КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2020 Т. 12 № 2 С. 275–300
DOI: 10.20537/2076-7633-2020-12-2-275-300

Ки&М

МАТЕМАТИЧЕСКИЕ ОСНОВЫ И ЧИСЛЕННЫЕ МЕТОДЫ МОДЕЛИРОВАНИЯ

УДК: 519.642

Численное решение нелинейных интегральных уравнений второго рода типа Урысона методом последовательных квадратур с использованием погруженной схемы Дормана–Принса 5(4)

И. И. Маглеванный[a], Т. И. Карякина[b]

Волгоградский государственный социально-педагогический университет,
Россия, 400066, г. Волгоград, пр. Ленина, д. 27

E-mail: [a] sianko@list.ru, [b] karyakina@fizmat.vspu.ru

Получено 25.03.2019, после доработки — 12.11.2019.
Принято к публикации 24.12.2019.

Представлен итерационный алгоритм, который численно решает нелинейные одномерные несингулярные интегральные уравнения Фредгольма и Вольтерры второго рода типа Урысона. Показано, что метод последовательных приближений Пикара может быть использован при численном решении такого типа уравнений. Сходимость числовой схемы гарантируется теоремами о неподвижной точке. При этом квадратурный алгоритм основан на явной форме встроенного правила Рунге–Кутты пятого порядка с адаптивным контролем размера шага. Возможность контроля локальных ошибок квадратур позволяет создавать очень точные автоматические числовые схемы и значительно уменьшить основной недостаток итераций Пикара, а именно чрезвычайно большое количество вычислений с увеличением глубины рекурсии. Наш алгоритм организован так, что по сравнению с большинством подходов нелинейность интегральных уравнений не вызывает каких-либо дополнительных вычислительных трудностей, его очень просто применять и реализовывать в программе. Наш алгоритм демонстрирует практически важные черты универсальности. Во-первых, следует подчеркнуть, что метод столь же прост в применении к нелинейным, как и к линейным уравнениям типа Фредгольма и Вольтерры. Во-вторых, алгоритм снабжен правилами останова, по которым вычисления могут в значительной степени контролироваться автоматически. Представлен компактный C++-код описанного алгоритма. Реализация нашей программы является самодостаточной: она не требует никаких предварительных вычислений, никаких внешних функций и библиотек и не требует дополнительной памяти. Приведены числовые примеры, показывающие применимость, эффективность, надежность и точность предложенного подхода.

Ключевые слова: уравнения типа Фредгольма и Вольтерры, теорема о неподвижной точке, анализ погрешностей ошибок, итерационные методы, погруженный метод Рунге–Кутты пятого порядка, адаптивный контроль величины шага

COMPUTER RESEARCH AND MODELING
2020 VOL. 12 NO. 2 P. 275–300
DOI: 10.20537/2076-7633-2020-12-2-275-300

Ки&М

MATHEMATICAL MODELING AND NUMERICAL SIMULATION

UDC: 519.642

Numerical solution of Urysohn type nonlinear second kind integral equations by successive quadratures using embedded Dormand and Prince scheme 5(4)

I. I. Maglevanny[a], T. I. Karyakina[b]

Volgograd State Socio-Pedagogical University,
27 pr. Lenina, Volgograd, 400066, Russia

E-mail: [a] sianko@list.ru, [b] karyakina@fizmat.vspu.ru

Received 25.03.2019, after completion — 12.11.2019.
Accepted for publication 24.12.2019.

We present the iterative algorithm that solves numerically both Urysohn type Fredholm and Volterra nonlinear one-dimensional nonsingular integral equations of the second kind to a specified, modest user-defined accuracy. The algorithm is based on descending recursive sequence of quadratures. Convergence of numerical scheme is guaranteed by fixed-point theorems. Picard's method of integrating successive approximations is of great importance for the existence theory of integral equations but surprisingly very little appears on numerical algorithms for its direct implementation in the literature. We show that successive approximations method can be readily employed in numerical solution of integral equations. By that the quadrature algorithm is thoroughly designed. It is based on the explicit form of fifth-order embedded Runge–Kutta rule with adaptive step-size self-control. Since local error estimates may be cheaply obtained, continuous monitoring of the quadrature makes it possible to create very accurate automatic numerical schemes and to reduce considerably the main drawback of Picard iterations namely the extremely large amount of computations with increasing recursion depth. Our algorithm is organized so that as compared to most approaches the nonlinearity of integral equations does not induce any additional computational difficulties, it is very simple to apply and to make a program realization. Our algorithm exhibits some features of universality. First, it should be stressed that the method is as easy to apply to nonlinear as to linear equations of both Fredholm and Volterra kind. Second, the algorithm is equipped by stopping rules by which the calculations may to considerable extent be controlled automatically. A compact C++-code of described algorithm is presented. Our program realization is self-consistent: it demands no preliminary calculations, no external libraries and no additional memory is needed. Numerical examples are provided to show applicability, efficiency, robustness and accuracy of our approach.

Keywords: nonlinear Volterra–Fredholm integral equations, fixed point theorem, error analysis, iterative methods, fifth-order embedded Runge–Kutta rule, adaptive step-size control

Citation: *Computer Research and Modeling*, 2020, vol. 12, no. 2, pp. 275–300 (Russian).

1. Introduction

Integral equations are encountered in a variety of applications in many fields including continuum mechanics, potential theory, geophysics, electricity and magnetism, kinetic theory of gases, hereditary phenomena in physics and biology, renewal theory, quantum mechanics, radiation, optimization, optimal control systems, communication theory, mathematical economics, population genetics, queuing theory, medicine, mathematical problems of radiative equilibrium, the particle transport problems of astrophysics and reactor theory, acoustics, fluid mechanics, steady state heat conduction, fracture mechanics, and radiative heat transfer problems.

A computational approach to solving integral equation is an essential work in scientific research. One may find in the references [Atkinson, 1992, 1997; Atkinson, Potra, 1987; Linz, 1985], a review of classical numerical methods for solving integral equations.

Some novel methods for solving second kind integral equation are available in the open literature and very large variety of novel techniques which emphasize the importance of interdisciplinary effort for advancing the study on numerical methods for solving integral equations have been applied to obtaining numerical solutions of the nonlinear integral equations during the past years. Among them we note a different method [Borzabadi et al., 2006], discrete Adomian decomposition method [Behiry et al., 2010], homotopy perturbation method [Biazar, Ghazvini, 2008]. Positive definite function method [Alipanah, Dehghan, 2007] is based on interpolation by radial basis functions. Triangular functions method [Maleknejad et al., 2010] and harmonic wavelet method [Cattani, Kudreykom, 2010] were utilized as a basis in the collocation method, sinc approximation was utilized in [Maleknejad et al., 2013], multigrid method was utilized in [Lee, 1997], optimal homotopy asymptotic method was introduced in [Hashmi et al., 2012]. Methods based on the use of Haar wavelets are presented in [Babolian, Shahsavaran, 2009; Erfanian et al., 2014]. Iterative methods based on discretization and projection techniques are presented in [Borzabadi, Fard, 2009; Javadi, 2014]. Random search algorithm is presented in [Farnoosh, Ebrahimi, 2008], Newton–Kantorovich-quadrature method is presented in [Saberi-Nadjafi, Heidari, 2010], etc.

However, each of these methods has its inherent advantages and disadvantages. Most of them consider nonlinear Fredholn and Volterra kind equations separately or deal with only Hammerstein type equations and do not consider more general Urysohn type equations. In general the algorithm structure is rather complicated, needs vast preliminary calculations and is therefore difficult to be programmed (most authors present no program code).

We propose to solve the integral equations of most general type by means of successive approximations method in its classical form using no collocation or projection technique. Picard's method of integrating successive approximations is of great importance for the existence theory of integral equations but surprisingly very little appears on numerical algorithms for its direct implementation in the literature. This may be partly due to the fact that there is the lack of powerful quadrature rules in evaluating multidimensional integrals. We show that successive approximations method can be readily employed in numerical solution of integral equations.

The goal of this article is to present the realization of an algorithm which exhibits some features of universality. First, it should be stressed that the method is as easy to apply to nonlinear as to linear equations of both Fredholm and Volterra kind and as compared to most approaches the nonlinearity of integral equations does not induce any additional computational difficulties. Second, the algorithm is equipped by stopping rules by which the calculations may to considerable extent be controlled automatically.

The algorithm is based on descending recursive sequence of quadratures. By that the quadrature algorithm is thoroughly designed. It is based on the explicit form of fifth-order embedded Runge-Kutta rule with adaptive step-size self-control. Since local error estimates may be cheaply obtained, continuous monitoring of the quadrature makes it possible to create very accurate automatic numerical

schemes and to reduce considerably the main drawback of Picard iterations namely the extremely large amount of computations with increasing recursion depth.

Any possible program realization will be self-consistent: our algorithm demands no preliminary calculations, no external functions and libraries and no additional memory is needed. Convergence of numerical scheme is guaranteed by fixed-point theorems and some numerical examples are provided to show applicability, efficiency, robustness and accuracy of the proposed approach.

The integral equations which will be treated by our schemes are the Urysohn integral equations of the second kind. The paper is organized as follows.

In Sect. 2, we present several tools of the trade [Atkinson, Han, 2009; Zemyan, 2012] that are indispensable for the comprehension of the material in this article. In Subsect. 2.1, we apply a fixed point theorem originally enunciated by Banach to establish the existence and uniqueness of a solution to an integral equation of Fredholm type, when the integrand satisfies a Lipschitz condition. An analogous theorem for nonlinear integral equations of Volterra type is also considered in Subsect. 2.3. Other fixed point theorems, e.g., those of Brouwer and Schauder [Krasnosel'skii, 1964; Krasnosel'skii, Zabreyko, 1984; Polyanin, Manzhirov, 2008], can also be invoked to prove existence and uniqueness theorems in different settings, but they are difficult to be implemented in computational scheme and we not pursue those here.

