МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ ЭЛЕКТРОННЫХ СОСТОЯНИЙ ДОНОРА В ЦИЛИНДРИЧЕСКИ СИММЕТРИЧНОМ ВНЕШНЕМ ПОТЕНЦИАЛЕ

Курсовой проект

Озолиня Алексея
Николаевича
студента 4 курса,
специальность
«прикладная математика»

Научный руководитель:
магистр физ.-мат. наук,
старший преподаватель
Е.А. Левчук

Минск, 2023

Оглавление

РЕФЕРАТ

Курсовая работа, 31 с., 4 источника, 3 рис., 8 таблиц, 4 прил.

НЕСТАЦИОНАРНОЕ УРАВНЕНИЕ ШРЁДИНГЕРА, СТАЦИОНАРНОЕ УРАВНЕНИЕ ШРЁДИНГЕРА, ВАРИАЦИОННЫЙ МЕТОД РИТЦА

Объект исследования – донор в полупроводнике с внешним затвором.

Цель работы – найти энергию основного состояния донора в полупроводнике с внешним затвором.

Методы исследования – вариационный метод Ритца.

Результаты работы: написана программа, реализующая вариационный метод Ритца для расчёта энергии основного состояния донора в полупроводнике с внешним затвором, найдены оптимальные пробные функции для вариационного метода.

РЭФЕРАТ

Курсавая работа, 31 с., 4 крыніц, 3 мал., 8 табліц, 4 дадаткі.

НЕСТАЦЫЯНАРНАЯ РАЎНАННЕ ШРЭДЫНГЕРА, СТАЦЫЯНАРНАЕ РАЎНАННЕ ШРЭДЫНГЕРА, ВАРЫЯЦЫЙНЫ МЕТАД РЫТЦА

Аб'ект даследавання – донар у паўправадніку з вонкавай засаўкай.

Мэта работы – знайсці энергію асноўнага стану донара ў паўправадніку з вонкавым засаўкай.

Метады даследавання – варыяцыйны метад Рытца.

Вынікі работы: напісана праграма, якая рэалізуе варыяцыйны метад Рытца для разліку энергіі асноўнага стану донара ў паўправадніку з вонкавым засаўкай, знойдзены аптымальныя пробныя функцыі для варыяцыйнага метаду.

SUMMARY

Course project, 31 p., 4 sources, 3 fig., 8 table, 4 applications.

NON-STATIONARY SCHROEDINGER EQUATION, STATIONARY SCHROEDINGER EQUATION, RITZ VARIATIONAL METHOD

Object of research – donor in a semiconductor with an external gate.

Purpose of the work – find the energy of the ground state of the donor in a semiconductor with an external gate.

Methods of research – Ritz variational method

Results of the work: was written a program that implements the Ritz variational method for calculating the energy of the ground state of a donor in a semiconductor with an external gate; optimal trial functions for the variational method are found.

ВВЕДЕНИЕ

В связи с открытиями в области физики и биологии в XX веке возникла потребность вести расчёты в полупроводниках очень малого размера. Такими полупроводниками и стали квантовые точки. Квантовая точка – фрагмент полупроводника, носители заряда в котором ограничены (в пределах 1-100нм) по всем трём измерениям, т.е. находится в трёхмерной потенциальной яме. Массивы таких точек используются в различных устройствах, таких как квантовый компьютер, лазеры, светоизлучающие панели, ячейки солнечных батарей, биологические маркеры. Для многих таких устройств необходимо контролировать взаимодействие электронов соседних доноров или квантовых точек с помощью внешних электрического и магнитного полей.

Таким образом, возникает необходимость изучения влияния магнитного поля и электрического поля, созданного электродами различной конфигурации, как на пару центров доноров, так и на пару квантовых точек.

В данной работе будет рассмотрена следующая структура: имеется пара доноров или пара квантовых точек, расположенные вблизи поверхности полупроводника, на которые воздействуют внешние электрическое и магнитное поля. Однородное магнитное поле направлено перпендикулярно поверхности полупроводника и прямой, соединяющей доноры (квантовые точки). В работе рассмотрено несколько типов потенциалов электрического поля: как однородные, так и неоднородные, записано уравнение Шрёдингера для данной структуры, решено уравнение Шрёдингера и исследовано влияние параметров точки на результаты.

1 Физическая постановка задачи

Рассмотрим структуру, когда D_1 – донор в полупроводнике сдвинутый на z_0 по оси z, J – затвор влияющий на донор D_1 и сдвинутый на x_0 по оси x. Задача не является цилиндрически симметричной, как показано на рисунке 1.

В задаче рассматривалось 2 вида затвора: модельный и тонкий диск.

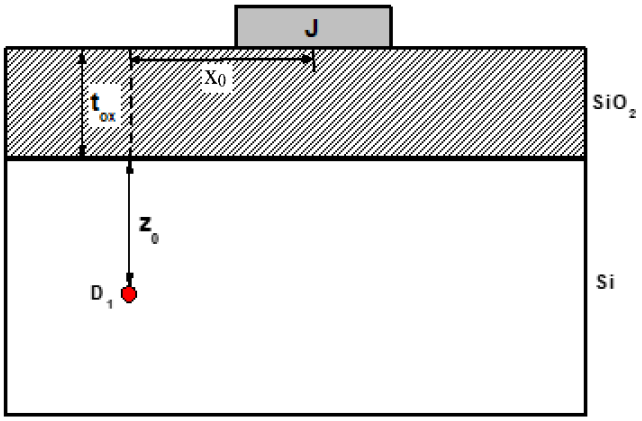

Рисунок 1 – Модель задачи

Переход от нестационарного уравнения Шрёдингера к стационарному

В общем виде нестационарное уравнение Шрёдингера имеет вид:

$i\hbar\frac{\partial}{\partial t}\Psi = \hat{H}(p,q)\Psi$ (1), где $\hat{H}$- гамильтониан, q- координаты, Ψ - волновая функция, ħ - постоянная Планка.

В координатном представлении для точечной частицы массы m, движущейся в потенциальном поле с потенциалом $V(\vec{r},t)$, уравнение принимает следующий вид:

$$i\hbar\frac{\partial}{\partial t}\Psi(\vec{r},t) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\vec{r},t)\right)\Psi(\vec{r},t) \quad (2),$$

где гамильтониан $H = -\frac{\hbar^2}{2m}\nabla^2 + V(\vec{r},t)$ (3)

При раскрытии скобок в формуле (2) уравнение перепишется следующим образом: $-\frac{\hbar^2}{2m}\Delta\Psi(\vec{r},t) + V(\vec{r},t)\Psi(\vec{r},t) = i\hbar\frac{\partial}{\partial t}\Psi(\vec{r},t)$ (5)

Одним из решений уравнения (5) является $\Psi(\vec{r},t) = \Psi(\vec{r})e^{-\frac{iEt}{\hbar}}$ (6)

Подстановка (6) в (5) даст следующее уравнение:

$-\frac{\hbar^2}{2m}\Delta\Psi(\vec{r}) + V(\vec{r})\Psi(\vec{r}) = E\Psi(\vec{r})$ - стационарное уравнение Шрёдингера.

