Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет»
Физический факультет
Кафедра вычислительной физики

Дипломная работа

Гусаров Дмитрий Сергеевич

«Квантово-механические методы исследования состояний молекулярного иона водорода»

Научный руководитель:
д.ф.-м.н., доц. Е. А. Яревский

Рецензент:
к.ф.-м.н., доц. Г. В. Филиппенко

Санкт-Петербург
2018

Оглавление

Введение

С самого начала развития квантовой механики молекулярный ион водорода H_2^+ изучался очень подробно, потому что это простейшая молекулярная система, состоящая только из одного электрона и двух одинаковых ядер. Несмотря на эту кажущуюся простоту, теоретическое исследование H_2^+ являлось достаточно трудной проблемой, поскольку это типичная трёхчастичная кулоновская система.

В 20-х годах успешно разрабатывались методы решения упрощённой задачи, когда система состоит из двух неподвижных ядер и движущегося электрона [1] [2]. В это время обнаружили, что для такой системы уравнение Шрёдингера допускает разделение переменных в сфероидальных координатах и позволяет вычислить электронную энергию точно. Это позволило применять адиабатические приближение к трёхчастичной системе, а также продвинуться в использовании вариационных приёмов. С развитием первых ЭВМ(60-е годы) начинаются активные расчёты спектра этой системы. В это время применяются как адиабатические, так и вариационные методы. Типичная схема адиабатического метода в это время [3]: Расчёт решений задачи двух кулоновских центров вариационным методом [1], аналитический расчёт адиабатических поправок, и, наконец, решение дифференциального уравнения второго порядка интегрированием. Также появились неадиабатические методы [4].

С последующим развитием ЭВМ расчёты улучшали свою точность, а методы становились изощрённее. Методы, применяемые в последнее время: вариационно-возмущенный метод [5], вариационный метод с использованием полного трехмерного гамильтониана тела [6] и методом, полученным из физики столкновений [7]. Для вариационных методов [8] [9] [10] точность достигает 10^{-15} и даже 10^{-20} для нижних уровней. Оказывается, что такие высокоточные расчёты чувствительны к отношению массы протона к массе электрона. Это даёт возможность найти это отношение, проводя высокоточное измерения оптических переходов в H_2^+, например между различными колебательно-вращательными уровнями электронного основного состояния $1s\sigma$.

Целью данной работы является исследование спектра молекулярного иона водорода H_2^+ различными методами, а также сравнение результатов этих методов. Дипломная работа включает в себя три задачи и имеет следующую структуру:

1. В первой главе приведён метод для решения приближенной системы - состоящей из двух неподвижных ядер и движущегося электрона.

2. Во второй главе описывается решение трёхчастичной задачи с помощью адиабатического разложения волновой функции. При его реализации используются результаты, получаемые для задачи двух кулоновских центров.

3. В третьей главе описано сведение исходной задачи к вариационной для последующе-

го её решения с помощью метода конечных элементов.

4. В приложениях приведены фрагменты исходного кода, разработанного для расчёта адиабатических поправок, а также для решения задачи двух кулоновских центров.

Глава 1

Задача двух кулоновских центров

Сделаем приближение исходной задачи: считая массы положительно заряженных частиц бесконечными, найдём волновую функцию и термы электрона в поле неподвижных зарядов Z_1 и Z_2, которые находятся на расстоянии R друг от друга(Рис. 1.1).

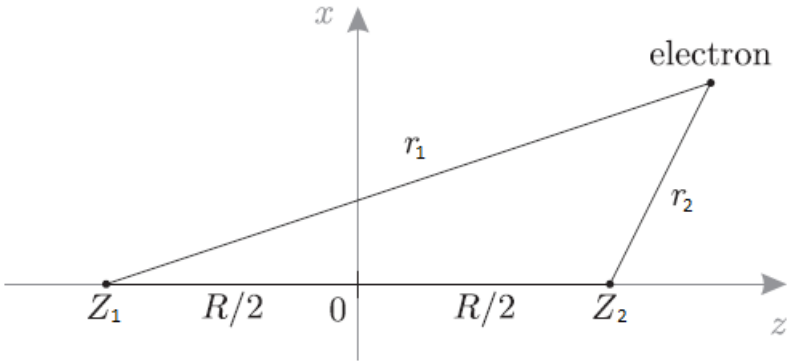

Рис. 1.1. Система $Z_1 e Z_2$

Уравнение Шрёдингера(в атомных единицах) принимает вид

$$\Delta\Psi + 2\left[E - \left(-\frac{Z_1}{r_1} - \frac{Z_2}{r_2}\right)\right]\Psi = 0, \tag{1.1}$$

где r_1 и r_2 - расстояния от нижнего и верхнего фокусов до электрона соответственно. Эту задачу удобно решать в вытянутых сфероидальных координатах. Они могут быть получены вращением вокруг большой оси эллипсов эллиптической системы координат(1.2) и связаны с прямоугольными координатами соотношениями:

$$\begin{aligned} &x = \frac{R}{2}[(\xi^2-1)(1-\eta^2)]^{1/2}\cos\varphi, \quad y = \frac{R}{2}[(\xi^2-1)(1-\eta^2)]^{1/2}\sin\varphi, \quad z = \frac{R}{2}\xi\eta, \\ &\xi = \frac{r_1+r_2}{R}, \quad \eta = \frac{r_1-r_2}{R}, \quad \varphi = \operatorname{arctg}\frac{y}{x}, \\ &\xi \in [1,\infty), \quad \eta \in [-1,1], \quad \varphi \in [0,2\pi). \end{aligned} \tag{1.2}$$

Удобство этих координат заключается в том, что уравнение (1.1) допускает разделение переменных

$$\Psi(\xi,\eta,\varphi) = \Pi(\xi)\Xi(\eta)\Phi(\varphi) \tag{1.3}$$

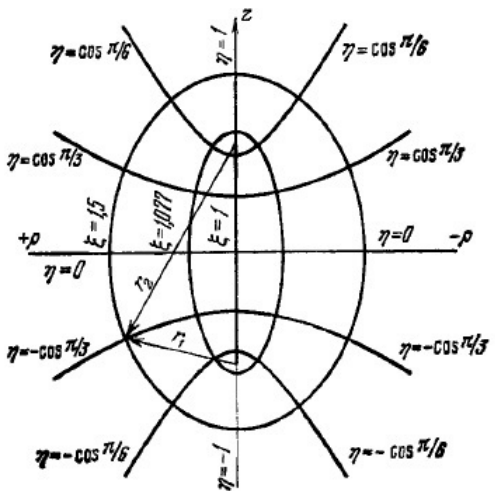

Рис. 1.2. Сечение сфероидальных координат плоскостью $\varphi = const$

и приводится [11] к системе

$$\begin{cases} \Phi_m = e^{\pm im\varphi},\ m \in \mathbb{Z}; & (1.4) \\ \dfrac{d}{d\xi}(\xi^2-1)\dfrac{d}{d\xi}\Pi + \left[-\lambda - p^2(\xi^2-1) + a\xi - \dfrac{m^2}{\xi^2-1}\right]\Pi = 0; & (1.5) \\ \dfrac{d}{d\eta}(1-\eta^2)\dfrac{d}{d\eta}\Xi + \left[\lambda - p^2(1-\eta^2) + b\eta - \dfrac{m^2}{1-\eta^2}\right]\Xi = 0. & (1.6) \end{cases}$$

В системе уравнений введены обозначения

$$a = R(Z_2 + Z_1), \quad b = R(Z_2 - Z_1), \quad p^2 = -\frac{ER^2}{2}. \tag{1.7}$$

1.1. Угловые кулоновские сфероидальные функции

Угловые кулоновские сфероидальные функции (у.к.с.ф.) $\Xi(p, b, \eta)$ определяются [11] как решения уравнения (1.6) с граничными условиями

$$|\Xi(p, b, \pm 1)| < \infty, \quad -1 \leqslant \eta \leqslant 1. \tag{1.8}$$

Эта задача Штурма-Лиувилля имеет бесконечный невырожденный дискретный спектр. Функции $\Xi_{mq}(p, b, \eta)$ и соответсвующие собственные значения $\lambda_{mq}^{(\eta}$ при заданных m, p, b нумеруются по числу нулей q на интервале $\eta \in [-1, 1]$.

Найдём производную $\lambda_{mq}^{(\eta)}$ по параметру p методом из книги [11]. Для этого рассмот-

рим нашу краевую задачу при двух различных значениях параметра p и p'

$$\frac{d}{d\eta}(1-\eta^2)\frac{d}{d\eta}\Xi_{mq}(p,\eta)+\left[\lambda_{mq}^{(\eta)}(p)-p^2(1-\eta^2)-b\eta-\frac{m^2}{1-\eta^2}\right]\Xi_{mq}(p,\eta)=0,$$
$$\frac{d}{d\eta}(1-\eta^2)\frac{d}{d\eta}\Xi_{mq}(p',\eta)+\left[\lambda_{mq}^{(\eta)}(p')-p'^2(1-\eta^2)-b\eta-\frac{m^2}{1-\eta^2}\right]\Xi_{mq}(p',\eta)=0.$$

Умножим первое уравнение на $\Xi_{mq}(p',\eta)$, а второе на $\Xi_{mq}(p,\eta)$, проинтегрируем по η от -1 до 1 и вычтем одно из другого.

$$\left(\lambda_{mq}^{(\eta)}(p)-\lambda_{mq}^{(\eta)}(p')\right)\int\limits_{-1}^{1}\Xi_{mq}(p,\eta)\Xi_{mq}(p',\eta)\,d\eta=\left(p^2-p'^2\right)\int\limits_{-1}^{1}\Xi_{mq}(p,\eta)\Xi_{mq}(p',\eta)(1-\eta^2)\,d\eta.$$

Устремив теперь $p'\to p$, перейдём к производной

$$\frac{1}{2p}\frac{\partial\lambda_{mq}^{(\eta)}(p)}{\partial p}=\int\limits_{-1}^{1}\overline{\Xi}_{mq}^2(p,b,\eta)(1-\eta^2)\,d\eta. \tag{1.9}$$

Аналогичным образом

$$\frac{\partial}{\partial b}\lambda_{mq}^{(\eta)}(p,b)=-\int\limits_{-1}^{1}\overline{\Xi}_{mq}^2(p,b,\eta)\,\eta\,d\eta, \tag{1.10}$$

Здесь $\overline{\Xi}_{mq}^2(p,b,\eta)$ - нормированы:

$$\int\limits_{-1}^{1}\overline{\Xi}_{mq}^2(p,b,\eta)\,d\eta=\delta_{qq'}.$$

Таким образом, $\lambda_{mq}^{(\eta)}(p,\,b)$ монотонно растущая по p, а по b знак производной может быть любой.

Разложение для функций $\Xi_{mq}(p,b,\eta)$ ищутся в виде [11]

$$\Xi_{mq}(p,b,\eta)=(1-\eta^2)^{m/2}e^{-p(1+\eta)}\sum_{s=0}^{\infty}c_s(1+\eta)^s, \tag{1.11a}$$
$$\Xi_{mq}(p,b,\eta)=(1-\eta^2)^{m/2}e^{-p(1-\eta)}\sum_{s=0}^{\infty}c'_s(1-\eta)^s. \tag{1.11b}$$

Подстановка в (1.6) приводит к трёхчленным рекуррентным соотношениям

$$\rho_{sq}c_{s+1}-\chi_s c_s+\delta_s c_{s-1}=0,\qquad c_{-1}=0, \tag{1.12}$$

Для случая разложения (1.11a) коэффициенты в (1.12)

$$\begin{gathered}\rho_s = 2(s+1)(s+m+1),\\ \chi_s = s(s+1) + (2s+m+1)(2p+m) + b - \lambda,\\ \delta_s = b + 2p(s+m).\end{gathered} \tag{1.13}$$

Бесконечная система (1.12) рекуррентных соотношений принимает вид

$$\mathrm{A}\overrightarrow{c} = \begin{pmatrix} -\chi_0 & \rho_0 & 0 & 0 & \dots \\ \delta_1 & -\chi_1 & \rho_1 & 0 & \dots \\ 0 & \delta_2 & -\chi_2 & \rho_2 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \end{pmatrix} = 0. \tag{1.14}$$

Собственные значения λ краевой задачи находятся из условия разрешимости этой однородной системы

$$J_0 = \det\mathrm{A} = 0. \tag{1.15}$$

Этот определитель является полиномом бесконечной степени от λ и его непосредственное вычисление, вообще говоря, невозможно, поэтому работать с ним неудобно. Заметим, что $J_0 = -\chi_0 J_1 - \rho_0\delta_1 J_2$, где J_1 и J_2 - определители той же матрицы без одного и двух первых строк и столбцов соответсвенно. Если предположить, что $J_1 \neq 0$, то нетрудно видеть, что дробь J_0/J_1 представима в виде бесконечной цепной дроби

$$-\frac{J_0}{J_1} = \chi_0 - \cfrac{\rho_0\delta_1}{\chi_1 - \cfrac{\rho_1\delta_2}{\chi_2 - \dots}} \equiv \chi_0 - \frac{\rho_0\delta_1}{\chi_1 -}\,\frac{\rho_1\delta_2}{\chi_2 -}\dots, \tag{1.16}$$

причём $J_0 = 0 \Longleftrightarrow \dfrac{J_0}{J_1} = 0.$

Условие

$$\left|\frac{\rho_{s-1}\delta_s}{\chi_{s-1}\chi_s}\right| < \frac{1}{4}, \qquad s \geqslant 1 \tag{1.17}$$

является достаточным для сходимости цепной дроби. Для разложения (1.11a) выполняется

$$\left|\frac{\rho_{s-1}\delta_s}{\chi_{s-1}\chi_s}\right| \xrightarrow[s\to\infty]{} \left(\frac{p}{s}\right)^2,$$

поэтому условие (1.17) начинает выполняться с номера $s > 2p+1$. Это следует иметь ввиду при выборе количества членов в цепной дроби (1.16).

Для коэффициентов c'_s из разложения (1.11b) соотношение (1.12) остаётся в силе, а в формулах (1.13) нужно заменить $p \to -p$. Такая замена не изменит вида цепной дроби

(1.16), так как χ_s и произведение $\rho_s\delta_{s+1}$ не зависят от знака p. Следовательно, в обоих случаях собственные значения λ находятся из одного и того же уравнения (1.15).

Поскольку бесконечная цепная дробь (1.16) сходится, её можно оборвать на достаточно большом элементе N, после чего собственные значения определителя J_0 вычисляются как корни полинома $F_{N+1}(p,b,\lambda)$, заданного соотношением

$$F^{(\eta)}_{N+1}(p,b,\lambda)=\frac{J_{0(N+1)}(p,b,\lambda)}{J_{1(N+1)}(p,b,\lambda)}=\chi_0-\frac{\rho_0\delta_1}{\chi_1-}\frac{\rho_1\delta_2}{\chi_2-}\cdots\frac{\rho_N\delta_{N+1}}{\chi_{N+1}}. \tag{1.18}$$

Из определения (1.18) вытекают рекуррентные соотношения для полиномов $J_{0k}(p,b,\lambda)$ и $J_{1k}(p,b,\lambda)$

$$J_{0(k+1)}=\chi_k J_{0k}-\rho_{k-1}\delta_k J_{0(k-1)},\quad J_{0(-1)}=1,\quad J_{00}=\chi_0.$$

$$J_{1(k+1)}=\chi_k J_{1k}-\rho_{k-1}\delta_k J_{1(k-1)},\quad J_{1(-1)}=0,\quad J_{10}=1.$$

1.2. Радиальные кулоновские сфероидальные функции

Радиальные кулоновские сфероидальные функции (р.к.с.ф.) $\Pi(p,a,\xi)$ определяются [11] как решения уравнения (1.5) с граничными условиями

$$|\Pi(p,a,1)|<\infty,\quad \Pi_{mk}(p,a,\xi)\underset{\xi\to\infty}{\to}0.$$

Эта задача при $p^2>0$ и фиксированных значениях m и a имеет дискретный спектр. Функции $\Pi_{mk}(p,a,\xi)$ и соответсвующие собственные значения $\lambda^{(\xi)}_{mk}$ при заданных m, p, a нумеруются по числу нулей $k=0,1,2,\ldots$ на интервале $\xi\in(1,\,\infty)$.

