ABSTRACT

The given Master's dissertation dedicated to studying modern approaches of the numerical solution to the boundary value problems of Fredholm integro-differential equations. The basis of the numerical solution is the method of parametrization which reduces the initial problem to the set of Cauchy problems followed by linear algebraic system of equations. This technique gives a solution in cases when standard classical methods like Nekrasov's are not applicable. Numerical solution to ODE is implemented using Bulirsch-Stoer method which shows better computational performance and slightly better level of approximation in general cases compared with Runge-Kutte $4^{th}$ order. Also parallel computing technologies which were applied in these numerical implementations, substantially sped up a running time.

**Keywords:** Fredholm integro-differential equations, boundary value problems, parametrization method, Bulirsch-Stoer method

# АННОТАЦИЯ

В данной магистерской диссертации были изучены современные методы численного решения краевых задач Фредгольмовых интегро-дифференциальных уравнений. В основе численной реализации лежит метод параметризации, который сводит начальную задачу к множеству подзадач Коши, после которых следует решение систем алгебраических уравнений. Этот метод дает решение даже в тех случаях, когда стандартные классические методы, такие как метод Некрасова, не применимы. Численное решение обыкновенных дифференциальных уравнений реализовано с помощью метода Булирша-Штёра, который дает высокую степень точности и вычислительную эффективность по сравнению с методом Рунге-Кутты 4-го порядка. Также были применены параллельные вычисления, которые дали существенный прирост к оптимизации алгоритма.

**Ключевые слова:** Интегро-дифференциальные уравнения Фредгольма, краевые задачи, метод параметризации, метод Булирша-Штёра

# АҢДАТПА

Берілген магистрлік диссертациялық жұмыста шеттік шарты бар Фредгольм интегро-дифференциалдық теңдеулердің есептік шешуін табудағы заманауи әдістер зерттелген. Параметрлеу әдісі - есептік шешімнің негізі болып табылады. Аталған әдіс бастапқы есепті бірнеше Коши есебіне түрлендіру және алгебралық теңдеулер жүйесін шешуге негізделген. Бұл әдіс Некрасов әдісі секілді классикалық тәсілдер жарамаған жағдайда теңдеуді шешуге мүмкіндік береді. Кәдімгі дифференциалдық теңдеулердің есептік шешуі Булирш-Штёр әдісі бойынша есептелген. Бұл әдіс 4-ші дәрежедегі Рунге-Кутта әдісіне қарағанда дәлірек шешім береді және оптимизация тұрғысы бойынша нәтижесі жоғары. Сонымен қатар бұл зерттеуде оптимизация одан әрі жоғары болу үшін параллельді есептеу әдістері қолданылды.

**Түйінді сөздер:** Фредгольм интегро-дифференциалдық теңдеулері, шеттік шарты бар есеп, параметрлеу әдісі, Булирш-Штёр әдісі

CONTENT

# INTRODUCTION

**Research relevance**. The study of integro-differential equations is motivated by real world applications. Various problems of physics, engineering, biology, etc. lead to the study of integral-differential equations and to the formulation of related specific tasks. There are many situations in which a nonlocal equation gives a significantly better model than a Partial Differential Equations. In connection with this, the theory of such equations has attracted the attention of mathematicians. The general form of integro-differential equation with boundary condition which is considered in this work given below:

$$\frac{dx}{dt} = A(t)x + \int_0^T K(t,s)x(s)ds + f(t), t \in (0,T), x \in R^n$$

$$Bx(0) + Cx(T) = d, d \in R^n$$

where the (n × n) matrices A(t) and K(t, s) are continuous on [0, T] and [0, T] × [0, T] , respectively. The n vector f(t) is continuous on [0, T].

A solution to this problem is a vector function x(t), continuous on [0, T] and continuously differentiable on (0, T) . It satisfies the integro-differential equation and boundary condition.

The given linear boundary problem can be solved analytically by several ways. For instance, Nekrasov's method and Green's function method are frequently used ones. Despite the fact that classical analytical solutions handle most of the cases, there are still problems in the cases when an integral equation is not solvable. The parametrization and interval partition method can be used to deal with this problem.

In general case, the smallness of interval's partition step is required in the algorithms for solving the linear boundary value problems for Fredholm integro-differential equations. In the parametrization method used in this work and proposed by Dr. Dzhumabaev, the arbitrary partitions of interval (including the case, when the interval is not divided into parts) are considered. Essential requirement to the partition is its regularity.

General overview of integral and integro-differential equations and examples of real-world applications are given in Chapter 1. Chapter 2 is devoted to the study of Fredholm integro-differential equation with degenerate kernel. Interval [0, T] is divided into N parts, then there were introduced the additional parameters as values of solution at the left-end points of subintervals. Unique solvability of the special Cauchy problem for a partition $\Delta_N$ is equivalent to the invertibility of matrix $I - G(\Delta_N)$ compiled by the fundamental matrix of differential part and matrices of integral term's kernel. Partition $\Delta_N$ is called regular if the matrix $I - G(\Delta_N)$ is invertible.

For the regular partition $\Delta_N$ by using $[I-G(\Delta_N)]^{-1}$, boundary condition, and conditions of continuity of solution at the interior partition points of interval [0, T], we construct a system of linear algebraic equations with respect to parameters. It is

established that the invertibility of system's matrix is equivalent to the well-posedness of considered boundary value problem.

Chapter 3 is devoted to the algorithms for finding a solution to the linear boundary value problem for integro-differential equation with degenerate kernel and its numerical experiments. For the selected partition $\Delta_N$ the matrix $G(\Delta_N)$ is computed. If the matrix $I - G(\Delta_N)$ is invertible, then we construct the system of linear algebraic equations. The elements of matrix $G(\Delta_N)$, the coefficients and right-hand side of this system are determined by solving the Cauchy problems for the ordinary differential equations and by calculating the definite integrals from the known functions on the subintervals of partition $\Delta_N$ . Solving the system, we find the values of solution at the left-end points of subintervals. Using the founded values of solution and the initial data of integro-differential equation, we compute the function $F^*(t)$, continuous on $[0, T]$ . Solving the Cauchy problem for the ordinary differential equations with right-hand side $F^*(t)$ , we find the values of desired function at the other points of interval $[0, T]$ .

**Research goals**. If it is possible to construct the fundamental matrix of differential part and evaluate the integrals explicitly, then the algorithm permits to find the solution in explicit form. However, in general, for the system of ordinary differential equations with variable coefficients the construction of fundamental matrix fails, and the integrals are calculated approximately. Therefore, the main goal of this work is implement numerical methods for solving Cauchy problems. Particularly, Cauchy problems for ordinary differential equations on subintervals are solved by the Bulirsch-Stoer method, whereas in original work Runge-Kutta method of 4th order was used. There was established that solution to integro-differential equation obtained after applying Bulirsch-Stoer method shows better level of approximation and computationally faster. Integrals are estimated by Simpson's method. Moreover, the elements of matrix $G(\Delta_N)$, the coefficients and right-hand side of the system of equations with respect to parameters were calculated by the parallel computing on the subintervals.

**Novelty**. In this research work, the numerical solution to the Fredholm integro-differential equations using Bulirsch-Stoer method showed a high accuracy and noticeably high computational efficiency compared with Runge-Kutta $4^{th}$ order method. Also applying parallel computing technologies substantially sped up a running time. The accuracy of approximate solution depends on the choice of approximating kernel as well as on the step's number of iterative process. Abovementioned claims were checked by *numerical experiments*. *Practical and theoretical importance* of the given work is that: necessary and sufficient conditions for the well-posedness of problem are obtained in the terms of properties of approximating boundary value problems for the integro-differential equations with degenerate kernels, modern numerical methods and parallel computing technologies were applied.

# 1 General overview of integral and integro-differential equations

## 1.1 Types and methods of solving Integro-Differential Equations

Originally, integro-differential equation came from the study of integral and differentials equations separately, where an integral equation is an equation in which the unknown function u(x) appears under an integral sign. A standard integral equation in u(x) is of the form:

$$u(x) = f(x) + \lambda \int_{g(x)}^{h(x)} K(x,t)u(t)dt,$$

where g(x) and h(x) are the limits of integration, λ is a constant parameter, and K(x, t) is a function of two variables x and t called the kernel or the nucleus of the integral equation. The function u(x) that will be determined appears under the integral sign, and it appears inside the integral sign and outside the integral sign as well. The functions f(x) and K(x, t) are given in advance. It is to be noted that the limits of integration g(x) and h(x) may be both variables, constants, or mixed.

Then, a new special type of equations was derived by further development of study such integral equations. This new type called an integro-differential equation. An integro-differential equation is an equation in which the unknown function u(x) appears under an integral sign and contains an ordinary derivative $u^{(n)}(x)$ as well. A standard integro-differential equation is of the form:

$$u^{(n)}(x) = f(x) + \lambda \int_{g(x)}^{h(x)} K(x,t)u(t)dt,$$

where g(x), h(x), f(x), λ and the kernel K(x, t) are as prescribed before.

As is typical with differential equations, obtaining a closed-form solution can often be difficult. In the relatively few cases where a solution can be found, it is often by some kind of integral transform, where the problem is first transformed into an algebraic setting. In such situations, the solution of the problem may be derived by applying the inverse transform to the solution of this algebraic equation. However, if exact solution does not exist, we use as many terms of the obtained series for numerical purposes to approximate the solution. The more terms we determine the higher numerical accuracy we can achieve.

### Integral equations

Integral equations and integro-differential equations can be classified into distinct types according to the limits of integration and the kernel K(x, t). Below there

is defined and investigated the basic classified types of integral equations and integro-differential.

Integral equations appear in many forms. Two distinct ways that depend on the limits of integration are used to characterize integral equations, namely:

1. If at least one limit is a variable, the equation is called a Volterra integral equation given in the form:

$$u(x) = f(x) + \lambda \int_a^x K(x,t)u(t)dt.$$

Vito Volterra (3 May 1860 – 11 October 1940) was an Italian mathematician and physicist, known for his contributions to mathematical biology and integral equations, being one of the founders of functional analysis. His most famous work was done on integral equations. He began this study in 1884 and in 1896 he published papers on what is now called 'an integral equation of Volterra type' (Fig 1.1.). He continued to study functional analysis applications to integral equations producing a large number of papers on composition and permutable functions.

Figure 1.1. Vito Volterra

Today it's a well-known fact that Vito Volterra is one of the first scientist which studied integral and integro-differential equations. Volterra integral equations arise in many scientific applications such as the population dynamics, spread of epidemics, and semi-conductor devices [1]. It was also shown that Volterra integral equations can be derived from initial value problems. Volterra started working on integral equations in 1884, but his serious study began in 1896. The name integral equation was given by du Bois-Reymond in 1888. However, the name Volterra integral equation was first coined by Lalesco in 1908.

There are a lot of extension and modification of abovementioned integral equation. For instance, Abel considered the problem of determining the equation of a curve in a vertical plane. In this problem, the time taken by a mass point to slide

under the influence of gravity along this curve, from a given positive height, to the horizontal axis is equal to a prescribed function of the height. In a such way, Abel derived the singular Abel's integral equation, a specific kind of Volterra integral equation.

Moreover, Volterra integral equations can be of the first kind or second kind. The first kind or the second kind, are characterized by a variable upper limit of integration [1]. For the first kind Volterra integral equations, the unknown function u(x) occurs only under the integral sign in the form:

$$f(x) = \int_0^x K(x,t)u(t)dt.$$

However, Volterra integral equations of the second kind, the unknown function u(x) occurs inside and outside the integral sign. The second kind is represented in the form:

$$u(x) = f(x) + \lambda \int_0^x K(x,t)u(t)dt.$$

The kernel K(x, t) and the function f(x) are given real-valued functions, and λ is a parameter.

2. If the limits of integration are fixed, the integral equation is called a Fredholm integral equation given in the form:

$$u(x) = f(x) + \lambda \int_a^b K(x,t)u(t)dt,$$

where a and b are constants.

Fredholm integral equations also arise in many scientific applications. Erik Ivar Fredholm (7 April 1866 – 17 August 1927) was a Swedish mathematician whose work on integral equations and operator theory foreshadowed the theory of Hilbert spaces. His theory of integral equations and his 1903 paper in Acta Mathematica played a major role in the establishment of operator theory. Fredholm introduced and analyzed a class of integral equations now called Fredholm equations. His analysis included the construction of Fredholm determinants, and the proof of the Fredholm theorems. In fact much of this work was accomplished during the months of 1899 which Fredholm spent in Paris studying the Dirichlet problem with Poincaré, Émile Picard, and Hadamard. In 1900 a preliminary report on his theory of Fredholm integral equations was published. Volterra had earlier studied some aspects of integral equations but before Fredholm little had been done. Of course Riemann, Schwarz, Carl Neumann, and Poincaré had all solved problems which now came under Fredholm's general case of an integral equation; this was an indication of how powerful his theory was [28].

Figure 1.2. Erik Ivar Fredholm

For the first kind Fredholm integral equations, the unknown function u(x) occurs only under the integral sign in the form:

$$f(x) = \int_a^b K(x,t)u(t)dt.$$

However, Fredholm integral equations of the second kind, the unknown function u(x) occurs inside and outside the integral sign. The second kind is represented by the form:

$$u(x) = f(x) + \lambda \int_a^b K(x,t)u(t)dt. \tag{1.1}$$

The kernel K(x, t) and the function f(x) are given real-valued functions, and λ is a parameter. When f(x) = 0, the equation is said to be homogeneous. K(x, t) is also known as kernel and can be several types. The most common one is degenerate kernel. A degenerate or a separable kernel is a function that can be expressed as the sum of the product of two functions each depends only on one variable. Such a kernel can be expressed in the form:

$$K(x,t) = \sum_{i=1}^{n} f_i(x)g_i(t).$$

Examples of separable kernels are x−t, $(x-t)^2$, 4xt, etc. In what follows we state, without proof, the Fredholm alternative theorem.

**Theorem 1.1** (Fredholm Alternative Theorem) If the homogeneous Fredholm integral equation

$$u(x) = \lambda \int_a^b K(x,t)u(t)dt$$

has only the trivial solution u(x) = 0, then the corresponding nonhomogeneous Fredholm equation (1.1) has always a unique solution. This theorem is known by the Fredholm alternative theorem [1].

**Theorem 1.2** (Unique Solution) If the kernel K(x, t) in Fredholm integral equation (1.1) is continuous, real valued function, bounded in the square a ≤ x ≤ b and a ≤ t ≤ b, and if f(x) is a continuous real valued function, then a necessary condition for the existence of a unique solution for Fredholm integral equation (1.1) is given by

$$|\lambda| M(b-a) < 1,$$

where

$$|K(x,t)| \leqslant M \in \mathbb{R}.$$

Integro-differential equations

As it was mentioned, integro-differential equations appear in many scientific applications, especially when we convert initial value problems or boundary value problems to integral equations. The integro-differential equations contain both integral and differential operators. The derivatives of the unknown functions may appear to any order. In classifying integro-differential equations, we will follow the same category used before.

1. Fredholm integro-differential equations appear when we convert differential equations to integral equations. The Fredholm integro-differential equation contains the unknown function u(x) and one of its derivatives $u^{(n)}(x)$, n ≥1 inside and outside the integral sign respectively. The limits of integration in this case are fixed as in the Fredholm integral equations. The equation is labeled as integro-differential because it contains differential and integral operators in the same equation. It is important to note that initial conditions should be given for Fredholm integro-differential equations to obtain the particular solutions. The Fredholm integro-differential equation appears in the form:

$$u^{(n)}(x) = f(x) + \lambda \int_a^b K(x,t)u(t)dt, \tag{1.2}$$

where u(n) indicates the nth derivative of u(x). Other derivatives of less order may appear with $u^{(n)}$ at the left side. Moreover, this type of equations is divided into first kind and second kind exactly as in the case of Fredholm integral equations. Examples of the Fredholm integrodifferential equations are given by

$$u'(x) = 1 - \frac{1}{3}x + \int_0^1 xu(t)dt, \quad u(0) = 0,$$

Because the resulted equation in (1.2) combines the differential operator and the integral operator, then it is necessary to define initial conditions u(0), u'(0), . . . , $u^{(n-1)}(0)$ for the determination of the particular solution u(x) of equation (1.2). Any Fredholm integro-differential equation is characterized by the existence of one or more of the derivatives u'(x), u''(x),. . . outside the integral sign. The Fredholm integro-differential equations of the second kind appear in a variety of scientific applications such as the theory of signal processing and neural networks [30].

2. Volterra integro-differential equations also appear when we convert initial value problems to integral equations. Volterra studied the hereditary influences when he was examining a population growth model. The Volterra integro-differential equation contains the unknown function u(x) and one of its derivatives u(n)(x), $n \geq 1$ inside and outside the integral sign. At least one of the limits of integration in this case is a variable as in the Volterra integral equations. The equation is called integro-differential because differential and integral operators are involved in the same equation. It is important to note that initial conditions should be given for Volterra integro-differential equations to determine the particular solutions. The Volterra integro-differential equation appears in the form:

$$u^{(n)}(x) = f(x) + \lambda \int_0^x K(x,t)u(t)dt,$$

where $u^{(n)}$ indicates the nth derivative of u(x). Other derivatives of less order may appear with $u^{(n)}$ at the left side. Moreover, this type of equations is divided into first kind and second kind exactly as in the case of integral Volterra equations. Examples of the Volterra integrodifferential equations are given by

$$u'(x) = -1 + \frac{1}{2}x^2 - xe^x - \int_0^x tu(t)dt, \; u(0) = 0,$$

The Volterra integro-differential equation appeared after its establishment by Volterra. It then appeared in many physical applications such as glassforming process, nanohydrodynamics, heat transfer, diffusion process in general, neutron diffusion and biological species coexisting together with increasing and decreasing rates of generating, and wind ripple in the desert. To determine a solution for the integro-differential equation, the initial conditions should be given, and this may be clearly seen as a result of involving u(x) and its derivatives. The initial conditions are needed to determine the exact solution.

