,...,N. \tag{4.26}$$

The points of the mesh (4.24) are symmetrical to $\frac{a}{2}$, the points of the mesh(4.25) are symmetrical to $\frac{a+b}{2}$, and those mesh-points (4.26) are symmetrical to $\frac{b+1}{2}$.

In the figure 4.4 was depicted the *Chebychev' polynomials of first kind*

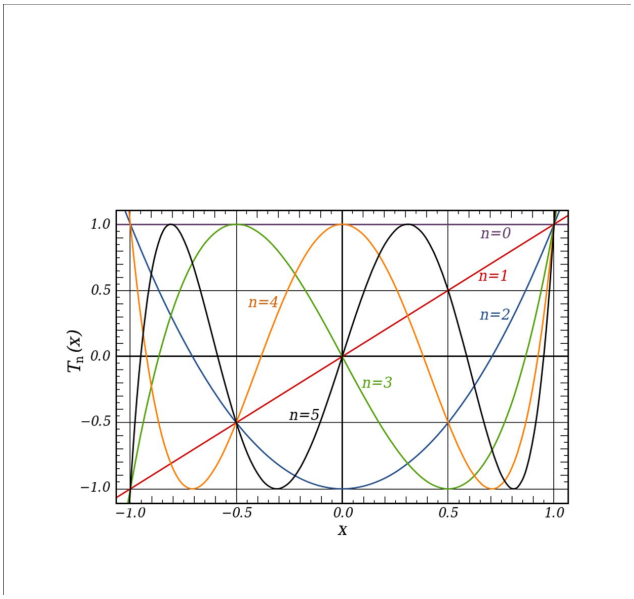


Figure 4.4: Chebychev orthogonal polynomials

Solving numerical two-points boundary value problems (BVPL1), (BVPL2) and (BVPL3).

In the paper [18], *C.Canuto* suggests to find approximate solution $u_N(x)$ using Lagrange interpolation polynomial. We use *C.C method* for determine the approximate solutions of two-points boundary value problems (BVPL1), (BVPL2) and (BVPL3) on collocation points given by (4.24), (4.25) and (4.26) as shown below.

Determination of the unknowns

$$c_j^2 := u_N(x_j^2),\ j = 0, 1, ..., N,$$

consists in solving the following linear algebraic system

$$\begin{cases} L_N(u_N(x_j^2)) - r((x_j^2)) = 0;\ j = 1, ..., N-1 \\ u_N(a) = \alpha;\ u_N(b) = \beta, \end{cases}, \tag{4.27}$$

where x_j^2 are given by (4.25) and

$$L_N(x) = \sum_{k=1}^{N} u(x_k) l_k(x), \tag{4.28}$$

is the *Lagrangian* interpolation polynomial.

Determination of the unknowns

$$c_j^1 := u_N(x_j^1),\ j = 0, 1, ..., N,$$

consists in solving the following linear algebraic system

$$\begin{cases} L_N(u_N(x_j^1)) - r((x_j^1)) = 0;\ j = 1, ..., N-1 \\ u_N(a) = \alpha;\ u_N(0) = \eta, \end{cases}, \tag{4.29}$$

where x_j^1 are given by (4.24). We build $u_N(0)$ so

$$u_N(0) = \alpha - a u'_N(a),$$

where the value $u'_N(a)$ is obtained by deriving *Lagrange* interpolation polynomial on $x = a$.

$$u'_N(a) = \sum_{j=0}^{N} \frac{1}{a - x_j}. \tag{4.30}$$

Determination of the unknowns

$$c_j^3 := u_N(x_j^3),\ j = 0, 1, ..., N,$$

consists in solving the following linear algebraic system

$$\begin{cases} L_N(u_N(x_j^3)) - r((x_j^3)) = 0;\ j = 1, ..., N-1 \\ u_N(b) = \beta;\ u_N(1) = \lambda, \end{cases}, \tag{4.31}$$

where x_j^3 are given by (4.26).

We build $u_N(1)$ so

$$u_N(1) = \beta + (1-b)u'_N(b),$$

where the value $u'_N(b)$ is obtained by deriving *Lagrange* interpolation polynomial on $x = b$.

$$u'_N(b) = \sum_{j=0}^{N} \frac{1}{b - x_j}. \tag{4.32}$$

Proposition 52 *If in collocation matrices of the systems (4.27), (4.29), (4.31) remove lines which are obtained from boundary conditions, first and last column we obtain a centrosymmetric matrices, which according ([20], [81]) are nonsingular.*

Using the above proposition, it follows that if we use the method presented can uniquely determine the values:

$$u_N(x_j^1),\ u_N(x_j^2) \text{ and } u_N(x_j^3).$$

4.3.1 Numerical results

The `Maple` code to determinate the approximate solutions of the two-points boundary value problems (1.75) and (1.76) using $C.C$ methods is in [26]. For establish the running costs (run time and internal memory) we set the pair of commands using Maple 14 `profile-show-profile [34]` and for errors in logarithmic scale we use the command `plots[plot-log`].

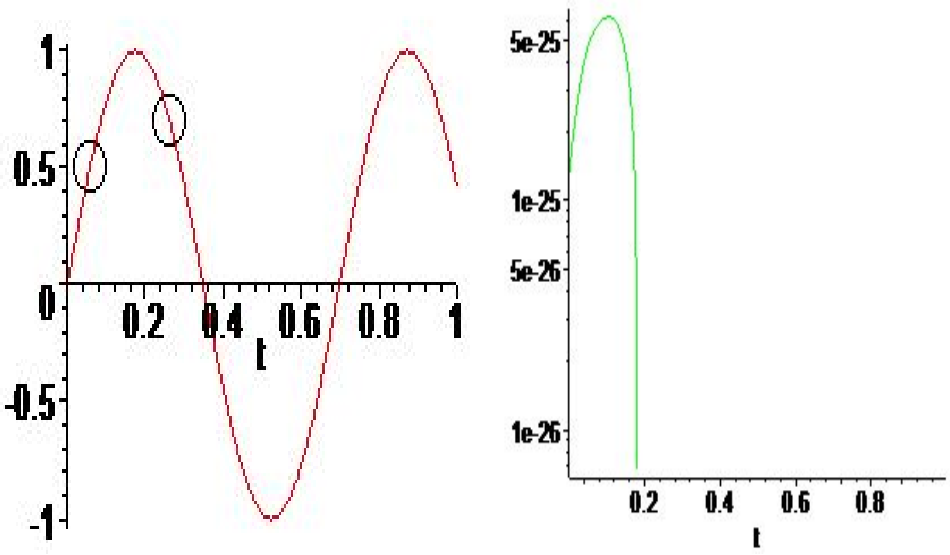
1. Greengard and Rokhlin problem (oscillatory solution)

For $k = 9$ and for different values of division: $N \in \{10, 36\}$ we solve the two-points boundary value problem (1.76) using $C.C$ method on the collocation points of the interval $[0, 1]$ given by (4.24), (4.25) and (4.26).

We got the following results

N	Run times	Internal memory	Tolerance
10	3.925	68943558	$0.1e^{-3}$
36	53.767	997685852	$8e^{-25}$

For $N = 36$ we represented the approximate solution in Figure 4.5(a) and the errors in logarithmic scale in Figure 4.5(b)



(a) Approximate solution (b) Errors in logarithmic scale

Figure 4.5: Greengard-Rokhlin problem, non-uniform-mesh, C.C method

2. Burden and Faires problem

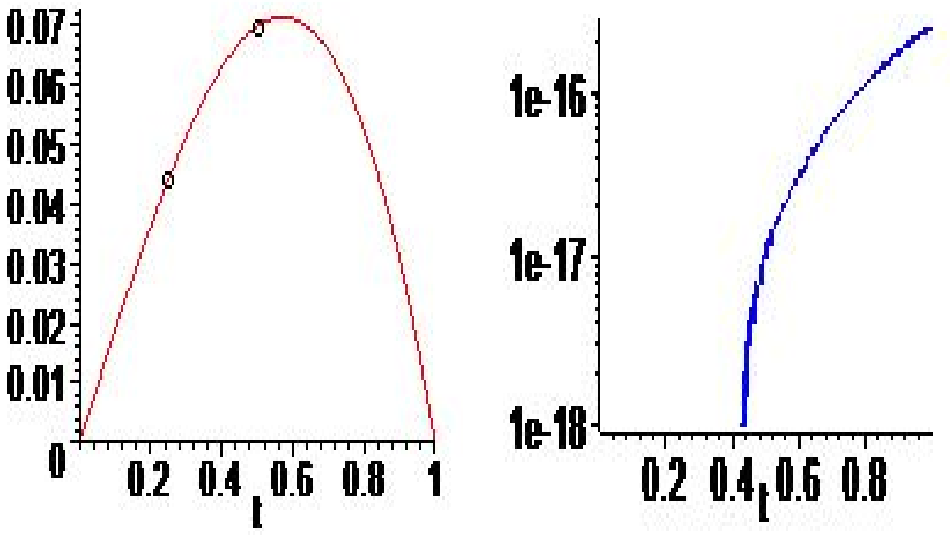
We determined the approximate solution of the problem (1.75) for two values of grid : $N \in \{10, 36\}$ using the collocation points of the interval $[0, 1]$ (4.24), (4.25) and (4.26).

We obtained the following results

N	Run times	Internal memory	Tolerance
10	0.320	8014024	e^{-10}
36	6.469	135614356	e^{-16}

.

For $N = 36$, we represented the approximate solution in Figure (4.6(a)), and the errors in Figure (4.6(b)).



(a) Approximate solution (b) Errors in logarithmic scale

Figure 4.6: Burden-Faires problem, C.C method

3. Reaction diffusion equation

For equation (1.74), we determined an approximate solution using $C.C$ method combined with a pair *Runge-Kutta* method. With conditions inside the interval $(0, 1)$, we obtained the following (BVPL) problem

$$\begin{cases} -y\,'' + 4y(x) = 10e^{-5x\ ^2-2x}; 0 \leq x \leq 1, \\ \qquad y(1/4) = \gamma_1,\ y(1/2) = \gamma_2, \end{cases} \tag{4.33}$$

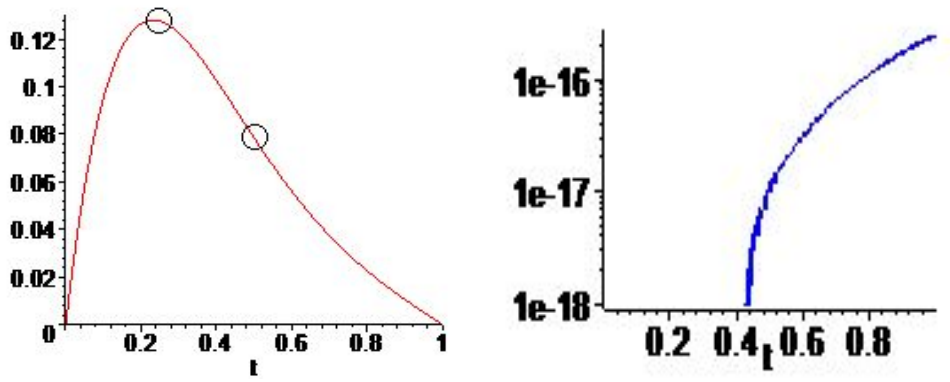
where the values $\gamma_1,\ \gamma_2$ are obtained using Maple 14.

For $N \in \{10, 36\}$, we determined an approximate solution of boundary value problem (4.33), and evaluate the costs (Run times and Internal memory used) and the errors, achieving the results

N	Run times	Internal memory	Tolerance
10	1.402	23586232	e^{-07}
36	13.610	249685652	e^{-13}

.

The approximate solution for $N = 36$ is in Figure 4.7(a), and the error in semi-logarithmic scale in Figure 4.7(b).



(a) Approximate solution (b) Errors in logarithmic scale

Figure 4.7: Reaction diffusion equation C.C method

4.4 Combined Runge-Kutta methods and B-spline functions

Principle method

The two-points boundary values problem (PVPL)(4.1) can be solve by dividing it in three above problems so

- A two-points *Dirichlet* boundary values problem (BVPL)

$$\begin{cases} -y''(x) + q(x)y(x) - r(x) = 0, & x \in (a,b) \\ y(a) = \alpha,\ y(b) = \beta. \end{cases}, \tag{4.34}$$

- Two initial value problems (*Cauchy* problems):

1. (IVP-Left)

$$\begin{cases} y' = z \\ z' = -f(x,y), & x \in [0,a] \\ y(a) = \alpha,\ y'(a) = \gamma. \end{cases}, \tag{4.35}$$

2. and (IVP-Right)

$$\begin{cases} y' = z \\ z' = -f(x,y), & x \in [b,1] \\ y(b) = \beta,\ y'(b) = \delta, \end{cases}, \tag{4.36}$$

where $f(x,y) = -q(x)y(x) + r(x)$.

The values $y'(a)$, $y'(b)$ necessary to determine solutions of the problems (IVP-Left) and (IVP-Right) are determined by differentiating the approximate solution of two-points boundary values problem (4.34) in the points $x = a$ and $x = b$.

For the two-points boundary values problem (BVPL) we use the collocation method with B-splines of order $k+1$, $k \in \mathbb{N}^*$ and to determine the approximate solutions of the problems (IVP-Left) and (IVP-Right) we use *Runge-Kutta* methods. We use division $\overline{\Delta}$

of the interval $[a, b]$ given by (4.3), where the multiplicity of points $a,\ b$ are $k+1$ and inside collocation points have multiplicity k.

We find the approximate solution $u_{\overline{\Delta}}(x)$ in the form

$$u_{\overline{\Delta}}(x) = \sum_{i=0}^{N+2} c_i B_{i,k}(x),$$

where $B_{i,k}(x)$ are B-splines functions of degree k, and real coefficients $c_i \in \mathbb{R}$ are determined by imposing the following conditions:

- **(C1)** the approximate solution $u_{\overline{\Delta}}(x)$ verifies the differential equation given by (4.34) inside the grid $\overline{\Delta}$ (multiplicity k).
- **(C2)** the approximate solution $u_{\overline{\Delta}}(x)$ verifies the conditions

  $$u_{\overline{\Delta}}(a) = \alpha, u_{\overline{\Delta}}(b) = \beta,\ a, b \in (0, 1),$$

  where the points a and b have the multiplicity $k+1$.

The conditions **(C1) +(C2)** lead to a linear system $Ac = \gamma$, with $(N+3)$ equations and $(N+3)$ unknowns.

The collocation matrix is

$$A = [a_{ij}]_{i,\ j=0,\ldots,N+2} = \begin{cases} -B\,''_{i,k}(\xi_j) + q(\xi_j)B_{i,k}(\xi_j), & \text{for } \ j = 1, 2, ..., N+1, \\ B_{i,k}(a), & \text{for } \ j = 0, \\ B_{i,k}(b), & \text{for } \ j = N+2 \end{cases} . \quad (4.37)$$

The right hand size of this system is

$$\gamma = \left[\gamma_1, r(\xi_1), \ldots, r(\xi_{N-1}), \gamma_2\right]^T .$$

We denote by $\overline{u}_{\overline{\Delta}}(x)$ the approximate solution of the problem (PVPL) (4.1), obtained by combined method B-splines and Runge-Kutta methods.

Theorem 53 ***(Daniel N.Pop)*** *[56] Suppose that there exists $p \geq k \geq 2$, such that:*

(a) The linear problem (4.34) is well-posed, in the sense that the collocation matrix A given by (4.37) has a condition number $\kappa_k \equiv cond(A)$, of moderate size ([75]) and

$$q, r \in C^p[a, b];$$

(b) The linear problem (4.34) has a unique solution and $y(x) \in C^p[a, b]$,

(c) The collocation points $\rho_1, \ldots, \rho_k$ given by (1.58) satisfy the orthogonality condition

$$\int_0^1 \Phi(x) \prod_{\ell=1}^{k} (x - \rho_\ell) dx = 0, \qquad \Phi \in \mathbb{P}_{p-k}, \ (p \leq 2k),$$

where $\mathbb{P}_{p-k}$ is the set of polynomials at most degree $p - k$.

Then for

$$h = \max_{i=1,\ldots,N} h_i,$$

sufficiently small our method (Collocation and two Runge-Kutta methods) is stable with constant $C \cdot \kappa_k N$ (where C is real constant) and leads to a unique solution $y(x)$ of two points boundary values problem (4.2). Furthermore at mesh-points $\xi_j \in \overline{\Delta}$ it holds

$$\left| y^{(\nu)}(\xi_j) - \overline{u}_{\overline{\Delta}}^{(\nu)}(\xi_j) \right| = O(h^p), \qquad \nu = 0, 1; \ j = 0, 1, ..., N + 2, \tag{4.38}$$

while, on the other hands , for $\forall x \in [\xi_j, \xi_{j+1}]$, $j = 0, 1 \ldots, N + 1$ occurs

$$\left| y^{(\nu)}(x) - \overline{u}_{\Delta}^{(\nu)}(x) \right| = O(h_j^{k+2-\nu}) + O(h^p), \qquad \nu = 0, 1, \tag{4.39}$$

where $h_j = \left| \xi_{j+1} - \xi_j \right|$, $j = 0, 1, 2, ...N + 1$, and $h = \max\limits_{0 \leq \ j \leq \ N+1} h_j$.

Proof. Using the results prove by *U. Ascher, R. M. Mattheij şi R. D. Russel* in ([6, **Theorem 5.140,** page: 253]) for linear boundary values problem (BVPL) (4.34), we obtained estimates (4.38) and (4.39). Errors obtained for the approximations $\alpha = y'_{\overline{\Delta}}(a)$ and $\beta = y'_{\overline{\Delta}}(b)$ are of order $O(h^p)$, ([28, **Theorem** 5.4.1, page: 293]). If we choose

and embedded pair of Runge-Kutta method of order at least $(p, p+1)$, and conditions in hypothesis of theorem are fulfilled, then the final error have the order $O(h^p)$. The stability of this method results from the equivalence of collocation method with a *Runge-Kutta* method [6, **Theorem** 5.73, page: 219].

So if the mesh is sufficiently fine, the embedded pair of *Runge-Kutta* methods does not increase the order of error. ■

Remark 54 *The condition number may grow rapidly when h is small. In the paper [6, page:129]* U.Ascher *give the following estimation:*

$$\kappa_\Delta \approx K \sum_{i=0}^{N+1} h_i^{-2} \max_{i=0,\ldots,N+1} \int_{x_i}^{x_{i+1}} |G(x_i, t)|\, dt,$$

where K is a generic constant and G is the Green's *function for (BVPL) problem.*

Remark 55 *We suggest using the combined approach method (Runge-Kutta and B-spline of degree k) if the problems (PVPL) (4.1) have singularities on the endpoints of interval* $[0, 1]$. *Detection can be done using the step control algorithm ([1, pp: 376-380], [78]).*

Conclusion 56 *The error estimation does not depend on the number of collocation points. Nevertheless, the* Runge-Kutta *methods requires an order greater or equal to the order of error for derivatives at a and b. We conclude: collocation method combined with* Runge-Kutta *method is an effective method to determinate an approximate solution of (PVPL) problem.*

Chapter 5

Nonlinear Boundary Value Problem

5.1 The statement of problem

Consider the second order nonlinear boundary value problem (PVPN) so

$$\begin{cases} y''(x) + f(x,y) = 0, & x \in [0,1] \\ y(a) = \alpha;\ y(b) = \beta, & a,b \in (0,1),\ a < b, \end{cases} \tag{5.1}$$

where $a, b,\ \alpha, \beta \in \mathbb{R}$ and $f : [0,1] \times \mathbb{R} \to \mathbb{R}$.

We try to solve the above problem using three methods:

1. a global collocation method based on B-splines of order $k+1$.

2. a combined method based on B-splines (order $k+1$) and a *Runge-Kutta* methods.

3. a pseudo-spectral collocation method (*C.C* method) with *Tchebychev* extreme points combined with a *Runge-Kutta* method.

Our choice to use these methods is based on the following reasons:

1. We write the code using the function `spcol in MATLAB Spline Toolbox ([49],[66],[69])`, and in `MATLAB dmsuite [80].`

2. It is the most suitable method, for a general purpose code, among the finite element ones. See ([8], [62], [63]), where complexity comparisons which support the above claim are made and collocation, when efficiently implemented, is shown to be competitive using extrapolation.

3. Theoretical results on the convergence of collocation method are given in ([39], [79]).

4. Several representative test problems demonstrate the stability and flexibility in [14].

5. For each *Newton* iteration, the resulting linear algebraic system of equations (after using *Newton* method with quasilinearization) is solved using methods given in [16].

6. There exists elegant theoretical results on the convergence of collocation method (see for example [36]).

We also consider the (BVPN) problem

$$\begin{cases} y''(x) + f(x, y) = 0, & x \in [a, b] \\ y(a) = \alpha;\ y(b) = \beta, & a, b, \alpha, \beta \in \mathbb{R} \end{cases} \tag{5.2}$$

To apply the collocation theory, we need to have an isolated solution $y(x)$ of the the problem (5.2), and this occurs if the above linearized problem for $y(x)$ is uniquely solvable. This is assuring by (**Theorem** 3) (see [61]).