Легко показать, как и в предыдущем пункте, что $\lambda^{(\xi)}_{mk}(p,a)$ являются монотонно убывающими по p и монотонно растущими по a.

Уравнение (1.5) для р.к.с.ф. после выделения особенностей [11] в точках $\xi=1$ и $\xi=\infty$

$$\Pi_{mk}(p,a,\xi)=(\xi^2-1)^{m/2}\,e^{-p(\xi-1)}\,f(\xi) \tag{1.19}$$

приведёт к следующему уравнению для функции $f(\xi)$

$$\left(\xi^2-1\right)f''(\xi)+\left(-2p\left(\xi^2-1\right)+2\left(m+1\right)\xi\right)f'(\xi)+(-\lambda+m(m+1)+2p\sigma\xi)\,f(\xi)=0, \tag{1.20}$$

где

$$\sigma=\frac{a}{2p}-(m+1).$$

Вид функции $f(\xi)$ был предложен Яффе [2]

$$f(\xi)=(\xi+1)^{\sigma}\sum_{s=0}^{\infty}g_s x^s, \tag{1.21}$$

где

$$x = (\xi - 1)/(\xi + 1).$$

Подстановка в уравнение (1.20) приводит рекуррентному соотношению

$$\alpha_s g_{s+1} - \beta_s g_s + \gamma_s g_{s-1} = 0, \tag{1.22}$$

с коэффициентами

$$\begin{gathered} \alpha_s = (s+1)(s+m+1), \\ \beta_s = 2s(s+2p-\sigma) - (m+\sigma)(m+1) - 2p\sigma + \lambda, \\ \gamma_s = (s-1-\sigma)(s-m-1-\sigma). \end{gathered} \tag{1.23}$$

Поскольку

$$\left|\frac{\alpha_{s-1}\gamma_s}{\beta_{s-1}\beta_s}\right| \xrightarrow[s\to\infty]{} \frac{1}{4}\left(1 - \frac{4p}{s}\right),$$

то цепная дробь

$$F^{(\xi)}(p,a,\lambda) = \beta_0 - \frac{\alpha_0\gamma_1}{\beta_1 -}\,\frac{\alpha_1\gamma_2}{\beta_2 -}\cdots = 0 \tag{1.24}$$

сходится при всех $p > 0$.

По аналогии с у.к.с.ф. для нахождения нулей (1.24) бесконечная дробь заменяется конечной

$$F^{(\xi)}_{N+1}(p,a,\lambda) = \frac{J_{0(N+1)}(p,a,\lambda)}{J_{1(N+1)}(p,a,\lambda)}, \tag{1.25}$$

в которой

$$J_{0(k+1)} = \beta_k J_{0k} - \alpha_{k-1}\gamma_k J_{0(k-1)}, \quad J_{0(-1)} = 1, \quad J_{00} = \beta_0.$$

$$J_{1(k+1)} = \beta_k J_{1k} - \alpha_{k-1}\gamma_k J_{1(k-1)}, \quad J_{1(-1)} = 0, \quad J_{10} = 1.$$

1.3. Алгоритм

Представим алгоритм действий для нахождения терма $E_{kqm}(R)$, то есть нахождения таких $p_{kqm}(R)$ и $\lambda_{kqm}(R)$, чтобы оба уравнения(1.5, 1.6) выполнялись.

1. Из множества корней цепных дробей (1.18) и (1.25) необходимо выбрать соответствующий квантовым числам kqm. Для этого можно сначала решить задачу при $R = R_0 \approx 0$. В этом случае система близка к водородоподобному атому с квантовы-

ми числами N', l', m', которые связаны с k, q, m исходной задачи выражениями

$$N' = k + l' + 1, \tag{1.26}$$
$$l' = q + m', \tag{1.27}$$
$$m' = m. \tag{1.28}$$

Энергию можно найти по теории возмущения([11]):

$$E_{kqm}(R_0) = -\frac{(Z1+Z2)^2}{2N'^2} - \frac{2Z_1Z_2[l'(l'+1) - 3m'^2]}{N'^3 l'(l'+1)(2l'-1)(2l'+1)(2l'+3)}(Z1+Z2)^2R^2,$$

отсюда $p_{kqm}(R_0) = \sqrt{-E_{kqm}(R_0)R_0^2/2}$.

Чтобы определить $\lambda_{kqm}(R_0)$, рассмотрим уравнение (1.6) с параметрами $p = 0$, $b = 0$:

$$\frac{d}{d\eta}(1-\eta^2)\frac{d}{d\eta}\Xi_{qm} + \left[\lambda - \frac{m^2}{1-\eta^2}\right]\Xi_{qm} = 0 \tag{1.29}$$

Уравнение (1.29) называется обобщённым уравнением Лежандра, а спектром являются $\lambda_{qm} = (q+m)(q+m+1)$. Поэтому искомый корень $\lambda_{kqm}(R_0)$ ищется в окрестности $(q+m)(q+m+1)$.

2. Затем, не меняя p, постепенно увеличиваем a и b до искомых значений $R(Z_1+Z_2)$ и $R(Z_2-Z_1)$ сответственно, следя за положением корней $\lambda_{km}^{(\xi)}$ и $\lambda_{qm}^{(\eta)}$.

3. Наконец, постепенно увеличивая $p(R)$, добиваемся того , чтобы корни $\lambda_{km}^{(\xi)}$ и $\lambda_{qm}^{(\eta)}$ совпали. Единственность p следует из того, что $\partial\lambda_{km}^{(\xi)}/\partial p$ и $\partial\lambda_{qm}^{(\eta)}/\partial p$ имеют разные знаки. Таким образом, энергия $E_{kqm}(R) = -2p_{kqm}^2/R^2$ найдена.

4. Найти волновые функции в виде рядов (1.11) и (1.21) можно, решая системы линейных уравнений (1.14) с найденными параметрами p и $\lambda = \lambda_{mk}^{(\xi)}(p,a) = \lambda_{mk}^{(\eta)}(p,b)$.

1.4. Результаты

Алгоритм, приведённый выше, был реализован на языке C++. Основные его фрагменты можно найти в приложении. В таблицах (1.1, 1.2) приведены результаты вычисления энергетического параметра p для некоторых состояний двух систем этим реализованным алгоритмом и программой ODKIL [12], основанной на методе Киллингбека [13].

Таблица 1.1. Сравнение параметра $p = \sqrt{-ER^2/2}$, $Z_1 = Z_2 = 1$

Программа	Расстояние между центрами				
	0.1	1	3	7	15
$1s\sigma$					
odkill	0.0994545637	0.8519936365	2.0246068478	3.9858566336	7.9846560361
данный алгоритм	0.0994545642	0.8519936366	2.0246068478	3.9858566336	7.9847044938
$2p\pi$					
odkill	0.0499834041	0.4868818935	1.3187104819	2.6033581804	4.6662461408
данный алгоритм	0.0499834040	0.4868818934	1.3187104819	2.6033581804	4.6662461408
$3d\sigma$					
odkill	0.0333354507	0.3355478267	1.0764615030	2.7488731881	4.9652952060
данный алгоритм	0.0333354507	0.3355478267	1.0764615030	2.7488731881	4.9652952059

Таблица 1.2. Сравнение параметра $p = \sqrt{-ER^2/2}$, $Z_1 = 1, Z_2 = 3$

Программа	Расстояние между центрами				
	0.1	1	3	7	15
$1s\sigma$					
odkill	0.1972930925	1.6614705377	4.6638652665	10.6653779065	22.6660552774
данный алгоритм	0.1972930927	1.6614705377	4.6638652665	10.6653779065	22.6660552774
$2p\pi$					
odkill	0.0999015466	0.9450977656	2.5471492924	5.5699335747	11.5776676634
данный алгоритм	0.0999015464	0.9450977650	2.5471492924	5.5699335747	11.5776676634
$3d\sigma$					
odkill	0.0666793919	0.6819065346	2.3173201493	4.7953134792	8.8767710638
данный алгоритм	0.0666793919	0.6819065346	2.3173201493	4.7953134792	8.8767710638

Как видно из таблиц, значения параметра p практически идентичны. Интересно знать, как выглядят термы, которые получаются в результате вычислений. На рисунках (1.3, 1.4, 1.5) приведены термы для некоторых систем.

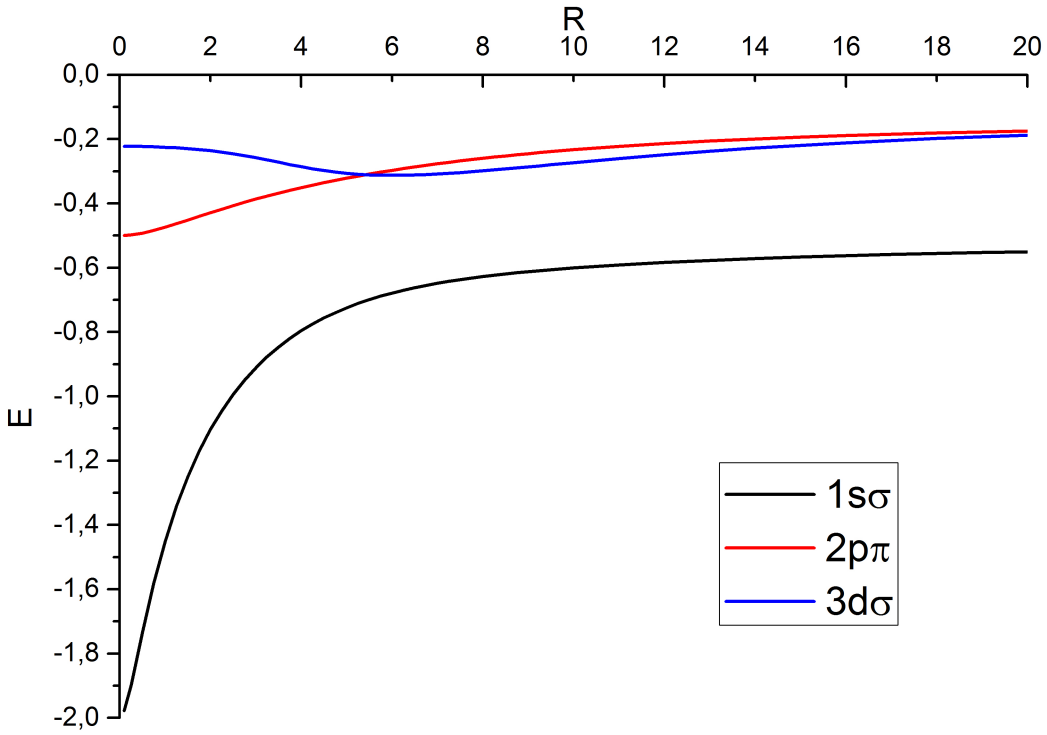

Рис. 1.3. Термы для системы $Z_1 = 1$, $Z_2 = 1$

Также интерес имеет распределение плотности вероятности электрона в пространстве. Оно приведено на рисунках (1.6, 1.7, 1.8) для различных конфигураций.

Отметим, что на малых межъядерных расстояниях распределение очень близко к распределению электрона водородоподобного атома.

