

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»**

**ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ДИНАМИКИ И ПРОЧНОСТИ МАШИН**

КУРСОВАЯ РАБОТА

по интегрированным системам проектирования и анализа, 5 курс, 10 семестр:

**«Автоматизация аналитического решения краевых задач
математической физики на сложных двумерных областях
методом R-функций»**

Выполнил: ст. гр. И-19Б
Гладков С. В.
Руководитель: доктор техн. наук, проф.
Львов Г.И.

Харьков 2004

АННОТАЦИЯ

В работе была реализована программа для автоматизации аналитического решения краевых задач математической физики на сложных двумерных областях методом R-функций с использованием системы компьютерной математики Maple 8.0. Были представлены тестовые примеры решения уравнений Софи-Жермен и Пуассона с однородными краевыми условиями Дирихле. Результаты сравнивались с численным решением на ANSYS 6.0. Был сделан вывод об их корректности.

ABSTRACT

It was done the program for automating of analytical solution of boundary value problems on complicated 2D domains using R-functions method on system of computer mathematics Maple 8.0. Is was given test examples of solving Sofi Zhermen and Poisson equations with homogenous Dirihlet boundary conditions. Results were compared with numerical solutions on ANSYS 6.0. Results are correct.

СОДЕРЖАНИЕ

Введение	4
1 Постановка задачи и математическая модель	5
2 Методика и алгоритм решения	6
3 Описание программы	11
4 Решение тестовых задач	16
4.1 Задача изгиба тонкой пластины	16
4.2 Задача стационарной теплопроводности	18
5 Текст программы	20
Заключение	29
Список источников информации	30

ВВЕДЕНИЕ

Задачи математической физики – такие, как задачи механики деформируемого твердого тела, механики жидкости и газа, электродинамики, теплофизики и другие – обычно сводятся к дифференциальным уравнениям с частными производными, которые нужно интегрировать при соответствующих начальных и краевых условиях. К сожалению, для очень многих таких уравнений трудно, а в большинстве случаев невозможно, получить точное аналитическое решение, поэтому для их решения приходится использовать либо приближенные методы, либо численные.

Во многих случаях задачу интегрирования дифференциального уравнения можно заменить равносильной вариационной задачей, т.е. задачей, состоящей в отыскании функции, сообщающей некоторому функционалу экстремальное или стационарное значение. Методы, позволяющие свести задачу решения дифференциального уравнения к равносильной вариационной проблеме, называют вариационными. Вопросы, связанные с построением функционалов, соответствующих тому или иному дифференциальному уравнению, подробно освещены в литературе.

На практике также используют численные методы решения начально-краевых задач, самые известные из которых – конечно-разностные методы, и метод конечных элементов. Однако следует отметить, что приближенное аналитическое решение по всем параметрам «лучше» численного – его можно свободно дифференцировать, интегрировать, получать решение в любой точке рассматриваемой области без интерполяции/аппроксимации.

Большой проблемой для решения краевых задач вариационными методами была невозможность учета геометрической информации об исследуемом объекте на аналитическом уровне. Эта проблема была решена академиком Рвачевым В.А. и его научной школой. Он предложил метод решения обратной задачи аналитической геометрии – построения единого аналитического выражения по его чертежу. Этот метод в последствии был назван методом R-функций.

В данной работе представлена программа, написанная на языке системы компьютерной математики Maple, служащая для автоматизации получения аналитического решения краевых задач математической физики на сложных двумерных областях вариационно-структурным методом R-функций. Забегая вперед, заметим, что несмотря на то, что мы работаем «в аналитике», без применения ЭВМ (т.е. на бумаге) даже довольно простую задачу построения границы исследуемой области решить очень трудно – просто очень много бумаги исписать придется. А задачу формирования системы Ритца, где требуется вычисление большого количества двойных определенных интегралов, вообще без ЭВМ решить невозможно.

На данном этапе программа протестирована на решении уравнения Пуассона и уравнения Софи-Жермен при однородных краевых условиях Дирихле, что и будет далее показано на примерах.

1 ПОСТАНОВКА ЗАДАЧИ И МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Требуется создать набор процедур для автоматизации аналитического решения уравнения Пуассона и уравнения Софи Жермен при однородных краевых условиях Дирихле в двумерной области. Приведем математическую постановку задачи.

Уравнение Пуассона имеет вид

$$\Delta T(x, y) = \frac{q(x, y)}{\lambda}, \quad (1.1)$$

где $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ – оператор Лапласа; $T(x, y)$ – температура; $q(x, y)$ – мощность внутренних источников; λ – коэффициент теплопроводности. Краевые условия:

$$T(x, y)|_{\partial\Omega} = 0, \quad (1.2)$$

где $\partial\Omega$ – граница рассматриваемой области Ω .

Уравнение Софи Жермен:

$$\Delta^2 u(x, y) = \frac{q(x, y)}{D}, \quad (1.3)$$

где $\Delta^2 \equiv \frac{\partial^4}{\partial x^4} + 2\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial y^4}$ – бигармонический оператор; $u(x, y)$ – нормальный прогиб пластины; $q(x, y)$ – нормальное давление; $D = \frac{E \cdot h^3}{12(1 - \nu^2)}$ – цилиндрическая жесткость пластины. Краевые условия:

$$\begin{aligned} u(x, y)|_{\partial\Omega} &= 0; \\ \frac{\partial u(x, y)}{\partial n} \Big|_{\partial\Omega} &= 0, \end{aligned} \quad (1.4)$$

где $\partial\Omega$ – граница рассматриваемой области Ω ; n – единичная нормаль.

2 МЕТОДИКА И АЛГОРИТМ РЕШЕНИЯ

Поставленную задачу будем решать **вариационно-структурным методом R-функций**. Для этого нужно свести поставленную краевую задачу к эквивалентной задаче на экстремум некоторого функционала. Вопросы, связанные с построением функционалов, соответствующих тому или иному дифференциальному уравнению, подробно освещены в литературе [1,2].

Краевая задача (1.1, 1.2) сводится к эквивалентной вариационной задаче нахождения минимума функционала следующего вида

$$F(T) = \iint_{\Omega} \left[(\vec{\nabla} T(x, y))^2 - 2 \frac{q(x, y)}{\lambda} T(x, y) \right] d\Omega, \quad (2.1)$$

где $\vec{\nabla} \equiv \frac{\partial^2}{\partial x^2} \vec{i} + \frac{\partial^2}{\partial y^2} \vec{j}$ – оператор набла.

А краевая задача (1.3, 1.4) – к вариационной задаче нахождения минимума функционала

$$F(u) = \iint_{\Omega} \left[(\Delta u(x, y))^2 - 2 \frac{q(x, y)}{D} u(x, y) \right] d\Omega. \quad (2.2)$$

Для нахождения функций, дающих значение функционалов (2.1) или (2.2) близкие к минимальным, Ритцем в 1908 году был предложен простой и эффективный метод. Суть его изложим на примере решения задачи (2.1).

Согласно этому методу, искомая функция $T(x, y)$ представляется в виде линейной комбинации

$$T(x, y) \approx T_n(x, y) = \sum_{i=1}^n C_i W_i(x, y), \quad (2.3)$$

где C_i – неопределенные пока числа; $W_i(x, y)$ – так называемые координатные функции, которые должны:

- 1) быть линейно независимыми;
- 2) удовлетворять краевым условиям поставленной задачи;
- 3) быть элементами некоторого полного функционального пространства.

Подставив это выражение в функционал (2.1) и выполнив необходимое дифференцирование, получим

$$F(C_1, C_2, \dots, C_n) = \iint_{\Omega} \left[\left(\sum_{i=1}^n C_i \frac{\partial W_i}{\partial x} \right)^2 + \left(\sum_{i=1}^n C_i \frac{\partial W_i}{\partial y} \right)^2 - 2 \frac{q}{\lambda} \sum_{i=1}^n C_i W_i \right] d\Omega, \quad (2.4)$$

т.е. функционал превратился в функцию n переменных C_1, C_2, \dots, C_n .

Так как нас интересует минимум этой функции, то числа C_1, C_2, \dots, C_n должны удовлетворять системе уравнений $\frac{\partial F}{\partial C_j} = 0$, т.е.

$$2 \iint_{\Omega} \left[\left(\sum_{i=1}^n C_i \frac{\partial W_i}{\partial x} \right)^2 \frac{\partial W_j}{\partial x} + \left(\sum_{i=1}^n C_i \frac{\partial W_i}{\partial y} \right)^2 \frac{\partial W_j}{\partial y} - \frac{q}{\lambda} W_j \right] d\Omega = 0; \quad (2.5)$$

$(j = 1, 2, \dots, n).$

В сокращенном виде ее можно записать как

$$\sum_{i=1}^n C_i a_{ij} = b_j \quad (j = \overline{1, n}), \quad (2.6)$$

где $a_{ij} = a_{ji} = \iint_{\Omega} (\vec{\nabla} W_i(x, y) \cdot \vec{\nabla} W_j(x, y)) d\Omega$; $b_i = \iint_{\Omega} \frac{q(x, y)}{\lambda} W_i(x, y) d\Omega$.

Из линейной системы (2.6), называемой **системой Ритца**, определяются коэффициенты C_1, C_2, \dots, C_n . Функция (2.3) с найденными коэффициентами, представляет собой приближенное решение рассматриваемой задачи. Точность приближения зависит от выбора координатных функций $W_i(x, y)$ и от их числа [1]. Матрица системы Ритца является матрицей Грамма для последовательности функций $\{W_i(x, y)\}_{i=1}^n$ и, так как эти функции линейно независимы, то определитель матрицы Грамма не равен 0 [3]. Таким образом, система (2.6) имеет, и притом единственное, решение.

Аналогично можно получить систему Ритца для задачи (2.2). Она будет иметь вид (2.6), отличными будут лишь коэффициенты $a_{ij} = a_{ji} = \iint_{\Omega} \Delta W_i(x, y) \Delta W_j(x, y) d\Omega$; $b_i = \iint_{\Omega} \frac{q(x, y)}{D} W_i(x, y) d\Omega$.

Одной из основных проблем, возникающих при использовании вариационных методов, при сколько-нибудь сложной форме области, - построение последовательности координатных функций, удовлетворяющих заданным краевым условиям.