A typical iterative algorithm in numerical analysis and scientific computing requires a stopping criterion. Such an algorithm involves a sequence of generated iterates or steps, an error tolerance, and a method to estimate *a priory* (or compute *a posteriori*) some quantity related to the error. If this error quantity is below the tolerance then the iterative procedure is stopped and success is declared. The sopping criteria are discussed in Subsect. 2.2 and Subsect. 2.4.

In Sect. 3 the design of our algorithm is outlined and in Subsect. 3.1 the algorithm for quadrature estimation is described.

In Sect. 4, we consider a large number of examples from different references in order to illustrate many phenomena that can and do occur in an application of our iterative algorithm to both of linear and nonlinear integral equations.

In Appendix, we present C++-code of our algorithm.

2. Contraction mappings and Banach fixed point theorems

Consider the space $C[a, b]$ consisting of the real-valued continuous functions that are defined on the interval $[a, b]$. It is a complete metric space with the metric $d(y, z) = \max_{t\in[a,b]} |y(t) - z(t)|$.

Let X be a metric space with metric d, and let A be a mapping from X into itself. Then A is called a contraction mapping if $d(Ay, Az) \le \beta d(y, z)$ for all $y \in X$, where $0 < \beta < 1$. The constant β does not depend in any way on the elements of X. If X is a space consisting of functions, then A is called a contraction operator and β is called a contractivity constant [Atkinson, Han, 2009; Zemyan, 2012].

Let A be a mapping from a metric space X into itself. If $A(y^*) = y^*$, then y^* is called a fixed point of the mapping A.

Corollary 1 (Banach). *Let X be a complete metric space endowed with the metric d. If A is a contraction mapping from X into itself, then there exists a unique point $y^* \in X$ such that $A(y^*) = y^*$.*

Not every continuous mapping from X into itself must be a contraction mapping. However, it may happen that some power A^n of A might be a contraction mapping, even if A is not. In this case, another fixed point theorem can be enunciated [Zemyan, 2012].

Corollary 2. *Let A be a continuous mapping from the complete metric space X into itself. If A^n is a contraction mapping for some positive integer n, then there exists a unique point $y^* \in X$ such that $A(y^*) = y^*$.*

2.1. Nonlinear Fredholm integral equations of Urysohn type

A general form for such an equation is

$$y(x) = F(x) + \lambda \int_a^b K(x, t, y(t))\, dt, \quad x \in [a, b], \tag{1}$$

where $y(x)$ is the unknown function and F and K are given functions. The function F is known as the driving or free term, and K as the kernel of the integral equation. We assume that the integrand $K(x, t, y)$ is continuous on the set $x, t \in [a, b]$. It is natural to ask whether solutions of nonlinear integral equations exist and, if so, whether these solutions are unique. It is also reasonable to explore the role that the parameter λ plays in the determination of these solutions.

Consider the operator $U : C[a, b] \to C[a, b]$ defined as

$$U(y) = U(y)(x) = F(x) + \lambda \int_a^b K(x, t, y(t))\, dt.$$

Suppose that the kernel function K satisfies a uniform Lipschitz condition with respect to its third argument

$$|K(x, t, y_1) - K(x, t, y_2)| < M|y_1 - y_2|, \quad \forall x, t \in [a, b], \tag{2}$$

The value $M > 0$ is the characteristic of kernel "size"and is called the Lipschitz constant.

Observe that for any $y, z \in C[a, b]$ and any $x \in [a, b]$, we have

$$|U(y)(x) - U(z)(x)| = |\lambda| \left| \int_a^b \Big[K(x, t, y(t)) - K(x, t, z(t)) \Big]\, dt \right| \le |\lambda| M \left| \int_a^b |y(t) - z(t)|\, dt \right| \le |\lambda| M (b - a) d(y, z).$$

It follows directly from this inequality that

$$d(U(y), U(z)) \le \alpha d(y, z), \quad \alpha = |\lambda| M (b - a).$$

In general, after m such integrations, we obtain [Atkinson, Han, 2009; Zemyan, 2012]

$$d(U^m(y), U^m(z)) \le \alpha^m d(y, z). \tag{3}$$

Hence, if parameter $\alpha < 1$, then U is a contraction operator from $C[a, b]$ into itself and we may set $\beta = \alpha$. Thus Banach's fixed point theorem can be used to state that the nonlinear Fredholm integral equation $y(x) = U(y(x))$ has a unique solution on the interval $[a, b]$ whenever $\alpha < 1$. This solution is defined as a limit of functional sequence

$$y(x) = \lim_{n \to \infty} y_n(x), \quad y_n(x) = F(x) + \lambda \int_a^b K(x, t, y_{n-1}(t))\, dt \tag{4}$$

for any initial approximation $y_0(x)$, and this sequence converges uniformly for $x \in [a, b]$.

2.2. Error estimations and stopping criteria for Fredholm integral equations

Iterative numerical algorithms should be equipped with a stopping criteria, where the iteration process is terminated when some error measure is deemed to be below a given tolerance. This is a useful setting for algorithm performance. There are several general criteria for stopping, which can be combined if necessary.

1. Maximum number of steps. Using this method for stopping, we impose some maximum number of steps in the iteration $n_{\max}$ and stop when $n = n_{\max}$. This provides a safeguard against infinite loops, but is not very efficient — i.e. even if we are very close to the solution after some iterations, this method will always run for the same number of iterations.

2. Tolerance on the size of the correction. Under this criterion, we are given some tolerance and stop when difference $|y_n(x) - y_{n-1}(x)|$ becomes sufficiently small.

From (4) it follows that the solution $y(x)$ is the sum of uniformly convergent series

$$y(x) = y_0(x) + \sum_{m=0}^{\infty} [y_{m+1}(x) - y_m(x)] = y_n(x) + \sum_{m=n}^{\infty} [y_{m+1}(x) - y_m(x)]. \tag{5}$$

Then

$$|y(x) - y_n(x)| = \left| \sum_{m=n}^{\infty} [y_{m+1}(x) - y_m(x)] \right| \le \sum_{m=n}^{\infty} |y_{m+1}(x) - y_m(x)| \le \sum_{m=n}^{\infty} d(y_{m+1}, y_m) = \sum_{m=n}^{\infty} d\,(U^m(y_1), U^m(y_0)).$$

And from (3) it follows the *a priori* uniform error estimation

$$|y(x) - y_n(x)| \le \sum_{m=n}^{\infty} \alpha^m d(y_1, y_0) = \frac{\alpha^n}{1-\alpha}\, d(y_1, y_0) \quad \forall x \in [a, b]. \tag{6}$$

From (6) it follows that the estimate of convergence rate is defined by condition

$$|y(x) - y_n(x)| \le \alpha |y(x) - y_{n-1}(x)| \tag{7}$$

which means that the convergence rate is linear.

Now let τ be some user-defined tolerance. Let $n_{\max}$ be the smallest number of iterations for which the inequality $|y(x) - y_n(x)| < \tau$ holds for all $x \in [a, b]$. Thus the value of $n_{\max}$ is defined by condition

$$\frac{\alpha^n}{1-\alpha}\, d(y_1, y_0) = \frac{[|\lambda| M(b-a)]^n}{1 - |\lambda| M(b-a)}\, d(y_1, y_0) \ge \tau, \quad n < n_{\max}, \quad \forall x \in [a, b], \tag{8}$$

and we break the iteration process at $n = n_{\max}$ just receiving the solution with accuracy τ.

In general the error defined by (6), is overestimated. Moreover it depends strongly on the hypothetical value of M which is hard to estimate for Urysohn kernels of general form. We could not find in literature the reasonable methods of estimation of M for Urysohn kernels of general form except for the case when the function $K(x, t, y)$ has the bounded partial derivative on y [Polyanin, Manzhirov, 2008]:

$$\left| \frac{\partial K}{\partial y} \right| \le L, \quad x, t \in [a, b], \quad y \in (-\infty, \infty). \tag{9}$$

If inequality (9) holds we may set $M = L$. But in general it is difficult to check numerically the inequality (9) so we decided to leave the value of M at user's response (but for nonlinear equations only) or set $M = 0$ when there is no guess on its value.

Note that for linear equations such problem does not appear (see below) and in this case the value of M may be estimated numerically so the calculations may be fully automated.

Thus the value $n_{\max}$ may be too large or may be unknown and this dictates to use the stopping criterion which is independent of M. Such criterion is based on the size of the correction and has the form

$$\delta = \frac{|y_n(x) - y_{n-1}(x)|}{atol + rtol \cdot \max(|y_n(x)|, ||y_{n-1}(x))} < 1, \quad n \geq n_{\min}. \tag{10}$$

Here *atol* and *rtol* are optional absolute and relative error tolerances, respectively. We use $atol = rtol = 0.1\tau$ in our code. This choice is the safest in general but may be changed if need be.

In (10) the value $n_{\min}$ reflects our knowledge at which iteration the operator U becomes contractive. If the value of Lipschitz constant M is established and the contractivity constant $\alpha = |\lambda|M(b-a) < 1$ then it is natural to set $n_{\min} = 1$. Moreover if we have no guess about the value of M (a typical case for nonlinear equations) we simply also set $n_{\min} = 1$ and accomplish iterations with hope that U is contractive and before reaching the prescribed value $n_{\max}$ at some $n < n_{\max}$ the equality (10) will be fulfilled just leading to breaking iterations and declaring the success. Maybe it looks as a some trick but it works! At least some nonlinear equations of Fredholm kind presented below as examples were solved using described strategy and good results were demonstrated in all cases.