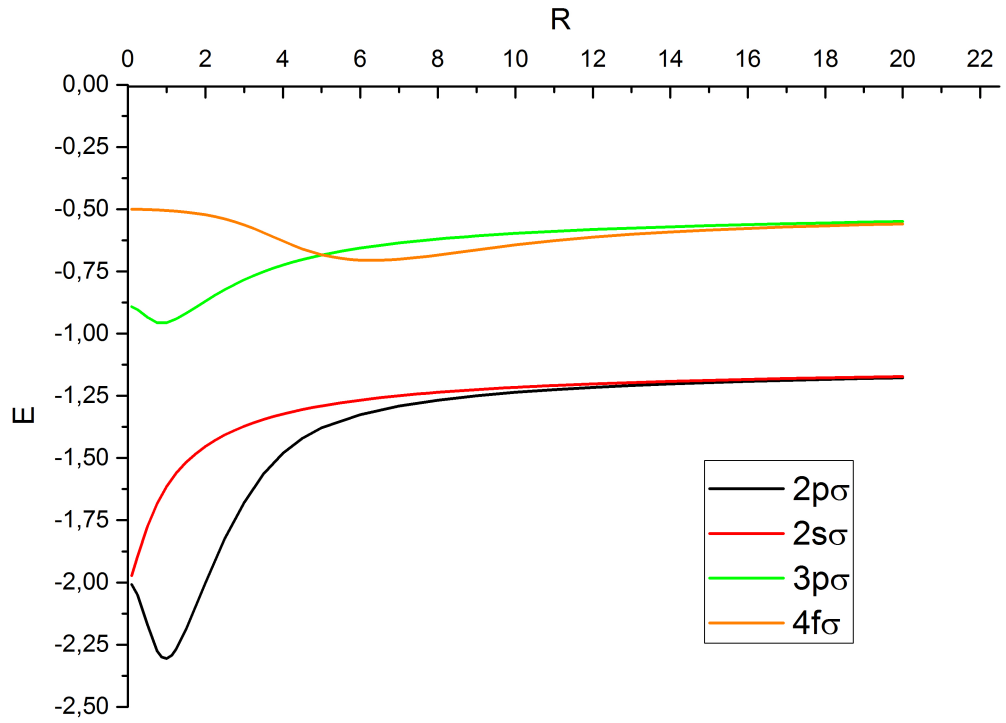

Рис. 1.4. Термы для системы $Z_1 = 1$, $Z_2 = 3$

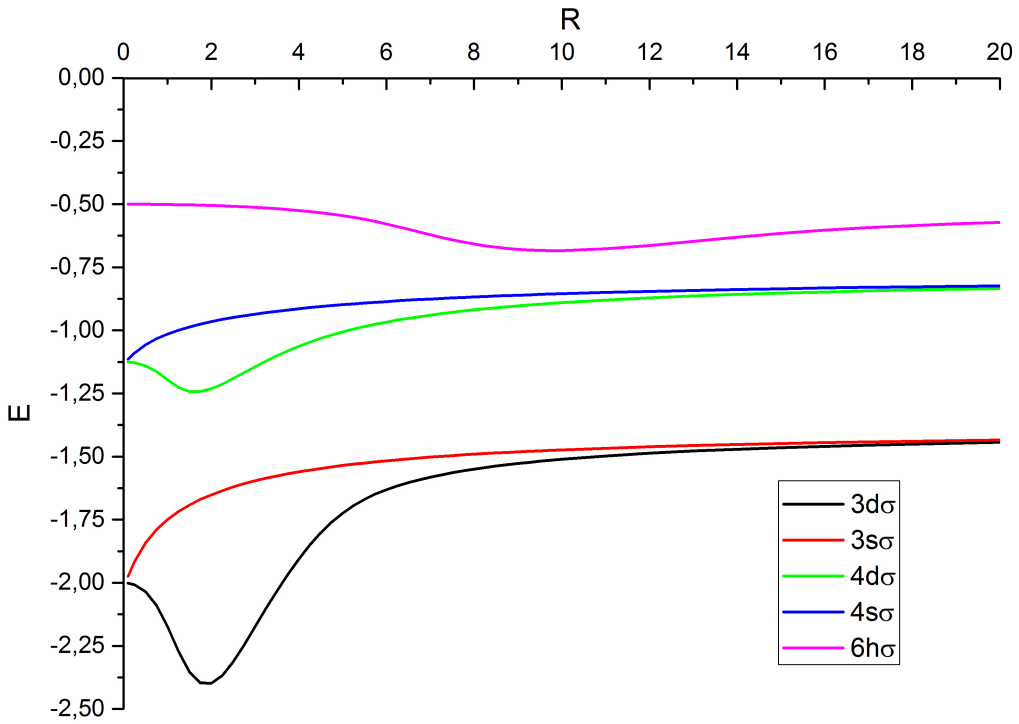

Рис. 1.5. Термы для системы $Z_1 = 1$, $Z_2 = 5$

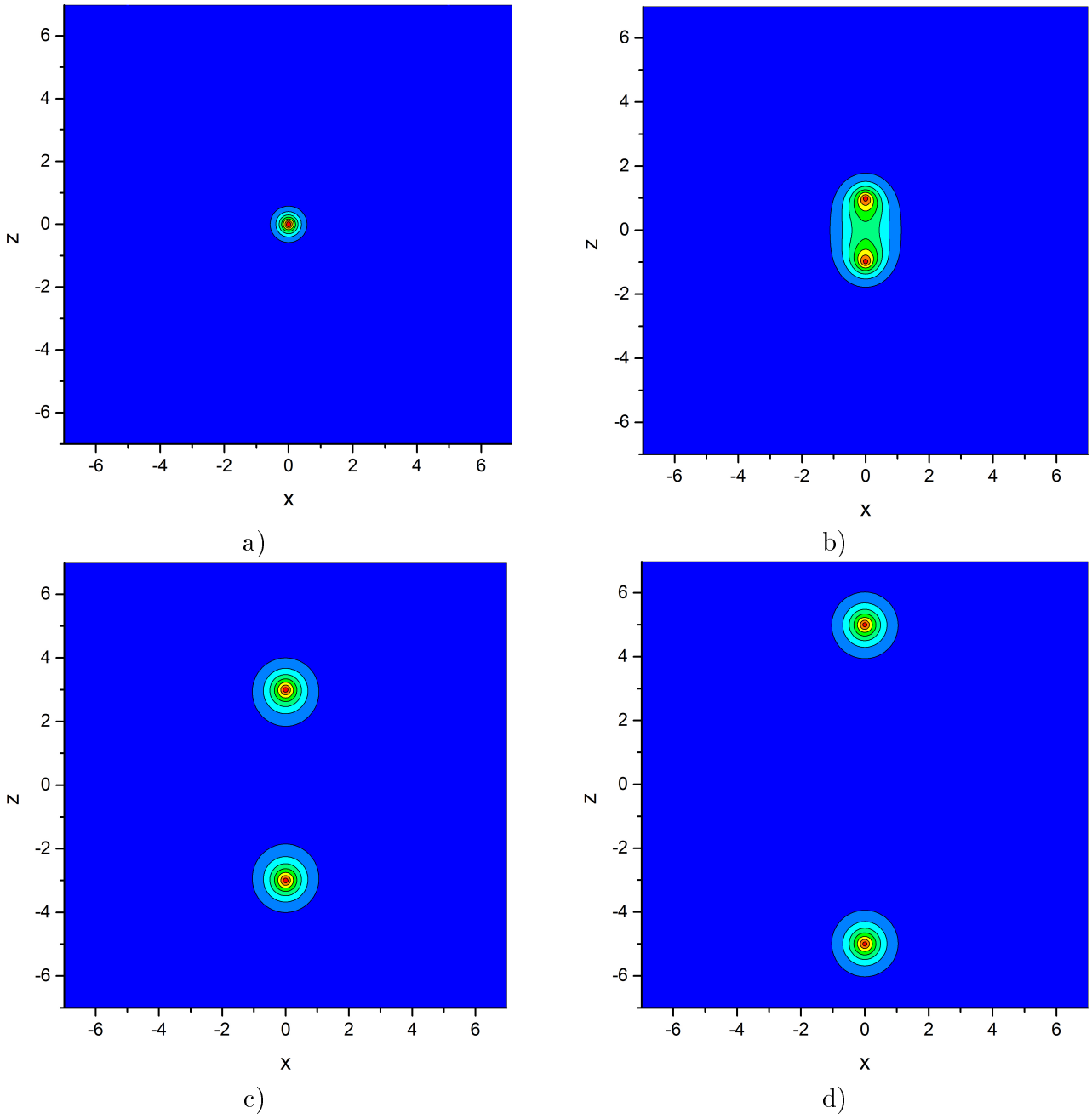

Рис. 1.6. Распределение плотности вероятности для системы $Z_1 = 1$, $Z_2 = 1$, $1s\sigma$: a) $R = 0,1$, b) $R = 2$, c) $R = 6$, d) $R = 10$. Расположение непожвижных зарядов совпадает с максимумами плотности

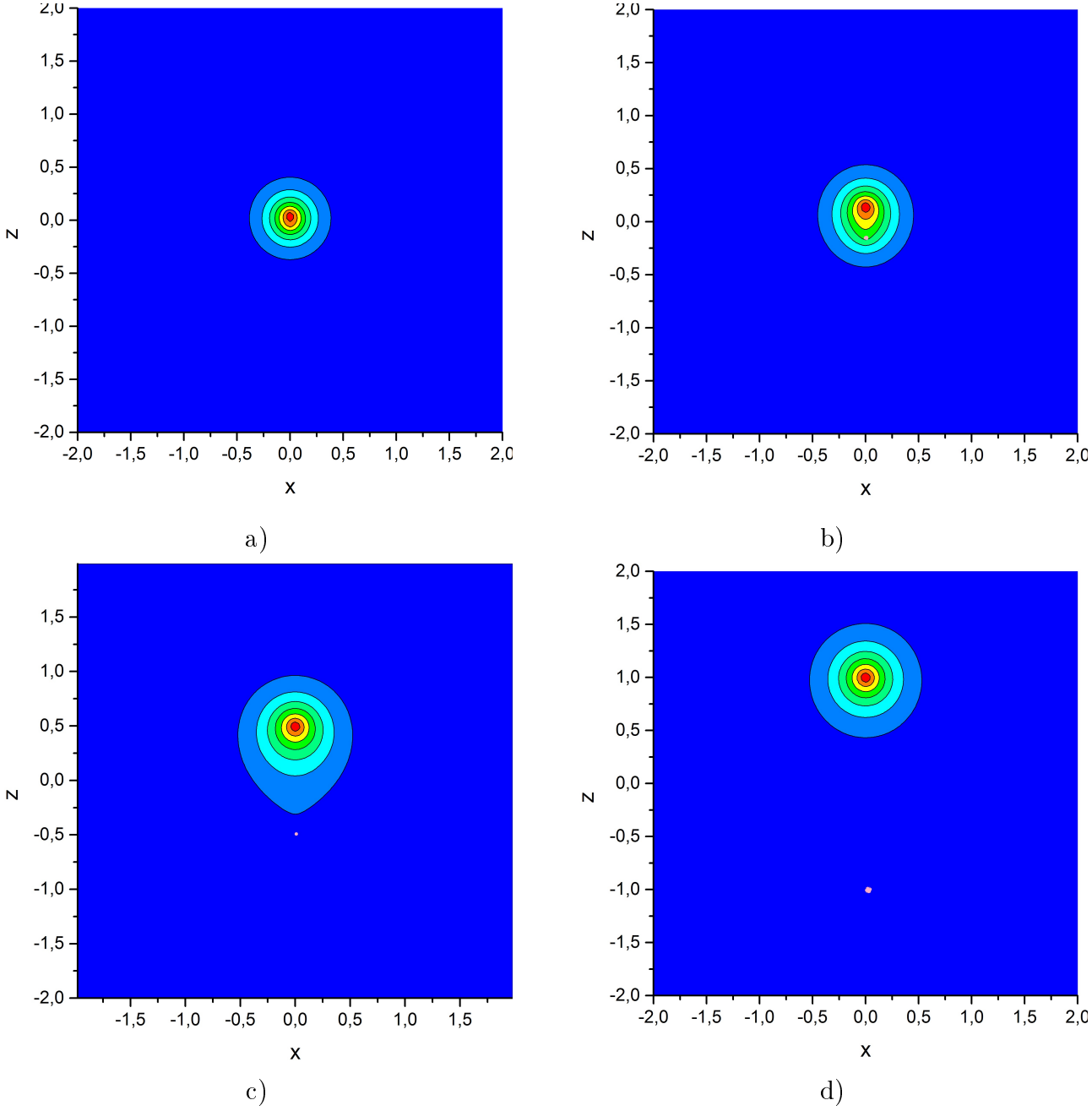

Рис. 1.7. Распределение плотности вероятности для системы $Z_1 = 1$, $Z_2 = 2$, $1s\sigma$: a) $R = 0,1$, b) $R = 0.3$, c) $R = 1$, d) $R = 2$. Положение заряда 2 соответсвует максимуму плотности, положение заряда 1 отмечено розовой точкой.

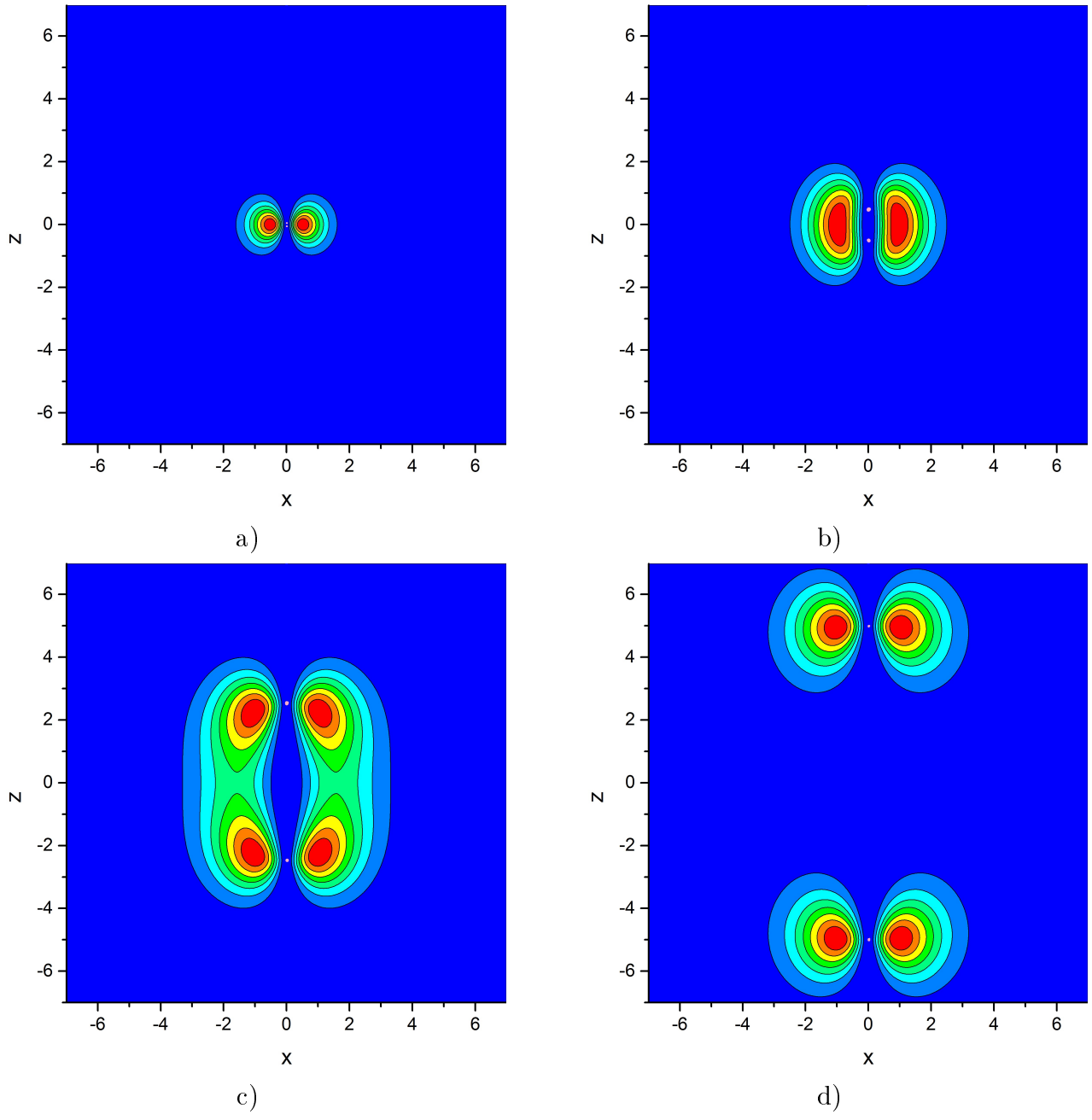

Рис. 1.8. Распределение плотности вероятности для системы $Z_1 = 2$, $Z_2 = 2$, $2p\pi$: a) $R = 0,1$, b) $R = 2$, c) $R = 5$, d) $R = 10$. Положение зарядов отмечено розовыми точками.

Глава 2

Адиабатическое разложение волновой функции

Рассматриваемая система состоит из трёх частиц массами M_1, M_2 и M_3 и зарядами Z_1e, Z_2e, $-e$ соответственно (Рис. 2.1).

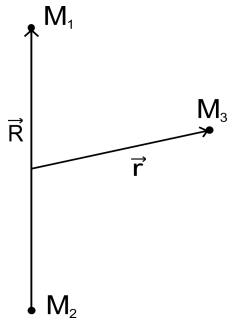

Рис. 2.1. Схема

Стационарное уравнение Шрёдингера в этом случае имеет вид

$$\left[-\frac{\hbar^2}{2}\left(\frac{1}{M_1}\Delta_1+\frac{1}{M_2}\Delta_2+\frac{1}{M_3}\Delta_3\right)+\frac{Z_1Z_2e^2}{|\mathbf{r_1}-\mathbf{r_2}|}-\frac{Z_1e}{|\mathbf{r_1}-\mathbf{r_3}|}-\frac{Z_2e}{|\mathbf{r_2}-\mathbf{r_3}|}\right]\psi(\mathbf{r_1},\mathbf{r_2},\mathbf{r_3})= \\ =W\psi(\mathbf{r_1},\mathbf{r_2},\mathbf{r_3}) \quad (2.1)$$

Удобно перейти в систему координат Якоби и преобразовать уравнение (2.1) к виду

$$\left[-\frac{\hbar^2}{2}\left(\frac{1}{M_t}\Delta_c+\frac{1}{M}\Delta_{\mathbf{R}}+\frac{1}{m}\Delta_{\mathbf{r}}\right)+V\right]\psi(\mathbf{R},\mathbf{r},\mathbf{r_c})=W\psi(\mathbf{R},\mathbf{r},\mathbf{r_c}), \quad (2.2)$$

в котором

$$\mathbf{r_c}=\frac{M_1\,\mathbf{r_1}+M_2\,\mathbf{r_2}+M_3\,\mathbf{r_3}}{M_t},\quad \mathbf{R}=\mathbf{r_1}-\mathbf{r_2},\quad \mathbf{r}=\mathbf{r_3}-\frac{M_1}{M_1+M_2}\,\mathbf{r_1}-\frac{M_2}{M_1+M_2}\,\mathbf{r_2}, \\ \frac{1}{M}=\frac{1}{M_1}+\frac{1}{M_2},\quad \frac{1}{m}=\frac{1}{M_3}+\frac{1}{M_1+M_2},\quad M_t=M_1+M_2+M_3, \quad (2.3) \\ V=\frac{Z_1Z_2e^2}{|\mathbf{R}|}-\frac{Z_1e}{|\mathbf{r}-\frac{M_2}{M_1+M_2}\mathbf{R}|}-\frac{Z_2e}{|\mathbf{r}+\frac{M_1}{M_1+M_2}\mathbf{R}|}.$$

Зависимость волновой функции $\psi(\mathbf{R},\mathbf{r},\mathbf{r_c})$ от движения центра масс исключается стандартным образом

$$\psi(\mathbf{R},\mathbf{r},\mathbf{r_c})=\exp\left(i(\mathbf{P_c},\mathbf{r_c})/\hbar\right)\Psi(\mathbf{r},\mathbf{R}), \quad (2.4)$$

где $\mathbf{P_c}$ - импульс, связанный с движением центра масс.

