

**Библиотека численных методов StudLib
для PascalABC.Net 3.2**

Ростов-на-Дону
2017

Описывается состав и даются рекомендации по использованию библиотеки численных методов StudLib, реализованной в среде программирования PascalABC.NET 3.2.

Для школьников старших классов, учащихся колледжей и студентов младших курсов вузов.

Содержание

Содержание.....	3
1. Введение	4
2. Система тестирования StudLibTest.....	5
3. Общая информация	6
3.1. Точность представления чисел типа real	6
4. Описание программ	8
4.1. Нахождение корней нелинейных уравнений.....	8
4.1.1. Теория вопроса	8
4.1.2. Изоляция корней уравнения $y(x)=0$ на заданном интервале табличным методом (RootsIsolation)	9
4.1.3. Нахождение действительного нуля функции на интервале изоляции (ZeroIn).....	11
4.2. Статистическая обработка данных, заданных в табличном виде	12
4.3. Интерполяция, дифференцирование и аппроксимация данных, заданных в табличном виде	13
4.3.1. Интерполяция табличной функции кубическим сплайном	13
4.4. Операции с полиномами	15
4.4.1. Описание класса <i>Polynom</i>	15
4.4.2. Полиномиальная арифметика	17
4.4.3. Операции с полиномами. Корни полиномов. Полиномиальная аппроксимация.....	18
4.4.4. Специальные полиномы.....	19
4.5. Линейная алгебра (операции с векторами и матрицами, решение систем линейных уравнений).....	20
4.6. Решение систем дифференциальных уравнений	21
4.7. Вычисление определенных интегралов.....	22
4.8. Поиск минимума функций	23
4.9. Задачи оптимизации.....	24
Литература	25

1. Введение

StudLib – свободно распространяемая библиотека (далее - пакет) программ, реализованная в системе программирования PascalABC.NET 3.2 и поставляющаяся вместе с ней в исходном коде (файл StudLib.pas). Для работы с пакетом модуль StudLib.pas следует загрузить и откомпилировать.

В пакете находятся программы, реализующие различные численные методы, а также вспомогательные программы, сопутствующие типы данных и отдельные классы.

С помощью пакета StudLib можно решать задачи из следующих областей:

- нахождение корней нелинейных уравнений;
- статистическая обработка данных, заданных в табличном виде;
- интерполяция, дифференцирование и аппроксимация данных, заданных в табличном виде;
- операции с полиномами;
- линейная алгебра (операции с векторами и матрицами, решение систем линейных уравнений);
- решение систем дифференциальных уравнений;
- вычисление определенных интегралов;
- поиск минимума функций одного и многих переменных;
- задачи оптимизации.

Часть программ переведена в паскаль на уровне исходного текста из существующих пакетов прикладных программ, таких как SSPLIB (на языке Фортран) или опубликованных в литературе. В этом случае подробная ссылка на источник приведена в тексте программы. Другая часть написана автором на основе алгоритмов, приведенных в различных источниках и в этом случае ссылка на источник также дается в тексте программы.

2. Система тестирования StudLibTest

После установки или обновления PascalABC.NET 3.2 рекомендуется выполнить тестирование пакета при помощи модуля StudLibTest.pas.

Тестирование заключается в решении ряда контрольных заданий и сличении полученных результатов с эталонами. Проведение тестирования является хорошим подтверждением работоспособности установленной версии.

Каждый модуль пакета тестируется на наборе тестовых примеров и при непрохождении теста с помощью Assert выдается сообщение с указанием полученных и ожидаемых результатов, позволяющее локализовать место ошибки. Несмотря на то, что весь пакет тщательно тестируется, допускается возможность непрохождения тестов в программах, использующих случайные числа. В таких случаях полезно попытаться выполнить тестирование несколько раз, чтобы убедиться в наличии четкой ошибки.

В ходе тестирования по мере прохождения тестов программных единиц на монитор выводится протокол.

Набор тестовых заданий содержит достаточное количество примеров, ознакомление с которыми может оказаться полезным для лучшего понимания работы с пакетом.

3. Общая информация

3.1. Точность представления чисел типа real

В PascalABC.NET тип real базируется на представлении данных System.Double платформы Microsoft .NET. Данное типа real занимает 8 байт при длине мантиссы 52 разряда, что обеспечивает точность не более 17 десятичных знаков.