Метод R-функций, используемый в данной работе, а также построение структур решения краевых задач, является решением этой проблемы.

Под **структурой решения краевой задачи** понимается пучок функций $T = f(\Phi, \Omega)$, удовлетворяющих заданным краевым условиям при любом выборе функции Φ , называемой неопределенной компонентой структуры решения. Функция $\Omega(x, y) = 0$ является уравнением границы области.

Приведем структуры решения для рассматриваемых задач. Для краевой задачи (1.2) структура решения имеет вид

$$T(x, y) = \Omega(x, y) \cdot \Phi(x, y). \quad (2.7)$$

Очевидно, что какова бы ни была дважды дифференцируемая функция $\Phi(x, y)$, функции из пучка (2.7) будут удовлетворять краевым условиям (1.2).

Для краевой задачи (1.4) структура решения определяется формулой

$$u(x, y) = \Omega(x, y)^2 \cdot \Phi(x, y). \quad (2.8)$$

Видим, что при любом выборе $\Phi(x, y)$ данная структура удовлетворяет краевым условиям (1.4) задачи. Действительно, $u(x, y)|_{\partial\Omega} = 0$, так как $\Omega(x, y)|_{\partial\Omega} = 0$, и

второе условие $\left. \frac{\partial u(x, y)}{\partial n} \right|_{\partial\Omega} = 0$ также выполняется, поскольку

$$\begin{aligned} \frac{\partial u(x, y)}{\partial n} &= 2\Omega(x, y) \frac{\partial \Omega(x, y)}{\partial n} \Phi(x, y) + \Omega(x, y)^2 \Phi(x, y) = \\ &= \Omega(x, y) \left(2 \frac{\partial \Omega(x, y)}{\partial n} \Phi(x, y) + \Omega(x, y) \Phi(x, y) \right) \Big|_{\partial \Omega} = 0. \end{aligned}$$

Также отметим, что функция $\Omega(x, y)$ должна обладать следующими свойствами:

- 1) $\Omega(x, y) = 0, (x, y) \in \partial \Omega$;
- 2) $\Omega(x, y) > 0, (x, y) \in \Omega$;
- 3) $\Omega(x, y) < 0, (x, y) \notin \Omega$.

Методы построения функции $\Omega(x, y)$ для практически произвольной области достаточно подробно изложены в монографиях В.Л. Рвачева[2,4] и базируются на теории R-функций. Следуя теории R-функций для построения уравнения границы некоторой области, необходимо вначале написать логическую формулу (предикатное уравнение) для этой области.

Пусть Θ есть некоторая область на плоскости xOy . Введем функцию

$$\Theta = \Theta(x, y) = \begin{cases} 0, \forall (x, y) \notin \Theta; \\ 1, \forall (x, y) \in \Theta, \end{cases}$$

называемую характеристической или **двузначным предикатом области** (для простоты предикат и соответствующую ему область будем обозначать одной и той же буквой). Пусть Θ_1 и Θ_2 – предикаты для соответствующих областей. Тогда предикат $\Theta_1 \wedge \Theta_2$ определяет пересечение $\Theta_1 \cap \Theta_2$ (\wedge – это конъюнкция, знак логической операции), предикат $\Theta_1 \vee \Theta_2$ – объединение $\Theta_1 \cup \Theta_2$ этих областей (\vee – дизъюнкция, знак логической операции), и $\bar{\Theta}$ – дополнение области Θ . Логические операции $\neg x$, $x_1 \wedge x_2$, $x_1 \vee x_2$ определяются значениями, приведенными в табл. 1.

Таблица 1. Значения введенных логических операций

x_1	x_2	$\neg x_1$	$\neg x_2$	$x_1 \wedge x_2$	$x_1 \vee x_2$
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Всякая логическая формула $Y = f(x_1, x_2, \dots, x_n)$, построенная с помощью символов конъюнкции, дизъюнкции и отрицания, может рассматриваться как функция, аргументы которой выбираются из множества $B_2 = \{0, 1\}$, состоящего из двух чисел 0 и 1 и значения которой также принадлежат множеству B_2 . Такие функции называют булевыми или переключательными. Далее предикаты для множества точек, определяемые некоторым условием (уравнением, неравенством), будем обозначать этим же условием, заключенным в скобки. Например,

предикат множества Θ_1 точек, заполняющих круг единичного радиуса, будем обозначать $\Theta_1 = [F_1 = 1 - x^2 - y^2 \geq 0]$. Аналогично, характеристические функции (предикаты) $\Theta_2 = [F_2 = y \geq 0]$, $\Theta_3 = \left[F_3 = 1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} \geq 0 \right]$, $\Theta_4 = [F_4 = 1 - x - y]$, которые соответствуют верхней полуплоскости, внутренности эллипса и полуплоскости, расположенной слева от прямой $x + y = 1$.

Применяя к множествам точек, определенным с помощью только что приведенных формул, логические операции, можно получить логические формулы для весьма сложных геометрических объектов. При этом Θ_i будем называть **опорными предикатами** для подобластей, составляющих главную область, а F_i – соответствующими им **опорными функциями**.

Предикат области служит базой для ее аналитического описания, т.е. построения функции $\Omega(x, y)$. Доказана теорема[2], суть которой состоит в том, что если для заданной области Θ построена логическая формула

$$\Theta = f(\Theta_1, \Theta_2, \dots, \Theta_m), \quad (2.9)$$

где f – известная булева функция; $\Theta_i = [F_i(x, y) \geq 0]$ – опорные предикаты для подобластей ($i = \overline{1, m}$), то построение функции, описывающей эту область или границу $\Omega(x, y) = f(F_1, F_2, \dots, F_m) \geq 0$, осуществляется формальной заменой символов Θ_i на $F_i(x, y)$, логических операций \neg, \wedge, \vee – соответствующими R-операциями \neg, \wedge^*, \vee^* . При этом функция $\Omega(x, y)$ обладает нужными нам свойствами.

В данной работе используется система R-операций, названная \mathfrak{R}_α . Существуют другие системы, однако для рассматриваемого класса задач данная система вполне подходит. Математически, R-операции в данной системе записываются так:

$$\begin{aligned} z_1 \wedge_\alpha z_2 &= \frac{1}{1 + \alpha} \left[z_1 + z_2 - \sqrt{z_1^2 + z_2^2 - 2\alpha z_1 z_2} \right]; \\ z_1 \vee_\alpha z_2 &= \frac{1}{1 + \alpha} \left[z_1 + z_2 + \sqrt{z_1^2 + z_2^2 - 2\alpha z_1 z_2} \right]; \\ \neg z_1 &= -z_1, \end{aligned} \quad (2.10)$$

где $-1 \leq \alpha \leq 1$ применяется для получения различных дополнительных (например, дифференциальных) свойств полученного выражения. Для рассматриваемого класса задач положим $\alpha = 0$, и будем использовать систему R-операций \mathfrak{R}_0 :

$$\begin{aligned} z_1 \wedge_0 z_2 &= z_1 + z_2 - \sqrt{z_1^2 + z_2^2}; \\ z_1 \vee_0 z_2 &= z_1 + z_2 + \sqrt{z_1^2 + z_2^2}; \\ \neg z_1 &= -z_1. \end{aligned} \quad (2.11)$$

Выбор других систем R-операций зарезервирован для решения других типов задач.

Располагая структурой решения, получим последовательность координатных функций, если неопределенную компоненту представим в виде

$$\Phi(x, y) \approx \sum_{i=1}^n C_i \varphi_i(x, y), \quad (2.12)$$

где $\varphi_i(x, y)$ – известные элементы некоторого функционального пространства, образующие в нем полную последовательность. Например, если $\Phi \in C^k(\Omega \cup \partial\Omega)$, то в качестве $\{\varphi_i(x, y)\}$ можно выбрать степенные полиномы, полиномы Чебышева, сплайны, тригонометрические функции[2] и т.д. В данной работе реализованы степенные полиномы.

Таким образом, искомое решение (2.7) с учетом (2.12) примет вид

$$T(x, y) \approx T_n(x, y) = \sum_{i=1}^n C_i W_i(x, y), \quad (2.13)$$

где координатные функции $W_i(x, y) = \Omega(x, y) \cdot \varphi_i(x, y)$. А искомое решение (2.8) с учетом (2.12) – вид

$$u(x, y) \approx u_n(x, y) = \sum_{i=1}^n C_i W_i(x, y), \quad (2.14)$$

где координатные функции $W_i(x, y) = \Omega(x, y)^2 \cdot \varphi_i(x, y)$.

3 ОПИСАНИЕ ПРОГРАММЫ

Поставленную задачу будем решать в системе компьютерной математики **Maple 8.0**. Система Maple была выбрана потому, что она изначально предоставляет многие математические функции, такие как дифференцирование, одномерное интегрирование, решение СЛАУ, визуализация, встроенные различные типы данных и процедуры для работы с ними[5,6]. Нам же оставалось «дописать» недостающие процедуры и из них, как из кирпичиков, построить программу. Другими словами, мы создаем инструментария для решения подобных задач.

Отметим, что была поставлена цель не создать отдельную программу для решения конкретных уравнений, а наоборот – создать программу, интегрированную в пакет Maple, чтобы пользователь мог изменять ее для своих нужд, а также пользоваться при этом всею мощностью системы Maple. Поэтому созданная программа оформлена в виде отдельного пакета для Maple названного **RFM** (международная аббревиатура метода R-функций). Работа с программой происходит в online-режиме – пользователь вводит команду, Maple на нее реагирует. Еще раз подчеркнем открытость написанной программы, то есть любой пользователь может воспользовавшись уже сделанным инструментарием, и дописав недостающий, решать нужные ему задачи – например, о собственных колебаниях, свободных или вынужденных колебаниях, задачи оптимизации и прочие.

В табл. 2 приведены процедуры, входящие в реализованный пакет RFM. Так как их довольно много, и их параметры либо очевидны, либо понятны из заголовка процедуры в тексте пакета, мы не стали приводить описание параметров.