Разложим волновую функцию по решениям задачи двух центров $\Phi_k(\mathbf{r};R)$, построен-

ных в предыдущей главе (1.1)

$$\Psi(\mathbf{r},\mathbf{R}) = \sum_k \Phi_k(\mathbf{r};R)\psi_k(\mathbf{R}). \tag{2.5}$$

Разложение (2.5) называется адиабатическим представлением задачи трёх тел, поскольку используется приближение Борна — Оппенгеймера. В этом приближении скорость электронного движения значительно выше скорости относительного движения ядер, поэтому движение электрона можно рассматривать при неподвижных ядрах.

Подстановка ряда (2.5) в исходное уравнение (2.2) с последующим усреднением по координатам $\mathbf{r}$ даёт бесконечную систему уравнений на функции $\psi_k(\mathbf{R})$

$$\left(-\frac{1}{2M}\Delta_{\mathbf{R}} + \frac{Z_1Z_2}{R} + E_k + C_{kk}(R)\right)\psi_k(\mathbf{R}) + \sum_j \left(H_{kj} + 2\mathbf{Q_{kj}}\nabla_{\mathbf{R}}\right)\psi_j(\mathbf{R}) = E\psi_k(\mathbf{R}). \tag{2.6}$$

В этом уравнении введены обозначения

$$\begin{aligned} C_{kj}(R) &= -\frac{1}{8M}\int \Phi_k^*(\mathbf{r};R)\Delta_{\mathbf{r}}\Phi_k(\mathbf{r};R)d\mathbf{r}, \\ \mathbf{Q_{kj}}(\mathbf{R}) &= -\frac{1}{2M}\int d\mathbf{r}\Phi_k^*(\mathbf{r};R)(-\nabla_{\mathbf{R}})\Phi_j(\mathbf{r};R) = -\mathbf{Q_{jk}}, \\ H_{kj}(\mathbf{R}) &= -\frac{1}{2M}\int d\mathbf{r}\Phi_k^*(\mathbf{r};R)\,\Delta_{\mathbf{R}}\,\Phi_j(\mathbf{r};R), \end{aligned} \tag{2.7}$$

В адиабатическом приближении недиагональных матричные элементы отбрасываются и система распадается на независимые уравнения для каждого состояния $\psi_k(\mathbf{R})$

$$\left(-\frac{1}{2M}\Delta_{\mathbf{R}} + \frac{Z_1Z_2}{R} + E_k + C_{kk}(R) + H_{kk}(\mathbf{R})\right)\psi_k(\mathbf{R}) = E\psi_k(\mathbf{R}). \tag{2.8}$$

Если $H_{kk}(\mathbf{R})$ не зависит от направления вектора $\mathbf{R}$ (что выполняется, например, для основного электронного состояния), то можно перейти к одномерному уравнению на радиальную функцию $\chi_j(R)$ известной подстановкой $\psi_j(\mathbf{R}) = \chi_j(R)Y_{Jm_j}(\theta_R,\varphi_R)/R$

$$\chi''_{j\nu}(R) + 2M\left[E_{j\nu} - E_j(R) - \frac{Z_1Z_2}{R} - (H_{jj}(R) + C_{jj}(R)) - \frac{J(J+1)}{2MR^2}\right]\chi_{j\nu}(R) = 0. \tag{2.9}$$

Здесь состояния нумеруются шестью квантовыми числами: $j = (Nlm)$ - от электронного движения и три $\nu = (vJm_j)$ от ядерного. m_j - проекция полного момента J на вдоль оси, проходящей через заряды Z_1 и Z_2, а v - вибрационное квантовое число, равное числу нулей функции $\chi_{j\nu}(R)$.

Краевые условия выбираются естественным образом

$$\begin{cases} \chi_{j\nu}(0) = 0; & (2.10) \\ \chi_{j\nu}(\infty) = 0. & (2.11) \end{cases}$$

2.1. Нахождение колебательного спектра для основного электронного состояния

Найдём спектр для основного электронного состояния $\{100\}$. Уравнение (2.9) решается методом конечных разностей.

Величина $H(R)$ вычисляется следующим образом [14]

$$(-2M)H_{11}(\mathbf{R}) = \int d\mathbf{r}\Phi^*_{100}(\mathbf{r};R)\,\Delta_{\mathbf{R}}\,\Phi_{100}(\mathbf{r};R) =$$
$$= \int \Phi^*_{100}(\mathbf{r};R)\,\frac{\partial^2\Phi_{100}(\mathbf{r};R)}{\partial R^2}d\mathbf{r} - \frac{1}{R^2}\int \Phi^*_{100}(\mathbf{r};R)\left(L_x^2+L_y^2\right)\Phi_{100}(\mathbf{r};R)d\mathbf{r}, \quad (2.12)$$

где L_x и L_y операторы углового момента относительно осей x и y прямоугольной системы координат $\{xyz\}$, ось z которой направлена вдоль вектора $\mathbf{R}$. Для вычисления этого выражения используем равенство

$$L_x^2 + L_y^2 = \mathbf{L}^2 - L_z^2$$

и известное выражение для квадрата углового момента и его компоненты

$$\mathbf{L}^2 = -\left(\tfrac{1}{\sin\theta}\tfrac{\partial}{\partial\theta}\left[\sin\theta\tfrac{\partial}{\partial\theta}\right] + \tfrac{1}{\sin^2\theta}\tfrac{\partial^2}{\partial\varphi^2}\right),$$
$$L_z = -i\tfrac{\partial}{\partial\varphi}.$$

С учетом $\frac{\partial}{\partial\varphi}\Phi_{100}(\mathbf{r};R) = 0$ окончательно получаем:

$$(-2M)H_{11}(\mathbf{R}) =$$
$$= \int \Phi^*_{100}(\mathbf{r};R)\,\frac{\partial^2\Phi_{100}(\mathbf{r};R)}{\partial R^2}d\mathbf{r} - \frac{1}{R^2}\int \Phi^*_{100}(\mathbf{r};R)\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left[\sin\theta\frac{\partial}{\partial\theta}\right]\right)\Phi_{100}(\mathbf{r};R)d\mathbf{r}. \quad (2.13)$$

Производные по R и θ вычисляются стандартно конечно-разностным методом по трём точкам. Шаг h для производной по R при этом составляет 0.02 для промежутка $0.1 \leqslant R \leqslant 10$, и 0.1 для $10 < R \leqslant 15$. Интегралы вычисляются адаптивным методом, используя квадратурную формулу Гаусса-Кронрода с 15 точками. Этот метод реализован в библиотеке gsl(C++) и используется для вычислений на дискретном наборе значений R. Параметры: с относительной ошибкой не более 0.01 и максимальным количеством подынтервалов 100000.

Шаг для производной по θ составляет 0.001.

Величина $C(R)$ вычисляется по формуле (2.7)

$$(-2M)C_{11}(R) = \frac{1}{4}\int \Phi^*_{100}(\mathbf{r};R)\Delta_{\mathbf{r}}\Phi_{100}(\mathbf{r};R)d\mathbf{r} =$$
$$= \frac{1}{4}\int \Phi^*_{100}(\mathbf{r};R)\left[-2E_{100} + \left(-\frac{Z_1}{r_1} - \frac{Z_2}{r_2}\right)\right]\Phi_{100}(\mathbf{r};R)d\mathbf{r} = \begin{bmatrix}\mathbf{r_1} = \mathbf{r} + \mathbf{R}/2\\ \mathbf{r_2} = \mathbf{r} - \mathbf{R}/2\end{bmatrix} =$$
$$= \frac{1}{4}\left[-2E_{100} - \int \Phi^*_{100}(\mathbf{r_1};R)\frac{Z_1}{r_1}\Phi_{100}(\mathbf{r_1};R)d\mathbf{r_1} - \int \Phi^*_{100}(\mathbf{r_2};R)\frac{Z_2}{r_2}\Phi_{100}(\mathbf{r_2};R)d\mathbf{r_2}\right].$$

Данный интеграл вычисляется с теми же параметрами, что и величина $H(R)$. Сравнивая результаты со значениями из статьи [15], приходим к выводу, что они отличаются друг от друга на величину порядка 10^{-7} а. ед. энергии. Ниже представлены графики.

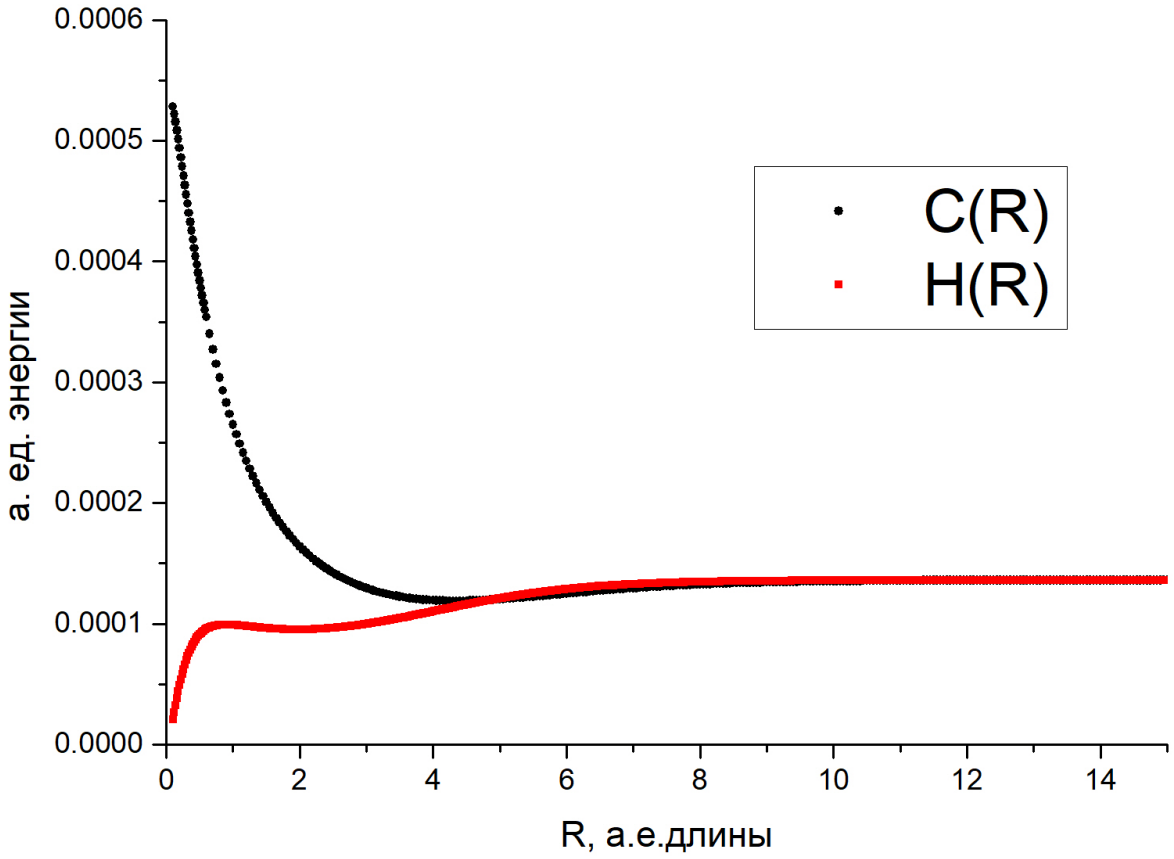

Рис. 2.2. Графики функций H(R) и C(R)

Значения терма электрона основного состояния $E_{100}(R)$ при $0.1 \leqslant R < 15$ вычисляются методом, описанным в первой главе. Они совпадают до 7 знака со значениями, используемыми [16]. Согласно статье [17], в качестве энергий для расстояний $R \geqslant 15$ можно использовать приближение $E_{100}(R) = -0.5 - 1/R - 9/(4R^2)$, а для $R \leqslant 0.1$ $E_{100}(R) = -2 + 8/3R^2$.

Собственные значения будут сравниваться со статьёй [16], поэтому значение параметра M выбирается 1836.09/2. Ограничивая правое краевое условие $R(600) = 0$ и выбирая шаг дискретизации 0.02, получим спектр для состояний моментами $J = 0$ и $J = 1$ с помощью функции “eigs” в Matlab.

Таблица 2.1. Спектр системы для первых вращательных уровней $J = 0$ и $J = 1$. Значения приводятся до 6 знаков после запятой, так как, предположительно, что погрешность вычисления потенциала вместе с поправками не превышает 10^{-6}.

	$J = 0$		$J = 1$	
ν	данная работа	H. Wind [16]	данная работа	H. Wind
0	-0.597140	-0.59713932	-0.596874	-0.59687398
1	-0.587160	-0.58715483	-0.586908	-0.58690345
2	-0.577762	-0.57775005	-0.577524	-0.57751216
3	-0.568928	-0.56890573	-0.568703	-0.56868092
4	-0.560636	-0.56060565	-0.560426	-0.56039359
5	-0.552881	-0.55283645	-0.552681	-0.55263686
6	-0.545643	-0.54558772	-0.545455	-0.54540041
7	-0.538917	-0.53885193	-0.538742	-0.53867676
8	-0.532698	-0.53262447	-0.532535	-0.53246140
9	-0.526984	-0.52690388	-0.526833	-0.52675294
10	-0.521777	0.52169191	-0.521638	-0.52155322
11	-0.517081	-0.51699381	-0.516954	-0.51686762
12	-0.512904	-0.51281871	-0.512790	-0.51270538
13	-0.509261	-0.50918001	-0.509161	-0.50908006
14	-0.506169	-0.50609594	-0.506083	-0.50601008
15	-0.503652	-0.50359010	-0.503580	-0.50351932
16	-0.501738	-0.50169188	-0.501683	-0.50163753
17	-0.500463	-0.50043456	-0.500426	-0.50039855
18	-0.499850	-0.49983658	-0.499830	-0.49982099
19	-0.499748	-0.49973177	-0.499746	-0.49972943

Как видно из таблицы, нижний уровень совпадает в пределах ожидаемой точности. Для более высоких уровней такого хорошего совпадения нет.

Для иллюстрации того, где локализованы волновые функции, построим графики некоторых из них.