Note that as distinct from uniform condition $n \leq n_{\max}$ the stopping criterion (10) is point-wise i.e. it depends on the value of x. This point-wise stopping criterion may speed up the calculations considerably, as will be shown below.

2.3. Nonlinear Volterra integral equations of Urysohn type

A general form for such an equation is

$$y(x) = F(x) + \lambda \int_a^x K(x, t, y(t))\, dt, \quad x \in [a, b], \tag{11}$$

where the kernel K satisfies a Lipschitz condition (2). Consider the operator $V\colon C[a,b] \to C[a,b]$ defined as

$$V(y) = V(y)(x) = F(x) + \lambda \int_a^x K(x, t, y(t))\, dt. \tag{12}$$

Observe that for any $y, z \in C[a,b]$ and any $x \in [a,b]$, we have

$$|V(y)(x) - V(z)(x)| = |\lambda| \left| \int_a^x \Big[K(x,t,y(t)) - K(x,t,z(t)) \Big]\, dt \right| \leq |\lambda| M \left| \int_a^x |y(t) - z(t)|\, dt \right| \leq |\lambda| M (x-a) d(y,z).$$

After m such integrations, we obtain [Atkinson, Han, 2009; Zemyan, 2012]

$$d(V^m(y), V^m(z)) \leq \frac{[|\lambda|M(x-a)]^m}{m!} d(y,z) \leq \frac{\alpha^m}{m!} d(y,z), \quad \alpha = |\lambda|M(b-a). \tag{13}$$

Hence, if m is sufficiently large, then V^m is a contraction operator from $C[a,b]$ into itself.

Now let $n_{\min}$ be the smallest n, for which operator V^n becomes contractive i.e. the following inequality holds

$$\frac{\alpha^n}{n!} \geq 1, \quad n < n_{\min}. \tag{14}$$

Then the operator $V^{n_{\min}}$ is contractive and the contractivity constant β is defined as $\beta = \alpha^{n_{\min}}/n_{\min}!$. Thus Banach's fixed point theorem can be used to state that the nonlinear Volterra integral equation

$y(x) = V(y(x))$ has a unique solution on the interval $[a, b]$. This solution is defined as a limit of functional sequence

$$y(x) = \lim_{n\to\infty} y_n(x), \quad y_n(x) = F(x) + \lambda \int_a^x K(x, t, y_{n-1}(t))\, dt \tag{15}$$

for any initial approximation $y_0(x)$, and this sequence converges uniformly for $x \in [a, b]$ at $n \geq n_{\min}$.

It means that, differently from the case of Fredholm kind equation, any Volterra operator (12) for which the kernel "size" is restricted according to (2) has the fixed point for any λ and, consequently, any such integral equation of Volterra kind may, in principle, be solved by iterations (15).

2.4. Error estimations and stopping criteria for Volterra integral equations

From (15) it follows that the solution $y(x)$ is the sum of uniformly convergent series of form (5). Then

$$|y(x) - y_n(x)| = \left|\sum_{m=n}^{\infty} [y_{m+1}(x) - y_m(x)]\right| \leq \sum_{m=n}^{\infty} |y_{m+1}(x) - y_m(x)| \leq \sum_{m=n}^{\infty} d(y_{m+1}, y_m) = \sum_{m=n}^{\infty} d\,(V^m(y_1), V^m(y_0)).$$

From (13) it follows the *a priori* uniform error estimation

$$|y(x) - y_n(x)| \leq \sum_{m=n}^{\infty} \frac{\alpha^m}{m!} d(y_1, y_0), \quad n \geq n_{\min}, \quad \forall x \in [a, b].$$

The sum on the right can be estimated by Lagrange's form of the remainder for the exponential series. In doing so, we obtain the uniform estimate

$$|y(x) - y_n(x)| \leq \frac{e^b \alpha^n}{n!} d(y_1, y_0), \quad \forall x \in [a, b]. \tag{16}$$

Now let τ be some user-defined tolerance. Let $n_{\max}$ be the smallest number of iterations for which the inequality $|y(x) - y_n(x)| < \tau$ is fulfilled for all $x \in [a, b]$. Thus the value of $n_{\max}$ is defined by condition

$$\frac{e^b \alpha^n}{n!} d(y_1, y_0) = \frac{e^b [|\lambda| M(b-a)]^n}{n!}\, d(y_1, y_0) \geq \tau, \quad n < n_{\max}, \quad \forall x \in [a, b], \tag{17}$$

and we break the iteration process at $n = n_{\max}$ just receiving the solution with accuracy τ. Note that $n_{\max} > n_{\min}$.

When performing iterations we may use the point-wise stopping criterion (10) only at $n \geq n_{\min}$. This fact may produce the problem for solution of Volterra kind equation with large $|\lambda|$. If $n_{\min}$ is large (its value depends not only on M and $b - a$, but also on $|\lambda|$), the computation time may appear to be too large to get the result in reasonable time. We shall illustrate this situation below and show that even linear Volterra equations may be quickly solved by iterations for small $|\lambda|$ and can not be solved (at least by described method) for larger values of $|\lambda|$.

As we already mentioned, in case when we have no guess about the value of M we simply set $n_{\min} = 1$ and accomplish iterations with hope that V will become contractive before reaching the prescribed value $n_{\max}$ and at some $n < n_{\max}$ the equality (10) will be fulfilled just leading to breaking iterations and declaring the success.

From (16) it follows that the estimate of convergence rate is defined by condition

$$|y(x) - y_n(x)| \leq \frac{\alpha}{n} |y(x) - y_{n-1}(x)|, \quad n \geq n_{\min}. \tag{18}$$

Thus iterative processes for Volterra equations in general converge faster then the Fredholm ones.

Naturally above mentioned aspects should be considered when constructing our algorithm. Thus for Volterra kind equations, if the value M is known, we first estimate $n_{\min}$ according to (14) and then estimate $n_{\max}$ according to (17).

2.5. Linear equations of both Fredholm and Volterra kind

There is no need to reformulate the problem for linear equations. We just set $K(x,t,y) = K(x,t)\cdot y$ and use the general algorithm without any changes the only difference being that in linear case the value of Lipschitz constant may be easily estimated numerically, e.g.

$$M = M_1 = \max_{x,t\in[a,b]} |K(x,t)| \quad \text{or} \quad M = M_2 = \frac{1}{b-a}\left[\int_a^b \left(\int_a^b K^2(x,t)\,dt\right) dx\right]^{1/2} \tag{19}$$

by that $M_2 \le M_1$.

3. Numerical algorithm

The Banach fixed-point theorem presented in the preceding subsections contains most of the desirable properties of a numerical method. Summarizing we may formulate our strategy of calculations in the following way.

1. If nonlinear equation of both Fredholm and Volterra kind should be solved and we have no guess about the value of of the Lipschitz constant M of integral operator, we set $\beta = 0$, $n_{\min} = 1$ and $n_{\max} = N_{\max}$, where $N_{\max}$ is the prescribed upper limit of iterations. The value $N_{\max}$ depends on program realization. In our C++-code we set $N_{\max} = 10$ this value may be increased or decreased if need be. By that the convergency of iterations is unknown in advance so we call such problem to be bad-defined. Convergence analysis is usually difficult for these problems and the approach in solving them will be largely empirical. Thus we accomplish iterations (4) or (15) for $n = 1, \ldots, N_{\max}$ for any initial approximation $y_0(x)$ and an each step check the point-wise stopping criterion (10). If the process appeared to be converged the solution for given x with prescribed tolerance will be received. Despite the absence of *a priori* number of iterations to achieve a prescribed solution accuracy before actual computations such strategy may lead to success (examples will be presented below).

2. If the Lipschitz constant M is estimated by some way (in case of linear equation we set $M = M_2$ according to (19) then the value $\alpha = |\lambda|M(b-a)$ may be defined.

a) For Fredholm equation, if $\alpha \ge 1$, the iterations will not converge, and we stop calculations. Else, we set $\beta = \alpha$, $n_{\min} = 1$ and estimate $n_{\max}$ according to (8)

$$n_{\max} = ceil\left(\ln\frac{\tau(1-\alpha)}{d(y_1,y_0)}\cdot\frac{1}{\ln\alpha}\right).$$

The function $ceil(x)$ returns the smallest integer greater than or equal to x.

b) For Volterra equation we estimate $n_{\min}$ according to (14). The calculation rule is represented by pseudo-code

$$n_{\min} := 1; \quad q := \alpha; \quad \text{while } (q \ge 1)\left\{n_{\min} := n_{\min} + 1; q := q\cdot\frac{\alpha}{n_{\min}};\right\}.$$

Then we estimate $n_{\max}$ according to (17). The calculation rule is represented by pseudo-code

$$n_{\max} := 1; \quad q := \alpha\cdot d(y_1,y_0); \quad \text{while } (q\cdot e^b > \tau)\left\{n_{\max} := n_{\max} + 1; q := q\cdot\frac{\alpha}{n_{\max}};\right\}.$$

Then we calculate $\beta = \alpha^{n_{\min}}/n_{\min}!$.

Under the condition $0 < \beta < 1$, the approximation sequence is well-defined, and it is convergent to the unique solution of the problem. Furthermore, we know the convergence rate (see (7) or (18)), we have an *a priori* number of iterations $n_{\max}$ to achieve a prescribed solution accuracy before actual computations take place, and we also have the iteration number $n_{\min}$ starting from which the pointwise stopping criterion (10) should be checked. Thus at $0 < \beta < 1$ the solution may be received by successive approximations and we call such problem to be well-defined.