There is also exists mixed type of abovementioned integro-differential equations - the Volterra-Fredholm integro-differential equations, which arise from parabolic boundary value problems, from the mathematical modelling of the spatio-

temporal development of an epidemic, and from various physical and biological models. The Volterra-Fredholm integro-differential equations appear in the literature in two forms, namely

$$u^{(k)}(x) = f(x) + \lambda_1 \int_a^x K_1(x,t)u(t)dt + \lambda_2 \int_a^b K_2(x,t)u(t)dt, \tag{1.3}$$

and the mixed form

$$u^{(k)}(x) = f(x) + \lambda \int_0^x \int_a^b K(r,t)u(t)dtdr, \tag{1.4}$$

where f(x) and K(x, t) are analytic functions. It is interesting to note that (1.3) contains disjoint Volterra and Fredholm integrals, whereas (1.4) contains mixed Volterra and Fredholm integrals. Moreover, the unknown functions u(x) appears inside and outside the integral signs. This is a characteristic feature of a second kind integral equation. If the unknown functions appear only inside the integral signs, the resulting equations are of first kind. Nevertheless, this type of equation is uncommon in practice and out of scope of this research.

A variety of analytic and numerical methods exist to solve abovementioned types of equations. Here is the review of the basic methods of solving integral and integro-differential equations. For instances, such methods as successive approximations method, Laplace transform method, spline collocation method, Runge-Kutta method, and others have been used to handle Volterra integral equations. Moreover there are plenty of recently developed methods, namely, the Adomian decomposition method (ADM), the modified Adomian decomposition method (mADM), and the variational iteration method (VIM) to handle Volterra integral equations [29]. Some of the traditional methods, namely, successive approximations method, series solution method, and the Laplace transform method are used as well. The theorems of uniqueness, existence, and convergence are important and can be found in the literature.

There is also variety of analytic and numerical methods have been used to handle Fredholm integral equations. The direct computation method, the successive approximations method, and converting Fredholm equation to an equivalent boundary value problem are among many traditional methods that were commonly used. The same modern method as for Volterra can be used here, i.e. the Adomian decomposition method (ADM), the modified decomposition method (mADM), and the variational iteration method (VIM) to handle the Fredholm integral equations [29].

Alternatively, a batch of works exist related to analytical and numerical solution to integro-differential equations. But in this research we consider the numeric solution for the second kind Fredholm integro-differential equation with degenerate kernel of the form:

$$\frac{dx}{dt} = A(t)x + \int_0^T K(t, s)x(s)ds + f(t), t \in (0, T), x \in R^n \quad (1.5)$$

$$Bx(0) + Cx(T) = d, d \in R^n \quad (1.6)$$

where the (n × n) matrices A(t) and K(t, s) are continuous on [0, T] and [0, T] × [0, T] , respectively. The n vector f(t) is continuous on [0, T].

A solution to the problems (1.5) and (1.6) is a vector function x(t), continuous on [0, T] and continuously differentiable on (0, T) . It satisfies the integro-differential equation (1.5) and boundary condition (1.6).

This linear boundary problem can be solved analytically by several ways. Reduction of problems for integro-differential equations to the integral equations is the main tool in investigating and solving the considered problems. For instance, Nekrasov's method and Green's function method are frequently used ones. In Nekrasov's method [31], the integral term of equation (1.5) is assumed as the right-hand side of corresponding differential equation. Using a fundamental system of solutions to this equation, the equation (1.5) is reduced to the Fredholm integral equation of the second kind. If the latter one is uniquely solvable, then the general solution for Fredholm integro-differential equation can be written down via the resolvent of integral equation. Now, the solvability of the problems (1.5) and (1.6) are equivalent to the solvability of linear system of algebraic equations compiled by the general solution and boundary condition (1.6).

Green's function method is another commonly used one. In this method, the integral term also refers to the right-hand side of differential equation. Under assumption on unique solvability of boundary value problem for the differential equation we construct its Green's function and then reduce the origin boundary value problem for Fredholm integro-differential equation to the Fredholm integral equation of second kind. Solving this equation, we find the desired function.

The mentioned methods are a base for the approximate methods considered in [32]. Under assumption on the solvability of Cauchy and boundary value problems for integro-differential equations, their approximate solutions are found by the method of averaging functional corrections, method of continuation on parameters, methods of S.A.Chaplygin, B.G.Galerkin, etc. Expanding the application areas of integro-differential equations and development of approximate methods for finding their solutions have led to the modification of known methods and creation of new ones. Overview of some modern and widely used approximate methods is contained in the monograph [29]. Effectiveness of direct computing method and method of successive approximations, variational-iterative method, decomposition method, method of series and other methods is illustrated by solving the problems for integro-differential equations. Conditions for the existence of solution to the considered problems are not given in this monograph. This is typical for the most of works devoted to the approximate methods for finding the solutions to initial and boundary value problems for the Fredholm integro-differential equations. Under assumption on

the solvability of problems they are focused on the algorithms for finding their approximate solutions.

Despite the fact that classical analytical solutions handle most of the cases, there are still problems in the cases when an integral equation is not solvable (i.e. uninvertibility of the matrices). The parametrization and interval partition method proposed by Dr. D. Dzhumabaev can be used to deal with this problem. This method with proofs and variations is fully described in his works [2,3,4].

## 1.2 Applications of integro-differential equations

The study of integro-differential equations is motivated by real world applications. There are many situations in which a nonlocal equation gives a significantly better model than a Partial Differential Equations, as explained next. In Mathematical Finance it is particularly important to study models involving jump processes, since the prices of assets are frequently modeled by stochastic processes. Note that jump processes are very natural in this situation, since asset prices can have sudden changes.

These models have become increasingly popular for modeling market fluctuations since the work of Merton [5] in 1976, both for risk management and option pricing purposes. For example, the obstacle problem for the fractional Laplacian can be used to model the pricing of American options [6, 7]; see also the nice introduction of [8] and also [9, 10]. Good references for financial modeling with jump processes are the books [11] and [12]. Just as an example, let us mention that in [13] Nolan examined the joint distribution of the German mark and the Japanese yen exchange rates, and observed that the distribution fits well in a integro-differential model.

Another real world example of integro-differential equation usage is option prices prediction. Particularly, precise link between option prices in exponential L´evy models and the related partial integro-differential equations (PIDEs) in the case of European options and options with single or double barriers was explored in [27] . The shortcomings of diffusion models in representing the risk related to large market movements have led to the development of various option pricing models with jumps, where large returns are represented as discontinuities of prices as a function of time. Models with jumps allow for more realistic representation of price dynamics and a greater flexibility in modelling.They applied the Markov property of the price which allowed them to express option prices as solutions of partial integro-differential equations (PIDEs) which involve, in addition to a (possibly degenerate) second-order differential operator, a non-local integral term which requires specific treatment both at the theoretical and numerical level.

Integro-differential equations appear also in Ecology. For instance, optimal search theory predicts that predators should adopt search strategies based on long jumps where prey is sparse and distributed unpredictably, Brownian motion being more efficient only for locating abundant prey; see [14]. Thus, reaction-diffusion problems with nonlocal diffusion such as

$$u_t + Lu = f(u) \quad \text{in} \quad \mathbb{R}^n$$

arise naturally when studying such population dynamics. Given equation appear also in physical models of plasmas and flames; see [15].

Another application is in describing biological and economical populations. Let u(x, t) be the density of some population. The space variable x can be the usual physical space or the space of some morphological parameter, such as the weight of the individuals in the population, the size of beaks for birds and so on. The nonlocal reaction-diffusion equation [42]

$$\frac{\partial u}{\partial t} = d\,\frac{\partial^2 u}{\partial x^2} + F(u, J(u)), \tag{1.7}$$

describes the evolution of the population density. Here F is the rate of reproduction of the population. Let

$$F(u, J(u)) = ku(K - bJ(u)), \quad J(u) = \int_\Omega \phi(x - y)u(y, t)dy, \tag{1.8}$$

where Ω is the spatial domain. If the kernel φ is a δ-function, then (1.8) coincides with the logistic equation. In this case, the reproduction rate is proportional to the density of the population u and to rate of change of the quantity of resources (K − bu). Here K is the rate of production of resources and bu the rate of their consumption. If individuals in the population can consume resources in some area around their average location, then consumption of resources becomes nonlocal, and the reproduction rate depends on the integral J(u).

If the space variable x corresponds to the physical space, then the diffusion term in equation (1.7) describes random motion of individuals in the population. In the case of a morphological parameter, the diffusion term corresponds to random mutations in offsprings which result in the variation of their morphological characteristics in comparison with their parents.

Stability and bifurcation analysis of the homogeneous-in-space solution allows us to make some conclusions about the evolution of the population. Integrating (1.7), we obtain

$$\int_{-\pi}^{\pi} u(x)dx = 2\pi u_0 + \frac{\pi m^2}{u_0 k_0}\,\epsilon^2.$$

Fig. 1.3. presents the numerical simulations of equation (1.7). The support of the initial condition is located at the center of the interval. After some time, the maxima of the solution go from the center to the extremities of the interval. It happens because the competition for resources there is less. Reprinted with permission from [42].

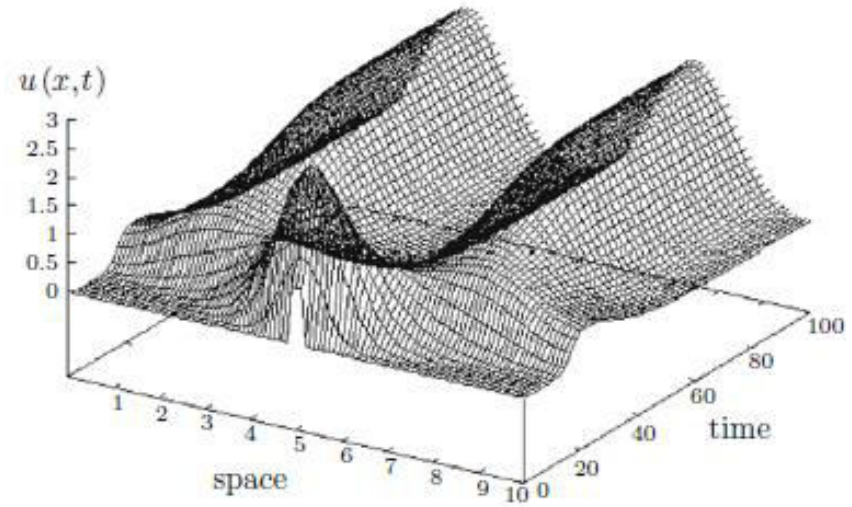


Figure 1.3. Numerical simulations of equation (1.7)

Hence the total size of the population in the case of a nonhomogeneous distribution increases in comparison with the homogeneous distribution. We will discuss below in this chapter that nonlocal reaction-diffusion equations provide a model to describe the emergence of biological species. In this case, x is a morphological parameter and the peaks of the population density correspond to different species. In the framework of this interpretation, this speciation begins as instability of the homogeneous-in-space solution. It is advantageous from the point of view of growth of the total biomass.

Another observation concerns the type of the bifurcation. In the case of a supercritical bifurcation, stable nonhomogeneous solutions gradually increase their amplitude when the bifurcation parameter crosses the critical value. In the case of a subcritical bifurcation, the transition from the homogeneous solution to a stable nonhomogeneous solution is discontinuous. Small amplitude solutions are unstable, and only nonhomogeneous solutions with a finite amplitude can be observed. This can be a possible explanation of the fact that intermediate forms during transition from one species to another are not known.

Natural selection is the mechanism which preserves favorable variations and removes injurious variations. It can result in the appearance of new species. In nonlocal reaction-diffusion models, this process manifests itself as instability of a homogeneous-in-space solution and emergence of spatial structures. Darwin discussed how various factors could act on natural selection, stimulate or retard it. We will consider in this section how some of these factors act on stability of solutions. Nonlocal reaction-diffusion equations and natural selection. Nonlocal reaction-diffusion equations describe intra-specific competition, reproduction of the population and random mutations (variations). These three conditions represent the minimal model to describe the instability of the homogeneous-in-space solutions and appearance of spatial structures.

How this model is related to natural selection which acts to preserve favorable variations and which is, according to Darwin, the main driving force leading to the emergence of species?

In order to answer this question, let us consider the result of numerical simulations of equation (1.7) with nonlinearity (1.8) (Fig 1.3). The initial condition is a step-wise constant function with a support in the center of the interval. After some time, when the density u(x, t) of the population in the center of the interval becomes sufficiently large, the individuals located far from the center have an advantage because the competition for resources there is weaker. Hence they begin to reproduce faster.

Thus, natural selection acts here according to Darwin's description. Favorable variations are transmitted to descendants. This property is not explicitly included in the model but it is described by this model due to its three features: nonlocal consumption of resources, reproduction, random variations. They result in the splitting of the population into two sub-populations.

Geographical isolation. We studied nonlocal reaction-diffusion equations either in the case of the usual physical space or in the space of genotypes. Consider now the two-dimensional case where one space variable $x_1$ corresponds to the physical space and another one, $x_2$ to the space of genotypes. It will allow us to analyze how the distribution of the population in space can influence its speciation. We consider equation (1.7) with possibly different diffusion coefficients for the two space variables. The eigenvalues of the linearized problem are given by formula (1.8). Thus, in some cases the size of the domain can influence the instability conditions and the emergence of spatial structures.

Several multi-scale models involves the models of living organisms where different levels, cellular, intracellular, tissue and organ, the whole organism interact with each other. Multi-scale models can also be considered for human populations. We will consider here an example where the population interacts with its economic environment. Let wealth u be described by the equation

$$\frac{\partial u}{\partial t} = d\,\frac{\partial^2 u}{\partial x^2} + ku(K(v) - b(v)u). \tag{1.9}$$

Nonlocal consumption of resources can also be considered. We suppose that the rate of production of resources K and the coefficient b in the consumption of resources depend on the density v of the population. On the other hand, this density is described by the equation (1.9).

It has two homogeneous-in-space solutions u = 0 and u = β. If this equation is considered on the whole axis, then there exist travelling wave solutions which provide transition between these stationary points. In fact, monotone waves exist for all values of the speed greater than or equal to the minimal speed. If we decrease w0 for all other parameters fixed, then the minimal speed increases. Hence the rate of economical invasion is greater for small values of relative wealth: poor population migrates faster.

Another example arises in economical populations. Consider a human population uniformly distributed in space. We suppose that individuals in this population do not move and that they possess some wealth u(x, t). Wealth can be considered as a function of the space variable x (since individuals are fixed in space) and of time t. Similar to the density of the biological population, under some assumptions the evolution of wealth can be described by equation (1.7). Production is proportional to the current wealth u and to available resources. If consumption of resources is nonlocal, then we will have the integral term J(u) in the equation. The diffusion term describes local redistribution (from rich to poor).

Consider the diffusion coefficient d as a bifurcation parameter. The stationary equation is written

$$u'' + \sigma\, u(1 - au - bJ(u)) = 0, \tag{1.10}$$

where σ = k/d. If we decrease d, and, consequently, increase σ, then for d = dc and some critical value dc, the homogeneous-in-space solution can lose its stability resulting in appearance of stable nonhomogeneous solutions. Thus, if redistribution of wealth is sufficiently small, then it becomes nonhomogeneous-in-space with some rich and poor regions (individuals). Here J:

$$F(u, J(u)) = ku(K - bJ(u)), \quad J(u) = \int_\Omega \phi(x - y)u(y, t)dy, \tag{1.11}$$

Nonhomogeneous-in-space solutions can facilitate transition from the lower branch to the upper branch of this curve (or conversely). Decreasing the value of b (consumption of resources) provides transition to the high-wealth branch, while increasing consumption, to the low-wealth branch. In the case of the whole axis, transition between different stationary solutions can propagate as a traveling wave.

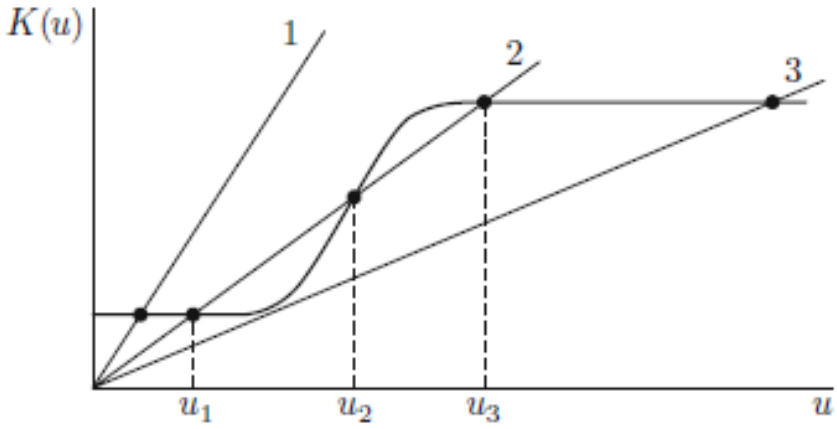


Figure 1.4. Graphical solution of equation (1.10). Depending on the value of b, there are from one to three solutions.

Due to this model, the total wealth of the population is greater in the case of a nonhomogeneous distribution than in the case of the homogeneous one. Hence redistribution of wealth decreases the total wealth of the population.

Let us also note that parameters of the model can depend on wealth. In particular, the rate of production of resources K can depend on u. This is the case of mineral resources which require certain level of investments for their production or technological resources which require research and education. Then the homogeneous-in-space solution can be found from the equation

$$K(u) = bu.$$

Depending on the function K(u) and on the value of b, it can have more than one solution with different values of wealth (Fig. 1.4.).

Transition of the function K(u) from low values to high values can be associated with technological revolution or with some other reasons, for example, financial investments. Nonlocal consumption of resources can also be considered. We suppose that the rate of production of resources K and the coefficient b in the consumption of resources depend on the density v of the population.

In the second case shown in the Fig. 1.4., where there are three solutions, $u_1$ and $u_3$ are stable solutions for the local reaction-diffusion equation. They can become unstable in the nonlocal case. Nonhomogeneous-in-space solutions can facilitate transition from the lower branch to the upper branch of this curve (or conversely). Decreasing the value of b (consumption of resources) provides transition to the high-wealth branch, while increasing consumption, to the low-wealth branch. In the case of the whole axis, transition between different stationary solutions can propagate as a traveling wave.