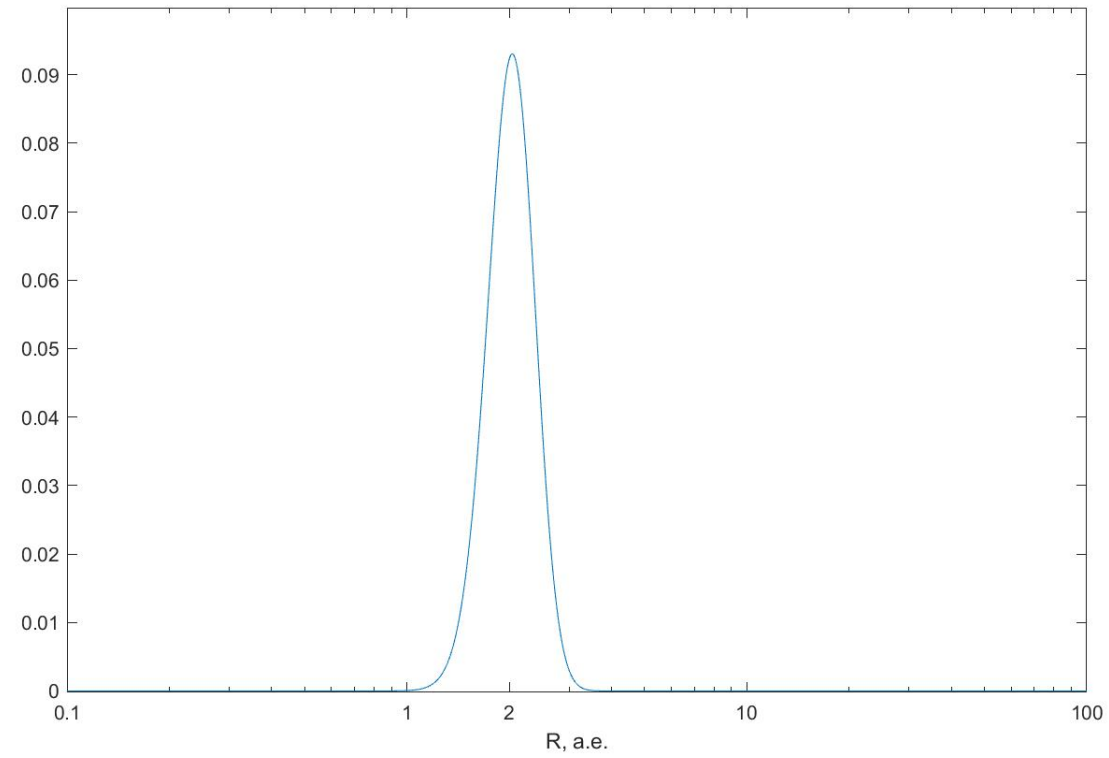

Рис. 2.3. Волновая функция $\chi(R)$ основного состояния для $J = 0$

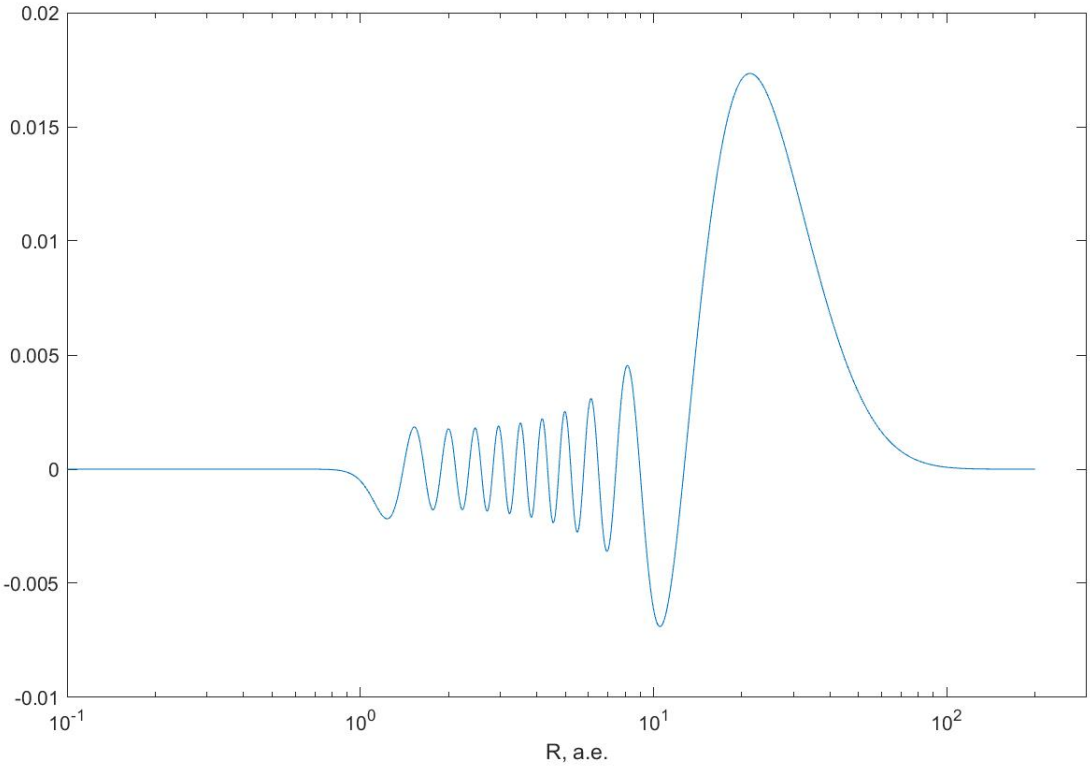

Рис. 2.4. Волновая функция $\chi(R)$ 19 колебательного уровня для $J = 0$

Глава 3

Решение задачи методом конечных элементов

Определим приведённые координаты Якоби $\mathbf{x}$ и $\mathbf{y}$ следующим образом

$$\mathbf{x} = \sqrt{2M}\,\mathbf{R} \tag{3.1}$$

$$\mathbf{y} = \sqrt{2m}\,\mathbf{r}, \tag{3.2}$$

где $\mathbf{R}$, $\mathbf{r}$, M и r определены ранее (2.3). Выражение для Гамильтониана в соответствующих координатах

$$H = H_0 + V(\mathbf{x}, \mathbf{y}) = -\Delta_{\mathbf{x}} - \Delta_{\mathbf{y}} + V(\mathbf{x}, \mathbf{y}), \tag{3.3}$$

Для исследования свойств симметрии удобно перейти в следующее представление для состояния системы

$$\{\mathbf{x}, \mathbf{y}\} = \{x, y, \theta, \Omega\}, \tag{3.4}$$

где θ - угол между векторами $\mathbf{x}$ и $\mathbf{y}$, $\Omega = \{\phi, \vartheta, \varphi\}$ - углы Эйлера, характеризующие ориентацию плоскости, содержащей все три частицы, по отношению к неподвижной системе координат.

Свободный гамильтониан H_0 имеет вид [18]

$$H_0 = -\frac{\partial^2}{\partial y^2} - \frac{2}{y}\frac{\partial}{\partial y} - \frac{\partial^2}{\partial x^2} - \frac{2}{x}\frac{\partial}{\partial x} - \left(\frac{1}{y^2} + \frac{1}{x}\right)\left(\frac{\partial^2}{\partial \theta^2} + \cot\theta\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\varphi^2}\right) + \frac{1}{y^2}\left(\mathbf{J}^2 - \mathbf{K}\right). \tag{3.5}$$

Здесь оператор $\mathbf{K}$ описывает кориолисово взаимодействие, возникающее из-за вращения трёхчастичной системы как целого [19]

$$\mathbf{K} = -2\frac{\partial^2}{\partial\varphi^2} + \left\{\sqrt{2}\frac{\partial}{\partial\vartheta}\left(\mathbf{J}^+ + \mathbf{J}^-\right) + \sqrt{2}\cot\vartheta\left(\mathbf{J}^+ - \mathbf{J}^-\right)\frac{\partial}{\partial\varphi}\right], \tag{3.6}$$

где

$$\mathbf{J}^{\pm} = \frac{i}{\sqrt{2}}e^{\mp i\varphi}\left\{\pm\cot\vartheta\frac{\partial}{\partial\varphi} + i\frac{\partial}{\partial\vartheta} \mp \frac{1}{\sin\vartheta}\frac{\partial}{\partial\phi}\right]. \tag{3.7}$$

Со свободным гамильтонианом H_0 коммутируют три оператора:

1. Квадрат полного углового момента $\mathbf{J}^2$

$$\mathbf{J}^2 = -\left[\frac{\partial^2}{\partial\vartheta^2} + \cot\vartheta\frac{\partial}{\partial\vartheta} + \frac{1}{\sin^2\vartheta}\left(\frac{\partial^2}{\partial\phi^2} - 2\cos\vartheta\frac{\partial^2}{\partial\phi\partial\varphi} + \frac{\partial^2}{\partial\varphi^2}\right)\right];$$

2. Проекция полного углового момента на ось лабораторной системы координат $\mathbf{J}_z$

$$\mathbf{J}_z = -i\frac{\partial}{\partial\phi};$$

3. Оператор полной инверсии координат P

$$P\Psi(x,y,\cos\theta,\phi,\vartheta,\varphi)=\Psi(x,y,\cos\theta,\pi+\phi,\pi-\vartheta,\pi-\varphi).$$

Поскольку потенциал взаимодействия $V(\mathbf{x},\mathbf{y})$ не зависит от Эйлеровских углов (ϕ, ϑ, φ), то эти операторы коммутируют также с полным гамильтонианом H. Следовательно, их собственные функции образуют одинаковые множества. Элементами этого множества являются симметризованные D-функции Вигнера $\mathcal{D}^{J\tau}_{MM'}$:

$$\begin{aligned}
\mathbf{J}^2\,\mathcal{D}^{J\tau}_{MM'} &= J(J+1)\,\mathcal{D}^{J\tau}_{MM'}, \quad J=0,1,\ldots\\
\mathbf{J}_z\,\mathcal{D}^{J\tau}_{MM'} &= -M\,\mathcal{D}^{J\tau}_{MM'}, \quad M=-J,\ldots,J\\
P\,\mathcal{D}^{J\tau}_{MM'} &= \tau\,\mathcal{D}^{J\tau}_{MM'}, \quad \tau=\pm(-1)^J,
\end{aligned}$$

которые выражены через стандартные D-функции Вигнера [20] следующим образом

$$\mathcal{D}^{J\tau}_{MM'}(\phi,\theta,\varphi)=\frac{(-1)^M}{\sqrt{2+2\delta_{M'0}}}\left((-1)^M\,\mathcal{D}^{J}_{MM'}+\tau(-1)^J\,\mathcal{D}^{J}_{M-M'}\right). \tag{3.8}$$

Волновая функция системы $\Psi(\mathbf{x},\mathbf{y})$, характеризующаяся полным угловым моментом J, его проекцией M и чётностью τ, представляется в виде

$$\Psi^{J\tau}_{M}(\mathbf{x},\mathbf{y})=\sum_{M'=0}^{J}(\mathcal{D}^{J\tau}_{MM'})^*(\phi,\vartheta,\varphi)\psi^{J\tau}_{M'}(x,y,\theta). \tag{3.9}$$

Подстановка представления (3.9) в уравнение Шрёдингера $H\Psi=E\Psi$ и проецирование на D-функции (3.8) даст систему уравнений

$$\sum_{M'=0}^{J}\left(H^{J\tau}_{MM'}-E\delta_{MM'}\right)\psi^{J\tau}_{M'}(x,y,\theta)=0, \quad M=0,...,J \tag{3.10}$$

Диагональные элементы матричного оператора $H^{J\tau}_{MM'}$ заданы формулами

$$\begin{aligned}
H^{J\tau}_{MM}=&\left(1+(1/2)(\tau(-1)^J-1)\delta_{M0}\right)\times\\
&\times\left[-\frac{1}{x}\frac{\partial^2}{\partial x^2}x-\frac{1}{y}\frac{\partial^2}{\partial y^2}y+\frac{J(J+1)-2M^2}{y^2}+V(x,y,\theta)-\right.\\
&\left.-\left(\frac{1}{x^2}+\frac{1}{y^2}\right)\left(\frac{\partial^2}{\partial\theta^2}+\cot\theta\frac{\partial}{\partial\theta}-\frac{M^2}{\sin^2\theta}\right)\right],
\end{aligned} \tag{3.11}$$

а внедиагональные компоненты отличны от нуля только если $M' = M + 1$ и равны

$$H^{J\tau}_{MM'} = \left(1 + (1/2)(\tau(-1)^J - 1)(\delta_{M0} + \delta_{M'0})\right) \times \\ \times \left[\pm \frac{\sqrt{J(J+1) - M(M \pm 1)}}{y^2} \left(\frac{\partial}{\partial\theta} + (1 \pm M)\cot\theta\right)\right]. \quad (3.12)$$

Система (3.10) в случае положительной чётности $\tau = (-1)^J$ состоит из $(J+1)$ уравнений, в случае отрицательной чётности состоит из J уравнений, так как $\mathcal{D}^{J-}_{M0}) \equiv 0$ и суммирование в (3.9) происходит начиная с $M' = 1$.

Осталось добавить граничные условия. В силу ограниченности полной волновой функции квантовой системы, функции $\psi^{J\tau}_{M'}(x, y, \theta)$ должны быть ограничены про $x = 0$, $y = 0$. В силу квадратичной интегрируемости, требуются условия убывания на бесконечности $\psi^{J\tau}_{M'}(x, y, \theta) = 0$ при $x \to \infty$ или $y \to \infty$. По переменной θ граничное условие можно сформулировать следующим образом: функция $\psi^{J\tau}_{M'}(x, y, \theta)/\sin^{M'}\theta$ должна быть ограничена.

3.1. Вариационная формулировка

Для использования метода конечных элементов необходимо переписать уравнение (3.10) в вариационной формулировке. Она формулируется следующим образом [18]: найти $\Psi \in \mathbf{H}^1(x \times y \times \theta)$ такую, что для любой $\widetilde{\Psi} \in \mathbf{H}^1(x \times y \times \theta)$ выполняется равенство:

$$\hat{H}(\Psi, \widetilde{\Psi}) - E\hat{S}(\Psi, \widetilde{\Psi}) = 0. \quad (3.13)$$

Функция Ψ состоит из компонент $\psi^{J\tau}_{M'}$, билинейные формы $\hat{H}(\Psi, \widetilde{\Psi})$ и $\hat{S}(\Psi, \widetilde{\Psi})$ являются матрицами по индексам компонент. Диагональные матричные элементы равны

$$\hat{H}_{MM}(\Psi, \widetilde{\Psi}) = \int dZ \left(\frac{\partial\psi^{J\tau}_M}{\partial x}\frac{\partial\widetilde{\psi}^{J\tau}_M}{\partial x} + \frac{\partial\psi^{J\tau}_M}{\partial y}\frac{\partial\widetilde{\psi}^{J\tau}_M}{\partial y} + \right. \\ \left. + \left[\frac{J(J+1) - 2M^2}{y^2} + V(x, y, \theta)\right]\psi^{J\tau}_M\widetilde{\psi}^{J\tau}_M + \left[\frac{1}{x^2} + \frac{1}{y^2}\right]\left[\frac{\partial\psi^{J\tau}_M}{\partial\theta}\frac{\partial\widetilde{\psi}^{J\tau}_M}{\partial\theta} + \psi^{J\tau}_M\widetilde{\psi}^{J\tau}_M\frac{M^2}{\sin^2\theta}\right]\right); \quad (3.14)$$

$$\hat{S}_{MM}(\Psi, \widetilde{\Psi}) = \int dZ\psi^{J\tau}_M\widetilde{\psi}^{J\tau}_M; \quad (3.15)$$

элементы на верхней и нижней субдиагонали матрицы $\hat{H}$

$$\hat{H}_{M\,M\pm1}(\Psi, \widetilde{\Psi}) = \pm \int dZ \left(H_{M\,M\pm1}\psi^{J\tau}_M\right)\widetilde{\psi}^{J\tau}_{M\pm1}; \quad (3.16)$$

остальные элементы матриц $\hat{H}(\Psi, \widetilde{\Psi})$ и $\hat{S}(\Psi, \widetilde{\Psi})$ нулевые. Интегрирование в приведённых интегралах ведётся по области $\int_0^\infty dx \int_0^\infty dy \int_0^\pi d\theta$ и $dZ = x^2 y^2 \sin\theta dx dy d\theta$. Исходная система (3.10) таким образом сводится к обобщённой задаче на собственные значения (3.13).

