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

ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ДИНАМИКИ И ПРОЧНОСТИ МАШИН

«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

Введение

Задачи математической физики – такие, как задачи механики деформируемого твердого тела, механики жидкости и газа, электродинамики, теплофизики и других – 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.Л. 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. Физическая интерпретация этих уравнений в двумерной области такова: уравнением Пуассона описывается кручение стержня, изгиб мембраны, стационарная теплопроводность, потенциал электростатического поля; уравнением Софи-Жермен – изгиб тонких пластин. 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. Нам же оставалось «дописать» недостающие процедуры и из них, как из кирпичиков, построить программу. Отметим, что была поставлена цель не создать отдельную программу для решения конкретных уравнений, а наоборот – создать программу, интегрированную в пакет компьютерной математики Maple, чтобы пользователь мог изменять ее для своих нужд, а также пользоваться при этом всею мощью 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.

Покажем на конкретных примерах последовательность решения задачи, давая по мере необходимости комментарии.

Пример 1

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 2e11 Па, коэффициент Пуассона 0.3. Краевые условия – жесткая заделка по всей границе. Нагрузка – равномерное давление, взятое равным 1 Па.

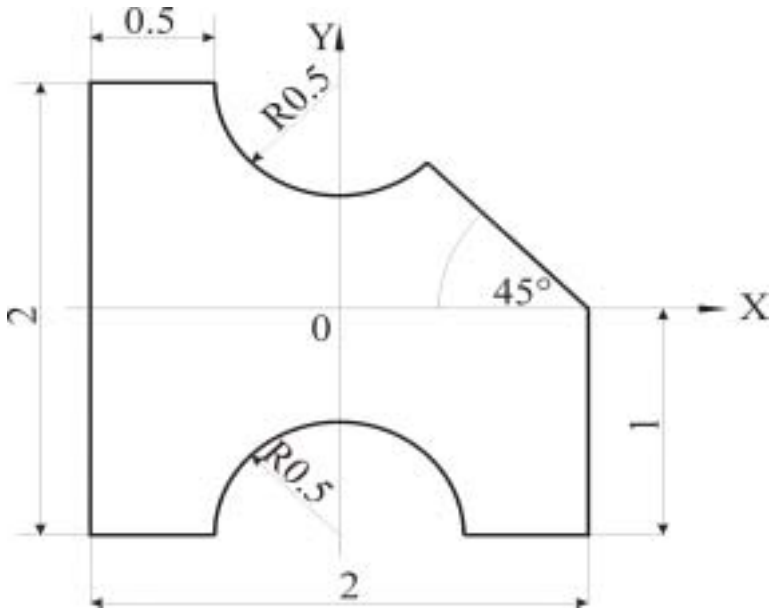


Рисунок 1 – Исследуемая пластина

Математическая постановка задачи. Решить уравнение Софи-Жермен

$$\Delta^2 u(x,y) = \frac{q(x,y)}{D}, \tag{1}$$

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. Здесь $q(x,y) = 1$, $E = 2 \cdot 10^{11}$, $h = 0.01$, $\nu = 0.3$.

Задача дополняется однородными краевыми условиями Дирихле:

$$u(x,y)\big|_{\partial\Omega} = 0; \qquad \left.\frac{\partial u(x,y)}{\partial n}\right|_{\partial\Omega} = 0, \tag{2}$$

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

При данных краевых условиях, эта задача сводится к эквивалентной вариационной задаче нахождения минимума функционала следующего вида

$$F(u) = \iint_{\Omega} \left[\left(\Delta u(x,y)\right)^2 - 2\frac{q(x,y)}{D} u(x,y) \right] d\Omega, \tag{3}$$

где $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ – оператор Лапласа.

Перейдем теперь к решению поставленной задачи, используя написанный пакет программ для Maple.

Шаг 1. Подключение пакета RFM и выбор системы R-операций (рис. 2).

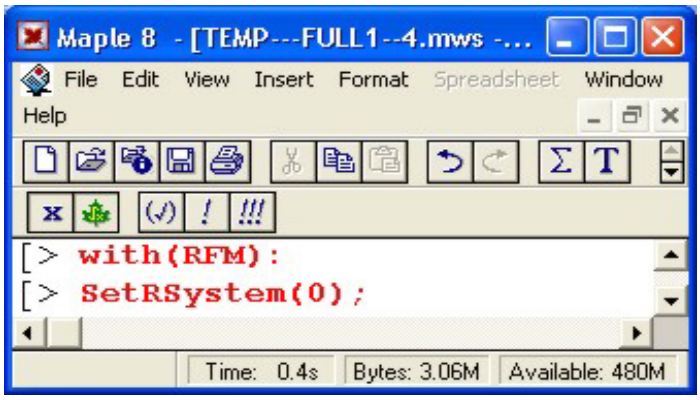


Рисунок 2 – Шаг 1. Подключение пакета RFM и выбор системы R-операций

В программе реализована система R-операций, названная $\Re_\alpha$. Существуют другие системы, однако для рассматриваемого класса задач данная система вполне подходит. Математически, R-операции в данной системе записываются так:

$$\begin{aligned} z_1 \wedge_\alpha z_2 &= \frac{1}{1+\alpha}\left[z_1 + z_2 - \sqrt{z_1^2 + z_2^2 - 2\alpha z_1 z_2}\right]; \\ z_1 \vee_\alpha z_2 &= \frac{1}{1+\alpha}\left[z_1 + z_2 + \sqrt{z_1^2 + z_2^2 - 2\alpha z_1 z_2}\right]; \\ \neg z_1 &= -z_1, \end{aligned} \tag{4}$$

где $-1 \le \alpha \le 1$ применяется для получения различных дополнительных (например, дифференциальных) свойств полученного выражения. Для рассматриваемого класса задач положим $\alpha = 0$, и будем использовать систему R-операций $\Re_0$:

$$z_1 \wedge_0 z_2 = z_1 + z_2 - \sqrt{z_1^2 + z_2^2};$$
$$z_1 \vee_0 z_2 = z_1 + z_2 + \sqrt{z_1^2 + z_2^2}; \quad (5)$$
$$\neg z_1 = -z_1.$$