5.2 Global collocation method based on B-splines function of order (k+1)

First we are interested to a global approach for the solution of the problem (PVPN) (5.1). We consider the grid $\overline{\Delta}$ of $[0,1]$ given by (4.3), and a number of such partitions $\overline{\Delta}_n$ $(n = 1, 2, 3, ...)$ which satisfy the condition

$$\lim_{n\to\infty} h(\overline{\Delta}_n) = 0, \text{ where } h := \max\{h_i\},\ h_i := x_{i+1} - x_i.$$

Definition 57 *A function $v(x)$ is in family $L(\overline{\Delta}_n, k, 2)$ if $v(x)$ is a polynomial of degree k on each subinterval of $\overline{\Delta}_n$, $v(x) \in C^p[0,1]$ and the subfamily*

$$L'(\overline{\Delta}_n, k, 2) \subset L(\overline{\Delta}_n, k, 2),$$

consists of all functions that satisfied the boundary value conditions

$$y(a) = \alpha;\ y(b) = \beta,\ a, b \in (0,1),\ a < b.$$

The next lemma gives a sufficient condition for convergence.

Lemma 58 *([61], [79]) Let $\overline{\Delta}$ be a grid (4.3) and if we form a set of points $S_n (n = 1, 2, 3, ...)$ like in (1.53) then for a large n, there is a unique element $u_{\overline{\Delta}_n}(x) \in L'(\overline{\Delta}_n, k, 2)$ satisfying (5.2) at each point of S_n and*

$$\left\| u^{(j)}_{\overline{\Delta}_n}(x) - y^{(j)} \right\| \leq \delta,\ j = 0, 1.$$

The approximate solution $u_{\overline{\Delta}_n}(x)$ and its derivatives up to the order two converge uniformly to exact solution $y(x)$. Moreover the rate of convergence is bounded by:

$$\left\| u^{(j)}_{\overline{\Delta}_n}(x) - y^{(j)} \right\| \leq \theta F_n(y''),\ j = 0, 1,$$

where θ is a constant independent of n and $F_n(y'')$ is the error of the best uniform approximation to $y''(x)$ in $L(\overline{\Delta}_n, k, 2)$.

For reasons of efficiency, stability, continuity, we choose as the basis B-splines of order $k+1$, where $k =$ degree of *Legendre* polynomial. We wish to find an approximate solution of (PVPN) problem (5.1) in $L(\overline{\Delta}_n, k, 2)$, having the following form

$$u_{\overline{\Delta}_n}(x) = \sum_{i=0}^{N+2} c_i B_{i,k}(x), \tag{5.3}$$

where $B_{i,k}(x)$ is a B-spline of order $(k+1)$ with mesh-points

$$\overline{\Delta} := \{\xi_i \ , i = -k+1, ..., N+k+1\},$$

and c_i are real constants to be determined.

Remark 59 *Our approximation method is inspired from [10, chap. 2,5].*

Let

$$J = \{0, 1, ..., N+2\} \backslash \{l, l+p\}.$$

We impose the conditions:

- **(c1)** The approximate solution (5.3) satisfies the differential equation (5.1) at ξ_j, $j \in J$, where ξ_j are the collocation points.
- **(c2)** The solution satisfies:

$$u_{\overline{\Delta}_n}(\xi_l) = \alpha, \ u_{\overline{\Delta}_n}(\xi_{l+p}) = \beta \ , \ a = \xi_l, \ b = \xi_{l+p}.$$

The conditions **(c1)** and **(c2)** yield a nonlinear system with $(N+3)$ equations

$$\begin{cases} \sum\limits_{i=0}^{N+2} c_i B_{i,k}(a) = \gamma_1, \\ \sum\limits_{i=0}^{N+2} c_i B_{i,k}(b) = \gamma_2, \\ \sum\limits_{i=1}^{N+1} c_i B_i''(\xi_j) + f(\xi_j, \sum\limits_{i=1}^{N+1} c_i B_i(\xi_j)) = 0, \end{cases} \tag{5.4}$$

with unknowns $\{c_i\}_{i=0}^{N+2}$. If $F = [F_0, F_1, ..., F_{N+2}]^T$ are the functions defined by equations of the nonlinear systems, using the quasilinearization of *Newton* [6, pp. 52-55], we find the next approximation by means

$$c^{(v+1)} = c^{(v)} - w^{(v)} \tag{5.5}$$

where $c^{(\upsilon)}$ is the vector of unknowns obtained at the $\upsilon-$th step and $w^{(\upsilon)}$ is the solution of the linear system

$$F\,'(c^{(\upsilon)})w = F(c^{(\upsilon)}). \tag{5.6}$$

The Jacobian matrix $F\,' = (J_{ij})$ is banded and it is given by

$$J_{ij} = \begin{cases} B_j(a), & \text{for } i = 0 \\ B_j(b), & \text{for } i = N+2 \\ B_j''(\xi_i) + \frac{\partial f}{\partial y}(\xi_i, \sum_{i=1}^{n+1} c_i B_j(\xi_i))y(\xi_i), & \text{for } i = 1, 2, 3, ...N+1 \end{cases} . \tag{5.7}$$

To solve (5.1) we used the method presented in [21]; an initial approximation $u^0 \in C^1\,[0,1]$ is required. The successful **stopping criterion** [7] is

$$\left\|u^{(k+1)} - u^{(k)}\right\| \leq abstol + \left\|u^{(k+1)}\right\| reltol,$$

where *abstol* and *reltol* is the absolute and the relative error tolerance, respectively and the norm is the usual uniform convergence norm. The reliability of error-estimation procedure being used for stopping criterion was verified in [4]. Papers on this topics exploit the almost block diagonal structure of collocation matrix and recommend LU factorization ([4], [16]).

5.3 A combined method using B-splines and Runge-Kutta methods

The nonlinear boundary value problem (PVPN) can be solved by dividing it in three problems like:

1. A two-points nonlinear boundary values (BVPN) (5.2) on $[a, b]$,

2. Two nonlinear problems *Cauchy* (IVPN-Left) on $[0, a]$

$$\begin{cases} y'(x) = z(x) \\ z'(x) = -f(x, y) \\ y(a) = \alpha;\ y'(a) = \gamma,\ \gamma \in \mathbb{R} \end{cases}, \tag{5.8}$$

3. and (IVPN-Right) on $[b, 1]$

$$\begin{cases} y'(x) = z(x) \\ z'(x) = -f(x, y) \\ y(b) = \beta;\ y'(b) = \delta,\ \delta \in \mathbb{R} \end{cases}, \tag{5.9}$$

or treating it globally.

Assuming that the two-points nonlinear boundary values problem (BVPN) given by (5.2) has the unique solution, the requirement $y \in C^1[0, 1]$ ensure the existence and the uniqueness of the exact solution of (5.1). For the existence and the uniqueness of the exact solutions of the problems (IVPN-Left) and (IVPN-Right) we use **Theorem** 4 [35, **Theorem 3.1,** page: 113].

We use the mesh $\overline{\Delta}$ of the interval $[0, 1]$ given by (4.3). To solve numerically the two-points nonlinear problem (BVPN), we use global collocation method with B-spline functions of degree k presented in the previous paragraph, and for the two problems (IVPN-Left) and (IVPN-Right) we use *Runge-Kutta* methods of order k.

Since $f(x, y) \in C^2\{[0, 1] \times \mathbb{R}\}$ and it must ensure continuity and differentiability in $x = a$ and $x = b$, we choose the order of multiplicity k for knots, $\xi_j \in \overline{\Delta},\ j \neq l, l + p$, and

for inner points $\xi_l = a$ şi $\xi_{l+p} = b$ the order of multiplicity $k+2$. The values $y\,'(a)$ şi $y'(b)$ can be computed by differentiating the approximate solution of the two-points boundary values problem (BVPN) (5.2) in points $x = a$ and $x = b$, and let $u_{\overline{\Delta}}(x)$ be the approximation computed by the combined method.

Theorem 60 *[58, section 3] If u is the isolated solution of (PVPN) problem, f has continuous second order partial derivative and its accuracy is $\mathbb{O}\ (h^{k+1})$, where h is the norm of the partition $\overline{\Delta}$.*

Proof. For the (PVPN) problem (5.1) we apply **Theorem** 5.147, page: 257 in [6].

We conclude that *Newton* method, applied to Δ, converges quadratically to the restriction of u to Δ, and the accuracy for the approximation and its derivative is $\mathbb{O}(h^{k+1})$, that is:

$$\left|u_{\overline{\Delta}}^{(j)}(x) - y^{(j)}(x)\right| = \mathbb{O}(h^{k+1}), \quad x \in [a,b],\ j = 0, 1.$$

We extend convergence and the accuracy to the whole interval $[0,1]$ by using the stability and the convergence of *Runge-Kutta* methods. A $(k+1)$-order explicit *Runge-Kutta* methods is consistent and stable, so it is convergent, and its accuracy is $\mathbb{O}(h^{k+1})$. Thus the final solution has the same accuracy. The stability and convergence of *Runge-Kutta* method are guaranteed by **Theorem 5.3.1, page 285** and **Theorem 5.3.2, page 288** in [28]. ■

5.4 Some considerations on complexity

We will give a rough estimation of the complexity of above methods. We start with the first method. In the sequel, B will be the cost for B-spline evaluation and f the time for a function evaluation. The time required to construct the collocation matrix is

$$C_0 = 2(Nk+1)(k+1)B.$$

To construct the Jacobian we need $Nk(k+1)(B+f)$. The construction of the right-hand side requires

$$(Nk+2)B + NkB + Nkf.$$

So, for the linear system construction we obtain

$$W_1 = ((B+f)k^2 + (4B+3f)kN + B.$$

For a banded linear system with bandwidth w the total cost for solution, using LU, with pivoting is

$$n\left(\frac{w^2}{2} + w\right),$$

(see [25, pp. 79-82],[33]). In our case, $n = Nk+2$, and $w = \frac{3}{2}(k+2)$, and the cost for the solution of linear system will be:

$$W_2 = \left(\frac{21}{2}k + \frac{9}{8}k^3 + \frac{15}{2}k^2\right)N + 15k + 21 + \frac{9k^2}{4}.$$

The cost of *Newton step* is $W_s = W_1 + W_2$, that is

$$W_s = \left[\frac{9k^3}{8} + \left(f + B + \frac{15}{2}\right)k^2 + \left(3f + 4B + \frac{21}{2}\right)k\right]N + 2B + 15k + 21 + \frac{9k^2}{4}.$$

The total cost is $IW_s + C_0$, where I is the number of steps required in *Newton* methods. Since the convergence is quadratic, if the final tolerance is ε, assuming $\delta_{i+1} = c\delta_i$, is the error at $i-th$ step, we obtain ([48, pp 295-297],[79]):

$$I = \frac{1}{\log 2}\log\frac{\log|c| + \log\varepsilon}{\log|c| + \log|\delta_0|}.$$

For the second method, the same analysis works for (BVPN) solution part. We have an additional amount of work for *Runge-Kutta* method. If the number of stages is s and the number of points is p, the cost is $\mathbb{O}(psf)$.

5.5 A combined method with pseudo-spectral and *Runge-Kutta* methods

Consider the grid

$$0 = x_{-q} < ... < x_{-1} < a = x_0 < x_1 < ... < x_N = b < x_{N+1} < ... < x_{N+p} = 1 \tag{5.10}$$

We choose *Gauss-Lobatto-Chebychev* type mesh:

on $[a, b]$

$$x_i = \frac{(b-a)\cos\frac{\pi i}{N} + a + b}{2}, i = 0, 1, ...N, \tag{5.11}$$

on $[0, a]$

$$x_i = \frac{a}{2}(-\cos\frac{\pi i}{N} + 1), i = -1, -2, ... - q, \tag{5.12}$$

and on $[b, 1]$

$$x_i = \frac{(b-1)\cos\frac{\pi i}{N} + 1 + b}{2}, i = N+1, N+2, ..., N+p. \tag{5.13}$$

To approximate the solution of two points boundary nonlinear problem (PVPN) follows the ideas in [20]. Considering the *Lagrange* basis (l_k) we have

$$y(x) = \sum_{k=0}^{N} \ell_k(x) y(x_k) + (R_N y)(x), \qquad x \in [a, b], \tag{5.14}$$

where

$$(R_N y)(x) = \frac{y^{(N+1)}(\xi)}{(N+1)!}(x - x_0)\cdots(x - x_N),$$

$$\min\{x_0, x_1, ..., x_N\} < \xi < \max\{x_0, x_1, ..., x_N\},$$

is the remainder of *Lagrange* interpolation.

Since $y(x)$ fulfills the differential equation given by (5.2), we obtain

$$\sum_{k=0}^{N} \ell_k''(x_i) y(x_k) + (R_N y)''(x_i) = -f(x_i, y(x_i)), \qquad i = 1, \ldots, N-1. \tag{5.15}$$

Setting $y(x_k) := y_k$ and ignoring the rest, one obtains the nonlinear system

$$\sum_{k=0}^{N} \ell_k''(x_i) y_k = -f(x_i, y(x_i)), \qquad i = 1, \ldots, N-1, \tag{5.16}$$

with unknowns y_k, $k = 1, 2, ..., N-1$, here $y_0 := y(x_0) = \alpha$,and $y_N := y(x_N) = \beta$. The approximate solution of (BVPN) problem given by (5.2) is in fact the *Lagrange* interpolation polynomial at nodes $\{x_k\}, k = -q, ..., 0, 1, ...N, ..., N+p$. The nonlinear system (5.16) can be rewritten as

$$AY_N = F(Y_N) + b_N$$

where

$$A = [a_{ik}], \quad a_{ik} = \ell_k''(x_i), \quad k, i = 0, 1, \ldots, N-1, N, \tag{5.17}$$

$$F(Y_N) = \begin{bmatrix} -f(x_1, y_1) \\ \vdots \\ -f(x_{N-1}, y_{N-1}) \end{bmatrix}, \qquad b_N = \begin{bmatrix} -\alpha \ell_0''(x_1) - \beta \ell_N''(x_1) \\ \vdots \\ -\alpha \ell_0''(x_{N-1}) - \beta \ell_N''(x_{N-1}) \end{bmatrix}. \tag{5.18}$$

Since

$$\ell_k(x_i) = \ell_{N-k}(x_{N-i}); \ell_k''(x_i) = \ell_{N-k}''(x_{N-i}), \quad k, i = 0, 1, \ldots, N-1, N,$$

the matrix A is centro-symmetric (see [19] for proof), so nonsingular.

We introduce

$$G(Y) = A^{-1} F(Y) + A^{-1} b_N, \tag{5.19}$$

where the solution Y will be fixed point of G. To solve numerically the problem(5.1) on mesh given by (5.11) we use $C.C$ method on $[a, b]$ and on $[0, a]$, $[b, 1]$ *Runge-Kutta* methods, where on $[0, a]$ we use the knots given by(5.12) and on $[b, 1]$ the knots given by (5.13). The derivative $y'(a)$ şi $y'(b)$ can be computed using the relations (4.30) and (4.32).

In [20, **Theorems: 2.1, 2.2**] *F.A. Costabile* and *A.Napoli* prove the following theorems.

Theorem 61 *If f verifies a* Lipschitz *condition with respect to second variable:*

$$|F(x, y_1) - F(x, y_2)| \leq L |y_1 - y_2|$$

and $\|A^{-1}\| L < 1$, *then the system (5.16) has a unique solution which can be calculated by successive approximation method*

$$Y^{(n+1)} = G(Y^{(n)}), n \in N^*, \tag{5.20}$$

where $Y^{(0)}$ *fixed and* G *given by (5.19), and the matrix* A *is (5.17).*

Theorem 62 *If* $Y = [y(x_1), y(x_2), ..., y(x_{N-1})]^T$, *where* $y(x) \in C^1[0,1]$ *is the exact solution of (BVPN) problem and:*

$$Y_N = [y_1, y_2, ..., y_{N-1}]$$

where y_i *are the values of approximated solution at* $x_i \in \Delta_1$ *computed by (5.16) and*

$$R = [-(R_N y)''(x_1), ..., -(R_N y)''(x_{N-1})]^T,$$

then for error: $\|Y - Y_N\|$ *it holds:*

$$\|Y - Y_N\| \leq \frac{\|A^{-1}\| \, \|R\|}{1 - \|A^{-1}\| \, L}. \tag{5.21}$$

Combining the results obtained by *F.A. Costabile, A.Napoli* [20, **Theorems: 2.1, 2.2**] with the stability and convergence of *Runge-Kutta* methods [28, **Theorem** 5.3.1 page 285, **Theorem** 5.3.2 page 288] we obtain

Theorem 63 ***(Daniel N.Pop, Radu T.Trâmbiţaş , I.Păvăloiu [57])*** *If*

$$\frac{\|A^{-1}\| \, \|R\|}{1 - \|A^{-1}\| \, L} \leq \mathbb{O}(h^k) \text{ and}$$

$$|y_N(a) - y'(a)| = \mathbb{O}(h^k),$$

$$|y_N(b) - y'(b)| = \mathbb{O}(h^k),$$

then for each point $x_i \in \Delta_1$, $i = -q, ..., N + p$ *occurs*

$$|y(x_i) - y_i| = \mathbb{O}(h^k).$$

Proof. The condition $\|A^{-1}\| L < 1$, assures us that G is a contraction. From *Banach's* fix-point theorem it follows that $(Y^{(n)})$ given by (5.20) is convergent to the exact solution $\overline{Y}$ of the system (5.16) and the following estimation holds:

$$\left\|\overline{Y} - Y^{(n)}\right\| \leq \frac{(\|A^{-1}\| \|L\|)^n}{1 - \|A^{-1}\| L} \left\|Y^{(1)} - Y^{(0)}\right\|.$$

If accuracy of the collocation method for the problem (5.2) is $\mathbb{O}(h^k)$ (i.e the approximate solutions y and its derivative $y'(a), y'(b)$ are within this accuracy limit), and if *Runge-Kutta* method is stable and has the order k, then the final solution has same accuracy. The stability and convergence of *Runge-Kutta* method are guaranteed by [28, **Theorems 5.3.1 page 285 and 5.3.2 page 288**]. ■

5.6 Numerical examples

In the following we make a comparative computational study by examples of the three numerical methods:

1. global method with B-spline functions,
2. combined collocation method with B-splines and *Runge-Kutta* methods,
3. *C.C* method combined with *Runge-Kutta* methods.

Using the idea given in ([10], [13], [22]) we write the code using `MATLAB Spline Toolbox and sparse matrices`, since the function `spcol` allows us to easily calculate the inverse of collocation matrix. We write and implemented two functions in `MATLAB` :

1. `polycollocnelin`, for global collocation method with B-spline functions,
2. `polycalnlinRK`, for combine method (B-spline collocation and *Runge-Kutta*).

5.6.1 The case where we know the exact solution

We gave in the paper [58], two numerical examples with know exact solution:

1. *Goldner'* ***problem*** [31]

$$\begin{array}{l} y''(x) + y^3(x) + \dfrac{4-(x-x^2)^3}{(x+1)^3} = 0;\ x \in (0,1) \\ y(1/4) = 3/20;\ y(1/2) = 1/6 \end{array}, \tag{5.22}$$

   with the exact solution

$$y(x) = \frac{x-x^2}{x+1}.$$

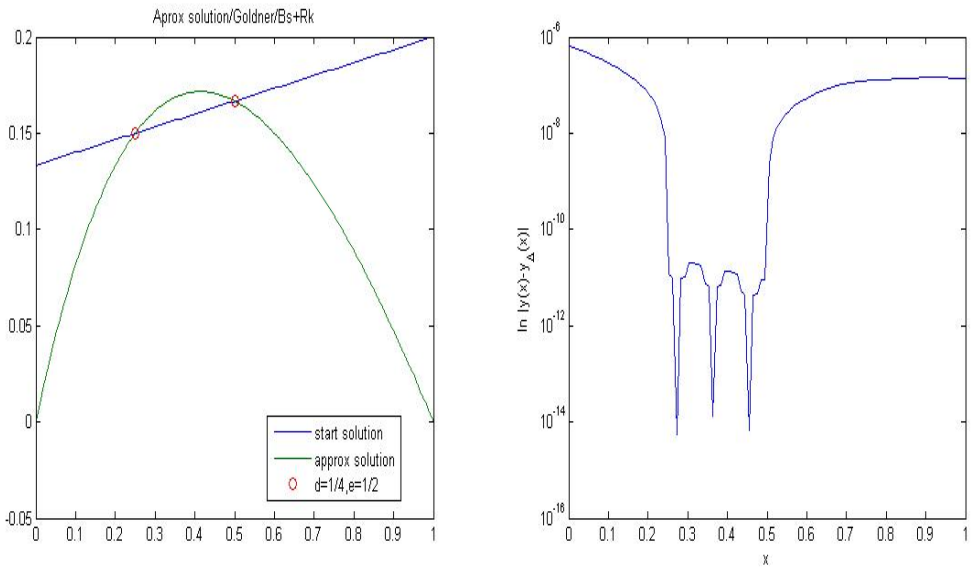
2. *Costabile'* ***problem*** [18]

$$\begin{array}{l} y''(x) + e^{-y(x)} = 0;\ x \in [0,1] \\ y(\pi/6) = \ln(3/2),\ y(\pi/4) = \ln((2+\sqrt{2})/2) \end{array}, \tag{5.23}$$

with the exact solution

$$y = \ln(\sin(x) + 1).$$

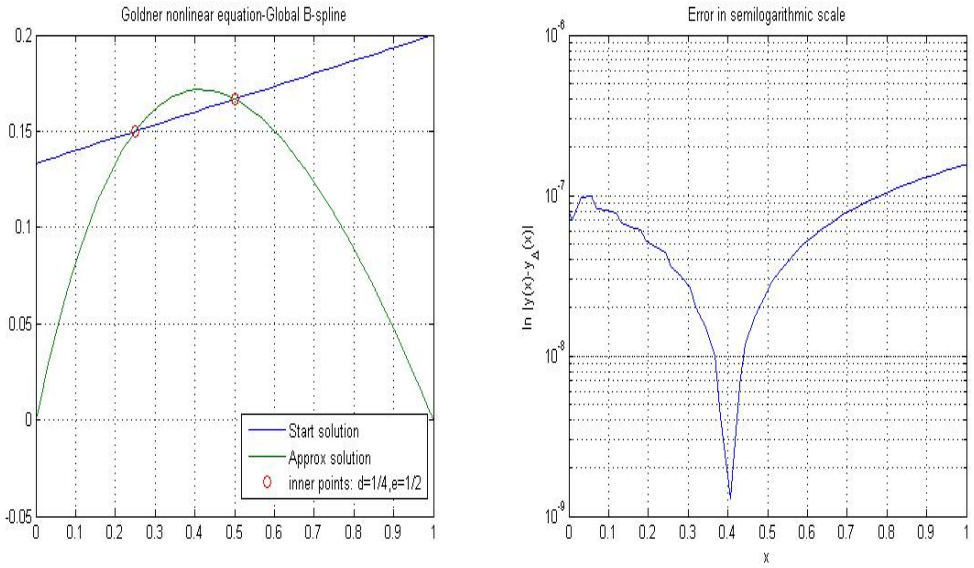
For the first example we applied two methods: global with *B-spline* of degree k and combined *B-spline* with *Runge-Kutta*. Figure 5.1(a) and 5.2(a) present approximate solution for the two-points boundary value problems (5.22).Figure5.3(a) present approximate solution for (5.23). Errors for the two problems are depicted in Figure 5.1(b),5.2(b) for the first example and Figure 5.3(b) for the second example. We have chosen as initial approximation *Lagrange* interpolation polynomial with the values α and β at a, $b \in (0, 1)$, computed using the `MATLAB` function `polyfit`.