3.2. Результаты вычислений

Решение задачи (3.13) методом конечных элементов выполняется в программе ACESPA (fortran 90). По задаваемым разбиению области, базисным функциям и их количеству вычисляются собственные значения и собственные функции системы. Более подробно о работе программы можно найти в диссертации [18]. На основе результатов, получаемых в предыдущих главах можно определить, где локализована волновая функция и, соответственно, построить оптимальное разбиение области. Для получения наиболее точной энергии основного состояния была подобрана область (3.1)

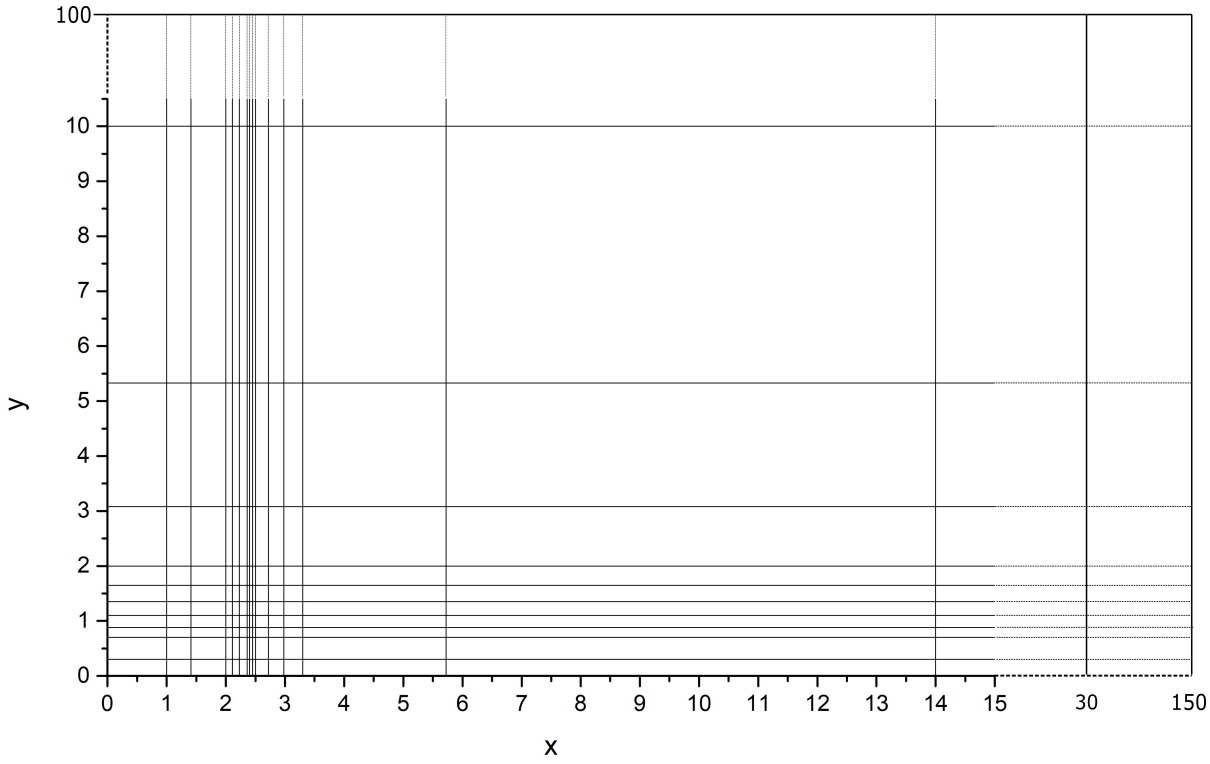

Рис. 3.1. Разбиение плоскости $x - y$ на конечные элементы. По осям отложены неприведённые координаты Якоби.

Таблица 3.1. Сравнение энергий основного состояния.

данная работа	-0.59713
J. Ph. Karr [21]	-0.59713906307939
Korobov V. I. [9]	-0.597139063123405074

Мы ожидаемо получили несколько завышенные значения, так как метод вариационный. Расчёты показали, наиболее сильное влияние оказывает степень полиномов и её погрешность оценивается в 10^{-4} для низких уровней. Для высоких уровней ещё хуже - порядка 10^{-3} а.ед.

В качестве проверки получаемых волновых функций можно вычислить различные физические величины. В таблице (3.2) приведены результаты вычисления некоторых величин, полученных с помощью квадратурных формул Гаусса - Лежандра.

Таблица 3.2. Сравнение средних величин, полученных для основного состояния. Область интегрирования: $[0, 15] \times [0, 10] \times [-1, 1]$ для неприведённых координат соответственно. Разбиение по каждой из координат осуществляется на $80, 40, 40$ точек соответственно.

Величина	на основе в.ф. из acespa	Bhatia [22]
$<1>$	0.999994	-
Обратное межъядерное расстояние $<1/R>$	0.491	0.49071
Межъядерное расстояние $<R>$	2.064	2.06392
Квадрат межъядерного расстояния $<R^2>$	4.314	4.31097
Квадрат расстояния между центром масс ядер и электроном $<r^2>$	2.481	2.48107
Расстояние от одного из ядер до электрона $<r_1>$	1.693	1.69297

Произведём подсчёты для двадцати колебательных уровней нулевого полного момента. Взято другое разбиение области (3.2). Результат вычисления энергий представлен в таблице 3.3.

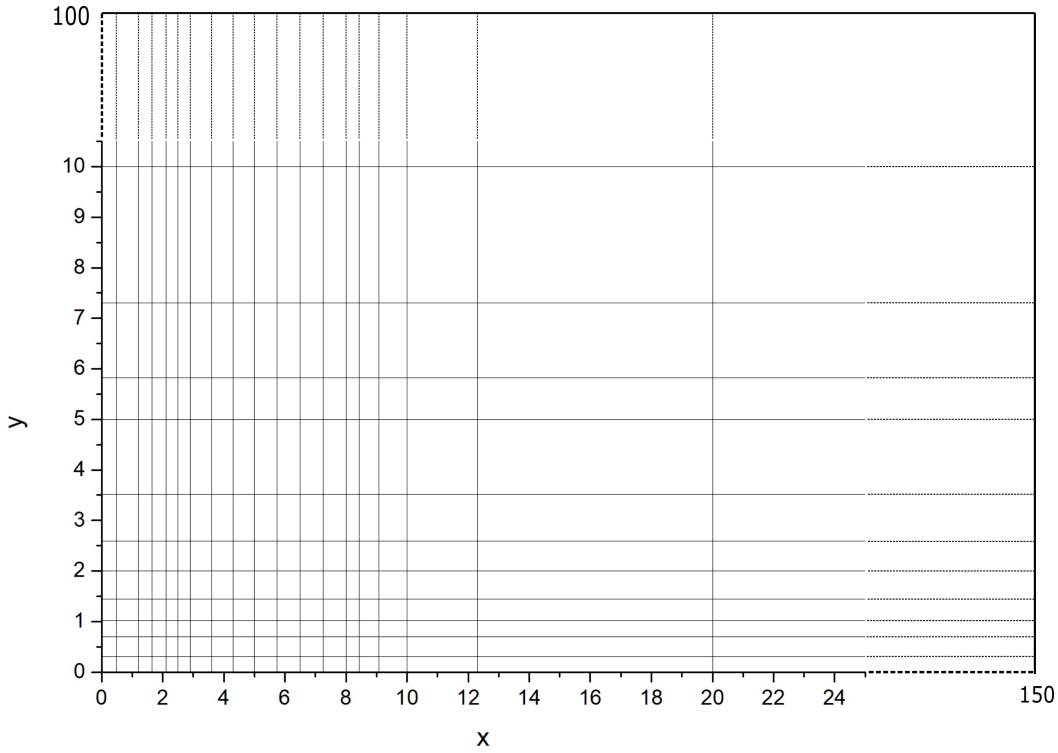

Рис. 3.2. Разбиение плоскости $x - y$ на конечные элементы. По осям отложены неприведённые координаты Якоби.

Таблица 3.3. Сравнение энергий для нулевого полного момента J=0

Колебательный уровень ν	данная работа	J. Ph. Karr [21]
0	-0.59711	-0.59713906307939
1	-0.58712	-0.58715567909619
2	-0.57771	-0.57775190441508
3	-0.56886	-0.56890849873086
4	-0.56054	-0.56060922084967
5	-0.55276	-0.55284074989655
6	-0.54551	-0.54559265099383
7	-0.53877	-0.53885738696741
8	-0.53253	-0.53263037935627
9	-0.52679	-0.52691012401632
10	-0.52157	-0.52169836901424
11	-0.51685	-0.51700036527875
12	-0.51261	-0.51282520314556
13	-0.50891	-0.50918624836829
14	-0.50579	-0.50610168096876
15	-0.50323	-0.50359508499922
16	-0.5011	-0.50169577338703
17	-0.4995	-0.50043704046015
18	-0.4986	-0.49983743203023
19	-0.4976	-0.4997312306492

Заключение

Таким образом, в настоящей работе были получены результаты:

- Реализован алгоритм для нахождения энергий и волновых функций задачи двух кулоновских центров, описанный в книге [11]. Сравнение с другим источником показывает (1.1), что точность вычисления энергий на небольших межъядерных расстояниях не хуже 10^{-9} а. ед. Из графического анализа, а также из вычисления различных поправок(во второй главе) на основе волновых функций можно сделать вывод о том, что волновые функции электрона строятся правильно.

- На основе результатов, получаемых данным алгоритмом были вычислены энергии и собственные функции иона водорода H_2^+ с помощью адиабатического разложения. Сравнение показывает, что нижние уровни совпадают с требуемой точностью(10^{-6}), однако вышележащие уровни имеют гораздо бóльшую погрешность.

- Произведены вычисления спектра в программе ACESPA, а также сравнение с результатами из статьи [21]. Можно сделать вывод, что в целом метод даёт правильные результаты, однако получены результаты с меньшей точностью, поскольку вычисления с помощью МКЭ требовательны к ресурсам вычислительной техники. Поэтому можно утверждать, что вычисления с бóльшими вычислительными ресурсами дали бы более точные результаты.

Список литературы

1. Guillemin V. Hydrogen-ion wave function / C. Zener // Proceedings of the National Academy of Sciences, 1929, Vol. 15, p. 314-318.
2. Jaffe G. Zur Theorie des Wasserstoffmolekiilions // Zeitschrift fur Physik, 1934, Vol. 87, p. 535–544.
3. Cohen S. Vibrational States of the Hydrogen Molecular Ion / J. R. Hiskes, R. J. Riddell // Physical Review, 1960, Vol. 119, num 3, p.1025 - 1027.
4. W. Kolos Nonadiabatic Theory for Diatomic Molecules and Its Application to the Hydrogen Molecule / L. Wolniewicz // Rewiev of Modern Physics, 1963, Vol. 35, Num. 3, p. 473-483.
5. L. Wolniewicz The $1s\sigma_g$ and $2p\sigma_u$ states of the H_2^+, D_2^+ and HD^+ ions / T. Orlikowski // Molecular Physics, 1991, Vol. 74, Num. 1, p. 103-111.
6. Moss R.E. Calculations for the vibration-rotation levels of H_2^+ in its ground and first excited electronic states // Molecular Physics, 1993, Vol. 80, Num. 6, p. 1541-1554.
7. G. G. Balint-Kurti Calculations of vibration-rotation energy levels of HD^+ / R. E. Moss, I. A. Sadler, M. Shapiro // Physical review A, 1990, Vol.41, Num. 9, p. 4913-4921.
8. Hilico L. Ab initio calculation of the $J = 0$ and $J = 1$ states of the H_2^+ , D_2^+ and HD^+ molecular ions / N. Billy, B. Gremaud, D. Delande // European physical jouranl D, 2000, Vol. 12, p. 449-466.
9. Korobov V. I. Coulomb three-body bound-state problem: Variational calculations of nonrelativistic energies // Physical Review A, 2000, Vol. 61, 064503.
10. Bailey D. H. Universal variational expansion for high-precision bound-state calculations in three-body systems. Applications to weakly bound, adiabatic and two-shell cluster systems / D. H. Bailey and A. M. Frolov // Journal of Physics B: Atomic, Molecular and Optical Physics, 2002, Vol. 35, p. 4287-4298.
11. Комаров И.В. Сфероидальные и кулоновские сфероидальные функции / Комаров И.В., Пономарев Л.И., Славянов С.Ю. - М.: Наука, 1976. - 320 с.
12. Scott T.C. New approach for the electronic energies of the hydrogen molecular ion / M. Aubert-Frecon, J. Grotendorst // Chemical Physics, 2006, Vol. 324, C. 323 - 338.
13. Hadinger G. The Killingbeck method for the one-electron two-centre problem / M. Aubert-Frecon, Gerold Hadinger // Journal of Physics: B Atomic Molecular & Optical Physics, 1989, № 5, C. 697 - 712.
14. R. McCarroll Adiabatic coupling between electronic and nuclear motion in molecules / A. Dalgarno // Proceedings of the Royal Society of London. Series A, 1956, Vol. 237, Num. 1210, p.383-394.
15. Kolos W. Some accurate results for three-particle systems // Acta Physica Academiae Scientiarum Hungaricae, 1969, num 27, p. 241-252.
16. Wind H. Vibrational States of the Hydrogen Molecular Ion // The journal of chemical physics, 1965, Vol. 43, num 9, p.2956 - 2958.

17. Wind H. Electron Energy for H_2^+ in the Ground State // The journal of chemical physics, 1965, Vol. 42, num 7, p.2371 - 2373.

18. Яревский Е. А. Единый аналитический и вычислительный подход к решению квантовой задачи трёх тел: Диссертация на соискание ученой степени доктора физико-математических наук. СПбГУ, Санкт-Петербург, 2017

19. Curtiss C. F. The Separation of the Rotational Coordinates from the N-Particle Schroedinger Equation / Hirschfelder J. O., Adler F. T. // The Journal of Chemical Physics. 1950. Vol. 18, no. 12. P. 1638–1642.

20. Варшалович Д.А. Квантовая теория углового момента / Варшалович Д.А., Москалев А.Н., Херсонский В.К. - Л.:Наука, 1975. - 441 с.

21. J. Ph. Karr High accuracy results for the energy levels of the molecular ions H_2^+ , D_2^+ and HD^+, up to $J = 2$ / L. Hilico // Journal of Physics B: Atomic, Molecular and Optical Physics, 2006, Vol. 39, p. 2095-2105.

22. Bhatia A. K. Properties of the ground state of the hydrogen molecular ion // Physical review A, 1998, Vol. 58, num 4, 2787-2789.

23. Cohen S. Mu-Mesozoic Molecules. I. Three-Body Problem / D. L. Judd, R. J. Riddell // Physical Review, 1959, Vol. 119, num 1, p.384 - 397.