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

Шаг 2. Создание границы рассматриваемой области. В результате этого шага нужно получить аналитическое выражение для границы нашей пластины. Согласно методу R-функций нам нужно записать аналитические выражения для опорных областей. Для исследуемой пластины опорными областями будут:

$\Omega_1 = \left[F1 = \frac{1}{2}(1 - y^2) \geq 0\right]$ – полоса, параллельная оси X полушириной 1;

$\Omega_2 = \left[F2 = \frac{1}{2}(1 - x^2) \geq 0\right]$ – полоса, параллельная оси Y полушириной 1;

$\Omega_3 = \left[F3 = 0.5^2 - x^2 - (y-1)^2 \geq 0\right]$ – круг с центром в точке (0,1) и радиусом 0.5;

$\Omega_4 = \left[F4 = 0.5^2 - x^2 - (y+1)^2 \geq 0\right]$ – круг с центром в точке (0,-1) и радиусом 0.5;

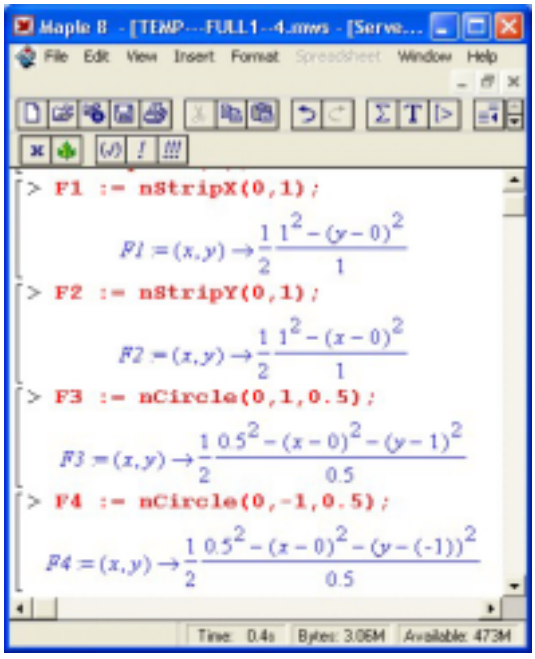


Рисунок 3 – Создание 1-4 опорных областей

$\Omega_5 = \left[F5 = -\frac{-1 + x + y}{\sqrt{3 - 2x - 2y + y^2 + x^2 + 2xy}} \geq 0\right]$ – полуплоскость, задаваемая координатами двух точек: (0,1) и (1,0).

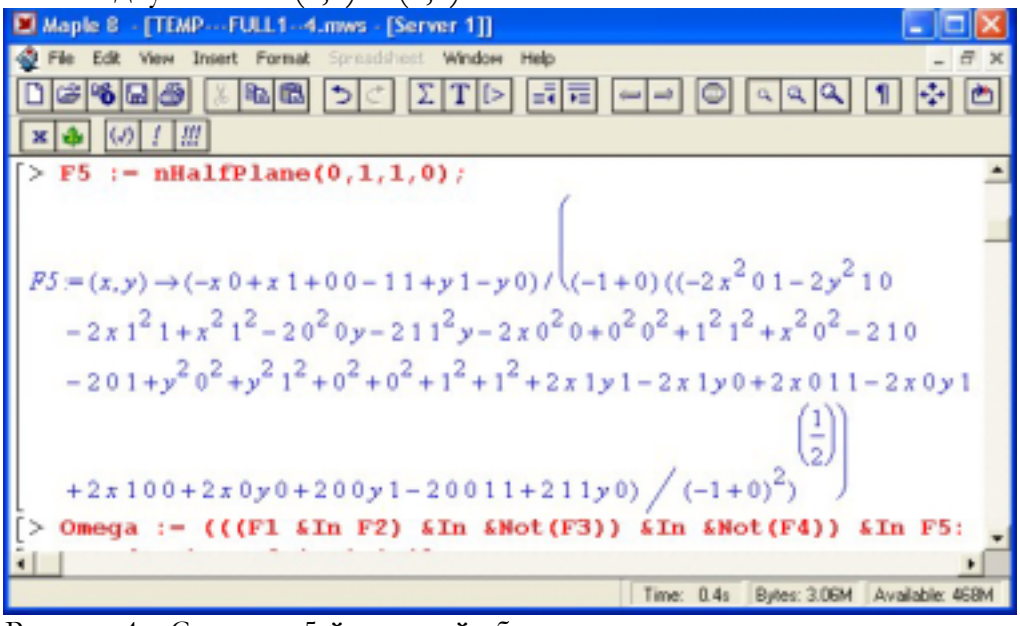


Рисунок 4 – Создание 5-й опорной области и генерация уравнения границы

Далее нужно написать логическую формулу (предикат) для области пользуясь логическими операциями дизъюнкции, конъюнкции и отрицания (рис. 4):

$$\Omega = \Big(\big((\Omega_1 \wedge \Omega_2) \wedge \neg\Omega_3\big) \wedge \neg\Omega_4\Big) \wedge \Omega_5 . \qquad (6)$$

Опишем процесс подробно. Делая пересечение полос 1 и 2 мы получаем квадрат со стороной 2. Далее мы пересекаем этот квадрат с отрицанием круга 3, получив при этом квадрат с вырезанным верхним полукругом. Делаем такую же процедуру с кругом 4, получив квадрат с вырезанными верхним и нижним полукругами. Последний шаг создания границы – пересечение полученной уже области с полуплоскостью 5, в результате имеем логически требуемую геометрию. В принципе, полученную логическую формулу(6) можно было бы упростить, пользуясь различными тождествами булевой алгебры, но мы сознательно этого не делаем, чтобы не потерять наглядность.