Таблица 2. Процедуры, входящие в пакет RFM и их описание

Процедуры, связанные с заданием опорных областей¹		
№	Название процедуры	Описание процедуры
1.	nStripX	нормализованное до первого порядка уравнение полосы, параллельной оси X
2.	nStripY	нормализованное до первого порядка уравнение полосы, параллельной оси Y
3.	nCircle	нормализованное до первого порядка уравнение круга
4.	HalfPlane	уравнение полуплоскости через 2 точки
5.	nHalfPlane	нормализованное до первого порядка уравнение полуплоскости через 2 точки
6.	Ellipse	уравнение эллипса
7.	Astroid	уравнение астроида
8.	OvalKassini	уравнение овала Кассини
Процедуры, связанные с R-операциями²		

¹ Пользователь может аналитически задать любую, нужную ему область

² Пользователь может использовать и другие R-системы

9.	SetRSystem	установка используемой R-системы
10.	`&Un`	бинарная R-дизъюнкция
11.	`&In`	бинарная R-конъюнкция
12.	`&Not`	унарное R-отрицание
Дифференциальные операторы и скалярное произведение		
13.	Grad	оператор Гамильтона
14.	Div	оператор дивергенции
15.	Lapl	оператор Лапласа
16.	`&.`	бинарная операция скалярного произведения вектор-функций
Процедуры визуализации		
17.	PlotDomain	изображение построенной области
18.	PlotDomain3d	изображение построенной области в 3D
19.	PlotIntPaths	изображение отрезков интегрирования ³
20.	PlotIntPathsM1	изображение отрезков интегрирования с выделением разными цветами различных применяемых формул интегрирования (разной точности)
21.	PlotSolution	изображение решения – поверхностей уровня и границы области
22.	PlotSolution3d	изображение решения в 3D
Процедуры, связанные с интегрированием и подготовкой к нему		
23.	PrepareForDbInt10	подготавливает область к интегрированию, т.е. создает «сетку»
24.	PrepareForDbInt10M1	подготавливает область к интегрированию с заданием для разных по длине отрезков разных по точности формул интегрирования
25.	DbIntGauss10	вычисление двойного интеграла по области от заданной функции, использует 10-ти точечную формулу интегрирования Гаусса по оси X, а по Y – встроенную одномерную процедуру
26.	DbIntGauss10M1	вычисление двойного интеграла по области от заданной функции, использует 10-ти точечную формулу интегрирования Гаусса по оси X, а по Y – форму Гаусса нужной точности
27.	DbIntGauss10hf	вычисление двойного интеграла по области от заданной функции, использует 10-ти точечную формулу интегрирова-

³ Процедура реализована для отладки процедур интегрирования и подготовки к нему

		ния Гаусса по осям X и Y, использует машинную арифметику ⁴
28.	DbIntGauss10M1hf	вычисление двойного интеграла по области от заданной функции, использует 10-ти точечную формулу интегрирования Гаусса по оси X и формулу нужной точности по оси Y, использует машинную арифметику
29.	IntGauss3	3-х точечная одномерная формула интегрирования Гаусса ⁵
30.	IntGauss3hf	3-х точечная одномерная формула интегрирования Гаусса с использованием машинной арифметики
31.	IntGauss5	5-ти точечная одномерная формула интегрирования Гаусса
32.	IntGauss5hf	5-ти точечная одномерная формула интегрирования Гаусса с использованием машинной арифметики
33.	IntGauss8	8-ми точечная одномерная формула интегрирования Гаусса
34.	IntGauss8hf	8-ми точечная одномерная формула интегрирования Гаусса с использованием машинной арифметики
35.	IntGauss10	10-ти точечная одномерная формула интегрирования
36.	IntGauss10hf	10-ти точечная одномерная формула интегрирования Гаусса с использованием машинной арифметики
37.	IntGaussA	одномерная формула интегрирования Гаусса с произвольным количеством узлов
Процедуры, связанные с отделением и уточнением корней уравнений		
38.	NewtonRaphsonBisection	уточнение корня гибридным методом Ньютона-Рафсона с бисекцией
39.	FindRoots	отделение корня – поиск с равным шагом
Процедуры генерации полных последовательностей функций		
40.	GenPowerPolynoms	генерация степенных полиномов заданной точности ⁶

⁴ Использование машинной арифметики очень сильно ускоряет процесс интегрирования, однако работает не со всеми видами функций, поэтому и были оставлены процедуры без использования ее

⁵ Одномерные формулы интегрирования были реализованы для того, чтобы ускорить, где это возможно, скорость вычисления интегралов, так как встроенная процедура одномерного интегрирования работает очень медленно, так как использует в основном 30-ти точечную формулу Гаусса-Кронрода

⁶ Пользователь может реализовать процедуры для генерации нужных ему последовательностей - полиномов Чебышева, Лежандра, тригонометрических, сплайнов, атомарных функций и пр.

Процедуры, связанные с процессом решения		
41.	GenCoordFunctions	генерирует список координатных функций по заданной полной последовательности и структуре решения
42.	CreateLeftMatrix	вычисляем матрицу системы Ритца
43.	CreateLeftMatrixM1	вычисляем матрицу системы Ритца с разными формулами интегрирования
44.	CreateLeftMatrixHF	вычисляем матрицу системы Ритца с использованием машинной арифметики
45.	CreateLeftMatrixM1hf	вычисляем матрицу системы Ритца с разными формулами интегрирования и использованием машинной арифметики
46.	CreateRightVector	вычисляет вектор правых частей системы Ритца
47.	CreateRightVectorM1	вычисляет вектор правых частей системы Ритца с разными формулами интегрирования
48.	SolveSystem	решает систему Ритца, используя пакет LinearAlgebra
49.	CreateSolution	генерирует решение задачи по решению системы Ритца и координатным функциям

Четкого алгоритма работы программы нет, так как мы создали *инструментарий для построения программы*. Однако можно дать пользователю некоторую рекомендуемую последовательность действий для получения решения:

Шаг 1	<i>Подключение пакета RFM и выбор системы R-операций</i>
	Подключение пакета RFM делается стандартной командой Maple with(<имя пакета>). Выбор системы R-операций делается процедурой №9 из табл. 2.
Шаг 2	<i>Создание границы рассматриваемой области</i>
	В результате этого шага нужно получить аналитическое выражение для границы исследуемой области. Согласно методу R-функций нужно записать выражения для опорных областей используя процедуры №1-8 или самостоятельно в виду нужных аналитических выражений. Далее нужно написать логическую формулу (предикат) области используя процедуры №10-12, а Maple заменяет логические операции на соответствующие R-операции. На этом же шаге лучше всего задать прямоугольник, ограничивающий построенную область в виде списка из 4-х элементов – координат верхней левой и нижней правой точек, и построить графически с помощью процедур №17-18 созданную область.
Шаг 3	<i>Подготовка области к интегрированию</i>
	Так как далее нам придется вычислять большое количество двойных

	<p>интегралов по созданной области, были сделаны процедуры для подготовки области к интегрированию, включающие в себя непосредственно процедуры подготовки к интегрированию (№23-24), и процедуры визуализации отрезков интегрирования (№19-20). Процедуры подготовки к интегрированию формируют сетку интегрирования по обоим направлениям. Они находят границы созданной области по оси Y при заданных координатах X (процедура №39), которая в свою очередь использует процедуру №38 – отыскание корней гибридным методом Ньютона-Рафсона с бисекцией. Эта процедура была сделана потому, что численное решение нелинейных уравнение в Maple реализовано очень плохо.</p>
Шаг 4	<i>Формирование координатных функций</i>
	<p>На этом шаге мы для начала генерируем список, состоящий из степенных полиномов нужной степени с помощью процедуры №40. А затем с помощью процедуры №41 генерируем список координатных функций.</p>
Шаг 5	<i>Формирование системы Рунца</i>
	<p>В программе мы записываем общий член матрицы правых частей a_{ij}. Для этого реализованы дифференциальные операторы градиента, Лапласа, дивергенции и векторные – скалярного произведения (они нужны для задачи Пуассона) – это процедуры №13-16. Далее запускаем процедуру формирования матрицы правых частей системы Рунца – она вычисляет интегралы по созданной области от конкретного значения общего члена (процедуры №25-37). Это пожалуй самая вычислительно-емкая часть программы. Для ускорения этого процесса были разработаны различные процедуры интегрирования – например, использовать для малых участков формулы интегрирования с меньшим числом узлов, или использование машинной арифметики напрямую (процедуры №42-45). Потом мы вводим численные параметры (например, модуль упругости E, толщину пластины h и коэффициент Пуассона ν для задачи изгиба тонкой пластины, или коэффициент теплопроводности для задачи стационарной теплопроводности). Записываем общий член правой части b_i и проводим интегрирование (процедуры №46-47).</p>
Шаг 6	<i>Решение системы Рунца и вывод результатов</i>
	<p>На предыдущем шаге мы получили систему линейных неоднородных уравнений относительно коэффициентов C_i. На этом шаге выполняется решение этой системы (процедура №48). Из полученного вектора решений формируется следующей процедурой №49 непосредственно аналитическое решение. Далее мы можем изобразить графически полученное решение – построить линии уровня процедурой №21 или решение в 3D процедурой №22.</p>

4 РЕШЕНИЕ ТЕСТОВЫХ ЗАДАЧ

4.1 Задача изгиба тонкой пластины

Решить задачу об изгибе пластины сложной формы в плане, изображенной на рис. 1. Геометрические характеристики указаны на рисунке в метрах. Толщина пластины 0.01 м, модуль упругости $2 \cdot 10^{11}$ Па, коэффициент Пуассона 0.3. Краевые условия – жесткая заделка по всей границе. Нагрузка – равномерное давление, взятое равным 1 Па. Математически данная задача представляет из себя уравнение Софи Жермен с однородными краевыми условиями Дирихле.

