МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра математического моделирования и управления

КОМОДЕЙ
Владислав Геннадьевич

ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧИ СОПРЯЖЕНИЯ УРАВНЕНИЙ ГИПЕРБОЛИЧЕСКОГО И ПАРАБОЛИЧЕСКОГО ТИПОВ МЕТОДОМ КОНЕЧНЫХ ЭЛЕМЕНТОВ

Дипломная работа

Руководитель:
кандидат физ.-мат. наук,
доцент Лемешевский С.В.

Допущен к защите
«____» _____________ 2017 г.
Зав. кафедрой,
кандидат физ.-мат наук,
доцент, Белько В.И.

Минск, 2017

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа, 23 стр., 9 рис., 2 источника, 1 приложение

Ключевые слова: ДИФФЕРЕНЦИАЛЬНОЕ УРАВНЕНИЕ, ВАРИАЦИОННАЯ ПОСТАНОВКА, КОНЕЧНЫЕ ЭЛЕМЕНТЫ, FENICS, PYTHON.

Объект исследования — задача сопряжения уравнений гиперболического и параболического типов.

Цель работы — численное решение задачи сопряжения уравнений гиперболического и параболического типов

Методы исследования — метод конечных элементов.

Результатами являются: вычислительный алгоритм и программа решениия задачи сопряжения уравнений гиперболического и параболического типов

Область применения — приближенное решение дифференциальных уравнений, математическое моделирование процессов, протекающих в разнородных средах.

РЭФЕРАТ

Дыпломная работа, 23 с., 9 мал., 2 крыніцы, 1 дадатак.

Ключавыя словы ДЫФЕРЕНЦЫАЛЬНАЕ РАУНАННЕ, ВАРЫЯЦЫЕННАЯ ПАСТАНОУКА, КАНЧАТКОВЫЯ ЭЛЕМЕНТЫ, FENICS, PYTHON.

Аб'ект даследавання — задача аб спалучэнні гіпербалічнага і парабалічнага раунання.

Мэта работы — выліковае рашэнне задачы аб спалучэнні гіпербалічнага і парабалічнага раунання.

Метады даследавання — метад канчатковых элементау.

Вынікамі з'яуляюцца: выліковы алгарытм і праграма рашэння задачы аб спалучэнні гіпербалічнага і парабалічнага раунання.

Вобласць прымяненя — прыблізнае рашэнне дыферэнцыяльных раунанняу, матэматычнае мадэляванне працэсау якія праходзяць у разнастайных ассяроддзях.

SUMMARY

Graduate work. 23 p., 9 pic., 2 sources, 1 appendix

Key words: DIFFERENCIAL EQUATION, VARIATION FORMULATION, FINITE ELEMENTS, FENICS, PYTHON.

Research object: problem about hyperbolic and parabolic equation conjugation.

Work goal: numerical solution of hyperbolic and parabolic equation conjugation problem.

Research methods – finite element method.

Results – computational algorithm and program for solving hyperbolic and parabolic equations conjugation problem.

Use area – approximate solutions of differential equations, mathematical modeing of processing in various media.

ВВЕДЕНИЕ

Необходимость рассмотрения сопряжения, когда на одной части области задано уравнение параболического типа, а на другой – уравнение гиперболического типа, была впервые высказана И.М. Гельфандом в 1959 г[1]. Наприме, к задаче сопряжения приводит изучение электрических колебаний в проводах.

Такого рода задачи встречаются также при изучении движения жидкости в канале, окруженной пористой средой, в теории рапространения электромагнитных полей и в ряде других областей физики. Так, в канале гидродинамическое давление жидкости удовлетворяет волновому уравнению, а в пористой среде – уравнению фильтрации, которое в данном случае совпадает с уравнением диффузии[2]. При этом на границе канала выполняются некоторые условия сопряжения. Аналогичная ситуация имеет место для магнитной напряженности электромагнитного поля в указанной выше неоднородной среде[3]. Большой интерес представляет изучение влияния вязкоупругих свойств нефти на различные технологические процессы ее добычи. Если рассмотреть совместное движение различных несмешивающихся жидкостей в трещинах и пористых пластах с учетом вязкоупругих характеристик, то движение вязкоупругой и вязкой жидкостей в плоской горизонтальной трещине без учета поверхностных явлений описывается одномерным гиперболическим уравнением и уравнением теплопроводности с интегро-дифференциальными условиями на границе раздела движущихся жидкостей.

В настоящей работе решаются задача о сопряжении гиперболического и параболического уравнений по пространственной переменной в конечных областях с помощью метода конечных элементов. Решение находится с помощью пакета вычислительной математики FEniCS, который предназначен для решения задач с помощью данного метода. Рассматривается погрешность полученного решения и порядок метода конечных элементов

1 Описание постановки задачи сопряжения для уравнения гиперболо параболического типа и методов ее решения

1.1 Постановка задачи сопряжения для уравнения гиперболо параболического типа

Пусть ограниченная область $\Omega \subset \mathbb{R}^2$ с границей $\partial\Omega$ разбивается кривой Γ на две подобласти Ω_1 и Ω_2. В области Ω_2 будем рассматривать уравнение параболического типа, а в Ω_1 – уравнение гиперболического типа по t. На $\partial\Omega$ задаются граничные условия, на Γ – условия сопряжения. Задача формулируется следующим образом, для неизвестной функции u:

$$u(x) = \begin{cases} u_1, & x \in \Omega_1 \\ u_2, & x \in \Omega_2 \end{cases}$$

рассмотрим следующие уравнения:

$$\frac{\partial^2 u_1}{\partial t^2} = div(k_1(x) grad u_1) + f_1(x,t), \quad x \in \Omega_1, \ t > 0, \tag{1}$$

$$\frac{\partial u_2}{\partial t} = div(k_2 grad u_2) + f_2(x,t) \tag{2}$$

Уравнения (1) и (2) дополним граничными условиями Дирихле:

$$u = 0, \quad x \in \partial\Omega. \tag{3}$$

На границе раздела задаются условия сопряжения:

$$(k_1(x) grad u_1, \overline{n}) = (k_2(x) grad u_2, \overline{n}), \quad x \in \Gamma, \tag{4}$$

$$u_1(x,t) = u_2(x,t), \quad x \in \Gamma$$

Также в области Ω_1 задаются начальные условия:

$$u_1(x,0) = \varphi_1(x), \quad x \in \Omega_1 \tag{5}$$

$$\frac{\partial u_1}{\partial t}(x,0) = \psi(x), \quad x \in \Omega_1 \tag{6}$$

$$u_2(x,0) = \varphi_2(x), \quad x \in \Omega_2 \tag{7}$$

Пусть $k(x)$:

$$k(x) = \begin{cases} k_1, & x \in \Omega_1 \\ k_2, & x \in \Omega_2 \end{cases}$$

Известно, что задача (1) - (7) имеет единственное решение

1.2 Введение в метод конечных элементов

Метод конечных элементов(МКЭ) — это численная процедура решения задач, сформулированных в виде дифференциального уравнения или вариационного принципа.[2] МКЭ возник как универсальный метод для решения дифференциальных уравнений. Метод приобрел большую популярность, так как он позволяет анализировать и решать широкий спектр задач.