Приложение А

Алгоритм задачи двух центров

Ниже приведён основной функционал алгоритма.

```
1  //вычисление P[N]/Q_N(N,q,m,p,b,lambda)
2  double fract_for_eta(double p, double b, double lambda)
3  {
4      //постоим последовательность из N+1 элементов rho[] для заданных q, m и p
5      double* rho = build_rho_for_lambda( b, p);
6      //постоим последовательность из N+2 элементов delta[] для заданных q, m и p
7      double* delta = build_delta_for_lambda(b, p);
8
9      //построим массив xi по значению lambda
10     double* chi = new double[N+1];
11     for (int i = 0; i < N+1; i++) chi[i] = (i + m)*(i + m + 1) - lambda;
12
13     double P[2], Q[2];
14
15     P[-1 + OFFSET2] = 1;
16     P[-2 + OFFSET2] = chi[0];
17     Q[-1 + OFFSET2] = 0;
18     Q[-2 + OFFSET2] = 1;
19     for (int i = 1; i <= N; i++)
20     {
21         P[i \% 2] = P[(i + 1) \% 2] * chi[i] - P[i \% 2] * delta[i] * rho[i -
               1];
22         Q[i \% 2] = Q[(i + 1) \% 2] * chi[i] - Q[i \% 2] * delta[i] * rho[i -
               1];
23         if (abs(P[i \% 2]) > 1E120 || abs(Q[i \% 2]) > 1E120)
24         {
25             P[0] *= 1E-120;
26             P[1] *= 1E-120;
27             Q[0] *= 1E-120;
28             Q[1] *= 1E-120;
29         }
30     }
31     delete[] rho;
32     delete[] delta;
33     delete[] chi;
34     return P[N \% 2]/ sqrt(1 + Q[N \% 2] * Q[N \% 2]);
35 }
36
37 //постоение последовательности из N элементов rho[] для заданных q, m и p
38 double* build_rho_for_lambda(double b, double p)
39 {
```

```
40      double* rho = new double[N + 1];
41      for (int i = 0; i < N+1; i++) rho[i] = (i + 2 * m + 1) * (b - 2 * p*(i + m +
            1)) / (2 * (i + m) + 3);
42      return rho;
43  }
44
45  //постоение последовательности из N элементов rho[] для заданных q, m и p
46  double* build_delta_for_lambda(double b, double p)
47  {
48      double* delta = new double[N + 1];
49      for (int i = 0; i < N+1; i++) delta[i] = (i) * (b + 2 * p*(i + m)) / (2 * (i
            + m) - 1);
50      return delta;
51  }
52
53  // build_rho, build_delta, build_chi отличаются от тех же с _for_lambda
54  // sign, чтобы использовать теже коэфф для c_i и c'_i
55  double* build_rho(double b, double p, int sign)
56  {
57      double* rho = new double[N + 1];
58      for (int i = 0; i < N + 1; i++) rho[i] = 2 * (i+1) * (i+m+1);
59      return rho;
60  }
61
62  double* build_delta(double b, double p, int sign)
63  {
64      double* delta = new double[N + 1];
65      for (unsigned i = 0; i < N + 1; i++) delta[i] = sign * b + 2*p*(i+m);
66      return delta;
67  }
68
69  double* build_chi(double b, double p, double lambda, int sign)
70  {
71      double* chi = new double[N + 1];
72      for (int i = 0; i < N + 1; i++) chi[i] = i*(i+1) + (2*i+m+1)*(2*p+m) + sign
            * b - lambda;
73      return chi;
74  }
75
76  //нахождение лямбда, удовлетворяющее ур-ию P/Q(0,a или b,lambda) = 0 вблизи
        begin_lambda
77  double find_lambda_from_b_for_eta_near(double b, double begin_lambda)
78  {
79      if (fract_for_eta(0, b, begin_lambda) == 0) return begin_lambda;
80      double step_l = 0.0005;
81      bool sign = (fract_for_eta(0, b, begin_lambda) > 0);
82      double left_l = begin_lambda;
```

```
83      double right_l = begin_lambda;
84      do
85      {
86          left_l -= step_l;
87          right_l += step_l;
88      } while (fract_for_eta(0, b, left_l) > 0 == sign && fract_for_eta(0, b,
            right_l) > 0 == sign);
89      if (fract_for_eta(0, b, left_l) > 0 != sign) begin_lambda =
            dividing_section_for_eta(0, b, left_l, left_l + step_l, !sign);
90      else begin_lambda = dividing_section_for_eta(0, b, right_l - step_l, right_l
            , sign);
91      return begin_lambda;
92  }
93
94  double find_lambda_from_a_for_xi_near(double p, double a, double begin_lambda)
95  {
96      double step_l = 0.1 > abs(begin_lambda)*5E-4 ? 0.1 : abs(begin_lambda)*5E-4;
97      bool sign = (fract_for_xi(p, a, begin_lambda - step_l) > 0);
98      double right_l = begin_lambda;
99      do
100     {
101         right_l += step_l;
102     } while (fract_for_xi(p, a, right_l) > 0 == sign);
103     return dividing_section_for_xi(p, a, right_l - step_l, right_l, sign);
104
105 }
106
107 double dividing_section_for_xi(double p, double a, double left, double right,
        bool sign)
108 {
109     do
110     {
111         double center = (right + left) / 2;
112         double val_center = fract_for_xi(p, a, center);
113         if ((right - left) / abs(center) < 1E-15) return center;
114         else if (val_center > 0 == sign) left = center;
115         else right = center;
116     } while (true);
117 }
118
119
120 //нахождение корня на отрезке [left, right](если знак полинома на левой стороне
        равен sign) методом деления пополам
121 double dividing_section_for_eta(double p, double b, double left, double right,
        bool sign)
122 {
123     do
```

```
124     {
125         double center = (right + left) / 2;
126         double val_center = fract_for_eta(p, b, center);
127         if (floor(val_center * 1E10) / 1E10 == 0 || (right - left) / abs(center)
                < 1E-13) return center;
128         else if (val_center > 0 == sign) left = center;
129         else right = center;
130     } while (true);
131 }
132
133 //для радиальной функции \Pi(p,a,lambda)
134 double* build_alpha_for_lambda(double a, double p)
135 {
136     double* alpha = new double[N + 1];
137     for (unsigned i = 0; i < N + 1; i++) alpha[i] = (i + 1)*(i + m + 1);
138     return alpha;
139 }
140
141 double* build_gamma_for_lambda(double a, double p)
142 {
143     double* gamma = new double[N + 1];
144     double sigma = (a / 2) / p - (m + 1);
145     for (int i = 0; i < N + 1; i++) gamma[i] = (i-1-sigma)*(i-m-1-sigma);
146     return gamma;
147 }
148
149 double* build_gamma_right(double a, double p)
150 {
151     double* gamma = new double[N + 1];
152     double sigma = (a / 2) / p - (m + 1);
153     for (int i = 0; i < N + 1; i++) gamma[i] = -(i - 1) * (i - 2 * sigma + m -
            3) + sigma * (sigma + m);
154     return gamma;
155 }
156
157 //для вычисления волновой функции рядом с полиномами Лагерра
158 double* build_alpha_Lagerr(double a, double p)
159 {
160     double* alpha = new double[N + 1];
161     double sigma = (a / 2) / p - (m + 1);
162     for (int i = 0; i < N + 1; i++) alpha[i] = (i + m + 1)*(i - m - sigma);
163     return alpha;
164 }
165
166 double* build_gamma_Lagerr(double a, double p)
167 {
168     double* gamma = new double[N + 1];
```

```
169     double sigma = (a / 2) / p - (m + 1);
170     for (int i = 0; i < N + 1; i++) gamma[i] = i * (i - 1 - sigma);
171     return gamma;
172 }
173
174 //вычисление P_N/Q_N(N,k,m,p,a,lambda)
175 double fract_for_xi(double p, double a, double lambda)
176 {
177     //постоим последовательность из N+1 элементов rho[] для заданных k, m, a и p
178     double* alpha = build_alpha_for_lambda(a, p);
179     //постоим последовательность из N+2 элементов delta[] для заданных k, m, a и
            p
180     double* gamma = build_gamma_for_lambda(a, p);
181     //построим массив beta по значению lambda
182     double* beta = build_beta(a, p, lambda);
183     double P[2], Q[2];
184     P[-1 + OFFSET2] = 1;
185     P[-2 + OFFSET2] = beta[0];
186     Q[-1 + OFFSET2] = 0;
187     Q[-2 + OFFSET2] = 1;
188     for (unsigned i = 1; i <= N; i++)
189     {
190         P[i \% 2] = (P[(i + 1) \% 2] * beta[i] - P[i \% 2] * alpha[i - 1] *
                gamma[i]);
191         Q[i \% 2] = Q[(i + 1) \% 2] * beta[i] - Q[i \% 2] * alpha[i - 1] * gamma
                [i];
192         if (abs(P[i \% 2]) > 1E120 || abs(Q[i \% 2]) > 1E120)
193         {
194             P[0] *= 1E-120;
195             P[1] *= 1E-120;
196             Q[0] *= 1E-120;
197             Q[1] *= 1E-120;
198         }
199     }
200     delete[] alpha;
201     delete[] gamma;
202     delete[] beta;
203     return P[N \% 2]/sqrt(1 + Q[N \% 2]* Q[N \% 2]);
204 }
205
206 double* build_beta(double a, double p, double lambda)
207 {
208     double* beta = new double[N + 1];
209     double sigma = (a / (2 * p)) - (m + 1);
210     for (int i = 0; i < N + 1; i++) beta[i] = 2 * i*(i + 2 * p - sigma) - (m +
            sigma)*(m + 1) - 2 * p*sigma + lambda;
211     return beta;
```

```
212 }
213 //начальное p из теории возмущений
214 double p_from_perturbation_theory(double r)
215 {
216     int Z = Z1 + Z2;
217     int l = q + m;
218     int N = l + k + 1;
219     double E;
220     if (l == 0) E = -(Z*Z / (2 * (double)N*N) + 2 * Z1*Z2*(l + 1) / (double)(N*N
        *N * (l + 1)*(2 * l - 1)*(2 * l + 1)*(2 * l + 3))*Z*Z*r*r);
221     else E = -(Z*Z / (2 * (double)N*N) + 2 * Z1*Z2*(l*(l + 1) - 3 * m*m) / (
        double)(N*N*N * l*(l + 1)*(2 * l - 1)*(2 * l + 1)*(2 * l + 3))*Z*Z*r*r);
222     double p = sqrt(-E*r*r/2.0);
223     return p;
224 }
225
226 //находим начальное lambda для p из теории возмущений(ZR = 0.1)
227 double find_begin_lambda()
228 {
229     double r = 0.1 / (Z1 + Z2);
230     double b = r * (Z2 - Z1);
231     double lambda = find_lambda_for_b_for_eta(b);//нашли lambda при p = 0
232     double p = p_from_perturbation_theory(r);
233     lambda = find_lambda_from_p_for_eta_near(p, b, lambda);
234     return lambda;
235 }
236
237 //решение системы уравнений alpha_k * c_k+1 - beta_k * c_k + gamma_k * c_k-1 = 0
238 vector<double> solve_system_equations(double alpha[], double beta[], double
       gamma[])
239 {
240     vector<double> koeff;
241     koeff.push_back(1);
242     koeff.push_back(beta[0] / alpha[0] * koeff[0]);
243     for (int i = 2; i <= N; i++)
244     {
245         koeff.push_back((beta[i - 1] * koeff[i - 1] - gamma[i - 1] * koeff[i -
            2]) / alpha[i - 1]);
246     }
247     return koeff;
248 }
249
250 void calculate_koeff_for_angular_function(double b, double p, double end_lambda,
       vector<double>& c1, vector<double>& c2)
251 {
252     //cout << "Koeff for angular function on segment (0,1)" << endl;
253     double * gamma = build_delta(b, p);
```

```
254     double * beta = build_chi(b, p, end_lambda);
255     double * alpha = build_rho(b, p);
256     c1 = solve_system_equations(alpha, beta, gamma);
257     double norm1 = 0; //нормирующий коэффициент, чтобы в нуле суммы
            коэффициентов совпадали
258     int size = c1.size();
259     for (int i = 0 ; i < size; ++i)
260     {
261         norm1 += c1[i];
262     }
263     delete [] alpha;
264     delete [] beta;
265     delete [] gamma;
266
267     //cout << "Koeff for angular function on segment (-1,0)" << endl;
268     double * delta = build_delta(b, p, -1);
269     double * chi = build_chi(b, p, end_lambda, -1);
270     double * rho = build_rho(b, p, -1);
271     c2 = solve_system_equations(rho, chi, delta);
272     double norm2 = 0;
273     size = c2.size();
274     for (int i = 0 ; i < size; ++i)
275     {
276         norm2 += c2[i];
277     }
278     // нормируем c'_i
279     norm1 /= norm2;
280     for (int i = 0 ; i < size; ++i)
281     {
282         c2[i] *= norm1;
283     }
284     delete [] rho;
285     delete [] chi;
286     delete [] delta;
287 }
288
289 void calculate_koeff_for_radial_function(double a, double p, double end_lambda,
        vector<double>& g)
290 {
291     double * alpha = build_alpha_for_lambda(a, p);
292     double * beta = build_beta(a, p, end_lambda);
293     double * gamma = build_gamma_for_lambda(a, p);
294     g = solve_system_equations(alpha, beta, gamma);
295     curtail_vector(g);
296     delete[] alpha;
297     delete[] beta;
298     delete[] gamma;
```

```
299 }
300
301 double build_Psi(params_for_Psi& par, double x, double z)
302 {
303     double p(par.p), a(par.a);
304     const vector <double> &c1(par.c1), &c2(par.c2), &g(par.g);
305     double R = par.R;
306     x = abs(x);
307     double xi, eta;
308     double sigma = (a / (2 * p)) - (m + 1);
309     double r1(sqrt(x*x + (R / 2 + z)*(R / 2 + z))), r2(sqrt(x*x + (-R / 2 + z)
            *(-R / 2 + z)));
310     xi = (r1 + r2) / R;
311     eta = (r1 - r2) / R;
312     if (abs(eta*R / r1) < 1E-15) eta = 0;
313     if (abs(eta) > 1) eta = (eta > 0 ? 1 : -1);
314     if (xi < 1) xi = 1; // связана с неточностью вычисления
315     const vector <double>& c = (eta > 0 ? c2 : c1);
316     return part_xi(p, xi, sigma, g) * part_eta(p, eta, c);
317 }
318
319 //переводит из сферических в x и z
320 double build_Psi_spher(params_for_Psi& par, double r, double theta, double
        vShift)
321 {
322     return build_Psi(par, r * sin(theta), r * cos(theta) + vShift);
323 }
324
325 //значение части волновой функции, зависящая от xi
326 double part_xi(double p, double xi, double sigma, const vector <double>& g)
327 {
328     double x = (xi-1)/(xi+1);
329     //double x = 2*p*(xi-1);
330     double sum = 0;//сумма ряда
331     for (int i = 0; i < /*10*/g.size(); ++i)
332     {
333         sum += g[i] * pow(x, i);
334     }
335     return pow(xi*xi - 1, m / 2.0) * exp(-p*(xi - 1)) * pow(xi+1, sigma) * sum;
336 }
337 //значение части волновой функции, зависящая от eta>0
338 double part_eta(double p, double eta, const vector <double>& c)
339 {
340     double sum = 0;//сумма ряда
341     eta = abs(eta); //для корректности следующих формул
342     for (int i = 0; i < c.size(); ++i) sum += c[i] * pow(1-eta, i);
343     return pow(1-eta*eta,m/2.0) * exp(-p*(1-eta)) * sum;
```

```
344 }
345
346 void find_energy_and_wave(double begin_p, koefficients& wave, double& Energy,
        double& Parameter, double& a)
347 {
348     double lambda_a, lambda_b, b, p;//a  понадобится вне
349     a = R * (Z1 + Z2);
350     b = R * (Z2 - Z1);
351     double begin_lambda = find_begin_lambda();
352     thread thra(find_lambda_for_a_for_xi, ref(lambda_a), begin_p, a,
            begin_lambda, 0.001);
353     lambda_b = find_lambda_for_b_for_eta(b, 0.0005);//здесь p=0
354     lambda_b = find_lambda_from_p_for_eta(begin_p, b, lambda_b - 0.0005);
355     thra.join();
356
357     double end_lambda = lambda_b;
358     p = dividing_section_for_p(begin_p, a, b, lambda_a, lambda_b, end_lambda);
359     double E = -2 * p*p / (R*R);
360     Energy = E;
361     Parameter = p;
362     //вычисляем коэффициенты c_i и c'_i для угловой части волновой функции
363     calculate_koeff_for_angular_function(b, p, end_lambda, wave.c1, wave.c2);
364
365     //вычисляем коэффициенты g_i для радиальной части волновой функции
366     calculate_koeff_for_radial_function(a, p, end_lambda, wave.g);
367 }
```