3. Now we outline our algorithm to calculate the result of nth iteration. For Fredholm kind equation define the integral operator

$$\widehat{U}(x, f(t)) = \int_a^b K(x, t, f(t))\, dt, \tag{20}$$

where K is the kernel function, f is given function and x is a parameter. Then

$$y_n(x) = F(x) + \lambda \int_a^b K(x, t_n, y_{n-1}(t_n))\, dt_n = F(x) + \lambda \widehat{U}(x, y_{n-1}(t_n)). \tag{21}$$

Any numerical algorithm for calculation of quadrature (20) in (21) needs a rule to calculate the integrand at any point $t_n \in [a, b]$. Thus the rule for calculation of $y_{n-1}(t_n)$ is needed and we have

$$y_{n-1}(t_n) = F(t_n) + \lambda \int_a^b K(t_n, t_{n-1}, y_{n-2}(t_{n-1}))\, dt_{n-1} = F(t_n) + \lambda \widehat{U}(t_n, y_{n-2}(t_{n-1}))$$

and

$$y_n(x) = F(x) + \lambda \widehat{U}\Big(x, F(t_n) + \lambda \widehat{U}(t_n, y_{n-2}(t_{n-1}))\Big).$$

Thus at calculations by formula (21) the operator $\widehat{U}$ calls itself recursively and needs the rule for calculation of $y_{n-2}(t_{n-1})$. Thus we have a recursive sequence

$$y_{n-2}(t_{n-1}) = F(t_{n-1}) + \lambda \int_a^b K(t_{n-1}, t_{n-2}, y_{n-3}(t_{n-2}))\, dt_{n-2} = F(t_{n-1}) + \lambda \widehat{U}(t_{n-1}, y_{n-3}(t_{n-3})),$$

$$\dots$$

$$y_1(t_2) = F(t_2) + \lambda \int_a^b K(t_2, t_1, y_0(t_1))\, dt_1 = F(t_2) + \lambda \widehat{U}(t_2, y_0(t_1)). \tag{22}$$

Therefore to calculate the value $y_n(x)$ the descending recursive scheme of depth n is realized which is equivalent to calculation of multidimensional integral of order n.

It may seem that such approach leads to large time wasting because at each iteration step the calculation for previous steps are repeated anew. But because the calculation time grows exponentially with n this time wasting becomes small. In addition by using our approach there is no need to save (and use) the results of previous iterations this speeds up the process considerably. Our numerical experiments show that problems (especially well-defined ones) for which the kernel is not strongly peaked in small regions may be solved quickly.

For linear Fredholm equation and at $y_0(x) = F(x)$ our algorithm is implicitly equivalent to numerical estimation of nth partial sum of Neumann series with iterated kernels (its radius of convergence is at least $1/(M(b - a))$) and equivalently to approximate calculation of solution by the resolvent kernel of the integral equation [Zemyan, 2012].

Note that if Volterra equation should be solved we simply set $b = x$ in formulas (20)–(22) and use the same scheme.

Described strategy dictates the necessity to construct a highly effective integrator to realize the operation (20). Saying "effective" me mean here the minimization of functional calls of integrand K by which the integral may be calculated with prescribed accuracy. Such integrator is considered in the next section.

3.1. Quadrature estimation

Consider the problem to estimate numerically the definite integral in (20). Numerical integration, which is also called quadrature, includes a large number of methods [Press et al., 1992, 2007]. A good integrator should exert some adaptive control over its own progress to achieve some predetermined accuracy in the solution with minimum computational effort. After some trials we managed to construct such method. The problem (20) is reduced to solving the initial value problem for auxiliary function $u(x,t)$:

$$\frac{du}{dt} = K(x,t,f(t)), \quad u(x,a) = 0, \quad t \in [a,b] \quad \Rightarrow \widehat{U}(x,f(t)) = u(x,t=b). \tag{23}$$

Then we solve the problem (23) using the ideas of Runge–Kutta methods for solving the systems of ordinary differential equations (ODE). These methods propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand), and then using the information obtained to match a Taylor series expansion up to some higher order [Hairer et al., 1993]. Advances in Runge–Kutta methods, particularly the development of higher-order methods, have made Runge–Kutta competitive with the other methods in many cases. Runge–Kutta succeeds virtually always; it is usually the fastest method when evaluating K is cheap and the accuracy requirement is not stringent, or in general when moderate accuracy ($\lesssim 10^{-5}$) is required [Press et al., 1992, 2007].

Each Runge–Kutta method can be organized to monitor internal consistency. This allows numerical errors, which are inevitably introduced into the solution, to be controlled by automatic (adaptive) changing of the fundamental stepsize. The most efficient stepsize adjustment algorithm is based on embedded Runge–Kutta formulas, originally invented by Merson and popularized in a method of Fehlberg, who discovered a fifth-order method with six function evaluations where another combination of the six functions gives a fourth-order method. The difference between the two estimates (Δ) can then be used as an estimate of the truncation error to adjust the step-size. Since Fehlberg's original formula, many other embedded Runge-Kutta formulas have been found [Hairer et al., 1993].

In fact our algorithm works faster as it would be if the non-adjusted code for solving the ODE systems of general kind was used. Namely because the righthand side of equation (23) does not contain the value of u we managed to exclude one functional call of K from calculations on each integration step. Thus we set up an efficient fifth-order Runge–Kutta method which needs only five function evaluations to accomplish a step and consists in the the following. Starting from $x = a$ and for given step h we accumulate increments with proper weights and estimate error as difference between fourth- and fifth-order methods

$$u(x,t+h) = u(x,t) + h\sum_{i=1}^{5} b_i k_i + o(h^6), \quad \Delta = h\sum_{i=1}^{5} e_i k_i \sim h^5, \quad k_i = K(x,t+c_i h, f(t+c_i h)). \tag{24}$$

Here the weight constants b_i, c_i, e_i depend on realization of embedded Runge–Kutta scheme.

Moreover further improvements were made. First we tried to use Cash–Karp scheme [Cash, Karp, 1990] recommended in earlier edition of Numerical Recipes [Press et al., 1992]. The results were quite satisfactory. But later we favored in more efficient method found by Dormand and Prince [Press et al., 2007; Hairer et al., 1993] with more effective computational scheme and better error properties. Using

Table 1. Dormand–Prince 5(4) parameters for embedded Runga–Kutta method

i	1	2	3	4	5
b_i	35/384	500/1113	125/192	−2187/6784	11/84
c_i	0	0.3	0.8	8/9	1
e_i	71/57600	−71/16695	71/1920	−17253/339200	71/4200

the information presented in [Press et al., 2007] we obtained the necessary constants which are defined in Table 1.

Not that as $c_1 = 0$ and $c_5 = 1$ we get $k_5 = K(x, t + h)$. Because $K(x, t + h)$ has to be evaluated anyway to start the next step, this costs nothing (unless the step is rejected because the error is too big). This trick is called "first-same-as-last" [Press et al., 2007]. Thus only four function evaluations are needed on each step when a sequence of non-rejected steps (24) is accomplished. As a result being incorporated into iterative process (4) this improvement leads to speeding up twice the solutions of typical integral equations as compared to Cash–Karp scheme.

After calculation (24) we know, at least approximately, what error Δ is, and we need to consider how to keep it within desired bounds. We calculate

$$err_{\max} = \frac{|\Delta|}{scale}, \quad scale = atol + rtol \cdot \max(|U(x,t)|, |U(x,t-h)|).$$

Here *atol* and *rtol* are optional absolute and relative error tolerances, respectively. We use $atol = rtol = \tau_U$ in our code, where τ_U is the user-defined tolerance for integration. This choice is the safest in general but may be changed if need be.

Now we accomplish the adaptive step size control according to the strategy described in [Press et al., 2007]. The new step is defined according to the rule

$$h_{\text{new}} = S \cdot (err_{\max})^{-0.2} \tag{25}$$

where $S = 0.9$ is a safety factor (it is introduced because our error estimates are not exact, but only accurate to the leading order in h).

Now we are to decide if used value of h is acceptable. If $err_{\max} \le 1$ we accept the step h and continue calculation (24) for new point $x \to x + h$ with $h \to h_{\text{new}} \ge h$, otherwise reject it and repeat calculation (24) at the same point x but with $h \to h_{\text{new}} < h$. The process is repeated until the final point $x + h = b$ will be reached.

Moreover, experience shows that it is not wise to let the step size increase or decrease too fast, and not to let the step size increase at all if the previous step was rejected. In our algorithm, the step size cannot increase by more than a factor of 10 nor decrease by more than a factor of 5 in a single step [Press et al., 1992, 2007; Hairer et al., 1993].

The examples of program implementation of similar algorithms may be found in [Press et al., 1992] (C-code) or in [Press et al., 2007] (C++-code) and [Hairer et al., 1993] (FORTRAN-code).

Finally if Volterra equation should be solved we simply set $b = x$ in (20) and (23) and use the same scheme.

4. Numerical examples

For presentation we have implemented a compact C++-code of our algorithm. We present the series of solutions of test problems that will illustrate the possibilities and the performance of our algorithm. All illustrative examples were taken from different references, described in Introduction. One may see that our approach can do without that rather complicated methods described in cited references.

The presentation of testing calculations is organized as follows.

1. The integral equation is given in the form (1) or (11) and the exact solution $y(x)$ is presented. The free term $F(x)$ for given $y(x)$ is defined so that $y(x)$ is an analytical solution of given equation at any λ. Thus

$$F(x) = y(x) - \lambda \int_a^{b,x} K(x, t, y(t))\, dt$$

and is always used as initial approximation, thus $y_0(x) = F(x)$.