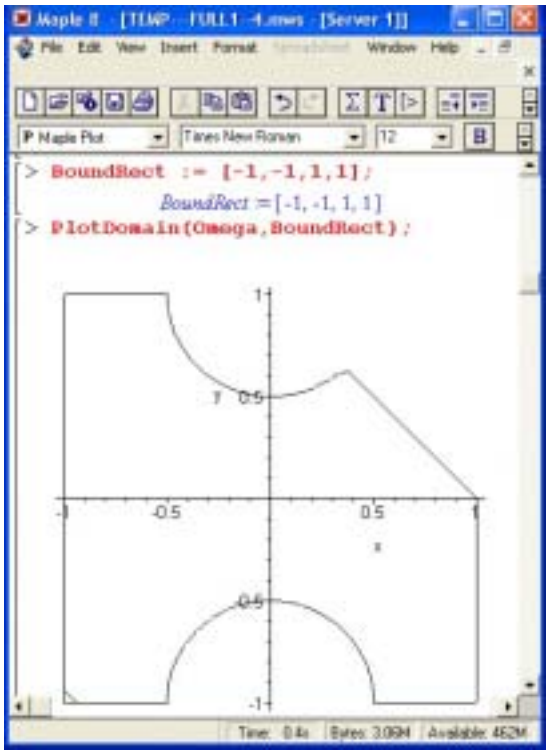


Рисунок 5 – Задание ограничивающего прямоугольника и построение графика области

Чтобы получить аналитически требуемую геометрию, следуя методу R-функций, мы должны формально заменить логические операции дизъюнкции, конъюнкции и отрицания на соответствующие R-операции (5):

$$\Omega = \Big(\big((F1 \wedge_0 F2) \wedge_0 \neg F3\big) \wedge_0 \neg F4\Big) \wedge_0 F5$$

что собственно программа и делает.

В результате мы получаем функцию, обладающую следующими свойствами:

$$\Omega(x,y) = 0, (x,y) \in \partial\Omega,$$

$$\Omega(x,y) > 0, (x,y) \in \Omega,$$

$$\Omega(x,y) < 0, (x,y) \notin \Omega.$$

Далее мы должны задать прямоугольник, ограничивающий нашу область. Теперь можно построить график полученной области, чтобы проверить правильность построения (рис. 5).

Шаг 3. Подготовка области к интегрированию. Далее нам придется вычислять большое количество двойных интегралов по созданной области, поэтому были сделаны процедуры для подготовки области к интегрированию. Интегрирование в данном примере будет выполнятся численно 10-ти узловой формулой Гаусса в каждом направлении.

Первая процедура формирует сетку интегрирования по обоим направлениям (рис. 6). Она находит границы созданной области по оси Y при заданных

координатах X. Так как численное решение нелинейных уравнение в Maple реализовано очень плохо, была написана подпрограмма, ищущая корни уравнения гибридным методом Ньютона-Рафсона с бисекцией.

Следующая процедура графически показывает отрезки интегрирования. Она сделана для отладки процедуры подготовки к интегрированию, так как возможно пользователь захочет для своей конкретной задачи написать свою процедуру интегрирования.

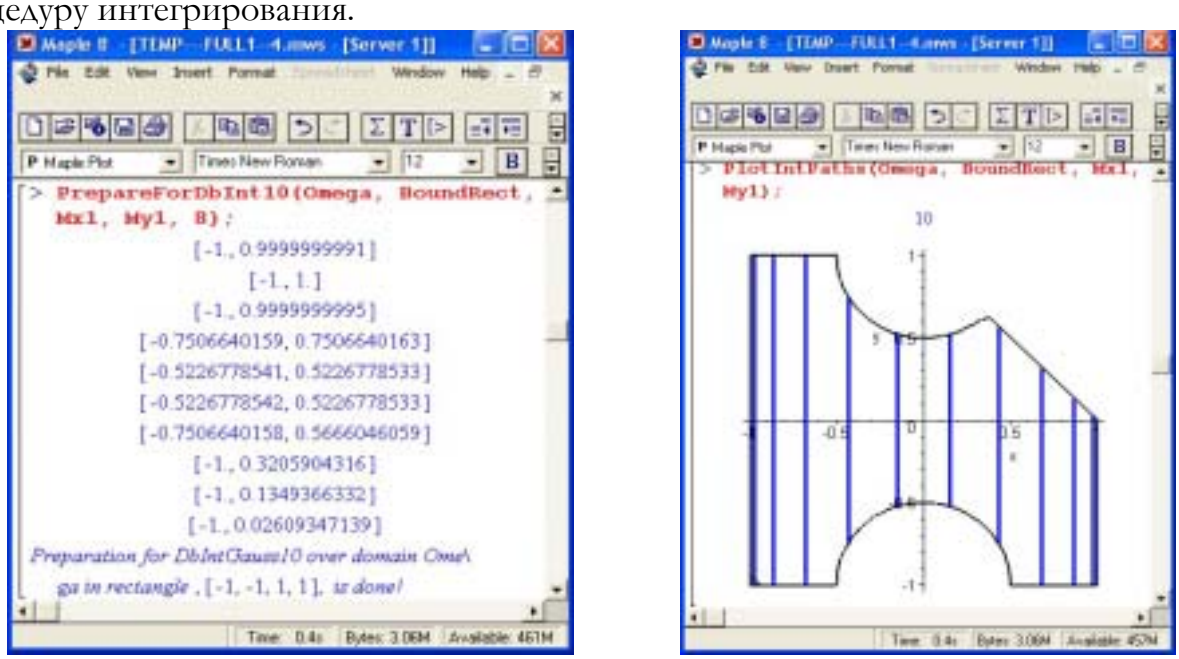


Рисунок 6 – Шаг 5. Подготовка области к интегрированию

Шаг 4. Формирование координатных функций. Как было сказано выше, решение поставленной краевой задачи(1,2) сводится к решению задачи о минимуме некоторого функционала(3). В.Л. Рвачевым вводится понятие **структуры решения**. Не вникая в подробности, укажем что структура решения должна удовлетворять краевым условиям задачи и отвечать требованиям полноты. Структуры решения для различных задач широко представлены в литературе. Для данной задачи структура решения:

$$u(x,y) = \Omega(x,y)^2 \cdot \Phi(x,y), \tag{7}$$

где $\Omega(x,y)$ – построенная на шаге 2 функция, описывающая границу исследуемой области; $\Phi(x,y)$ – неопределенная пока компонента решения. Видим, что при любом выборе $\Phi(x,y)$ данная структура удовлетворяет краевым условиям(2) задачи. Действительно, $u(x,y)\big|_{\partial\Omega} = 0$, так как $\Omega(x,y)\big|_{\partial\Omega} = 0$, и второе условие $\left.\dfrac{\partial u(x,y)}{\partial n}\right|_{\partial\Omega} = 0$ также выполняется, поскольку

$$\frac{\partial u(x,y)}{\partial n} = 2\Omega(x,y)\frac{\partial \Omega(x,y)}{\partial n}\Phi(x,y) + \Omega(x,y)^2\Phi(x,y) =$$