В задаче рассматривались потенциалы для донора вида:

$$\hat{V}_{D_1} = -\frac{2}{\sqrt{x^2+y^2+(z-z_0)^2}} \qquad \hat{V}_{D_1} = \begin{cases} -V_0, & \sqrt{x^2+y^2+(z-z_0)^2} < r_0 \\ 0, & \sqrt{x^2+y^2+(z-z_0)^2} > r_0 \end{cases}$$

А так же потенциалы для затвора вида:

$$\hat{V}_{D_1} = -l((x-x_0)^2 + y^2 + z^2)$$

$$\hat{V}_{D_1} = -\frac{2V_0}{\pi} arctg(\frac{d}{2}\sqrt{\frac{2}{\theta + \sqrt{\theta^2 + d^2 z^2}}})$$

Где $\theta = (x-x_0)^2 + y^2 + z^2$

V_0 - потенциал на диске, d - диаметр диска

2 Математическая постановка задачи

Для того, чтобы задачу можно было решить каким-либо вариационным методом, её необходимо обезразмерить.

$-\frac{\hbar^2}{2m}\Delta\Psi(\vec{r},t) + V(\vec{r},t)\Psi(\vec{r},t) = i\hbar\frac{\partial}{\partial t}\Psi(\vec{r},t)$ - стационарное уравнение Шредингера

Далее в качестве единиц длины использовался эффективный боровский радиус, а энергии – эффективный Ридберг.

В результате обезразмеривания уравнение будет иметь следующий вид:

$$(-\nabla^2 + \hat{V}_{D_1} + \hat{V}_E)\psi = E\psi,\ z > 0$$

В качестве параметров обезразмеривания используется эффективный боровский радиус для длины и эффективный Ридберг для энергий:

$$a^* = \frac{4\pi\varepsilon_0\varepsilon_s\hbar^2}{m^* e^2}; \qquad Ry^* = \frac{m^* e^4}{2\hbar^2\varepsilon_s^2}.$$

3 Вариационный метод Ритца

Вариационный метод является универсальным и может быть использован во всех тех случаях, когда уравнения представимы в вариационной форме. Основа метода состоит в следующем. Искомые решения принадлежат некоторому функциональному пространству *F*; произвольную функцию из этого пространства обозначим Ψ. Предположим, что решения исследуемого уравнения есть функции из *F*, для которых стационарен некоторый функционал Q[Ψ]. Тогда уравнение эквивалентно вариационному уравнению

$$\partial Q = 0$$

Прямой вариационный метод (или метод Ритца) заключается в следующем:

- Пусть требуется найти минимум некоторого функционала *J*(x) с областью определения D_J.
- Выберем координатную систему пробных функций $\varphi_1, \varphi_2, \varphi_3, \ldots, \varphi_n$, удовлетворяющую требованиям:
  - Элементы координатной системы, взятые в любом конечном количестве, линейно независимы;

- Координатная система полна в некоторой метрике, определенной на области D_J;
- При любых значениях постоянных $a_1, a_2, a_3, \ldots, a_n$ элемент $x_n = \sum_{i=1}^{n} a_i \varphi_i$ принадлежит D_J и выражение $J(x_n)$ имеет смысл.
- Рассматривая его как функцию конечного числа переменных $a_1, a_2, a_3, \ , a_n$, найдем те значения при которых $J(x_n)$ достигает минимума. С этой целью необходимо решить следующую систему уравнений:

$$\frac{\partial J(x_n)}{\partial a_i} = 0, \quad i = 1,2,\ldots,n$$

4 Сферически-симметричная задача

Рассматриваем одноэлектронную задачу следующего вида:

$$(-\nabla^2 + \hat{V}_{D_1} + \hat{V}_E)\psi = E\psi,\ z > 0$$

$$\psi\big|_{z=0} = 0; \psi \xrightarrow[|\vec{r}|\to\infty]{} 0$$

Где $\hat{V}_{D_1}$ - потенциал для донора вида:

$$\hat{V}_{D_1} = -\frac{2}{\sqrt{x^2 + y^2 + (z - z_0)^2}}, \qquad \hat{V}_{D_1} = \begin{cases} -V_0, & \sqrt{x^2 + y^2 + (z - z_0)^2} < r_0 \\ 0, & \sqrt{x^2 + y^2 + (z - z_0)^2} > r_0 \end{cases}$$

$\hat{V}_E -$ Оператор внешнего электрического поля

Волновую функцию мы будем брать в виде линейных комбинаций функций с максимумом вблизи донора и также с максимумом вблизи затвора.

Рассмотрим случай для донора, а именно рассмотрим задачу с изолированным донором. Эта задача будет сферически-симметричной:

$$(-\frac{1}{r^2}\frac{d}{dr}(r^2\frac{d\psi}{dr}) + \hat{V}_D)\psi = E\Psi, r > 0$$

$$\left|\psi\big|_{r=0}\right| < \infty, \psi \xrightarrow[|\vec{r}|\to\infty]{} 0$$

$$\hat{V}_D = -\frac{2}{r} \qquad \hat{V}_D = \begin{cases} -V_0, & r < r_0 \\ 0, & r > r_0 \end{cases}$$

Для решения данной сферически-симметричной задачи вариационным методом сначала выбираем оптимальные пробные функции в виде линейных комбинаций гауссиан:

$$\varphi(r) = \sum_{i=1}^{N} e^{-\alpha_i r^2}$$

Наша сферически-симметричная задача сводится к задаче поиска собственных значений:

$$\begin{vmatrix} (A\varphi_1,\varphi_1)-\lambda(\varphi_1,\varphi_1) & (A\varphi_2,\varphi_1)-\lambda(\varphi_2,\varphi_1) & \dots & (A\varphi_n,\varphi_1)-\lambda(\varphi_n,\varphi_1) \\ (A\varphi_1,\varphi_2)-\lambda(\varphi_1,\varphi_2) & (A\varphi_2,\varphi_2)-\lambda(\varphi_2,\varphi_2) & \dots & (A\varphi_n,\varphi_2)-\lambda(\varphi_n,\varphi_2) \\ \dots & \dots & & \dots \\ (A\varphi_1,\varphi_n)-\lambda(\varphi_1,\varphi_n) & (A\varphi_2,\varphi_n)-\lambda(\varphi_2,\varphi_n) & \dots & (A\varphi_n,\varphi_n)-\lambda(\varphi_n,\varphi_n) \end{vmatrix}$$

Где $(A\varphi_i,\varphi_j)=\left\langle \varphi_i \mid \hat{H} \mid \varphi_j \right\rangle$ и $(\varphi_i,\varphi_j)=\left\langle \varphi_i \mid \varphi_j \right\rangle$

Параметры α_i для $\varphi(r)$ выбирались из условия минимизации энергии основного состояния. Минимизация производилась методом Нелдера — Мида.