2. Note that multidimensional integration is likely to be very slow if we try for too much accuracy. Because the multiple integration smoothes the truncation error of quadrature to large extent we are almost certainly increase the tolerance in quadrature rule to 10^{-4} or bigger. We are also decrease a tolerance for solution to avoid a lot of waiting around for an answer. Thus we set the user-defined values $\tau_U = 0.0001$ (tolerance for integration) and $\tau = 0.0001$ (tolerance for solution) and they are the same for all examples.

3. Parameters defining the calculational process such as estimates of contractivity constant β and the values $n_{\min}$ and $n_{\max}$ are presented.

4. The point-wise approximation errors with respect to known exact solution $y(x)$ are calculated as

$$\varepsilon(x) = \frac{|y(x) - \widetilde{y}(x)|}{\tau(1 + \max(|y(x)|, |\widetilde{y}(x)|))} \tag{26}$$

where $\widetilde{y}(x)$ is the approximate solution received by our program.

5. To estimate the calculational time the approximate number $N(x)$ of functional calls for kernel function K is presented in normalized form as $N/10^n$ where n is the number of iterations. This parameter is also point-wise.

6. Some results of self-consistency checks which characterize the numerical solution are presented. These are the value of point-to-point convergence error δ defined by (10) and two global errors, defining the quality of numerical solution by itself. They are coarsely estimated by trapezoidal rule based on the set $(x_i, \widetilde{y}_i)$ presented in output table, namely

$$E_2 = \left(\sum_{i=0}^{k} \left(\widetilde{y}(x_i) - F(x_i) - \widetilde{U}_i \right)^2 \right)^{1/2}, \qquad E_\infty = \max_{i=0,\ldots,k} \left(|\widetilde{y}(x_i) - F(x_i) - \widetilde{U}_i| \right) \tag{27}$$

where k is the number of intervals on segment $[a, b]$. For Fredholm kind equation the values $\widetilde{U}_i$ are defined as

$$\widetilde{U}_i = h \sum_{j=0}^{k} w_i K(x_i, x_j, \widetilde{y}(x_j)), \quad h = \frac{b-a}{k}, \quad w_0 = w_k = 0.5, \quad w_i = 1, \quad i = 1, \ldots, k-1. \tag{28}$$

For Volterra kind equation the upper limit in sum (28) should be changed $k \to i$.

4.1. Linear equations

We start with linear equations because for this case the algorithm is well-defined and no uncertainties appear. There exists the robust procedure to set all necessary parameters such as $n_{\min}$, $n_{\max}$ and contractivity constant β.

Problem 1. Consider a standard test problem for Fredholm linear equation with the Runge kernel. It arises in electrostatics where it is called Love's equation [Atkinson, Shampine, 2008]. The kernel function has a peak along $x = t$ when the positive parameter c is small. The equation is

$$y(x) = F(x) + \lambda \int_0^1 \frac{cy(t)}{c^2 + (x-t)^2}\, dt, \quad x \in [0, 1].$$

The function $F(x)$ is defined as

$$F(x) = 1 + x^2 - \lambda \left[c + cx \ln \frac{c^2 + (1-x)^2}{c^2 + x^2} + (1 - c^2 + x^2) \left(\arctan \frac{1-x}{c} + \arctan \frac{x}{c} \right) \right].$$

This equation has analytical solution $y(x) = 1 + x^2$ at any λ. For the numerical results we set $\lambda = -0.1$ and solve equation for $c = 0.5$. The computational results are presented in Table 2.

Table 2. Solution of Problem 1

$\lambda = -0.1$, $c = 0.5$, $\beta = 0.1488$, $n_{\min} = 1$, $n_{\max} = 5$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	1	0.99995	0.26	5	9.13	3.4
0.1	1.01	1.0099	0.42	5	9.09	3.9
0.2	1.04	1.0399	0.33	5	7.28	4.2
0.3	1.09	1.0895	2.5	5	2.48	4.5
0.4	1.16	1.16	0.12	5	11.1	4.5
0.5	1.25	1.2499	0.36	5	6.68	4.4
0.6	1.36	1.36	0.044	5	10.6	4.1
0.7	1.49	1.49	0.17	5	10.7	3.8
0.8	1.64	1.64	0.16	5	11.2	3.3
0.9	1.81	1.81	0.023	5	6.56	2.8
1	2	2	0.13	5	6.59	2.3
$E_2 = 0.001235$, $E_\infty = 0.0008485$						

In this table and thereafter the first row represents the parameter λ and also estimations of contractivity constant β and the values of $n_{\min}$, $n_{\max}$. The column x is the knot values of argument, y and $\widetilde{y}$ are the exact and approximate solutions at given x respectively. Other columns visualize the point-wise details of the process of computation. The value ε is defined by (26), n is the number of iterations which was used to get solution at given x, N is normalized number of kernel function calls, δ is the point-to-point convergence error defined by (10). The last row shows the global errors (27).

4.2. Volterra kind nonlinear equations

For Volterra equations our solver demonstrates high efficiency since the method of successive approximations copes most successfully with this type of equations.

Problem 2. Consider Volterra–Hammerstein nonlinear equation [Javadi, 2014]

$$y(x) = x - \frac{\lambda}{2} x^3 \left(\frac{1}{2} \sin 2x + x \right) + \lambda \int_0^x x^3 \cos(t) \cos(y(t))\, dt, \quad x \in [0, 1].$$

This equation has analytical solution $y(x) = x$ at any λ. The problem is well-defined because we may use the estimation (9) and set $M = 1$. For the numerical results we solve equation for $\lambda = -1$. The computational results are presented in Table 3.

Table 3. Solution of Problem 2

$\lambda = -1$, $\beta = 0.641$, $n_{\min} = 1$, $n_{\max} = 8$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	0	0	0	1	0	0
0.1	0.1	0.1	4e-12	2	0.3	2e-05
0.2	0.2	0.2	2e-08	2	0.3	0.007
0.3	0.3	0.3	6e-06	2	0.3	0.24
0.4	0.4	0.4	1e-06	3	0.135	0.003
0.5	0.5	0.5	2e-05	3	0.135	0.05
0.6	0.6	0.6	0.0003	3	0.135	0.53
0.7	0.7	0.7	0.0003	4	0.056	0.041
0.8	0.8	0.8	0.0013	4	0.056	0.35
0.9	0.9	0.9	0.0088	5	0.0231	0.07
1	1	0.99999	0.031	5	0.0231	0.4
$E_2 = 0.001112$, $E_\infty = 0.0007643$						

Note that convergency rate is large and the point-wise stopping criterion (10) at $n < n_{\max}$ was used for all x.

Problem 3. Consider Volterra–Hammerstein nonlinear equation [Javadi, 2014]

$$y(x) = \cos x - \lambda\left(\frac{1}{2} - \frac{1}{6}\cos 2x - \frac{1}{3}\cos x\right) + \lambda\int_0^x \sin(x-t)y^2(t)\,dt, \quad x \in [0,1].$$

This equation has analytical solution $y(x) = \cos x$ at any λ. The problem is bad-defined and we set $\beta = 0$, $n_{\min} = 1$, and $n_{\max} = 10$. For the numerical results we solve equation for $\lambda = -3$ (Table 4).

Table 4. Solution of Problem 3

$\lambda = -3$, $\beta = 0$, $n_{\min} = 1$, $n_{\max} = 10$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	1	1	0	1	0	0
0.1	0.995	0.995	2e-6	3	0.135	0.01
0.2	0.98007	0.98007	3e-4	3	0.135	0.48
0.3	0.95534	0.95534	1e-4	4	0.056	0.05
0.4	0.92106	0.92106	6e-4	4	0.056	0.49
0.5	0.87758	0.87758	0.03	5	0.0226	0.03
0.6	0.82534	0.82532	0.09	5	0.0226	0.21
0.7	0.76484	0.7648	0.24	5	0.0226	0.97
0.8	0.69671	0.69661	0.59	6	0.0091	0.05
0.9	0.62161	0.62139	1.3	6	0.0091	0.2
1	0.5403	0.53987	2.8	6	0.0109	0.7
$E_2 = 0.0008594$, $E_\infty = 0.0004246$						

Problem 4. Consider Volterra–Hammerstein nonlinear equation [Javadi, 2014]

$$y(x) = e^{-x} - \lambda\left(\frac{3}{2} - e^{-x} - \frac{1}{2}e^{-2x}\right) + \lambda\int_0^x \Big(y(t) + y^2(t)\Big)\,dt, \quad x \in [0,1].$$

Table 5. Solution of Problem 4

$\lambda = 1, \beta = 0, n_{\min} = 1, n_{\max} = 10$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	1	1	0	1	0	0
0.1	0.90484	0.90484	3e-3	5	0.023	0.55
0.2	0.81873	0.81873	1e-3	7	0.004	0.11
0.3	0.74082	0.74082	1e-3	8	0.002	0.16
0.4	0.67032	0.67032	1e-3	9	6e-4	0.15
0.5	0.60653	0.60653	7e-3	9	6e-4	0.84
0.6	0.54881	0.54881	9e-3	10	2e-4	0.52
0.7	0.49659	0.49659	0.01	10	2e-4	1.8
0.8	0.44933	0.44933	0.01	10	2e-4	5.2
0.9	0.40657	0.40656	0.1	10	2e-4	13
1	0.36788	0.36783	0.36	10	2e-4	28
$E_2 = 0.2001, E_\infty = 0.2$						

This equation has analytical solution $y(x) = e^{-x}$ at any λ. The problem is bad-defined and we set $\beta = 0$, $n_{\min} = 1$, and $n_{\max} = 10$. For the numerical results we solve equation for $\lambda = 1$. The computational results are presented in Table 5.

At large $x > 0.6$ the point-wize convergency was not reached ($\delta > 1$) but nevertheless the received solution is quite satisfactory.