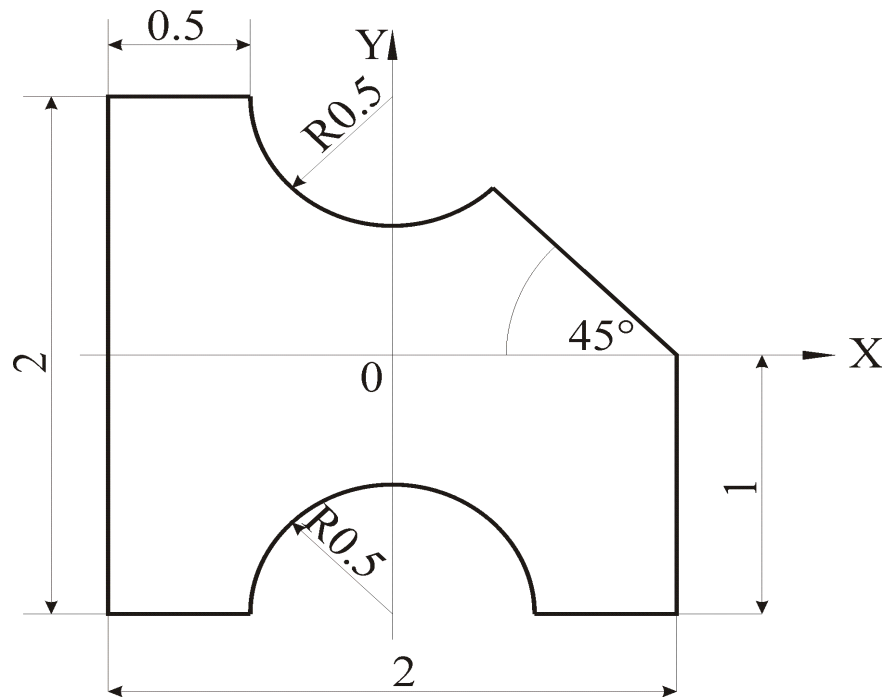


Рисунок 1 – Исследуемая пластина

Отметим, что для данной области, опорными областями будут:

$$\Theta_1 = \left[F1 = \frac{1}{2}(1 - y^2) \geq 0 \right] \text{ – полоса, параллельная оси } X \text{ полушириной } 1;$$

$$\Theta_2 = \left[F2 = \frac{1}{2}(1 - x^2) \geq 0 \right] \text{ – полоса, параллельная оси } Y \text{ полушириной } 1;$$

$$\Theta_3 = \left[F3 = 0.5^2 - x^2 - (y - 1)^2 \geq 0 \right] \text{ – круг с центром в точке } (0,1) \text{ и радиусом } 0.5;$$

$$\Theta_4 = \left[F4 = 0.5^2 - x^2 - (y + 1)^2 \geq 0 \right] \text{ – круг с центром в точке } (0,-1) \text{ и радиусом } 0.5;$$

$$\Theta_5 = \left[F5 = -\frac{-1 + x + y}{\sqrt{3 - 2x - 2y + y^2 + x^2 + 2xy}} \geq 0 \right] \text{ – полуплоскость, задаваемая координатами двух точек: } (0,1) \text{ и } (1,0).$$

Предикат области: $\Theta = (((\Theta_1 \wedge \Theta_2) \wedge \neg\Theta_3) \wedge \neg\Theta_4) \wedge \Theta_5$.

Аналитическое выражение для границы области:

$$\Omega(x, y) = (((F1 \wedge_0 F2) \wedge_0 \neg F3) \wedge_0 \neg F4) \wedge_0 F5.$$

Сравним полученное аналитическое решение с численным, полученным на ANSYS 6.0 (см. рис. 2 и табл. 3). Численное решение на ANSYS было проведено с использованием конечного элемента SHELL93. Это 8-ми узловый конечный элемент с 6-ю степенями свободы в узле: перемещениями и углами поворота в трехмерном пространстве. Решение было получено с использованием 655 конечных элементов.

Таблица 3. Сходимость решения при различном числе координатных функций

Степень аппроксимирующего полинома	Количество координатных функций	Максимальный прогиб, м	Максимальный прогиб на ANSYS, м	Относительная погрешность, %
2	6	0.1471e-6	0.188e-6	21.75
3	10	0.1607e-6		14.52
4	15	0.1803e-6		4.1

Отметим, что полученная относительная погрешность при 15 координатных функциях в 4.1% не превышает требуемой при инженерных расчетах максимальной относительной погрешности в 5%. При увеличении числа координатных функций точность будет увеличиваться.

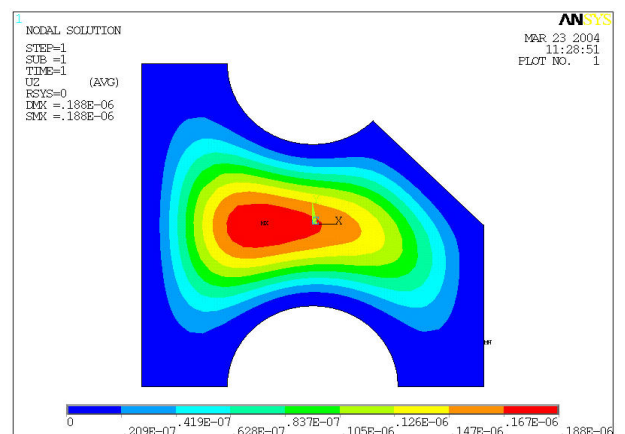
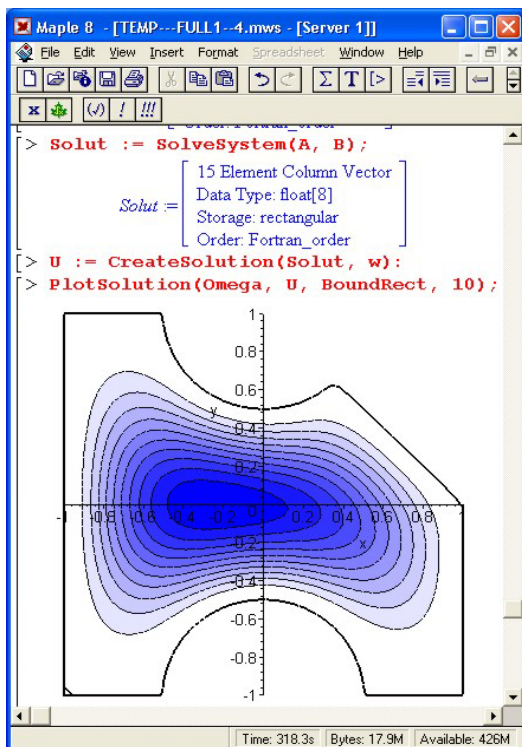


Рисунок 2 – Решение задачи на Maple (слева).
Решение задачи на ANSYS 6.0 (справа)

4.2 Задача стационарной теплопроводности

Решить задачу о распределении температуры в довольно длинном трехмерном теле, сечение которого изображено на рис. 3. Геометрические характеристики указаны на рисунке в метрах. Коэффициент теплопроводности 0.02 Вт/мК . На границе тела известна температура: 0°C . Мощность внутренних источников постоянна, и равна $0.02 \text{ Дж/м}^3\text{с}$. Математически эта задача выражается уравнением Пуассона с однородными условиями Дирихле.

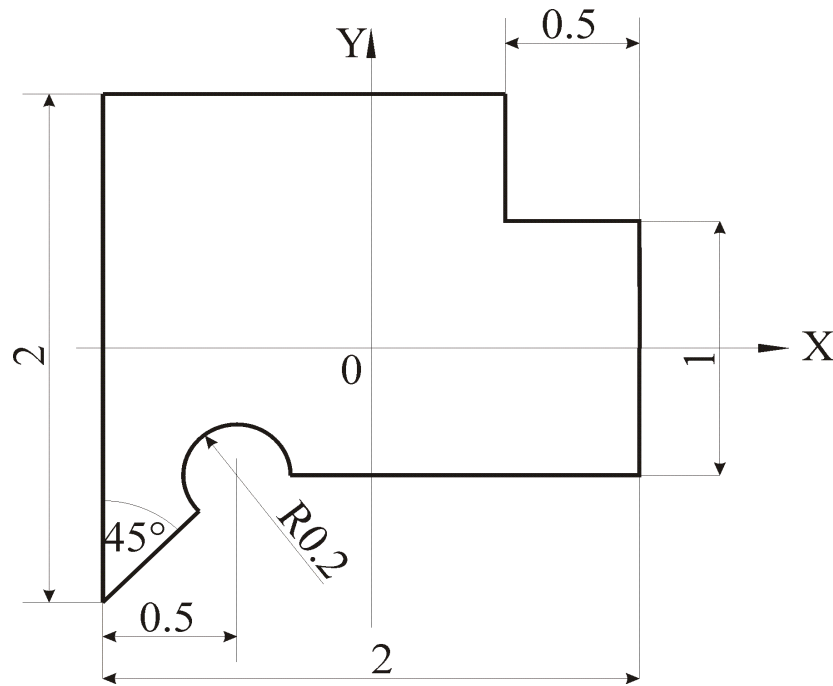


Рисунок 3 – Сечение исследуемого тела

Опорными для данной области будут следующие области:

$$\Theta_1 = \left[F1 = \frac{1}{2}(1 - y^2) \geq 0 \right] - \text{полоса, параллельная оси X полушириной 1;}$$

$$\Theta_2 = \left[F2 = \frac{1}{2}(1 - x^2) \geq 0 \right] - \text{полоса, параллельная оси Y полушириной 1;}$$

$$\Theta_3 = \left[F3 = 0.5^2 - (y - 1)^2 \geq 0 \right] - \text{полоса, параллельная оси X полушириной 0.5 и поднятая относительно оси на 1;}$$

$$\Theta_4 = \left[F4 = 0.5^2 - (x - 1)^2 \geq 0 \right] - \text{полоса, параллельная оси Y полушириной 0.5 и сдвинутая вправо относительно оси на 1;}$$

$$\Theta_5 = \left[F5 = 2.5 \left(0.2^2 - (x + 0.5)^2 - (y + 0.5)^2 \right) \geq 0 \right] - \text{круг радиуса 0.2 с центром в точке } (-0.5, -0.5);$$

$$\Theta_6 = \left[F6 = \frac{2(0.25 - 0.5y)}{\sqrt{y + y^2 + 1.25}} \geq 0 \right] - \text{полуплоскость, проходящая через точки } (0, -0.5) \text{ и } (-0.5, -0.5);$$

$\Theta_7 = \left[F7 = \frac{x - y}{\sqrt{-2xy + x^2 + y^2 + 2}} \geq 0 \right]$ – полуплоскость, проходящая через точки (-0.5,-0.5) и (-1,-1).