5 Результаты численного эксперимента

Рассмотрим задачу с затвором. Пробные функции для вариационного метода использовались брались в виде

$$\varphi = xe^{-\alpha x^2}$$

Где α - параметр

В результате получим энергию основного состояние и значение параметров для пробных функций, при которых данное значение энергии достигается. Результаты приведены в таблице 1:

Таблица 1 – Результаты для задачи с затвором

N	E[0],r_0=1	$\alpha_{\text{рез}}$, r_0=1	E[0], r_0=2	$\alpha_{\text{рез}}$, r_0=2
2	-0.9087855603546542	0.226, 1.070	-3.34237622133244	0.311, 0.311
3	-0.9093855141484993	0.072, 0.259, 1.088	-3.41397203228711	$-6.498*10^{-6}$, 0.332, 0.332
4	-0.9219040453965398	0.285, 0.007, 1.427, 3.161	-3.34861817508963	$5.071*10^{-3}$, 0.229, 0.238, 0.229
5	-0.9247679940534397	7.805, 0.291, 1.407, 8.693, 8.166	-3.35022850634923	0.214, -0.633, 0.195, 0.227, 0.097

Где N – количество параметров для пробных функций, E[0] – энергия основного состояния, $\alpha_{\text{рез}}$ – параметры при которых, достигается E[0].

Код программы на языке Python приведён в Приложении А.

Рассмотрим задачу с донором. Пробные функции для вариационного метода использовались брались в виде

$$\varphi = xe^{-\alpha x^2}$$

Где α - параметр

В результате получим энергию основного состояние и значение параметров для пробных функций, при которых данное значение энергии достигается. Результаты приведены в таблице 2:

Таблица 2 – Результаты для задачи с донором

N	E[0]	$\alpha_{\text{рез}}$
2	-0.9716175734403496	1.313, 0.200
3	-0.9905608051528709	0.445, 0.199, 2.302
4	-0.9889755840904845	1.149, 0.160, 1.410, 1.552
5	-0.9940706797362844	2.090, 2.105, 0.088, 1.214, 0.266

Где N – количество параметров для пробных функций, E[0] – энергия основного состояния, $\alpha_{\text{рез}}$ – параметры при которых, достигается E[0].

Код программы на языке Python приведён в Приложении Б.

Рассмотрим задачу с затвором и донором, где будем использовать 2 пробные функции. Потенциал будем рассматривать в виде:

$$\hat{V}_D = -\frac{2}{r} \quad \text{или} \quad \hat{V}_D = \begin{cases} -V_0, & r < r_0 \\ 0, & r > r_0 \end{cases}$$

Пробные функции для вариационного метода использовались вида:

$$\varphi_1 = ze^{(-a(x-x_0)^2-by^2-cz^2)} \qquad \varphi_2 = e^{(-ax^2-by^2-c(z-z_0)^2)}$$

Для функции φ_1 параметры a, b и c выбирались в виде:

$$a = \frac{\sqrt{l}}{2} \qquad b = \frac{\sqrt{l}}{2} \qquad c = \frac{\sqrt{l}}{2}$$

Для функции φ_2 параметры a, b и c выбирались, как результаты минимизации с первой задачи.

Результаты полученной энергии основного состояния для заданных параметров приведены в таблице 3.

Таблица 3 – Результаты для задачи с донором и затвором для двух пробных функций

l	*a*	*b*	*c*	E[0]
0.0001	0.2	0.2	0.2	-0,826
0.5	1.313	0.2	1.313	3.346
1	1.313	0.2	1.313	2.224
0.0001	1.313	0.2	1.313	-0.26

Где *l, a, b, c* – параметры, E[0] – энергия основного состояния.

Рассмотрим задачу с затвором и донором, где будем использовать 3 пробные функции.

Пробные функции для вариационного метода использовались вида:

$$\varphi_1 = ze^{(-a(x-x_0)^2-by^2-cz^2)}$$

$$\varphi_{21} = e^{(-a_1x^2-b_1y^2-c_1(z-z_0)^2)}$$

$$\varphi_{22} = e^{(-a_2x^2-b_2y^2-c_2(z-z_0)^2)}$$

Для φ_1 параметры a, b и c выбирались в виде:

$$a = \frac{\sqrt{l}}{2} \quad b = \frac{\sqrt{l}}{2} \quad c = \frac{\sqrt{l}}{2}$$

Для φ_{21} и φ_{22} параметры a, b и c выбирались, как результаты минимизации с первой задачи.

Результаты полученной энергии основного состояния для заданных параметров приведены в таблице 4.

Таблица 4 – Результаты для задачи с донором и затвором для трех пробных функций

l	a_1	b_1	c_1	a_2	b_2	c_2	E[0]
1	0.2	0.2	0.2	1.313	1.313	1.313	2.76
0.5	0.2	0.2	0.2	1.313	1.313	1.313	-0.18
0.0001	0.2	0.2	0.2	1.313	1.313	1.313	-0.61
1	1.313	1.313	0.2	0.2	0.2	1.313	3.13
0.5	1.313	1.313	0.2	0.2	0.2	1.313	1.15
0.0001	1.313	1.313	0.2	0.2	0.2	1.313	-0.49
1	2.302	0.445	0.199	1.313	0.2	1.313	3.59
0.5	2.302	0.445	0.199	1.313	0.2	1.313	1.29
0.0001	2.302	0.445	0.199	1.313	0.2	1.313	-0.578

Где *l, a, b, c* – параметры, E[0] – энергия основного состояния

Рассмотрим задачу с затвором и донором, где будем использовать 4 пробные функции: две для затвора и две для донора.

Пробные функции для вариационного метода использовались вида:

$$\varphi_1 = ze^{(-a(x-x_0)^2-by^2-cz^2)}$$

$$\varphi_2 = e^{(-a(x-x_0)^2 - by^2 - cz^2)}$$

$$\varphi_{21} = e^{(-a_1x^2 - b_1y^2 - c_1(z-z_0)^2)}$$

$$\varphi_{22} = e^{(-a_2x^2 - b_2y^2 - c_2(z-z_0)^2)}$$

Для φ_1 и φ_2 параметры a, b и c выбирались в виде:

$$a = \frac{\sqrt{l}}{2} \quad b = \frac{\sqrt{l}}{2} \quad c = \frac{\sqrt{l}}{2}$$

Для φ_{21}, φ_{22} параметры α, β выбирались, как результаты минимизации с первой задачи.

Результаты полученной энергии основного состояния для заданных параметров приведены в таблице 5.

Таблица 5 – Результаты для задачи с донором и затвором для четырех пробных функций

l	a_1	b_1	c_1	a_2	b_2	c_2	E[0]
1	0.2	0.2	0.2	1.313	1.313	1.313	3.51
0.5	0.2	0.2	0.2	1.313	1.313	1.313	-2.73
0.0001	0.2	0.2	0.2	1.313	1.313	1.313	-0.61

Где l, α, β – параметры, E[0] – энергия основного состояния

Код программы на языке Python приведён в Приложении В.

Рассмотрим задачу с затвором и донором, где будем использовать 4 пробные функции: одну для затвора и три для донора.