В отличии от других методов, метод конечных элементов имеент одну особенность. В данном методе аппроксимирующая функция является комбинацией кусочно-гладких конечных функций. Данные функции являются ненулевыми только в определенном интервале(в методе конечных элементов такие интервалы называются конечными элементами, на которые, собственно и разбивается область Ω.

Сам метод конечных элементов включает в себя достаточно много технологий. Процесс построения решения для данного метода включает определенную последовательность шагов. Перечислим эти шаги.

1. Для начала необходимо посторить сетку для нашей области Ω. Для задания коэффициентов уравнений, надо знать определенные свойства материала. Область Ω в нашем случае необходима быть покрыта подобластями, удовлетворяющим свойствам - они все должны быть попарно непересекаемыми [3]. Такие подобласти и называются конечными элементами (КЭ)(Рисунок 1).

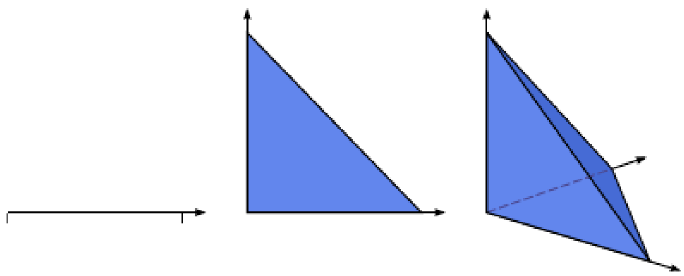

Рисунок 1 – Пример конечно-элементных подобластей для пространств размерности n = 1,2,3

Соответственно, совокупность всех КЭ называется конечно-элементной сеткой[4]. Вершины конечных элементов называются узлами. Узлы бывают двух типов: внешние и внутренние. На границах конечных элементов расположены внешние узлы(они соединяют соседние КЭ). Внутренние узлы конечного элемента используются для более конкретного описания искомых функций[3].

Рассмотрим решение в узле. Части решения в конкретном узле называются степенями свободы. Понятно, что в зависимости от типа задачи число степеней свободы в узле различно. В задаче теплопроводности, например, ищется одно значение температуры(одна степень свободы)[3]. В случае двумерной задачи упругости, число частей решения будет равно двум(т.к. $u = (ux, uy)$. Значения функции в узлах могут фигурировать в качестве степеней свободы[3]. Важно знать материал, который присутствует в задаче. В нашем случае из этого материала сделаны сами конечные элементы.

Как только область разбита на соответствующие подобласти, на каждой такой подобласти можно определить функциональное пространство V и использовать каждое V для определения глобального функционального пространства V_h. Подобласть T вместе с определенным на ней функциональным пространством V называется конечным элементом. Более строгое определение:

- Подобласть T - замкнута, с кусочно-гладкой границей
- Область $V(T)$ - конечное функциональное пространство на T

2. Приведение поиска решения к решению обычной системы линейных уравнений с учетом граничных условий.

3. Решение полученной системы уравнений. После решения данной системы, мы получим коэффициенты. Причем, так как на каждом КЭ мы задали функциональное пространство и базисные функции, то после отыскания коэффициентов решения, наше решение не только будет совпадать в узлах сетки: $u_h(x_i, t) = u(x_i, t) = u_i^t$, а так же являться непрерывным в остальных точках[4].

2 Поиск решения задачи сопряжения гиперболо параболического уравнения

2.1 Вариационная формулировка задачи сопряжения гиперболо параболического уравнения

Для применения метода конечных элементов введем следующие пространства:

$$V = \{u : u_1 : x \in \Omega_1; u_2 : x \in \Omega_2; u = 0, x \in \partial\Omega; \} \tag{8}$$

$$\hat{V} = \{v \in H^1(\Omega), v = 0, x \in \partial\Omega\} \tag{9}$$

Пространство H^1 - пространство Соболева, содержащие такие функции v, что функции v^2 и $||\nabla v||^2$ имеют конечные интегралы по области Ω

На нулевом слое нам известно точное значение функции u. Это значение можно увидеть из начальных условий (5), (7).

Разберемся с первым слоем. Для Ω_1 нам дано (6). Домножим (6) на v с обеих сторон и проинтегрируем по области Ω_1. Соответственно, получим:

$$\int\limits_{\Omega_1} \frac{\partial u_1}{\partial t} v dx = \int\limits_{\Omega_1} \psi v dx$$

Распишем первую производную по t по разностной формуле $\frac{\partial u}{\partial t} = \frac{u^1 - u_0}{\tau}$.

Соответственно, получим:

$$\int\limits_{\Omega_1} \frac{u^1 - u_0}{\tau} v dx = \int\limits_{\Omega_1} \psi v dx \tag{10}$$

Перенесем все части с u^1 в (10) в левую часть, а остальные - в правую. Получим:

$$\int\limits_{\Omega_1} u_1^1 v dx = \tau \int\limits_{\Omega_1} \psi v dx + \int\limits_{\Omega_1} \phi_1 v dx \tag{11}$$

Из (11) выделим следующие члены:

$$a_1(u_1^1, v) = \int\limits_{\Omega_1} u_1^1 v dx \tag{12}$$

$$L_1(v) = \tau \int\limits_{\Omega_1} \psi v dx + \int\limits_{\Omega_1} \phi_1 v dx \tag{13}$$

В литературе a называется билинейной формой, а L - линейной формой.

Так как у нас нет условий на Ω_2 для первого слоя, то необходимо использовать (2). Домножим (2) на v с обеих сторон и проинтегрируем по пространству Ω_2. Получим:

$$\int\limits_{\Omega_2} \frac{u_2^1 - u_2^0}{\tau} v dx = \int\limits_{\Gamma} k_1(\mathrm{grad\,u_1^1, \tilde{n})vdS} - \int\limits_{\Omega_2} \mathrm{k_2(gradu_2^1, gradv)dx} + \int\limits_{\Omega_2} \mathrm{f_2vdx} \tag{14}$$

Для производной u по t использовали разностную подстановку определенную выше.