Логическая формула всей области:

$$\Theta = (((\Theta_1 \wedge \Theta_2) \wedge \neg(\Theta_3 \wedge \Theta_4)) \wedge \neg\Theta_5) \wedge \neg(\Theta_6 \wedge \Theta_7).$$

В «аналитике» получаем, заменяя логические операции R-операциями системы \mathfrak{R}_0 :

$$\Omega(x, y) = (((F1 \wedge_0 F2) \wedge_0 \neg(F3 \wedge_0 F4)) \wedge_0 \neg F5) \wedge_0 \neg(F6 \wedge_0 F7).$$

Сравним полученное аналитическое решение с численным, полученным на ANSYS 6.0 (см. рис. 4 и табл. 4). Численное решение на ANSYS было проведено с использованием конечного элемента PLANE35. Это 6-ти узловый конечный элемент с 1-й степенью свободы в узле: температурой. Этот элемент предназначен для решения задач тепло- и массообмена. Решение было получено с использованием 983 конечных элементов.

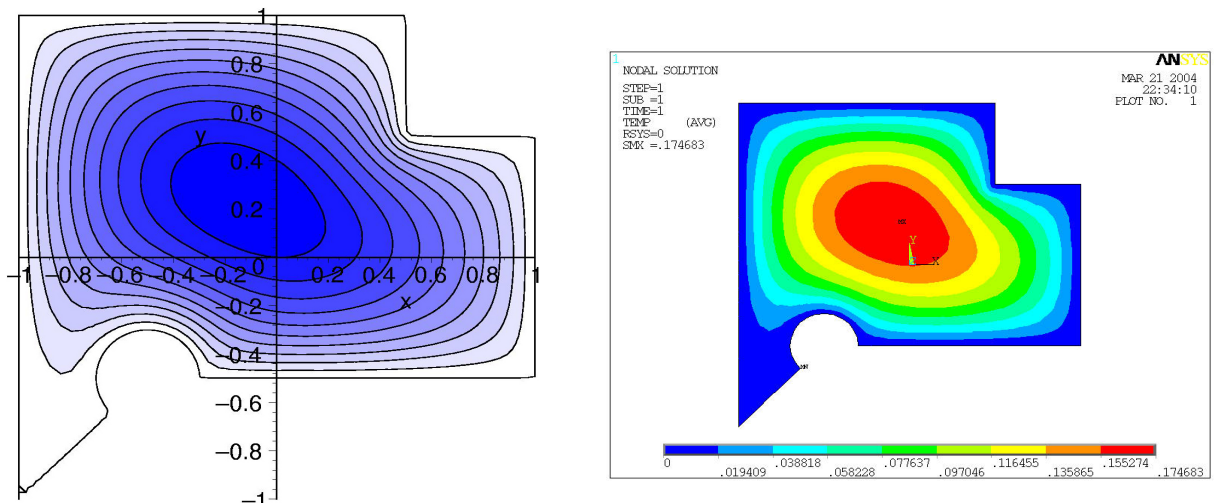


Рисунок 4 – Аналитическое решение на Maple (слева).
Численное решение задачи на ANSYS 6.0 (справа)

Таблица 4. Сходимость решения при различном числе координатных функций

Степень аппроксимирующего полинома	Количество координатных функций	Максимальная температура, °C	Максимальная температура на ANSYS, °C	Относительная погрешность, %
2	6	0.1649	0.1747	5.61
3	10	0.1705		2.4
4	15	0.1736		0.63

Отметим, что полученная относительная погрешность даже при 10 координатных функциях в 2.4% не превышает требуемой при инженерных расчетах максимальной относительной погрешности в 5%. При увеличении числа координатных функций точность будет увеличиваться.

5 ТЕКСТ ПРОГРАММЫ

```

> protect(x,y);
> nStripX := proc(DisplY::numeric,
HalfWidth::numeric)
  global x,y;
  if HalfWidth=0 then error "zero
half width of strip" end if;
  (x,y) -> (HalfWidth^2 - (y -
DisplY)^2)/2/HalfWidth;
end proc;
> nStripY := proc(DisplX::numeric,
HalfWidth::numeric)
  global x,y;
  if HalfWidth=0 then error "zero
half width of strip" end if;
  (x,y) -> (HalfWidth^2 - (x -
DisplX)^2)/2/HalfWidth;
end proc;
> nCircle := proc(DisplX::numeric,
DisplY::numeric, Radius::numeric)
  global x,y;
  if Radius=0 then error "zero radius
of circle" end if;
  (x,y) -> (Radius^2 - (x - DisplX)^2
- (y - DisplY)^2)/2/Radius;
end proc;
> HalfPlane := proc(X1::numeric,
Y1::numeric, X2::numeric,
Y2::numeric)
  global x,y;
  if X1=X2 and Y1=Y2 then error
"can't create line through one
point"
  elif X1=X2 and Y1<Y2 then (x,y) -
> x - X1;
  elif X1=X2 and Y1>Y2 then (x,y)
-> -x + X1;
  else (x,y) -> (-Y2+Y1)/(-X2+X1)*x
+ (X1*Y2-Y1*X2)/(-X2+X1) - y;
  end if;
end proc;
> nHalfPlane := proc(X1::numeric,
Y1::numeric, X2::numeric,
Y2::numeric)
  global x,y;
  if X1=X2 and Y1=Y2 then error
"can't create line through one
point"
  elif X1=X2 and Y1<Y2 then (x,y) -
> (x-X1)/(x^2-2*x*X1+X1^2+1)^(1/2);
  elif X1=X2 and Y1>Y2 then (x,y)
-> -(x-X1)/(x^2-
2*x*X1+X1^2+1)^(1/2);
  else (x,y) -> (-x*Y2+x*Y1+X1*Y2-
Y1*X2+y*X2-y*X1)/(-
X2+X1)/((2*x*Y1*y*X2-2*x*Y1*y*X1-
2*x^2*Y2*Y1-2*x*Y2^2*X1-
2*x*Y1^2*X2+2*x*Y2*Y1*X2-
2*x*Y2*y*X2+2*x*Y1*X1*Y2+2*x*Y2*y*X1
-2*X1^2*Y2*y-2*Y1*X2^2*y-
2*y^2*X2*X1+Y1^2*X2^2+x^2*Y2^2+x^2*y
1^2+X1^2*Y2^2+y^2*X2^2+y^2*X1^2+Y2^2
-2*Y2*Y1+Y1^2+X2^2-
2*X2*X1+2*X1*Y2*y*X2-
2*X1*Y2*Y1*X2+2*Y1*X2*y*X1+X1^2)/(-
X2+X1)^(1/2);
  end if;
end proc;
> Ellipse := proc(DisplX::numeric,
DisplY::numeric, HalfAxeX::numeric,
HalfAxeY::numeric, Angle)
  global x,y;
  if HalfAxeX=0 or HalfAxeY=0 then
error "zero half axis of ellipse"
end if;
  (x,y) -> 1 - ((x -
DisplX)*cos(Angle) + (y -
DisplY)*sin(Angle))^2/HalfAxeX^2 -
(-(x - DisplX)*sin(Angle) + (y -
DisplY)*cos(Angle))^2/HalfAxeY^2;
end proc;
> Astroid := proc(DisplX::numeric,
DisplY::numeric, HalfWidth::numeric,
Angle)
  global x,y;
  if HalfWidth=0 then error "zero
half width of astroid" end if;
  (x,y) -> -root((x - DisplX)^2,3) -
root((y - DisplY)^2,3) + Half-
Width^(2/3);
end proc;
> OvalKassini :=
proc(DisplX::numeric,
DisplY::numeric, HalfCen-
ter::numeric, Parameter::numeric)
  global x,y;
  if HalfCenter=0 then error "zero
positions of centers of oval"
  elif HalfCenter>=Parameter then
error "parameter must be greater
then position of center";
  end if;
  (x,y) -> -((x - DisplX)^2 + (y -
DisplY)^2 + HalfCenter^2)^2 +
4*HalfCenter^2*(x - DisplX)^2 + Pa-
rameter^4;
end proc;
> SetRSystem := proc(alpha::numeric)
  global Alpha;
  unprotect('Alpha');
  if (alpha>1) or (alpha<-1) then
Alpha := 'Alpha';
  error "alpha must be in range [-
1,1]"
  end if;
  Alpha := alpha;
  protect('Alpha');
end proc;
> `&Un` := proc(Func_1::procedure,
Func_2::procedure)

```

```

if not type('Alpha',protected) then
error "the R system is not set, use
SetRSystem(alpha)" end if;
unapply((Func_1(x,y) + Func_2(x,y)
+ sqrt(Func_1(x,y)^2 + Func_2(x,y)^2
- 2*Alpha*x*y))/(1 + Alpha),x,y);
end proc;
> `&In` := proc(Func_1::procedure,
Func_2::procedure)
if not type('Alpha',protected) then
error "the R system is not set, use
SetRSystem(alpha)" end if;
unapply((Func_1(x,y) + Func_2(x,y)
- sqrt(Func_1(x,y)^2 + Func_2(x,y)^2
- 2*Alpha*x*y))/(1 + Alpha),x,y);
end proc;
> `&Not` := proc(Func_1::procedure)
if not type('Alpha',protected) then
error "the R system is not set, use
SetRSystem(alpha)" end if;
unapply(-Func_1(x,y),x,y);
end proc;
> Grad := proc(Func::algebraic)
Vec-
tor([diff(Func,x),diff(Func,y)]);
end proc;
> Div := proc(Vect::Vector)
diff(Vect[1],x) + diff(Vect[2],y);
end proc;
> `&.` := proc(Vect1::Vector,
Vect2::Vector)
LinearAlgebra[Transpose](Vect1) .
Vect2;
end proc;
> Lapl := proc(Func::algebraic)
diff(Func,x,x) + diff(Func,y,y);
end proc;
> PlotDomain := proc(Domain::name,
Boundary::list)

plots[implicitplot](Domain(x,y),x=Bo
undary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],numpoints
=1000, scaling=constrained,
color=black);
end proc;
> PlotDomain3d := proc(Domain::name,
Boundary::list, zmax::numeric)

plot3d(Domain(x,y),x=Boundary[1]..Bo
undary[3],
y=Boundary[2]..Boundary[4], scal-
ing=constrained,view=0..zmax,axes=bo
xed, grid=[50,50]);
end proc;
> PlotIntPaths := proc(Domain::name,
Boundary::list, MeshX::table,
MeshY::table)
local i, j, k, pp;
pp[1] :=
plots[implicitplot](Domain(x,y),x=Bo
undary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],numpoints