Приложение Б

Фрагменты кода для вычисления функций $C(R)$ и $H(R)$

Содержимое главного файла

```
1  #pragma once
2  #include "stdafx.h"
3  #include "functions.h"
4  using namespace std;
5  int Z1, Z2, k, q, m, N;
6  double R;
7
8  int main()
9  {
10     double  R_max, R_min, h;
11     bool inversion = false;
12     Z1 = Z2 = 1;
13     correctInput(R_min, []() {string str = "Input min R>"; return str +
           to_string(0.1 / abs(Z1 + Z2)); }(), [](wchar_t* input) {return ((wcstod(
           input, NULL) <= 0.1 / abs(Z1 + Z2))); });
14     correctInput(R_max, "Input max R>" + to_string(R_min),  [R_min](wchar_t*
           input) {return ((wcstod(input, NULL) <= R_min)); });
15     cout << "Input step dR= ";
16     cin >> h;
17
18     correctInput(N, "Input main quantium number N>0", [](wchar_t* input) {return
           (!(isPosInt(input)) || (_wtoi(input) <= 0)); });//главное кв число
19     correctInput(q, "Input orbital quantium number l>=0", [](wchar_t* input) {
           return (!(isPosInt(input)) || (_wtoi(input) < 0)); });//орбит кв число
20     k = N - 1 - q;
21     correctInput(m, []() {string str = "Input magnetic quantium number m<=";
           return str + to_string(q); }(), [](wchar_t* input) {return (!(isPosInt(
           input)) || (_wtoi(input) < 0) || (_wtoi(input) > q)); });
22     q = q - m;
23
24     N = 70;
25     double r = 0.1 / (Z1 + Z2);// для теории возмущений
26     double begin_p = p_from_perturbation_theory(r);
27     double max_r;
28     cout << "Take size of field for integration\n";
29     cout << "Type max_r= "; cin >> max_r;
30
31     int j = static_cast<int>((R_max - R_min) / h) + 1;// j = всего точек, дл
           якаждой из них будут вычисляться параметры для вычисления волновой
           функции
32     vector<koefficients> koeff(j);
```

```
33      vector<double> Energies(j);
34      vector<double> Parameters(j);
35      vector<double> A(j);
36      vector<double> Dist(j);
37      for (int i = 0; i < j; ++i)
38      {
39          Dist[i] = R_min + i * h;
40          R = Dist[i];
41          find_energy_and_wave(begin_p, koeff[i], Energies[i], Parameters[i], A[i
                ]);
42
43          //вычисляем интеграл от квадрата волновой функции
44          funct<square_struct<params_for_Psi>> func = squarer;
45          square_struct<params_for_Psi> params = { build_Psi_spher, params_for_Psi
                ({Dist[i], Parameters[i], A[i], koeff[i].c1, koeff[i].c2, koeff[i].g
                }) };
46          double integr = gsl_integral<square_struct<params_for_Psi>>(func, 0,
                max_r, 0, M_PI, params, 1000);
47          cout << "Integral in R = " << R << " equals " << integr << '\n';
48          //нормировка волновой функции на единицу
49          for_each(koeff[i].g.begin(), koeff[i].g.end(), [integr](double& g){g = g
                /sqrt(integr);});
50      }
51      ofstream fout;
52      fout.open("D:\\Диплом\\адиабатическое разложение\\Задача двух кулоновских
            центров\\результаты\\" + create_name(R_min, h, R_max) + ".csv", ios_base
            ::out);
53      if (!fout.is_open())
54          cout << "file isnt open" << endl;
55      else
56      {
57          fout << "R,E,C,H,L_2,sum\n";
58
59          for (int i = 2; i < j - 2; ++i)
60          {
61              //нужно взять интеграл от Psi * (оператор лапласа от)Psi по всему
                    двумерному пространству (=C1(R))
62              //он заменяется на интеграл от psi^2 *(-2*E - 2*Z1/r1 - 2*Z2/r2)
63              //берём интеграл интеграл от psi^2/r1. Переходим в полярные
                    координаты с центром в точке, где находится Z1, и получаем
64              //интеграл от psi^2 * theta dr dtheta.
65              R = R_min + i * h;
66              double integr1 = -2 * Energies[i];
67              funct<two_param_struct<square_struct<params_for_Psi>,
                    struct_for_Columb_pot>> func1 = multiplier;
68              two_param_struct<square_struct<params_for_Psi>,
                    struct_for_Columb_pot> params1 = { squarer, columb_pot,
```

```
                square_struct<params_for_Psi>({build_Psi_spher, params_for_Psi({
                    Dist[i], Parameters[i], A[i], koeff[i].c1, koeff[i].c2, koeff[i].
                    g })}), struct_for_Columb_pot({}) };
69              integr1 += -2 * Z2 * gsl_integral(func1, 0, max_r, 0, M_PI, params1,
                     100000, R / 2);
70              integr1 += -2 * Z1 * gsl_integral(func1, 0, max_r, 0, M_PI, params1,
                     100000, -R / 2);
71              integr1 = integr1 / 4;
72
73              //интеграл от второй производной
74              funct<two_param_struct<params_for_Psi, params_for_derivative_Psi>>
                    func2 = multiplier;
75              two_param_struct<params_for_Psi, params_for_derivative_Psi> params2
                    = { build_Psi_spher, my_second_derivative_Psi_3, params_for_Psi{
                    Dist[i], Parameters[i], A[i], koeff[i].c1, koeff[i].c2, koeff[i].
                    g } , params_for_derivative_Psi({ i, h, koeff, Parameters, A,
                    Dist }) };
76              double integr2 = gsl_integral(func2, 0, max_r, 0, M_PI, params2,
                    1000000, 0);
77              funct<two_param_struct<params_for_Psi, params_for_L_y>> func3 =
                    multiplier;
78              two_param_struct<params_for_Psi, params_for_L_y> params3 = {
                    build_Psi_spher, squareAngularMomentum_theta, params_for_Psi{
                    Dist[i], Parameters[i], A[i], koeff[i].c1, koeff[i].c2, koeff[i].
                    g } , params_for_L_y{ params_for_Psi{ Dist[i], Parameters[i], A[i
                    ], koeff[i].c1, koeff[i].c2, koeff[i].g }, 0.001 } };
79              double integr3 = gsl_integral(func3, 0, max_r, 0, M_PI, params3,
                    1000000, 0);
80              integr3 = -integr3 / (R * R);
81              integr2 += integr3;
82              cout << R << ',' << setprecision(10) << Energies[i] << ',' << -
                    integr1 << ',' << integr2 << integr2 - integr1 << setprecision(ss
                    ) << '\n';
83              fout << R << ',' << setprecision(10) << Energies[i] << ',' << -
                    integr1 << ',' << integr2 << integr2 - integr1 << setprecision(ss
                    ) << '\n';
84          }
85          fout.close();
86      }
87      system("pause");
88      return 0;
89  }
```

Основные фрагменты из файла «functions.h»

```
1  #pragma once
2  #include "gsl/gsl_integration.h"
3  #include "gsl/gsl_deriv.h"
```

```
4  extern int Z1, Z2, k, q, m, N;
5  extern double R;
6
7  using namespace std;
8  struct koefficients
9  {
10     koefficients() :c1(N), c2(N), g(N) {};
11     vector<double> c1, c2, g;
12 };
13 template<typename parameters>
14 using funct = double(*)(parameters&, double r, double z, double vShift);
15 //все параметры для Psi, кроме r и z
16 struct params_for_Psi
17 {
18     double R;
19     double p, a;
20     const vector <double> &c1, &c2, &g;
21 };
22
23 //структура для передачи параметров при интегрировании по r
24 template<typename parameters>
25 struct gsl_struct_first
26 {
27     funct<parameters> func;
28     parameters& par;
29     double z;
30     double vShift;
31 };
32
33 template<typename parameters1, typename parameters2>
34 struct two_param_struct
35 {
36     funct<parameters1> func1;
37     funct<parameters2> func2;
38     parameters1& par1;
39     parameters2& par2;
40 };
41
42 //интегрировании по theta
43 template<typename parameters>
44 struct gsl_struct_second
45 {
46     funct<parameters> func;
47     double min_r;
48     double max_r;
49     parameters& par;
50     int count;
```

```
51      double vShift;
52  };
53
54  //подынтегральная функция одномерного интеграла
55  template<typename parameters>
56  double gsl_integrand_first(double r, void* param)
57  {
58      gsl_struct_first<parameters>* par = static_cast<gsl_struct_first<parameters
            >*>(param);
59      double res = (par->func)(par->par, r, par->z, par->vShift)*r*r;
60      return res;
61  }
62
63  template<typename parameters>
64  struct square_struct
65  {
66      funct<parameters> func;
67      parameters& par;
68      double vShift;
69  };
70
71  //возводит в квадрат
72  template<typename parameters>
73  double squarer(parameters par, double r, double z, double vShift = 0)
74  {
75      double res = (par.func)(par.par, r, z, vShift);
76      return res * res;
77  }
78
79  //когда под интегралом умножение функций func1 и func2 с параметрами par1 и par2
80  template<typename parameters1, typename parameters2>
81  double multiplier(two_param_struct<parameters1, parameters2>& param, double r,
        double theta, double vShift = 0)
82  {
83      double res1 = (param.func1)(param.par1, r, theta, vShift);
84      double res2 = (param.func2)(param.par2, r, theta, vShift);
85      return res1 * res2;
86  }
87  //интеграл по отрезку от min_r до max_r
88  template<typename parameters>
89  double gsl_integral_r(funct<parameters> func, double min_r, double max_r,
        parameters& par, double theta, int count, double vShift = 0)
90  {
91      double res, err;
92      gsl_function F;
93      F.function = &gsl_integrand_first<parameters>;
94      gsl_struct_first<parameters> params = { func, par, theta, vShift };
```

```
95      F.params = &params;
96      gsl_integration_workspace* workspace = gsl_integration_workspace_alloc(count
            );
97      gsl_integration_qag(&F, min_r, max_r, 0.0001, .01, count, 1, workspace, &res
            , &err);
98      gsl_integration_workspace_free(workspace);
99      return res;
100 }
101
102 //функция, связывающая двумерный интеграл с одномерным
103 template<typename parameters>
104 double gsl_integral_theta(double theta, void* params)
105 {
106     gsl_struct_second<parameters>* par = static_cast<gsl_struct_second<
            parameters>*>(params);
107     return gsl_integral_r(par->func, par->min_r, par->max_r, par->par, theta,
            par->count, par->vShift)*sin(theta);
108 }
109
110 template<typename parameters>
111 double gsl_integral(funct<parameters> func, double min_r, double max_r, double
        min_theta, double max_theta, parameters& par, int count, double vShift = 0)//
        verticalShift - сдвиг по вертикали
112 {
113     double res, err;
114     gsl_function F;
115     F.function = &gsl_integral_theta<parameters>;
116     gsl_struct_second<parameters> params = { func, min_r, max_r, par, count,
            vShift };
117     F.params = &params;
118     gsl_integration_workspace* workspace = gsl_integration_workspace_alloc(count
            );
119     gsl_integration_qag(&F, min_theta, max_theta, 0.0001, .01, count, 1,
            workspace, &res, &err);
120     gsl_integration_workspace_free(workspace);
121     return res;
122 }
123
124 //все параметры для производной Psi по R, кроме r и z
125 struct params_for_derivative_Psi
126 {
127     int i; double h;
128     vector<koefficients>& koeff;
129     vector<double>& Parameters;
130     vector<double>& A;
131     vector<double>& Dist;
132 };
```

```
133
134 //для производной от любой функции
135 template<typename parameters>
136 struct params_for_derivative_5
137 {
138     funct<parameters> func;
139     parameters& par_minus_two;
140     parameters& par_minus_one;
141     parameters& par_zero;
142     parameters& par_plus_one;
143     parameters& par_plus_two;
144     double h;
145 };
146 template<typename parameters>
147 double my_derivative_5(params_for_derivative_5<parameters>& par, double r,
    double theta, double vShift = 0)
148 {
149     double minus_two = par.func(par.par_minus_two, r, theta, vShift);
150     double minus_one = par.func(par.par_minus_one, r, theta, vShift);
151     double zero = par.func(par.par_zero, r, theta, vShift);
152     double plus_one = par.func(par.par_plus_one, r, theta, vShift);
153     double plus_two = par.func(par.par_plus_two, r, theta, vShift);
154     return (minus_two - 8 * minus_one + 8 * plus_one - plus_two) / (12 * par.h);
155 }
156 struct struct_for_Columb_pot{};
157 double columb_pot(struct_for_Columb_pot&, double r, double theta, double vShift)
    ;
```

Основные фрагменты из файла «functions.cpp»

```
1 double my_derivative_Psi_5(params_for_derivative_Psi& par, double r, double
    theta, double vShift)
2 {
3     double minus_two = build_Psi_spher(params_for_Psi{ par.Dist[par.i - 2], (par
        .Parameters)[par.i - 2], par.A[par.i - 2], (par.koeff)[par.i - 2].c1, (
        par.koeff)[par.i - 2].c2, (par.koeff)[par.i - 2].g }, r, theta, vShift);
4     double minus_one = build_Psi_spher(params_for_Psi{ par.Dist[par.i - 1], (par
        .Parameters)[par.i - 1], par.A[par.i - 1], (par.koeff)[par.i - 1].c1, (
        par.koeff)[par.i - 1].c2, (par.koeff)[par.i - 1].g }, r, theta, vShift);
5     double plus_one = build_Psi_spher(params_for_Psi{ par.Dist[par.i + 1], (par.
        Parameters)[par.i + 1], par.A[par.i + 1], (par.koeff)[par.i + 1].c1, (par
        .koeff)[par.i + 1].c2, (par.koeff)[par.i + 1].g }, r, theta, vShift);
6     double plus_two = build_Psi_spher(params_for_Psi{ par.Dist[par.i + 2], (par.
        Parameters)[par.i + 2], par.A[par.i + 2], (par.koeff)[par.i + 2].c1, (par
        .koeff)[par.i + 2].c2, (par.koeff)[par.i + 2].g }, r, theta, vShift);
7     return (minus_two - 8 * minus_one + 8 * plus_one - plus_two) / (12 * par.h);
8 }
9 double my_second_derivative_Psi_3(params_for_derivative_Psi& par, double r,
```

```
    double theta, double vShift)
{
    double minus_one = build_Psi_spher(params_for_Psi{ par.Dist[par.i - 1], (par
        .Parameters)[par.i - 1], par.A[par.i - 1], (par.koeff)[par.i - 1].c1, (
        par.koeff)[par.i - 1].c2, (par.koeff)[par.i - 1].g }, r, theta, vShift);
    double zero = build_Psi_spher(params_for_Psi{ par.Dist[par.i], (par.
        Parameters)[par.i], par.A[par.i], (par.koeff)[par.i].c1, (par.koeff)[par.
        i].c2, (par.koeff)[par.i].g }, r, theta, vShift);
    double plus_one = build_Psi_spher(params_for_Psi{ par.Dist[par.i + 1], (par.
        Parameters)[par.i + 1], par.A[par.i + 1], (par.koeff)[par.i + 1].c1, (par
        .koeff)[par.i + 1].c2, (par.koeff)[par.i + 1].g }, r, theta, vShift);
    return (plus_one - 2*zero + minus_one) / (par.h * par.h);
}
double columb_pot(struct_for_Columb_pot&, double r, double theta, double vShift
    = 0)
{
    return 1 / r;
}
```