(a) Approximate solution

(b) Errors in semilogarithmic scale

Figure 5.1: Goldner nonlinear problem-Collocation method with B-spline+RungeKutta



(a) Approximate solution

(b) Errors in semilogarithmic scale

Figure 5.2: Goldner nonlinear problem-Global collocation method

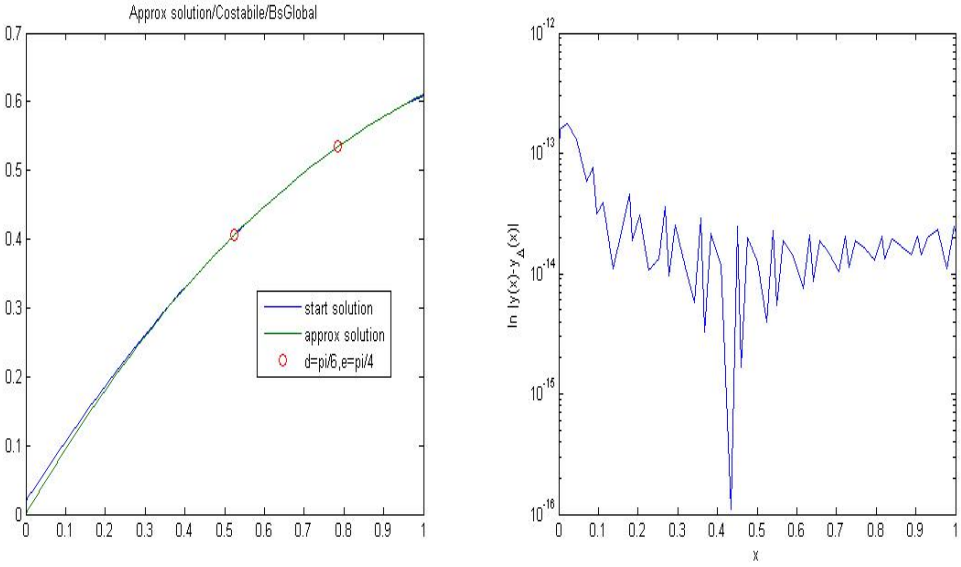
Table 5.1 gives the residuals:

$$e_{\Delta}^{(j)} \left\| y^{(j)} - u_{\Delta}^{(j)} \right\|, \text{ for } j = 0, 1, 2,$$

for the global method based on B-splines. For the residuals it holds:

$$\left\| y^{(j)} - u_{\Delta}^{(j)} \right\| = \mathbb{O}(\|\Delta\|)^{k+2-j}, \text{ for } j = 0, 1, 2$$

In order to compare the costs (run times) experimentally we used `MATLAB` functions `tic` and `toc`,the results are given in Table 5.2. The time for combined method is a bit larger.



(a) Approximate solution

(b) Errors in logarithmic scale

Figure 5.3: Costabile-non linear problem

N	$\|y - u_{\overline{\Delta}}\|$	$\|y' - u'_{\overline{\Delta}}\|$	$\|y^{"} - u^{"}_{\overline{\Delta}}\|$	$\|y - u_{\overline{\Delta}}\|$	$\|y' - u'_{\overline{\Delta}}\|$	$\|y^{"} - u^{"}_{\overline{\Delta}}\|$
5	$6e-05$	0.000123	0.002290	$1.04e-05$	$2.15e-05$	0.000172
6	$6e-05$	0.000123	0.002290	$3.72e-06$	$8.47e-06$	0.000124
7	$6e-05$	0.000123	0.002290	$1.59e-06$	$3.96e-06$	0.000106
8	$6e-05$	0.000123	0.002290	$7.37e-07$	$2.02e-06$	$5.65e-05$
9	$6.9e-06$	$1.63e-05$	0.000769	$3.84e-07$	$1.16e-06$	$3.61e-05$
10	$6.9e-06$	$1.63e-05$	0.000769	$2.04e-07$	$6.81e-07$	$2.42e-05$
11	$6.9e-06$	$1.63e-05$	0.000769	$1.21e-07$	$4.46e-07$	$2.14e-05$
12	$6.9e-06$	$1.63e-05$	0.000769	$1.82e-08$	$2.02e-07$	$1.81e-05$
13	$1.4e-06$	$3.64e-06$	0.000346	$1.30e-08$	$1.44e-07$	$9.77e-06$
14	$1.4e-06$	$3.64e-06$	0.000346	$7.54e-09$	$1.09e-07$	$1.51e-05$
15	$1.4e-06$	$3.64e-06$	0.000346	$5.54e-09$	$8.32e-08$	$6.08e-06$
16	$1.4e-06$	$3.64e-06$	0.000346	$3.54e-09$	$6.22e-08$	$7.92e-06$
17	$3.99e-07$	$1.22e-06$	0.000148	$2.81e-09$	$4.65e-08$	$4.65e-06$
18	$3.99e-07$	$1.22e-06$	0.000148	$1.89e-08$	$3.51e-08$	$4.76e-06$
19	$3.99e-07$	$1.22e-06$	0.000148	$1.43e-09$	$2.98e-08$	$4.14e-06$
20	$3.99e-07$	$1.22e-06$	0.000148	$1.02e-09$	$2.50e-08$	$3.70e-06$

Table 5.1: Error table for example (1) left, and example (2) right

	Method 1	Method 2
First example	0.0117501	0.022751
Second example	0.0163870	0.021535

Table 5.2: Run times

5.6.2 Case of unknown exact solution

To implement C.C method combined with Runge-Kutta methods we used the MATLAB functions:

`cebdif,cebint,cebdiff,`

from the library `dmsuite` described in ([80], [70]), and for *Cauchy* problems on the intervals $[0, a]$, $[b, 1]$ we use `ode45`. We chose $\{x_k, k = 0, 1, 2, ..., n\}$ as extreme *Tchebychev* points on the interval $[0, 1]$. Since the successive approximation method is slow, we solve the nonlinear system by *Newton' method.* For this purpose we wrote the function `solvepolylocalceb` and then we called solver `ode45` for *Runge Kutta* methods, and the derivatives $y'(a), y'(b)$ are computed by the function `cebdifft`.

- **The first example**: ***Bratu*' problem** (1.77) for $\lambda = 1$.

$$\begin{cases} y\,''(x) + e^{y(x)} = 0,\ 0 < x < 1 \\ y(0.2) = 0.08918993462883 \\ y(0.8) = 0.08918993462883 \end{cases} . \tag{5.24}$$

We took $N = 128$. We solve the two-points boundary values nonlinear problem (5.24) using three methods:

1. *C.C* method combined with *Runge-Kutta* methods, where we chose initial start value

$$y^{(0)}(x) = x(1 - x), \tag{5.25}$$

2. global collocation method with B-splines, where we chose initial start value

$$y^{(0)}(x) = 3.9x(1 - x)/7,$$

3. combined method B-splines with *Runge-Kutta* methods ,where we chose initial start value

$$y^{(0)}(x) = x(1 - x).$$

4. BVP4c solver.

We used to compare the cost, the MATLAB functions `tic` and `toc` and we obtained the following results

ε	$C.C + R.K$	$B.S + R.K$	$Global - BS$	$Bvp4c$
10^{-3}	0.221242	0.040019	0.034571	0.214106
10^{-4}	0.230269	0.046225	0.022515	0.20669
10^{-5}	0.238405	0.054828	0.032165	0.18435

The graph of approximate nonlinear solution of the problem (5.24) is obtained after 4 iterations, and we plotted it in Figure 5.4.

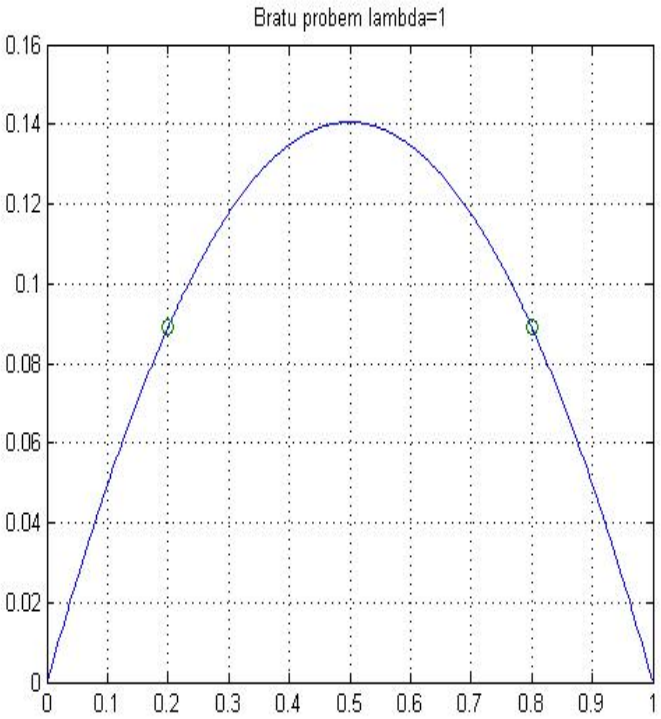


Figure 5.4: Bratu-problem for $\lambda = 1$

- **The second example: The average temperature of reaction-diffusion problem.**

C.Gheorghiu, D.Trif (1.79) show that actually this positive solution and its numerical approximation are respectively the stationary unstable hyperbolic points of the continuous and discrete dynamical systems. Straightforward direct (`based on Newton's method`) and indirect approaches of this positive steady state proved to be unsatisfactory. Thus, they suggest a modified indirect algorithm, which takes into account the integral condition, and prove its convergence for $p = 3$.

$$\begin{cases} u'' + u^3 = 0;\ 0 < x < 1 \\ u(0.2) = 1.929990320692795 \\ u(0.8) = 1.929990320692795 \end{cases} . \tag{5.26}$$

In this numerical example we took $N = 128$, the tolerance $\epsilon = 10^{-8}$ and for the initial solution we chose

$$u^{(0)} = \frac{12\pi}{\sqrt[2]{2}} x(1-x), \tag{5.27}$$

for C.C method combined with *Runge-Kutta*, *B-spline* with *Runge-Kutta* and for the initial start value we built a spline that passes through the points $[0;\ 0.2;\ 0.5;\ 0.8; 1]$. We have determined the values $u(0.2),\ u(0.5)$ and $u(0.8)$ writing code in `Maple14.0.`

After 8 iterations, we obtained the approximate nonlinear solution of the problem (5.26) in Figure 5.5.

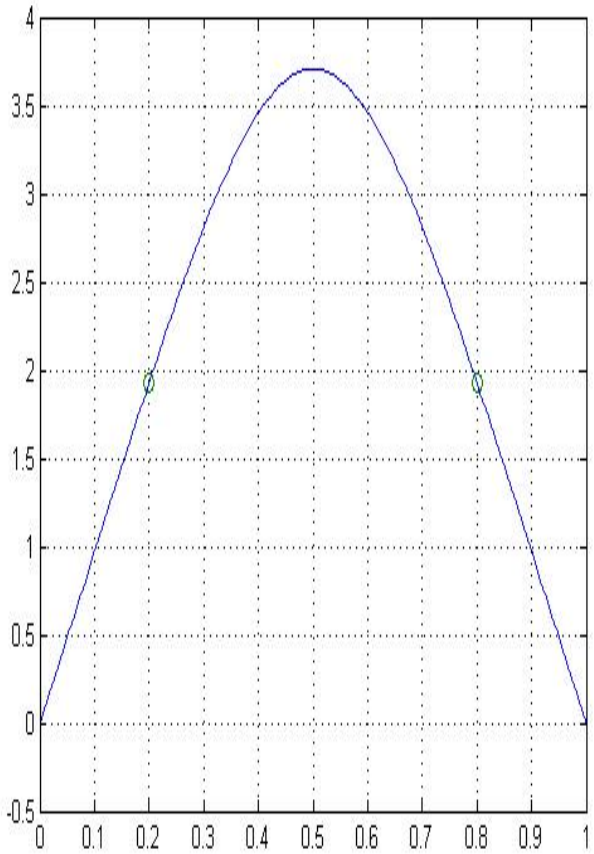


Figure 5.5: The average temperature in reaction-diffusion process $p = 3$

We solved the two-points boundary nonlinear problem (5.26) using three methods:

1. combined method C.C with Runge-Kutta,
2. global collocation method with B-splines,
3. combined method B-splines with Runge-Kutta.
4. BVP4c

Using again the functions `MATLAB tic` and `toc` to compare the cost (run-times), we obtained the following results:

ε	$C.C + R.K.$	$B.S + R.K.$	$Global - B.S$	$BVP4c$
$1.e-3$	0.221242	0.040019	0.034571	0.234253
$1.e-4$	0.230269	0.046225	0.022515	0.240021
$1.e-5$	0.238405	0.054828	0.032165	0.239204

where ε is again the tolerance in stop criterion.

Conclusion 64 *In both examples studied global B-splines method is faster.*

Remark 65 *Cost and number of iterations depend on the number of division points N. The tolerance used in stop criterion ϵ and the initial value $u^{(0)}$ is essential because it ensures convergence method.*

Chapter 6

Boundary Conditions

When modeling a physical problem with BVP, it may not be clear what kinds of boundary conditions to use and where to apply them. It is not at all unusual for BVPs to be singular. The case when we have singularities at end points of the interval in the sense that either the ODE are singular at an end points, or the interval is infinite, or both. The standard theory does not apply to such problems so if we are to solve them, we must bring to bear all insight that we can gain from the underlying physical problem and analysis the mathematical problem.

The total number of boundary conditions must be equal to the sum of the order of the ODEs plus number of unknown parameters. BVP solvers will fail if you do not supply enough boundary conditions or if you supply too many. Usually the boundary conditions arise from the nature of problem and usually it is straightforward to determine their type and where to apply them. However, there are occasions when

- It may be difficult to determine one or more of the boundary conditions because all the direct physical constraints on the solution have been imposed and there are still not enough boundary conditions. In such case you should look for additional boundary conditions arising from the requirement that certain integrals of the system of ODEs should be conserved.

- Some of the boundary conditions may be of a special form. For example, the solution must remain bounded at a singularity or the solution must decay, or decay in a particular way, as the independent variable tends to infinity. In such cases it is usually possible to convert this information into standard boundary conditions, possibly by introducing unknown parameters.

- There may be too many boundary conditions. Sometimes your knowledge of the underlying problem leads to knowing more about the solution than is necessary to specify the BVP. In such cases you should drop those boundary conditions that are consequences of the other boundary conditions and the ODEs. Usually (but not always) such boundary conditions involve higher derivates of the solution.

6.1 Solving BVPs in Matlab with BVP4C

The theoretical approach to BVPs is based on the solution of IVPs and the solution of nonlinear algebraic equations. Since there are effective programs for both tasks, it is natural to combine them for solving BVP. A method of this kind is called a *shooting method.* Because it appears so straightforward to use quality numerical tools for the solution of BVPs by shooting, it is perhaps surprising that the most popular solvers are not shooting codes.

The basic difficulty with shooting is that a well-possed BVP can required the integration of IVPs that are unstable. That is, the solution of a BVP can be insensitive to changes in its boundary values, yet the solution of the IVPs arising in the shooting method for solving the BVP are sensitive to changes in the initial values.

In the book [67, pp.156-167] *L.R. Shampine* gives an suggestive example.

In this chapter we use a variety of examples from the literature to illustrate the numerical solution of BVPs with `MATLAB` solver `bvp4c`.BVPs arise in such diverse forms that many requires some preparation for their solution by standard software and some require extensive preparation. Several examples illustrate how to do this for the most common forms and the most winkle used BVP solvers. Other examples show how you might be able to speed up significantly the solution of a BVP with `bvp4c`.

In simplest use `bvp4c` has only three arguments :

1. a function `odes` for evaluating the ODEs
2. a function `bcs` for evaluating the residual in the boundary conditions
3. a structure `solinit` that provides a guess for a mesh and the solution on this mesh.

The ODEs are handled exactly as in the `matlab IVP` solvers and so require no further discussion here. Boundary conditions

$$g(y(a), y(b)) = 0$$

are coded so that, when the function `bcs` to given the current approximation $ya \approx y(a)$, and $yb \approx y(b)$, it evaluates and returns the residual $g(ya, yb)$ as a column vector.

When solving BVPs you must provide a guess (initial or starting) for the solution, both to identify the solution that interests you and to assist the solver in computing that solution.The guess is supplied to `bvp4c` as a structure formed by the auxiliary function `bvpinit`.The first argument of `bvpinit` is a guess for a mesh that reveals the behavior of the solution. The derivates of guess $y(x)$ is used as the guess for $y'(x)$. Alternatively if we take all the estimated solution components to be constant then we can provide the vector of these constants directly.

When solving BVPs returns a solution structure, called here `sol`.This is an option with IVP solvers, but it is the only form of output from `bvp4c`.As the IVP solvers, the field `sol.x` contains the mesh found by the solver and the field `sol.y` contains the solution on the mesh.

The cost of solving a BVP depends strongly on the number of mesh points, so `bvp4c` uses no more than necessary. The values plotted are accurate , so we need only approximate the solution at more points in order to plot a smooth graph. The solver `bvp4c` produces a solution $S(x) \in C^1[a, b]$ and returns in `sol`,all the information needed to evaluate this smooth approximate solution. Just as with the IVP solvers, this is done with auxiliary function `deval`.

Recall that the first argument of `deval` is the solution structure and the second is an array of points where the solution is desired. The function `deval` is evaluating a piece-wise cubic function and it is vectorizing , so obtaining even a great many solution values for plotting is inexpensive.

Like the `MATLAB` programs for solving stiff IVPs, by default `bvp4c` forms Jacobian using finite differences (i.e computes a numerical Jacobian). Often it is to vectorizing the evaluation of the ODE function for a $f(x, y)$ for a given value x and many array of vectors y. This can reduce the run time substantially, so the stiff IVP solvers have an option for this. It is a natural option for BVPs too, but important additional gains are possible if

the function is also vectorizing with respect to the values of x.That is since, in discretized the ODEs in the BVP context, all the value of the arguments are known in advance. Generalizing what is done for IVPs there is an option there is an option for us to code the function $f(x, y)$ so that, when given a vector

$$x = [x_1, x_2, ...x_n]$$

and a corresponding array of column vectors

$$y = [y_1, y_2, ...y_n]$$

it returns an array of column vectors

$$[f(x_1, y_1),\ f(x_2, y_2)\ ...f(x_n, y_n)]$$

Vectorizing the ODE functions in this way often greatly reduce the run time. Most BVP solvers require you to provide analytical partial derivatives. Generally this is both inconvenient and unnecessary for the typical problem solved in `MATLAB`, so `bvp4c` does not. There may be good reasons for going to the trouble of providing analytical partial derivates. One of that it may not be much trouble after all.

For instance, linear ODEs can be written in the form

$$y'(x) = J(x)y(x) + q(x)$$

and it is generally convenient to supply a function for evaluating the Jacobian $J(x)$ is that BVPs are generally solved much faster analytical partial derivatives Although the `numejac` function that `bvp4c` uses to approximate partial derivates numerically is very effective,the task is a difficult one and so analytical partial derivatives improve the robustness of the solver. For some difficult BVPs, supplying analytical partial derivatives is the difference between success and failure.

The code `bvp4c` permits us to supply analytical partial derivatives for either the ODEs or the boundary the boundary conditions or both. It is far more important to

provide partial for the ODEs than the boundary conditions. We inform the solver that you have written a function for evaluating $\frac{\partial\ f}{\partial\ y}$ by providing its handle as the value of the `FJacobian` option. Similarly, we inform the solver of a function for evaluating analytical partial derivatives of the boundary conditions with option `BCJacobian`.The function for evaluating the residual $g(ya, yb)$ in the boundary conditions involves two vectors, the approximate values y_a, y_bat the two ends of the interval. Accordingly, the function we write for evaluating partial derivatives of the boundary conditions must evaluate and return two matrices, $\frac{\partial\ g}{\partial\ ya}, \frac{\partial\ g}{\partial\ yb}$.

A few comments about working out the Jacobian

$$J = \frac{\partial f}{\partial y} = \left(\frac{\partial f_i}{\partial y_j}\right)$$

and coding it as a function may be helpful.