Таким образом, для первого слоя для Ω_2 получили следующие линейные и билинейные формы:

$$a_2(u_2^1, v) = \int\limits_{\Omega_2} \frac{u_2^1 - u_2^0}{\tau} v dx \tag{15}$$

$$L_2(v) = \int k_1(\mathrm{\ grad\ u_1^1, \tilde{n})vdS} - \int\limits_{\Omega_2} \mathrm{k_2(gradu_2^1, gradv)dx} + \int\limits_{\Omega_2} \mathrm{f_2vdx} \tag{16}$$

Теперь, когда у нас есть все данные для нулевого и первого слоя - распишем задачу для $n = 2, \ldots$.

Вариационная задача строится аналогичным образом: домножаем уравнения (1)-(2) на $v \in V$ с обеих сторон и интегрируем их по пространству. Уравнение (1) интегрируем по пространству Ω_1, уравнение (2) интегрируем по пространству Ω_2.

Для преобразования подынтегральных выражений вида $div(\overline{a})v$ применяем формулу:

$$div(\overline{a})v = div(\overline{a}v) - (\overline{a}, gradv)$$

$$\int\limits_{\Omega_1} \frac{\partial^2 u}{\partial t^2} vdx = \int\limits_{\Gamma} v(k_1 gradu_1, \overline{n})dS - \int\limits_{\Omega_1} (k_1 gradu_1, gradv)dx + \int\limits_{\Omega_1} f_1 vdx$$

Применим начальное условие Дирихле (3) и получим:

$$\int\limits_{\Omega_1} \frac{\partial^2 u}{\partial t^2} vdx = \int\limits_{\Gamma} v(k_1 gradu_1, \overline{n})dS - \int\limits_{\Omega_1} (k_1 gradu_1, gradv)dx + \int\limits_{\Omega_1} f_1 vdx \tag{17}$$

$\overline{n}$ - внешняя нормаль к области Ω_1 от границы Γ. Члены вида $(gradu, \overline{n})$ - производные по направлению от границы. Функция v в литературе называется тестовой функцией(Test Function)[4], u - триальной функцией(Trial Function).

Домножим уравнение (2) на тестовую функцию v с обеих сторон, проинтегрируем полученное равенство по области Ω_2 и применим условие, что $v = 0, x \in \partial\Omega$:

$$\int\limits_{\Omega_2} v\frac{\partial u_2}{\partial t} = \int\limits_{\partial\Omega_2} k_2(x)v(gradu_2,\overline{n})dS - \int\limits_{\Omega_2}(k_2(x)gradu_2, gradv)dx + \int\limits_{\Omega_2} f_2 vdx \quad (18)$$

К уравнению (17) прибавим (18), применяя условие сопряжения (4):

$$\int\limits_{\Omega_1} \frac{\partial^2 u_1}{\partial t^2} vdx + \int\limits_{\Omega_2} \frac{\partial u_2}{\partial t} vdx = -\int\limits_{\Omega} k(x)(gradu, gradv)dx + \int\limits_{\Omega} fvdx \qquad (19)$$

Таким образом, мы получили уравнение (19) для всей области Ω вместо исходных (1), (2) для Ω_1 и Ω_2. Заметим, что (19) равносильно уравнениям (1)-(2).

Произведем дискретизацию по времени. Разобьем отрезок $[0, T]$ на N частей с шагом τ и распишем член $\frac{\partial^2 u}{\partial t^2}$ и $\frac{\partial u}{\partial t}$ по формуле разностной производной:

$$\frac{\partial u}{\partial t} = \frac{u^{n+1} - u^n}{\tau} \qquad (20)$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{u^{n+1} - 2u^n + u^{n-1}}{\tau^2} \qquad (21)$$

Домножим на v и проинтегрируем начальные условия (6). Получим:

$$\int\limits_{\Omega_1} \frac{\partial u_1}{\partial t} vdx = \int\limits_{\Omega_1} \psi vdx \qquad (22)$$

Подставим в (22) выражение (20):

$$\int\limits_{\Omega_1} \frac{u_1^1 - u_1^0}{\tau} vdx = \int\limits_{\Omega_1} \psi vdx$$

После нехитрых преобразований получим конструкцию вида:

$$a(u^1, v) = L(v) \qquad (23)$$

Где:

$$a_1(u^{n+1}) = \int\limits_{\Omega_1} u_1 v dx$$

$$L_2(v) = \int\limits_{\Omega_1} \varphi_1 v dx + \tau \int\limits_{\Omega_1} \psi v dx$$

Таким образом мы получили разложение для Ω_1 на первом слое. Теперь мы должны получить соответствующее разложение для Ω_2. Для этого воспользуемся уравнением (19);

$$\int\limits_{\Omega_2} v dx = \int\limits_{\Gamma} k_1 (gradu_1^1, \overline{n}) v ds - \int\limits_{\Omega_2} k_2 (gradu_2^1, gradv) dx + \int\limits_{\Omega_2} f_2 v_2 dx$$

После преобразований получим окончательное выражение линейной и билинейной формы для первого слоя на Ω_2:

$$a_2(u_2^1, v) = \int\limits_{\Omega_2} v dx + \tau \int\limits_{\Omega_2} k_2 (gradu_2^1, gradv) dx$$

$$L_2(v) = \int\limits_{\Gamma} (gradu_1^1, \overline{n}) v dS + \int\limits_{\Omega_2} v dx v dx$$

Проделаем аналогичные преобразования(перенос членов в левую и правые части) для (19):

$$a(u, v) = \int\limits_{\Omega_2} v dx + \tau \int\limits_{\Omega_2} u^{n+1} v dx + \tau^2 \int\limits_{\Omega} k (gradu^{n+1}, gradv) dx \qquad (24)$$

$$L(v) = \int\limits_{\Omega_1} (2u^n - u^{n-1}) v dx + \tau \int\limits_{\Omega_2} u^n v dx + \tau^2 \int\limits_{\Omega} f v dx \qquad (25)$$

2.2 Построение СЛАУ для задачи сопряжения гиперболо параболического уравнения

Для решения нашей задачи численно, необходимо применить процесс дискретизации по пространству к исходной задаче (1)-(7). Функцию, которую будем находить обозначим u_h.