Пробные функции для вариационного метода использовались вида:

$$\varphi_1 = ze^{(-a(x-x_0)^2 - by^2 - cz^2)}$$

$$\varphi_{21} = e^{(-a_1x^2 - b_1y^2 - c_1(z-z_0)^2)}$$

$$\varphi_{22} = e^{(-a_2x^2 - b_2y^2 - c_2(z-z_0)^2)}$$

$$\varphi_{23} = e^{(-a_3x^2 - b_3y^2 - c_3(z-z_0)^2)}$$

Для φ_1 параметры a, b и c выбирались в виде:

$$a = \frac{\sqrt{l}}{2} \quad b = \frac{\sqrt{l}}{2} \quad c = \frac{\sqrt{l}}{2}$$

Для φ_{21}, φ_{22}, φ_{23} параметры $\boldsymbol{\alpha}$, β выбирались, как результаты минимизации с первой задачи.

Результаты полученной энергии основного состояния для заданных параметров приведены в таблице 6.

Таблица 6 – Результаты для задачи с донором и затвором для четырех пробных функций

l	a_1	b_1	c_1	a_2	b_2	c_2	a_3	b_3	c_3	E[0]
1	0.2	0.2	0.2	1.313	1.313	1.313	0.445	0.445	0.445	4.51
0.5	0.2	0.2	0.2	1.313	1.313	1.313	0.445	0.445	0.445	3.23
0.0001	0.2	0.2	0.2	1.313	1.313	1.313	0.445	0.445	0.445	-2.79

Где l, $\boldsymbol{\alpha}$, β – параметры, E[0] – энергия основного состояния

Код программы на языке Python приведён в Приложении Г.

Сравнение с аналитическим решением

После реализации программы, необходимо убедиться, что она работает корректно и результаты не сильно отличаются от аналитического решения.

Для этого рассмотрим 2 случая:

1. При игнорировании в потенциале 1го вида части

$$\hat{V}_{D_1} = -l((x-x_0)^2 + y^2 + z^2)$$

получились значения E[0], приведенные в таблице 7:

Таблица 7 – Результаты для энергии основного состояния при отсутствии 1ой части потенциала

При двух пробных функциях	При трех пробных функциях
-0.837	-0.969

Точным решением в данном случае будет являться значение равное -1.

2. При игнорировании в потенциале 1го вида части

$$\hat{V}_{D_1} = -\frac{2}{\sqrt{x^2 + y^2 + (z-z_0)^2}}$$

получились следующие значения E[0], приведенные в таблице 8:

Таблица 8 – Результаты для энергии основного состояния при отсутствии 2ой части потенциала

При двух пробных функциях	При трех пробных функциях
5	5

Точным решением в данном случае будет выражение $5\sqrt{l}$, где в данном случае $l = 1$.

Полученные результаты свидетельствуют о том, что написанная программа работает корректно.

Код программы на языке Python приведён в Приложении Д.

ЗАКЛЮЧЕНИЕ

- Проведено моделирование энергии донора в поле цилиндрически симметричного затвора с помощью вариационного метода.
- Исследованы различные пробные функции для вариационного метода. С помощью решения сферически симметричной задачи для изолированного донора получены оптимальные параметры для пробных функций.
- Получены зависимости энергии основного состояния донорного электрона от расстояния от донора до границы полупроводника, а также от смещения донора с оси симметрии затвора.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. А. Мессиа Квантовая механика. Т.2 – М. Наука, 1978-1979.
2. Демидович Б. П. Математические основы квантовой механики. – Лань, 2005.
3. Михлин С. Г. Вариационные методы в математической физике. – М. Наука, 1970.
4. В. Смайт Электростатика и электродинамика. – 1954.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Код программы на языке Python

```
import math
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import dia_matrix
from scipy import integrate
from scipy import linalg as LA
from scipy.optimize import minimize
from scipy import optimize
from mpmath import *


def phi(x, alpha):
    return x*np.exp((-1)*alpha*x**2)

def phi_x(r, r0):
    if r < r0:
        return -5
    else:
        return 0

def dr2_phi(x, alpha):
    return 2*alpha*x*np.exp((-1)*alpha*x**2)*(2*alpha*x**2-3)

def fin_func(alpha):
    x0 = 2
    N = len(alpha)
    for i in range(N):
        if alpha[i] <= 0:
            alpha[i] = 0.00000001
    phi_m = np.zeros(shape=(N, N))
    for i in range(N):
        for j in range(N):
            res = integrate.quad(lambda x: phi(x, alpha[i]) * phi(x, alpha[j]),
0, np.inf)[0]
            phi_m[i][j] = (res)

    Aphi_m = np.zeros(shape=(N, N))
    for i in range(N):
        for j in range(N):
            res = (-1) * integrate.quad(lambda x: phi(x, alpha[i]) * dr2_phi(x,
alpha[j]) + phi(x, alpha[i]) * phi_x(x, x0), 0, np.inf)[0]
            Aphi_m[i][j] = (res)

    #вывод
    print("Альфа:")
    print(alpha)
    print("Матрица Aphi")
    print(Aphi_m)
    print("Матрица phi")
    print(phi_m)
    print()
    eigenVal, eigenVectors = LA.eig(Aphi_m, phi_m)
    print("Собственные значения")
    print(eigenVal)
    print("Собственные векторы")
    print(eigenVectors)
```

```
    return min(eigenVal)

alpha = [0.1, 0.2]
result_scipy = minimize(fin_func, alpha, method='Nelder-Mead')
print()
print(result_scipy)
```

ПРИЛОЖЕНИЕ Б

Код программы на языке Python

```
import math
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import dia_matrix
from scipy import integrate
from scipy import linalg as LA
from scipy.optimize import minimize
from scipy import optimize
from mpmath import *

def phi(x, alpha):
    return x*np.exp((-1)*alpha*x**2)

def phi_x(x, alpha):
    return 2*np.exp((-1)*alpha*x**2)

def dr2_phi(x, alpha):
    return 2*alpha*x*np.exp((-1)*alpha*x**2)*(2*alpha*x**2-3)

def fin_func(alpha):
    N = len(alpha)
    for i in range(N):
        if alpha[i] <= 0:
            alpha[i] = 0.00000001
    phi_m = np.zeros(shape=(N, N))
    for i in range(N):
        for j in range(N):
            res = integrate.quad(lambda x: phi(x, alpha[i]) * phi(x, alpha[j]),
0, np.inf)[0]
            phi_m[i][j] = (res)

    Aphi_m = np.zeros(shape=(N, N))
    for i in range(N):
        for j in range(N):
            res = (-1) * integrate.quad(lambda x: phi(x, alpha[i]) * dr2_phi(x,
alpha[j]) + phi(x, alpha[i]) * phi_x(x, alpha[j]), 0, np.inf)[0]
            Aphi_m[i][j] = (res)

    #вывод
    print("Альфа:")
    print(alpha)
    print("Матрица Aphi")
    print(Aphi_m)
    print("Матрица phi")
    print(phi_m)
    print()
    eigenVal, eigenVectors = LA.eig(Aphi_m, phi_m)
    print("Собственные значения")
    print(eigenVal)
    print("Собственные векторы")
    print(eigenVectors)

    return min(eigenVal)

alpha = [0.1, 0.2, 0.3]
result_scipy = minimize(fin_func, alpha, method='Nelder-Mead')
print()
print(result_scipy)
```