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

Задача же о нахождении минимума будет решаться методом Ритца. Решение мы представляем в виде структуры решения(7), где неопределенную компоненту мы будем аппроксимировать обычным двумерным степенным полиномом. Т.е. мы представляем решение в виде

$$u_n(x,y) \approx \Omega(x,y)^2 \cdot \sum_{i=1}^{n} C_i \varphi_i(x,y) = \sum_{i=1}^{n} C_i W_i(x,y), \qquad (8)$$

где $W_i(x,y) = \Omega(x,y)^2 \varphi_i(x,y)$ – **координатные функции**; C_i – неопределенные пока числа.

Для генерации слагаемых полинома нужного порядка предусмотрена специальная процедура (рис. 7). Следующая процедура создает массив координатных функций.

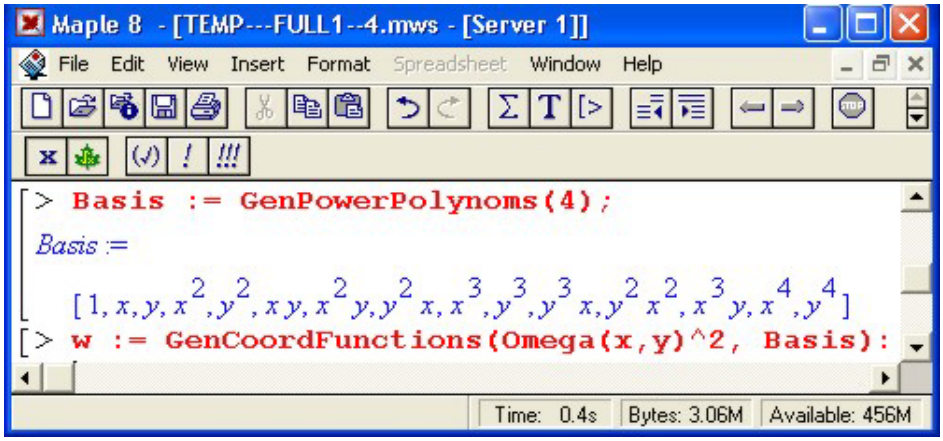


Рисунок 7 – Шаг 4. Формирование координатных функций

Шаг 5. Формирование системы Ритца. Подставляя(8) в функционал(3) и проведя некоторые преобразования получаем, что система Ритца для данной задачи имеет вид:

$$\sum_{i=1}^{n} C_i a_{ij} = b_j \quad (j = \overline{1,n}), \qquad (9)$$

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

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). Далее запускается процедура формирования матрицы правых частей системы Ритца – она вычисляет интегралы по созданной области от конкретного значения общего члена. 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, что и делается в этой процедуре (об этом говорит HF в конце названия процедуры).

Далее мы вводим модуль упругости E, толщину пластины h и коэффициент Пуассона ν. Вычисляем цилиндрическую жесткость D (она входит в правую часть системы Ритца b_i), записываем общий член и проводим интегрирование (рис. 8).

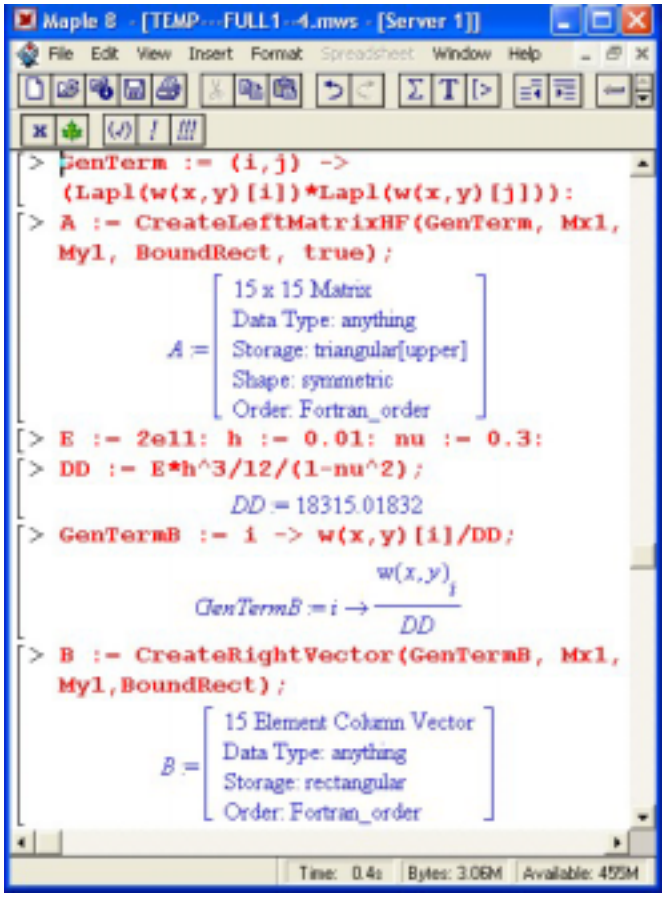
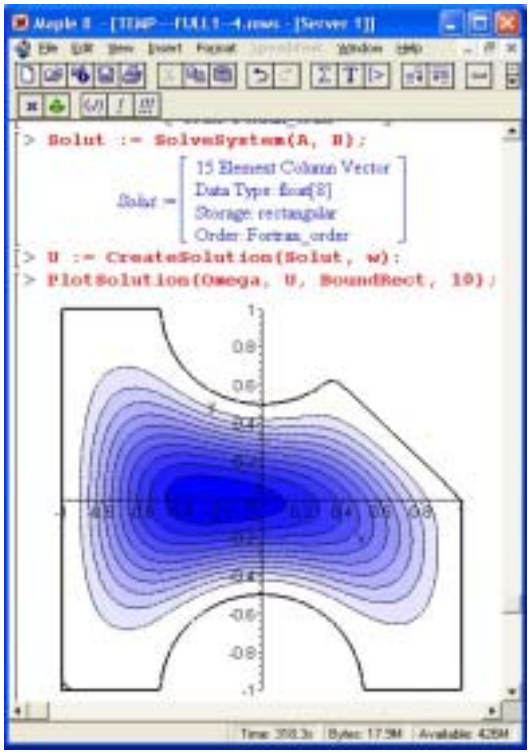