In Fluid Mechanics, many equations are nonlocal in nature. A clear example is the surface quasi-geostrophic equation, which is used in oceanography to model the temperature on the surface [16]. The regularity theory for this equation relies on very delicate regularity results for nonlocal equations in divergence form; see [17, 18]. Another important example is the Benjamin-Ono equation

$$(-\Delta)^{1/2}u = -u + u^2,$$

which describes one-dimensional internal waves in deep water [19]. Also, the half-Laplacian plays a very important role in the understanding of the gravity water waves equations in dimensions 2 and 3. In Elasticity, there are also many models that involve nonlocal equations.

An important example is the Peierls-Nabarro equation, arising in crystal dislocation models [20]. Also, other nonlocal models are used to take into account that in many materials the stress at a point depends on the strains in a region near that point [21]. Long range forces have been also observed to propagate along fibers or laminae in composite materials, and nonlocal models are important also in composite analysis [22].

Other Physical models arising in macroscopic evolution of particle systems or in phase segregation lead to nonlocal diffusive models such as the fractional porous media equation [23]. Related evolution models with nonlocal effects are used in superconductivity [24]. Moreover, other continuum models for interacting particle systems involve nonlocal interaction potentials.

Other examples in which integro-differential equations are used are Image Processing (where nonlocal denoising algorithms are able to detect patterns and contours in a better way than the local PDE based models [25]) and Geometry (where the conformally invariant operators, which encode information about the manifold, involve fractional powers of the Laplacian [26]).

## 2 Parametrization method for solving the integro-differential equations

### 2.1 Parametrization method for solving the linear boundary value problem for the equation with degenerate kernel

As it was mentioned above, the integro-differential equations can be solved analytically by several ways. Reduction of problems for integro-differential equations to the integral equations is the main tool in investigating and solving the considered problems. For instance, Nekrasov's method and Green's function method are frequently used ones. Despite the fact that classical analytical solutions handle most of the cases, there are still problems in the cases when an integral equation is not solvable. Especially, the Fredholm integro-differential equation has a row of features that must be taken into consideration while creating the approximate methods for finding its solutions.

Let's consider the equation

$$\frac{dx}{dt} = Ax + \frac{1}{2\pi}\int_0^{2\pi} Bx(s)ds + f, t \in (0, 2\pi), x \in R^2,$$

where

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, f = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

As follows from the results of [33, p.71] this Fredholm integro-differential equation has no solution. At the same time the linear ordinary differential equations and linear Volterra integro-differential equations are solvable for any right-hand side f , and they have a family of solutions. For the linear ordinary differential equation on [a, b] the Cauchy problem with initial condition given at any point of this interval has a unique solution. If we consider the linear Volterra integro-differential equation on [a, b] , and the left-end of interval is the lower limit of integral term, then the Cauchy problem for this equation with condition at the point a is also uniquely solvable. Therefore, the Cauchy problem is often used as an intermediate problem under investigating the qualitative properties of boundary value problems for the ordinary differential equations and Volterra integro-differential equations.

However, the Cauchy problem for the Fredholm integro-differential equation can be unsolvable, but the boundary value problem for this equation may have a unique solution. Indeed, if we suppose that a function x*(t) satisfies the integro-differential equation

$$\frac{dx}{dt} = 2\int_0^1 x(s)ds + 2t - \frac{2}{3}, t \in (0, 1), x \in R^1$$

and the initial condition x(0) = 1, then after certain easy calculations we come to the contradiction: $-1 = 0$ . As it was shown in [4, p. 1160], the boundary value problem for this equation with the condition x(0) = x(1) has a unique solution x(t) = t(t − 1).

These features of Fredholm integro-differential equations require an examination of qualitative properties of the problem (1.5), (1.6), before starting to construct the approximate methods for its solving. The numerical methods which solve Fredholm integro-differential equations given in this work based on parametrization method proposed by Dr. Dzhumabaev D.S. [2]. This method is based on dividing the interval [0, T] into N parts and introducing the additional parameters. While applying the method to problem (1.5), (1.6), the necessity of solving an intermediate problem also arises. Intermediate problem here is the special Cauchy problem for the system of integro-differential equations with parameters. But unlike the intermediate problems of above mentioned methods, the special Cauchy problem is always uniquely solvable for sufficiently small partition step. This property of intermediate problem allowed establish in [6] the necessary and sufficient conditions for solvability and unique solvability of problem (1.5),(1.6). Below, the description of this method supported by theorems and proofs is given.

Consider the Fredholm integro-differential equation of the second kind with degenerate kernel

$$\frac{dx}{dt} = A(t)x + \sum_{k=1}^{m} \int_0^T \varphi_k(t)\psi_k(s)x(s)ds + f(t), t \in (0,T), x \in R^n \tag{2.1}$$

$$Bx(0) + Cx(T) = d, d \in R^n \tag{2.2}$$

where the matrices A(t), $\varphi_k(t)$, $\psi_k(s)$, 0<k<m and vector f(t) are continuous on [0, T]. Given the points: $t_0 = 0 < t_1 < \ldots < t_N = T$ , and let $\Delta_N$ denote the partition of interval [0, T) into N subintervals $[0, T) = \bigcup_{r=1}^{N} [t_{r-1}, t_r)$ . [2]

The case, when the interval [0, T] is not divided into parts, we denote by $\Delta_1$.

Let $x_r(t)$ be the restriction of function x(t) to the r -th interval $[t_{r-1}, t_r)$, i.e. $x_r(t) = x(t)$, for $t \in [t_{r-1}, t_r)$, r = 1,N.

Introducing the additional parameters $\lambda_r = x_r(t_{r-1})$ and performing a replacement of the function $u_r(t) = x_r(t) - \lambda_r$ on each r -th interval, we obtain the boundary value problem with the following parameters:

$$\frac{du_r}{dt} = A(t)(u_r + \lambda_r) + \sum_{j=1}^{N} \sum_{k=1}^{m} \int_{t_{j-1}}^{t_j} \varphi_k(t)\psi_k(s)(u_r(s) + \lambda_r)ds + f(t)$$

$$t \in (t_{r-1}, t_r),\ r = \overline{1, N},$$

(2.3)

$$u_r(t_{r-1}) = 0, r = 1...N \tag{2.4}$$

$$B\lambda_1 + C\lambda_N + C \lim_{t \to T-0} u_N(t) = d \tag{2.5}$$

$$\lambda_p + \lim_{t \to t_p - 0} u_p(t) - \lambda_{p+1} = 0, p = 1...N-1 \tag{2.6}$$

where (2.6) are conditions for matching the solution at the interior points of the partition $\Delta_N$. Note, that conditions (2.6) and integro-differential equations (2.3) also ensure the continuity of solution's derivatives at these points.

Using the fundamental matrix $X_r(t)$ of differential equation dx/dt = A(t)x on $[t_{r-1}, t_r]$ , we reduce the special Cauchy problem for the system of integro-differential equations with parameters (2.3), (2.4) to the equivalent system of integral equations.

$$\begin{aligned} u_r(t) &= X_r \int_{t_{r-1}}^{t} X_r^{-1}(\tau) A(\tau) d\tau \lambda_r + \\ &+ X_r \int_{t_{r-1}}^{t} X_r^{-1}(\tau) A(\tau) \sum_{j=1}^{N} \sum_{k=1}^{m} \int_{t_{j-1}}^{t_j} \varphi_k(t) \psi_k(s) (u(s)_r + \lambda_r) ds d\tau + \\ &+ X_r \int_{t_{r-1}}^{t} X_r^{-1}(\tau) A(\tau) f(\tau) d\tau, t \in [t_{r-1}, t_r), r = 1...N \end{aligned} \tag{2.7}$$

Multiplying both sides of (2.7) by $\psi_p(t)$ , integrating on the interval $[t_{r-1}, t_r]$ and summing up over r , we have the system of linear algebraic equations with respect to $\mu = (\mu_1, \ldots, \mu_m) \in R^{nm}$:

$$\mu_p = \sum_{k=1}^{m} G_{p,k}(\Delta_N) \mu_k + \sum_{r=1}^{N} V_{p,r}(\Delta_N) \lambda_r + g_p(f, \Delta_N), p = 1...m \tag{2.8}$$

where

$$\mu_k = \sum_{j=1}^{N} \int_{t_{j-1}}^{t_j} \psi_k(s) u_j(s) ds$$

with the (n × n) matrices

$$G_{p,k}(\Delta_N) = \sum_{r=1}^{N} \int_{t_{r-1}}^{t_r} \psi_r(\tau) X_r(\tau) \int_{t_{r-1}}^{\tau} X_r^{-1}(\tau) \phi_k(s) ds d\tau \tag{2.9}$$

$$\begin{aligned} V_{p,r}(\Delta_N) &= \int_{t_{r-1}}^{t_r} \psi_p(\tau) X_r(\tau) \int_{t_{r-1}}^{\tau} X_r^{-1}(\tau) A(s) ds d\tau + \\ &+ \sum_{j=1}^{N} \sum_{k=1}^{m} \int_{t_{j-1}}^{t_j} \psi_p(\tau) X_j(\tau) \int_{t_{j-1}}^{\tau} X_j^{-1}(\tau_1) \phi_k(\tau_1) d\tau_1 d\tau \int_{t_{r-1}}^{t_r} \psi_k(s) ds \end{aligned} \tag{2.10}$$

and vectors of dimension n

$$g_p(f,\Delta_N)=\sum_{r=1}^{N}\int_{t_{r-1}}^{t_r}\psi_p(\tau)X_r(\tau)\int_{t_{r-1}}^{\tau}X_r^{-1}(\tau)f(s)dsd\tau \tag{2.11}$$

Using these matrices we can rewrite the system (2.8) in the form

$$[I-G(\Delta_N)]\mu=V(\Delta_N)\lambda+g(f,\Delta_N), \tag{2.12}$$

where I is the identity matrix of dimension nxm.

**Definition 2.1** Partition $\Delta_N$ is called regular if the matrix $I - G(\Delta_N)$ is invertible. It is established that the invertability of system's matrix is equivalent to the well-posedness of considered boundary value problem [2].

Any fundamental matrix of differential equation on $[t_{r-1}, t_r]$ has the form $X_r(t) = X^0_r(t) \cdot C_r$, where $X^0_r(t)$ is the normal fundamental matrix ($X^0_r(t_{r-1}) = I$) and $C_r$ is an arbitrary invertible matrix. Therefore, the equalities

$$G_{p,k}(\Delta_N)=\sum_{r=1}^{N}\int_{t_{r-1}}^{t_r}\psi_p(\tau)X_r^0(\tau)C_r\int_{t_{r-1}}^{\tau}[X_r^0(s)C_r]^{-1}\phi_k(s)dsd\tau=$$

$$=\sum_{r=1}^{N}\int_{t_{r-1}}^{t_r}\psi_p(\tau)X_r^0(\tau)\int_{t_{r-1}}^{\tau}[X_r^0(s)]^{-1}\phi_k(s)dsd\tau \tag{2.13}$$

hold, and the regularity of partition $\Delta_N$ does not depend on the choice of fundamental matrix of equation's differential part.

Let $\sigma(m, [0, T])$ denote the set of regular partitions $\Delta_N$ of $[0, T]$ for the equation (2.1).

**Definition 2.2** The special Cauchy problem (2.3), (2.4) is called uniquely solvable, if for any $\lambda \in R^{nN}$, $f(t) \in C([0, T],R^n)$ it has a unique solution.

Special Cauchy problem (2.3), (2.4) is equivalent to the system of integral equations (2.7). This system by virtue of the kernel degeneracy is equivalent to the system of algebraic equations (2.9) with respect to $\mu = (\mu_1, \ldots, \mu_m) \in R^{nm}$. Therefore, the special Cauchy problem is uniquely solvable if and only if the partition $\Delta_N$, generating this problem, is regular.

Since the special Cauchy problem is uniquely solvable for sufficiently small partition step $h > 0$. Then according to (2.12) the elements of the vector $\mu \in R^{nm}$ can be determined by the equalities

$$\mu_k=\sum_{j=1}^{N}(\sum_{p=1}^{m}M_{k,p}(\Delta_N)V_{p,r}(\Delta_N))\lambda_j+\sum_{p=1}^{m}M_{k,p}(\Delta_N)g_p(f,\Delta_N), \tag{2.14}$$

Where $[I-G(\Delta_N)]^{-1} = \left(M_{k,p}(\Delta_N)\right)$ are the square matrices of dimension n. Essential requirement to the partition is its regularity. If the matrix $I - G(\Delta_N)$ is not invertible, then other splitting of interval [0, T] must be chosen. Then, in (2.7), substituting the right-hand side of (2.13) instead of $\mu_k$ , we get the representation of functions $u_r(t)$ via $\lambda_j$

$$\begin{aligned} u_r(t) = \sum_{j=1}^{N} \Bigg\{ \sum_{k=1}^{m} X_r(t) \int_{t_{r-1}}^{t} X_r^{-1}(\tau)\phi_k(\tau) \\ \times \left[ \sum_{p=1}^{m} M_{k,p}(\Delta_N) V_{p,j}(\Delta_N) + \int_{tj-1}^{t_j} \psi_k(s)ds \right] \lambda_j + \\ + X_r(t) \int_{t_{r-1}}^{t} X_r^{-1}(\tau)A(\tau)d\tau\lambda_r + X_r(t) \int_{t_{r-1}}^{t} X_r^{-1} \times \\ \times \left[ \sum_{k=1}^{m} \phi_k(\tau) \sum_{p=1}^{m} M_{k,p}(\Delta_N) g_p(f, \Delta_N) + f(\tau) \right] d\tau, \end{aligned} \tag{2.15}$$

Introduce notations

$$\begin{aligned} D_{r,j}(\Delta_N) = \sum_{k=1}^{m} X_r(t) \int_{t_{r-1}}^{t} X_r^{-1}(\tau)\phi_k(\tau) \times \\ \times \left[ \sum_{p=1}^{m} M_{k,p}(\Delta_N) V_{p,j}(\Delta_N) + \int_{tj-1}^{t_j} \psi_k(s)ds \right] \end{aligned} \tag{2.16}$$

$$\begin{aligned} D_{r,r}(\Delta) = \sum_{k=1}^{m} X_r(t_r) \int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)\phi_k(\tau)d\tau \times \\ \times \left[ \sum_{p=1}^{m} M_{k,p}(\Delta_N) V_{p,j}(\Delta_N) + \int_{tj-1}^{t_j} \psi_k(s)ds \right] + \end{aligned} \tag{2.17}$$

$$\begin{aligned} + X_r(t_r) \int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)A(\tau)d\tau \\ F_r(\Delta_N) = \sum_{k=1}^{m} X_r(t_r) \int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)A(\tau)d\tau \sum_{p=1}^{m} M_{k,p}(\Delta_N) g_p(f, \Delta_N) + \\ + \sum_{k=1}^{m} X_r(t_r) \int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)f(\tau)d\tau, r = 1...N, \end{aligned} \tag{2.18}$$

Then from (2.14) we have

$$\lim_{t \to t_r - 0} u_r(t) = \sum_{j=1}^{N} D_{r,j}(\Delta_N)\lambda_j + F_r(\Delta_N). \tag{2.19}$$

Substituting the right-hand side of (2.18) into the boundary condition (2.5) and conditions of matching solution (2.6), we obtain the following system of linear algebraic equations with respect to parameters $\lambda_r$, r = (1,N) :

$$\begin{aligned}[B + CD_{N,1}(\Delta_N)]\,\lambda_1 + \sum_{j=2}^{N-1} CD_{N,j}(\Delta_N)\lambda_j + \\ + C\,[I + D_{N,N}(\Delta_N)]\,\lambda_N = d - CF_N(\Delta_N),\end{aligned} \tag{2.20}$$

$$\begin{aligned}[I + D_{p,p}(\Delta_N)]\,\lambda_p - [I + D_{p,p+1}(\Delta_N)]\,\lambda_{p+1} + \\ + \sum_{j=1}^{N} D_{p,j}(\Delta_N)\lambda_j = -F_p(\Delta_N), p = 1 \ldots N-1\end{aligned} \tag{2.21}$$

By denoting the matrix corresponding to the left-hand side of the system of equations (2.19), (2.20) by Q, the system can be written as the following:

$$Q_*(\Delta_N)\lambda = -F_*(\Delta_N), \lambda \in R^{nN}, \tag{2.22}$$

where

$$F_*(\Delta_N) = (-d + CF_N(\Delta_N), F_1(\Delta_N), \ldots, F_{N-1}(\Delta_N)) \in R^{nN} \tag{2.23}$$

Then from the equation (2.21) we can easily find λ and substitute it to (2.14) to calculate u. Finally, performing a replacement of the function $u_r(t) = x_r(t) - \lambda_r$ on each r -th interval to obtain the values of the vector function x.

**Lemma 2.1** For $\Delta_N \in \sigma(m, [0, T])$ the following assertions hold:

a) the vector $\lambda^* = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*) \in R^{nN}$, composed by the values of solution x*(t) to problem (2.1), (2.2) at the partition points $\lambda_r^* = x^*(t_{r-1})$, r = 1,N , satisfies the system (2.21);

b) if $\lambda = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*) \in R^{nN}$ is a solution to the system (2.21) and the function system $u(t) = (u_1^*(t), u_2^*(t), \ldots, u_N^*(t))$ is a solution to the special Cauchy problem (2.3), (2.4) with $\lambda_r = \lambda_r^*$, r = 1…N, then the function x*(t), defined by the equalities: $x^*(t) = \lambda_r^* + u_r^*(t)$, $t \in [t_{r-1}, t_r)$, r = 1…N, $x^*(T) = \lambda_N^* + \lim_{n \to \infty} u_r^*(t)$ is a solution to problem (2.1), (2.2).