ПРИЛОЖЕНИЕ В

Код программы на языке Python

```
import math
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import dia_matrix
from scipy import integrate
from scipy import linalg as LA
from scipy.optimize import minimize
from scipy import optimize
from mpmath import *
import time
import multiprocessing

start = time.time()

#Общие параметры
x0 = 0
z0 = 6
l=0.0001
L=100.0
R = 6.0


#Параметры для фи1
a1 = (0.5)*(math.sqrt(l))
b1 = (0.5)*(math.sqrt(l))
c1 = (0.5)*(math.sqrt(l))

#Параметры для фи21
a2 = 0.2
b2 = 0.2
c2 = 0.2

#Параметры для фи22
a3 = 1.313
b3 = 1.313
c3 = 1.313

def phi1(x, y, z):
    return z*math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)
def phi21(x, y, z):
    return math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2)
def phi22(x, y, z):
    return math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2)



def phi1_dr2(x, y, z):
    return (2*a1*z*(2*a1*((x0-x)**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*b1*z*(2*b1*(y**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*c1*z*(2*c1*(z**2)-3)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2)))
def phi21_dr2(x, y, z):
    return (2*a2*(2*a2*(x**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2)))
+ (2*b2*(2*b2*(y**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2))) +
(2*c2*(2*c2*((z0-z)**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2)))
def phi22_dr2(x, y, z):
    return (2*a3*(2*a3*(x**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2)))
+ (2*b3*(2*b3*(y**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2))) +
(2*c3*(2*c3*((z0-z)**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2)))
```

```
def phi1_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (z
* math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) * math.exp(-c1 * z **
2))/2
    #else:
    #    return 0.0
def phi21_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-
z0)**2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (math.exp(-
a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2))/2
    #else:
    #    return 0.0
def phi22_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-
z0)**2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (math.exp(-
a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2))/2
    #else:
    #    return 0.0



def phi1_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return z*math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)*J(r,
teta)
def phi21_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2)*J(r,
teta)
def phi22_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2)*J(r,
teta)



def phi1_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)+ z0
    return 2 * (z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (
                z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 2
def phi21_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)+ z0
    return 2 * (math.exp(-a2 * x ** 2) * math.exp(-b2 * y ** 2) * math.exp(-c2 *
```

```
(z - z0) ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (math.exp(-a2 * x ** 2) * math.exp(-
b2 * y ** 2) * math.exp(-c2 * (z - z0) ** 2)) / 2
def phi22_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta) + z0
    return 2 * (math.exp(-a3 * x ** 2) * math.exp(-b3 * y ** 2) * math.exp(-c3 *
(z - z0) ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (math.exp(-a3 * x ** 2) * math.exp(-
b3 * y ** 2) * math.exp(-c3 * (z - z0) ** 2)) / 2


def J(r, teta):
    return r * math.sin(teta)




def integrate_phi00():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1(x, y, z), 0, L,
-L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi01():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi02():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi10():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi11():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi12():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, 40, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi20():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi21():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi22():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res



def integrate_Aphi00_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
```

```
def integrate_Aphi01_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi10_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi20_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi21_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res



def integrate_Aphi00_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi01_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi10_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
```

```
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi20_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi21_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res



def integrate_Aphi00_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi01_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi10_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi20_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi21_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res


if __name__ == '__main__':
    def get_phi(N):
        start1 = time.time()
        pool1 = multiprocessing.Pool(processes=9)
```

```
    res00 = pool1.apply_async(integrate_phi00)
    res01 = pool1.apply_async(integrate_phi01)
    res02 = pool1.apply_async(integrate_phi02)

    res10 = pool1.apply_async(integrate_phi10)
    res11 = pool1.apply_async(integrate_phi11)
    res12 = pool1.apply_async(integrate_phi12)

    res20 = pool1.apply_async(integrate_phi20)
    res21 = pool1.apply_async(integrate_phi21)
    res22 = pool1.apply_async(integrate_phi22)



    phi_m = np.zeros(shape=(N, N))

    phi_m[0][0] = res00.get()
    phi_m[0][1] = res01.get()
    phi_m[0][2] = res02.get()

    phi_m[1][0] = res10.get()
    phi_m[1][1] = res11.get()
    phi_m[1][2] = res12.get()

    phi_m[2][0] = res20.get()
    phi_m[2][1] = res21.get()
    phi_m[2][2] = res22.get()

    end1 = time.time()
    print("Время на матрицу phi")
    print(end1 - start1)
    print()

    return phi_m
def get_Aphi_1(N):
    start2 = time.time()

    pool2 = multiprocessing.Pool(processes=9)

    resA00 = pool2.apply_async(integrate_Aphi00_1)
    resA01 = pool2.apply_async(integrate_Aphi01_1)
    resA02 = pool2.apply_async(integrate_Aphi02_1)

    resA10 = pool2.apply_async(integrate_Aphi10_1)
    resA11 = pool2.apply_async(integrate_Aphi11_1)
    resA12 = pool2.apply_async(integrate_Aphi12_1)

    resA20 = pool2.apply_async(integrate_Aphi20_1)
    resA21 = pool2.apply_async(integrate_Aphi21_1)
    resA22 = pool2.apply_async(integrate_Aphi22_1)



    Aphi_m = np.zeros(shape=(N, N))

    Aphi m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()
```

```
    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()

    end2 = time.time()
    print("Время на матрицу Aphi_1")
    print(end2 - start2)
    print()

    return Aphi_m
def get_Aphi_2(N):
    start2 = time.time()

    pool3 = multiprocessing.Pool(processes=9)

    resA00 = pool3.apply_async(integrate_Aphi00_2)
    resA01 = pool3.apply_async(integrate_Aphi01_2)
    resA02 = pool3.apply_async(integrate_Aphi02_2)

    resA10 = pool3.apply_async(integrate_Aphi10_2)
    resA11 = pool3.apply_async(integrate_Aphi11_2)
    resA12 = pool3.apply_async(integrate_Aphi12_2)

    resA20 = pool3.apply_async(integrate_Aphi20_2)
    resA21 = pool3.apply_async(integrate_Aphi21_2)
    resA22 = pool3.apply_async(integrate_Aphi22_2)



    Aphi_m = np.zeros(shape=(N, N))

    Aphi_m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()

    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()

    end2 = time.time()
    print("Время на матрицу Aphi_2")
    print(end2 - start2)
    print()

    return Aphi_m

def get_Aphi_3(N):
    start2 = time.time()

    pool3 = multiprocessing.Pool(processes=9)

    resA00 = pool3.apply_async(integrate_Aphi00_3)
    resA01 = pool3.apply_async(integrate_Aphi01_3)
    resA02 = pool3.apply_async(integrate_Aphi02_3)

    resA10 = pool3.apply_async(integrate_Aphi10_3)
    resA11 = pool3.apply_async(integrate_Aphi11_3)
    resA12 = pool3.apply_async(integrate_Aphi12_3)
```

```
    resA20 = pool3.apply_async(integrate_Aphi20_3)
    resA21 = pool3.apply_async(integrate_Aphi21_3)
    resA22 = pool3.apply_async(integrate_Aphi22_3)



    Aphi_m = np.zeros(shape=(N, N))

    Aphi_m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()

    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()

    end2 = time.time()
    print("Время на матрицу Aphi_3")
    print(end2 - start2)
    print()

    return Aphi_m


phi_m = get_phi(3)
print("Матрица phi")
print(phi_m)
print("\n")

Aphi_m1 = get_Aphi_1(3)
print("Матрица Aphi_1")
print(Aphi_m1)
print("\n")


Aphi_m2 = get_Aphi_2(3)
print("Матрица Aphi_2")
print(Aphi_m2)
print("\n")

Aphi_m3 = get_Aphi_3(3)
print("Матрица Aphi_3")
print(Aphi_m3)
print("\n")


Aphi_m = Aphi_m1 + Aphi_m2 + Aphi_m3

eigenVal, eigenVectors = LA.eig(Aphi_m, phi_m)
print("Собственные значения")
print(eigenVal)
print("\n")
print("Собственные векторы")
print(eigenVectors)
```