4.3. Fredholm kind nonlinear equations

It is well known that convergence difficulties do arise, more so for Fredholm equations than for equations of the Volterra type.

First we consider an example that we found to be the most popular in literature.

Problem 5. The Fredholm–Hammerstein nonlinear equation is [Biazar, Ghazvini, 2008; Maleknejad et al., 2010; Hashmi et al., 2012; Saberi-Nadjafi, Heidari, 2010]

$$y(x) = \sin(\pi x) + \frac{1}{5}\int_0^1 \cos(\pi x)\sin(\pi t)y^3\,dt, \quad x \in [0, 1].$$

This equation has analytical solution

$$y(x) = \sin(\pi x) + \frac{20 - \sqrt{391}}{3}\cos(\pi x).$$

The problem is bad-defined and we set $\beta = 0$, $n_{\min} = 1$, and $n_{\max} = 10$. The computational results are presented in Table 6.

Problem 6. Consider Fredholm–Hammerstein nonlinear equation [Erfanian et al., 2014]

$$y(x) = x - \frac{x^2}{12} + \frac{1}{2}\int_0^1 x^2 t^3 \arctan y(t)\,dt, \quad x \in [0, 1].$$

This equation has analytical solution $y(x) = x$. The problem is well-defined because we may use the estimation (9) and set $M = 1$. The computational results are presented in Table 7.

Table 6. Solution of Problem 5

$\lambda = 0.2, \beta = 0, n_{\min} = 1, n_{\max} = 10$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	0.075427	0.075378	0.45	4	56.9	0.18
0.1	0.38075	0.38072	0.25	4	56.9	0.094
0.2	0.64881	0.64881	0.00028	3	20.7	0.91
0.3	0.85335	0.85336	0.031	3	21.2	0.48
0.4	0.97436	0.9745	0.71	4	40.4	0.071
0.5	1	1	2.3e-14	1	0.5	0
0.6	0.92775	0.92761	0.72	4	40.4	0.072
0.7	0.76468	0.76468	0.032	3	21.2	0.51
0.8	0.52676	0.52676	0.0003	3	20.7	0.98
0.9	0.23728	0.23732	0.28	4	56.9	0.11
1	-0.075427	-0.075378	0.45	4	56.9	0.18
$E_2 = 0.0002147, E_\infty = 0.0001391$						

Table 7. Solution of Problem 6

$\lambda = 0.5, \beta = 0.5, n_{\min} = 1, n_{\max} = 11$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	0	0	0	1	0.5	0
0.1	0.1	0.1	0.0043	3	0.19	0.17
0.2	0.2	0.2	0.016	3	0.19	0.64
0.3	0.3	0.30001	0.039	4	0.097	0.064
0.4	0.4	0.40001	0.065	4	0.097	0.11
0.5	0.5	0.50001	0.095	4	0.097	0.15
0.6	0.6	0.60002	0.13	4	0.097	0.21
0.7	0.7	0.70003	0.16	4	0.097	0.27
0.8	0.8	0.80004	0.2	4	0.097	0.33
0.9	0.9	0.90005	0.24	4	0.097	0.39
1	1	1.0001	0.28	4	0.097	0.46
$E_2 = 0.001807, E_\infty = 0.001135$						

Problem 7. Consider Fredholm–Urysohn nonlinear equation [Lee, 1997; Maleknejad et al., 2013]

$$y(x) = F(x) + \lambda \int_0^1 \frac{dt}{2 + x + y(t)}, \quad x \in [0, 1],$$

$$F(x) = \cos(0.3\pi x) - \frac{20\lambda}{3\pi\sqrt{3 + 4x + x^2}} \arctan\left(\frac{1 + x}{\sqrt{3 + 4x + x^2}} \cdot \tan\frac{3\pi}{20}\right).$$

Equation has analytical solution $y(x) = \cos(0.3\pi x)$ at any λ.

For $\lambda = 1$ this equation has been solved in [Lee, 1997] by three algorithms based on multigrid method and also in [Maleknejad et al., 2013] by double exponential sinc Nyström method.

For the numerical results we solve equation for $\lambda = 1$. The problem is well-defined because we may use the estimation (9) and set $M = 0.25$. The computational results are presented in Table 8.

Table 8. Solution of Problem 7

$\lambda = 1, \beta = 0.25, n_{\min} = 1, n_{\max} = 7$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	1	1	0.001	6	0.024	0.14
0.1	0.99556	0.99556	0.001	6	0.024	0.13
0.2	0.98229	0.98229	0.001	6	0.024	0.13
0.3	0.96029	0.96029	0.001	6	0.024	0.12
0.4	0.92978	0.92978	0.001	6	0.024	0.11
0.5	0.89101	0.89101	0.001	6	0.024	0.11
0.6	0.84433	0.84433	0.001	6	0.024	0.11
0.7	0.79016	0.79015	0.001	6	0.024	0.1
0.8	0.72897	0.72897	0.001	6	0.024	0.1
0.9	0.66131	0.66131	0.001	6	0.024	0.1
1	0.58779	0.58779	0.001	6	0.024	0.1
$E_2 = 0.0002339, E_\infty = 9.517e-05$						

Table 9. Solution of Problem 8

$\lambda = 0.6, \beta = 0.389711, n_{\min} = 1, n_{\max} = 10$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	0	0.00011724	1.2	10	0.00153	0.75
0.1	0.1	0.10012	1.1	10	0.00153	0.68
0.2	0.2	0.20012	0.98	10	0.00153	0.62
0.3	0.3	0.30012	0.9	10	0.00153	0.58
0.4	0.4	0.40012	0.84	10	0.00153	0.54
0.5	0.5	0.50012	0.78	10	0.00153	0.5
0.6	0.6	0.60012	0.73	10	0.00153	0.47
0.7	0.7	0.70012	0.69	10	0.00153	0.44
0.8	0.8	0.80012	0.65	10	0.00153	0.42
0.9	0.9	0.90012	0.62	10	0.00153	0.39
1	1	1.0001	0.59	10	0.00153	0.37
$E_2 = 0.001334, E_\infty = 0.0004023$						

Problem 8. Consider Fredholm–Hammerstein nonlinear equation [Alipanah, Dehghan, 2007]

$$y(x) = x - \lambda\frac{\pi}{4} + \lambda\int_0^1 \frac{dt}{1+y^2(t)}, \quad x \in [0,1].$$

This equation has analytical solution $y(x) = x$ at any λ. The problem is well-defined because we may use the estimation (9) and set $M = 3\sqrt{3}/8$. For the numerical results we solve equation for $\lambda = 0.6$. The computational results are presented in Table 9.

Note that at $\beta > 0.5$ the convergency rate is small.

Problem 9. Consider Fredholm–Hammerstein nonlinear equation [Borzabadi et al., 2006; Borzabadi, Fard, 2009]

$$y(x) = x - \lambda\left[(x+b-1)e^b - x + 1\right] + \lambda\int_0^b \left((x+t)e^{y(t)}dt, \quad x \in [0,b].\right.$$

Table 10. Solution of Problem 9

$b = 1$, $\lambda = -0.15$, $\beta = 0$, $n_{\min} = 1$, $n_{\max} = 10$						
x	y	$\widetilde{y}$	ε	n	$N/10^n$	δ
0	0	2.4e-06	0.024	10	0.00192	0.75
0.1	0.1	0.1	0.024	10	0.00192	0.79
0.2	0.2	0.2	0.025	10	0.00192	0.82
0.3	0.3	0.3	0.025	10	0.00192	0.85
0.4	0.4	0.4	0.025	10	0.00192	0.87
0.5	0.5	0.5	0.025	10	0.00192	0.89
0.6	0.6	0.6	0.026	10	0.00192	0.91
0.7	0.7	0.7	0.026	10	0.00192	0.92
0.8	0.8	0.8	0.026	10	0.00192	0.94
0.9	0.9	0.9	0.026	10	0.00192	0.95
1	1	1	0.026	10	0.00192	0.96
$E_2 = 0.002223$, $E_\infty = 0.0007761$						

This equation has analytical solution $y(x) = x$ at any λ. The problem is bad-defined and we set $\beta = 0$, $n_{\min} = 1$ and $n_{\max} = 10$.

The equation was solved for $\lambda = -0.15$, $b = 1$. The computational results are presented in Table 10.

5. Discussion

Our analysis of recent surveys has revealed that the majority of numerical methods for the solution of integral equations use one of two main techniques, or their combination is used. Either the unknown function is expanded as a combination of basis set functions and the resulting coefficients found, or the integral is discretized using quadrature formulas. The latter results in converting the integral equation to a system of equations for the solution at the quadrature abscissas which can become very difficult to solve if nonlinearities arise. Most approaches also suffer the lack of generality which is a highly desirable feature of an integral equation solver since many physical problems can be reduced to integral equations of one form or another [Pisarenko, Maglevanny, 1990].

In contrast, our method based on successive quadratures has proved to be very powerful, especially in connection with nonlinear equations. Within the theoretical limits of this method, computational scheme is designed to allow the solution of as wide a range of problems as possible. Our investigations confirmed that the successive quadratures approach when implemented with the Runge–Kutta based quadrature scheme is capable of delivering very accurate results with relatively few function evaluations and arithmetic manipulations.

In addition, there is no problem with computer memory required.

6. Conclusions

The article proposes technique based on direct iterative method, which holds no restrictions for nonlinear integral equations. New implementations of successive approximations method appear. A powerful routine for the solution of a wide range of nonlinear one-dimensional integral equations has resulted with the inclusion of a new efficient method for calculating quadratures with error control.