PascalABC.NET предоставляет несколько констант платформы .NET, связанных с типом real:

- real.MinValue - минимальное значение, примерно равное $-1.7976931348623157E+308$;
- real.MaxValue - максимальное значение, примерно равное $1.7976931348623157E+308$;
- real.Epsilon - минимальное положительное число, отличное от нуля («машинная точность»), которое выводится с несколько странным значением $4.94065645841247E-324$ (к этому мы еще вернемся ниже);
- NaN – «Not a Number» («не число»), возникает при делении $0/0$, вычислении функций с недопустимыми аргументами и т.п. Возникнув, имеет тенденцию распространяться на всю правую часть оператора присваивания, в связи с чем при программировании рекомендуется принимать меры к изоляции NaN при помощи проверки `IsNaN(x)`, возвращающей true для $x=NaN$;
- real.NegativeInfinity – «отрицательная бесконечность», возникающая при делении отрицательной величины на ноль. Проверяется при помощи `IsNegativeInfinity(x)`;
- real.PositiveInfinity - «положительная бесконечность», возникающая при делении положительной величины на ноль. Проверяется при помощи `IsPositiveInfinity(x)`.

Когда знак бесконечности не имеет значения, можно воспользоваться проверкой `IsInfinity(x)`.

Вернемся к рассмотрению машинной точности. Практическое использование константы `real.Epsilon` приводит к «удивительным» (в нехорошем смысле этого слова) результатам, чем и объясняется решение отказаться использования этой константы при написании пакета StudLib.

В [1] предлагается считать точностью (машинным эпсилон) такую минимальную величину ϵ , для которой $1 + \epsilon > 1$

При попытке воспользоваться `real.Epsilon` были получены совершенно неудовлетворительные результаты, в частности,

$$1.0 + \text{real.Epsilon} = 1.0 + \text{real.Epsilon} * 1e100$$

Понятно, что это делает невозможными сравнения точности с величинами ϵ , 2ϵ , ... 1000ϵ и т.д, поэтому в программах машинная точность определяется следующим образом:

```
var eps:=1.0;
```

```
while eps+1.0>1.0 do eps*=0.5;
```

В этом случае найденная машинная точность оказалась равной $1.11022302462516e-16$ ($7\text{FFFFFFFFFFFFFFFF}_{16}$), что и ожидалось.

4. Описание программ

4.1. Нахождение корней нелинейных уравнений

4.1.1. Теория вопроса

Пусть задана некоторая функция $y=F(x)$. Требуется отыскать одно или более значений x , для которых $F(x)=0$. В этом случае все такие x будут называться корнями уравнения $F(x)=0$.

Теория утверждает, что если $x \in [a;b]$ и функция $F(x)$ на интервале $[a;b]$ меняет знак ровно один раз, то внутри этого интервала найдется отрезок $[\alpha;\beta]$ длины, не превышающей некоторого значения, на котором $F(x)$ также меняет знак и с точностью этот интервал можно считать корнем уравнения $F(x)=0$. Отрезок $[a;b]$ называется интервалом изоляции корня и далее предполагается, что $F(a)$ и $F(b)$ имеют разные знаки, либо $F(a) = 0$, $F(b) \neq 0$, либо $F(a) \neq 0$, $F(b) = 0$.

Почему вместо точного значения корня уравнения мы говорим о некотором интервале $[\alpha;\beta]$ с длиной, не превышающей заданную точность ε ? Все дело в дискретности (и вытекающей из этого точности) представления чисел типа *real*.

Решение задачи на практике состоит из нескольких шагов. На первом шаге определяются интервалы изоляции корней уравнения, а на последующих для каждого интервала изоляции с заданной точностью вычисляется очередной корень.

Одним из самых простых и надежных приемов отделения корня является табличный метод. Для совокупности равноотстоящих точек на оси x вычисляются значения $y(x)$ до тех пор, пока не будет выявлен интервал изоляции.

Другой способ отделения корней основан на методе Монте-Карло. На некотором «разумном» интервале случайным образом заданных значений x вычисляются значения функции $y(x)$ и запо-

минаются те точки x_0 , в которых значение $y(x_0)$ наиболее близко к нулю. Затем делается шаг, например, в сторону уменьшения x , т.е. полагаем $x_1=x_0-h$, и если значение функции $y(x_1)$ также уменьшается, делается следующий шаг в этом же направлении, пока функция не изменит знак. Если при первоначальном шаге функция увеличила значение, то делаем шаг удвоенной длины в обратном направлении, приходя в точку $x_1=x_0+h$ и ведем поиск в новом направлении. Это способ, несмотря на большую, чем предыдущий сложность, в некоторых случаях может быстрее приводить к нужному результату.