ПРИЛОЖЕНИЕ Г

Код программы на языке Python

```
import math
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import dia_matrix
from scipy import integrate
from scipy import linalg as LA
from scipy.optimize import minimize
from scipy import optimize
from mpmath import *
import time
import multiprocessing

start = time.time()

#Общие параметры
N = 4
x0 = 0
z0 = 6
l = 0.5
L = 100.0
R = 6.0


#Параметры для фи1
a1 = (0.5)*(math.sqrt(l))
b1 = (0.5)*(math.sqrt(l))
c1 = (0.5)*(math.sqrt(l))

#Параметры для фи21
a2 = 0.2
b2 = 0.2
c2 = 0.2

#Параметры для фи22
a3 = 1.313
b3 = 1.313
c3 = 1.313

def phi1(x, y, z):
    return z*math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)
def phi2(x, y, z):
    return math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)
def phi21(x, y, z):
    return math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2)
def phi22(x, y, z):
    return math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2)



def phi1_dr2(x, y, z):
    return (2*a1*z*(2*a1*((x0-x)**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*b1*z*(2*b1*(y**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*c1*z*(2*c1*(z**2)-3)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2)))
def phi2_dr2(x, y, z):
    return (2*a1*(2*a1*((x0-x)**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*b1*(2*b1*(y**2)-1)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2))) + (2*c1*(2*c1*(z**2)-3)*math.exp((-a1*(x0-x)**2)-(b1*y**2)-
(c1*z**2)))
```

```
def phi21_dr2(x, y, z):
    return (2*a2*(2*a2*(x**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2)))
+ (2*b2*(2*b2*(y**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2))) +
(2*c2*(2*c2*((z0-z)**2)-1)*math.exp((-a2*x**2)-(b2*y**2)-(c2*(z0-z)**2)))
def phi22_dr2(x, y, z):
    return (2*a3*(2*a3*(x**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2)))
+ (2*b3*(2*b3*(y**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2))) +
(2*c3*(2*c3*((z0-z)**2)-1)*math.exp((-a3*x**2)-(b3*y**2)-(c3*(z0-z)**2)))



def phi1_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (z
* math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) * math.exp(-c1 * z **
2))/2
    #else:
    #    return 0.0
def phi2_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) *
(math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) * math.exp(-c1 * z **
2))/2
    #else:
    #    return 0.0
def phi21_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-
z0)**2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (math.exp(-
a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2))/2
    #else:
    #    return 0.0
def phi22_(x, y, z):
    #if math.sqrt(x ** 2 + y ** 2 + (z - z0) ** 2) > 0.35:
    return 2 * (math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-
z0)**2))/math.sqrt(x**2+y**2+(z-z0)**2) - l*(x**2+y**2+z**2) * (math.exp(-
a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2))/2
    #else:
    #    return 0.0



def phi1_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return z*math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)*J(r,
teta)
def phi2_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return math.exp(-a1*(x-x0)**2)*math.exp(-b1*y**2)*math.exp(-c1*z**2)*J(r,
teta)
def phi21_r(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return math.exp(-a2*x**2)*math.exp(-b2*y**2)*math.exp(-c2*(z-z0)**2)*J(r,
teta)
def phi22_r(r, teta, phi):
```

```
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)
    return math.exp(-a3*x**2)*math.exp(-b3*y**2)*math.exp(-c3*(z-z0)**2)*J(r,
teta)



def phi1_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)+ z0
    return 2 * (z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (
                z * math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 2
def phi2_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)+ z0
    return 2 * (math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (
                math.exp(-a1 * (x - x0) ** 2) * math.exp(-b1 * y ** 2) *
math.exp(-c1 * z ** 2)) / 2
def phi21_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta)+ z0
    return 2 * (math.exp(-a2 * x ** 2) * math.exp(-b2 * y ** 2) * math.exp(-c2 *
(z - z0) ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (math.exp(-a2 * x ** 2) * math.exp(-
b2 * y ** 2) * math.exp(-c2 * (z - z0) ** 2)) / 2
def phi22_r_(r, teta, phi):
    x = r*math.sin(teta)*math.cos(phi)
    y = r*math.sin(teta)*math.cos(phi)
    z = r*math.cos(teta) + z0
    return 2 * (math.exp(-a3 * x ** 2) * math.exp(-b3 * y ** 2) * math.exp(-c3 *
(z - z0) ** 2)) / 0.05 - l * (
                x ** 2 + y ** 2 + z ** 2) * (math.exp(-a3 * x ** 2) * math.exp(-
b3 * y ** 2) * math.exp(-c3 * (z - z0) ** 2)) / 2


def J(r, teta):
    return r * math.sin(teta)




def integrate_phi00():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1(x, y, z), 0, L,
-L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi01():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi2(x, y, z), 0, L,
-L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi02():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
```

```
def integrate_phi03():
    res = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_phi10():
    res = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi1(x, y, z), 0, L,
-L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi11():
    res = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi2(x, y, z), 0, L,
-L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi12():
    res = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi13():
    res = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_phi20():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi21():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi2(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi22():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi23():
    res = integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, 40, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_phi30():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi31():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi2(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi32():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_phi33():
    res = integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22(x, y, z), 0,
L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res



def integrate_Aphi00_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi01_1():
```

```
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi2_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi03_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi10_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi2_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi13_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi20_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi21_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi2_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi23_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi30_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi31_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi2_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi32_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21 dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi33_1():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22_dr2(x,
y, z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
```

```
def integrate_Aphi00_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi01_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi2_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi03_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi1_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res

def integrate_Aphi10_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi2_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi2_r(r, teta, phi) *
phi2_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi2_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi13_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi2_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res

def integrate_Aphi20_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi21_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi2_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21 r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi23_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi21_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
```

```
    return res

def integrate_Aphi30_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi1_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi31_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi2_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi32_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi21_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res