This method has two advantages that encourages to use it. In one hand, its convergence is guaranteed by fixed-point theorems and so the corresponding algorithm is robust. On the other hand,

this algorithm is very simple to apply and to make a program realization. Numerical results are verified that the method employed in the paper is valid. It is worthy to note that this method can be used as a robust accurate algorithm for solving linear and nonlinear integral equations arising in physics and other fields of applied mathematics.

We note also that with minimal changes in integration procedure this algorithm may be adapted for solving the singular equations with integrable peculiarities.

Список литературы (References)

Alipanah A., Dehghan M. Numerical solution of the nonlinear Fredholm integral equations by positive definite functions // Appl. Math. Comput. — 2007. — Vol. 190. — P. 1754–1761.

Atkinson K., Potra F. Projection and iterated projection methods for nonlinear integral equations // SIAM J. Numer. Anal. — 1987. — Vol. 24. — P. 1352–1373.

Atkinson K. A survey of numerical methods for solving nonlinear integral equations // Journal of Integral Equations and Applications. — 1992. — Vol. 4. — P. 15–46.

Atkinson K. The Numerical Solution of Integral Equations of the Second Kind. — Cambridge University Press, 1997.

Atkinson K. E., Shampine L. F. Algorithm 876: Solving Fredholm integral equations of the second kind in MATLAB // ACM Trans. Math. Softw. — 2008. — Vol. 34, No. 4. — P. 1–20.

Atkinson K., Han W. Theoretical numerical analysis, a functional analysis frameworks. — 3rd ed. // Texts in applied mathematics. — Vol. 39. — New York: Springer-Verlag, 2009.

Babolian E., Shahsavaran A. Numerical solution of nonlinear Fredholm integral equations of the second kind using Haar wavelets // J. Comput. Appl. Math. — 2009. — Vol. 225. — P. 87–95.

Behiry S. H., Abd-Elmonem R. A., Gomaa A. M. Discrete Adomian decomposition solution of nonlinear Fredholm integral equation // Ain. Shams. Eng. J. — 2010. — Vol. 1. — P. 97–101.

Biazar J., Ghazvini H. Numerical solution of special non-linear Fredholm integral equation by HPM // Appl. Math. Comput. — 2008. — Vol. 195. — P. 681–687.

Borzabadi A. H., Kamyad A. V., Mehne H. H. A different approach for solving the nonlinear Fredholm integral equations of the second kind // Appl. Math. Comput. — 2006. — Vol. 173. — P. 724–735.

Borzabadi A. H., Fard O. S. A numerical scheme for a class of nonlinear Fredholm integral equations of the second kind // J. Comput. Appl. Math. — 2009. — Vol. 232. — P. 449–454.

Cash J. R., Karp A. H. A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Vaying Right-Hand Sides // ACM Transactions on Mathematical Software. — 1990. — Vol. 16. — P. 201–222.

Cattani C., Kudreykom A. Harmonic wavelet method towards solution of the Fredholm type integral equations of the second kind // Appl. Math. Comput. — 2010. — Vol. 215. — P. 4164–4171.

Erfanian M., Gachpazan M., Beiglo H. Rationalized haar wavelet bases to approximate solution of nonlinear Volterra–Fredholm–Hammerstein integral equations with error analysis // J. Appl. Computat. Math. — 2014. — Vol. 4. — P. 195–199.

Hairer E., Nørsett S. P., Wanner G. Solving ordinary differential equations I. Nonstiff problems (2nd Edition). — New York: Springer, 1993.

Hashmi M. S., Khan N., Iqbal S. Optimal homotopy asymptotic method for solving nonlinear Fredholm integral equations of second kind // Appl. Math. Comput. — 2012. — Vol. 218. — P. 10982–10989.

Farnoosh R., Ebrahimi M. Monte Carlo method for solving Fredholm integral equations of the second kind // Appl. Math. Comput. — 2008. — Vol. 195. — P. 309–315.

Javadi Sh. A Modification in successive approximation method for solving nonlinear Volterra–Hammerstein integral equations of the second kind // Journal of Mathematical Extension. — 2014. — Vol. 8, No. 1. — P. 69–86.

Krasnosel'skii M. A. Topological methods in the theory of nonlinear integral equations. — New York: Pergamon Press, 1964.

Krasnosel'skii M., Zabreyko P. Geometric methods of nonlinear analysis. — Berlin Heidelberg: Springer-Verlag, 1984.

Lee H. Multigrid method for nonlinear integral equations // J. Appl. Math. Comput. — 1997. — Vol. 4. — P. 427–440.

Linz P. Analytical and numerical methods for Volterra integral equations. — SIAM, Philadelphia, PA, 1985.

Maleknejad K., Almasieh H., Roodaki M. Triangular functions (TF) method for the solution of nonlinear Volterra–Fredholm integral equations // Commun. Nonlinear. Sci. — 2010. — Vol. 15. — P. 3293–3298.

Maleknejad K., Nedaiasl K., Moradi B. Double exponential sinc Nyström solution of the Urysohn integral equations // Proceedings of the World Congress on Engineering. — London, U.K., 2013.

Pisarenko V. G., Maglevanny I. An approximate solution to the problem of neutron transfer in an inhomogeneous medium by the method of boundary integral equations // USSR Academy of Sciences, Dalnevost. Separation, Cupid. complex. Research institutes. — Prep. — Blagoveshchensk: AmurKNII, 1990 (1991). — 20 p.

Polyanin A. D., Manzhirov A. V. Handbook of Integral Equations. — CRC Press, 2008.

Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. Numerical Recipes in C: The Art of Scientific Computing (2nd Edition). — New York: Cambridge University Press, 1992.

Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. Numerical Recipes: The Art of Scientific Computing (3rd ed.). — New York: Cambridge University Press, 2007.

Saberi-Nadjafi J., Heidari M. Solving nonlinear integral equations in the Urysohn form by Newton–Kantorovich-quadrature method // Computers and Mathematics with Applications. — 2010. — Vol. 60, No. 7. — P. 2058–2065.

Zemyan S. M. The Classical Theory of Integral equations: A Concise Treatment. — Boston: Birkhauser, 2012.

Appendix. Program implementation

In order our method would be more clear we present the design of the C++-code for formulas (20)–(25).

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define NMAX 10//Maximal possible interation number in this routine
//Global data. The shortest way to realize the inter-program interface
static double aG,bG,lambdaG,xG,zG,TOLforINT_G,
(*FG)(double x),(*y0G)(double x),
(*KG)(double x,double t,double y);
```

```
static int VolterraG,LinearEqG;// control parameters
static const double  // Dormand-Prince 5(4) parameters
c2=0.3,c3=0.8,c4=8./9.,b1=35./384.,b2=500./1113.,b3=125./192.,
b4=-2187./6784.,b5=11./84.,e1=71./57600.,e2=-71./16695.,e3=71./1920.,
e4=-17253./339200.,e5=71./4200.;
```

Here follows the implementation of the fifth-order Dormand–Prince step (24). The routine `Stepper` computes $u(x, t + h)$ and the error Δ.

```
//The routine Stepper to take a Dormand-Prince Runge-Kutta step:
static double Stepper(double U,double dUdt,double t,double h,
double (*f)(double t),double x,double *Delta,double *Knew
                     )
//Given value for U and its derivative dUdt known at t,
//use the fifth-order Dormand-Prince Runge-Kutta method
//to advance the integral over an interval h
//and return the incremented variables as Uout.
//Also return an estimate of the
//local truncation error Delta in Uout
//using the embedded fourth-order method.
//The routine f(t) defines the nonlinearity of Uryson kernel at t.
//Kg() is a pointer to kernel function
{double t1,k2,k3,k4,k5,Uout;
t1=t+c2*h;k2=KG(x,t1,f(t1));t1=t+c3*h;k3=KG(x,t1,f(t1));
t1=t+c4*h;k4=KG(x,t1,f(t1));t1=t+h;k5=KG(x,t1,f(t1));*Knew=k5;
//Accumulate increments with proper weights.
Uout=U+h*(b1*dUdt+b2*k2+b3*k3+b4*k4+b5*k5);
//Estimate error as difference between fourth and fifth order methods.
*Delta=h*(e1*dUdt+e2*k2+e3*k3+e4*k4+e5*k5);return Uout;
}
```

The routine `INTEGRATOR` calculates the value of (24). It attempts a step h, invokes the controller to decide whether to accept the step or try again with a smaller step-size.

```
static inline double max1(double x,double y){return x<y?y:x;}
static inline double min1(double x,double y){return x<y?x:y;}
static const int MAXSTP=32000;
static const double minscale=0.2,maxscale=10,alpha=0.2,safe=0.9;
/////////////////
double INTEGRATOR(
double (* f)(double t),
//This function defines the nonlinearity of Urysohn kernel
double a,double b,//Integration limits
double x,//Parameter
double TOL//Local tolerance for integration
               )
// Numerical integration.
//This is an embedded Runge-Kutta method of order (4)5 due
// to Dormand-Prince (with step-size control).
{if(a==b)return 0;int reject=0,nstp=0;
double hmin=1.e-15,U,h,t,hnext,scale,Kt,errmax,htemp,Delta,Utemp,Knew,
atol=TOL,rtol=TOL;
//This general-purpose choice can be modified if need be.
t=a;U=0;h=b-a;Kt=KG(x,t,f(t));
while(++nstp<=MAXSTP){//Take at most MAXSTP steps.
for(;;){Utemp=Stepper(U,Kt,t,h,f,x,&Delta,&Knew);//Take a step.
```

```
scale=atol+rtol*max1(fabs(U),fabs(Utemp));
//Step per step scaling to monitor accuracy.
errmax=fabs(Delta/scale);//Evaluate accuracy to required tolerance.
if(errmax<=1.0)break;//Step succeeded. Start from the next point.
reject=1;//Step rejected. Try again with reduced h set
//Truncation error too large, reduce stepsize.
scale=max1(safe*pow(errmax,-alpha),minscale);h*=scale;
if(t+h==t){printf("Stepsize underflow in INTEGRATOR\n");exit(0);}
       }//end for(;;)
//Step succeeded. Return result or compute size of next step.
t+=h;U=Utemp;Kt=Knew;
if((t-b)*(b-a)>=0.0)//Are we done?
return U;//Normal exit.
//Compute size of next step.
if(errmax==0.)scale=maxscale;
else {
scale=safe*pow(errmax,-alpha);
//Ensure minscale < hnext/h < maxscale.
if(scale<minscale)scale=minscale;if(scale>maxscale)scale=maxscale;
     }
//Don't let step increase if last one was rejected.
if(reject)hnext=h*min1(scale,1.);else hnext=h*scale;
if(fabs(hnext)<=hmin)
{printf("Step size too small in INTEGRATOR\n");exit(0);}
h=hnext;reject=0;
if((t+h-b)*(t+h-a)>0.0)h=b-t;//If stepsize can overshoot, decrease.
       }//end while
printf("Too many steps in routine INTEGRATOR\n");exit(0);
}
```

The routine `ITER` accomplishes an iteration.