The proof with minor changes is similar in spirit to the proof of Lemma 1 [4, p. 1155]. Let us introduce the notations

$$\alpha = \max_{t \in [0,T]} ||A(t)||, \overline{w} = \max_{r=1 \ldots N} (t_r - t_{t-1}),$$

$$\overline{\phi}(m) = \max_{r=1...N} \int_{t_{r-1}}^{t_r} ||\phi_k(t)||dt,$$

$$\overline{\psi}(T) = \max_{r=1...m} \int_0^T ||\psi_p(t)||dt.$$

**Theorem 2.1** Let $\Delta_N \in \sigma(m, [0, T])$ and the matrix $Q_*(\Delta_N) : R^{nN} \to R^{nN}$ be invertible. Then problem (2.1),(2.2) has a unique solution x*(t) for any $f(t) \in C([0, T], R^n)$, $d \in R^n$, and the estimate holds:

$$||x^*||_1 \le N(m, \Delta_N) \max(||d||, ||f||_1), \tag{2.24}$$

where

$$N(m, \Delta_N) = e^{\alpha\overline{w}} \left\{ \overline{\phi}(m)[||[I - G(\Delta_N)]^{-1}||]\overline{\psi}(T) \times \right.$$

$$\times \left( e^{\alpha\overline{w}} - 1 + e^{\alpha\overline{w}}\overline{\phi}(m)\overline{\psi}(T) \right) + \overline{\psi}(T)] + 1 \} \gamma_*(\Delta_N) \times$$

$$\times (1 + ||C||) max \left\{ 1, \overline{w}e^{\alpha\overline{w}}[1 + e^{\alpha\overline{w}}\overline{\phi}(m)||[I - G(\Delta_N)]^{-1}||\overline{\psi}(T) \right\} +$$

$$+ \overline{w}e^{\alpha\overline{w}}\overline{w}[\overline{\phi}(m)]||[I - G(\Delta_N)]^{-1}||\overline{\psi}(T)e^{\alpha\overline{w}} + 1]. \tag{2.25}$$

**Proof.** Let $\Delta_N \in \sigma(m, [0, T])$ and $f(t) \in C([0, T], R^n)$, $d \in R^n$ be set. Using the invertibility of matrix $Q^*(\Delta_N)$ , we find a unique solution to the system of linear algebraic equations (2.22):

$$Q_*(\Delta_N)\lambda = -F_*(\Delta_N), \lambda \in R^{nN},$$

Solving the special Cauchy problem (2.3), (2.4) with $\lambda = \lambda^*$ , define the function system $u^*[t] = (u_1^*(t), u_2^*(t), \ldots, u_N^*(t))$. The regularity of partition $\Delta N$ leads to the existence of unique function system u*[t] with the elements $u_r^*(t)$, defined by the right-hand side of (2.15) at $\lambda = \lambda^* = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*) \in R^{nN}$. Then, according to Lemma 2.1, the function x(t), defined by the equalities: $x^*(t) = \lambda_r^* + u_r^*(t)$, $t \in [t_{r-1}, t_r)$, $r = 1,N$, $x^*(T) = \lambda_N^* + \lim_{n\to\infty} u_r^*(t)$ is a solution to problem (2.1), (2.2). Uniqueness of solution is proved by contradiction. Let us prove the estimate (2.24).

From the equalities

$$X_r(t) \int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)P(\tau)d\tau = \int_{t_{r-1}}^{t_r} P(\tau_1)d\tau_1 +$$

$$+ \int_{t_{r-1}}^{t_r} A(\tau_1) \int_{t_{r-1}}^{\tau_1} P(\tau_2)d\tau_2 d\tau_1 + \int_{t_{r-1}}^{t} A(\tau_1) \int_{t_{r-1}}^{\tau_1} A(\tau_2) \int_{t_{r-1}}^{\tau_2} P(\tau_3)d\tau_3 d\tau_2 d\tau_3 + \ldots$$

$$t \in [t_{r-1}, t_r] \tag{2.26}$$

we obtain the estimates

$$||X_r(t_r)\int_{t_{r-1}}^{t_r} X_r^{-1}(\tau)\phi_k(\tau)d\tau|| \le e^{\alpha(t_{r-1}-t_r)}\int_{t_{r-1}}^{t_r}||\phi_k(t)||dt,\ r=1...N \tag{2.27}$$

Using (2.12), (2.18), (2.27) we obtain the inequalities

$$\begin{aligned}
&||g_p(f,\Delta_N)|| \le \sum_{r=1}^{N}\int_{t_{r-1}}^{t_r}||\psi_p(\tau)||\cdot||X_r(\tau)\int_{t_{r-1}}^{t_r}X_r^{-1}(s)f(s)ds||d\tau \le \\
&\le \sum_{r=1}^{N}\int_{t_{r-1}}^{t_r}||\psi_p(\tau)||d\tau\cdot e^{\alpha\overline{w}}\overline{w}||f||_1 = \int_0^T||\psi_p(t)||dt\cdot e^{\alpha\overline{w}}\overline{w}||f||_1,\ p=1...m \\
&||F_r(\Delta_N)|| \le e^{\alpha w_r}\sum_{k=1}^{m}\int_{t_{r-1}}^{t_r}||\psi_k(t)||dt||[I-G(\Delta_N)]^{-1}||\times \\
&\times \max_{p=1...m}||g_p(f,\Delta_N)||e^{\alpha w_r}w_r||f||_1.
\end{aligned} \tag{2.28}$$

From the inequality

$$||F_*(\Delta_N)|| \le max(||d||, \max_{r=1...N}||F_r(\Delta_N)||)(1+||C||),$$

and on the basis of (2.16),(2.24) and (2.28), we have

$$\begin{aligned}
\|F_*(\Delta_N)\| \le (1+\|C\|)\max\Big\{1, \overline{\omega}e^{\alpha\overline{\omega}}\Big[1+e^{\alpha\overline{\omega}}\max_{r=\overline{1,N}}\int_{t_{r-1}}^{t_r}\sum_{k=1}^{m}\|\varphi_k(t)\|dt\|[I-G(\Delta_N)]^{-1}\|\times \\
\times\max_{p=\overline{1,m}}\int_0^T\|\psi_p(t)\|dt\Big]\Big\}\max(\|d\|,\|f\|_1).
\end{aligned} \tag{2.29}$$

Equality (2.22), inequality (2.29) and the invertibility of matrix Q*(ΔN) yield the next estimate

$$\begin{aligned}
&\|\lambda^*\| \le \|[Q_*(\Delta_N)]^{-1}\|\|F_*(\Delta_N)\| \le \gamma_*(\Delta_N)(1+\|C\|)\max\Big\{1, \\
&e^{\alpha\overline{w}}\overline{w}[1+e^{\alpha\overline{w}}\max_{r=1...N}\int_{t_{r-1}}^{t_r}\sum_{r=1}^{m}||\phi_k(t)||dt]\cdot|||[I-G(\Delta_N)]^{-1}||\times \\
&\times\max_{r=1...m}\int_0^T\sum_{r=1}^{m}||\psi_p(t)||dt\}max(||d||,||f||_1).
\end{aligned} \tag{2.30}$$

Representations (2.15) and formulas (2.11), allow us obtain

$$\|u^*[\cdot]\|_2 \le \Big\{ e^{\alpha\overline{\omega}} \max_{r=\overline{1,N}} \int_{t_{r-1}}^{t_r} \sum_{k=1}^{m} \|\varphi_k(t)\|dt \|[I - G(\Delta_N)]^{-1}\| \max_{p=\overline{1,m}} \int_0^T \|\psi_p(t)\|dt \times$$

$$\times \Big( e^{\alpha\overline{\omega}} - 1 + e^{\alpha\overline{\omega}} \max_{r=\overline{1,N}} \int_{t_{r-1}}^{t_r} \max_{k=\overline{1,m}} \|\varphi_k(t)\|dt \max_{p=\overline{1,m}} \int_0^T \|\psi_p(t)\|dt \Big) +$$

$$+ e^{\alpha\overline{\omega}}\overline{\omega} \Big[ \max_{r=\overline{1,N}} \int_{t_{r-1}}^{t_r} \sum_{k=1}^{m} \|\varphi_k(t)\|dt \| \max_{p=\overline{1,m}} \int_0^T \|\psi_p(t)\|dt \cdot e^{\alpha\overline{\omega}} + 1 \Big] \cdot \|f\|_1.$$

Using (2.28), (2.29) and the relation $\|x^*\|_1 \le \|\lambda^*\| + \|u[\cdot]\|_2$ , we get the estimate (2.23).

Theorem 2.1 is proved.

**Definition 2.3** Problem (2.1), (2.2) is called well-posed if for any pair (f(t), d), $f(t) \in C([0, T],R^n)$, $d \in R^n$, it has a unique solution x(t) , and the next inequality holds:

$$\|x\|_1 \le K \max(\|f\|_1, \|d\|),$$

where K is a constant, independent of f(t) and d.

**Theorem 2.2** Problem (2.1),(2.2) is well-posed if and only if for any $\Delta_N \in \sigma(m, [0, T])$ the matrix $Q^*(\Delta_N) : R^{nN} \to R^{nN}$ is invertible.

**Proof.** At fixed m and $\Delta_N \in \sigma(m, [0, T])$ , the number $N(m,\Delta_N)$ , defined by (2.25), does not depend on f(t) and d. Therefore, the sufficiency of theorem conditions for the well-posedness of problem (2.1), (2.2) follows from Theorem 2.1.

Prove the necessity. Let problem (2.1), (2.2) be well-posed and $\Delta_N \in \sigma(m, [0, T])$. Let's assume the opposite, and the matrix $Q^*(\Delta_N) : R^{nN} \to R^{nN}$ is not invertible. This is possible only on the existence of non-zero solution to the homogeneous system of equations

$$Q_*(\Delta_N)\lambda = 0, \quad \lambda \in R^{nN}. \tag{2.31}$$

Assuming that $\lambda^* = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*)$ is the non-zero solution (i.e. $\| \lambda^* \| \neq 0$ ) to system (2.31), we consider the homogeneous boundary value problem (2.1), (2.2) with f(t) = 0, d = 0. For this problem, the system (2.22) coincides with (2.31). Therefore, according to Lemma 2.1, the function x*(t), defined by the equalities $x^*(t) = \lambda_r^* + u_r^*(t)$, $t \in [t_{r-1}, t_r)$, r = 1,N, $x^*(T) = \lambda_N^* + \lim_{n\to\infty} u_r^*(t)$ is a non-zero solution to the homogeneous boundary value problem. Here the function system $u^*[t] = (u_1^*(t), u_2^*(t), \ldots, u_N^*(t))$ is a solution to the special Cauchy problem (2.3), (2.4) with $\lambda = \lambda^*$ and f(t) = 0. This contradicts to the well-posedness of problem (2.1), (2.2).

Theorem 2.2 is proved.

2.2 Numerical Solution to the linear boundary value problem for the equation with degenerate kernel

Cauchy problems for ordinary differential equations on subintervals are significant part of proposed algorithm. Let's denote the solution to the Cauchy problem by $a_r(P,t)$ :

$$a_r(P,t) = X_r(t) \int_{t_{r-1}}^{t} X_r^{-1}(\tau) P(\tau) d\tau, t \in [t_{r-1}, t_r) \tag{2.32}$$

where $X_r(t)$ is a fundamental matrix of differential equation on the r -th interval. The equation (2.32) can be re-written by the following alternative way:

$$\frac{dx}{dt} = A(t)x + P(t), x(t_{r-1}) = 0, t \in [t_{r-1}, t_r], r = 1...N \tag{2.33}$$

Here P(t) is either (n×n) matrix, or n vector, both continuous on $[t_{r-1}, t_r]$, r = 1…N . Consequently, solution to problem (2.33) is a square matrix or a vector of dimension n.

Choosing of a regular partition is another significant part of this algorithm. Logically, the algorithm can be divided into the following steps:

I. Let us choose the partition of interval [0, T] into N parts: $t_0 = 0 < t_1 < \ldots < t_{N-1} < t_N = T$, which we denote by $\Delta_N$, N = 1, 2, ... .

II. Solving the Nm Cauchy problems for matrix ordinary differential equations

$$\frac{dx}{dt} = A(t)x + \phi_k(t), x(t_{r-1}) = 0, t \in [t_{r-1}, t_r] \tag{2.34}$$

we obtain the matrix functions

$$a_r(\phi_k, t), t \in [t_{r-1}, t_r], r = 1...N, k = 1...m \tag{2.35}$$

III. Multiply each (n × n) matrix (2.35) to the (n × n) matrix $\psi_p(t)$, p = 1…m and integrate on $[t_{r-1}, t_r]$ :

$$\widehat{\psi}_{p,r}(\phi_k) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(\phi_k, t) dt, \tag{2.36}$$

Summing up (2.36) over r, on the base of equalities (2.10), (2.32), we obtain the (n × n) matrices

$$G_{p,k}(\Delta_N) = \sum_{r=1}^{N} \widehat{\psi}_{p,r}(\varphi_k), \quad p,k = \overline{1,m}.$$

Compose the (nm × nm) matrix $G(\Delta_N) = (G_{p,k}(\Delta_N))$, p, k = 1…m and check the invertibility of matrix $[I - G(\Delta_N)] : Rnm \to Rnm$.

If this matrix is invertible, then we find its inverse and represent it in the form $[I - G(\Delta_N)]^{-1} = (M_{p,k}(\Delta_N))$, where $M_{p,k}(\Delta_N))$ is (n × n) matrix, p, k = 1…N . Then move on to the next step of algorithm.

If this matrix has no inverse, i.e. the partition $\Delta_N$ is not regular, then we take a new partition of interval [0, T] , and the algorithm starts over. The simplest way for selecting of a new partition is to choose the partition $\Delta_{2N}$, where each interval of partition $\Delta_N$ is divided into two parts.

IV. Solving again the Cauchy problem for the ordinary differential equations

$$\frac{dx}{dt} = A(t)x + A(t), \quad x(t_{r-1}) = 0, \quad t \in [t_{r-1}, t_r],$$

$$\frac{dx}{dt} = A(t)x + f(t), \quad x(t_{r-1}) = 0, \quad t \in [t_{r-1}, t_r], \quad r = \overline{1,N}$$

we find $a_r(A, t)$ and $a_r(f, t)$, r = 1…N.

V. Evaluate the integrals

$$\widehat{\psi}_{p,r}(A) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(A,t) dt,$$
$$\widehat{\psi}_{p,r}(f) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(f,t) dt,$$
$$\widehat{\psi}_{p,r} = \int_{t_{r-1}}^{t_r} \psi_p(t) dt,$$

From the equalities (2.11), (2.12) and (2.32) we define the (n × n) matrices

$$V_{p,r}(\Delta_n) = \widehat{\psi}_{p,r}(A) + \sum_{j=1}^{N} \sum_{k=1}^{m} \widehat{\psi}_{p,r}(\phi_k) \cdot \widehat{\psi}_{p,r}$$

and n vectors

$$g_p(f, \Delta_n) = \sum_{j=1}^{N} \widehat{\psi}_{p,r}(\Delta_N), p = 1...m, r = 1...N.$$

VI. Form the system of linear algebraic equations with respect to parameters

$$Q_*(\Delta_N)\lambda = -F_*(\Delta_N), \lambda \in R^{nN} \tag{2.37}$$

The elements of matrix $Q^*(\Delta_N) : RnN \to RnN$ and vector $F^*(\Delta_N) = (-d + CF_N(\Delta_N), F_1(\Delta_N, \ldots, F_{N-1}(\Delta_N)) \in R^{nN}$ are defined by the equalities (2.16), (2.17), (2.18). Solving the system (3.6), we find $\lambda = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*) \in R^{nN}$.

VII. By the equalities

$$\mu_k^* = \sum_{j=1}^{N} \Big( \sum_{p=1}^{m} M_{k,p}(\Delta_N) V_{p,r}(\Delta_N) \Big) \lambda_j^* + \sum_{p=1}^{m} M_{k,p}(\Delta_N) g_p(f, \Delta_N) \tag{2.38}$$

we find the components of $\mu^* = (\mu_1^*, \mu_2^*, \ldots, \mu_m^*) \in R^{nm}$ and construct the function

$$\mathcal{F}^*(t) = \sum_{k=1}^{m} \varphi_k(t) \Big[ \mu_k^* + \sum_{r=1}^{N} \int_{t_{r-1}}^{t_r} \psi_k(s) ds \lambda_r^* \Big] + f(t). \tag{2.39}$$

Recall that $\lambda_r^* = x^*(t_{r-1})$, where $x^*(t)$ is a solution to the boundary value problem (2.1), ( 2.2). Therefore, solving system (2.37), we find the values of solution at the left-end points of subintervals.

The values of function $x^*(t)$ at the other points of subinterval $[t_{r-1}, t_r)$ are determined by solving the following Cauchy problem for ordinary differential equation

$$\frac{dx}{dt} = A(t)x + \mathcal{F}^*(t), \quad x(t_{r-1}) = \lambda_r^*, \quad t \in [t_{r-1}, t_r), \quad r = \overline{1, N}$$

Thus, the offered algorithm consists of seven interconnected steps.

In the original paper [2] Numerical solution for Cauchy problems is implemented using Runge-Kutta 4$^{th}$ order method, whereas in this experiment Bulirsch-Stoer method was used [8].

Runge-Kutta method (also called as RK method) is the generalization of the concept used in Modified Euler's method. These methods were developed around 1900 by the German mathematicians C. Runge and M. W. Kutta. The advantage of Runge–Kutta method is their greater stability, especially when applied to stiff equations.

In Modified Euler's method the slope of the solution curve has been approximated with the slopes of the curve at the end points of the each sub interval in computing the solution. The natural generalization of this concept is computing the slope by taking a weighted average of the slopes taken at more number of points in each sub interval. However, the implementation of the scheme differs from Modified Euler's method so that the developed algorithm is explicit in nature. The final form of the scheme is of the form

$$y_{i+1} = y_i + (\textit{weighted average of the slopes}) \quad \textit{for } i = 0, 1, 2 \ldots \tag{2.40}$$

where h is the step length and $y_i$ and $y_{i+1}$ are the values of y at $x_i$ and $x_{i+1}$ respectively.