def integrate_Aphi33_2():
    res = (-1) * integrate.tplquad(lambda r, teta, phi: phi22_r(r, teta, phi) *
phi22_r_(r, teta, phi), 0.0, R, 0.0, math.pi, 0.0, 2*math.pi, epsabs=1.49e-5,
epsrel=1.49e-3)[0]
    return res



def integrate_Aphi00_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi01_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi2_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi02_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi03_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi10_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi11_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi2_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi12_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi13_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi22 (x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi20_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
```

```
def integrate_Aphi21_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi2_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi22_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi23_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi21(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res

def integrate_Aphi30_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi1_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi31_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi2_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi32_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi21_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res
def integrate_Aphi33_3():
    res = (-1) * integrate.tplquad(lambda x, y, z: phi22(x, y, z) * phi22_(x, y,
z), 0, L, -L, L, -L, L, epsabs=1.49e-5, epsrel=1.49e-3)[0]
    return res


if __name__ == '__main__':
    def get_phi(N):
        start1 = time.time()
        phi_m = np.zeros(shape=(N, N))
        pool1 = multiprocessing.Pool(processes=8)

        res00 = pool1.apply_async(integrate_phi00)
        res01 = pool1.apply_async(integrate_phi01)
        res02 = pool1.apply_async(integrate_phi02)
        res03 = pool1.apply_async(integrate_phi03)

        res10 = pool1.apply_async(integrate_phi10)
        res11 = pool1.apply_async(integrate_phi11)
        res12 = pool1.apply_async(integrate_phi12)
        res13 = pool1.apply_async(integrate_phi13)

        phi_m[0][0] = res00.get()
        phi_m[0][1] = res01.get()
        phi_m[0][2] = res02.get()
        phi_m[0][3] = res03.get()

        phi_m[1][0] = res10.get()
        phi_m[1][1] = res11.get()
        phi_m[1][2] = res12.get()
        phi_m[1][3] = res13.get()

        pool2 = multiprocessing.Pool(processes=8)

        res20 = pool2.apply_async(integrate_phi20)
        res21 = pool2.apply_async(integrate_phi21)
        res22 = pool2.apply_async(integrate_phi22)
        res23 = pool2.apply_async(integrate_phi23)
```

```
    res30 = pool2.apply_async(integrate_phi30)
    res31 = pool2.apply_async(integrate_phi31)
    res32 = pool2.apply_async(integrate_phi32)
    res33 = pool2.apply_async(integrate_phi33)

    phi_m[2][0] = res20.get()
    phi_m[2][1] = res21.get()
    phi_m[2][2] = res22.get()
    phi_m[2][3] = res23.get()

    phi_m[3][0] = res30.get()
    phi_m[3][1] = res31.get()
    phi_m[3][2] = res32.get()
    phi_m[3][3] = res33.get()

    end1 = time.time()
    print("Время на матрицу phi")
    print(end1 - start1)
    print()

    return phi_m
def get_Aphi_1(N):
    start2 = time.time()
    Aphi_m = np.zeros(shape=(N, N))
    pool1 = multiprocessing.Pool(processes=8)

    resA00 = pool1.apply_async(integrate_Aphi00_1)
    resA01 = pool1.apply_async(integrate_Aphi01_1)
    resA02 = pool1.apply_async(integrate_Aphi02_1)
    resA03 = pool1.apply_async(integrate_Aphi03_1)

    resA10 = pool1.apply_async(integrate_Aphi10_1)
    resA11 = pool1.apply_async(integrate_Aphi11_1)
    resA12 = pool1.apply_async(integrate_Aphi12_1)
    resA13 = pool1.apply_async(integrate_Aphi13_1)

    Aphi_m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()
    Aphi_m[0][3] = resA03.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()
    Aphi_m[1][3] = resA13.get()

    pool2 = multiprocessing.Pool(processes=8)

    resA20 = pool2.apply_async(integrate_Aphi20_1)
    resA21 = pool2.apply_async(integrate_Aphi21_1)
    resA22 = pool2.apply_async(integrate_Aphi22_1)
    resA23 = pool2.apply_async(integrate_Aphi23_1)

    resA30 = pool2.apply_async(integrate_Aphi30_1)
    resA31 = pool2.apply_async(integrate_Aphi31_1)
    resA32 = pool2.apply_async(integrate_Aphi32_1)
    resA33 = pool2.apply_async(integrate_Aphi33_1)

    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()
    Aphi_m[2][3] = resA23.get()
```

```
    Aphi_m[3][0] = resA30.get()
    Aphi_m[3][1] = resA31.get()
    Aphi_m[3][2] = resA32.get()
    Aphi_m[3][3] = resA33.get()

    end2 = time.time()
    print("Время на матрицу Aphi_1")
    print(end2 - start2)
    print()

    return Aphi_m
def get_Aphi_2(N):
    start2 = time.time()
    Aphi_m = np.zeros(shape=(N, N))
    pool1 = multiprocessing.Pool(processes=8)

    resA00 = pool1.apply_async(integrate_Aphi00_2)
    resA01 = pool1.apply_async(integrate_Aphi01_2)
    resA02 = pool1.apply_async(integrate_Aphi02_2)
    resA03 = pool1.apply_async(integrate_Aphi03_2)

    resA10 = pool1.apply_async(integrate_Aphi10_2)
    resA11 = pool1.apply_async(integrate_Aphi11_2)
    resA12 = pool1.apply_async(integrate_Aphi12_2)
    resA13 = pool1.apply_async(integrate_Aphi13_2)

    Aphi_m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()
    Aphi_m[0][3] = resA03.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()
    Aphi_m[1][3] = resA13.get()

    pool2 = multiprocessing.Pool(processes=8)

    resA20 = pool2.apply_async(integrate_Aphi20_2)
    resA21 = pool2.apply_async(integrate_Aphi21_2)
    resA22 = pool2.apply_async(integrate_Aphi22_2)
    resA23 = pool2.apply_async(integrate_Aphi23_2)

    resA30 = pool2.apply_async(integrate_Aphi30_2)
    resA31 = pool2.apply_async(integrate_Aphi31_2)
    resA32 = pool2.apply_async(integrate_Aphi32_2)
    resA33 = pool2.apply_async(integrate_Aphi33_2)

    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()
    Aphi_m[2][3] = resA23.get()

    Aphi_m[3][0] = resA30.get()
    Aphi_m[3][1] = resA31.get()
    Aphi_m[3][2] = resA32.get()
    Aphi_m[3][3] = resA33.get()

    end2 = time.time()
    print("Время на матрицу Aphi_2")
    print(end2 - start2)
    print()
```

```
    return Aphi_m
def get_Aphi_3(N):
    start2 = time.time()
    Aphi_m = np.zeros(shape=(N, N))
    pool1 = multiprocessing.Pool(processes=8)

    resA00 = pool1.apply_async(integrate_Aphi00_3)
    resA01 = pool1.apply_async(integrate_Aphi01_3)
    resA02 = pool1.apply_async(integrate_Aphi02_3)
    resA03 = pool1.apply_async(integrate_Aphi03_3)

    resA10 = pool1.apply_async(integrate_Aphi10_3)
    resA11 = pool1.apply_async(integrate_Aphi11_3)
    resA12 = pool1.apply_async(integrate_Aphi12_3)
    resA13 = pool1.apply_async(integrate_Aphi13_3)

    Aphi_m[0][0] = resA00.get()
    Aphi_m[0][1] = resA01.get()
    Aphi_m[0][2] = resA02.get()
    Aphi_m[0][3] = resA03.get()

    Aphi_m[1][0] = resA10.get()
    Aphi_m[1][1] = resA11.get()
    Aphi_m[1][2] = resA12.get()
    Aphi_m[1][3] = resA13.get()

    pool2 = multiprocessing.Pool(processes=8)

    resA20 = pool2.apply_async(integrate_Aphi20_3)
    resA21 = pool2.apply_async(integrate_Aphi21_3)
    resA22 = pool2.apply_async(integrate_Aphi22_3)
    resA23 = pool2.apply_async(integrate_Aphi23_3)

    resA30 = pool2.apply_async(integrate_Aphi30_3)
    resA31 = pool2.apply_async(integrate_Aphi31_3)
    resA32 = pool2.apply_async(integrate_Aphi32_3)
    resA33 = pool2.apply_async(integrate_Aphi33_3)

    Aphi_m[2][0] = resA20.get()
    Aphi_m[2][1] = resA21.get()
    Aphi_m[2][2] = resA22.get()
    Aphi_m[2][3] = resA23.get()

    Aphi_m[3][0] = resA30.get()
    Aphi_m[3][1] = resA31.get()
    Aphi_m[3][2] = resA32.get()
    Aphi_m[3][3] = resA33.get()

    end2 = time.time()
    print("Время на матрицу Aphi_3")
    print(end2 - start2)
    print()

    return Aphi_m



phi_m = get_phi(N)
print("Матрица phi")
print(phi_m)
print("\n")
```

```
Aphi_m1 = get_Aphi_1(N)
print("Матрица Aphi_1")
print(Aphi_m1)
print("\n")


Aphi_m2 = get_Aphi_2(N)
print("Матрица Aphi_2")
print(Aphi_m2)
print("\n")

Aphi_m3 = get_Aphi_3(N)
print("Матрица Aphi_3")
print(Aphi_m3)
print("\n")


Aphi_m = Aphi_m1 + Aphi_m2 + Aphi_m3

eigenVal, eigenVectors = LA.eig(Aphi_m, phi_m)
print("Собственные значения")
print(eigenVal)
print("\n")
print("Собственные векторы")
print(eigenVectors)
```