Suppose there are d equations. If J is sparse we should initialize it to a sparse matrix of zeros with $J = sparse\ (d, d)$. For each value of $i = 1, 2, ...d$ we might then examine $f_{\ i}$ for the presence of components y_i and the code the function to evaluate it as $J(i, j)$. Of course, we can evaluate the components of J in any order that is convenient.This is a reasonable way to proceed even when we treating J as a dense matrix. The only difference is that we would initialize the matrix with

$$J = zeros(d, d)(\text{or } J = zeros(d))$$

The function we write for $f(x, y)$ must return a column vector f. A systematic way of working out partial derivates when the matrix is not sparse is to proceed one column at a time. That is, for $j = 1, 2, ...d$ work out the partial derivates

$$J(i, j) = \frac{\partial\ f}{\partial\ y_j} = \left(\frac{\partial f_i}{\partial y_j}\right)$$

Remark 66 *Once we have worked out expression for all the partial derivatives, we can evaluate them in any convenient order.*

The `MATLAB` Symbolic Toolbox has a function `jacobian` that can be very helpful when working out partial derivatives for complicated functions. We find it convenient to use

names like `dBCdya` when providing analytical partial derivates to remind ourselves wich partial derivatives which partial derivatives are being computed. With the help entry for `jacobian` and the variable names we have chosen the program should be easy to follow.

Example 67 *[67, pag 186]*

syms res ya1 ya2 yb1 yb2 alpha beta infty

res = [ya2 /(ya1-1) -alpha

*yb1/exp(beta*infty)-1];*

dBCdya = jacobian (res, [ya1;ya2])

dbcdyb = jacobian (res, [yb1;yb2])

This program generates the output:

$$dBcdya = \begin{bmatrix} -ya2/(ya1-1)\hat{}2 & 1/(ya1-1) \\ 0 & 0 \end{bmatrix}$$

$$dBcdyb = \begin{bmatrix} 0 & 0 \\ 1/\exp(beta * \inf ty) & 0 \end{bmatrix}$$

When a program was run with default values for all options except for `FJacobian,` the BVP was solved for example *(6.6)* in 4.72 *seconds*, a fair improvement over the 5.76 *seconds* without this options. When `BCJacobian` was also supplied, the BVP *(6.6)* was solved in 4.56 *seconds*. Sometimes vectorization reduces the run time more than supplying analytical Jacobian: other times, less. The options are independent, so you can do one or the other depending on what is convenient.

Because the cost depends strongly on the mesh points, `bvp4c` tries to use as few as possible. Nevertless, the goal is to solve the problem and just to solve it with the fewest points possible, so for the sake of robustness `bvp4c` is cautious about discarding mesh points. Indeed, if the initial mesh and the guess for the solution on this mesh are satisfactory, it will not discard any. Notice that in *(6.6)* we specify a value of `infty` that depends on the wave speed c.

It would seem natural simply to use a value large enough to solve the BVP for all the

values of c to be considered. However, if we set `infty=250` the the computing will fail with the message

```
?? Error using ==>bvp4c Unable to refine the mesh any further --- -
the Jacobian of the collocation equations is singular
```

For the smaller values of c the solution approaches its limit values rapidly , so rapidly that with default tolerances, the solver cannot recognize the proper behavior at $Z = 250$.

For instance, when $c = 5$,

$$u(250) \approx e^{250\beta} \approx 2.1846 \cdot 10^{-23}$$

This is too small for default tolerances, but by choosing

$$\inf ty = 10 * c$$

instead we obtain

$$u(50) \approx 2.9368 \cdot 10^{-5}$$

a value more comparable in size to the tolerances.

Remark 68 *When the various kinds of solution behavior cannot be distinguished in the discretized, the linear system is singular. When you receive this kind of error message, we need to consider whether you have the boundary conditions is to deal analytically with the solution where it is difficult to approximate numerically.*

Remark 69 *Hence, we must take* ***`infty large`*** *enough that the asymptotic boundary condition properly describes the solution and yet not so large that we defeat the purpose of the asymptotic approximation.*

The code writing with `bvp4c` of the problem *(6.6)* is presented in chapter 5 section III.

6.2 Boundary Conditions at Singular Points

In this section we discuss singularities at finite points. They often occur in the reduction of a PDE to an ODE by cylindrical or spherical symmetry. For instance, *Bratu's equation,*

$$\Delta y + e^y = 0$$

is used to model spontaneous combustion. In this cases the interval is $[0, 1]$ and the ODE is

$$y'' + k\frac{y'}{x} + e^y = 0 \tag{6.1}$$

with $k = 1$ for cylindrical symmetry and $k = 2$ for spherical symmetry. The obvious difficulty of (6.1) is the behavior of the term y'/x near $x = 0$.Indeed, because of symmetry we must have

$$y'(0) = 0$$

so the term is indeterminate rather the infinite. The standard theory of convergence has been extended to BVPs like this for all the popular numerical methods [37]. For the methods that do not evaluate the ODE at $x = 0$ solving a BVP like (6.1) is straightforward. Obvioulsly it is not straightforward for methods that do evaluate at $x = 0$ such as the method of `BVP4c`. However, some particular solver can deal with singularities at finite points.

The idea is first to approximate the solution near the singular point by analytical means. The means employed depend very much on the problem and what you expect of the solution on physical grounds [11]. For the example (6.1) we expect a smooth solution that we can expand in a *Taylor* series. Taking into account symmetry this series has the form

$$y(x) = y(0) + \frac{y''(0)}{2}x^2 + \frac{y^{(4)}(0)}{4!}x^4 + ...$$

Whatever the assumed form of the solution, we substitute it into the ODE. Using the boundary conditions at $x = 0$ we then determine the coefficients so as to satisfy the

equations to as high an order as possible as $x \to 0$. For example

$$(y''(0) + ...) + \frac{k}{x}(y''(0)x + ...) + e^{y(0)+\cdots} = 0$$

Equating to zero the leading (constants) terms, we find that we must take

$$y''(0) = -\frac{e^{y(0)}}{k+1}$$

If we drop the higher -order terms of the *Taylor* series, this is already enough to provide a good approximation to the solution $y(x)$ on an interval $[0, \delta]$ for a small $\delta > 0$.The derivative of this approximations provides an approximation to $y\,'(x)$. We now solve the ODEs numerically on the interval $[\delta, b]$ where the equations are not singular.

Clearly we want the analytical approximate solution on $[0, \delta]$ to agree at $x = \delta$ with the numerical solution on $[\delta, b]$.

For (6.1) this is

$$\begin{aligned} y(\delta) &= p - \frac{e^p}{2(k+1)}\delta^2 \\ y'(\delta) &= -\frac{e^p}{(k+1)}\delta \end{aligned}$$

Here we have written $p = y(0)$ to emphasize that it is an unknown parameter that must be determined as a part of solving the BVP. This example is typical of the way that unknown parameters are introduced to deal with singularities.

A boundary condition tells us how the solution of interest behaves as it approaches an end point. This is not always as simple as approaching a given value. To describe more complex behavior we use the notation

$$f(x) \sim F(x)$$

as $x \to x_0$ to mean

$$\lim_{x \to x_0} \frac{f(x)}{F(x)} = 1$$

The boundary condition

$$y'(0) = 0$$

implies that the solution of *Bratu's problem* tends to a constant as $x \to 0$ but other boundary conditions are also possible for this ODE. To explore this, we'll investigate solutions that are not bounded at the origin. we begin with $k = 1$ and supposing that derivatives of a solution grow more rapidly than the solution itself as $x \to 0$, we might try approximating the ODE with

$$0 = y'' + \frac{y'}{x} + e^y \sim y'' + \frac{y'}{x}$$

Solving the approximating equation [67, pag :141] we find that

$$y(x) \sim b + a\log(x) = b + \log(x'')$$

for constants a and b.We now substitute this analytical approximation into ODE to see when is might be valid

$$y'' + \frac{y'}{x} + e^y \sim -\frac{a}{x^2} + \frac{a}{x^4} + x^a e^b$$

To satisfy ODE asymptotically, we must have $a > 0$. Boundary conditions like

$$y(x) \sim \log(x) \text{ as } x \to 0$$

arise, for example, when solving potential problems with a line charge at the origin. We see now that the ODE has solutions that behave in this way, so this is a legitimate boundary conditions. Similarly, when $k = 2$.

$$y(x) \sim b + ax^{-1}$$

at it is necessary that $a < 0$.

Boundary conditions like

$$y(x) \sim -x^{-1}$$

arise for example when solving potential problem a point charge at the origin.

Shampine observed it is no harder to solve *Bratu's problem* with the boundary condition

$$y(x) \sim \log(x)$$

than with symmetry condition; he use the analytical approximations

$$y(x) \approx \log(x) \text{ and } y'(x) \approx x^{-1} \text{ on } [0, \delta]$$

for some small δ and solve numerically the ODEs on $[\delta, 1]$ with boundary value $y(\delta) = \log(\delta)$. Note that with particular boundary condition there is no unknown parameter $y(0)$ as there was with the symmetry condition.

Bender and Orszag [11, pp.170-2] discuss the solution of the ODE

$$yy'' = -1 \tag{6.2}$$

with boundary conditions

$$y(0) = 0,\ y(1) = 0.$$

If $y(x) \to 0$, as x tends to 0 and 1, then the ODE requires $y"(x)$ be unbounded there. Clearly we must supply a solver with must supply a solver with more than just limit value for $y(x)$.They observe that the solution is symmetric about $x = 0.5$, so we need only investigate the behavior of the solution near the origin. Since the symmetry we expect two solutions of BVP if $y(x)$ is the solution then so is $-y(x)$.

The solution vanishes at the origin and they are infinite, so we try something simple like

$$y(x) \sim ax^b,$$

for constants a, b.

Substituting into the ODE, we find that

$$-1 = y(x)y''(x) \sim (ax^b)(b(b-1)x^{b-2}) = ab(b-1)x^{2b-2}.$$

If the power $2b - 2 < 0$, then the right-hand side does not have a limit as $x \to 0$. If $2b-2 = 0$ the right-sides are identically zero, and if $2b-2 > 0$ there is a limit it is zero.

Thus we see that there is no choice that allows us to satisfy the ODE, even asymptotically. This tell us that the form we have assumed for the solution is wrong. If we try the form

$$y(x) \sim ax(-\log(x))^b.$$

Substituting this form into ODE, we find after some manipulation that

$$-1 = y(x)y''(x) \sim -a^2 b(-\log(x))^{2b-1}[1-(b-1)(-\log(x))^{-1}]$$

If the right-hand side is to have a limit as $x \to 0$ then we must have the power $2b-1=0$, hence $b=0.5$. With this we pass to the limit find $a=\pm\sqrt{2}$, it appears that are exactly two solutions. The one is positive for positive x is

$$y(x) \sim x\sqrt[2]{-2\log(x)}$$

In writing this second-order equation as the first-order system for its numerical solution we introduce $y'(x)$ as an unknown. Because it is infinite at the origin, we need to know how it behaves there if we are to solve the BVP. On the other hand, all positive solutions of the ODE that vanish at the origin behave in the same way to leading order, so it is not necessary to introduce an unknown parameter if we we are interested only in modest accuracy.

6.2.1 Examples

1.Latzko's equation

If the solution of a BVP behaves well at a singular point, it may not be necessary to use an analytical approximation in the neighborhood of the point. To illustrate this we consider a *Sturm-Liouville* eigenproblem called *Latzko's equation*

$$\frac{d}{dt}\left((1-x^7)\frac{dy}{dx}\right)+\lambda x^7 y = 0$$

with boundary conditions

$$y(0)=0,\ y(1) \text{ finite.}$$

Scot [71] collects approximations to the first three eigenvalues and compares them to his approximations: $8,728; 152,45; 435,2$. To write this equation as a first-order system,

it is convenient to use the unknowns $y(x), v(x) = (1 - x^7)y'(x)$ and so to obtain

$$\begin{aligned} y' &= \frac{v}{1 - x^7} \\ v' &= -\lambda x^7 y \end{aligned}$$

This is an eigenvalue problem, so we must specify a normalizing condition ($y(1) = 1$) for the solution. The differential equation is singular at $x = 1$. If we are looking for a solution with $y'(x)$ finite, the first equation of the system states that we must have $v(1) = 0$. Using the second equation, *l'Hôpital* rule then says that

$$y'(1) = \lim_{x \to 1} \frac{v(x)}{1 - x^7} = \lim_{x \to 1} \frac{v'(x)}{-7x^6} = \lim_{x \to 1} \frac{-\lambda x^7 y(x)}{-7x^6} = \frac{\lambda}{7}$$

We can solve the ODEs for $y(x)$ and $v(x)$ subject to the boundary conditions

$$y(0) = 0, y(1) = 1, v(1) = 0$$

provided that we evaluate them properly at $x = 1$.

When solving a BVP with an unknown parameter, `BVP4C` calls the function for evaluating the ODEs with a mesh point x and the current approximations to $y(x)$ and λ.

If $x = 1$, this functions must return the limit value for $y'(1)$. We must code for this solver always evaluates the ODEs at the end points of interval. With the smooth solution, the solver will not need to evaluate the ODEs at points x so close to the singular point that there are difficulties evaluating $y'(x)$, difficulties that we avoid for other kinds of problems by using analytical approximations near the singularity.

Of course, the guess for $v(x)$ should have proper behavior near $x = 1$. This is easily accomplished by using the guess for $y(x)$ to form

$$(1 - x^7)y'(x)$$

as a guess for $v(x)$.

A different way to perform this analysis is to look for a solution

$$y(x) \sim 1 + c(x - 1) \text{ as } x \to 1$$

If there is a solution of this form , then $y'(x) \sim c$. Because $y'(1)$ is finite for a solution of this form, it follows that $v(1) = 0$. Determine the constant c by substituting this assumed form into the `ODE` for $y(x)$. In this it be will helpful to show that $(1 - x^7) \sim 7(1 - x)$ near $x = 1$. If the function for evaluating the `ODEs` is coded properly then it is easy enough to solve this eigenproblem with `BVP4C`. What is not easy to compute specific eigenvalues and eigenfunctions. This is because it is not clear which solution you will compute with given guesses for $y(x)$ and λ or whether even you will compute a solution. Using `BVP4C` we try to confirm one or more of *Scott's* approximate eigenvalues.

If we used the guess [67, pag 144]

$$y(x) \approx x \sin(2.5\pi x)$$

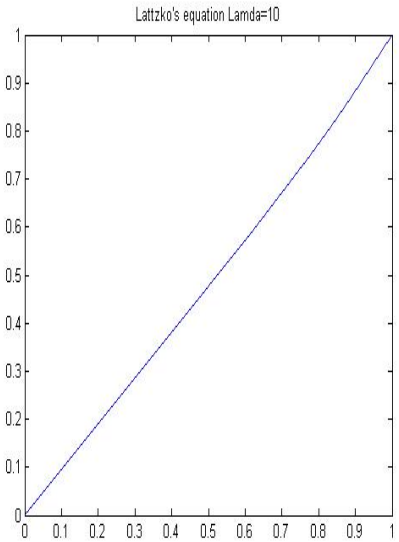
and an initial mesh of `linspace (0,1,10)`yielded convergence to different eigenvalues for guesses $\lambda = 10, 100, 500, 1000, 5000, 10000$; these computed eigenvalues include the first three

We write codes in `MATLAB` and obtain the following results:

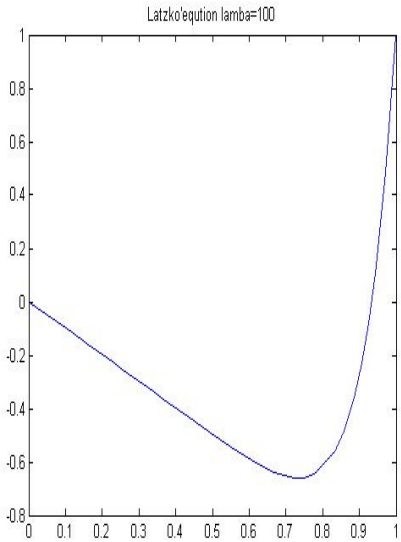
```
> ex1

Guess for eigenvalue: 10

Computed the eigenvalue 8.72727.

> ex1

Guess for eigenvalue: 100

Computed the eigenvalue 152.39.

> ex1
```

```
Guess for eigenvalue: 500

Computed the eigenvalue 435.022.

> ex1

Guess for eigenvalue: 1000

Computed the eigenvalue 855.492.

> ex1

Guess for eigenvalue: 5000

Computed the eigenvalue 2110.51.

> ex1

Guess for eigenvalue: 10000

Computed the eigenvalue 5025.87.
```

We plot them in Figure 6.1(a), 6.1(b), 6.2(a), 6.2(b) 6.3(a), 6.3(b)

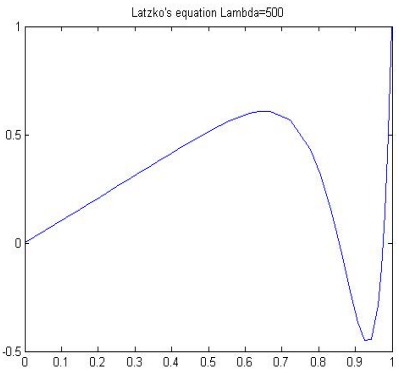


(a) λ=10

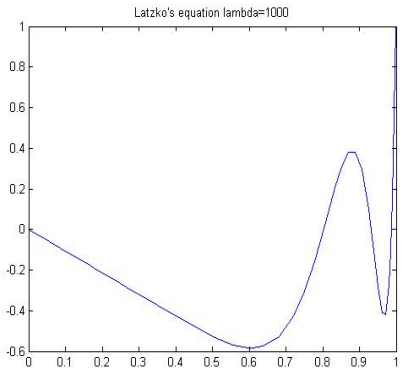


(b) λ=100

Figure 6.1: Eigenvalues -1

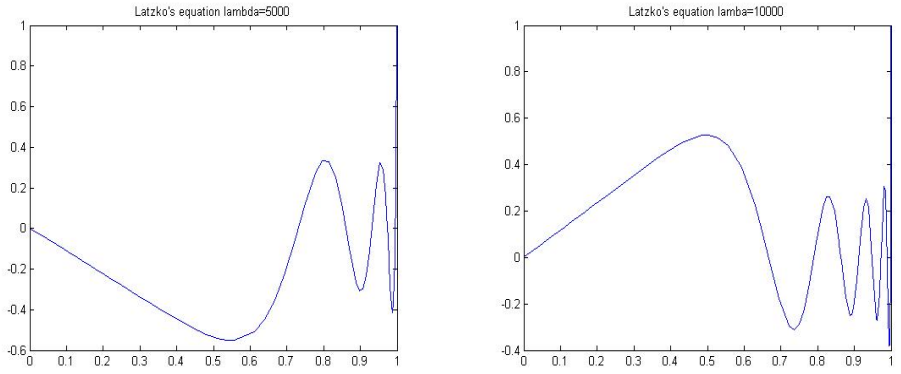


(a) λ=500



(b) λ=1000

Figure 6.2: Eigenvalues -2



(a) λ=5000 (b) λ=10000

Figure 6.3: Eigenvalues -3

2.The steady concentration of a substrate in an enzyme-catalyzed reaction

In [40, section 6.2] Keller discusses the numerical solution of a model of the steady concentration of a substrate in an enzyme-catalyzed reaction with *Michaelis-Menten* kinetics. A spherical region is considered and the PDE is to reduced an ODE by symmetry . The equation

$$y'' + 2\frac{y'}{x} = \frac{y}{\varepsilon(y+k)}$$

involves two physical parameters, ε and k.The boundary conditions are

$$\begin{aligned} y(1) &= 1 \\ y'(0) &= 0 \end{aligned}$$

approximate $y(x)$ on the interval $[0, d]$ with a few terms of a *Taylor* series expansion and we solve `BVP` numerically on $[d, 1]$ using `BVP4C,` where $\varepsilon = 0.1$, $k = 0.1$ and we obtain the following result in the Figure 6.4. To plot on all $[0, 1]$ augment the numerical solution on the interval $[d, 1]$ with the values at $x = 0$ provided by the unknown parameter $p = y(0)$, and $y'(0) = 0$. Two terms of a *Taylor* series for $y(x)$ and one for $y'(x)$ are satisfactory when $d = 0.001$. We will need to communicate d and the parameters ε and

k to our functions for evaluating the `ODEs` and boundary conditions (see the code in next chapter).