Далее, область Ω мы разобьем на конечные элементы. В нашем случае мы будем разбивать область на треугольники. Процесс называется триангуляцией. Разобьем область на M конечных элементов и зададим на каждом базисную функцию ϕ_k. Причем:

$$u_h(x) = \sum_{k=1}^{M} u_k^{n+1}\phi_k(x) \tag{26}$$

Где:

$$v(x) = \phi_j \tag{27}$$

Как известно из метода конечных элементов, мы оперируем линейной и билинейной формой.

$$a(u, v) = L(v) \tag{28}$$

Подставим в (28) выражения (26) и (27) соответственно. Получим следующее равенство:

$$a(u^{n+1}, \phi_i) = a(\sum_{k=1}^{M} u_k^{n+1}\phi_k, \phi_i) \tag{29}$$

Известно, что оператор a линеен, то есть:

$$a(\lambda_1 u_1 + \lambda_2 u_2, v) = \lambda_1 a(u_1, v) + \lambda_2 a(u_2, v)$$

Таким образом, перепишем (29):

$$a(u^{n+1}, \phi_i) = \sum_{k=1}^{M} u_k^{n+1} a(\phi_k, \phi_i) \tag{30}$$

Таким образом, получили следующую задачу: $AU = F$, где матрица A состоит из членов $a_{ki} = a(\phi_k, \phi_i)$. Искомый вектор U имеет вид:

$$U = \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ \dots \\ u_M^{n+1} \end{bmatrix}$$

Получили систему, которую, например, можно решить методом LU факторизации.

2.3 Апостериорная оценка погрешности для решения задачи сопряжения гиперболо параболического уравнения

Обозначим за u^n - решение задачи (1)-(7), а u_h^n - наше решение, полученное методом конечных элементов.

В данном разделе исследована погрешность по методу Рунге-Эйткена[4]. Саму погрешность представим в следующем разложении:

$$E_0 = Mh^p \tag{31}$$

В (31) коэффициент M - некая константа, которая определяется методом решения. h - соответственно, шаг дифференцирования. p - порядок метода. В свою очередь - E_0 называется главным членом погрешности.

Распишем следующую величину:

$$||e_h|| = ||u_h^n - u^n|| = ||u_h^n|| + M_1 h^p + O(h^{p+1}) \tag{32}$$

Теперь, вычислим ту же самую разность, но уже с новым шагом kh:

$$||e_{kh}|| = ||u_{kh} - u^n|| = ||u_{kh}|| + M(kh)^p + O((kh)^{p+1}) \tag{33}$$

Где коэффициент пропорциональности k может быть как больше, так и меньше единицы. Коэффициент M будет одинаковым, так как вычисляется одна и та же переменная, одним и тем же методом, а от величины шага M не зависит.

Пренебрегая бесконечно малыми величинами, приравняем (32) и (33):

$$||u_h^n|| + E_0 = ||u_{kh}|| + k^p E_0$$

Откуда найдем главный член погрешности:

$$E_0 = \frac{||u_h^n|| - ||u_{kh}||}{k^p - 1} \tag{34}$$

Формула (34) называется первой формулой Рунге и она позволяет оценить погрешность. Формула (34) имеет большое практическое значение, так как позволяет провести оценку погрешности без изменения алгоритма метода.

В нашем случае неизвестен порядок метода – степень p. Для этого необходимо третий раз вычислить значение величины e_h с шагом k^2h, то есть:

$$||e_h|| = ||u_{k^2h}|| + k^{2p}E_0 \tag{35}$$

Приравния правые части (34) и (35), можем выразить k^p:

$$k^p = \frac{||u_{kh}|| - ||u_{k^2h}||}{||u_h|| - ||u_{kh}||} \tag{36}$$

Прологарфмируя соотношение (36) определим порядок p.

$$p = \frac{ln\left(\frac{||u_{kh}||-||u_{k^2h}||}{||u_h||-||u_{kh}||}\right)}{lnk} \tag{37}$$

3 Вычислительный эксперимент

3.1 Описание пакета FEniCS

Для применения метода конечных элементов будем использовать пакеты FEniCS и FEniCSTools. Пакет FEniCS состоит из большого количества библиотек(написанных на C++, которые транспилируются в python модули, что говорит о довольно быстрой реализации данного пакета) призванных упростить решения различных дифференциальных уравнений. Использование FEniCS подразумевает собой, что пользователь должен иметь абстрактные знания о методе конечных элементов(основной метод, которым FEniCS решает уравнения). Но на самом деле, FEniCS скрывает от пользователя конечную реализацию алгоритма. Таким образом, код приложения весьма лаконичен и понятен человеку, который знает лишь базовые понятия языка программирования python. Несмотря на то, что все пакеты можно установить из репозиториев вашего дистрибутива(как то apt-get install fenics), я все же рекомендую делать это через исходный код самих пакетов. Таким образом мы можем подобрать корректную версию FEniCS и FEniCSTools. Пакет FEniCS предназначен для решения задач методом конечных элементов в различных вариациях. FEniCS базируется на библиотеке Dolfin, которую так же нужно установить. FEniCS позволяет проводить сложные вычисления вводя в программу лишь аналитический вид уравнения практически в "чистом"виде. Так же FEniCS проводит дискретизацию области по пространству различными способами и с различными типами базисных функций(готовую сетку можно отдать FEniCS в входном файле, что дает довольно большой простор для действий). Пакет FEniCSTools используется для экстраполяции функции из подобласти Ω_1 или Ω_2 на всю подобласть Ω. Так же стоит отметить, что пакет FEniCS высокоуровневый и низкоуровневый одновременно, что позволяет настроить работу программы практически под любые действия, не используя при этом множество сторонних библиотек. FEniCS активно разрабатыватся в настоящее время, поэтому практически всегда можно найти решения возникающих проблем или задать вопрос разработчикам данного пакета.

3.2 Построение решения задачи сопряжения гиперболо параболического уравнения с помощью пакета FEniCS

Рассмотрим следующую задачу сопряжения гиперболо-параболического уравнения:

$$\frac{\partial^2 u_1}{\partial t^2} = div(gradu_1) + 20t\sin(\pi xy), \quad x \in \Omega_1,\ t > 0 \tag{38}$$

$$\frac{\partial u_2}{\partial t} = div(k_2 gradu_2, \overline{n}) + 10(t+1)\sin(\pi xy), \quad x \in \Omega_2,\ t > 0 \tag{39}$$

Уравнения (38) и (39) дополним граничными условиями Дирихле:

$$u = 0, \quad x \in \partial\Omega$$

На границе раздела задаются условия сопряжения:

$$(gradu_1, \overline{n}) = (gradu_2, \overline{n}), \quad x \in \Gamma$$

$$u_1(x,t) = u_2(x,t), \quad x \in \Gamma$$

Начальные условия:

$$u_1(x,0) = \sin(\pi x)\sin(\pi y), \quad x \in \Omega_1$$

$$u_2(x,0) = (t+1)\sin(\pi x)\sin(\pi y), \quad x \in \Omega_2$$

$$\frac{\partial u_1}{\partial t}(x,0) = \sin(\pi x)\sin(\pi y), \quad x \in \Omega_1$$

За область возьмем:

$$\Omega = [0,1] \times [0,1]$$

На первом слое решение нам уже известно. На втором слое, мы знаем уравнение для Ω_1, для Ω_2 мы строили основное уравнение. Построим триангуляцию области Ω.

