ДЛЯ УГЛУБЛЕННОГО ИЗУЧЕНИЯ

А.С. КОНДРАТЬЕВ
А.В. ЛЯПЦЕВ

ФИЗИКА

Задачи на компьютере

Допущено Учебно-методическим объединением по направлениям педагогического образования в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению 540200 (050200) Физико-математическое образование, и учащихся школ с углубленным изучением физики

МОСКВА
ФИЗМАТЛИТ®
2008

УДК 53(075.8)
ББК 22.3
К 64

Кондратьев А.С., Ляпцев А.В. **Физика. Задачи на компьютере.** — М.: ФИЗМАТЛИТ, 2008. — 400 с. — ISBN 978-5-9221-0917-8.

Пособие представляет собой сборник задач по физике, решение которых может быть выполнено с использованием персонального компьютера. Книга входит в учебно-методические комплект для школ с углубленным изучением физики (Е.И. Бутиков, А.С. Кондратьев, В.М. Уздин. Физика в 3-х т.; А.С. Кондратьев, В.М. Уздин. Сборник задач). В сборнике приведены задачи, от простых до достаточно сложных, решение которых представляет собой математическое моделирование рассматриваемых процессов или явлений. Приведен подробный разбор всех задач.

На отдельной дискете приведены программы к задачам, выполненные в среде MatLab. Эти программы помещены на сайте издательства.

Допущено УМО по направлениям педагогического образования в качестве учебного пособия для студентов вузов, обучающихся по направлению 540200 (050200) Физико-математическое образование, и учащихся школ с углубленным изучением физики.

ISBN 978-5-9221-0917-8

ОГЛАВЛЕНИЕ

ЭЛЕКТРОДИНАМИКА И ОПТИКА

СТРОЕНИЕ И СВОЙСТВА ВЕЩЕСТВА

ПРЕДИСЛОВИЕ

Сборник задач на компьютере представляет собой составную часть учебного пособия для школ и классов с углубленным изучением физики (авторы Е. И. Бутиков, А. С. Кондратьев, В. М. Уздин). Три книги, посвященные изложению теоретического материала, и сборник задач уже выходили в издательстве ФИЗМАТЛИТ (2000, 2001, 2004, 2005). Полезным дополнением к этому пособию является краткий физико-математический справочник (авторы А. Г. Аленицын, Е. И. Бутиков, А. С. Кондратьев). Данное пособие представляет собой сборник задач, решение которых может быть выполнено с использованием персонального компьютера. Значительная часть задач являются оригинальными. Спектр приведенных задач достаточно широк: от относительно простых, решение которых не представляет принципиальных трудностей, но для получения полного ответа требует проведения аккуратных численных расчетов, до весьма сложных задач, последовательное решение которых по существу представляет собой математическое моделирование рассматриваемых процессов или явлений, включая проведение вычислительного эксперимента. Все задачи снабжены подробными решениями, включая описание или указания относительно используемых программных средств.

Необходимость в таком учебном пособии для изучения физики в школах указанного типа обусловлена современным состоянием методологии физики, характеризующимся широким внедрением персонального компьютера как в научные исследования, так и в процесс обучения. Сбалансированность качественного и количественного подходов в современной физике очень точно характеризуется словами Э. Ферми, который подчеркивал, что «…настолько легко ошибиться, что не следует верить результату длинных и сложных математических выкладок, если нельзя понять его физического смысла; в то же время нельзя также полагаться на длинную и сложную цепь физических доводов, если нельзя продемонстрировать ее математически» (*Ферми Э.* Научные труды. — М.: Наука, 1971).

Сборник начинается двумя задачами, которые могут быть решены чисто аналитически, но могут быть проанализированы и численными методами. Это позволяет в полной мере реали-

зовать и продемонстрировать возможности взаимного контроля как результатов качественного анализа на основе численных расчетов, так и результатов численного расчета на основе аналитического решения. Все последующие задачи не могут быть решены «до числа» аналитически и требуют использования компьютера для получения исчерпывающего ответа. Большинство типов предлагаемых в пособии задач раньше не включались в аналогичные пособия, поскольку их решение требовало изучения специфических вычислительных методов. Развитие программного обеспечения, связанное с внедрением языков высокого уровня, привело в настоящее время к тому, что появились вычислительные среды, содержащие множество процедур, применение которых облегчает труд программиста, позволяя решать достаточно сложные задачи без глубокого освоения вычислительных методов. Ситуация здесь аналогична той, с которой мы сталкиваемся при проведении натурного эксперимента: часто использование очень сложных приборов не предполагает детального знания принципов их устройства. Решение задач проводится таким образом, чтобы добывать максимально возможную информацию об изучаемых явлениях на основе качественного исследования уравнений, позволяющую установить и понять физический смысл вопроса. Однако все задачи требуют проведения численных расчетов, так что значительная часть задач может рассматриваться как компьютерные лабораторные работы по математическому моделированию.

В настоящее время при переходе на профильное обучение в старшей школе предполагается введение в учебные планы раздела «Учебно-исследовательская деятельность». Многие задачи, которые рассматриваются в данном пособии могут служить основой для организации и проведения такого рода занятий с использованием компьютера. Большая часть предлагаемых задач рассчитана на достаточно длительный последовательный анализ рассматриваемых вопросов. К каждой из задач предлагаются дополнительные темы исследования, некоторые из которых являются даже более трудоемкими, чем сама задача.