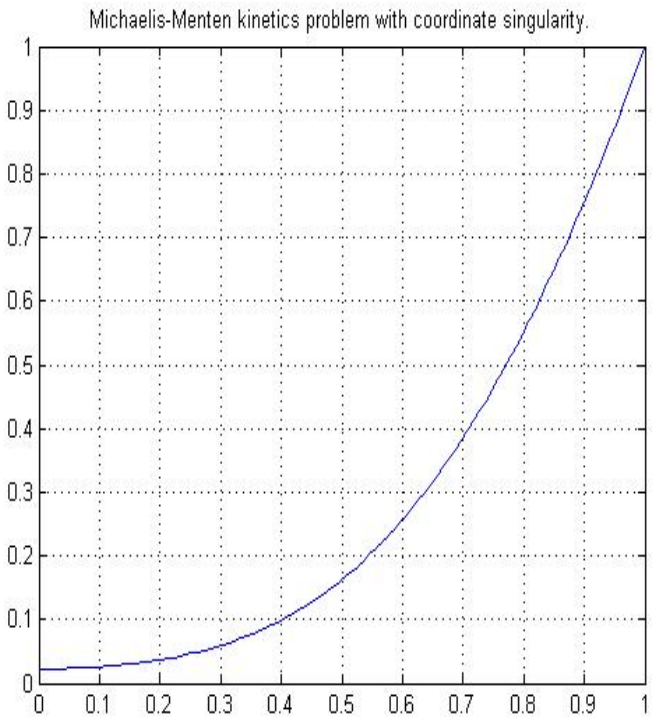


Figure 6.4: The steady concentration of a substrate in an enzyme-catalyzed reaction

3.Bender and Orszag problem

Bender and Orszag (1999) in the book [11] propose to solve the BVP:

$$\begin{aligned} yy'' &= -1 \\ y(0) &= 0 \\ y'(0.5) &= 0 \end{aligned}$$

They shows that

$$y(x) \sim x\sqrt{-2\log(x)}$$

as $x \to 0$.It will be convenient to code a subfunction to evaluate the approximation

$$v(x) = x\sqrt{-2\log(x)}$$

Move the boundary condition from the singular point at the origin to, say, $d = 0.001$ by requiring that $y(d) = v(d)$.We compute $y(x)$ on $[d, 0.5]$ using `BVP4C`,with default tolerances and guesses of

$$\begin{aligned} y(x) &= x(1-x) \\ y'(x) &\approx 1 - 2x \end{aligned}$$

We plot both $y(x)$ and $v(x)$ with `axis([0 0.5 0 0.5]` and obtain the Figure 6.5

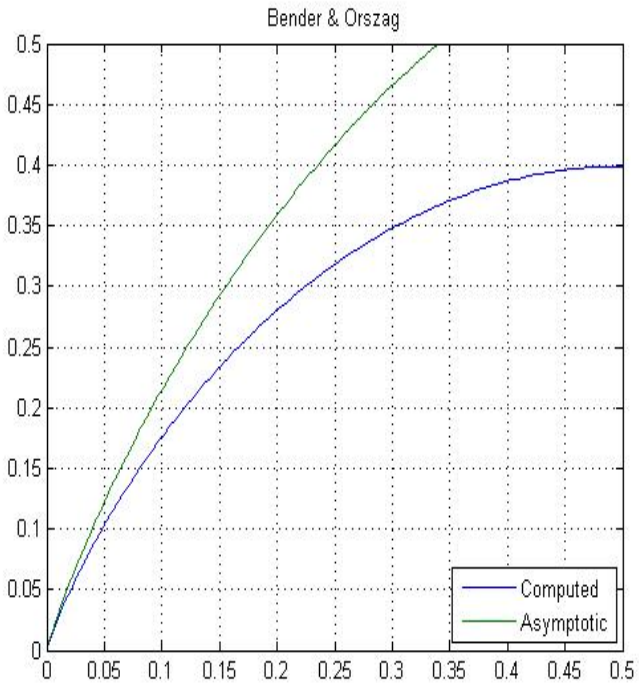


Figure 6.5: Bender-Orszag equation

6.3 Boundary Conditions at Infinity

Let a system of linear ODEs with constant coefficients:

$$y' = Jy + q \tag{6.3}$$

to give more insight. For the sake of simplicity, let us suppose that the *Jacobian matrix* J is nonsingular. This implies that

$$p(x) = -J^{-1}q$$

is a constant particular solution of the ODE. Further assume that J has a complete set of eigenvectors $\{v^{\,j}\}$ with corresponding eigenvalues $\{\lambda_j\}$.

This that are constants α_j such that

$$y(a) - p(a) = \sum \alpha_j v^{\,j}$$

It is the verified easily that the general solution of the system ODEs is

$$y(x) = p(x) + \sum \alpha_j e^{\lambda_{\,j}\,(x-a)} v^j$$

We see from this that $y(x)$ is finite on $[a, +\infty)$ only when the boundary conditions at $x = a$ imply that $\alpha_m = 0$ for all m with $\mathrm{Re}(\lambda_m) > 0$. Correspondingly, the BVP is well-conditioned for $b > a$ only when the boundary conditions at $x = a$ exclude the terms that grow exponentially fast as x increases. Similarly, using the expansion

$$y(b) - p(b) = \sum \beta_j v^{\,j}$$

we see that if $b > a$ then we must have boundary conditions at $x = b$ that implies $\beta_m = 0$ for any value m such that $\mathrm{Re}(\lambda_m) < 0$, if the BVP is to be well-conditioned. Roughly speaking, components that decay rapidly from left to right must be determined by the boundary conditions at the left end of the interval, and components that grow rapidly on this direction must be determined by the boundary conditions at the right end.

These linear, constant coefficient problems are rather special, but they provide valuable insight. More generally we recognize that the boundary conditions the various kinds of behavior that solutions can have. If there are solutions of `ODE` that approach one another very quickly as x increases from a to b, then for $b > a$ a code will not able to distinguish the solutions numerically when applying the boundary conditions at $x = b$. Because of this, these solutions must be distinguished by boundary conditions at the point $x = a$ rather at $x = b$.

There is a *dichotomy* of the solution space of the `ODE` that must be reflected in the placement of the boundary conditions. This is easy enough to sort out for linear, constant-coefficients ODEs, but for more general `ODEs` some careful physical reasoning is normally needed to decide where and what kind of boundary conditions are appropriate [6].

6.3.1 Examples

1. The charge density in atoms of high atomic number.

Global collocation method with B-splines can not be applied for the solution of non-linear problems which has singularities at the end points of the interval $[0, 1]$. In this case we must choose the combined B-spline with Runge-Kutta methods. Since near the end points we have singularities we chose the solver `ODE23tb` for Runge-Kutta methods. We exemplified this by *Fermi-Thomas* problem (1.81)with boundary conditions (1.82).

It should be noted that this problem has been studied by *Shampine, Gladwell and Thomson* [68, pp 155-156] using the solver `bvp4c.` We use for start solution

$$y(x) = 144x^{-3},$$

and for inner points $d = 0.015$, and $e = 59$.

Method	Run-Times
Shampine-Gladwell-Thomson	0.838735 seconds.
B-spline and Runge-Kutta	0.493448 seconds.

The graph of approximate nonlinear solution of *Fermi-Thomas* problem is presented in Figure 6.6.

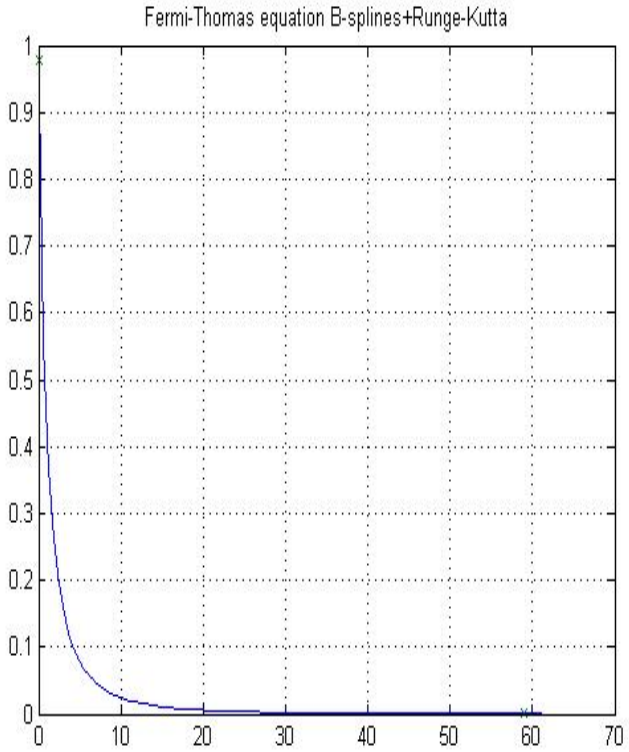


Figure 6.6: Thomas-Fermi equation

Remark 70 *The running time for B-spline collocation is the shortest, since its collocation matrix is banded.*

2. Models for a transport, reaction and dissipation of pollutants in rivers.[37],[25]:

A natural choice of the start solution for this problems is:

$$h(t) = \frac{1}{(t+1)^3},\ f(t) = \frac{1}{(t+1)^3}$$

since:

$$h(0) = 1,\ h(\infty) = 0,$$
$$f(0) = 1,\ f(\infty) = 0.$$

We obtained the approximation solutions of the problem (1.87) with conditions (1.86), like:

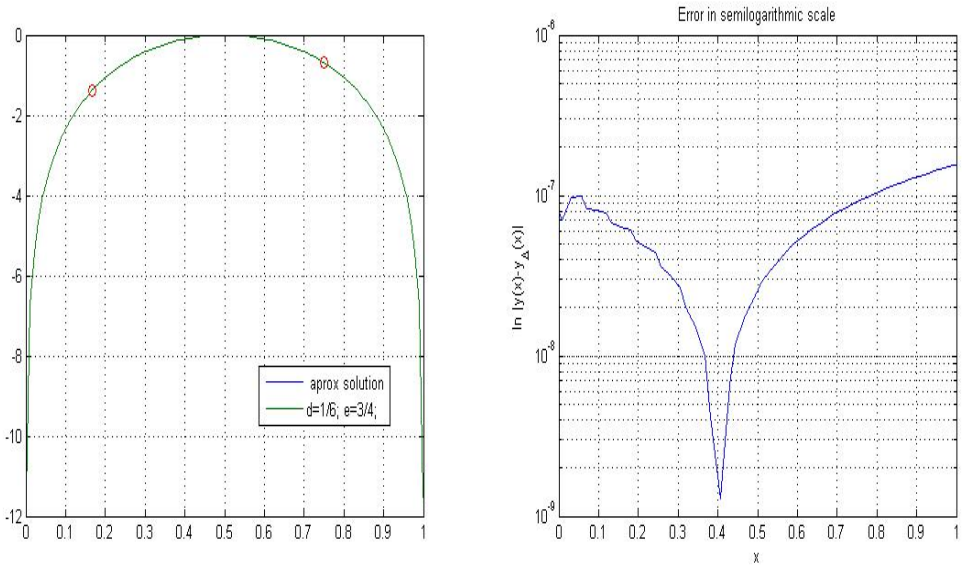
1. On the interval $[0,\ 10]$ with *B-splines functions* of order $(k+1)$,
2. And on $[10,\ \delta)$ with *Runge-Kutta* methods ($k-$stages), solver `ode113` or `ode45`.
3. On the interval $[0,\ 10]$ with *B-splines functions* of order $(k+1)$,
4. And on $[10,\ \delta)$ with *Runge-Kutta* methods ($k-$stages), solver `ode113` or `ode45`.

Remark 71 *The convergence of these methods is given in ([31]).*

We obtained the following results depicted in fig 6.7(a), fig:6.7(b). In the paper Solving ODE's with Matlab (pag 153) *Shampine, Gladwell and Thomson* , used `BVP4C,` to solve these types of problems. Since they used 5 iterations to obtained the approximate solution, our method is faster.

We compute (`tic-toc MATLAB functions`) the run _times and we conclude:

Problems	Method	Tolerance	Run_times
(1.87) + (1.86)	$B\_splines + Runge\_Kutta$	10^{-15}	0.990373 *seconds*,
(1.87) + (1.86)	$BVP4C$	10^{-15}	6.9354 *seconds*.



(a) Approximate solution (b) Errors in semilogarithmic scale

Figure 6.7: Models for a transport, reaction and dissipation of pollutants in rivers

3. Costabile problem

We consider the two-points boundary values linear problem:

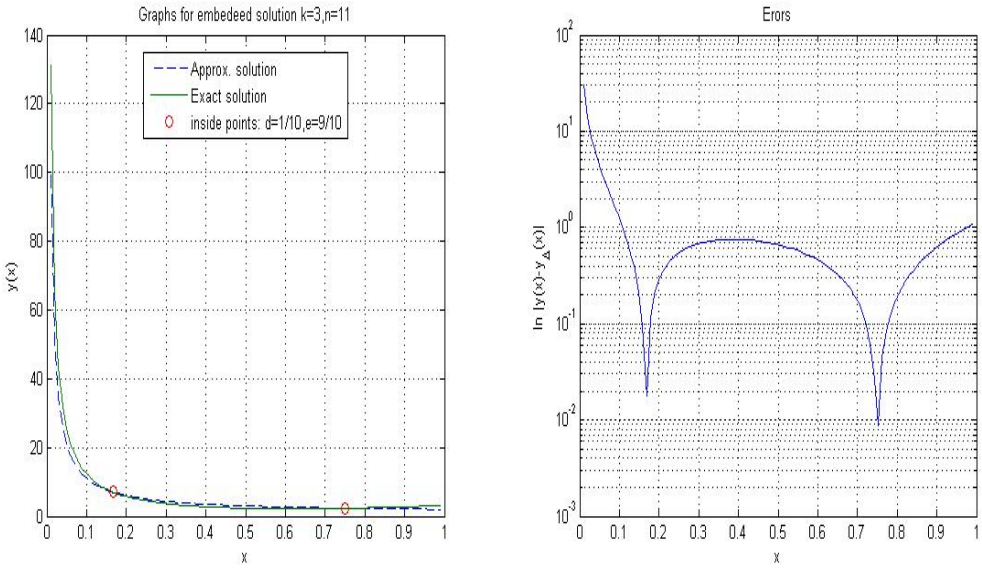
$$\begin{cases} y'' = \frac{2}{x^2}(y-1), & x \in (0,1) \\ y\left(\frac{1}{4}\right) = 5,\ y\left(\frac{1}{2}\right) = 3. \end{cases} \tag{6.4}$$

The above problem has the exact solution:

$$y(x) = 1 + 1/x. \tag{6.5}$$

Using the step control algorithm ([1, pp: 376-380], [78]), we set that the exact solution has a singularity in $x = 0$. For $N = 11$ and $k = 3$, we got the approximate solution and we represent it in Figure 6.8(a) and the errors in Figure 6.8(b). We wrote the programs in `MATLAB 10a` and run times given by functions `tic-toc`:

```
Elapsed time is 0.150000  seconds.
```



(a) Approximate solution (b) Errors in logarithmic scale

Figure 6.8: Singularity in $x = 0$

4. Flame propagation in nuclear reactors [6, pag 148-151].

Let BVP:

$$u" + cu' + u(1-u) = 0 \tag{6.6}$$

with boundary conditions:

$$u(-\infty) = 1,\ u(\infty) = 0$$

Using the solver `BVP4C [67, 148-151]` we obtain the following results for $c \in \{5, 20, 35, 50, 65, 80, 100\}$.There are depicted in figure 6.9

Remark 72 *By examine trajectories in the phase plane,* Murray [50] *argues that if* $c \geq 2$ *then there exists a solution of BVP 6.6 such that, on approach to the point* $(0, 0)$

$$u'(x) \sim \beta\ u(x)$$

where $\beta = (-c + \sqrt{c^2 - 4}/2)$ *and on approach to the point (1,0)*

$$(u(x) - 1)' \sim \alpha(u(x) - 1)$$

with $\alpha = (-c + \sqrt{c^2 + 4}/2)$.

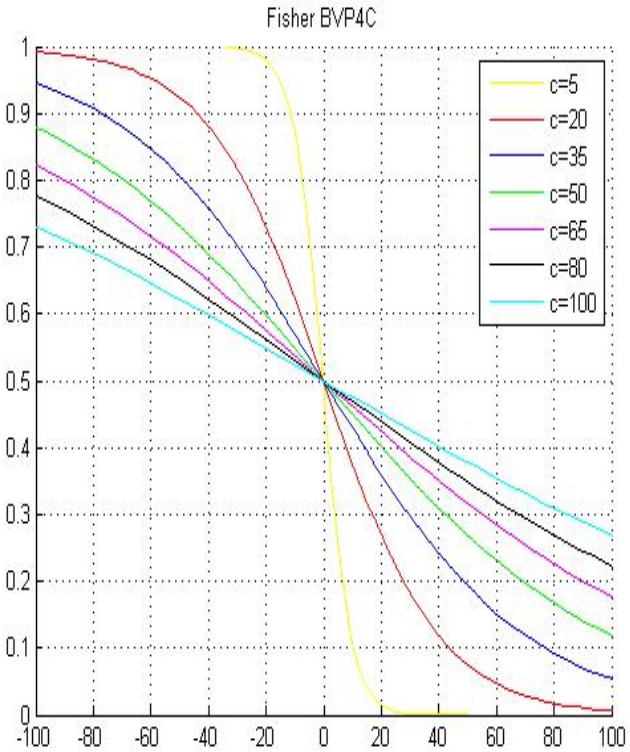


Figure 6.9: Fischer equation

5. The unsteady flow of gas through a semi-infinite porous medium initially filled with gas at a uniform pressure.

Bayley et al consider the following `bvp` [9]:

$$w''(x) + \frac{2x}{\sqrt{1-\alpha w(x)}} w'(x) = 0$$

with boundary conditions

$$w(0) = 1,\ w(+\infty) = 0$$

The range of values of the parameter is $0 < \alpha < 1$ and we solve using `BVP4C` for $\alpha = 0.8$.*Shampine* solve this problem by replacing the boundary conditions at infinity with the boundary condition $w(x) = 0$ for some finite value x.Our guess for $w(x)$ should respect the physical requirements that

$$0 \leq w(x) \leq 1.$$

To understand better what is goings on, , we first observe that because $w(+\infty) = 0$, the ODE is approximately

$$w''(x) + 2xw'(x) = 0$$

for large x. Solving this approximating equation, we find that

$$w'(x) \sim \beta e^{-x^2}$$

Integrating and imposing the boundary condition at infinity, we then find that

$$w(x) \sim -\beta \int_x^\infty e^{-t^2} dt = (-\frac{\beta \sqrt[2]{\pi}}{2}) \operatorname{erf} c(x)$$

The standard asymptotic representation of the complementary error function

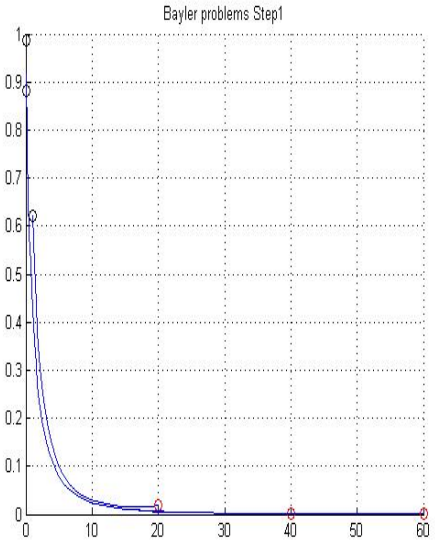
$$\operatorname{erf} c(x) \sim \frac{e^{-x^2}}{x \sqrt[2]{\pi}}$$

shows that $w(x)$ approaches its boundary value $w(\infty) = 0$ very quickly. That is why we can impose the numerical conditions $w(x) = 0$ at what seems like a very small value

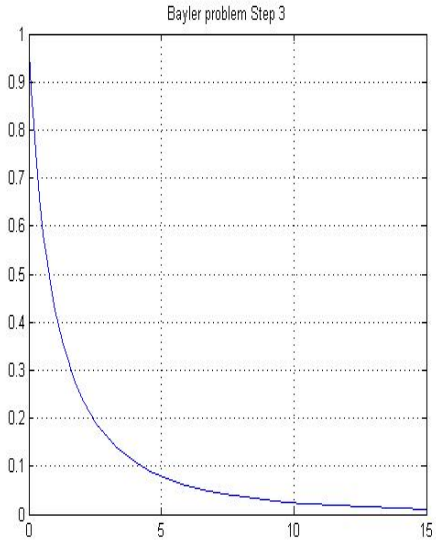
of x. In fact we must use a small value x since the solver cannot distinguish $w(x)$ from the identically zero solution of the ODE when x is large. Sometimes we must supply more information about how a solution behaves near a singular point if we are to compute a numerical solution at all. For this BVP, we could do that by treating the constant β as an unknown parameter and imposing the two numerical boundary conditions

$$\begin{aligned} w'(x) &= \beta e^{-x^2} \\ w(x) &= (-\frac{\beta\sqrt{\pi}}{2}) \operatorname{erf} c(x) \end{aligned}$$

instead of $w(x) = 0$. The analytical approximation to $w(x)$ and $w'(x)$ are accurate only for large x but they provide guesses for the solver for all x. We need two step to resolve this problem, and we plot the graphs in a Figure 6.10(a) and 6.10(b).



(a) Bayley problem-Step1



(b) Bayley problem-Step2

Figure 6.10: The unsteady flow of gas through a semi-infinite porous medium initially filled with gas at a uniform pressure.

Chapter 7

Matlab and Maple Codes

The first ODE solver of MATLAB was based on a FORTRAN programs written by *Carl de Boor* [10], *Lary Shampine* and *H.A. (Buddy) Watts* [67]. Also *Carl de Boor* write a library in MATLAB named Spline Toolbox([12], [13]). In this library there is a function *spcol,* which provide us a simple way to calculate the inverse of B-splines collocation matrix. The theoretical results on the convergence of collocation method ([14], [15]), together with those on error estimation and mesh selection [64], are more general than for the other methods. This, and the basic simplicity of the collocation procedure [22], also make programming of the method reasonably straightforward. *U. Ascher, S. Pruess* and *R.D. Russell* ([4], [5], [7]), shows that for low continuity piecewise polynomials, some of B-splines and their derivatives are largely mesh independent and hence cheap to evaluate. Unfortunately, the mesh matrices have condition numbers which grow very rapidly as the mesh is refined. The ODE Suite has evolved considerably as a result of further work by *L. Shampine, J. Kierzenka* and *M.W.Reichelt* [69]. *L.N. Trefeten* introduces in his monograph [76], some fairly useful MATLAB codes which implements spectral method. *Weideman* and *Reddy* in their paper present [80], a software suite which consists of 17 MATLAB functions for solving differential problems by the spectral collocation methods. These functions enable us to generate spectral differential matrices based on *Chebychev*

interpolantes, and we use these functions in implementation of our $C.C$ methods.
`MAPLE` is a general-purpose commercial computer algebra system. It was first developed in 1980 by the Symbolic Computation Group at the University of Waterloo in Waterloo, Ontario, Canada.