Для начала разобьем область Ω на Ω_1 и Ω_2, с границей сопряжения Γ в точке $x = 0.5$. (Рисунок 2)

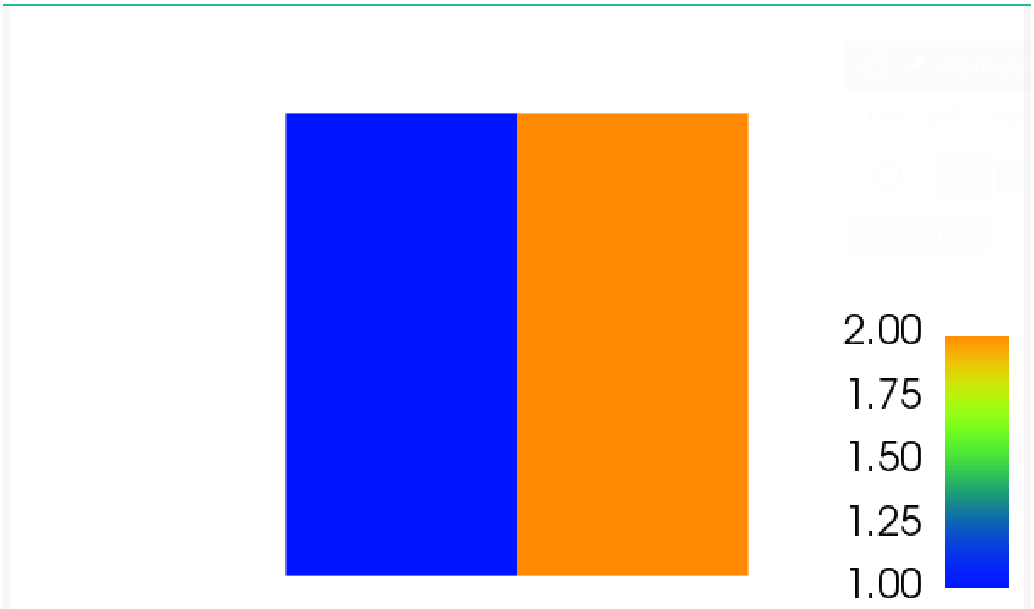

Рисунок 2 – Рассчетная область Ω с обозначенными синим цветом Ω_1, оранжевым - Ω_2

Проведем триангуляцию области $n = 100$ треугольниками и построим решение на первом временном слое, при $t = 0$. (Рисунок 3):

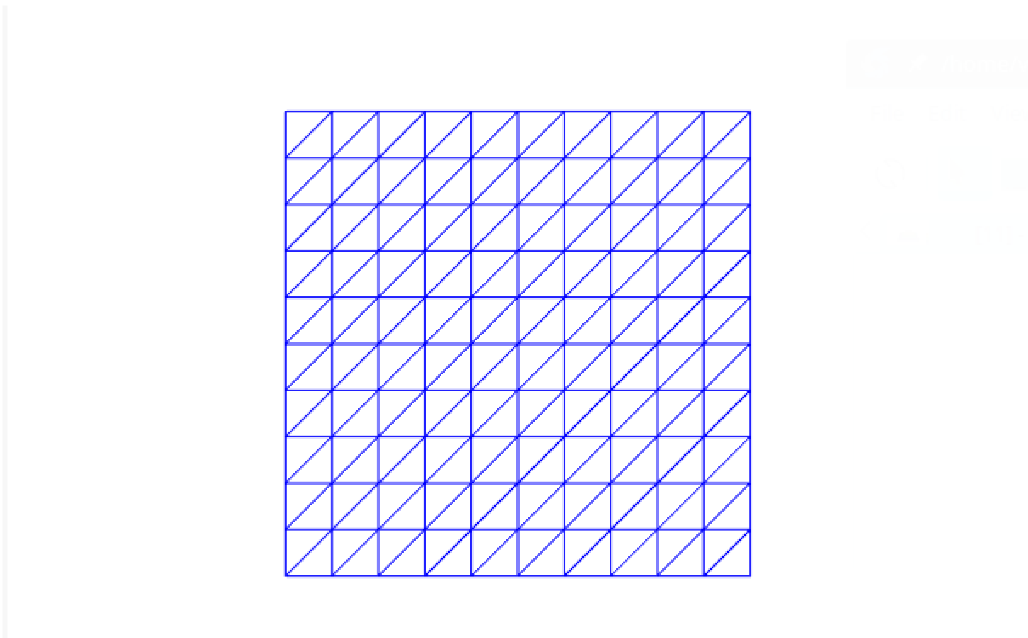

Рисунок 3. – Триангуляция области Ω при $n = 100$

Пример решения на $n = 100$ треугольниках и при $t = 10$ приведен на рисунке 4:

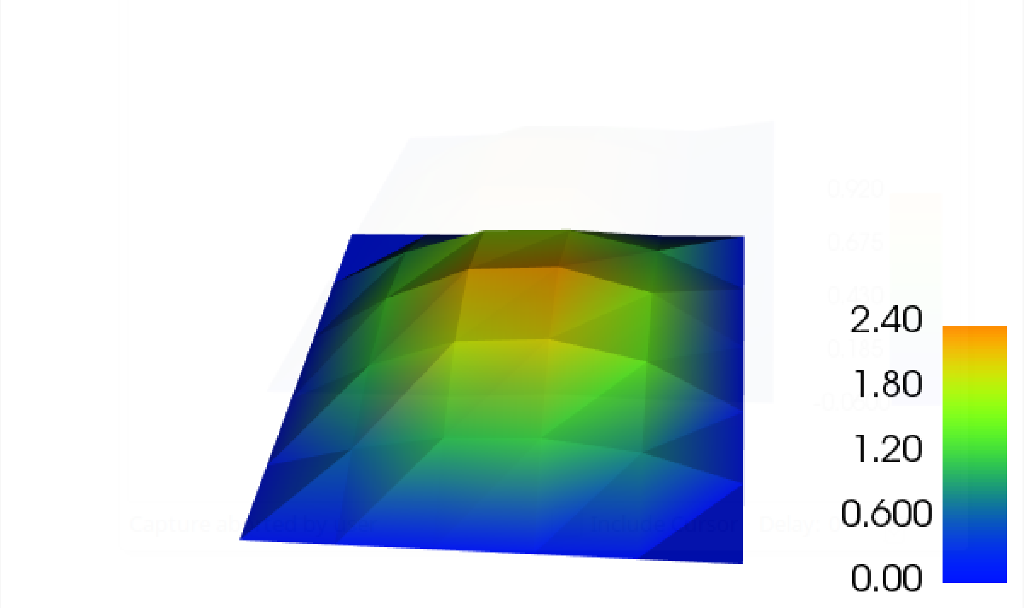

Рисунок 4 – Решение задачи (38)-(39) при $t = 0$

Триангуляция Ω при 400 треугольниках. (Рисунок 5)