Рисунок 8 – Шаг 5. Формирование системы Ритца

Шаг 6. Решение системы Ритца и вывод результатов. На предыдущем шаге мы получили систему линейных неоднородных уравнений относительно коэффициентов C_i. На этом шаге выполняется решение этой системы. Из полученного вектора решений формируется следующей процедурой непосредственно аналитическое решение. Далее мы можем изобразить графически полученное решение – построить линии уровня (рис. 9).



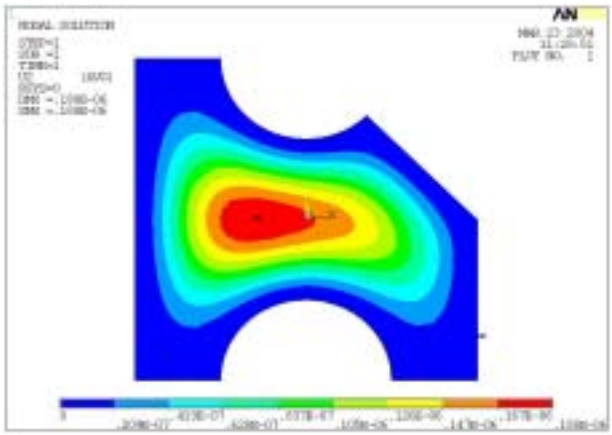


Рисунок 9 – Шаг 6. Решение системы Ритца и вывод результатов (*слева*). Решение задачи на ANSYS 6.0 (*справа*)

Сравним полученное аналитическое решение с численным, полученным на ANSYS 6.0. Численное решение на ANSYS было проведено с использованием конечного элемента **SHELL93**. Это 8-ми узловой 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 1. 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.

Пример 2

Решить задачу об распределении температуры в трехмерном теле, сечение которого изображено на рис. 10. Геометрические характеристики указаны на рисунке в метрах. Коэффициент теплопроводности 0.02 Вт/мК. На границе тела известна температура: 0°С. Мощность внутренних источников постоянна, и равна 0.02 Дж/м^{3}с.

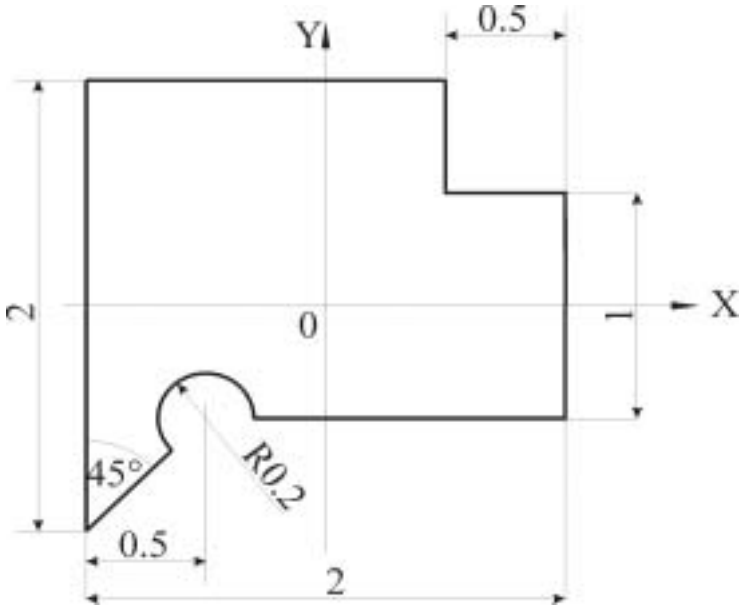


Рисунок 10 – Сечение исследуемого тела

Математическая постановка задачи. Так как при заданных температурных нагрузках температурное поле $T = T(x,y)$ не зависит от z, задачу можно решать в двумерной постановке. Решить уравнение Пуассона

$$\Delta T(x,y) = \frac{q(x,y)}{\lambda}, \tag{10}$$

где $T(x,y)$ – температура; $q(x,y)$ – мощность внутренних источников; λ – коэффициент теплопроводности. Здесь $q(x,y) = 0.02$, $\lambda = 0.02$.

Задача дополняется однородными краевыми условиями Дирихле:

$$T(x,y)\big|_{\partial\Omega} = 0, \tag{11}$$

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

При данных краевых условиях, эта задача сводится к эквивалентной вариационной задаче нахождения минимума функционала следующего вида

$$F(T) = \iint\limits_{\Omega} \left[\left(\vec{\nabla} T(x,y) \right)^2 - 2\frac{q(x,y)}{\lambda} T(x,y) \right] d\Omega, \tag{3}$$

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

В этом примере не будет проводить пошаговое решение, а укажем только отличия от предыдущего примера, так как алгоритм решения такой же.