```

```

=1000, scaling=constrained, thick-
ness=2, color=black):
print(nops([indices(MeshX)]));
k := 2;
for i from 1 to
nops([indices(MeshX)]) do
for j from 1 to
nops([indices(MeshY[i])]) do
pp[k] :=
plots[pointplot]([MeshX[i][1],MeshY
[i][j][1],[MeshX[i][1],
MeshY[i][j][2]]},connect=true,
color=blue, thickness=3):
k := k + 1;
end do;
end do;
i := 1;
plots[display]({seq(pp[i],i=1..(k-
1))});
end proc;

> PlotIntPathsM1 :=
proc(Domain::name, Boundary::list,
MeshX::table, MeshY::table)
local i, j, k, pp, SetColor;
pp[1] :=
plots[implicitplot](Domain(x,y),x=Bo
undary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],numpoints
=1000, scaling=constrained, thick-
ness=2, color=black):
print(nops([indices(MeshX)]));
k := 2;
SetColor := x -> piece-
wise(x=3,blue,x=5,red,x=8,green,yell
ow);
for i from 1 to
nops([indices(MeshX)]) do
for j from 1 to
nops([indices(MeshY[i])]) do
pp[k] :=
plots[pointplot]([MeshX[i][1],MeshY
[i][j][1],[MeshX[i][1],
MeshY[i][j][2]]},connect=true,
color=SetColor(MeshY[i][j][3]),
thickness=3):
k := k + 1;
end do;
end do;
i := 1;
plots[display]({seq(pp[i],i=1..(k-
1))});
end proc;

> PlotSolution := proc(Domain::name,
Solution::name, Boundary::list,
ContNumber::posint)
local f, a, b;
f := piece-
wise(Domain(x,y)>0,Solution(x,y),0):
a := labelledcontour-
plot(f,x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],filled=tr
ue,contours=ContNumber,coloring=[whi

```

```

te,blue], scaling=constrained,
grid=[50,50]):
b := implicit-
plot(Domain(x,y),x=Boundary[1]..Boun-
dary[3], y=Boundary[2]..Boundary[4],
numpoints=2000, thick-
ness=2,scaling=constrained,
color=black):
plots[display]({a,b});
end proc;
> PlotSolution3d :=
proc(Domain::name, Solution::name,
Boundary::list, ContNumber::posint)
local f, a, b;
f := piece-
wise(Domain(x,y)>0,Solution(x,y),0):
a := contour-
plot3d(f,x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],filled=tr-
ue,contours=ContNumber,coloring=[whi-
te,blue], scaling=constrained,
grid=[50,50]):
plots[display]({a});
end proc;
> PrepareForDbInt10 :=
proc(Domain::procedure, Bound-
ary::list, MeshX::name, MeshY::name,
NumOfSteps::posint)
local Nodes, p, g, CurrentNodes, i,
j, BoundsY, Temp, k;
global Eps;
Nodes := [-0.9739065286, -
0.8650633668, -0.6794095683, -
0.4333953941, -0.1488743390,
0.1488743390, 0.4333953941,
0.6794095683, 0.8650633668,
0.9739065286];
Eps := 1.0e-6;
MeshX := 'MeshX';
MeshY := 'MeshY';
p := (Boundary[1] + Boundary[3])/2;
g := (Boundary[3] - Boundary[1])/2;
for i from 1 to 10 do Current-
Nodes[i] := g*Nodes[i] + p; end do;
j := 1;
for i from 1 to 10 do
Temp := Fin-
dRoots(Domain(CurrentNodes[i],y),
evalf(Boundary[2]),
evalf(Boundary[4]), NumOfSteps,
Eps);
print(Temp);
if nops(Temp)=0 or
type(nops(Temp),odd) then next;
else MeshX[j] := CurrentNodes[i],
i;
BoundsY[j] := Temp;
j := j + 1;
end if;
end do;
if (j-1)<=2 then error "bad domain
or bad boundary rectangle" end if;
for i from 1 to (j-1) do
Temp := 1;
for k from 1 to
nops(BoundsY[i])/2 do
MeshY[i][k] :=
BoundsY[i][Temp],
BoundsY[i][Temp+1], Int-
Type(abs(BoundsY[i][Temp]-
BoundsY[i][Temp+1]));
for k from 1 to
nops(BoundsY[i])/2 do
MeshY[i][k] :=
BoundsY[i][Temp],
BoundsY[i][Temp+1];
Temp := Temp + 2;
end do;
end do;
Temp := convert(Domain,string);
print(`Preparation for DbIntGauss10
over domain`||` `||`Temp`||` in rec-
tangle `Boundary`,` is done!`);
end proc;
> PrepareForDbInt10M1 :=
proc(Domain::procedure, Bound-
ary::list, MeshX::name, MeshY::name,
NumOfSteps::posint)
local Nodes, p, g, CurrentNodes, i,
j, BoundsY, Temp, k, size, IntType;
global Eps;
Nodes := [-0.9739065286, -
0.8650633668, -0.6794095683, -
0.4333953941, -0.1488743390,
0.1488743390, 0.4333953941,
0.6794095683, 0.8650633668,
0.9739065286];
Eps := 1.0e-6;
MeshX := 'MeshX';
MeshY := 'MeshY';
p := (Boundary[1] + Boundary[3])/2;
g := (Boundary[3] - Boundary[1])/2;
size := Boundary[4] - Boundary[2];
IntType := x -> piece-
wise(x<size/3,3,x<size/2,5,x<size/1.
1,8,10);
for i from 1 to 10 do Current-
Nodes[i] := g*Nodes[i] + p; end do;
j := 1;
for i from 1 to 10 do
Temp := Fin-
dRoots(Domain(CurrentNodes[i],y),
evalf(Boundary[2]),
evalf(Boundary[4]), NumOfSteps,
Eps);
print(Temp);
if nops(Temp)=0 or
type(nops(Temp),odd) then next;
else MeshX[j] := CurrentNodes[i],
i;
BoundsY[j] := Temp;
j := j + 1;
end if;
end do;
if (j-1)<=2 then error "bad domain
or bad boundary rectangle" end if;
for i from 1 to (j-1) do
Temp := 1;
for k from 1 to
nops(BoundsY[i])/2 do
MeshY[i][k] :=
BoundsY[i][Temp],
BoundsY[i][Temp+1], Int-
Type(abs(BoundsY[i][Temp]-
BoundsY[i][Temp+1]));

```

```

    Temp := Temp + 2;
  end do;
end do;
Temp := convert(Domain,string);
print(`Preparation for DbInt-
Gauss10M1 over domain`||`
`||`Temp`||` in rectangle
`,Boundary`,` is done!`);
end proc;
> DbIntGauss10 := proc(MeshX::table,
MeshY::table, Func::algebraic,
Boundary::list)
  local Weights, p, g, CurrentNodes,
i, j, k, IntegralsY;
  Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
  p := (Boundary[1] + Boundary[3])/2;
  g := (Boundary[3] - Boundary[1])/2;
  for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
  k := 1;
  for i from 1 to
nops([indices(MeshX)]) do
    for j from 1 to
nops([indices(MeshY[i])]) do
      IntegralsY[k] :=
evalf(Int(Func(MeshX[i][1],y),
y=MeshY[i][j][1]..MeshY[i][j][2]))*W
eights[MeshX[i][2]];
      k := k + 1;
    end do;
  end do;
  i := 1;
  evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;
> DbIntGauss10M1 :=
proc(MeshX::table, MeshY::table,
Func::algebraic, Boundary::list)
  local Weights, p, g, CurrentNodes,
i, j, k, IntegralsY;
  Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
  p := (Boundary[1] + Boundary[3])/2;
  g := (Boundary[3] - Boundary[1])/2;
  for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
  k := 1;
  for i from 1 to
nops([indices(MeshX)]) do
    for j from 1 to
nops([indices(MeshY[i])]) do
      if MeshY[i][j][3]=3 then Inte-
gralsY[k] := Int-
Gauss3(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=5 then In-
tegralsY[k] := Int-
Gauss5(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=8 then In-
tegralsY[k] := Int-
Gauss8(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      else IntegralsY[k] := Int-
Gauss10(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]];
      end if;
      k := k + 1;
    end do;
  end do;
  i := 1;
  evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;
> DbIntGauss10hf :=
proc(MeshX::table, MeshY::table,
Func::algebraic, Boundary::list)
  local Weights, p, g, CurrentNodes,
i, j, k, IntegralsY;
  Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
  p := (Boundary[1] + Boundary[3])/2;
  g := (Boundary[3] - Boundary[1])/2;
  for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
  k := 1;
  for i from 1 to
nops([indices(MeshX)]) do
    for j from 1 to
nops([indices(MeshY[i])]) do
      IntegralsY[k] := Int-
Gauss10hf(MeshY[i][j][1],MeshY[i][j]
[2],evalf(Func(MeshX[i][1],y)))*Weig-
hts[MeshX[i][2]];
      k := k + 1;
    end do;
  end do;
  i := 1;
  evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;
> DbIntGauss10M1hf :=
proc(MeshX::table, MeshY::table,
Func::algebraic, Boundary::list)
  local Weights, p, g, CurrentNodes,
i, j, k, IntegralsY;
  Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
  p := (Boundary[1] + Boundary[3])/2;
  g := (Boundary[3] - Boundary[1])/2;
  for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
  k := 1;
  for i from 1 to
nops([indices(MeshX)]) do
    for j from 1 to
nops([indices(MeshY[i])]) do
      if MeshY[i][j][3]=3 then Inte-
gralsY[k] := Int-
Gauss3(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=5 then In-
tegralsY[k] := Int-
Gauss5(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=8 then In-
tegralsY[k] := Int-
Gauss8(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      else IntegralsY[k] := Int-
Gauss10(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]];
      end if;
      k := k + 1;
    end do;
  end do;
  i := 1;
  evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;
> DbIntGauss10M1hf :=
proc(MeshX::table, MeshY::table,
Func::algebraic, Boundary::list)
  local Weights, p, g, CurrentNodes,
i, j, k, IntegralsY;
  Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
  p := (Boundary[1] + Boundary[3])/2;
  g := (Boundary[3] - Boundary[1])/2;
  for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
  k := 1;
  for i from 1 to
nops([indices(MeshX)]) do
    for j from 1 to
nops([indices(MeshY[i])]) do
      if MeshY[i][j][3]=3 then Inte-
gralsY[k] := Int-
Gauss3(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=5 then In-
tegralsY[k] := Int-
Gauss5(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      elif MeshY[i][j][3]=8 then In-
tegralsY[k] := Int-
Gauss8(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weights
[MeshX[i][2]];
      else IntegralsY[k] := Int-
Gauss10(MeshY[i][j][1],MeshY[i][j][2]
,evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]];
      end if;
      k := k + 1;
    end do;
  end do;
  i := 1;
  evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;