Since 1988, it has been developed and sold commercially by Waterloo Maple Inc. (also known as Maplesoft), a Canadian company also based in Waterloo, Ontario. The current major version is version `Maple` 15 which was released in April 2011. Users can enter mathematics in traditional mathematical notation. There is extensive support for numeric computations, to arbitrary precision, as well as symbolic computation and visualization.Maple is based on a small kernel, written in C, which provides the Maple language. Most functionality is provided by libraries, which come from a variety of sources. Many numerical computations are performed by the NAG Numerical Libraries, ATLAS libraries, or GMP libraries. Different functionality in Maple requires numerical data in different formats. Symbolic expressions are stored in memory as directed acyclic graphs. The standard interface and calculator interface are written in *Java*. The language permits variables of lexical scope.
However, in this **Section** we introduce some `MATLAB` and `MAPLE` codes which implement **Global B-splines methods, Combined methods based on B-splines and Runge-Kutta, and C.C methods combined with Runge-Kutta methods.**

Remark 73 *The following programs solve also the classical problems(BVP), if $a = d$ and $e = b$ but might apply to delay differential equations (DDE).*

7.1 Linear Case

7.1.1 MATLAB Codes

=======

```
%Linear case
%Standard equation :y"+q(x)y+r(x)=0, a<x<b
%Inner conditions: y(d)=alpha, y(d)=beta, a<d<e<b
%number of collocation points N=nk+2, k=degree of B-splines


q = @(x) -ones(size(x));
r = @(x) x;
%exact solution
solex= @(x) -x+sin(x)/sin(1);
a=0; b=1;
d=1/4;
e=1/2;
alpha=solex(d);
beta=solex(e);
t=linspace(a,b,100);
n=36;
k=3;
tic;
[x,y]=BVPcollocRK(q,r,a,b,d,e,alpha,beta,n,k,t);
toc;
figure(1)
plot(t,solex(t),x,y,[d,e],[alpha,beta],'o')
title('PVP problems .... ')
```

```
legend('Approximate solution','d=1/4,e=1/2',0)
grid on
figure(2)
semilogy(x,abs(solex(x)-y),'-')
title('Errors in semilogarithmic scale ')
grid on
```

=======

```
function [x,y]=BVPcollocRK(q,r,a,b,d,e,alpha,beta,N,k,t)
%BVPCOLLOCRK - collocation for linear polylocal problem
%-y"(x)+q(x)*y(x)=r(x);
%call [sp]=BVPcolloc(q,r,a,b,alpha,beta,N,k)
%q,r - functions, define BVP
%a,b - endpoints
%d,e - inner points
%alpha, beta - values at endpoints
%N - number of subintervals
%k -degree of B-splines
%t - evaluation points
%y - solution values at t
%ode23tb for stiff problem and ode45 for nonstiff
%prepare meshes
ti=t(t>=d & t<=e);
tleft=t(t<d);
tright=t(t>e);
%solve BVP
sp=BVPcolloc(q,r,d,e,alpha,beta,N,k);
yi=fnval(sp,ti);
```

```
ydi=fnval(fnder(sp),[d,e]);
x=ti';
y=yi';
%solver options
opts=odeset('AbsTol',1e-8,'Reltol',1e-7);
%solve right IVP
if ~is empty(tright)
    tright=[e,sort(tright)];
    [tr,wr]=ode23tb(@rhs,tright,[beta,ydi(2)],opts);
    x=[x;tr(2:end)];
    y=[y;wr(2:end,1)];
end
%solve left IVP
if ~is empty(tleft)
    tleft=[d,sort(tleft,'descend')];
    [tl,wl]=ode23tb(@rhs,tleft,[alpha,ydi(1)],opts);
    x=[tl(end:-1:2);x];
    y=[wl(end:-1:2,1);y];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nested function for RK method
    function dy=rhs(x,u)
        dy=[u(2); q(x)*u(1)-r(x)];
    end
end
```

7.1.2 MAPLE codes

=======

Cubic spline collocation method- uniform mesh

```
>#-y"+Pi^2*y=2*Pi^2*sin(Pi*x);0<=x<=1;
>#y(1/4)=sqrt(2)/2;y(1/2)=1;
>#R.Burden,J.D.Faires
>#Numerical Analysis-pag 558
>with(CurveFitting):
>S:=x->eval(BSpline(4,t,knots=[-2,-1,0,1,2]),t=x):
> plot(S(x),x=-2..2,title = "Cubic B-spline"):
> n:=3:h:=1/(n+1):
> x:=Array(0..n+1,[seq(i/(n+1),i=0..n+1)]):
> F:=proc(t,x,i,n,h)
> if i=0 then
> return S(t/sqrt(h))-S((t+h)/sqrt(h));
> elif i=n+1 then
> return S((t-(n+1)*h)/sqrt(h))-S((t-(n+2)*h)/sqrt(h)):
> else
>    return S((t-x[i])/sqrt(h)):
> end if;
> end proc:
> F1:=proc(t,x,i,n,h)
> elif i=1 then
> return S((t-x[1])/h)-S((t+h)/h):
> elif i=n then
> return S((t-x[n])/h)-S((t-(n+2)*h)/h):
```

```
> else
> return S((t-x[i])/h):
> end if
> end proc:
>for i from 0 to n+1 do m[i+1]:=plot(F(t,x,i,n,h),t=0..1):end do:
> with(plots):
> display([seq(m[i],i=1..n+2)]):
>plot(F(t,x,0,n,h),t=0..1):
>for i from 1 to n do m[i+1]:=plot(F1(t,x,i,n,h),t=0..1):end do:
> with(plots):
> display([seq(m[i],i=1..n+2)]):
>plot(F1(t,x,1,n,h),t=0..1):plot(F1(t,x,n,n,h),t=0..1):
genspline:=proc(x,n,q,r,e,d,alpha,beta)
>   local k, ecY,ecd, C, h, Y,  c;
>   global F,S;
>   h:=1/(n+1);
>   Y:=0;
>   for k from 0 to n+1 do
>      Y:=Y+(t-e)*(t-d)*c[k]*F(t,x,k,n,h):
>   end do;
>   Y:=simplify(Y):
> Y:=Y+(t-d)*alpha+(t-e)*beta;
>   ecY:=-diff(Y,t$2)+q(t)*Y=r(t):
>   ecd:=Array(0..n+1);
> for k from 0 to n+1 do
>      ecd[k]:=eval(ecY,t=x[k]):simplify(ecd[k]):expand(%):
>print(ecd[k]);
>   end do;
```

```
>   C:=solve({seq(ecd[k],k=0..n+1)},[seq(c[k],k=0..n+1)]):
>   assign(C):
>   return Y:
> end proc:genspline1:=proc(x,n,q,r)
> local k, ecU,ecd, h, U, b,B;
>   global F1,S:
> h:=1/(n+1):
> U:=0:
> for k from 0 to n+1 do
>      U:=U+b[k]*F1(t,x,k,n,h);
>   end do;
 U:=simplify(U):
>   ecU:=-diff(U,t$2)+q(t)*U=r(t):
>   ecd:=Array(0..n+1);
>   for k from 1 to n do
>      ecd[k]:=eval(ecU,t=x[k]):simplify(ecd[k]): print(ecd[k]):
>   end do:
> ecd[0]:=eval(U,t=e)=alpha:
> ecd[n+1]:=eval(U,t=d)=beta:
>   B:=solve({seq(ecd[k],k=0..n+1)},[seq(b[k],k=0..n+1)]):
>   assign(B):
> return U:
> end proc:
>q:=t->Pi^2: r:=t->2*Pi^2*sin(Pi*t):e:=1/4:d:=3/4:
>ecz:=-diff(Z(t),t$2)+q(t)*Z(t)=r(t):
> dsolve({ecz,Z(0)=0,Z(1)=0},Z(t)):
> Z:=t->sin(Pi*t);Z(e);Z(d):alpha:=Z(e)/(e-d);beta:=Z(d)/(d-e):
>Y:=genspline(x,n,q,r,e,d,alpha,beta):
```

```
>U:=genspline1(x,n,q,r):
>plot(U,t=0..1):
>plot(Z(t),t=0..1,title="Approx-solution",color=[BLUE]):
>plot(Y,t=0..1,title="Exact solution",color=[BLACK]):
>p1:=plot([Y,Z(t),U],t=0..1,title="Exact+Approx solution"):
>p2:=plots[pointplot]([[1/4,Z(1/4)],[3/4,Z(3/4)]],
>symbol=circle,symbolsize=34,color=[GREEN]):
>plots[display]({p1,p2}):
```

B-splines collocation method-nonuniform mesh

```
> restart; with(CurveFitting): with(orthopoly):
>#Here we solve a problem (using B-splines functions)
>#In this situation the error in approximating
>#the exact solution
>#u(x)=sin(p*x) is of order O(h^p).
<#We study the case p=9,n =10.
> Digits:=30:
> genpoints:=proc(a,b,N,k)
> local L,i,j,xu,xc,pol,pol2,sol,h,nL:
> L:=[a]:
> h:=(b-a)/(N+1):
> xc:=a-h; pol:=P(k,t):
> pol2:=expand(subs(t=2*x-1,pol)):
> sol:=fsolve(pol2):
> for i from 0 to N+2 do
>   xu:=xc+h:
>   for j from 1 to k do
```

```
>        L:=[op(L),xc+(xu-xc)*sol[j]]:
>   end do:
>     xc:=xu:
> end do:
> L:=[op(L),b]:
> L:=sort(L): nL:=nops(L):
> return L:
> end proc:
> S:=(x,u,k)->eval(BSpline(4,t,knots=[seq(u[i],i=k-2..k+2)]),t=x):
> genspline:=proc(x,n,q,r,e,d,alpha,beta)
> #x - mesh
> #n - number of points
> #q, r - functions of differential equation
> #e,d - inner points
> local k, ecY,ecd, Y,a0, b0, b,B,u:
>   global S, bcopy:
> Y:=0:
> for k from -1 to n+2 do
>      Y:=Y+b[k]*S(t,x,k):
> end do:
> Y:=simplify(Y):
> ecY:=diff(Y,t$2)-q(t)*Y=r(t):
> ecd:=Array(-1..n+2):
> for k from 1 to n+1 do
>      ecd[k]:=eval(ecY,t=x[k]):simplify(ecd[k]):
> end do:
> ecd[-1]:=eval(Y,t=e)=alpha:simplify(ecd[-1]):
> ecd[n+2]:=eval(Y,t=d)=beta:simplify(ecd[n+2]):
```

```
> B:=solve({seq(ecd[k],k=-1..n+2)},[seq(b[k],k=-1..n+2)]):
> print(nops({seq(ecd[k],k=-1..n+2)}));
> print(nops([seq(b[k],k=-1..n+2)])):
> assign(B):
> for u from -1 to n+2 do bcopy[u]:=b[u]:
> end do:
> return Y:
> end proc:
> q:=t->-86: r:=t->5*sin(9*t):
> ecz:=diff(Z(t),t$2)-q(t)*Z(t)=r(t):
> dsolve({ecz,Z(0)=0,Z(1)=sin(9)},Z(t)):simplify(%):
> assign(%):
> gendiv:=proc(a,b,e,d,alpha,beta,n)
> local h,x,Y:
> h:=(b-a)/(n+1):
> x:=Array(-2..n+3,[seq(a+i*h,i=-2..n+3)]):
> Y:=genspline(x,n,q,r,e,d,alpha,beta):return Y:
> end proc:
> gendivLeg:=proc(a,b,n,k)
> local h,x,Y,L,nn,j:
> L:=genpoints(a,b,n,k);
> L:=convert(L,rational,exact):
> nn:=nops(L)- 2*k; print(nn):
> x:=Array(-k..nn+k-1,L):
> return x;
> end proc:
> b:=1: a:=0: e:=Pi/54: d:=Pi/12:
> alpha:=simplify(eval(Z(t),t=e)):beta:=simplify(eval(Z(t),t=d)):
```

```
> n:=10:alpha:beta:
> plot(Z(t),t=0..1,color=[BLUE],title="Exact solution"):
> k:=3:
> x:=gendivLeg(a,b,n,k):
> nn:=ArrayNumElems(x)-2*k-3:
> bcopy:=Array(-k..nn+2):
> profile(genspline):
> Y:=genspline(x,nn,q,r,e,d,alpha,beta):showprofile(genspline):
> p1:=plot([Y,Z(t)],t=0..1,title=" Approximate solution"):
> p2:=plots[pointplot]([[Pi/54,eval(Z(t),t=Pi/54)],
[Pi/12,eval(Z(t),t=Pi/12)]],symbol=circle,symbolsize=30,
color=[BLACK]):
> plots[display]({p1,p2}):
> plots[logplot](Y-Z(t),t=0.001..0.999,
title="Errors in semi-logarithmic scale",
color=[BLUE],axis=[gridlines=[100, color=blue]]):
> plot([seq(S(t,x,k),k=-1..nn+k)],t=x[-k]..x[nn+k+2],
title=" B-splines bases"):
```

=======

Chebychev Collocation

```
> restart; with(orthopoly):with(CodeTools):with(plots):
>#The (BVP) Greengard and Rokhlin  solve with Collocation
>#  Chebishev method.
>#We study the case $p=9$,$n=10$.
>#In this situation the error in approximating the exact
>#solution u(x)=sin(p*x)is
```

```
>#of order O(10^-8).
> S:=(x,k,a,b)->T(k,((b-a)*x+a+b)/2):
> Digits:=30:
> genceb:=proc(x,n,q,r,c0,d0,alpha,beta)
>   local k, ecY, ecd, C, h, Y, c, a, b;
>   global S;
>   a:=x[0]; b:=x[n-1]:
>   Y:=0;
>   for k from 0 to n+1 do
>      Y:=Y+c[k]*S(t,k,a,b):
>   end do;
>   Y:=simplify(Y):
>   ecY:=diff(Y,t$2)-q(t)*Y=r(t):
>   ecd:=Array(0..n+1);
>   for k from 0 to n-1 do
>       ecd[k]:=eval(ecY,t=x[k]):
>   end do;
>   ecd[n]:=eval(Y,t=c0)=alpha:
>   ecd[n+1]:=eval(Y,t=d0)=beta:
>   C:=solve({seq(ecd[k],k=0..n+1)},[seq(c[k],k=0..n+1)]):
>   assign(C):
>   return Y:
> end proc:
> n:=34:
> q:=t->-86: r:=t->5*sin(9*t):
> ecz:=diff(Z(t),t$2)-q(t)*Z(t)=r(t):
> dsolve({ecz,Z(0)=0,Z(1)=sin(9)},Z(t)):
> assign(%):
```

```
>  b:=1: a:=0: c0:=Pi/54: d0:=Pi/12:
> u:=[Pi/54,seq((b-a)/2*cos(k*Pi/n)+(a+b)/2,k=1..n),Pi/12]:
> u:=sort(evalf(u)):
> n:=nops(u):alpha:=simplify(eval(Z(t),t=Pi/54)):
> beta:=simplify(eval(Z(t),t=Pi/12)):
> x:=Array(0..n-1,u):
> eval(x):profile(genceb):
> Y:=genceb(x,n,q,r,Pi/54,Pi/12,alpha,beta) :
>showprofile(genceb):
> p1:=plot([Y,Z(t)],t=0..1,title="Approximate solution"):
>p2:=plots[pointplot]([[Pi/54,eval(Z(t),t=Pi/54)],
[Pi/12,eval(Z(t),t=Pi/12)]],
symbol=circle,symbolsize=30,color=[BLACK]):
> plots[display]({p1,p2}):
> plots[logplot](Y-Z(t),t=0.001..0.999,
title="Errors in semi-logarithmic scale",
color=[BLUE],axis=[gridlines=[10, color=blue]]):
```

=======

7.2 Nonlinear form

7.2.1 Global B-splines

=======

```
%Nonlinear equation Goldner(Global B-splines)
%y"+y^3+(4-(t-t^2)^3)/(t+1)^3=0
%inner conditions: y(1/4)=3/20;y(1/2)=1/6;
%exact solution y=(t-t^2)/(t+1);
f=@(t,y) -y.^3-(4-(t-t.^2).^3)./(t+1).^3;
a=0; b=1;
% alpha=0;
% beta=0;
fe=@(t) (t-t.^2)./(t+1);
fd=@(x,y) -3*y.^2;
startv=@(t) (t+2)/15;
d=1/4; e=1/2;
alpha=fe(d); beta=fe(e);
N=15;
k=3;
t=linspace(a,b,100);
%Tolerance
err=1e-6;
[x,y]=polycollocnelin(f,fd,a,b,d,e,alpha,beta,N,k,startv,t,err);
close all;
figure(1)
plot(x,startv(x), x,y, [d,e],[alpha,beta],'o');
title('Goldner nonlinear equation-Global B-spline','FontSize',10)
```

```
legend('Start sol.','Approx sol.','inner points: d=1/4,e=1/2',0);
grid on
figure(2)
semilogy(x,abs(fe(x)-y))
xlabel('x','FontSize',10)
ylabel('ln |y(x)-y_{\Delta}(x)|','FontSize',10)
title('Error in semilogarithmic scale','FontSize',10)
grid on
```

7.2.2 Combined Method B-splines and Runge-Kutta

```
=======

%Costabile combined method B-Spline and Runge-Kutta
%y''+exp(-y)=0; in [0,1]
%inner conditions: y(pi/6)=ln(3/2), y(pi/4)=ln((2+sqrt(2))/2)
%exact solution y=ln(sin(x)+1)
%startv =start solution
%N=number of meshpoints
%k= degree of B-splines
%err=tolerance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f=@(x,y) -exp(-y);
fd=@(x,y) exp(-y);
fe=@(x) log(sin(x)+1);
N=11;
k=5;
a=0; b=1;
d=pi/6; e=pi/4;
```

```
p3=(d+e)/2;
alpha=fe(d); beta=fe(e);
t=linspace(a,b,100);
err=1e-6;
%construction of start  solution using polyfit on :d,e,(d+e)/2
u=[d,e,p3]; v=fe(u);
c=polyfit(u,v,2);
startv=@(x) c(1)*x.^2+c(2)*x+c(3);
tic;
[x,y]=polycalnlinRK(f,fd,a,b,d,e,alpha,beta,N,k,startv,t,err);
toc;
close all;
figure(1)
plot(x,startv(x), x,y, [d,e],[alpha,beta],'o');
title('Nonlinear eq. Costabile-Combined method Bs+Rk','FontSize',10)
legend('start sol.','approx. sol.','inner points: d=pi/6,e=pi/4',0);
grid on
figure(2)
semilogy(x,abs(fe(x)-y))
xlabel('x','FontSize',10)
ylabel('ln |y(x)-y_{\Delta}(x)|','FontSize',10)
title('Error in semilogarithmic scale','FontSize',10)
grid on
```

========

```
function [x,y]=polycollocnelin(f,fd,a,b,d,e,alpha,beta,N,k,startv,t,err)
%POLYCOLLOCNELIN solve a nonlinear polylocal problem
%y"(x) =f(x,y)
```

```
% inner conditions: y(d)=alpha, y(e)=beta;
%call spr=polycalnlin(f,a,b,d,e,alpha,beta,N,k,startv,t)
%f - function, define  by (BVP)
%fd - derivative of  f against the second variable
%a,b - endpoints
%d,e - inner points
%alpha, beta - values at inner-points
%N -number of subintervals
%k - number of Legendre points
%startv - starting value (function)
%t - evaluation points
%err - tolerance
%x,y - solution (output)
%prepare mesh
%n=k*N+2; number of mesh-points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ok=0; while(~ok)
breaks=linspace(a,b,N+1);
degree=k+1;
order=degree+1;
knots = augknt(breaks,order,k);
rho=LegendrePoints(k);
tau=[];
for i=1:N
    tau=[tau,(breaks(i+1)+breaks(i)+rho*(breaks(i+1)-breaks(i)))/2];
end
%check d,e
tauv=sort([tau,d,e]);
```

```
pozdv=find(tauv = = d);
pozev=find(tauv = = e);
if (length(pozdv)~=1 ||
length(pozev)~=1)
warning(' d or e illegal! increment N');
    N=N+1;
else
    ok=1;
end
end
%prepare iteration
x=sort([a,tau,b,d,e])';
tl=(x~=d & x~=e);
y=startv(x);
% compute inverse of matrix collocation
Minit=spcol(knots,order, x, 'sparse');
coeffin = Minit\y; size(coeffin);
spi=spmak(knots,coeffin.');
yd2=fnval(fnder(spi,2),x(tl));
%build collocation matrix
xdd=sort([a,b,tau]);
tauxx=sort([d,e,brk2knt(xdd,3)])';
Mcol1=spcol(knots,order,tauxx,'sparse');
v=1:length(tauxx); pozd=find(tauxx==d);
poze=find(tauxx==e);
v([pozd,poze])=[];
vf=v(1:3:end); vd2=v(3:3:end);
vfc=sort([vf,pozd,poze])';
```

```
x=x(tl); y=y(tl);
% iteration while 1
Valjac=repmat(fd(x,y),1,length(x));
Mcol=[Mcol1(vd2,:)-Valjac.*Mcol1(vf,:);
Mcol1([pozd,poze],:)];
rhs=[yd2-f(x,y);0;0];
%solve the nonlinear system
coeffs=coeffin-Mcol\rhs;
if norm(coeffin-coeffs,inf)\<= err
    sp=spmak(knots,coeffs');
return
end
coeffin=coeffs; y=Mcol1(vf,:)*coeffin;
yd2=Mcol1(vd2,:)*coeffin;
end
sp = spmak(knots,coeffs.');
x=t;
y = fnval(sp,x);
%roots of \ Legendre polynomial
function r=LegendrePoints(k)
switch k
    case 2 p=[3/2,0,-1/2];
    case 3 p=[5/2,0,-3/2,0];
    case 4 p=[35/8,0,-15/4,0,3/8];
    case 5 p=[63/8,0,-35/4,0,15/8,0];
    case 6 p=[231/16,0,-315/16,0,105/16,0,-5/16];
    case 7 p=[429/16,0,-693/16,0,+315/16,0,-35/16,0];
otherwise
```

```
    p=[6435/128,0,-3003/32,0,+3465/64,0,-315/32,0,+35/128];
end
r=sort(roots(p))';
```