4.1.2. Изоляция корней уравнения $y(x)=0$ на заданном интервале табличным методом (RootsIsolation)

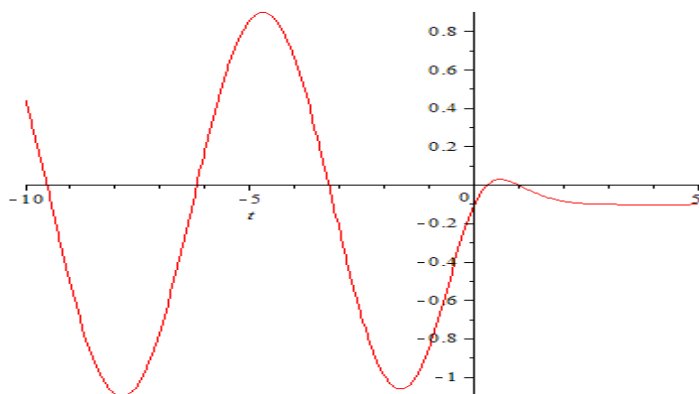
Может применяться для получения интервалов изоляции произвольного количества корней произвольной функции одной переменной. Заданный интервал просмотра $[a;b]$ последовательно просматривается с шагом h . Если корень уравнения попадает на границу интервала b , интервал изоляции выбирается равным $[b-h/2;b+h/2]$. Правильный выбор шага h имеет важное значение.

Рассмотрим пример нахождения интервалов изоляции уравнения, график которого представлен далее.

$$\frac{\sin(t)}{1 + (e^t)^2} - 1 = 0$$

Конечно, если имеется график, интервалы изоляции можно оценить по нему. Но в данном случае график представлен чтобы показать, как выбор слишком большого шага h приводит к ошибкам в решении.

Как видно из графика, при отрицательных значениях аргумента корни отстоят друг от друга примерно на 3, а при положительных - меньше чем на 1. Следовательно, разумно будет выбрать шаг $h < 1$, например, 0.5.



Пусть интервал поиска корней составит $[-10;5]$.

```
var f:real->real:=t->sin(t)/(1+Sqr(Exp(t)))-0.1;
```

```
var (a,b,h):=(-10,5,0.5);
```

```
Writeln(RootsIsolation(f,a,b,h));
```

Получаем решение:

```
[(-10,-9.5),(-6.5,-6),(-3.5,-3),(0,0.5),(1,1.5)]
```

Это правильный результат, потому что корни уравнения приблизительно равны -9.52495 , -6.18307 , -3.24191 , 0.27789 , 1.00272 .

Попробуем задать шаг, который будет в данных условиях неприемлемым, например $h=2$.

Получаем решение:

```
[(-10,-8),(-8,-6),(-4,-2)]
```

и мы потеряли два интервала изоляции.

Наконец, совсем большой шаг $h=6$ приводит к еще более катастрофическим результатам: $[(-4,2)]$ – мы теряем четыре из пяти интервалов изоляции.

4.1.3. Нахождение действительного нуля функции на интервале изоляции (Zeroin)

Zeroin – один из лучших имеющихся машинных алгоритмов, сочетающих безотказность бисекции с асимптотической скоростью метода секущих для случая гладких функций [1]. В пакет включена функция Zeroin, переведенная с фортрана в систему программирования PascalABC.NET 3.2.

Предполагается без проверки, что интервал изоляции корня определен, в противном случае пользоваться Zeroin некорректно.

Типичное обращение имеет вид:

```
var root:=Zeroin(a,b,f,tol);
```

Здесь a, b – интервал изоляции корня [a;b] функции f(x), a tol – величина интервала неопределенности решения.

Рассмотрим пример решения уравнения

$$f(x) = x^3 - 2x - 5 = 0$$

x	-1	0	1	2	3	4
f(x)	-4	-5	-6	-1	16	51

Интервал изоляции корня находится на отрезке [2;3]. Значение tol положим равным 10^{-12} .

```
var f:real->real:=x->x*(x*x-2)-5;
```

```
var root:=Zeroin(2,3,f,1e-12);
```

Полная программа решения этого уравнения может выглядеть так:

```
uses StudLib;
```

```
begin
```

```
  Writeln(Zeroin(2,3,x->x*(x*x-2)-5,1e-12));
```

```
end.
```

4.2. Статистическая обработка данных, заданных в табличном виде

4.3. Интерполяция, дифференцирование и аппроксимация данных, заданных в табличном виде

4.3.1. Интерполяция табличной функции кубическим сплайном

Задача одномерной интерполяции набора n точек $P(x, y)$ с вещественными координатами сводится к построению некоторой функции $F(x)$, для которой $F(x_i) = y_i$ для всех n точек и при этом в промежутках между точками функция принимает некие «разумные значения» [1].