Опорными для данной области будут следующие области:

$\Omega_1 = \left[F1 = \frac{1}{2}(1 - y^2) \geq 0 \right]$ – полоса, параллельная оси X полушириной 1;

$\Omega_2 = \left[F2 = \frac{1}{2}(1 - x^2) \geq 0 \right]$ – полоса, параллельная оси Y полушириной 1;

$\Omega_3 = \left[F3 = 0.5^2 - (y-1)^2 \geq 0 \right]$ – полоса, параллельная оси X полушириной 0.5 и поднятая относительно оси на 1;

$\Omega_4 = \left[F4 = 0.5^2 - (x-1)^2 \geq 0 \right]$ – полоса, параллельная оси Y полушириной 0.5 и сдвинутая вправо относительно оси на 1;

$\Omega_5 = \left[F5 = 2.5\left(0.2^2 - (x+0.5)^2 - (y+0.5)^2\right) \geq 0 \right]$ – круг радиуса 0.2 с центром в точке (-0.5,-0.5);

$\Omega_6 = \left[F6 = \frac{2(0.25 - 0.5y)}{\sqrt{y + y^2 + 1.25}} \geq 0 \right]$ – полуплоскость, проходящая через точки (0,-0.5) и (-0.5,-0.5);

$\Omega_7 = \left[F7 = \frac{x - y}{\sqrt{-2xy + x^2 + y^2 + 2}} \geq 0 \right]$ – полуплоскость, проходящая через точки (-0.5,-0.5) и (-1,-1).

Логическая формула всей области:

$$\Omega = \left(\left((\Omega_1 \wedge \Omega_2) \wedge \neg(\Omega_3 \wedge \Omega_4) \right) \wedge \neg\Omega_5 \right) \wedge \neg(\Omega_6 \wedge \Omega_7).$$

В «аналитике» получаем, заменяя логические операции R-операциями системы $\Re_0$:

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

Структура решения этой задачи: $u(x,y) = \Omega(x,y) \cdot \Phi(x,y)$.

Координатные функции имеют вид: $W_i(x,y) = \Omega(x,y) \cdot \varphi_i(x,y)$.

Общие члены матрицы Ритца:

$$a_{ij} = a_{ji} = \iint\limits_{\Omega} \left(\vec{\nabla} W_i(x,y) \cdot \vec{\nabla} W_j(x,y)\right) d\Omega;$$

$$b_i = \iint\limits_{\Omega} \frac{q(x,y)}{\lambda} W_i(x,y) d\Omega.$$

На рис. 11 показано полученное решение.

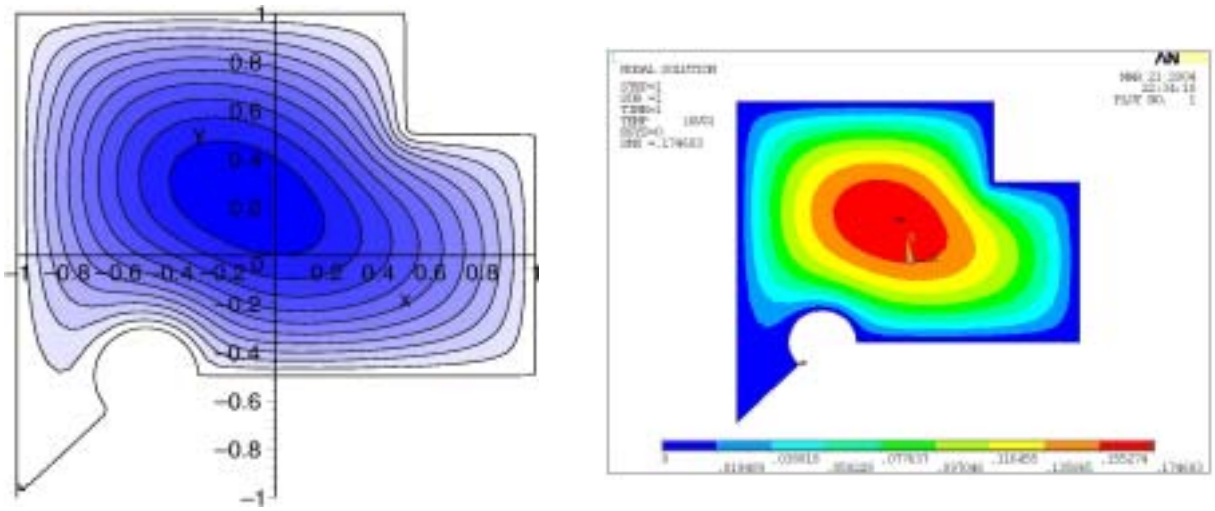


Рисунок 11 – Аналитическое решение на Maple (*слева*).
Численное решение задачи на ANSYS 6.0 (*справа*)

Сравним полученное аналитическое решение с численным, полученным на ANSYS 6.0. Численное решение на ANSYS было проведено с использованием конечного элемента **PLANE35**. Это 6-ти узловой 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.

Таблица 2. 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.

Заключение

В работе была реализована программа для автоматизации аналитического решения краевых задач математическое 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 Ритца 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) интергированность 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.

Приложение 1. Список реализованных процедур

Процедуры, связанные с заданием опорных областей[1]		
№	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 астроиды
8.	**OvalKassini**	уравнение овала Кассини
Процедуры, связанные с R-операциями[2]		
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»

[1] Пользователь может аналитически задать любую, нужную ему область
[2] Пользователь может использовать и другие R-системы
[3] Процедура реализована для отладки процедур интегрирования и подготовки к нему

24.	**PrepareForDbInt10M1**	подготавливает область к интегрирования 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 – форуму Гаусса нужной точности
27.	**DbIntGauss10hf**	вычисление двойного интеграла по области от заданной функции, использует 10-ти точечную формулу интегрирования 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

[4] Использования машинной арифметики очень сильно ускоряет процесс интегрирования, однако работает не со всеми видами функций, поэтому и были оставления 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 работаем очень медленно, так как использует в основном 30-ти точечную формулу Гаусса-Кронрода

		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 корны – поиск с равным шагом
Процедуры генерации полных последовательностей функций		
40.	**GenPowerPolynoms**	генерация степенных полиномов заданной точности[6]
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