=======

```
function [x,y]=polycalnlinRK(f,fd,a,b,d,e,alpha,beta,N,k,startv,t,err)
%POLYCALNLIN solve a nonlinear polylocal problem B-splines combined with
%Runge-Kutta methods
%y"=f(x,y)
% inner conditions: y(d)=alpha, y(e)=beta;
%call spr=polycalnlin(f,a,b,d,e,alpha,beta,N,k,startv,t)
%f - function, define BVPN or BVPL
%fd - derivative of f depending on second variable
%a,b - endpoints
%d,e - inner points
%alpha, beta - values at inside points
%N - number of subintervals
%k - number  of Legendre points
%startv - starting value (function)
%t - evaluation points
%err - error
%x,y - solution (output)
%prepare meshes
ti=t(t>=d & t<=e);
tleft=t(t<d);
tright=t(t>e);
%solve BVP
[sp]=BVPcollocnelin(f,fd,a,b,alpha,beta,N,k,startv,err);
```

```
yi=fnval(sp,ti);
ydi=fnval(fnder(sp),[d,e]);
x=ti'; y=yi';
%solver options
opts=odeset('AbsTol',1e-8,'Reltol',1e-7);
%solve right (IVP)
if ~is empty(tright)
    tright=[e,sort(tright)];
    [tr,wr]=ode45(@rhs,tright,[beta,ydi(2)],opts);
    x=[x;tr(2:end)]; y=[y;wr(2:end,1)];
end
%solve left (IVP)
IVP if ~is empty(tleft)
    tleft=[d,sort(tleft,'descend')];
    [tl,wl]=ode45(@rhs,tleft,[alpha,ydi(1)],opts);
    x=[tl(end:-1:2);x]; y=[wl(end:-1:2,1);y];
end
% nested function for Runge-Kutta method
function dy=rhs(x,u) dy=[u(2); f(x,u(1))];
end
end
```

7.2.3 Combined Method: Chebychev Collocation with Runge-Kutta

=======

```
%the temperature of reaction-diffusion problem, p=3
%u"+u^3=0; u(0)=u(1)=0;
```

```
%inner conditions: u(0.2)=u(0.8)=1.929990320692795
%f0=start solution


f=@(x,y) -y.^3;
df=@(x,y) -3*y.^2;
err=1e-8;
NMAX=50;
N=128;
f0=@(x) 12*pi/sqrt(2)*x.*(1-x);
a=0; b=1;
c=0.2;
d=0.8;
alpha=1.92990320692795; bet=1.92990320692795;
tic;
[x,y,ni]=solvepolylocalceb(N,f,df,a,b,c,d,alpha,bet,f0,err,NMAX);
toc;
ni;
figure(1);
plot(x,y,[c,d],[alpha,bet],'o')
title('The average temp. of reaction-diffusion eq.  C.C+R-K method')
grid 'on'
```

7.2.4 Compare methods

```
   =======

%Bratu's nonlinear problem (Lamda=1)
%Compare methods: Global B-splines, B-splines+Runge-Kutta,C.C+Runge-Kutta
```

```
% start solution : y(x)=x*(1-x)
%u"+exp(u)=0;
%inner conditions: u(0.2)=u(0.8)=0.08918993462883;
%this values are computed using Maple codes
%err=tolerance
%NMAX=max number of iterations
%N=number of degree freedom
%ni=number of iterations

f=@(x,y) -exp(y);
df=@(x,y) -exp(y);
err=1e-10;
NMAX=50;
N=15;
f0=@(x) x.*(1-x);
a=0; b=1;
c=0.2;
d=0.8;
alpha=0.08918993462883; bet=0.08918993462883;
%1.Pseudospectral method(C.C)+Runge-Kutta
tic;
[x,y,ni]=solvepolylocalceb(N,f,df,a,b,c,d,alpha,bet,f0,err,NMAX);
toc;
ni;
figure(1);
plot(x,y,[c,d],[alpha,bet],'o')
grid on
title('Bratu's problem CC+RK')
```

```
%2.Combined B-splines+Runge-Kutta
k=3;d=0.2;
e=0.8;N=15;
t=linspace(a,b,100);
tic;
[x,y]=polycalnlinRK(f,df,a,b,d,e,alpha,bet,N,k,f0,t,err);
toc;
figure(2);
plot(x,y,[d,e],[alpha,bet],'x')
grid on
title('Bratu's problem BS+RK')
%3.Global B-splines
N=15;
%start solution
startv=@(x) 3.9*x.*(1-x)/7;
[x,y]=polycollocnelin(f,df,a,b,d,e,alpha,bet,N,k,startv,t,err);
toc;
figure (3);
plot(x,y,[d,e],[alpha,bet],'*')
grid on
title(' Bratu' problem- global method B-Splines')
```

=======

```
function [X,y,ni]=solvebilocalceb(N,f,dfdy,alpha,bet,f0,err,NMAX)
%SOLVBILOCALCEB - solution of a BVP with Tchebychev collocation
%call [X,y,ni]=solvebilocalceb(N,f,dfdy,alpha,bet,u0,err,NMAX)
%x - abscissas
%y - solution values on Chebychev grid
```

```
%f0 - starting value (function)
%ni - eff # of iters
%N - #points
%f - rhs function
%dfdy - derivative of f wrt y
%alpha, bet - boundary values
%err - error
%NMAX max # of iter.s.

if nargin<8, NMAX=20; end
if nargin<7, err=1e-8; end
[X,D]=chebdif(N,2);
if nargin<6
    u0=zeros(N,1);
else
    u0=f0(X);
end
i=2:N-1;
I=eye(N);
u=u0;
for k=1:NMAX % Newton iterations
    F=[D(i,:,2)*u-I(i,:)*f(X,u); u(N)-bet; u(1)-alpha];
    dF=[D(i,:,2)-I(i,:)*diag(dfdy(X,u));I(N,:);I(1,:)];
    un=u-dF\F;
    if norm(u-un,inf)<err
        %y=[alpha;un;bet];
        y=un;
        ni=k;
```

```
        return
    end
    u=un;
end
error('too many iterations')
```

=======

```
function [X,Y,ni]=solvepolylocalceb(N,f,dfdy,a,b,c,d,alpha,bet,f0,err,NMAX)
%SOLVBILOCALCEB - solution of a BVP with Tchebychev collocation
%call [X,y,ni]=solvepolylocalceb(N,f,dfdy,a,b,alpha,bet,f0,err,NMAX)
%x - abscissas
%y - solution values on Chebychev grid
%f0 - starting value (function)
%ni - number of iterations
%N - number of mesh points
%f - rhs function
%dfdy - derivative of f wrt y
%a,b - interval endpoints
%c,d - inner point
%alpha, bet - boundary values
%f0 starting function
%err - tolerance
%NMAX max number of iterations.

if nargin<12, NMAX=50; end
if nargin<11, err=1e-3; end
g=@(t,y) (d-c)^2/4*f(((d-c)*t+(c+d))/2,y);
dg=@(t,y) (d-c)^2/4*dfdy(((d-c)*t+(c+d))/2,y);
```

```
if (nargin<10)||isempty(f0)
    g0=@(t) zeros(size(t));
else
    g0=@(t) f0(((d-c)*t+(d+c))/2);
end
[x,y,ni]=solvebilocalceb(N,g,dg,alpha,bet,g0,err,NMAX);
xx=linspace(-1,1,10*N)';
Y=chebint(y,xx);
format long
%chebint(y,[-0.6,0.6])
X=((d-c)*xx+(c+d))/2;
D1f=chebdifft(y, 1);
Dd=2/(d-c)*D1f(1); Dc=2/(d-c)*D1f(end);
Dd1=double(Dd);
Dc1=double(Dc);
%solver options
opts=odeset('AbsTol',err,'Reltol',1e-7);
%solve right IVP
if d~=b
     tright=[d,b];
     [tr,wr]=ode23tb(@rhs,tright,[bet,Dd1],opts);
     X=[X;tr(2:end)];
     Y=[Y;wr(2:end,1)];
end
% solve left IVP
if c~=a
    tleft=[c,a];
    [tl,wl]=ode23tb(@rhs,tleft,[alpha,Dc1],opts);
```

```
    X=[tl(end:-1:2);X];
    Y=[wl(end:-1:2,1);Y];
end
% nested function for RK method
    function dy=rhs(x,u)
        dy=[u(2); f(x,u(1))];
    end
end
```

7.3 BVP4C-codes

```
   ================================================

function ex2

\% Solve ((1 - x^7\)y')' + lambda*x^7*y = 0. Let v = (1 - x^7}%
*y so

\% that the equations are y' = v/(1 - x^7), v' = - lambda*x^7
*y,

\% with boundary conditions y(0) = 0, y(1) = 1.

lambda = input('Guess for eigenvalue: ');

sol = bvpinit(linspace(0,1,10),@guess,lambda);

sol = bvp4c(@odes,@bcs,sol);

fprintf('Computed the eigenvalue \%g\n',sol.parameters)

plot(sol.x,sol.y(1,:));

\%============================================================

function v = guess(x)

g = 5*pi/2;
```

```
y = x*sin(g*x);

yp = sin(g*x) + g*x*cos(g*x);

v = [ y; (1 - x^7)*yp];

function dydx = odes(x,y,lambda)

if x == 1

yp = lambda/7;

else

yp = y(2)/(1 - x^7);

end

dydx = [ yp; -lambda*x^7*y(1) ];

function res = bcs(ya,yb,lambda)

res = [ ya(1); yb(1)-1; yb(2) ];

============================================================

function ex2
```

```
epsilon = 0.1;

k = 0.1;

d = 0.001;

solinit = bvpinit(linspace(d,1,5),[0.01; 0],0.01);

sol = bvp4c(@odes,@bcs,solinit,[],epsilon,k,d);

p = sol.parameters; \% unknown parameter

xint = linspace(d,1);

Sxint = deval(sol,xint);

\% Augment the solution array with the values y(0) = p, y'(0) = 0

\% to get a solution on [0, 1]. For this problem the solution is

\% flat near x = 0, but if it had been necessary for a smooth graph,

\% other values in [0,d] could have been obtained from the series.

x = [0 xint];

y = [[p; 0] Sxint];
```

```
plot(x,y(1,:))

grid on

title('Michaelis-Menten kinetics problem with coordinate singularity.')

\%=========================================

function dydx = odes(x,y,p,epsilon,k,d)

dydx = [ y(2)

-2*(y(2)/x) + y(1)/(epsilon*(y(1) + k)) ];

function res = bcs(ya,yb,p,epsilon,k,d)

\% The boundary conditions at x = d are that y and y'

\% have values yatd and ypatd obtained from series

\% expansions. The unknown parameter p = y(0) is used

\% in the expansions.

\% It is evaluated as a limit in the differential equation.

yp2atd = p/(3*epsilon*(p + k));
```

```
yatd = p + 0.5*pi/2;


ypatd = yp2atd*d;


res = [ yb(1) - 1


ya(1) - yatd


ya(2) - ypatd ];
============================================================
function sol = ex3
global yatd
d = 0.001;
yatd = asymp(d);
solinit = bvpinit(linspace(d,1/2,10),@guess);
sol = bvp4c(@odes,@bcs,solinit);
xint = linspace(d,1/2);
yint = deval(sol,xint);
x = [0 xint];
y = [0 yint(1,:)];
asympy = [0 asymp(xint)];
plot(x,y,x,asympy)
grid on
axis([0 0.5 0 0.5])
legend('Computed','Asymptotic',4)
%============================================================
function v = guess(x)
```

```
v = [x*(1-x); 1-2*x];


function dydx = odes(x,y)
dydx = [y(2); -1/y(1)];


function res = bcs(ya,yb)
global yatd
res = [ya(1)-yatd; yb(2)];


function v = asymp(x)
v = x.*sqrt(-2*log(x));
============================================================
%Fischer  BVP problem:
%u"+cu'+u(1-u)=0
%u(-infinit)=1;u(infinit)=0
%this code is writen by Shampine and Gladwell
function Fisher
global c alpha beta infty
options=[];
%options:
options=bvpset(options,'Vectorized','on');
options=bvpset(options,'FJacobian',@odeJac);
options=bvpset(options,'BCJacobian',@bcJac);
color= ['y','r','b','g','m','k','c'];
ws=[5,20,35,50,65,80,100];
hold on
for i=1:7
    c=ws(i);
```

```
    alpha=(-c+sqrt(c^2+4))/2;
    beta=(-c+sqrt(c^2-4))/2;
    infty=10*c;
    solinit=bvpinit(linspace(-infty,infty,20),@guess);
    tic;
    sol=bvp4c(@ode,@bc,solinit,options);
    toc;
    figure(1)
    plot(sol.x,sol.y(1,:),color(i));
    title('Fisher BVP');
    axis([-100 100 0 1]);
    grid on
    hold on
    drawnow

end

legend('c=5','c=20','c=35','c=50','c=65','c=80','c=100',7);
hold off
function v = guess(z)
global c alpha beta infty
if z>0
    v=[exp(beta*z); beta*exp(beta*z)];
else
    v=[(1-exp(alpha*z)); -alpha*exp(alpha*z)];
end
function dydz = ode(z,y)
global c alpha beta infty
```

```
dydz = [y(2,:);-(c*y(2,:)+y(1,:).*(1-y(1,:)))];

    function dFdy = odeJac(z,y)
        global c alpha beta infty
        dFdy = [0,1
            (-1+2*y(1)),-c];

        function res = bc(ya,yb)
            global c alpha beta infty
res = [ya(2)/(ya(1)-1)-alpha
    yb(1)/exp(beta*infty)-1];

            function [dBCdya,dBCdyb] = bcJac(ya,yb)
                global c alpha beta infty
dBCdya = [-ya(2)/(ya(1)-1)^2,1/(ya(1)-1)
    0 0];
dBCdyb = [0 0
    1/exp(beta*infty), 0];
```

Appendix A

Errors and Floating Point Arithmetic

In order to measure the size of vectors and matrices (and in particular of errors involving them) we need to provide our linear space with a norm [78].

Definition 74 *A vector norm* $|\cdot| : \mathbb{R}^n \to \mathbb{R}_+ = \{x \in \mathbb{R} : x > 0\}$ *is a nonnegative function for wich:*

1. $|x| = 0$ *iff* $x = 0$
2. $|\gamma x| = |\gamma|\,|x|$, *for all* $\gamma \in \mathbb{R}$
3. $|x + y| \leq |x| + |y|$ *(triangular inequality)*

We mainly deal with so-called *Holder norms,*

$$|x|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, p \geq 1$$

and especially

$$|x|_1 = \left(\sum_{i=1}^{n} |x_i| \right) \text{ , the 1-norm}$$

$$|x|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}, \text{the Euclidian} \tag{A.1}$$

$$|x|_\infty = \max_{1 \leq i \leq n} |x_i|, \text{the max or sup norm} \tag{A.2}$$

All norms on a finite dimensional space define the same topology, since they are equivalent. Nevertheless, for some applications one particular norm is preferred over another.In this book we use the max norm, and denote it simply by $|x|$. For matrices we can also define a norm.

Definition 75 *Le $A \in \mathbb{R}^{n \times n}$.The polynomial*

$$\rho(\lambda) = \det(A - \lambda I)$$

is called ***characteristic polynomial*** *of A, the zeros of p are called* ***eigenvalues*** *of A. If λ is an* ***eigenvalue*** *of A , the vector $x \neq 0$ such that $(A - \lambda I)x = 0$ is an* ***eigenvector*** *of A, corresponding to the* ***eigenvalue*** *λ.*

Definition 76 *We also define*

$$\rho(A) = \max\{|\lambda|, \ \lambda \ \textbf{\textit{eigenvalue}} \ \text{of } A\}$$

the spectral radius of the matrix A.

Definition 77 *A* ***matrix norm*** *is a map $\|\cdot\| : \mathbb{R}^{m \times n} \to \mathbb{R}_+$ is a nonnegative function for wich:*

1. $\|A\| = 0$ *iff* $A = 0$
2. $\|\gamma A\| = |\gamma| \, \|A\|$ *for all* $\gamma \in R$
3. $\|A + B\| \leq \|A\| + \|B\|$
4. $\|AB\| \leq \|A\| \, \|B\|$

A simple way to obtain matrix norm is: given a vector norm $\|\cdot\|$ on C^n the map $\|\cdot\| : \mathbb{C}^{n\times n} \to \mathbb{R}$ the map $\|\cdot\| : \mathbb{C}^{n\times n} \to \mathbb{R}$

$$\|A\| = \sup_{v\in C^n, v\neq 0} \frac{\|Av\|}{\|v\|} = \sup_{v\in C^n, \|v\| \leq 1} \|Av\| = \sup_{v\in C^n, \|v\| = 1} \|Av\|$$

is a matrix norm called `subordinate matrix norm` or `natural norm`.

Remark 78 *A subordinate matrix norm verifies* $\|I\| = 1$.

Theorem 79 *[78, pag 22] Let* $A \in \mathbb{K}^{n\times n}(\mathbb{C})$. *Then*

$$\|A\|_1 := \max_j \sum_i |a_{ij}|$$

$$\|A\|_2 := \sqrt[2]{\rho(AA^*)} = \sqrt[2]{\rho(A^*)}\ \|A^*\|_2$$

where A^* *is the conjugate transpose of* A.

$$\|A\|_\infty := \max_i \sum_j |a_{ij}|$$

The norm $\|\cdot\|_2$ *is invariant to the unit transforms*

$$UU^* = I => \|A\|_2 = \|AU\|_2 = \|UA\|_2 = \|U^*AU\|_2$$

If A *is normal then*

$$AA^* = A^*A => \|A\|_2 = \rho(A)$$

Remark 80 *The norm* $\|\cdot\|_\infty$ *is called* ***`Chebychev norm`*** *or m-norm,* $\|\cdot\|_1$ *is called* ***`Minkowski norm`*** *and* $\|\cdot\|_2$ *is the* ***`Euclidian norm.`***

Definition 81 *[33, pag 61] Let* A *be a square matrix, then we call*

$$\mu(A) = \lim_{h\to 0, h>0} \frac{\|I + hA\| - 1}{h} \tag{A.3}$$

the ***`logarithmic norm of A.`***

Theorem 82 *[23] The* `logarithmic norm of A (A.3)` *is obtained by the following forms for the* `Euclidian norm` *:*

$$\mu(A) = \lambda_{\max} = \textit{largest eigenvalue of } \frac{1}{2}(A^T + A)$$

and for the `max -norm`

$$\mu(A) = \max_{k\ =1,\ldots n} (a_{kk} + \sum_{i \neq k} |a_{ki}|$$

The prof of these theorem is due in book [33, pag 61].

Theorem 83 *[33] If Q_1 and Q_2 are arbitrary orthogonal matrices then*

$$\|Q_1 A Q_2\|_2 = \|A\|_2$$

This theorem indicates why orthogonal matrices are so important in numerical analysis: In many applications we have to multiply a certain matrix by many transformation matrices. If the transformations are orthogonal, and we have a method to compute them reasonably well, the resulting product will have roughly the same 2-norm;i. e., the multiplication may be expected to be numerically stable!

We shall also need to formalize the notion of angles between vectors and matrices.

Definition 84 *The angle between two vectors $u, v \ \in R^n$ is defined by*

$$\theta = \arccos \frac{\left|u^T v\right|}{|u|_2 \, |v|_2}$$

This generalizes to a notion of angles between matrices or subspaces as follows: Let U and V be in $R^{n\times k}$ and $R^{n\times l}$, respectively. Then the angle between $range(U)$ and $range(V)$ (or simply the angle between U and V) is defined by

$$\theta = \arccos \max_{u\in U} \max_{v\ \in V} \frac{\left|u^T v\right|}{|u|_2 \, |v|_2}$$

Remark 85 *Note that*

$$0 \leq \theta \leq \pi/2$$

If $range(A) \ \cap range(B) = \{0\}$, then $\theta \geq 0$.

Definition 86 *Let A be square and nonsingular, and $\|\cdot\|$ a norm. Then*

$$cond(A) := \|A\| \left\|A^{-1}\right\|$$

is called the **condition number** *of the matrix A .*

We note that $cond(A) > 1$ and $cond(\gamma A) = cond(A)$ for any $\gamma \in R$. The value of $cond(A)$ depends, of course, on the matrix norm used in its definition. But we often avoid being specific about the norm used for this concept, emphasizing its qualitative nature.

If $cond(A)$ is large, then the matrix A is called *ill-conditioned.*

Since $cond\ (A) = 1$ if A is orthogonal ($AA^T = A^TA = I, A$ real), we conclude that orthogonal matrices are optimally *well-conditioned* with respect to the 2-norm.