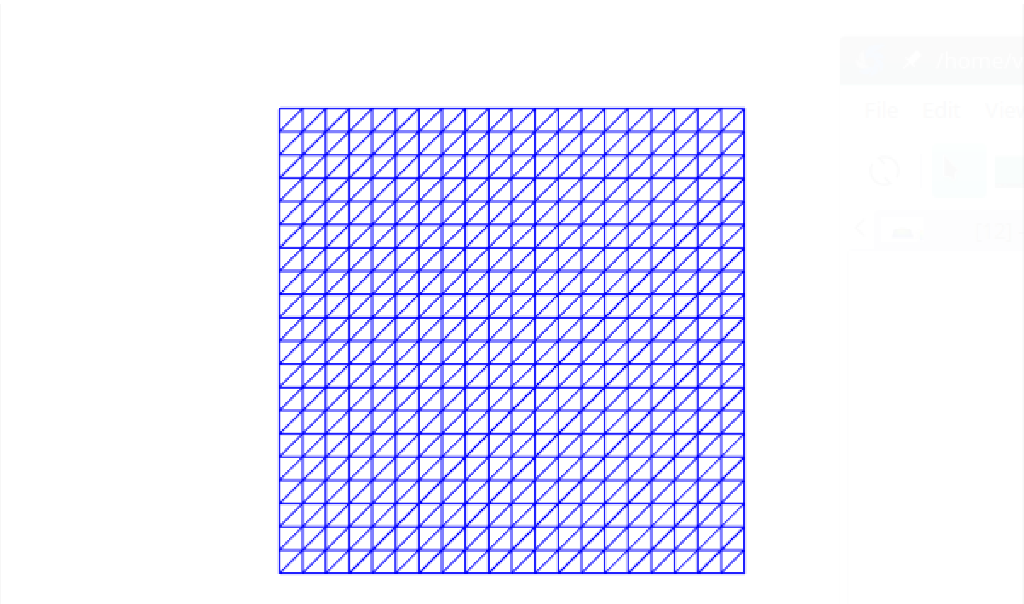

Рисунок 5 – Триангуляция области Ω при $n = 400$

Решение (22)-(28) начальном временном слое. (Рисунок 6)

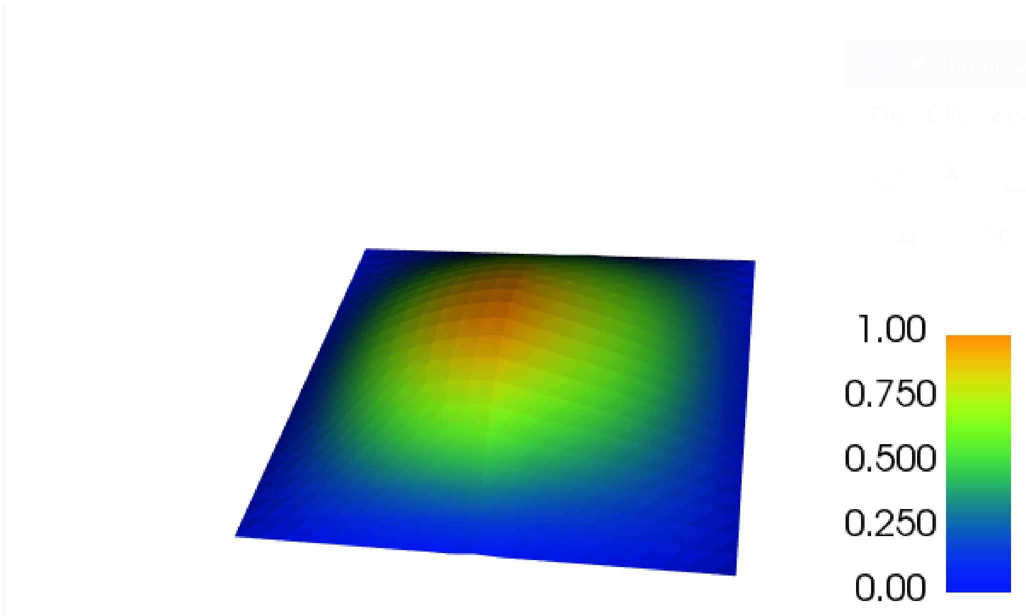

Рисунок 6 – Решение задачи (38)-(39) при $t = 0$, $n = 400$

Решение задачи (22)-(28) при 400 треугольниках и $t = 10$. (Рисунок 7)

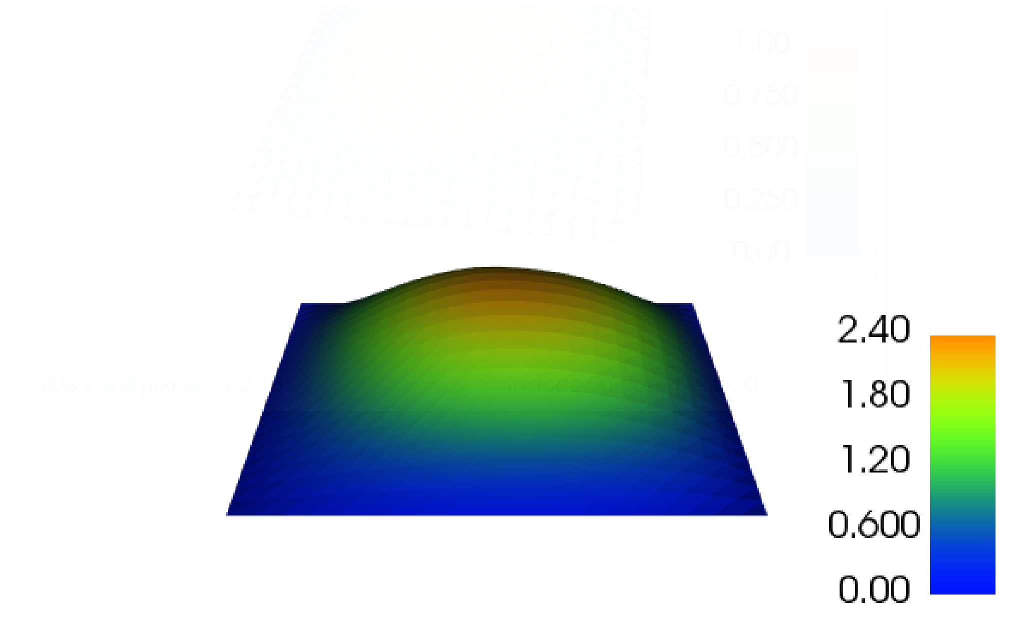

Рисунок 7 – Решение задачи (38)-(39) при $t = 10$, $n = 400$

Решение на $t = 100$, при триангуляции 10000 треугольниками(Рисунок 9). Триангуляция приведедена на рисунке 8:

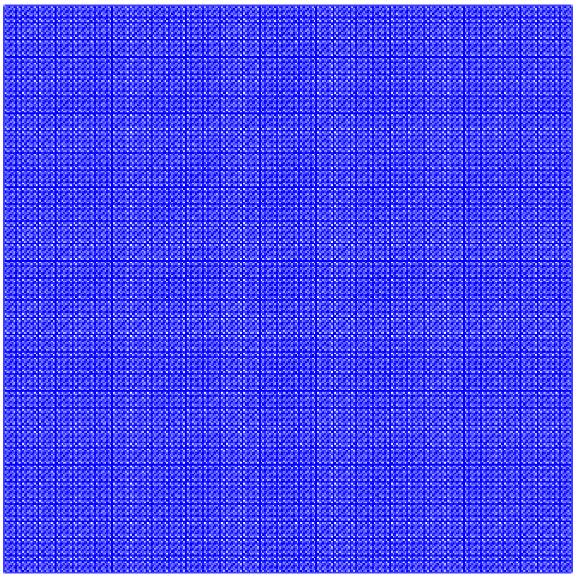

Рисунок 8 – Триангуляция области Ω при $n = 10000$

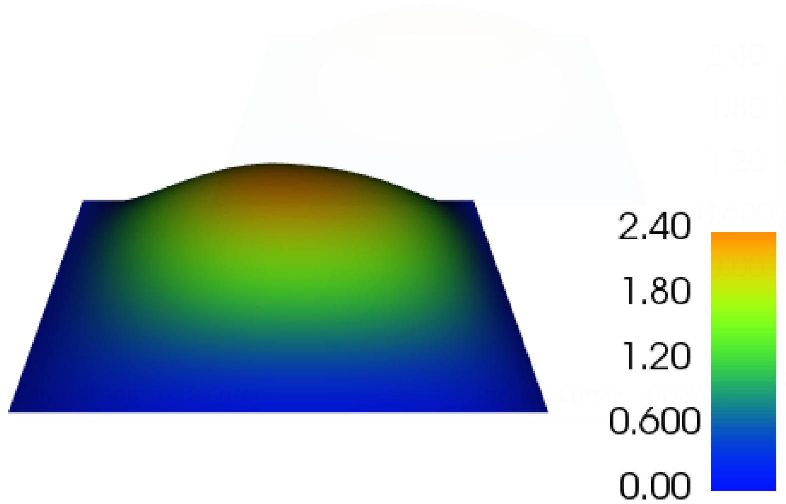

Рисунок 9 – Решение задачи (38)-(39) при $t = 10$, $n = 10000$

Проведем исследование погрешности по формулам (36) и порядок метода (37). Коэффициент k возьмем равным $\frac{1}{2}$, а константу $M = 1$

n треугольников	$\|\|u\|\|$
100	1.847176
400	1.89189
1600	1.903268

Таблица 1 – Норма найденного решения при различных n

Итого получили $E_0 = 0.025$, и порядок метода $p = 2$.

ЗАКЛЮЧЕНИЕ

В работе на основе метода конечных элементов построен вычислительный алгоритм решения задачи сопряжения уравнений гиперболического и параболического типов. Разработана программа, реализующая построенный алгоритм, использующая возможности вычислительного пакета FEniCS. Проведен вычислительный эксперимент, на основе которого получена апостериорная оценка точности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гельфанд, И.И., Некоторые вопросы анализа и дифференциальных уравнений / И.И. Гельфанд // Диф. ур-ия. – 1959. – Т.14, №3. – С. 3-19.

2. Лейбезон, Л.Л. Движение природных жидкостей и газов в пористой среде / Л.Л. Лейбезон., – ОГИЗ, Гостехиздат, 1947. - 244с.

3. Hughes Thomas, J.R. The finite element method: linear static and dynamic finite element analysis / Hughes Thomas J. R., – Dover Publications, 2012. – 704 с.

4. Anders Logg, Automated Solution of Differencial Equations by the Finite Element Method / Anders Logg, – Rent Publications, 2014. – 600 с.

5. Anders Logg, Automated Solution of Differential Equations by the Finite Element Method. The FEniCS book / Anders Logg, Kent-Andre Mardal, Garth N. Wells, – Berlin: Shpringer, 2011. – 720 с.