In genaral, the slope is computed at various points x's in each sub interval [$x_i$, $x_{i+1}$] and multiplied them with the step length h and then weighted average of it is then added to $y_i$ to compute $y_{i+1}$. Thus the RK method with v slopes called as v-stage RK method and can be written as

$$K1 = h\,f(x_i, y_i)$$
$$K2 = h\,f(x_i + c_2h, y_i + a_{21}K_1)$$
$$K3 = h\,f(x_i + c_3h, yi + a_{31}K_1 + a_{32}K_2)$$
. . .
. . .
. . .
$$Kv = h\,f(x_i + c_vh, y_i + a_{v1}K_1 + a_{v2}K_2 + \ldots + a_{vv-1}K_{v-1})$$
*and*
$$y_{i+1} = y_i + (W_1 K_1 + W_2 K_2 + \ldots + W_v K_v) \qquad \text{for } i = 0, 1, 2 \ldots$$

Listing 2.1. General case of Runge-Kutta algorithm

To determine the parameters c's, a's and W's in the above equation, $y_{i+1}$ defined in the scheme is expanded in terms of step length h and the resultant equation is then compared with Taylor series expansion of the solution of the differential equation up to a certain number of terms say p. Then the v-stage RK method will be of order p or is an $p^{th}$ order RK method. Here for any v>4 the maximum possible order p of the RK method is always less than v. However, for any v less than or equal to 4, it is possible derive an RK method of order p = v. The classical implementation of Runge-Kutta algorithm uses 3rd or 4th order which is given in Table 2.1.

Table 2.1.
Classical RK methods of order three and four

| | | |
|---|---|---|
| 1 | RK method of order three (v = 3) | $K_1 = h\,f(x_i, y_i)$<br>$K_2 = h\,f(x_i + h/2, y_i + K_1/2)$<br>$K_3 = h\,f(x_i + h, y_i - K_1 + 2K_2)$<br>$y_{i+1} = y_i + (K_1 + 4K_2 + K_3)/6$ |
| 2 | RK method of order three (v = 4) | $K_1 = h\,f(x_i, y_i)$<br>$K_2 = h\,f(x_i + h/2, y_i + K_1/2)$<br>$K_3 = h\,f(x_i + h/2, y_i + K_2/2)$<br>$K_4 = h\,f(x_i + h, y_i + K_3)$<br>$y_{i+1} = y_i + (K_1 + 2K_2 + 2K_3 + K_4)/6$ |

In this research work the abovementioned classical Runge-Kutta algorithm compared against modern method called Bulirsch–Stoer algorithm. Bulirsch–Stoer algorithm is a method for the numerical solution of ordinary differential equations

which combines three powerful ideas: Richardson extrapolation, the use of rational function extrapolation in Richardson-type applications, and the modified midpoint method, to obtain numerical solutions to ordinary differential equations (ODEs) with high accuracy and comparatively little computational effort.

The idea of Richardson extrapolation is to consider a numerical calculation whose accuracy depends on the used step size h as an (unknown) analytic function of the step size h, performing the numerical calculation with various values of h, fitting a (chosen) analytic function to the resulting points, and then evaluating the fitting function for h = 0, thus trying to approximate the result of the calculation with infinitely fine steps.

Bulirsch and Stoer recognized that using rational functions as fitting functions for Richardson extrapolation in numerical integration is superior to using polynomial functions because rational functions are able to approximate functions with poles rather well (compared to polynomial functions), given that there are enough higher-power terms in the denominator to account for nearby poles. While a polynomial interpolation or extrapolation only yields good results if the nearest pole is rather far outside a circle around the known data points in the complex plane, rational function interpolation or extrapolation can have remarkable accuracy even in the presence of nearby poles.

The modified midpoint method by itself is a second-order method, and therefore generally inferior to fourth-order methods like the fourth-order Runge–Kutta method. However, it has the advantage of requiring only one derivative evaluation per substep (asymptotically for a large number of substeps), and, in addition, as discovered by Gragg, the error of a modified midpoint step of size H, consisting of n substeps of size h = H/n each, and expressed as a power series in h, contains only even powers of h. This makes the modified midpoint method extremely useful to the Bulirsch–Stoer method as the accuracy increases two orders at a time when the results of separate attempts to cross the interval H with increasing numbers of substeps are combined.

Richardson extrapolation, presented in Fig. 2.1., used in the Bulirsch-Stoer method. In numerical analysis, Richardson extrapolation is a sequence acceleration method, used to improve the rate of convergence of a sequence. It is named after Lewis Fry Richardson, who introduced the technique in the early 20th century. Practical applications of Richardson extrapolation include Romberg integration, which applies Richardson extrapolation to the trapezoid rule, and the Bulirsch–Stoer algorithm for solving ordinary differential equations which is used in this research. A large interval H is spanned by different sequences of finer and finer substeps. Their results are extrapolated to an answer that is supposed to correspond to infinitely fine substeps. In the Bulirsch-Stoer method, the integrations are done by the modified midpoint method (given in Fig. 2.2.), and the extrapolation technique is rational function or polynomial extrapolation.

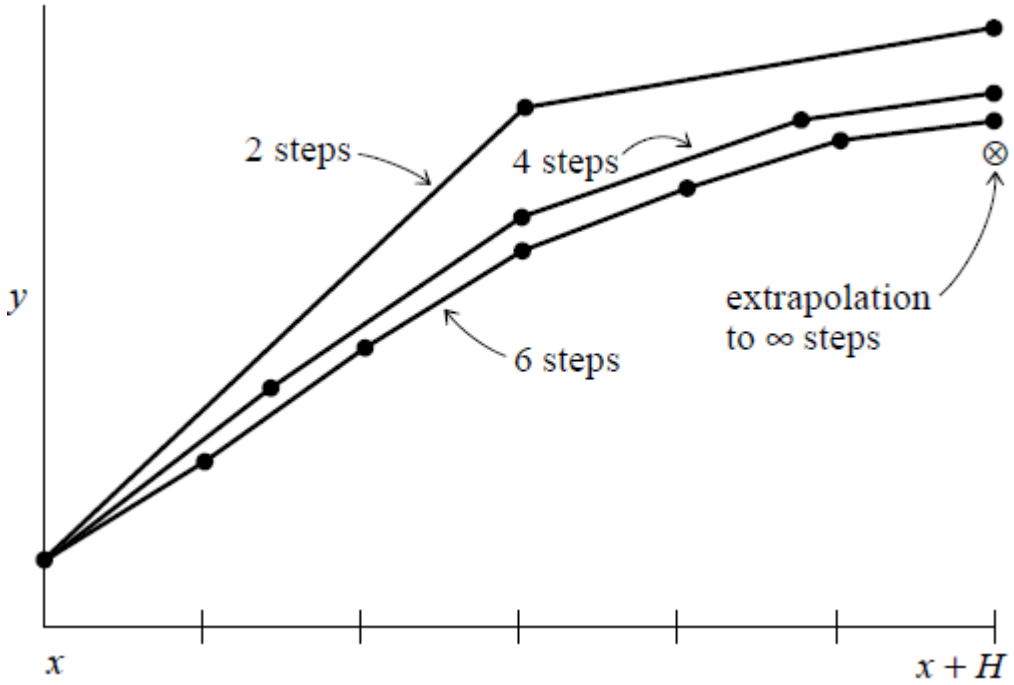


Figure 2.1. Richardson extrapolation visualization

$$\tilde{y}_0 = y(t_0)$$

**First Step: Euler**

$$\tilde{y}_1 = \tilde{y}(t_1) = \tilde{y}_0 + h f(\tilde{y}_0, t_0)$$

**Modified Midpoint**

$$\tilde{y}_2 = \tilde{y}(t_2) = \tilde{y}_0 + 2h f(\tilde{y}_1, t_1)$$

$$\tilde{y}_3 = \tilde{y}(t_3) = \tilde{y}_1 + 2h f(\tilde{y}_2, t_2)$$

i = 1... n-1

$$\tilde{y}_{i+1} = \tilde{y}(t_{i+1}) = \tilde{y}_{i-1} + 2h f(\tilde{y}_i, t_i)$$

**Combination**

$$y(t_0 + H) = \frac{1}{2}[\tilde{y}_n + \tilde{y}_{n-1} + h f(\tilde{y}_n, t_n)]$$

Figure 2.2. Modified midpoint method

In general case, different types of extrapolation can be applied in this method. The example shown below:

$$Y_j^{(k+1)} = Y_j^{(k)} + \frac{Y_j^{(k)} - Y_{j-1}^{(k)}}{\left(\frac{n_j}{n_{j-k}}\right)^2 - 1} \tag{2.41}$$

where n is the number of substeps.

A commonly used sequence of "n"s is: n = {2, 4, 6, 8, 10,... } $n_j$=2j. After each $n_j$, extrapolate and obtain error estimate. This technique (and extrapolation in general) works best for smooth functions. If not very smooth, today is commonly used adaptive RK, since it does a better job of negotiating abruptly changing regions of the domain.

Finally, error rate of the Bulirsch-Stoer's method is:

$$y(t_0+H)-y(t_0)=\sum k_i h^{2i} \tag{2.42}$$

By applying one of the mentioned numerical methods the Cauchy problem can be solved. Then solving the Cauchy problem for the ordinary differential equations we obtain $a_p(P,t)$ and then evaluate the integrals

$$\widehat{\psi}_{p,r}(\phi_k) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(\phi_k, t) dt,$$
$$\widehat{\psi}_{p,r}(A) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(A, t) dt,$$
$$\widehat{\psi}_{p,r}(f) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(f, t) dt,$$
$$\widehat{\psi}_{p,r} = \int_{t_{r-1}}^{t_r} \psi_p(t) dt,$$

Integrals were calculated by Simpson's method. In numerical analysis, Simpson's rule/method is a method for numerical integration, the numerical approximation of definite integrals. Specifically, it is the following approximation:

$$\int_a^b f(x)\,dx \approx \tfrac{b-a}{6}\left[f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right], \tag{2.43}$$

Simpson's rule also corresponds to the three-point Newton-Cotes quadrature rule. The method is credited to the mathematician Thomas Simpson (1710–1761) of Leicestershire, England. Kepler used similar formulas over 100 years prior. For this reason the method is sometimes called Kepler's rule, or Keplersche Fassregel in German.

In Simpson's Rule, we will use parabolas to approximate each part of the curve. This proves to be very efficient since it's generally more accurate than the other numerical methods we've seen. We divide the area into n equal segments of width Δx. The approximate area is given by the following.

**Simpson's Rule**

$$\text{Area} = \int_a^b f(x)\,dx \approx \frac{\Delta x}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 \ldots + 4y_{n-1} + y_n)$$

where $\Delta x = \dfrac{b-a}{n}$.

Figure 2.3. Simpson's rule in general case

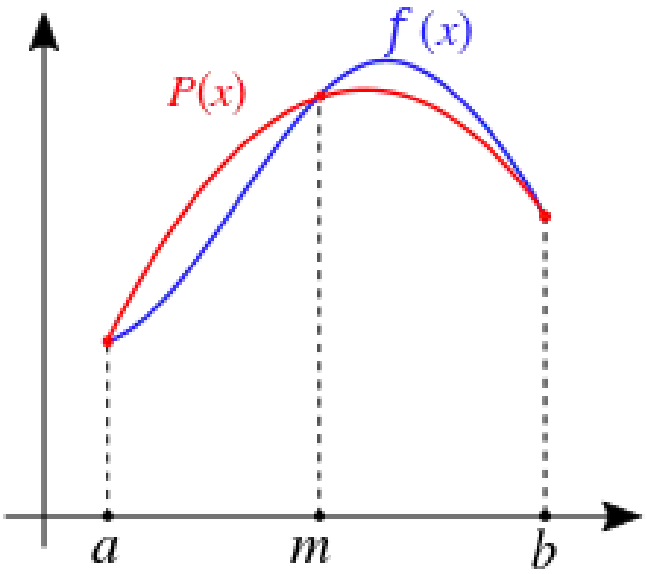


Figure 2.4. Simpson's rule can be derived by approximating the integrand f (x) (in blue) by the quadratic interpolant P(x) (in red)

The error in approximating an integral by Simpson's rule is

$$\frac{1}{90}\left(\frac{b-a}{2}\right)^5 \left|f^{(4)}(\xi)\right|,$$

Where ξ is some number between **a** and **b**. The error is asymptotically proportional to $(b-a)^5$. However, the above derivations suggest an error proportional to $(b-a)^4$. Simpson's rule gains an extra order because the points at which the integrand is evaluated are distributed symmetrically in the interval [a, b].

Since the error term is proportional to the fourth derivative of f at ξ , this shows that Simpson's rule provides exact results for any polynomial f of degree three or less, since the fourth derivative of such a polynomial is zero at all points.

If the interval of integration [a,b] is in some sense "small", then Simpson's rule will provide an adequate approximation to the exact integral. By small, what we really mean is that the function being integrated is relatively smooth over the interval [a,b]. For such a function, a smooth quadratic interpolant like the one used in Simpson's rule will give good results.

However, it is often the case that the function we are trying to integrate is not smooth over the interval. Typically, this means that either the function is highly oscillatory, or it lacks derivatives at certain points. In these cases, Simpson's rule may give very poor results. One common way of handling this problem is by breaking up the interval [a,b] into a number of small subintervals. Simpson's rule is then applied to each subinterval, with the results being summed to produce an approximation for the integral over the entire interval. This sort of approach is termed the composite Simpson's rule.

In practice, it is often advantageous to use subintervals of different lengths, and concentrate the efforts on the places where the integrand is less well-behaved. This leads to the adaptive Simpson's method.

From the equalities (3.3) it can be possible to calculate matrices G,V,g. Consequently we obtain the matrices Q, F and form the system of linear algebraic equations with respect to parameters λ. Solving again Cauchy problem with obtained λ we calculate u. The full algorithmic path with detailed explanation can be found in [2].

The all calculations and numerical implementations related to this research work were done in MATLAB language. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows to user to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation. It has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

```
[J,I] = meshgrid(1:n);
A = mod(I + J - (n + 3) / 2, n);
B = mod(I + 2 * J - 2, n);
M = n * A + B + 1;
```

Listing 2.2. Sample code in MATLAB environment

The MATLAB system consists of five main parts:

1. The MATLAB language. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.
2. The MATLAB working environment. This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.
3. Handle Graphics. This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.
4. The MATLAB mathematical function library. This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.
5. The MATLAB Application Program Interface (API). This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

MATLAB can solve differential equations with or without initial conditions by using built-in packages like *dsolve*, *ode45*, etc. The following ones are the most common types of differential equation that can be solved using MATLAB: First-Order Linear ODE with Initial Condition, Nonlinear Differential Equation with Initial Condition, Second-Order ODE with Initial Conditions, Third-Order ODE with Initial Conditions, etc.

Parallel computing was also applied in this research. Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously[41].

The origins of true (MIMD) parallelism go back to Luigi Federico Menabrea and his Sketch of the Analytic Engine Invented by Charles Babbage. In April 1958, S. Gill (Ferranti) discussed parallel programming and the need for branching and waiting. Also in 1958, IBM researchers John Cocke and Daniel Slotnick discussed the use of parallelism in numerical calculations for the first time. Burroughs Corporation introduced the D825 in 1962, a four-processor computer that accessed up to 16 memory modules through a crossbar switch. In 1967, Amdahl and Slotnick published a debate about the feasibility of parallel processing at American Federation

of Information Processing Societies Conference. It was during this debate that Amdahl's law was coined to define the limit of speed-up due to parallelism [41].

Parallel computers can be traced back to the 1970s. The motivation behind early SIMD computers was to amortize the gate delay of the processor's control unit over multiple instructions. In 1964, Slotnick had proposed building a massively parallel computer for the Lawrence Livermore National Laboratory. His design was funded by the US Air Force, which was the earliest SIMD parallel-computing effort, ILLIAC IV. The key to its design was a fairly high parallelism, with up to 256 processors, which allowed the machine to work on large datasets in what would later be known as vector processing. However, ILLIAC IV was called "the most infamous of supercomputers", because the project was only one fourth completed, but took 11 years and cost almost four times the original estimate. When it was finally ready to run its first real application in 1976, it was outperformed by existing commercial supercomputers such as the Cray-1 [41].

Large problems can often be divided into smaller ones, which can then be solved at the same time. There are several different forms of parallel computing: bit-level, instruction-level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors [41].

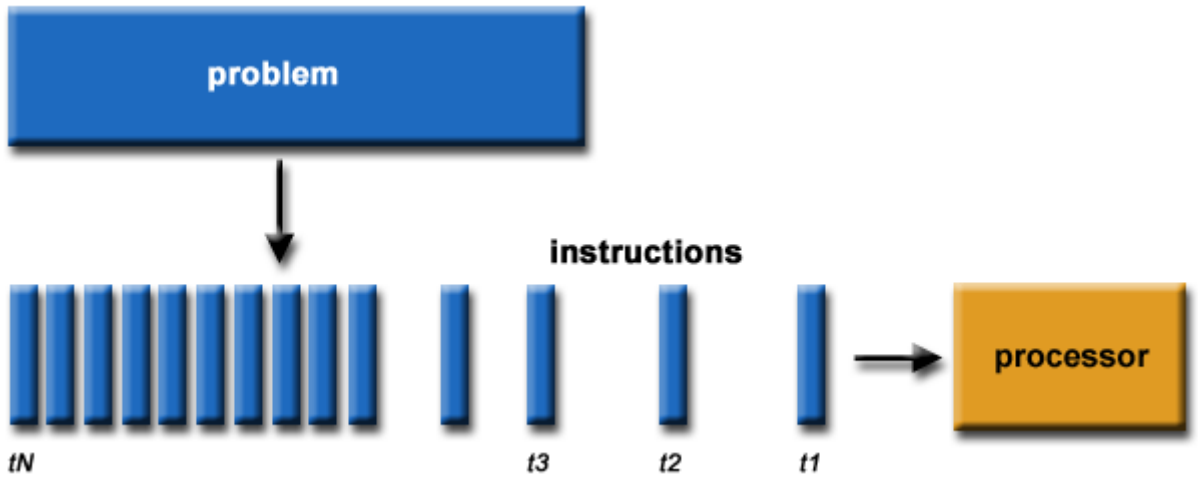


Figure 2.5. Parallel computing architecture

Parallel computing is closely related to concurrent computing—they are frequently used together, and often conflated, though the two are distinct: it is possible to have parallelism without concurrency (such as bit-level parallelism), and concurrency without parallelism (such as multitasking by time-sharing on a single-core CPU). In parallel computing, a computational task is typically broken down in several, often many, very similar subtasks that can be processed independently and whose results are combined afterwards, upon completion. In contrast, in concurrent computing, the various processes often do not address related tasks; when they do, as

is typical in distributed computing, the separate tasks may have a varied nature and often require some inter-process communication during execution [41].