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

Приложение 2. Листинг написанной программы

```
> protect(x,y);
> nStripX := proc(DisplY::numeric, Half-
Width::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, Half-
Width::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*Y1^2
+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)^2)^(1/2);
 end if;
end proc;
> Ellipse := proc(DisplX::numeric,
DisplY::numeric, HalfAxeX::numeric, Hal-
fAxeY::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) + HalfWidth^(2/3);
end proc;
> OvalKassini := proc(DisplX::numeric,
DisplY::numeric, HalfCenter::numeric, Parame-
ter::numeric)
 global x,y;
 if HalfCenter=0 then error "zero positions of cen-
ters of oval"
  elif HalfCenter>=Parameter then error "pa-
rameter must be greater then position of center";
 end if;
 (x,y) -> -((x - DisplX)^2 + (y - DisplY)^2 + Half-
Center^2)^2 + 4*HalfCenter^2*(x - DisplX)^2 +
Parameter^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)
 Vector([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, Bound-
ary::list)

plots[implicitplot](Domain(x,y),x=Boundary[1]..B
oundary[3],
y=Boundary[2]..Boundary[4],numpoints=1000,
scaling=constrained, color=black);
end proc;
> PlotDomain3d := proc(Domain::name, Bound-
ary::list, zmax::numeric)
 plot3d(Domain(x,y),x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4], scal-
ing=constrained,view=0..zmax,axes=boxed,
grid=[50,50]);
end proc;
> PlotIntPaths := proc(Domain::name, Bound-
ary::list, MeshX::table, MeshY::table)
 local i, j, k, pp;
 pp[1] :=
plots[implicitplot](Domain(x,y),x=Boundary[1]..B
oundary[3],
y=Boundary[2]..Boundary[4],numpoints=1000,
scaling=constrained, thickness=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]],[Me
shX[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, Bound-
ary::list, MeshX::table, MeshY::table)
 local i, j, k, pp, SetColor;
 pp[1] :=
plots[implicitplot](Domain(x,y),x=Boundary[1]..B
oundary[3],
y=Boundary[2]..Boundary[4],numpoints=1000,
scaling=constrained, thickness=2, color=black):
 print(nops([indices(MeshX)]));
 k := 2;
 SetColor := x -> piece-
wise(x=3,blue,x=5,red,x=8,green,yellow);
 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]],[Me
shX[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, Solu-
tion::name, Boundary::list, ContNumber::posint)
local f, a, b;
f := piecewise(Domain(x,y)>0,Solution(x,y),0):
a := labelledcontour-
plot(f,x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],filled=true,contours
=ContNumber,coloring=[white,blue], scal-
ing=constrained, grid=[50,50]):
b := implicit-
plot(Domain(x,y),x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4], numpoints=2000,
thickness=2,scaling=constrained, color=black):
plots[display]({a,b});
end proc;
> PlotSolution3d := proc(Domain::name, Solu-
tion::name, Boundary::list, ContNumber::posint)
local f, a, b;
f := piecewise(Domain(x,y)>0,Solution(x,y),0):
a := contourplot3d(f,x=Boundary[1]..Boundary[3],
y=Boundary[2]..Boundary[4],filled=true,contours
=ContNumber,coloring=[white,blue], scal-
ing=constrained, grid=[50,50]):
plots[display]({a});
end proc;
> PrepareForDbInt10 :=
proc(Domain::procedure, Boundary::list,
MeshX::name, MeshY::name, NumOf-
Steps::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 CurrentNodes[i] :=
g*Nodes[i] + p; end do;
 j := 1;
 for i from 1 to 10 do
   Temp := FindRoots(Domain(CurrentNodes[i],y),
evalf(Boundary[2]), evalf(Boundary[4]), NumOf-
Steps, 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];
    Temp := Temp + 2;
  end do;
 end do;
Temp := convert(Domain,string);
print(`Preparation for DbIntGauss10 over do-
main`||` `||`Temp`||` in rectangle `,Boundary,` is
done!`);
end proc;
> PrepareForDbInt10M1 :=
proc(Domain::procedure, Boundary::list,
MeshX::name, MeshY::name, NumOf-
Steps::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 CurrentNodes[i] :=
g*Nodes[i] + p; end do;
 j := 1;
 for i from 1 to 10 do
  Temp := FindRoots(Domain(CurrentNodes[i],y),
evalf(Boundary[2]), evalf(Boundary[4]), NumOf-
Steps, 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 DbIntGauss10M1 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, Inte-
gralsY;
 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]))*Weights[Mesh
X[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, Inte-
gralsY;
 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 IntegralsY[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 IntegralsY[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 IntegralsY[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)))*Weights[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, Inte-
gralsY;
 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(Fu
nc(MeshX[i][1],y)))*Weights[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, Inte-
gralsY;
 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 IntegralsY[k] := Int-
Gauss3hf(MeshY[i][j][1],MeshY[i][j][2],evalf(Fun
c(MeshX[i][1],y)))*Weights[MeshX[i][2]];
    elif MeshY[i][j][3]=5 then IntegralsY[k] := Int-
Gauss5hf(MeshY[i][j][1],MeshY[i][j][2],evalf(Fun
c(MeshX[i][1],y)))*Weights[MeshX[i][2]];
elif MeshY[i][j][3]=8 then IntegralsY[k] := Int-
Gauss8hf(MeshY[i][j][1],MeshY[i][j][2],evalf(Fun
c(MeshX[i][1],y)))*Weights[MeshX[i][2]];

    else IntegralsY[k] := Int-
Gauss10hf(MeshY[i][j][1],MeshY[i][j][2],evalf(Fu
nc(MeshX[i][1],y)))*Weights[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, Integr, Func1;
 Nodes := [-0.7745966692, 0., 0.7745966692];
 Weights := [0.5555555558, 0.8888888888,
0.5555555558];
 p := (p2 + p1)/2;
 g := (p2 - p1)/2;
 Integr := 0.0;
 Func1 := unapply(Func,y);
 for i from 1 to 3 do
  Weights[i] := g*Weights[i];
  Nodes[i] := g*Nodes[i] + p;
  Integr := Integr + Func1(Nodes[i])*Weights[i];
  end do;
 Integr;
end proc;
> IntGauss3hf := proc(p1::numeric, p2::numeric,
Func::algebraic)
 local Weights, p, g, Nodes, i, Integr, Func1,