```

```

0.2190863632, 0.1494513492,
0.6667134392e-1];
p := (Boundary[1] + Boundary[3])/2;
g := (Boundary[3] - Boundary[1])/2;
for i from 1 to 10 do Weights[i] :=
g*Weights[i]; end do;
k := 1;
for i from 1 to
nops([indices(MeshX)]) do
for j from 1 to
nops([indices(MeshY[i])]) do
if MeshY[i][j][3]=3 then Inte-
gralsY[k] := Int-
Gauss3hf(MeshY[i][j][1],MeshY[i][j][
2],evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]];
elif MeshY[i][j][3]=5 then In-
tegralsY[k] := Int-
Gauss5hf(MeshY[i][j][1],MeshY[i][j][
2],evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]]; elif
MeshY[i][j][3]=8 then IntegralsY[k]
:= Int-
Gauss8hf(MeshY[i][j][1],MeshY[i][j][
2],evalf(Func(MeshX[i][1],y)))*Weigh-
ts[MeshX[i][2]];
else IntegralsY[k] := Int-
Gauss10hf(MeshY[i][j][1],MeshY[i][j][
2],evalf(Func(MeshX[i][1],y)))*Weig-
hts[MeshX[i][2]];
end if;
k := k + 1;
end do;
end do;
i := 1;
evalf(add(IntegralsY[i],i=1..(k-
1))):
end proc;
> IntGauss3 := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Funcl;
Nodes := [-0.7745966692, 0.,
0.7745966692];
Weights := [0.5555555558,
0.88888888888, 0.5555555558];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Funcl := unapply(Func,y);
for i from 1 to 3 do
Weights[i] := g*Weights[i];
Nodes[i] := g*Nodes[i] + p;
Integr := Integr +
Funcl(Nodes[i])*Weights[i];
end do;
Integr;
end proc;
> IntGauss3hf := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Funcl, NodesHF;
Nodes := [-0.7745966692, 0.1e-30,
0.7745966692];
Weights := [0.5555555558,
0.88888888888, 0.5555555558];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Funcl := unapply(Func,y);
NodesHF := hfarray(1..3);
for i from 1 to 3 do
Weights[i] := g*Weights[i];
NodesHF[i] := g*Nodes[i] + p;
Integr := Integr +
evalhf(Funcl(NodesHF[i]))*Weights[i]
;
end do;
Integr;
end proc;
> IntGauss5 := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Funcl;
Nodes := [-0.9061798459, -
0.5384693101, 0., 0.5384693101,
0.9061798459];
Weights := [0.2369268850,
0.4786286716, 0.5688888888,
0.4786286716, 0.2369268850];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Funcl := unapply(Func,y);
for i from 1 to 5 do
Weights[i] := g*Weights[i];
Nodes[i] := g*Nodes[i] + p;
Integr := Integr +
Funcl(Nodes[i])*Weights[i];
end do;
Integr;
end proc;
> IntGauss5hf := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Funcl, NodesHF;
Nodes := [-0.9061798459, -
0.5384693101, 0.1e-30, 0.5384693101,
0.9061798459];
Weights := [0.2369268850,
0.4786286716, 0.5688888888,
0.4786286716, 0.2369268850];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Funcl := unapply(Func,y);
NodesHF := hfarray(1..5);
for i from 1 to 5 do
Weights[i] := g*Weights[i];
NodesHF[i] := g*Nodes[i] + p;
Integr := Integr +
evalhf(Funcl(NodesHF[i]))*Weights[i]
;
end do;
Integr;
end proc;
> IntGauss8 := proc(p1::numeric,
p2::numeric, Func::algebraic)

```



```

local Weights, p, g, Nodes, i, In-
tegr, Func1;
Nodes := [-0.9602898565, -
0.7966664774, -0.5255324099, -
0.1834346425, 0.1834346425,
0.5255324099, 0.7966664774,
0.9602898565];
Weights := [0.1012285359,
0.2223810344, 0.3137066460,
0.3626837832, 0.3626837832,
0.3137066460, 0.2223810344,
0.1012285359];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Func1 := unapply(Func,y);
for i from 1 to 8 do
Weights[i] := g*Weights[i];
Nodes[i] := g*Nodes[i] + p;
Integr := Integr +
Func1(Nodes[i])*Weights[i];
end do;
Integr;
end proc;
> IntGauss8hf := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Func1, NodesHF;
Nodes := [-0.9602898565, -
0.7966664774, -0.5255324099, -
0.1834346425, 0.1834346425,
0.5255324099, 0.7966664774,
0.9602898565];
Weights := [0.1012285359,
0.2223810344, 0.3137066460,
0.3626837832, 0.3626837832,
0.3137066460, 0.2223810344,
0.1012285359];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Func1 := unapply(Func,y);
NodesHF := hfarray(1..8);
for i from 1 to 8 do
Weights[i] := g*Weights[i];
NodesHF[i] := g*Nodes[i] + p;
Integr := Integr +
evalhf(Func1(NodesHF[i]))*Weights[i]
;
end do;
Integr;
end proc;
> IntGauss10 := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Func1;
Nodes := [-0.9739065286, -
0.8650633668, -0.6794095683, -
0.4333953941, -0.1488743390,
0.1488743390, 0.4333953941,
0.6794095683, 0.8650633668,
0.9739065286];
Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Func1 := unapply(Func,y);
NodesHF := hfarray(1..10,Nodes);
for i from 1 to 10 do
Weights[i] := g*Weights[i];
NodesHF[i] := g*Nodes[i] + p;
Integr := Integr +
evalhf(Func1(NodesHF[i]))*Weights[i]
;
end do;
Integr;
end proc;
> IntGaussA := proc(a::numeric,
b::numeric, Func::algebraic, NumOf-
Nodes::posint)
local Weights, Nodes, Integr, m,
xm, xl, z1, i, z, p1, p2, p3, j, pp,
EPS, Func1;
EPS := 1.0e-10;
m := (NumOfNodes+1)/2;
xm := 0.5*(b+a);
xl := 0.5*(b-a);
z1 := evalf(cos(Pi*(1-
0.25)/(NumOfNodes+0.5)))+5:
for i from 1 to m do
z := evalf(cos(Pi*(i-
0.25)/(NumOfNodes+0.5))):
while (abs(z-z1) > EPS) do
p1 := 1;
p2 := 0;
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Func1 := unapply(Func,y);
for i from 1 to 10 do
Weights[i] := g*Weights[i];
Nodes[i] := g*Nodes[i] + p;
Integr := Integr +
Func1(Nodes[i])*Weights[i];
end do;
Integr;
end proc;
> IntGauss10hf := proc(p1::numeric,
p2::numeric, Func::algebraic)
local Weights, p, g, Nodes, i, In-
tegr, Func1, NodesHF;
Nodes := [-0.9739065286, -
0.8650633668, -0.6794095683, -
0.4333953941, -0.1488743390,
0.1488743390, 0.4333953941,
0.6794095683, 0.8650633668,
0.9739065286];
Weights := [0.6667134392e-1,
0.1494513492, 0.2190863632,
0.2692667196, 0.2955242250,
0.2955242250, 0.2692667196,
0.2190863632, 0.1494513492,
0.6667134392e-1];
p := (p2 + p1)/2;
g := (p2 - p1)/2;
Integr := 0.0;
Func1 := unapply(Func,y);
NodesHF := hfarray(1..10,Nodes);
for i from 1 to 10 do
Weights[i] := g*Weights[i];
NodesHF[i] := g*Nodes[i] + p;
Integr := Integr +
evalhf(Func1(NodesHF[i]))*Weights[i]
;
end do;
Integr;
end proc;
> IntGaussA := proc(a::numeric,
b::numeric, Func::algebraic, NumOf-
Nodes::posint)
local Weights, Nodes, Integr, m,
xm, xl, z1, i, z, p1, p2, p3, j, pp,
EPS, Func1;
EPS := 1.0e-10;
m := (NumOfNodes+1)/2;
xm := 0.5*(b+a);
xl := 0.5*(b-a);
z1 := evalf(cos(Pi*(1-
0.25)/(NumOfNodes+0.5)))+5:
for i from 1 to m do
z := evalf(cos(Pi*(i-
0.25)/(NumOfNodes+0.5))):
while (abs(z-z1) > EPS) do
p1 := 1;
p2 := 0;