ПРИЛОЖЕНИЕ A

Программа для решения задачи о сопряжении гиперболического и эллиптического уравнений

```
from __future__ import division
from dolfin import *
import numpy as np
from helpers import *
import sys

n = 10
tau = 0.5
T = 3
# Function definitions
u1_def = "sin(pi*x[0])*sin(pi*x[1])"
phi_def = u1_def
f2_def = "10*(t+1)*sin(pi*x[0]*x[1])"
psi_def = "sin(pi*x[0])*sin(pi*x[1])"
f1_def = "20*t*sin(pi*x[0]*x[1])"
# Defining [0,1]x[0,1] mesh with finite elements of Lagrange type
mesh = UnitSquareMesh(n, n)
V = FunctionSpace(mesh, "Lagrange", 1)

domains = define_domains(mesh, V, n)

# Create submesh and boundaries for resolving equation on Omega_2
mesh2 = SubMesh(mesh, domains, 2)
boundaries = define_mesh2_boundaries(mesh2)


V2 = FunctionSpace(mesh2, "Lagrange", 1)
u_omega1_layer0 = interpolate(Expression(u1_def), V2)

# --- Define measures
ds = Measure("ds")[boundaries]

# -------- define space, finite-element basis function v, and u

u_omega2_layer0 = TrialFunction(V2)
v = TestFunction(V2)
n = FacetNormal(mesh2)
f1 = Expression(f1_def, t=0)
f2 = Expression(f2_def, t=0)
psi = Expression(psi_def)
```

```
u_0 = Expression(u1_def)
# ------ Second layer

u2_1 = TrialFunction(V2)
a = u2_1*v*dx + tau*inner(nabla_grad(u2_1), nabla_grad(v))*dx
# a = inner(nabla_grad(u2_1), nabla_grad(v))*dx
u1_1 = interpolate(Expression("sin(pi*x[0])*sin(pi*x[1])", t=tau), V2)
bcs = [DirichletBC(V2, u1_1, boundaries, 2), DirichletBC(V2, 0, boundaries, 1)]
L =  f2*v*dx + inner(grad(u1_1), n)*v*ds(2)

u2_1 = Function(V2)

solve(a == L, u2_1, bcs=bcs)
u1_1 = interpolate(Expression("sin(pi*x[0])*sin(pi*x[1])", t=tau), V)
u_1 = get_whole_function(V, mesh, u1_1, u2_1, domains)

# plot(u_1)

# ------- Layers 2..T



dx = Measure("dx")[domains]
t = 1
boundaries = define_mesh_boundaries(mesh)

u_n = u_1
u_nm1 = u_0
# u_np1 = TrialFunction(V)
# v = TestFunction(V)

bc = DirichletBC(V, 0, boundaries, 1)
f1 = Expression(f1_def, t=t)

while t <= T:
    u_np1 = TrialFunction(V)
    v = TestFunction(V)
    f1.t = t
    f2.t = t
    # L = f1*v*dx(1) + f2*v*dx(2) + 1/t**2*(2*u_1 - u_0)*v*dx(1)

    u_np1 = Function(V)
    solve(a == L, u_np1, bcs=bc)
    t += tau
    u_nm1 = u_n
    u_n = u_np1
    p = plot (u_np1, title="t={}".format(t), interactive=False)
    p.write_png('t={}'.format(t))
```

```python
print norm(u_np1, 'l2')
plot(u_np1, title="t={}".format(t))
print "t = {}".format(t)

interactive()

from dolfin import *
from fenicstools import interpolate_nonmatching_mesh

def get_whole_function(V, mesh,u1,u2,domains):
    u2 = interpolate_nonmatching_mesh ( u2 , V)
    V_dofmap = V.dofmap()
    chi1 = Function(V)
    chi2 = Function(V)
    gamma_dofs = []
    for cell in cells(mesh): # set the characteristic functions
        if domains[cell] == 1:
            chi1.vector()[V_dofmap.cell_dofs(cell.index())] = 1
            gamma_dofs.extend(V_dofmap.cell_dofs(cell.index()))
        else:
            chi2.vector()[V_dofmap.cell_dofs(cell.index())]=1
    gamma_dofs = list(set(gamma_dofs))
    u2.vector()[gamma_dofs] = 0
    u_0 = project(chi1*u1, V)
    u_0 += project(chi2*u2, V)
    return u_0

def define_domains (mesh, V, cells_num) :
    domains = CellFunction("size_t",mesh, 0)
    domains.set_all(1)
    right_domain = AutoSubDomain (lambda x : x[0] >= 0.5 )
    right_domain.mark(domains, 2)
    return domains
def define_mesh_boundaries ( mesh ) :
    boundaries = MeshFunction ("size_t", mesh, 1)

    boundaries.set_all(0)
    class DirichletBCBoundary(SubDomain):
        def inside(self,x, on_boundary):
            return on_boundary
    d_boundary = DirichletBCBoundary()
    d_boundary.mark(boundaries, 1)
    return boundaries

def define_mesh2_boundaries ( mesh ) :
    boundaries= MeshFunction("size_t", mesh, 1)
    boundaries.set_all(0)
    class DirichletBCBoundary(SubDomain):
        def inside(self, x, on_boundary):
```

```
            return on_boundary and x[0] - 1 < DOLFIN_EPS and x[0] - 0.5
    d_boundary = DirichletBCBoundary()
    d_boundary.mark(boundaries,1)
    class Gamma(SubDomain) :
        def inside(self, x, on_boundary):
            return x[0] > 0.5 - DOLFIN_EPS and x[0] < 0.5 + DOLFIN_EPS
    gamma = Gamma( )
    gamma.mark(boundaries, 2)
    return boundaries

import matplotlib.pyplot as plt
import matplotlib.tri as tri
from dolfin import *
import numpy as np

domain = Rectangle(-1, -1, 1, 1) - Circle(0, 0, 0.5)
mesh = Mesh(domain, 20)
n = mesh.num_vertices()
d = mesh.geometry().dim()

# Create the triangulation
mesh_coordinates = mesh.coordinates().reshape((n, d))
triangles = np.asarray([cell.entities(0) for cell in cells(mesh)])
triangulation = tri.Triangulation(mesh_coordinates[:, 0],
                                  mesh_coordinates[:, 1],
                                  triangles)

# Plot the mesh
plt.figure()
plt.triplot(triangulation)
plt.savefig('mesh.png')

# Create some function
V = FunctionSpace(mesh, 'CG', 1)
f_exp = Expression('sin(2*pi*(x[0]*x[0]+x[1]*x[1]))')
f = interpolate(f_exp, V)

# Get the z values as face colors for each triangle(midpoint)
plt.figure()
zfaces = np.asarray([f(cell.midpoint()) for cell in cells(mesh)])
plt.tripcolor(triangulation, facecolors=zfaces, edgecolors='k')
plt.savefig('f0.png')

# Get the z values for each vertex
plt.figure()
z = np.asarray([f(point) for point in mesh_coordinates])
plt.tripcolor(triangulation, z, edgecolors='k')
plt.savefig('f1.png')
```

```python
# Comment to prevent pop-up
plt.show()

# for ipython notebook
%matplotlib inline


def mesh2triang(mesh):
    xy = mesh.coordinates()
    return tri.Triangulation(xy[:, 0], xy[:, 1], mesh.cells())

def plot(obj):
    plt.gca().set_aspect('equal')
    if isinstance(obj, Function):
        mesh = obj.function_space().mesh()
        if (mesh.geometry().dim() != 2):
            raise(AttributeError)
        if obj.vector().size() == mesh.num_cells():
            C = obj.vector().array()
            plt.tripcolor(mesh2triang(mesh), C)
        else:
            C = obj.compute_vertex_values(mesh)
            plt.tripcolor(mesh2triang(mesh), C, shading='gouraud')
    elif isinstance(obj, Mesh):
        if (obj.geometry().dim() != 2):
            raise(AttributeError)
        plt.triplot(mesh2triang(obj), color='k')

# example
mesh = UnitSquareMesh(10, 10)
plt.figure()
plot(mesh)
plt.show()
Q = FunctionSpace(mesh, "CG", 1)
F = interpolate(Expression("x[0]"), Q)
plot(F)
plt.show()
```