Считается, что если функция $F(x)$ гладкая и вычисленные по ней значения не превышают допустимой ошибки, задача имеет удовлетворительное решение.

Если набор точек $P(x, y)$ не зашумлен ошибками, то интерполяция гладкой функцией уместна. В противном случае используются приёмы, нейтрализующие шум.

Конечно, можно попытаться провести через n точек полином степени $n-1$, но теория доказывает, что практически это оказывается очень плохим решением для $n > 10$.

Математики обожают, когда функция может быть дважды дифференцируема и при этом не вырождается в ноль или иную константу, поэтому минимальная степень интерполяционного полинома равна трем и он будет проходить через четыре исходные точки. Кубический полином – это самая гладкая функция, обладающая необходимыми для интерполяции свойствами. Но точек-то обычно куда больше, чем четыре...

С другой стороны, с древних времен известен чертежный инструмент, называемый лекало. Он позволяет гладко соединить множество точек, нанесенных на плоскость. Суть используемого приема в том, чтобы соединять точки линией по две-три, постепенно перемещая лекало вдоль заданных точек. Немецкие чертежники в

особо важных случаях вместо лекал использовали тонкие металлические рейки, называемые «сплайн».

«Скользющие» кубические полиномы также получили название кубических сплайн-функций или просто сплайнов.

Поскольку сплайн – это кубический полином, т.е.

$$y = F(x) = ax^3 + bx^2 + Cx + d,$$

$$y' = F'(x) = 3ax^2 + 2bx + c, \quad y'' = F''(x) = 6ax + 2b,$$

то можно легко найти первую и вторую производную от таблично заданной функции.

В пакете имеется класс *Spline*, позволяющий выполнить интерполяцию кубическим сплайном. Он является результатом переработки фортран-программ SPLINE и SEVAL [1].

Вспомогательный класс *Point* реализует точку с координатами x, y типа *real*. Класс *Spline* в своем свойстве *P* хранит вектор координат исходных точек (узлов интерполяции) класса *Point*.

Можно рекомендовать следующий порядок проведения интерполяции

- создать вектор исходных точек, например, следующим образом:

```
var f:=x->(3*x-8)/(8*x-4.1);
```

```
var pt:=Partition(1.0,10.0,18).Select(x->new Point(x,f(x))).ToArray;
```

- создать объект класса *Spline*, при этом конструктор автоматически вызовет метод *MakeSpline*, вычисляющий коэффициенты сплайна:

```
var Sp:=new Spline(pt);
```

- для получения значения сплайна в нужной точке x использовать метод *Value*:

```
var r:=Sp.Value(x);
```

Метод *Diff* возвращает кортеж из первой и второй производных в указанной точке:

```
var (d1,d2):=Sp.Diff(x);
```

4.4. Операции с полиномами

4.4.1. Описание класса *Polynom*

Обычно полином $u(x)$ принято записывать в порядке убывания степеней, т.е. в виде $u(x) = u_n x^n + \dots + u_1 x + u_0$

В то же время, большинство алгоритмов для работы с полиномами используют обратный порядок записи – в порядке возрастания степеней: $u(x) = u_0 + u_1 x + \dots + u_n x^n$

Именно такой порядок следования коэффициентов полинома используется, например, в пакетах программ SSPLIB фирмы IBM [2] и IMSL фирмы Rogue Software [3]. Исходя из изложенного выше, было принято использовать такой же порядок следования и в пакете StudLib.

Полиномы реализованы в виде класса *Polynom*, имеющего два свойства:

a – динамический массив типа *real* с коэффициентами полинома, расположенными в порядке **возрастания** степени;

n – количество членов полинома.

Для создания полинома используются две формы вызова конструктора класса.

Вызов вида *new Polynom(k)* создает полином с k членами (т.е. степени $k-1$) и все коэффициенты полинома устанавливает в значение 0.0.

Вызов *new Polynom(a₀, a₁, ... a_m)* создаст полином с указанными значениями коэффициентов. Вместо списка коэффициентов можно указать динамический массив типа *real* (примеры в 4.4.2).