Theorem 87 *1.Let A be an arbitrary square matrix and $\|\cdot\|$ a certain matrix (subordinate or no). Then*

$$\rho(A) \leq \|A\|$$

2. Given a matrix A and a number $\varepsilon > 0$, there exits a subordinate matrix norm such that

$$\|A\| \leq \rho(A) + \varepsilon$$

The proof of this theorem is given in book [77].

Remark 88 *For complex valued matrices the above formulas remain valid if one replace A by A^* and a_{kk} by* $\operatorname{Re} a_{kk}$.

Let

$$x = [x_1, ..., x_m]^T \in \mathbb{R}^m,\ y = [y_1, ..., y_n] \in \mathbb{R}^n$$

We think $y_\nu = f\ (x_1, ..., x_m),\ \nu = \overline{1, n}$, as a function of one single variable x_μ

$$\gamma_{\nu\mu}\ = (cond_{\nu\mu} f) = \left| \frac{x_\mu \quad \frac{\partial f\ \nu}{\partial\ x_\mu}}{f\ \nu(x)} \right| \tag{A.4}$$

These give us a matrix of condition numbers

$$\nu_\mu = \begin{pmatrix} \frac{x_1 \frac{\partial \ f_1}{\partial \ x_1}}{f_{\ 1}(x)} & \cdots & \frac{x_m \frac{\partial \ f_{\ 1}}{\partial \ x_{\ m}}}{f_{\ 1}(x)} \\ \vdots & \ddots & \vdots \\ \frac{x_1 \frac{\partial \ f_1}{\partial \ x_1}}{f_{\ n}(x)} & \cdots & \frac{x_m \frac{\partial \ f_1}{\partial \ x m}}{f_{\ n}(x)} \end{pmatrix} = [\gamma_{\nu\mu}(x)] \tag{A.5}$$

and we shall as `condition number` [77, pag 10]

$$(cond\ f)(x) = \|\nu_\mu\| \tag{A.6}$$

Example 89 ***(Systems of linear algebraic equations)*** *Given a nonsingular square matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$ solve the system*

$$Ax = b \tag{A.7}$$

Here the input data are the elements of A and b, and the result is the vector x. To simplify matters let's assume that A is a fixed matrix not subject to change, and only b is undergoing perturbations. We have a map $f : \mathbb{R}^n \to \mathbb{R}^n$ given by

$$x = f(b) := A^{-1}b$$

which is linear. There $\frac{\partial f}{\partial b} = A^{-1}$,

$$(cond\ f) = \frac{\|b\|\,\|A^{-1}\|}{\|A^{-1}b\|} = \frac{\|Ax\|\,\|A^{-1}\|}{\|A^{-1}b\|}$$

$$\max_{b\in\mathbb{R}^n}(cond\ f)(b) = \max_{x\in\mathbb{R}^n}\frac{\|Ax\|}{\|x\|}\left\|A^{-1}\right\| = \|A\|\left\|A^{-1}\right\|$$

Also we define [77, pag 12] the `condition of an algorithm A at x` by comparing the relative error with `eps`

$$(cond\ A)(x) = \inf_{x\ _A}\frac{\|x_A - x\|}{\|x\|}/eps$$

where $x\ _A$ is an input data for an any algorithm. The infimum is over all $x\ _A$ satisfying

$$y_A = f\ (x_A)$$

In practice one take any such $x_{\ A}$ and then obtain an upper bound for the condition number

$$(cond\ A)(x) \leq \frac{\frac{\|x_A - x\|}{\|x\|}}{eps}$$

We recall some considerations concerning **overall eror** [77, pag13]. The problem to be solved is again

$$f : \mathbb{R}^m \to \mathbb{R}^n, y = f(x) \tag{A.8}$$

This is the mathematical (idealized) problem, where the data are exact real numbers, and the solution is mathematically exact solution. When solving such a problem on a computer, in floating-point arithmetic with precision `eps` and using some algorithm A one first of all rounds the data, then applies to these rounded data not f but f_A.

$$x^* = x \text{ rounded}, \frac{\|x^* - x\|}{\|x\|} = \varepsilon,\ y^*_{\ A} = f_A(x^*)$$

Here ε represents the rounding error in the data. The total error that we wish to estimate is

$$\frac{\|y^*_A - y\|}{\|y\|}$$

By the basic assumption $f_{\ A}(x) = f\ (x_A)$ made on the algorithm A and choosing x_A optimally, we have

$$f_{\ A}(x^*) = f(x^*_A)$$
$$\frac{\|x^*_A - x^*\|}{\|x^*\|} = (cond\ A)(x^*)eps \tag{A.9}$$

Let $y^* = f(x^*_A)$, using the triangle inequality we have

$$\frac{\|y^*_A - y\|}{\|y\|} \leq \frac{\|y^*_A - y^*\|}{\|y^*\|} + \frac{\|y^* - y\|}{\|y\|} \approx \frac{\|y^*_A - y\|}{\|y^*\|} + \frac{\|y^* - y\|}{\|y^*\|}$$

We supposed $\|y\| \approx \|y^*\|$. Bz virtue of (A.9) we have the first term on the right

$$\frac{\|y^*_A - y\|}{\|y^*\|} = \frac{\|f_{\ A}(x^*) - f(x^*)\|}{\|f(x^*)\|} = \frac{\|f\ (x^*_A) - f(x^*)\|}{\|f(x^*)\|} \leq (cond\ f)(x^*)\frac{\|x^*_A - x\|}{\|x^*\|} = (cond\ f)(cond\ A)(x^*)\epsilon$$

and for the second

$$\frac{\|y^* - y\|}{\|y\|} = \frac{\|f\ (x^*) - f(x)\|}{\|f(x)\|} \leq (cond\ f)(x)\frac{\|x^* - x\|}{\|x\|} = (cond\ f)(x)\varepsilon$$

Assuming finally that $(cond\ f)(x^*) \approx\ (cond\ f\ )(x)$ we get

$$\frac{\|y_A^* - y\|}{\|y^*\|} \leq (cond\ f)(x)[\varepsilon + (cond\ A)(x^*)eps] \tag{A.10}$$

Conclusion 90 *The data error and* `eps` *contribute together towards the total error.*

If the condition number of a problem is large ($(condf)(x) \gg 1$), then even for small (relative) errors, huge errors in output data could be expected. Such problems are `ill-conditioned problems.`It is not possible to draw a clear separation line between well-conditioned problems.The classification depends on precision specifications. It we wish

$$\frac{\|y^* - y_A^*\|}{\|y\|} < \tau$$

and on (A.10) $(cond\ f)(x) \geq\ \tau$,then the problem is surely ill-conditioned. It is very important to choose a reasonable boundary off error, since otherwise, even if we increase the iteration number, we can not achieve the desired accuracy.

If the result of mathematical problem depends discontinuously on continuous input data, then it is impossible to obtain an accurate numerical solution in a neighborhood of the discontinuity. In such cases the result is significantly perturbed,even if the input data are accurate and the computation is performed using multiple precision. These problems are called `ill-posed problems.`

An ill-posed problem can appear if for the example an integer result is computed from real input data.

We recall some basic results concerning **stability** the due to *Radu T.Trâmbiţaş [77, pag: 16-18].*

Definition 91 *We say an algorithm* f_A *for a problem* $y = f(x),\ f : X \to Y$ *is* `stable` *if for each* $x \in X$

$$\frac{\|f_A(x) - f(\widetilde{x})\|}{\|f(\widetilde{x})\|} = \mathbb{O}(eps) \tag{A.11}$$

for some $\widetilde{x}$ *with*

$$\frac{\|\widetilde{x} - x\|}{\|x\|} = \mathbb{O}(eps)$$

Definition 92 *We say that an algorithm f_A for the problem is* `backward stable` *if*

$$\forall\ x \in X,\ \exists\ \widetilde{x}\ \textit{with}\ \frac{\|\widetilde{x} - x\|}{\|x\|} = \mathbb{O}(eps)$$

such that

$$f_A(x) = f(\widetilde{x})$$

This is tighttening of the definition of stability is the $\mathbb{O}(eps)$ in (A.11) was replace by zero.

Theorem 93 *Suppose a backward stable algorithm f_A is applied to solve a problem $f : X \to Y$ with condition number (cond f)(x) on a computer satisfying* `of the fundamental axiom of floating-point arithmetic.` *Then the relative error satisfies*

$$\frac{\|f_A(x) - f(x)\|}{\|f(x)\|} = \mathbb{O}((cond\ f)(x)eps)$$

Bibliography

[1] O.Agratini, I.Chiorean, G.Coman, R.Trîmbiţaş, *Analiză numerică şi Teoria Aproximării,* Presa Universitară Clujeană, Cluj-Napoca, 2002 (in romanian).

[2] W.Ames, E.Lohner, *Nonlinear models of reaction-diffusion in rivers,* R.Vichnevetsky and R.Stepleman Eds.,1981.

[3] V.Anisiu, *Calcul simbolic cu Maple,* Presa Universitară Clujeană, 2006 (in romanian).

[4] U.Ascher, J. Christiansen, and R.D. Russel, A Collocation Solver for Mixed Order Systems of Boundary Value Problem, *Mathematics of Computation, vol. 33(146), pp: 659-679, 1979.*

[5] U.Ascher, S.Pruess and R.D. Russel, On spline basis selection for solving differential equations, *SIAM J. Numer. Anal. 20 nr:1, 1983.*

[6] U. Ascher, R.M. Mattheij, and R.D. Russel, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, 1997.

[7] U. Ascher, Solving Boundary Value Problems with a Spline-Collocation Code, *Journal of Comput. Physics, vol 34(3), pp: 401-413, 1980.*

[8] U. Ascher, Discrete least squares approximations for ordinary differential equations, *SIAM J.Numer.Anal.,15, pp: 478-496, 1978.*

[9] P.B.Bailey,L.F.Shampine,P.E.Waltman, *Nonlinear Two Point Boundary Value Problems,* NewYork: Academic Press,1968.

[10] C. de Boor, *A practical guide to splines,* Springer-Verlag, Berlin, Heidelberg, New York, 1978.

[11] C.M.Bender,S.A. Orszag, *Advanced Mathematical Methods for Scientists and Engineers , Asymptotic Mrthods and Perturbation Theory,* New York: Springer Verlag 1999.

[12] C. deBoor, *Package for calculating with B-spline,* SIAM J. Numer. Anal., 14, No.3, pp: 441-472, 1977.

[13] C. deBoor, *Spline Toolbox 3,* MathWorks Inc, Nattick, MA, 2008.

[14] C. deBoor, *Good approximation by splines with variable knots II,* Lecture Notes in Mathematics Series 363. New York, Springer Verlag, 1973.

[15] C. deBoor, B. Swartz, Collocation at Gaussian points, *SIAM J. Numer. Anal., Vol10, pp: 582-606, 1973.*

[16] C. deBoor, R. Weiss, *SOLVEBLOCK: A Package for Solving Almost Block Diagonal Linear Systems, with Application to Spline Approximations and the Numerical Solution of Ordinary Differential Equations,* MRC TSR #1625, Madison, Wisconsin, 1976.

[17] L.R. Burden, J.D. Faires, *Numerical Analysis*, PWS-Kent Publishing Company, New York, 1985.

[18] C. Canuto, Boundary Conditions in *Chebishev* and Legendre Methods, *SIAM J.Numer.Anal.23, pp: 815-831, 1986.*

[19] F.A Costabile, F.Dell' Accio, On the nonsingularity of a special class of centrosymetric matrices arising in spectral methods in BVPs, *Applied Mathematics and Computation 206, pp: 991-993, 2008.*

[20] F.A Costabile, A. Napoli, A collocation method for global approximation of general second order BVPs, *Computing Letters 3, pp: 23-24, 2007.*

[21] J. Christiansen, R.D. Russel, U. Ascher, A Collocation Solver for Mixed Order Systems of Boundary Value Problems, *Math.Comp., 13, no. 146, pp 659-679, 1979.*

[22] J. Christiansen, R.D. Russel, Error analysis for spline collocation methods with application to knot selection, *Math.Comp-v.32, pp: 415-419, 1978.*

[23] G.Dahlquist, *Convergence and stability in the numerical integrations of ordinary differential equations,* Math.Scand., Vol.4,p33-53.[III.2],[III.3],[III.4],1956.

[24] H.T.Davis, *Introduction to Nonlinear Differential and Integral Equations,* NewYork:Dover,1962.

[25] J.W. Demmel, *Applied Numerical Linear Algebra,* SIAM, 1997.

[26] **E. Drăghici, Daniel N.Pop,** Solution of a polylocal problems using Chebishev polynomials, *General Mathematics, Vol.16, Nr.4, pp: 47-59, Sibiu, 2008.*

[27] L. Fox, I.B Parker, *Chebishev Polynomials in Numerical Analysis,* Oxford Mathematical HandBooks, O.U.P, 1968.

[28] W. Gautschi, *Numerical Analysis, An Introduction,* Birkhauser Verlag, Basel, 1997.

[29] C.I. Gheorghiu, *Spectral methods for differential problems,* Casa Cărţii de Ştiinţă, Cluj Napoca, 2007.

[30] C.I Gheorghiu, D. Trif, *The numerical approximation to positive solution of some reaction -diffusion problems,* P.U.M.A 11, pp: 243-253, 2000.

[31] G. Goldner, Radu T.Trîmbiţaş, *A combined method for two point boundary value problem,* P.U.M.A, vol 11, No2, pp: 255-264, 2000.

[32] L. Greengard, V. Rokhlin, *On the Numerical Solution of Two-Point Boundary Value Problems,* Comm.Pure Appl.Math.XLIV, pp: 419-452, 1991.

[33] E. Hairer, S.P Norset, G. Wanner, *Solving Ordinary Differential Equations I, Nonstiff Problems, Second Revised Editions,* Springer Verlag Berlin, New-York, Heidelberg, 2000.

[34] A.Heck, *Introduction to Maple,* Springer-Verlag, Berlin, Heidelberg, 1997.

[35] P. Henrici, *Discrete Variable Methods in Ordinary Differential Equations*, John Willey and Sons, New York, 1962.

[36] W.Heinrichs, *Strong convergence Estimates for Pseudospectral Methods,* Appl. Math. pp:401-417, 1992.

[37] F.R.Hoog,R.Weiss, *Collocation methods for singular Boundary value problems,* SIAM J, Numer Anal.15: pp 198-217,1978.

[38] D. Hinton, *Sturm's 1836 Oscillation Results -Evolution of the Sturm-Liouville Theory, Past &Present,* W.O. Amrein, A.M. Hinz, D.B. Pearson(editors) Birkhauser Verlag,Basel-Boston-Berlin, pp: 1-29, 2005.

[39] E.Houstis, *A collocation method for systems of nonlinear ordinary differential equtions,* J. Math.Anal. Appl., **62** pp: 24-37, 1978.

[40] H.B.Keller,*Numerical Methods for Two- point Boundary Value Problems,* New York: Dover,1992.

[41] L.V.Kantorovich,G.PAkilov,*Functional Analysis in Normed Spaces,* New York: Pergamon Press,1964.

[42] Ph.Korman, *Average temperature in a reaction-diffusion process,* Problem 97-8, SIAM Review, 39, p318,1997.

[43] C.Lanczos, *Applied Analysis,* Prentice Hall Inc., Englewood Cliffs, N.J, 1956.

[44] A.Lupaş, *Metode numerice,* Editura Constant, Sibiu, 2001 (in romanian).

[45] H.A.Levine,*The role of critical exponents in blowup theorems,*SIAM Review, 32, pp.262-288,1990

[46] Patrick A.Domenico, Franklin W.Schwartz., *Physical and Chemical Hydrology (Second edition),* John Willey and Sons, Inc. 1998.

[47] I. Păvaloiu, N. Pop, *Interpolare şi Aplicaţii,* Risoprint, Cluj-Napoca, 2005 (in romanian).

[48] S.M Pizer, W.L Wallance, *To Compute Numerically. Concepts and Strategies,* Little, Brown and Company, 1983.

[49] Mathlab 2011.b, *Set of manuals,* MathWorks, Inc-Natick-MA, 2011.

[50] J.D. Murray, *Mathematical Biology,* 2nd, Berlin:Springer Verlag,1993.

[51] S.Micula, G.Micula, *Handbook of Splines,* Kluwer Academic Publishers, Netherlands, 1999.

[52] M.A. Diminescu, V. Nistorean, *Gospodărirea durabilă a resurselor de apă,* Conf. Internaţ. Energie-Mediu, CIEM Bucureşti, 2005 (in romanian).

[53] **Daniel N.Pop,** *A Collocation Method using Cubic B-Splines Functions for solving second order linear value problems with condition inside the interval* $[0,1]$*,* Studia Univ."Babeş-Bolyai", Mathematica, Volume LV, Number 2, pp: 177-187, June, Cluj-Napoca, 2010.

[54] **Daniel N.Pop, Radu T.Trâmbitas**, *A comparison between two collocation methods for linear polylocal problems-a Computer Algebra based approach,* International Conferences in Modelling and Development of Intelligent Systems-MDIS'09, pp: 164-174, Sibiu, 2009.

[55] **Daniel N. Pop, Radu T.Trâmbitas,** {{*New Trends in Approximation, Optimization and Classification*}}, *ch.*Solution of a polylocal problem - a Computer Algebra based approach, Lucian Blaga University Press, Proceedings of International Workshop New Trends in Approximation, Optimization and Classification, Sibiu, pp: 53-63, Sibiu, 2008.

[56] **Daniel N.Pop**, Error bound for the solution of a polylocal problem with combined methods, *Studia Univ."Babeş-Bolyai" Mathematica, vol LIV, Number 4, pp: 115-123, Cluj-Napoca, 2009.*

[57] **Ion Păvăloiu, Daniel N.Pop, Radu T.Trâmbitas,** Solution of polylocal problem with a pseodospectral method, *Annals "Tiberiu Popovici", Seminar of functional equations, approximaion and convexity, vol 8, pp: 53-65, Cluj-Napoca, 2010.*

[58] **Daniel N.Pop**, **Radu T.Trâmbitas,** An approximation methods for second order nonlinear value polylocal problems using B-splines and Runge-Kutta metods, *N.A.T.2010, Studia Univ."Babeş-Bolyai" Mathematica, vol LVI, Number 2, pp: 515-527, Cluj-Napoca, 2009. .*

[59] J.J Riessler, *Méthodes mathématiques pour la CAO,* Editure Macon, Paris, Milan, Barcelone, Bonn, 1991.

[60] I.A Rus, Paraschiva Pavel, *Ecuaţii Diferenţiale,* Editura Didactică şi Pedagogică, Bucureşti, 1982 (in romanian).

[61] R.D.Russel, L.F.Shampine, *A coolocation Method for Boundary Value Problems,* Numer.Math.19, Springer-Verlag, pp: 1-28, 1972.

[62] R.D Russel, Efficients of B-splines methods for solving differential eqution, Proc Fifth Conference on Numerical Mathematics, *Utilitas Math.Winipeg, Manitoba, pp: 599-617, 1975.*

[63] R.D Russel, A comparation of collocation and finite differences for two point boundary value problems, *SIAM, J.Numer., Anal vol 14, pp: 19-39, 1977.*

[64] R.D Russel, J. Christiansen, Adaptive mesh selection strategies for solving boundary value problems, *SIAM, J.Numer.Anal v.15, pp: 59-80, 1978.*

[65] R.D Russel, Mesh selection methods, in Codes for Boundary Value Problems, *Lecture Notes in Computer Science 74 Childs et al.eds, Springer Verlag, Berlin, 1979.*

[66] L.F Shampine, *Design of software for ODE,* J. Comput. Appl. Math, 205, 2007.

[67] L.F. Shampine, Conservation laws and the numerical solution of ODE's , *Comput. Math. Appl. 12B, pp:1287-1396, 1986.*

[68] L.F Shampine, I. Gladwell, S. Thompson, *Solving ODE's with MATLAB,* Cambridge University Press, the Edinburgh Building Cambridge, United Kingdom, 2003.

[69] L.F Shampine, J. Kierzenka, M.W. Reichelt, Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c, *J. Comput. Appl. Math, 26, 2000.*

[70] L.F Shampine, J. Gladwell, S. Thomson, *Solving Ode's with MATLAB,* Cambridge University Press, 2003.

[71] M.R.Scot, *Invariant Imbedding and Its Applications to Ordinary Differential Equations, An Introduction,* Reading, Ma:Addison-Wesley, 1973.

[72] M.H Schultz, *Spline Analysis,* Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1972.

[73] A. Solomonoff, E. Turkel, Global Properties of Pseudospectral Methods, *J. Comp. Phys.81, pp: 239-276, 1989.*

[74] A. Solomonoff, A Fast Algorithm for Spectral Differentiation, *J. Comput. Phys.81, pp. 239-276, 1989.*

[75] B. Swartz, Conditioning Collocation, *SIAM J. Numer. Anal, Vol. 25(1), pp: 124-146, 1988.*

[76] L.N Trefeten, *Spectral methods in MATLAB,* SIAM, Philadelphia, PA, 2000.

[77] Radu T.Trîmbiţaş, *Numerical Analysis,* Cluj Universitary Press, 2006.

[78] Radu T.Trîmbiţaş**,** *Numerical Analysis in MATLAB,* Presa Universitară Clujeană, 2009.

[79] G.M. Vainniko, *On the stability and convergence of collocation method,* Differentiasial'nye Uravneniya vol. 1, pp: 244-254, 1965.

[80] J.A.C Weideman, S.C Reddy, A MATLAB Differentiation Matrix Suite, *A.C.M Trans. on Math. Software, 26, pp: 465-519, 2000.*

[81] Z.Y Liu, Some properties of centrosymmetric matrices, *ELSEVIER, Applied Mathematics and Computation 141, pp: 297-306, 2003.*

Index

ISBN: 978-973-595-878-7