```
double ITER(double (* f)(double t),
//Defines the nonlinearity of Urysohn kernel
double a,double b,//Integration limits
double x,//Parameter
double TOLforINT//Local tolerance for integration
){return FG(x)+lambdaG*INTEGRATOR(f,a,VolterraG?x:b,x,TOLforINT);}
```

The routine `ITER_n` calculates the *n*th iteration as the descending recursive sequence of quadratures.

```
//The imbedded sequence of integrands. No more then 10 iterations
double ITER_1(double x){return ITER(y0G,aG,bG,x,TOLforINT_G);}
double ITER_2(double x){return ITER(ITER_1,aG,bG,x,TOLforINT_G);}
double ITER_3(double x){return ITER(ITER_2,aG,bG,x,TOLforINT_G);}
double ITER_4(double x){return ITER(ITER_3,aG,bG,x,TOLforINT_G);}
double ITER_5(double x){return ITER(ITER_4,aG,bG,x,TOLforINT_G);}
double ITER_6(double x){return ITER(ITER_5,aG,bG,x,TOLforINT_G);}
double ITER_7(double x){return ITER(ITER_6,aG,bG,x,TOLforINT_G);}
double ITER_8(double x){return ITER(ITER_7,aG,bG,x,TOLforINT_G);}
double ITER_9(double x){return ITER(ITER_8,aG,bG,x,TOLforINT_G);}
double ITER_10(double x){return ITER(ITER_9,aG,bG,x,TOLforINT_G);}
////////////////////////////////////
static double ITER_n(int n,double x)
//The number of possible iterations is restricted to 10.
```

```
//This sequence of functions
//imitates the descending recursion procedure to calculate
//the multiple integrals
//with the recursion depth being just prior to 10.
//The user may add additional lines
//in this code defining in advance the functions ITER_11(double x), ...
//according to the code presented above.
{switch(n){
case 1:return ITER_1(x);case 2:return ITER_2(x);
case 3:return ITER_3(x);case 4:return ITER_4(x);
case 5:return ITER_5(x);case 6:return ITER_6(x);
case 7:return ITER_7(x);case 8:return ITER_8(x);
case 9:return ITER_9(x);case 10:return ITER_10(x);
default:printf("n_{max} exceeded\n");exit(0);
        }//end switch
}
```

The routine `LipschitzConstant` calculates $M = M_2$ according to (19). We use the trapezoidal rule for coarse estimation of double integral.

```
double LipschitzConstant(void)
{int i,j,k=500;//k is the number of intervals on [a,b]
double M2,h=(bG-aG)/k,p,s1=0,s2=0,s3=0,s0=pow(KG(aG,aG,1),2)
+pow(KG(aG,bG,1),2)+pow(KG(bG,aG,1),2)+pow(KG(bG,bG,1),2);
for(i=1;i<k;i++){p=KG(aG+i*h,aG,1);s1+=p*p;p=KG(aG+i*h,bG,1);s1+=p*p;}
for(j=1;j<k;j++){p=KG(aG,aG+j*h,1);s2+=p*p;p=KG(bG,aG+j*h,1);s2+=p*p;}
for(j=1;j<k;j++)for(i=1;i<k;i++){p=KG(aG+j*h,aG+i*h,1);s3+=p*p;}
s3+=0.25*s0+0.5*(s1+s2);M2=h*sqrt(s3)/(bG-aG);return M2;
}
```

The routine `IntegralEquation` prepares the program system to calculations: sets all global parameters, estimates $n_{\min}$ and $n_{\max}$ and returns the contractivity constant β.

```
double IntegralEquation(
//Picard's iterative method driver.
//Stores intermediate results in global variables
//Returns contraction parameter
int LinearEq,//Solve linear or nonlinear integral equation
int Volterra,//Solve equation of Volterra or Fredholm kind
double (*K)(double x,double t,double y),//Kernel function
double (*F)(double x),//Right-hand side function
double (*y0)(double x),//Initial approximation function
double a,double b,//Integration limits
double lambda,//Parameter
double TOLforINT,//Tolerance for integration
double TOL,//Tolerance for stopping criterium for iterations
int *nmin,//Iteration number to start trials
int *nmax,//User-defined maximal number of iterations, <=10
double M//User-defined estimation of Lipschitz constant
)
{//Set the global values
VolterraG=Volterra;LinearEqG=LinearEq;KG=K;FG=F;y0G=y0;aG=a;bG=b;
lambdaG=lambda;TOLforINT_G=TOLforINT;
*nmin=1;if(M==0&&LinearEq)M=LipschitzConstant();if(M==0.)return 0.;
int i,k=500;//k is the number of intervals on [a,b]
double al=fabs(lambda)*M*(b-a),norm=-1,h=(b-a)/k,q,hi;
```

```
for(i=0;i<=k;i++){hi=a+i*h;norm=max1(norm,fabs(ITER_n(1,hi)-y0(hi)));}
if(!Volterra){//Fredholm kind  equation
//Estimate nmax
if(al<1.)*nmax=
(int) ceil(log(TOL*(1-al)/norm)/log(al));//Contractive operator
return al;
            }
//Estimate nmax for Volterra kind equation
q=al*norm;*nmax=1;
while(q*exp(b)>TOL){(*nmax)++;q*=(al/(*nmax));}
//Estimate nmin and contractivity constant for Volterra kind equation
q=al;while(q>=1){(*nmin)++;q*=(al/(*nmin));}
return q;
}
```

The routine `GetSolution` calculates the solution at given value of x and should be used in practical calculations after routine `IntegralEquation` was called.

```
double GetSolution(
//At point x returns the solution of Fredholm integral equation
//of second kind received by Picard iterative method
double x,//Argument
double TOL,//Tolerance for stopping criterium for iterations
int nmin,//Starting iteration number
int nmax,//Maximal number of iterations, <=10
int *n,//Number of iterations used
double *DEL//Point-to-point convergence error
)
{//Set the global values
xG=x;if(nmin>NMAX)nmin=NMAX;
if(nmax>NMAX)nmax=NMAX;//No more then NMAX iterations
double atol=TOL,rtol=TOL,scale,yn,yn1=(nmin<2)?y0G(x):ITER_n(nmin-1,x);
for((*n)=nmin;(*n)<=nmax;(*n)++){yn=ITER_n(*n,x);
//Relative error with previous iteration
scale=0.1*(atol+rtol*max1(fabs(yn),fabs(yn1)));
*DEL=fabs((yn-yn1)/scale);//Evaluate accuracy to required tolerance.
if(*DEL<1.)break;yn1=yn;
                          }//n
if((*n)>nmax)(*n)--;return yn;
}
```

The above-presented code is self-sufficient, no external routines or data bases are needed. Anyone may compile our routines (or rewrite the C++-code by other programming language) and get rather universal and effective solver for solution of large class of integral equations.

We now give some fragments from notional calling programs, to clarify the use of our solver. This fragment shows how one may solve Problem 1 (see Sect. 4).

```
double lambda=-0.1;// Define parameter
double c=0.5;
double K(double x,double t,double y)//Kernel function
{double xt=x-t;return c*y/(c*c+xt*xt);}
double F(double x)
{return 1+x*x-lambda*(c+c*x*log((c*c+(x-1)*(x-1))/(c*c+x*x))
+(1-c*c+x*x)*(atan((1-x)/c)+atan(x/c)));
}
double y0(double x){return F(x);}//Initial approximation function
```

```
//////////////////////////////////////
void main()
//This sample program shows how to solve
//a linear  Fredholm equation of the second kind
//using the IntegralEquation and GetSolution routines.
{// Define flags to determine the type of equation
int Volterra=0,LinearEq=1;
double M=0,a=0,b=M_PI/2;//Lipschitz constant and integration limits
double TOLforINT=1.e-4,TOL=1.e-4;//Tolerances
int nmin,nmax,n,k=10,i,j;double h=(b-a)/k,x,tildey,DEL,
beta=IntegralEquation(LinearEq,Volterra,K,F,y0,a,b,::lambda,TOLforINT,
TOL,&nmin,&nmax,M);
if(beta>=1){
printf("This problem can not be solved by given program\n");exit(0);}
for(i=0;i<=k;i++){x=a+i*h;
tildey=GetSolution(x,TOL,nmin,nmax,&n,&DEL);
printf("x=%-.5g    \\tilde y=%-.5g\n",x,tildey);
               }//i
}
```