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks [41].

In some cases parallelism is transparent to the programmer, such as in bit-level or instruction-level parallelism, but explicitly parallel algorithms, particularly those that use concurrency, are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance.

A theoretical upper bound on the speed-up of a single program as a result of parallelization is given by Amdahl's law.

Traditionally, computer software has been written for serial computation. To solve a problem, an algorithm is constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit on one computer. Only one instruction may execute at a time after that instruction is finished, the next one is executed [41].

Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Frequency scaling was the dominant reason for improvements in computer performance from the mid-1980s until 2004. The runtime of a program is equal to the number of instructions multiplied by the average time per instruction. Maintaining everything else constant, increasing the clock frequency decreases the average time it takes to execute an instruction. An increase in frequency thus decreases runtime for all compute-bound programs [41].

However, power consumption P by a chip is given by the equation $P = C \times V^2 \times F$, where C is the capacitance being switched per clock cycle (proportional to the number of transistors whose inputs change), V is voltage, and F is the processor frequency (cycles per second). Increases in frequency increase the amount of power used in a processor. Increasing processor power consumption led ultimately to Intel's May 8, 2004 cancellation of its Tejas and Jayhawk processors, which is generally cited as the end of frequency scaling as the dominant computer architecture paradigm.

Moore's law is the empirical observation that the number of transistors in a microprocessor doubles every 18 to 24 months. Despite power consumption issues,

and repeated predictions of its end, Moore's law is still in effect. With the end of frequency scaling, these additional transistors (which are no longer used for frequency scaling) can be used to add extra hardware for parallel computing [41].

Understanding data dependencies is fundamental in implementing parallel algorithms. No program can run more quickly than the longest chain of dependent calculations (known as the critical path), since calculations that depend upon prior calculations in the chain must be executed in order. However, most algorithms do not consist of just a long chain of dependent calculations; there are usually opportunities to execute independent calculations in parallel.

There are several types of parallelism[41]:

1. Bit-level parallelism is a form of parallel computing based on increasing processor word size. Increasing the word size reduces the number of instructions the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word. (For example, consider a case where an 8-bit processor must add two 16-bit integers. The processor must first add the 8 lower-order bits from each integer, then add the 8 higher-order bits, requiring two instructions to complete a single operation. A 16-bit processor would be able to complete the operation with single instruction.)
2. Instruction-level parallelism (ILP) is a measure of how many of the instructions in a computer program can be executed simultaneously. A computer program is, in essence, a stream of instructions executed by a processor. Without instruction-level parallelism, a processor can only issue less than one instruction per clock cycle (IPC < 1). These processors are known as subscalar processors. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s.
3. Task parallelisms is the characteristic of a parallel program that "entirely different calculations can be performed on either the same or different sets of data". This contrasts with data parallelism, where the same calculation is performed on the same or different sets of data. Task parallelism involves the decomposition of a task into sub-tasks and then allocating each sub-task to a processor for execution. The processors would then execute these sub-tasks simultaneously and often cooperatively. Task parallelism does not usually scale with the size of a problem.

Main memory in a parallel computer is either shared memory (shared between all processing elements in a single address space), or distributed memory (in which each processing element has its own local address space). Distributed memory refers to the fact that the memory is logically distributed, but often implies that it is physically distributed as well. Distributed shared memory and memory virtualization combine the two approaches, where the processing element has its own local memory

and access to the memory on non-local processors. Accesses to local memory are typically faster than accesses to non-local memory.

Computer architectures in which each element of main memory can be accessed with equal latency and bandwidth are known as uniform memory access (UMA) systems. Typically, that can be achieved only by a shared memory system, in which the memory is not physically distributed. A system that does not have this property is known as a non-uniform memory access (NUMA) architecture. Distributed memory systems have non-uniform memory access.

Computer systems make use of caches—small and fast memories located close to the processor which store temporary copies of memory values (nearby in both the physical and logical sense). Parallel computer systems have difficulties with caches that may store the same value in more than one location, with the possibility of incorrect program execution. These computers require a cache coherency system, which keeps track of cached values and strategically purges them, thus ensuring correct program execution. Bus snooping is one of the most common methods for keeping track of which values are being accessed (and thus should be purged). Designing large, high-performance cache coherence systems is a very difficult problem in computer architecture. As a result, shared memory computer architectures do not scale as well as distributed memory systems do.

A multi-core processor is a processor that includes multiple processing units (called "cores") on the same chip. This processor differs from a superscalar processor, which includes multiple execution units and can issue multiple instructions per clock cycle from one instruction stream (thread); in contrast, a multi-core processor can issue multiple instructions per clock cycle from multiple instruction streams.

Simultaneous multithreading (of which Intel's Hyper-Threading is the best known) was an early form of pseudo-multi-coreism. A processor capable of simultaneous multithreading includes multiple execution units in the same processing unit—that is it has a superscalar architecture—and can issue multiple instructions per clock cycle from multiple threads. Temporal multithreading on the other hand includes a single execution unit in the same processing unit and can issue one instruction at a time from multiple threads.

A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more difficult if they are not. The most common type of cluster is the Beowulf cluster, which is a cluster implemented on multiple identical commercial off-the-shelf computers connected with a TCP/IP Ethernet local area network. Beowulf technology was originally developed by Thomas Sterling and Donald Becker. The vast majority of the TOP500 supercomputers are clusters[41].

Because grid computing systems (described below) can easily handle embarrassingly parallel problems, modern clusters are typically designed to handle more difficult problems-problems that require nodes to share intermediate results with each other more often. This requires a high bandwidth and, more importantly, a

low-latency interconnection network. Many historic and current supercomputers use customized high-performance network hardware specifically designed for cluster computing, such as the Cray Gemini network. As of 2014, most current supercomputers use some off-the-shelf standard network hardware, often Myrinet, InfiniBand, or Gigabit Ethernet.

Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the Internet to work on a given problem. Because of the low bandwidth and extremely high latency available on the Internet, distributed computing typically deals only with embarrassingly parallel problems. Many distributed computing applications have been created, of which SETI@home and Folding@home are the best-known examples.

Most grid computing applications use middleware (software that sits between the operating system and the application to manage network resources and standardize the software interface). The most common distributed computing middleware is the Berkeley Open Infrastructure for Network Computing (BOINC). Often, distributed computing software makes use of "spare cycles", performing computations at times when a computer is idling.

General-purpose computing on graphics processing units (GPGPU) is a fairly recent trend in computer engineering research. GPUs are co-processors that have been heavily optimized for computer graphics processing. Computer graphics processing is a field dominated by data parallel operations—particularly linear algebra matrix operations.

In the early days, GPGPU programs used the normal graphics APIs for executing programs. However, several new programming languages and platforms have been built to do general purpose computation on GPUs with both Nvidia and AMD releasing programming environments with CUDA and Stream SDK respectively. Other GPU programming languages include BrookGPU, PeakStream, and RapidMind. Nvidia has also released specific products for computation in their Tesla series. The technology consortium Khronos Group has released the OpenCL specification, which is a framework for writing programs that execute across platforms consisting of CPUs and GPUs. AMD, Apple, Intel, Nvidia and others are supporting OpenCL.

Concurrent programming languages, libraries, APIs, and parallel programming models (such as algorithmic skeletons) have been created for programming parallel computers. These can generally be divided into classes based on the assumptions they make about the underlying memory architecture—shared memory, distributed memory, or shared distributed memory. Shared memory programming languages communicate by manipulating shared memory variables. Distributed memory uses message passing. POSIX Threads and OpenMP are two of the most widely used shared memory APIs, whereas Message Passing Interface (MPI) is the most widely used message-passing system API. One concept used in programming parallel programs is the future concept, where one part of a program promises to deliver a required datum to another part of a program at some future time.

CAPS entreprise and Pathscale are also coordinating their effort to make hybrid multi-core parallel programming (HMPP) directives an open standard called OpenHMPP. The OpenHMPP directive-based programming model offers a syntax to efficiently offload computations on hardware accelerators and to optimize data movement to/from the hardware memory. OpenHMPP directives describe remote procedure call (RPC) on an accelerator device (e.g. GPU) or more generally a set of cores. The directives annotate C or Fortran codes to describe two sets of functionalities: the offloading of procedures (denoted codelets) onto a remote device and the optimization of data transfers between the CPU main memory and the accelerator memory.

As parallel computers become larger and faster, it becomes feasible to solve problems that previously took too long to run. Parallel computing is used in a wide range of fields, from bioinformatics (protein folding and sequence analysis) to economics (mathematical finance). Common types of problems found in parallel computing applications are:

- dense linear algebra;
- sparse linear algebra;
- spectral methods (such as Cooley–Tukey fast Fourier transform) N-body problems (such as Barnes–Hut simulation);
- structured grid problems (such as Lattice Boltzmann methods);
- unstructured grid problems (such as found in finite element analysis);
- Monte Carlo method;
- combinational logic (such as brute-force cryptographic techniques);
- graph traversal (such as sorting algorithms);
- dynamic programming;
- branch and bound methods;
- graphical models (such as detecting hidden Markov models and constructing Bayesian networks);
- finite-state machine simulation.

Parallel computing can also be applied to the design of fault-tolerant computer systems, particularly via lockstep systems performing the same operation in parallel. This provides redundancy in case one component should fail, and also allows automatic error detection and error correction if the results differ. These methods can be used to help prevent single event upsets caused by transient errors.[48] Although additional measures may be required in embedded or specialized systems, this method can provide a cost effective approach to achieve n-modular redundancy in commercial off-the-shelf systems.

MATLAB also has built in Parallel Computing Toolbox which lets to solve computationally and data-intensive problems using multicore processors, GPUs, and computer clusters. High-level constructs—parallel for-loops, special array types, and parallelized numerical algorithms—allow parallelize MATLAB applications without CUDA or MPI programming. It enables using the toolbox with Simulink to run multiple simulations of a model in parallel.

This toolbox lets to use the full processing power of multicore desktops by executing applications on workers (MATLAB computational engines) that run locally. Without changing the code, user can run the same applications on a computer cluster or a grid computing service (using MATLAB Distributed Computing Server). User can run parallel applications interactively or in batch.

```
tic
ticBytes(gcp);
n = 200;
A = 500;
a = zeros(n);
parfor i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
tocBytes(gcp)
toc
```

Listing 2.3. Code execution in parallel mode using Matlab toolbox

## 3 Numerical Experiments

Now we consider on [0, T] the linear boundary value problem for Fredholm integro-differential equation (2.1) and (2.2) with degenerate kernel, where

$$T = 1, m = 2, A(t) = \begin{pmatrix} 0 & t \\ t^2 & 0 \end{pmatrix}, B = -C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, d = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$f(t) = \begin{pmatrix} 2\pi cos(2\pi t) - tcos(2\pi t) + (t^4 exp(t/4))/(4\pi) \\ -2\pi sin(2\pi t) - t^2 sin(2\pi t) \end{pmatrix}$$

$$\phi_1(t) = \begin{pmatrix} t^3 exp(t/4) & t \\ t & 2t^3 exp(t/4) \end{pmatrix}, \phi_2(t) = \begin{pmatrix} \frac{t^4}{2} exp(t/4) & 0 \\ 0 & t^4 exp(t/4) \end{pmatrix}$$

$$\psi_1(t) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \psi_2(t) = \begin{pmatrix} t - \frac{1}{2} & 0 \\ 0 & t - \frac{1}{2} \end{pmatrix}, n = 2$$

Listing 3.1. Initial data

Here the matrix of differential part is variable and the construction of fundamental matrix breaks down. We use the numerical implementation of algorithm. For the integro-differential equation in the given problem any partition of interval [0, T] , including $\Delta_1$ , is regular. Accuracy of solution depends on the accuracy of solving the Cauchy problem on subintervals and evaluating of definite integrals. Detailed algorithm which consists of seven steps considered in Chapter 2.2:

I. Suppose we have a partition $\Delta_N : t_0 = 0 < t_1 < \ldots < t_{N-1} < t_N = T$ . Divide each r –th interval $[t_{r-1}, t_r]$, r = 1…N to $N_r$ parts with step $h_r = (t_r - t_{r-1})/N_r$.

II. Using the Bulirsch-Stoer method, we find the numerical solutions to Cauchy problems (2.33) and define the values of (n × n) matrices $a_r(\varphi_k,t)$ on the set $\{t_{r-1}, t_r\}$ , r = 1,N , k = 1, m. Below presented numerical solution to Cauchy problem:

```
% a(fi_1(), t)-1
[a1_fi1_1, ~]=BulirschStoer(@F3_1,t0,[0;0],tol);
a1_fi1_1 = a1_fi1_1';
[a2_fi1_1, ~]=BulirschStoer(@F3_1,t1,[0;0],tol);
a2_fi1_1 = a2_fi1_1';
% a(fi_1(), t)-2
[a1_fi1_2, ~]=BulirschStoer(@F4_1,t0,[0;0],tol);
a1_fi1_2 = a1_fi1_2';
[a2_fi1_2, ~]=BulirschStoer(@F4_1,t1,[0;0],tol);
a2_fi1_2 = a2_fi1_2';
% a(fi_2(), t)-1
[a1_fi2_1, ~]=BulirschStoer(@F3_2,t0,[0;0],tol);
a1_fi2_1 = a1_fi2_1';
[a2_fi2_1, ~]=BulirschStoer(@F3_2,t1,[0;0],tol);
a2_fi2_1 = a2_fi2_1';
% a(fi_2(), t)-2
[a1_fi2_2, ~]=BulirschStoer(@F4_2,t0,[0;0],tol);
a1_fi2_2 = a1_fi2_2';
```

```
[a2_fi2_2, ~]=BulirschStoer(@F4_2,t1,[0;0],tol);
a2_fi2_2 = a2_fi2_2';
```

Listing 3.2. Numerical solution of Cauchy problem for $a_r(\varphi_k,t)$

III. Using the values of (n × n) matrices $\psi_k(s)$ and $a_r(\varphi_k,t)$ on $\{t_{r-1}, t_r\}$ , and Simpson's method, we calculate the (n × n) matrices

$$\widehat{\psi}_{p,r}(\phi_k) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(\phi_k, t) dt,$$

And its implementation in MATLAB:

```
psi1_a1_fi1 = [simps(t0,a1_fi1_1(:,1)) simps(t0,a1_fi1_2(:,1));
               simps(t0,a1_fi1_1(:,2)) simps(t0,a1_fi1_2(:,2))];
psi1_a1_fi2 = [simps(t0,a1_fi2_1(:,1)) simps(t0,a1_fi2_2(:,1));
               simps(t0,a1_fi2_1(:,2)) simps(t0,a1_fi2_2(:,2))];
psi1_a2_fi1 = [simps(t1,a2_fi1_1(:,1)) simps(t1,a2_fi1_2(:,1));
               simps(t1,a2_fi1_1(:,2)) simps(t1,a2_fi1_2(:,2))];
psi1_a2_fi2 = [simps(t1,a2_fi2_1(:,1)) simps(t1,a2_fi2_2(:,1));
               simps(t1,a2_fi2_1(:,2)) simps(t1,a2_fi2_2(:,2))];

psi2_a1_fi1 = [simps(t0,(x0 - 1/2).*a1_fi1_1(:,1)') simps(t0,(x0 -
1/2).*a1_fi1_2(:,1)');
               simps(t0,(x0 - 1/2).*a1_fi1_1(:,2)') simps(t0,(x0 -
1/2).*a1_fi1_2(:,2)')];
psi2_a1_fi2 = [simps(t0,(x0 - 1/2).*a1_fi2_1(:,1)') simps(t0,(x0 -
1/2).*a1_fi2_2(:,1)');
               simps(t0,(x0 - 1/2).*a1_fi2_1(:,2)') simps(t0,(x0 -
1/2).*a1_fi2_2(:,2)')];
psi2_a2_fi1 = [simps(t1,(x1 - 1/2).*a2_fi1_1(:,1)') simps(t1,(x1 -
1/2).*a2_fi1_2(:,1)');
               simps(t1,(x1 - 1/2).*a2_fi1_1(:,2)') simps(t1,(x1 -
1/2).*a2_fi1_2(:,2)')];
psi2_a2_fi2 = [simps(t1,(x1 - 1/2).*a2_fi2_1(:,1)') simps(t1,(x1 -
1/2).*a2_fi2_2(:,1)');
               simps(t1,(x1 - 1/2).*a2_fi2_1(:,2)') simps(t1,(x1 -
1/2).*a2_fi2_2(:,2)')];

g_f = [psi1_a1f + psi1_a2f; psi2_a1f + psi2_a2f];
```

Listing 3.3. Numeric implementation of Simpson's method in MATLAB

Summing up over r, on the base of equalities (2.10), (2.32), we obtain the (n × n) matrices

$$G_{p,k}(\Delta_N) = \sum_{r=1}^{N} \widehat{\psi}_{p,r}(\varphi_k), \quad p,k = \overline{1,m}.$$

Compose the (nm × nm) matrix $G(\Delta_N) = (G_{p,k}(\Delta_N))$, p, k = 1…m and check the invertibility of matrix $[I - G(\Delta_N)] : Rnm \rightarrow Rnm$.

If this matrix is invertible, then we find its inverse and represent it in the form $[I - G(\Delta_N)]^{-1} = (M_{p,k}(\Delta_N))$, where $M_{p,k}(\Delta_N))$ is (n × n) matrix, p, k = 1…N . Then move on to the next step of algorithm.