Перечислим методы класса *Polynom*:

Value(x) – возвращает значение полинома для аргумента x ;

Copy – создает копию объекта класса *Polynom*;

PDif – возвращает коэффициенты полинома, полученного дифференцированием исходного;

PInt – возвращает коэффициенты полинома, полученного при взятии неопределенного интеграла от исходного;

Print, Println – выводят на монитор коэффициенты полинома в строку через пробел. Println затем обеспечивает перевод вывода на новую строку;

Print(c), Println(c) – вместо пробела используется разделитель c.

Примеры работы с классом *Polynom*

1. $p(x) = 4x^3 + 6.5x^2 - 18$

```
var p:=new Polynom(-18, 0, 6.5, 4);
```

2. Вычислить для $x=-7.16$ значение $u(x) = x^5 + 3.8x^2 - 6x + 2$

```
var u:=(new Polynom(2, -6, 0, 3.8, 0, 1)).Value(-7.16);
```

3. Поместить в $p(x)$ полином $q(x)$

```
var p:=q.Copy;
```

4. Для $t(x) = -2x^4 - 3x^3 + 12x^2 - 7x + 1$ получить

$$p(x) = \int t(x) dx, \quad q(x) = t'(x)$$

```
var t:=new Polynom(1, -7, 12, -3, -2);
```

```
var (p,q):=(t.PInt, t.PDif);
```

```
p.Println; q.Println;
```

Будет получен результат

0 1 -3.5 4 -0.75 -0.4

-7 24 -9 -8

Запишем по этому результату полученные полиномы:

$$p(x) = -0.4x^5 - 0.75x^4 + 4x^3 - 3.5x^2 + x + C,$$

$$q(x) = -8x^3 - 9x^2 + 24x - 7$$

4.4.2. Полиномиальная арифметика

К полиномиальной арифметике отнесены операции сложения, вычитания, умножения и деления полиномов.

Для сложения, вычитания и умножения полиномов, а также для случая, когда один из операндов является константой, соответствующие арифметические операции для класса *Polynom* перегружены, что позволяет записывать арифметические выражения в привычном виде. Перегружены также унарный минус и операция «=».

Пример:

```
var a:=new Polynom(6.5,-4,2.12,1);
var b:=new Polynom(3,0,-3.8);
var c:=new Polynom(ArrGen(5,i->i*i+1.0));
(-c+(a-2*b)*a+11.5*(1-b)).Println;
```

Деление полиномов выполняется посредством расширения DivMod, возвращающего кортеж из двух полиномов – частного и остатка. Вычислим следующее выражение:

$$\frac{2x^6 - x^5 + 12x^3 - 72x^2 + 3}{x^3 + 2x^2 - 1}$$

```
var p:=new Polynom(3,0,-72,12,0,-1,2);
var q:=new Polynom(-1,0,2,1);
var (a,b):=p.DivMod(q);
a.Println; b.Println;
```

Будет получен ответ

```
-6 10 -5 2
-3 10 -65
```

который трактуется как частное $2x^3 - 5x^2 + 10x - 6$ и остаток $-65x^2 + 10x - 3$

Решение выглядит следующим образом

$$\frac{2x^6 - x^5 + 12x^3 - 72x^2 + 3}{x^3 + 2x^2 - 1} = 2x^3 - 5x^2 + 10x - 6 + \frac{-65x^2 + 10x - 3}{x^3 + 2x^2 - 1}$$

4.4.3. Операции с полиномами. Корни полиномов. Полиномиальная аппроксимация.

4.4.4. Специальные полиномы

4.5. Линейная алгебра (операции с векторами и матрицами, решение систем линейных уравнений)

4.6. Решение систем дифференциальных уравнений

4.7. Вычисление определенных интегралов

4.8. Поиск минимума функций

4.9. Задачи оптимизации

Литература

1. Дж.Форсайт, М.Малькольм, К.Моулер. Машинные методы математических вычислений. Пер. с англ. – М.: Мир, 1980.
2. System/360 Scientific Subroutine Package (360A-СМ-03Х) Programmer's Manual: IBM, Technical Publication Department, 112 East Post Road, White Plains, , N.Y. 10601
3. IMSL® Fortran Math Library. Version 7.1.0. Rogue Wave Software, Inc. 5500:Flatiron Parkway, Boulder, CO 80301 USA
4. Мудров А.Е. Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. – Томск: МП «РАСКО», 1991.
5. Шуп Т. Решение инженерных задач на ЭВМ: Практическое руководство. Пер. с англ. – М.: Мир, 1982.