NodesHF;
 Nodes := [-0.7745966692, 0.1e-30, 0.7745966692];
 Weights := [0.5555555558, 0.8888888888,
0.5555555558];
 p := (p2 + p1)/2;
 g := (p2 - p1)/2;
 Integr := 0.0;
 Func1 := 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(Func1(NodesHF[i]))*Weights[i];
  end do;
 Integr;
end proc;
> IntGauss5 := proc(p1::numeric, p2::numeric,
Func::algebraic)
 local Weights, p, g, Nodes, i, Integr, Func1;
 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;
 Func1 := unapply(Func,y);
 for i from 1 to 5 do
  Weights[i] := g*Weights[i];
  Nodes[i] := g*Nodes[i] + p;
  Integr := Integr + Func1(Nodes[i])*Weights[i];
  end do;
 Integr;
end proc;
> IntGauss5hf := proc(p1::numeric, p2::numeric,
Func::algebraic)
```

```
 local Weights, p, g, Nodes, i, Integr, Func1,
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;
 Func1 := 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(Func1(NodesHF[i]))*Weights[i];
   end do;
 Integr;
end proc;
> IntGauss8 := proc(p1::numeric, p2::numeric,
Func::algebraic)
 local Weights, p, g, Nodes, i, Integr, 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, Integr, 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, Integr, 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);
 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, Integr, 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, NumOfNodes::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))):
 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-xl*z:
Nodes[NumOfNodes+1-i] := xm+xl*z:
Weights[i] := 2*xl/((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, xl, 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
   xl := x1;
   xr := x2;
 else
   xl := 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-xl)*df-f)>0
or abs(2*f)>abs(dxold*df) then
     dxold := dx;
     dx := (xr-xl)/2.0;
     GuessRoot := xl + dx;
     if xl=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 GuessRoot; end if;
   f := evalf(Func(GuessRoot));
   df := evalf(Deriv(GuessRoot));
   if f<0 then xl := 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, NumOfSteps::posint, Eps::numeric)
 local roots, Func, Deriv,step, position, flag, i,
Func_pos, Func_pos_old, temp;
 #option trace;
 Func := unapply(Expr,y);
 Deriv := unapply(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 := NewtonRaphsonBisection(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 := NewtonRaphsonBisection(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 := combinat[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,
BasisFuncList::list)
 unapply(map(proc(Temp) Domain*Temp end
proc,BasisFuncList),x,y);
end proc;
> CreateLeftMatrix :=
proc(GenericMember::procedure, MeshX::table,
MeshY::table, Boundary::list, Symme-
try::boolean)
local i,j,Temp, A;
if Symmetry=true then
  A := Ma-
trix(NumberOfBasisFunc,shape=symmetric):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to i do
      Temp := unapply(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 NumberOfBasisFunc do
      Temp := unapply(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, Boundary::list, Symme-
try::boolean)
local i,j,Temp, A;
if Symmetry=true then
  A := Ma-
trix(NumberOfBasisFunc,shape=symmetric):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to i do
      Temp := unapply(GenericMember(i,j),x,y);
      A[i,j] := DbIntGauss10M1(MeshX, MeshY,
Temp, Boundary):
    end do;
  end do;
else
  A := Matrix(NumberOfBasisFunc):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to NumberOfBasisFunc do
      Temp := unapply(GenericMember(i,j),x,y);
      A[i,j] := DbIntGauss10M1(MeshX, MeshY,
Temp, Boundary):
    end do;
  end do;
end if;
return A;
end proc;
> CreateLeftMatrixHF :=
proc(GenericMember::procedure, MeshX::table,
MeshY::table, Boundary::list, Symme-
try::boolean)
local i,j,Temp, A;
#options trace;
if Symmetry=true then
  A := Ma-
trix(NumberOfBasisFunc,shape=symmetric):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to i do
      Temp := unapply(GenericMember(i,j),x,y);
#print(i);
#print(j);
      A[i,j] := DbIntGauss10hf(MeshX, MeshY,
Temp, Boundary):
#print(A[i,j]);

    end do;
  end do;
else
  A := Matrix(NumberOfBasisFunc):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to NumberOfBasisFunc do
      Temp := unapply(GenericMember(i,j),x,y):
      A[i,j] := DbIntGauss10hf(MeshX, MeshY,
Temp, Boundary):
    end do;
  end do;
end if;
return A;
end proc;
> CreateLeftMatrixM1hf :=
proc(GenericMember::procedure, MeshX::table,
MeshY::table, Boundary::list, Symme-
try::boolean)
local i,j,Temp, A;
#options trace;
if Symmetry=true then
  A := Ma-
trix(NumberOfBasisFunc,shape=symmetric):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to i do
      Temp := unapply(GenericMember(i,j),x,y);
      A[i,j] := DbIntGauss10M1hf(MeshX, MeshY,
Temp, Boundary):
    end do;
  end do;
else
  A := Matrix(NumberOfBasisFunc):
  for i from 1 to NumberOfBasisFunc do
    for j from 1 to NumberOfBasisFunc do
      Temp := unapply(GenericMember(i,j),x,y);
      A[i,j] := DbIntGauss10M1hf(MeshX, MeshY,
Temp, Boundary):
    end do;
  end do;
end if;
return A;
end proc;
```

```
> CreateRightVector :=
proc(GenericMember::procedure, MeshX::table,
MeshY::table, Boundary::list)
local i,Temp, B;
B := Vector(NumberOfBasisFunc):
for i from 1 to NumberOfBasisFunc do
   Temp := unapply(GenericMember(i),x,y);
   B[i] := DbIntGauss10(MeshX, MeshY, Temp,
Boundary):
end do;
return B;
end proc;
> CreateRightVectorM1 :=
proc(GenericMember::procedure, MeshX::table,
MeshY::table, Boundary::list)
local i,Temp, B;
B := Vector(NumberOfBasisFunc):
for i from 1 to NumberOfBasisFunc do
   Temp := unapply(GenericMember(i),x,y);
   B[i] := DbIntGauss10M1(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, Co-
ordFunc::procedure)
local i;
unap-
ply(add(Solution[i]*CoordFunc(x,y)[i],i=1..Numbe
rOfBasisFunc),x,y);
end proc;
```