If this matrix has no inverse, i.e. the partition $\Delta_N$ is not regular, then we take a new partition of interval [0, T] , and the algorithm starts over. The simplest way for selecting of a new partition is to choose the partition $\Delta_{2N}$, where each interval of partition $\Delta_N$ is divided into two parts.

IV. Solving again the Cauchy problem using the Bulirsch-Stoer method, for the ordinary differential equations

$$\frac{dx}{dt} = A(t)x + A(t), \quad x(t_{r-1}) = 0, \quad t \in [t_{r-1}, t_r],$$

$$\frac{dx}{dt} = A(t)x + f(t), \quad x(t_{r-1}) = 0, \quad t \in [t_{r-1}, t_r], \quad r = \overline{1, N}$$

we find $a_r(A, t)$ and $a_r(f, t)$, r = 1…N. Below the numeric implementation of the given Cauchy problem:

```
tol = 1e-12;
% a(A(), t)-1
[a1_A1, ~]=BulirschStoer(@F1,t0,[0;0],tol);
a1_A1 = a1_A1';
[a2_A1, ~]=BulirschStoer(@F1,t1,[0;0],tol);
a2_A1 = a2_A1';
% a(A(), t)-2
[a1_A2, ~]=BulirschStoer(@F2,t0,[0;0],tol);
a1_A2 = a1_A2';
[a2_A2, ~]=BulirschStoer(@F2,t1,[0;0],tol);
a2_A2 = a2_A2';
% a(f(), t)
[a1_F, ~]=BulirschStoer(@F5,t0,[0;0],tol);
a1_F = a1_F';
[a2_F, ~]=BulirschStoer(@F5,t1,[0;0],tol);
a2_F = a2_F';
```

Listing 3.4. Numerical solution of Cauchy problem for $a_r(A,t)$ and $a_r(f,t)$

V. The following matrices can be evaluated by applying the Simpson's method

$$\widehat{\psi}_{p,r}(A) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(A, t) dt,$$

$$\widehat{\psi}_{p,r}(f) = \int_{t_{r-1}}^{t_r} \psi_p(t) a_r(f, t) dt,$$

$$\widehat{\psi}_{p,r} = \int_{t_{r-1}}^{t_r} \psi_p(t)dt,$$

And its implementation in MATLAB:

```
psi1_a1A = [simps(t0,a1_A1(:,1)) simps(t0,a1_A2(:,1));
            simps(t0,a1_A1(:,2)) simps(t0,a1_A2(:,2))];
psi1_a2A = [simps(t0,a2_A1(:,1)) simps(t0,a2_A2(:,1));
            simps(t0,a2_A1(:,2)) simps(t0,a2_A2(:,2))];

psi2_a1A = [simps(t0,(x0 - 1/2).*a1_A1(:,1)') simps(t0,(x0 -
1/2).*a1_A2(:,1)');
            simps(t0,(x0 - 1/2).*a1_A1(:,2)') simps(t0,(x0 -
1/2).*a1_A2(:,2)')];
psi2_a2A = [simps(t0,(x1 - 1/2).*a2_A1(:,1)') simps(t0,(x1 -
1/2).*a2_A2(:,1)');
            simps(t0,(x1 - 1/2).*a2_A1(:,2)') simps(t0,(x1 -
1/2).*a2_A2(:,2)')];

G = [psi1_a1_fi1 + psi1_a2_fi1 psi1_a1_fi2 + psi1_a2_fi2;
     psi2_a1_fi1 + psi2_a2_fi1 psi2_a1_fi2 + psi2_a2_fi2];
```

Listing 3.5. Numeric implementation of step V in MATLAB

From the equalities (2.11), (2.12) and (2.32) we define the (n × n) matrices

$$V_{p,r}(\Delta_n) = \widehat{\psi}_{p,r}(A) + \sum_{j=1}^{N}\sum_{k=1}^{m} \widehat{\psi}_{p,r}(\phi_k) \cdot \widehat{\psi}_{p,r}$$

and n vectors

$$g_p(f, \Delta_n) = \sum_{j=1}^{N} \widehat{\psi}_{p,r}(\Delta_N), p = 1...m, r = 1...N.$$

VI. Form the system of linear algebraic equations with respect to parameters

$$Q_*(\Delta_N)\lambda = -F_*(\Delta_N), \lambda \in R^{nN}$$

And its implementation in MATLAB:

```
Q_11 = B + C * (a2_fi1_N*L_11 + a2_fi2_N*L_21);
Q_12 = C + C * (a2A_N + a2_fi1_N*L_12 + a2_fi2_N*L_22);
Q_21 = I + I * (a1A_N + a1_fi1_N*L_11 + a1_fi2_N*L_21);
Q_22 = I * (a1_fi1_N*L_12 + a1_fi2_N*L_22) - I;

d_1 = d - C * (a2_fi1_N*F_1 + a2_fi2_N*F_2 + a2f_N);
d_2 = -(a1_fi1_N*F_1 + a1_fi2_N*F_2 + a1f_N);
```

```
Q = [Q_11 Q_12; Q_21 Q_22];
D = [d_1;d_2];

lambda = Q\D;
```

Listing 3.6. Numeric implementation of step VI in MATLAB

The elements of matrix $Q^*(\Delta_N) : R^{nN} \to R^{nN}$ and vector $F^*(\Delta_N) = (-d + CF_N(\Delta_N), F_1(\Delta_N, \ldots, F_{N-1}(\Delta_N)) \in R^{nN}$ are defined by the equalities (2.16), (2.17), (2.18). Solving the system (3.6), we find $\lambda = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_N^*) \in R^{nN}$.

VII. By the equalities

$$\mu_k^* = \sum_{j=1}^{N} \Big( \sum_{p=1}^{m} M_{k,p}(\Delta_N) V_{p,r}(\Delta_N) \Big) \lambda_j^* + \sum_{p=1}^{m} M_{k,p}(\Delta_N) g_p(f, \Delta_N)$$

we find the components of $\mu^* = (\mu_1^*, \mu_2^*, \ldots, \mu_m^*) \in R^{nm}$ and construct the function

$$\mathcal{F}^*(t) = \sum_{k=1}^{m} \varphi_k(t) \Big[ \mu_k^* + \sum_{r=1}^{N} \int_{t_{r-1}}^{t_r} \psi_k(s) ds \lambda_r^* \Big] + f(t).$$

Recall that $\lambda_r^* = x^*(t_{r-1})$, where $x^*(t)$ is a solution to the boundary value problem (2.1), ( 2.2). Therefore, solving system (2.37), we find the values of solution at the left-end points of subintervals.

The values of function $x^*(t)$ at the other points of subinterval $[t_{r-1}, t_r)$ are determined by solving the following Cauchy problem for ordinary differential equation

$$\frac{dx}{dt} = A(t)x + \mathcal{F}^*(t), \quad x(t_{r-1}) = \lambda_r^*, \quad t \in [t_{r-1}, t_r), \quad r = \overline{1, N}$$

As it was mentioned, numerical solution for Cauchy problem and evaluation of definite integrals are implemented using Bulirsch-Stoer method and Simpson's method, respectively. We provide the results of the numerical implementation of algorithm by partitioning the interval [0, 1] with step h = 0.5 and partitioning the subintervals [0, 0.5] and [0.5, 1] with step h1 = h2 = 0.05. In the table below $x_1(t)$ and $x_2(t)$ are numerical solutions of the integro-differential equation.

Table 3.1.
Numerical and exact solutions

| t | $x(t)_1$ | $x(t)_1^*$ | $x(t)_2$ | $x(t)_2^*$ |
|---|---|---|---|---|
| 0 | -0.00000355 | 0.00000000 | 1.00000070 | 1.00000000 |
| 0,05 | 0.30901344 | 0.30901699 | 0.95105724 | 0.95105652 |

continue of Table 3.1.

| | | | | |
|---|---|---|---|---|
| 0,1 | 0.58778171 | 0.58778525 | 0.80901774 | 0.80901699 |
| 0,15 | 0.80901346 | 0.80901699 | 0.58778601 | 0.58778525 |
| 0,2 | 0.95105300 | 0.95105652 | 0.30901777 | 0.30901699 |
| 0,25 | 0.99999651 | 1.00000000 | 0.00000079 | 0.00000000 |
| 0,3 | 0.95105305 | 0.95105652 | -0.30901620 | -0.30901699 |
| 0,35 | 0.80901356 | 0.80901699 | -0.58778445 | -0.58778525 |
| 0,4 | 0.58778185 | 0.58778525 | -0.80901619 | -0.80901699 |
| 0,45 | 0.30901363 | 0.30901699 | -0.95105571 | -0.95105652 |
| 0,5 | -0.00000333 | 0.00000000 | -0.99999920 | -1.00000000 |
| 0,55 | -0.30902029 | -0.30901699 | -0.95105572 | -0.95105652 |
| 0,6 | -0.58778851 | -0.58778525 | -0.80901621 | -0.80901699 |
| 0,65 | -0.80902023 | -0.80901699 | -0.58778448 | -0.58778525 |
| 0,7 | -0.95105973 | -0.95105652 | -0.30901623 | -0.30901699 |
| 0,75 | -1.00000321 | -1.00000000 | 0.00000075 | -0.00000000 |
| 0,8 | -0.95105973 | -0.95105652 | 0.30901774 | 0.30901699 |
| 0,85 | -0.80902025 | -0.80901699 | 0.58778598 | 0.58778525 |
| 0,9 | -0.58778857 | -0.58778525 | 0.80901772 | 0.80901699 |
| 0,95 | -0.30902041 | -0.30901699 | 0.95105723 | 0.95105652 |
| 1 | -0.00000355 | -0.00000000 | 1.00000070 | 1.00000000 |

Exact solution to the given problem is

$$x^*(t) = \begin{pmatrix} sin(2\pi t) \\ cos(2\pi t) \end{pmatrix}$$

and the following estimate is true

$$\max_{j=1\ldots20} ||x(t_j) - x^*(t_j)|| < 0.000003552$$

whereas the results obtained from the Runge-Kutta 4$^{th}$ order method is:

$$\max_{j=1\ldots20} ||x(t_j) - x^*(t_j)|| < 0.000003662$$

Furthermore, Bulirsch-Stoer method showed better performance in terms of computation time: spent_time (Bulirsch-Stoer) = 3.1511s versus spent_time (Runge-Kutta 4) = 3.9654s, which is 25.8% faster.

Moreover, parallel execution of the given code in MATLAB environment gave further computational improvement: 3.1511s versus 2.8546s, which is faster up to 11% than initial code without parallelizing. Listing 3.7 shows parallel execution part of the program.

```
x_final = zeros(length(u_1), 2);
parfor i=1:length(u_1)
    if i <= N1
```

```
        x_final(i,:) = [u_1(i,1) u_2(i,1)] + [lambda_1(1)
lambda_2(1)];
    else
        x_final(i,:) = [u_1(i,1) u_2(i,1)] + [lambda_1(2)
lambda_2(2)];
    end
end
```

Listing 3.7. Running computations in parallel mode.

# CONCLUSION

The interest to the integro-differential equations have risen extremely in the last several decades. It can be clearly seen from the number of research done in this direction. Moreover, the vast majority of scientific societies have begun to apply such equations to solve real-world problems in the following fields: financial mathematics, fluid mechanics, computer vision, etc. Therefore the study of integro-differential equation and their solution is important nowadays.

Beginning from the XIX century plenty of great research works were done related to this field. Such methods like Nekrasov's method and Green's function method are known to be the classical method for obtaining analytical solution. Despite the fact that classical analytical solutions handle most of the cases, there are still problems in the cases when an integral equation is not solvable. Therefore it is important to study numerical solution of the integro-differential equations. Moreover, most of the modern methods do not provide necessary and sufficient conditions for the well-posedness of a problem. In the case of parametrization and interval partition method which was used in the given work, necessary and sufficient conditions were provided.

In this research work, the numerical solution to the Fredholm integro-differential equations using Bulirsch-Stoer method showed a high accuracy and noticeably high computational efficiency compared with Runge-Kutta $4^{th}$ order method. Particularly, Bulirsch–Stoer algorithm was used for the numerical solution of ordinary differential equations. This algorithm combines three powerful ideas: Richardson extrapolation, the use of rational function extrapolation in Richardson-type applications, and the modified midpoint method in order to obtain numerical solutions to ordinary differential equations (ODEs) with high accuracy and comparatively little computational effort. Obviously, accuracy of approximate solution can be improved with a choice of closer approximating kernel as well as with increasing the step's number of iterative process.

Furthermore, there was applied parallel computation for the numerical solutions for Fredholm integro-differential equations. It is extremely useful in cases when we use higher order matrices, because arithmetic operations over the matrices and vectors can be easily distributed in parallel mode.

Abovementioned claims were proved in experimental part by numerical computation. There was considered the example where the matrix of differential part is variable and the construction of fundamental matrix breaks down. Therefore the obtaining analytical solution becomes difficult. In this situation, using the numerical implementation would be the most appropriate approach. Bulirsch-Stoer method showed better performance in terms of computation time: running time of Bulirsch-Stoer's method is equal to 3.1511 seconds, whereas the running time of Runge-Kutta $4^{th}$ order method is equal to 3.9654 seconds, which is faster by 25.8%. Also Bulirsch-Stoer's method showed better accuracy by approximately 11%.

In general, this research demonstrates that numerical solution shows high level of accuracy and that variety of numerical methods can be applied to solve integro-

differential equations. Moreover, numerical solution can be computationally efficient. Obviously, all of these observations open new opportunities for studying and solving real-world problems using integro-differential equations in a flexible, efficient and simple way.

## REFERENCES


[1] M. Bocher, Integral Equations, Cambridge University Press, London, (1974).
[2] D.S. Dzhumabaev, On one approach to solve the linear boundary value problems for Fredholm integro-differential equations, Journal of Computational and Applied Mathematics (2015), http://dx.doi.org/10.1016/j.cam.2015.08.023
[3] D.S. Dzhumabayev, Criteria for the unique solvability of a linear boundary-value problem for an ordinary differential equation. USSR Comput. Maths. Math. Phys. 29 (1989) 34-46.
[4] D.S. Dzhumabaev, A Method for Solving the Linear Boundary Value Problem for an Integro-Differential Equation. Comput. Math. Math. Phys. 50 (2010) 1150-1161.
[5] R. Merton, Option pricing when the underlying stock returns are discontinuous, J. Finan. Econ. 5 (1976), 125-144.
[6] S. Z. Levendorski, Pricing of the American put under Levy processes, Int. J. Theor. Appl. Finance 7 (2004), 303-335.
[7] H. Pham, Optimal stopping, free boundary, and American option in a jump-diffusion model, Appl. Math. Optim. 35 (1997), 145-164.
[8] L. Caffarelli, A. Figalli, Regularity of solutions to the parabolic fractional obstacle problem, J. Reine Angew. Math., 680 (2013), 191-233.
[9] L. Silvestre, Regularity of the obstacle problem for a fractional power of the Laplace operator, Comm. Pure Appl. Math. 60 (2007), 67-112.
[10] L. Caffarelli, S. Salsa, L. Silvestre, Regularity estimates for the solution and the free boundary of the obstacle problem for the fractional Laplacian, Invent. Math. 171 (2008), 425-461.
[11] R. Cont, P. Tankov, Financial Modelling With Jump Processes, Financial Mathematics Series. Chapman & Hall/CRC, Boca Raton, FL, 2004.
[12] W. Schoutens, Levy Processes in Finance: Pricing Financial Derivatives, Wiley, New York 2003.
[13] J. P. Nolan, Fitting Data and Assessing Goodness-of-fit with stable distributions, Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics, Washington DC, (1999).
[14] N. E. Humphries et al., Environmental context explains Levy and Brownian movement patterns of marine predators, Nature, 465 (2010), 1066-1069.
[15] R. Mancinelli, D. Vergni, A. Vulpiani, Front propagation in reactive systems with anomalous diffusion, Phys. D 185 (2003), 175-195.
[16] P. Constantin, Euler equations, Navier-Stokes equations and turbulence, In Mathematical foundation of turbulent viscous flows, Lecture Notes in Math., pages 1-43. Springer, Berlin, 2006.
[17] L. Caffarelli, A. Vasseur, Drift diffusion equations with fractional diffusion and the quasi-geostrophic equation, Ann. of Math. 171 (2010), 1903-1930.
[18] L. Caffarelli, A. Vasseur, The De Giorgi method for regularity of solutions of elliptic equations and its applications to fluid dynamics, Discrete Contin. Dyn. Syst. Ser. S 3 (2010), 409-427.