ПРИЛОЖЕНИЕ Д

Код программы на языке Python

```
import math
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import dia_matrix
from scipy import integrate
from scipy import linalg as LA
from scipy.optimize import minimize
from scipy import optimize
from mpmath import *

l=1

def phi1(x, y, z):
    a = (0.5)*(np.sqrt(l))
    b = (0.5)*(np.sqrt(l))
    c = (0.5)*(np.sqrt(l))
    x0 = 0
    return z*np.exp(-a*(x-x0)**2)*np.exp(-b*y**2)*np.exp(-c*z**2)

def phi2(x, y, z):
    a = 0.2
    b = 0.2
    c = 0.2
    z0 = 6
    return np.exp(-a*x**2)*np.exp(-b*y**2)*np.exp(-c*(z-z0)**2)

def phi1_dr2(x, y, z):
    x0 = 0
    a = (0.5)*(np.sqrt(l))
    b = (0.5)*(np.sqrt(l))
    c = (0.5)*(np.sqrt(l))
    return (2*a*z*(2*a*((x0-x)**2)-1)*np.exp((-a*(x0-x)**2)-(b*y**2)-(c*z**2)))
+ (2*b*z*(2*b*(y**2)-1)*np.exp((-a*(x0-x)**2)-(b*y**2)-(c*z**2))) +
(2*c*z*(2*c*(z**2)-3)*np.exp((-a*(x0-x)**2)-(b*y**2)-(c*z**2)))

def phi2_dr2(x, y, z):
    a = 0.2
    b = 0.2
    c = 0.2
    z0 = 6
    return (2*a*(2*a*(x**2)-1)*np.exp((-a*x**2)-(b*y**2)-(c*(z0-z)**2))) +
(2*b*(2*b*(y**2)-1)*np.exp((-a*x**2)-(b*y**2)-(c*(z0-z)**2))) + (2*c*(2*c*((z0-
z)**2)-1)*np.exp((-a*x**2)-(b*y**2)-(c*(z0-z)**2)))

def V(x, y, z, v0, d):
    x0 = 0
    t = (x-x0)**2 + y**2 + z**2
    return (2*v0/math.pi) * math.atan((d/2)*np.sqrt(2/(t+np.sqrt(t**2 +
(d**2)*(z**2)))))

#np.inf
#epsrel=1.49e-5

def fin_func(N, v0, d):
    phi_m = np.zeros(shape=(N, N))
```

```
    print("1.1")

    phi_m[0][0] = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi1(x, y,
z), 0, 100, -100, 100, -100, 100, epsabs=1.49e-5, epsrel=1.49e-3)[0]


    print("1.2")

    phi_m[0][1] = integrate.tplquad(lambda x, y, z: phi1(x, y, z) * phi2(x, y,
z), 0, 100, -100, 100, -100, 100, epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("1.3")

    phi_m[1][0] = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi1(x, y,
z), 0, 100, -100, 100, -100, 100, epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("1.4")

    phi_m[1][1] = integrate.tplquad(lambda x, y, z: phi2(x, y, z) * phi2(x, y,
z), 0, 100, -100, 100, -100, 100, epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print()
    print("Матрица phi")
    print(phi_m)
    print()

    Aphi_m = np.zeros(shape=(N, N))

    print("2.1")

    Aphi_m[0][0] = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * V(x,
y, z, v0, d) + phi1(x, y, z) * phi1_dr2(x, y, z), 0, 100, -100, 100, -100, 100,
epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("2.2")

    Aphi_m[0][1] = (-1) * integrate.tplquad(lambda x, y, z: phi1(x, y, z) * V(x,
y, z, v0, d) + phi1(x, y, z) * phi2_dr2(x, y, z), 0, 100, -100, 100, -100, 100,
epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("2.3")

    Aphi_m[1][0] = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * V(x,
y, z, v0, d) + phi2(x, y, z) * phi1_dr2(x, y, z), 0, 100, -100, 100, -100, 100,
epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("2.4")

    Aphi_m[1][1] = (-1) * integrate.tplquad(lambda x, y, z: phi2(x, y, z) * V(x,
y, z, v0, d) + phi2(x, y, z) * phi2_dr2(x, y, z), 0, 100, -100, 100, -100, 100,
epsabs=1.49e-5, epsrel=1.49e-3)[0]

    print("final")
    print()
    print()
    print()
    print()
    print()
    print()


    print("Матрица phi")
```

```
    print(phi_m)
    print()

    print("Матрица Aphi")
    print(Aphi_m)
    print()

    eigenVal, eigenVectors = LA.eig(Aphi_m, phi_m)
    print("Собственные значения")
    print(eigenVal)
    print()
    print("Собственные векторы")
    print(eigenVectors)

    return min(eigenVal)

v0 = 2
d = 1

while (v0 <=7):
    fin_func(2, v0, d)
    v0 += 1
```