```

```

        for j from 1 to NumOfNodes do
            p3 := p2;
            p2 := p1;
            p1 := ((2*j-1)*z*p2-(j-
1)*p3)/j;
        end do;
        pp := NumOfNodes*(z*p1-
p2)/(z*z-1);
        z1 := z;
        z := z1-p1/pp;
    end do;
Nodes[i] := xm-x1*z;
Nodes[NumOfNodes+1-i] := xm+x1*z;
Weights[i] := 2*x1/((1-z*z)*pp*pp);
Weights[NumOfNodes+1-i] :=
Weights[i];
end do;
Nodes := convert(Nodes,list);
Weights := convert(Weights,list);
Integr := 0.0;
Func1 := unapply(Func,y);
for i from 1 to NumOfNodes do
    Integr := Integr +
Func1(Nodes[i])*Weights[i];
end do;
Integr;
end proc;
> NewtonRaphsonBisection :=
proc(x1::numeric, x2::numeric,
fl::numeric, fr::numeric,
Func::procedure, Deriv::algebraic,
Eps::numeric)
    local ITMAX, GuessRoot, i, dx, x1,
xr, dxold, f, df, temp;
    ITMAX := 20;
    if (fl>0 and fr>0) or (fl<0 and
fr<0) then error "Root must be
bracketed!" end if;
    #if abs(fl)<Eps then return x1; end
if;
    #if abs(fr)<Eps then return x2; end
if;
    if fl<0 then
        x1 := x1;
        xr := x2;
    else
        x1 := x2;
        xr := x1;
    end if;
    GuessRoot := evalf((x1 + x2)/2);
    dxold := evalf(abs(x2-x1));
    dx := dxold;
    f := evalf(Func(GuessRoot));
    df := evalf(Deriv(GuessRoot));
    for i from 1 to ITMAX do
        if ((GuessRoot-xr)*df-
f)*((GuessRoot-x1)*df-f)>0 or
abs(2*f)>abs(dxold*df) then
            dxold := dx;
            dx := (xr-x1)/2.0;
            GuessRoot := x1 + dx;
            if x1=GuessRoot then return
GuessRoot; end if;
        else
            dxold := dx;
            dx := f/df;
            temp := GuessRoot;
            GuessRoot := GuessRoot - dx;
            if temp=GuessRoot then return
GuessRoot; end if;
            end if;
            if abs(dx)<Eps then return Guess-
Root; end if;
            f := evalf(Func(GuessRoot));
            df := evalf(Deriv(GuessRoot));
            if f<0 then x1 := GuessRoot;
            else xr := GuessRoot; end if;
        end do;
        error "Maximum number of iterations
exceeded in NewtonRaphsonBisection";
    end proc;
> FindRoots := proc(Expr::algebraic,
c::numeric, d::numeric, NumOf-
Steps::posint, Eps::numeric)
    local roots, Func, Deriv, step, po-
sition, flag, i, Func_pos,
Func_pos_old, temp;
    #option trace;
    Func := unapply(Expr,y);
    Deriv := unap-
ply(diff(Func(y),y),y);
    roots := [];
    step := (d-c)/NumOfSteps;
    position := evalf(c);
    Func_pos := evalf(Func(position));
    if abs(Func_pos)<Eps then
        roots := [op(roots),position];
        position := position + step;
        Func_pos :=
evalf(Func(position));
    end if;
    if Func_pos>=0 then flag := true;
    else flag := false; end if;
    while ((position+step)<=d) do
        position := position + step;
        Func_pos_old := Func_pos;
        Func_pos :=
evalf(Func(position));
        if Func_pos<0 and flag=true then
            flag := false;
            temp := NewtonRaphsonBi-
section(position-step, position,
Func_pos_old, Func_pos, Func, Deriv,
Eps);
            roots := [op(roots),temp];
        elif Func_pos>=0 and flag=false
then
            flag := true;
            temp := NewtonRaphsonBi-
section(position-step, position,
Func_pos_old, Func_pos, Func, Deriv,
Eps);
            roots := [op(roots),temp];
        end if;
    end do;
    temp := evalf(Func(d));
    if abs(temp)<Eps then roots :=
[op(roots),d]; end if;

```

```

    if abs(roots[nops(roots)]-
roots[nops(roots)-1])<Eps then
        roots := [op(1..(nops(roots)-
1),roots)];
    end if;
    sort(roots);
end proc;
> GenPowerPolynoms :=
proc(n::posint)
global NumberOfBasisFunc;
local i, temp, j, BasisList, k;
if n=1 then error "power of polynoms
must be greated than 1" end if;
BasisList[1] := 1;
BasisList[2] := x;
BasisList[3] := y;
j := 4;
for i from 2 to n do
    temp := combi-
nat[composition](i,2);
    temp := temp union {[i,0],[0,i]};
    for k from 1 to nops(temp) do
        BasisList[j] :=
x^(temp[k][1])*y^(temp[k][2]);
        j := j + 1;
    end do;
end do;
NumberOfBasisFunc := j-1;
convert(BasisList,list);
end proc;
> GenCoordFunctions :=
proc(Domain::algebraic, BasisFunc-
cList::list)
unapply(map(proc(Temp) Domain*Temp
end proc,BasisFuncList),x,y);
end proc;
> CreateLeftMatrix :=
proc(GenericMember::procedure,
MeshX::table, MeshY::table, Bound-
ary::list, Symmetry::boolean)
local i,j,Temp, A;
if Symmetry=true then
    A := Ma-
trix(NumberOfBasisFunc,shape=symmetr-
ic):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to i do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            A[i,j] := DbInt-
Gauss10M1(MeshX, MeshY, Temp, Bound-
ary):
        end do;
    end do;
else
    A := Matrix(NumberOfBasisFunc):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to NumberOfBasis-
Func do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            A[i,j] := DbIntGauss10(MeshX,
MeshY, Temp, Boundary):
        end do;
    end do;
else
    A := Matrix(NumberOfBasisFunc):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to NumberOfBasis-
Func do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            A[i,j] := DbIntGauss10(MeshX,
MeshY, Temp, Boundary):
        end do;
    end do;
end if;
return A;
end proc;
> CreateLeftMatrixM1 :=
proc(GenericMember::procedure,
MeshX::table, MeshY::table, Bound-
ary::list, Symmetry::boolean)
local i,j,Temp, A;
if Symmetry=true then
    A := Ma-
trix(NumberOfBasisFunc,shape=symmetr-
ic):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to i do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            A[i,j] := DbInt-
Gauss10M1(MeshX, MeshY, Temp, Bound-
ary):
        end do;
    end do;
else
    A := Matrix(NumberOfBasisFunc):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to NumberOfBasis-
Func do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            A[i,j] := DbInt-
Gauss10M1(MeshX, MeshY, Temp, Bound-
ary):
        end do;
    end do;
end if;
return A;
end proc;
> CreateLeftMatrixHF :=
proc(GenericMember::procedure,
MeshX::table, MeshY::table, Bound-
ary::list, Symmetry::boolean)
local i,j,Temp, A;
#options trace;
if Symmetry=true then
    A := Ma-
trix(NumberOfBasisFunc,shape=symmetr-
ic):
    for i from 1 to NumberOfBasisFunc
do
        for j from 1 to i do
            Temp := unap-
ply(GenericMember(i,j),x,y);
            #print(i);
            #print(j);
            A[i,j] := DbInt-
Gauss10hf(MeshX, MeshY, Temp, Bound-
ary):
            #print(A[i,j]);
        end do;
    end do;
end if;
return A;
end proc;

```

```

else
  A := Matrix(NumberOfBasisFunc):
  for i from 1 to NumberOfBasisFunc
  do
    for j from 1 to NumberOfBasis-
    Func do
      Temp := unap-
      ply(GenericMember(i,j),x,y):
      A[i,j] := DbInt-
      Gauss10hf(MeshX, MeshY, Temp, Bound-
      ary):
    end do;
  end do;
end if;
return A;
end proc;
> CreateLeftMatrixMlhf :=
proc(GenericMember::procedure,
MeshX::table, MeshY::table, Bound-
ary::list, Symmetry::boolean)
local i,j,Temp, A;
#options trace;
if Symmetry=true then
  A := Ma-
  trix(NumberOfBasisFunc,shape=symmetr-
  ic):
  for i from 1 to NumberOfBasisFunc
  do
    for j from 1 to i do
      Temp := unap-
      ply(GenericMember(i,j),x,y);
      A[i,j] := DbInt-
      Gauss10Mlhf(MeshX, MeshY, Temp,
      Boundary):
    end do;
  end do;
else
  A := Matrix(NumberOfBasisFunc):
  for i from 1 to NumberOfBasisFunc
  do
    for j from 1 to NumberOfBasis-
    Func do
      Temp := unap-
      ply(GenericMember(i,j),x,y);
      A[i,j] := DbInt-
      Gauss10Mlhf(MeshX, MeshY, Temp,
      Boundary):
    end do;
  end do;
end if;
return A;
end proc;
> CreateRightVector :=
proc(GenericMember::procedure,
MeshX::table, MeshY::table, Bound-
ary::list)
local i,Temp, B;
B := Vector(NumberOfBasisFunc):
for i from 1 to NumberOfBasisFunc do
  Temp := unap-
  ply(GenericMember(i),x,y);
  B[i] := DbIntGauss10(MeshX,
  MeshY, Temp, Boundary):
end do;
return B;
end proc;
> SolveSystem := proc(A::Matrix,
B::Vector)
LinearAlgebra[LinearSolve](A,B);
end proc;
> CreateSolution :=
proc(Solution::Vector, Coord-
Func::procedure)
local i;
unap-
ply(add(Solution[i]*CoordFunc(x,y)[i
],i=1..NumberOfBasisFunc),x,y);
end proc;

```

ЗАКЛЮЧЕНИЕ

В работе была реализована программа для автоматизации аналитического решения краевых задач математической физики на сложных двумерных областях методом R-функций. Были представлены тестовые примеры решения уравнений Софи-Жермен и Пуассона. Результаты сравнивались с численным решением на ANSYS 6.0. Был сделан вывод об их правильности.

Достоинства данной работы:

- 1) был автоматизирован процесс построения сложной области из областей более простой формы (опорных областей);
- 2) был автоматизирован процесс решения – по заданной структуре решения и заданному общему члену матрицы Рунца программа строит аналитическое решение;
- 3) получаемое аналитическое решение по всем параметрам «лучше» численного;
- 4) весь процесс происходит в «аналитике», следовательно свобода пользователя не ограничена ни чем, т.е. пользователь может создавать свои опорные области напрямую, решать различные задачи при различных начально-краевых условиях;
- 5) интегрированность данной программы в систему компьютерной математики Maple дает возможность использовать ее огромнейший инструментарий «аналитики» и численного анализа.

Программа предназначена для исследователей в области теории приближений и решения начально-краевых задач на сложных областях – ее можно применять для проведения численных экспериментов. Однако она может использоваться и для проектирования различных объектов (например, пластин). Стоимость использования при этом примерно в 20 раз меньше, чем стоимость лицензии на ANSYS в минимальном варианте на 1 год. Это также является немаловажным достоинством.

СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ

1. *Михлин С.Г.* Вариационные методы в математической физике. – М.: Наука, 1970.
2. *Рвачев В.А.* Теория R-функций и некоторые ее приложения. – Киев: Наукова думка, 1982.
3. *Бойко Б.Т., Курпа А.В., Сенчук Ю.Ф.* Уравнения математической физики. Учебное пособие. – Харьков: Бизнес Информ, НТУ «ХПИ», 2002.
4. *Рвачев В.А.* Геометрические приложения алгебры логики. – Киев: Техника, 1967.
5. *Maple 8. Advanced Programming Guide* (в электронном виде).
6. *Maple 8. Help* (в электронном виде).