[19] C. J. Amick, J. F. Toland, Uniqueness and related analytic properties for the Benjamin-Ono equation, Acta Math., 167 (1991), 107-126.
[20] S. Dipierro, G. Palatucci, E. Valdinoci, Dislocation dynamics in crystals: A macroscopic theory in fractional Laplace setting, Comm. Math. Phys., to appear.
[21] A. C. Eringen, Linear theory of nonlocal elasticity and dispersion of plane waves, Int. J. Engng Sci 10 (1972), 425-435.
[22] W. J. Drugan, Two Exact Micromechanics-Based Nonlocal Constitutive Equations for Random Linear Elastic Composite Materials, Journal of the Mechanics and Physics of Solids 51 (2003), 1745-1772.
[23] S. Dipierro, G. Palatucci, E. Valdinoci, Dislocation dynamics in crystals: A macroscopic theory in fractional Laplace setting, Comm. Math. Phys., to appear.
[24] S. J. Chapman, J. Rubinstein, M. Schatzman, A mean-field model for superconducting vortices, Eur. J. Appl. Math. 7 (1996), 97-111.
[25] L. P. Yaroslavsky, Digital Picture Processing, an Introduction, Springer-Verlag, Berlin, 1985.
[26] C. R. Graham, M. Zworski, Scattering matrix in conformal geometry, Invent. Math. 152 (2003), 89-118.
[27] R. Cont, E. Voltchkova, Integro-differential equations for option prices in exponential L'evy models.
[28] Biography of Erik Ivar Fredholm (Online resorouce): http://www-groups.dcs.st-and.ac.uk/history/Biographies/Fredholm.html
[29] Wazwaz AMed. Linear and Nonlinear Integral Equations. Springer: Heidelberg, 2011.
[30] H.T. Davis, Introduction to Nonlinear Differential and Integral Equations, Dover, New York, (1962).
[31] A. I. Nekrasov, On a Class of Linear Integro- Differential Equations, Tr. Tsentr. Aerogidrodin. Inst. 190 (1934) 1-25.
[32] L. E. Krivoshein, Approximate Methods for Solving Linear Ordinary Integro-Differential Equations. Akad. Nauk Kirg. SSR, Frunze, 1962. [in Russian]
[33] A.A. Boichuk, A.M. Samoilenko, Generalized inverse operators and Fredholm boundary-value problems, VSP, Utrecht, Boston, 2004.
[34] D.S. Dzhumabaev, An Algorithm for Solving a Linear Two-Point Boundary Value Problem for an Integro-Differential Equation. Comput. Math. Math. Phys. 53 (2013) 736-758.
[35] D.S. Dzhumabaev, E.A. Bakirova, Criteria for the Unique Solvability of a Linear Two-Point Boundary Value Problem for Systems of Integro-Differential Equations. Differential Equations, 49 (2013) 914-937.
[36] J. Pruss, Evolutionary Integral equations and Applications, Basel etc. Birkhauser Verlag. 1993.
[37] M.Turkyilmazoglu, An effective approach for numerical solutions of high- order Fredholm integro-differential equations. Appl. Math. Comput. 227 (2014) 384-398.
[38] S.Yuzbasi, N.Sahin, M.Sezer, Numerical solutions of systems of linear Fredholm integrodifferential equations with Bessel polynomial bases. Comp. Math. Appl. 61 (2011) 3079-3096.

[39] W. Yulan, T. Chaolu, P. Ting, New algorithm for second order boundary value problems of integro-differential equation. J. Comput. Appl. Math. 229 (2009) 1-6.
[40] A. Pedas, E. Tamme, A discrete collocation method for Fredholm integro-differential equation with weakly singular kernels. Appl. Numer. Math. 61 (2011) 738-751.
[41] Web resource: https://en.wikipedia.org/wiki/Parallel_computing
[42] V. Volpert: Elliptic Partial Differential Equations (Vol. 2). Springer Basel (2014) pp 534-548.

# APPENDIX A: Critical code listing

## File main.m

```
clc;
clear all;
tic;
syms t
A(t)=[[0, t];[t^2, 0]];
fi_1(t)= [(t^3)*exp(t/4) t; 1 2*(t^3)*exp(t/4)];
fi_2(t)= (t^4/2)*exp(t/4)*[1 0; 0 2];

psi_1(t) = [t/t 0; 0 t/t];
psi_2(t) = (t-1/2)*[1 0; 0 1];
xx(t)=[sin(2*pi*t); cos(2*pi*t)];
r_1(t)=psi_1(t)*xx(t);
r_2(t)=psi_2(t)*xx(t);
f(t)=diff(xx)-A(t)*xx(t)-fi_1*int(r_1,0,1)-fi_2*int(r_2,0,1);

h=0.05;
t0=0:h:0.5;
t1=0.5:h:1;
x0=0:h:0.5;
x1=0.5:h:1;
N1 = size(t0,2);
N2 = size(t1,2);

tol = 1e-12;
% a(A(), t)-1
[a1_A1, ~]=BulirschStoer(@F1,t0,[0;0],tol);
a1_A1 = a1_A1';
[a2_A1, ~]=BulirschStoer(@F1,t1,[0;0],tol);
```

```
a2_A1 = a2_A1';
% a(A(), t)-2
[a1_A2, ~]=BulirschStoer(@F2,t0,[0;0],tol);
a1_A2 = a1_A2';
[a2_A2, ~]=BulirschStoer(@F2,t1,[0;0],tol);
a2_A2 = a2_A2';
% a(f(), t)
[a1_F, ~]=BulirschStoer(@F5,t0,[0;0],tol);
a1_F = a1_F';
[a2_F, ~]=BulirschStoer(@F5,t1,[0;0],tol);
a2_F = a2_F';
% a(fi_1(), t)-1
[a1_fi1_1, ~]=BulirschStoer(@F3_1,t0,[0;0],tol);
a1_fi1_1 = a1_fi1_1';
[a2_fi1_1, ~]=BulirschStoer(@F3_1,t1,[0;0],tol);
a2_fi1_1 = a2_fi1_1';
% a(fi_1(), t)-2
[a1_fi1_2, ~]=BulirschStoer(@F4_1,t0,[0;0],tol);
a1_fi1_2 = a1_fi1_2';
[a2_fi1_2, ~]=BulirschStoer(@F4_1,t1,[0;0],tol);
a2_fi1_2 = a2_fi1_2';
% a(fi_2(), t)-1
[a1_fi2_1, ~]=BulirschStoer(@F3_2,t0,[0;0],tol);
a1_fi2_1 = a1_fi2_1';
[a2_fi2_1, ~]=BulirschStoer(@F3_2,t1,[0;0],tol);
a2_fi2_1 = a2_fi2_1';
% a(fi_2(), t)-2
[a1_fi2_2, ~]=BulirschStoer(@F4_2,t0,[0;0],tol);
a1_fi2_2 = a1_fi2_2';
[a2_fi2_2, ~]=BulirschStoer(@F4_2,t1,[0;0],tol);
a2_fi2_2 = a2_fi2_2';
```

```
psi_11 = double(int(psi_1, 0, 0.5));

psi_12 = double(int(psi_1, 0.5, 1));

psi_21 = double(int(psi_2, 0, 0.5));

psi_22 = double(int(psi_2, 0.5, 1));


psi1_a1f=[simps(t0,a1_F(:,1));simps(t0,a1_F(:,2))];

psi1_a2f=[simps(t1,a2_F(:,1));simps(t1,a2_F(:,2))];

psi2_a1f=[simps(t0,(x0 - 1/2).*a1_F(:,1)');simps(t0,(x0 -
1/2).*a1_F(:,2)')];

psi2_a2f=[simps(t1,(x1 - 1/2).*a2_F(:,1)');simps(t1,(x1 -
1/2).*a2_F(:,2)')];




psi1_a1A = [simps(t0,a1_A1(:,1)) simps(t0,a1_A2(:,1));

            simps(t0,a1_A1(:,2)) simps(t0,a1_A2(:,2))];

psi1_a2A = [simps(t0,a2_A1(:,1)) simps(t0,a2_A2(:,1));

            simps(t0,a2_A1(:,2)) simps(t0,a2_A2(:,2))];


psi2_a1A = [simps(t0,(x0 - 1/2).*a1_A1(:,1)') simps(t0,(x0 -
1/2).*a1_A2(:,1)');

            simps(t0,(x0 - 1/2).*a1_A1(:,2)') simps(t0,(x0 -
1/2).*a1_A2(:,2)')];

psi2_a2A = [simps(t0,(x1 - 1/2).*a2_A1(:,1)') simps(t0,(x1 -
1/2).*a2_A2(:,1)');

            simps(t0,(x1 - 1/2).*a2_A1(:,2)') simps(t0,(x1 -
1/2).*a2_A2(:,2)')];



psi1_a1_fi1 = [simps(t0,a1_fi1_1(:,1)) simps(t0,a1_fi1_2(:,1));

               simps(t0,a1_fi1_1(:,2)) simps(t0,a1_fi1_2(:,2))];

psi1_a1_fi2 = [simps(t0,a1_fi2_1(:,1)) simps(t0,a1_fi2_2(:,1));
```

```
                simps(t0,a1_fi2_1(:,2)) simps(t0,a1_fi2_2(:,2))];

psi1_a2_fi1 = [simps(t1,a2_fi1_1(:,1)) simps(t1,a2_fi1_2(:,1));

                simps(t1,a2_fi1_1(:,2)) simps(t1,a2_fi1_2(:,2))];

psi1_a2_fi2 = [simps(t1,a2_fi2_1(:,1)) simps(t1,a2_fi2_2(:,1));

                simps(t1,a2_fi2_1(:,2)) simps(t1,a2_fi2_2(:,2))];


psi2_a1_fi1 = [simps(t0,(x0 - 1/2).*a1_fi1_1(:,1)') simps(t0,(x0 -
1/2).*a1_fi1_2(:,1)');

                simps(t0,(x0 - 1/2).*a1_fi1_1(:,2)') simps(t0,(x0 -
1/2).*a1_fi1_2(:,2)')];

psi2_a1_fi2 = [simps(t0,(x0 - 1/2).*a1_fi2_1(:,1)') simps(t0,(x0 -
1/2).*a1_fi2_2(:,1)');

                simps(t0,(x0 - 1/2).*a1_fi2_1(:,2)') simps(t0,(x0 -
1/2).*a1_fi2_2(:,2)')];

psi2_a2_fi1 = [simps(t1,(x1 - 1/2).*a2_fi1_1(:,1)') simps(t1,(x1 -
1/2).*a2_fi1_2(:,1)');

                simps(t1,(x1 - 1/2).*a2_fi1_1(:,2)') simps(t1,(x1 -
1/2).*a2_fi1_2(:,2)')];

psi2_a2_fi2 = [simps(t1,(x1 - 1/2).*a2_fi2_1(:,1)') simps(t1,(x1 -
1/2).*a2_fi2_2(:,1)');

                simps(t1,(x1 - 1/2).*a2_fi2_1(:,2)') simps(t1,(x1 -
1/2).*a2_fi2_2(:,2)')];



g_f = [psi1_a1f + psi1_a2f; psi2_a1f + psi2_a2f];


G = [psi1_a1_fi1 + psi1_a2_fi1 psi1_a1_fi2 + psi1_a2_fi2;

     psi2_a1_fi1 + psi2_a2_fi1 psi2_a1_fi2 + psi2_a2_fi2];


V_11 = psi1_a1A + psi1_a1_fi1.*psi_11 + psi1_a1_fi2.*psi_21 +
psi1_a2_fi1*psi_11 + psi1_a2_fi2*psi_21;

V_12 = psi1_a2A + psi1_a1_fi1.*psi_12 + psi1_a1_fi2.*psi_22 +
psi1_a2_fi1*psi_12 + psi1_a2_fi2*psi_22;

V_21 = psi2_a1A + psi2_a1_fi1.*psi_11 + psi2_a1_fi2.*psi_21 +
psi2_a2_fi1*psi_11 + psi2_a2_fi2*psi_21;
```

```
V_22 = psi2_a2A + psi2_a1_fi1.*psi_12 + psi2_a1_fi2.*psi_22 +
psi2_a2_fi1*psi_12 + psi2_a2_fi2*psi_22;

V = [V_11 V_12; V_21 V_22];


K = (eye(4,4) - G)\V;

F = (eye(4,4) - G)\g_f;


L_11 = K(1:2,1:2) + psi_11;

L_12 = K(1:2,3:4) + psi_12;

L_21 = K(3:4,1:2) + psi_21;

L_22 = K(3:4,3:4) + psi_22;

F_1 = F(1:2);

F_2 = F(3:4);


I = eye(2,2);

B = eye(2,2);

C = -B;

d = [0;0];


a1_fi1_N = [a1_fi1_1(N1,1) a1_fi1_2(N1,1);
            a1_fi1_1(N1,2) a1_fi1_2(N1,2)];
a1_fi2_N = [a1_fi2_1(N1,1) a1_fi2_2(N1,1);
            a1_fi2_1(N1,2) a1_fi2_2(N1,2)];
a2_fi1_N = [a2_fi1_1(N2,1) a2_fi1_2(N2,1);
            a2_fi1_1(N2,2) a2_fi1_2(N2,2)];
a2_fi2_N = [a2_fi2_1(N2,1) a2_fi2_2(N2,1);
            a2_fi2_1(N2,2) a2_fi2_2(N2,2)];


a1A_N = [a1_A1(N1,1) a1_A2(N1,1);
         a1_A1(N1,2) a1_A2(N1,2)];
a2A_N = [a2_A1(N2,1) a2_A2(N2,1);
```

```
         a2_A1(N2,2) a2_A2(N2,2)];

a1f_N=[a1_F(N1,1);a1_F(N1,2)];
a2f_N=[a2_F(N2,1);a2_F(N2,2)];

Q_11 = B + C * (a2_fi1_N*L_11 + a2_fi2_N*L_21);
Q_12 = C + C * (a2A_N + a2_fi1_N*L_12 + a2_fi2_N*L_22);
Q_21 = I + I * (a1A_N + a1_fi1_N*L_11 + a1_fi2_N*L_21);
Q_22 = I * (a1_fi1_N*L_12 + a1_fi2_N*L_22) - I;

d_1 = d - C * (a2_fi1_N*F_1 + a2_fi2_N*F_2 + a2f_N);
d_2 = -(a1_fi1_N*F_1 + a1_fi2_N*F_2 + a1f_N);

Q = [Q_11 Q_12; Q_21 Q_22];
D = [d_1;d_2];

lambda = Q\D;

lambda_1 = [lambda(1); lambda(2)];
lambda_2 = [lambda(3); lambda(4)];

a1_fi1 = [a1_fi1_1(:,1) a1_fi1_2(:,1);
           a1_fi1_1(:,2) a1_fi1_2(:,2)];
a1_fi2 = [a1_fi2_1(:,1) a1_fi2_2(:,1);
           a1_fi2_1(:,2) a1_fi2_2(:,2)];
a2_fi1 = [a2_fi1_1(:,1) a2_fi1_2(:,1);
           a2_fi1_1(:,2) a2_fi1_2(:,2)];
a2_fi2 = [a2_fi2_1(:,1) a2_fi2_2(:,1);
           a2_fi2_1(:,2) a2_fi2_2(:,2)];

a1A = [a1_A1(:,1) a1_A2(:,1);
```

```
         a1_A1(:,2) a1_A2(:,2)];
a2A = [a2_A1(:,1) a2_A2(:,1);
         a2_A1(:,2) a2_A2(:,2)];


a1f = [a1_F(:,1);a1_F(:,2)];
a2f = [a2_F(:,1);a2_F(:,2)];


u_1 = (a1A + a1_fi1*L_11 + a1_fi2 * L_21) * lambda_1 +
(a1_fi1*L_12 + a1_fi2*L_22) * lambda_2 + a1_fi1*F_1 + a1_fi2*F_2 +
a1f;
u_2 = (a2A + a2_fi1*L_12 + a2_fi2 * L_22) * lambda_2 +
(a2_fi1*L_11 + a2_fi2*L_21) * lambda_1 + a2_fi1*F_1 + a2_fi2*F_2 +
a2f;


x_final = zeros(length(u_1), 2);
parfor i=1:length(u_1)
    if i <= N1
        x_final(i,:) = [u_1(i,1) u_2(i,1)] + [lambda_1(1)
lambda_2(1)];
    else
        x_final(i,:) = [u_1(i,1) u_2(i,1)] + [lambda_1(2)
lambda_2(2)];
    end
end


x_true = zeros(length(u_1),2);
parfor k=1:size(t0,2)
  x_true(k,:) = [sin(2*pi*t0(1,k)) sin(2*pi*t1(1,k))];
end


for k=1:size(t0,2)
    x_true(11+k,:) = [cos(2*pi*t0(1,k)) cos(2*pi*t1(1,k))];
end
```

```
max_error = max(max(abs(x_true-x_final)))
sum_error = sum(sum(abs(x_true-x_final)))

TimeSpent = toc;
```

File F1.m

```
function xp=F1(t,x)
xp=zeros(2,1); % since output must be a column vector
xp(1)=t*x(2);
xp(2)=t^2*x(1)+t^2;
```

File F2.m

```
function xp=F2(t,x)
xp=zeros(2,1); % since output must be a column vector
xp(1)=t*x(2)+t;
xp(2)=t^2*x(1);
```

File F3_1.m

```
function xp=F3_1(t,x)
xp=zeros(2,1); % since output must be a column vector
xp(1) = t*x(2) + t^3*exp(t/4);
xp(2) = t^2*x(1) + 1;
```

File F3_2.m

```
function xp=F3_2(t,x)
xp=zeros(2,1); % since output must be a column vector
xp(1) = t*x(2) + (t^4*exp(t/4))/2;
xp(2) = t^2*x(1);
```

File F4_1.m

```
function xp=F4_1(t,x)

xp=zeros(2,1); % since output must be a column vector

xp(1) = t*x(2) + t;

xp(2) = t^2*x(1) + 2*t^3*exp(t/4);
```

File F4_2.m

```
function xp=F4_2(t,x)

xp=zeros(2,1); % since output must be a column vector

xp(1) = t*x(2);

xp(2) = t^2*x(1) + t^4*exp(t/4);
```

File F5.m

```
function xp=F5(t,x)

xp=zeros(2,1); % since output must be a column vector

xp(1) = t*x(2) + 2*pi*cos(2*pi*t) - t*cos(2*pi*t) +
(t^4*exp(t/4))/(4*pi);

xp(2) = t^2*x(1) - 2*pi*sin(2*pi*t) - t^2*sin(2*pi*t);
```

File F6.m

```
function xp=F6(t,x)

xp=zeros(2,1); % since output must be a column vector

xp(1) = x(2) + (1378650775960005119 * t)/540431955284459520 +
2*t*(t - 1) - 2795952374767185/2251799813685248;

xp(2) = t^2 * x(1) - t^3 * (t - 1) - 4*t +
(2942530675988587*t^2)/6755399441055744 +
10099266730594293337/540431955284459520;
```

APPENDIX B: The list of scientific publications on the topic of master's work

1. D.S. Dzhumabaev, A.S. Zharmagambetov, Numerical method for solving a linear boundary value problem for Fredholm integro-differential equations. News of the national academy of sciences of the RK. March-April 2017. 2(232)
2. A.S. Zharmagambetov, Solving Fredholm integro-differential equations by partition method. Materials of conference “Language and IT: forming new linguocultural paradigm”. Almaty, 2016