Последовательность материала в пособии в целом соответствует последовательности изложения в трех книгах теоретического курса. Здесь следует учесть, что для большинства рассматриваемых колебательных явлений существуют их электромагнитные аналоги. В этом смысле включенный в пособие ма-

териал является сбалансированным по отношению к различным разделам физики и последовательности их изучения.

В сборник включены два приложения. В одном из них кратко излагаются основы качественного исследования дифференциальных уравнений, которое широко и продуктивно используется при решении многих задач, связанных с построением фазового портрета системы. Во втором приложении кратко изложены вопросы использования гармонического анализа при численном моделировании колебательных и волновых процессов.

МЕХАНИКА

1. ДИНАМИКА И ЗАКОНЫ СОХРАНЕНИЯ

1.1. Брызги от вращающегося колеса

Условие

С колеса буксующего автомобиля срываются комки грязи. Найдите максимальную дальность полета грязи и соответствующий угол, под которым направлена начальная скорость сорвавшегося с колеса комка грязи. Сделайте аналогичный расчет для высоты подъема комков грязи.

Решение

Рассмотрим вначале мокрое колесо, которое равномерно вращается в вертикальной плоскости вокруг неподвижной оси. С обода колеса срываются капли. Будем считать, что полет капель не ограничен крыльями автомобиля. Найдем границу «сухой» области, до которой не долетают капли, пренебрегая сопротивлением воздуха.

Сила тяжести сообщает всем каплям одинаковое ускорение $\mathbf{g}$. Поэтому отвлечемся сначала от наличия тяготения. При этом все оторвавшиеся от обода капли движутся с одинаковыми по модулю постоянными скоростями v_0 по прямым линиям. В любой момент времени t все капли находятся на окружности радиусом r, для которого с помощью теоремы Пифагора имеем (рис. 1.1.1)

$$r^2(t) = R^2 + (v_0 t)^2, \qquad (1.1.1)$$

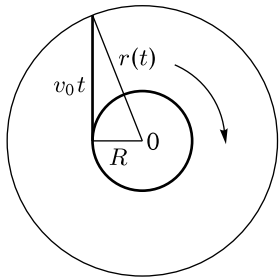

Рис. 1.1.1

где R — радиус колеса. Радиус окружности r увеличивается со временем, а при наличии тяготения окружность еще и падает с ускорением свободного падения. В любой момент времени ордината центра окружности равна $-gt^2/2$, если начало координат выбрано в центре колеса. Уравнение падающей окружности

имеет вид

$$x^2 + \left(y + \frac{gt^2}{2}\right)^2 = r^2(t). \tag{1.1.2}$$

В действительности (1.1.2) — это уравнение семейства окружностей: при разных значениях t получаем окружности, на которых капли находятся в этот момент времени. Искомая граница сухой области есть огибающая этого семейства окружностей. Высшая точка этой границы, очевидно, лежит точно над осью колеса. Для ответа на вопрос задачи нужно найти границу области, заштрихованной на рис. 1.1.2. Чтобы сделать это, заметим, что капли, оторвавшиеся от колеса в один и тот же момент времени, достигают границы в разные моменты, поскольку граница касается разных окружностей.

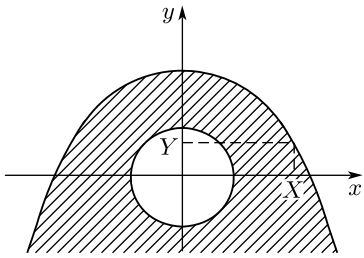

Рис. 1.1.2

Проведем горизонтальную прямую на некотором уровне Y и найдем на ней наиболее удаленную от оси y мокрую точку. Абсциссу X этой точки, т. е. точки пересечения прямой с границей, можно найти, подставив в уравнение окружности (1.1.2) ординату $y = Y$ и радиус r из уравнения (1.1.1):

$$X^2 = R^2 + v_0^2 t^2 - \left(Y + \frac{gt^2}{2}\right)^2. \tag{1.1.3}$$

Теперь остается найти максимальное возможное значение X. Поскольку (1.1.3) — это квадратный трехчлен относительно t^2:

$$X^2 = -\frac{g^2 t^4}{4} + (v_0^2 - gY)t^2 + R^2 - Y^2, \tag{1.1.4}$$

то его максимальное значение есть

$$X^2 = R^2 + \frac{v_0^4}{g^2} - \frac{2v_0^2}{g}Y. \tag{1.1.5}$$

Уравнение (1.1.5) — это и есть уравнение искомой границы. Переписав его в виде

$$Y = -\frac{g}{2v_0^2}X^2 + \frac{gR^2}{2v_0^2} + \frac{v_0^2}{2g}, \tag{1.1.6}$$

убеждаемся, что граница — это парабола, ветви которой направлены вниз, а вершина находится на оси y на высоте $gR^2/(2v_0^2) + v_0^2/(2g)$ над осью колеса.

В действительности траектории отдельных капель — это параболы, поэтому граница является огибающей этих парабол.

Данные, полученные при аналитическом решении, могут служить критерием правильности численных решений более сложных задач, для которых нет возможности получить аналитические решения. Пусть, например, необходимо смоделировать задачу, в которой исследуются полет комочков грязи от буксующего колеса заднеприводной машины. В этом случае полет частиц ограничен дополнительными условиями, а именно деталями конструкции машины (рис. 1.1.3). Частицы не должны пролетать выше точки с координатами (x_0, y_0).

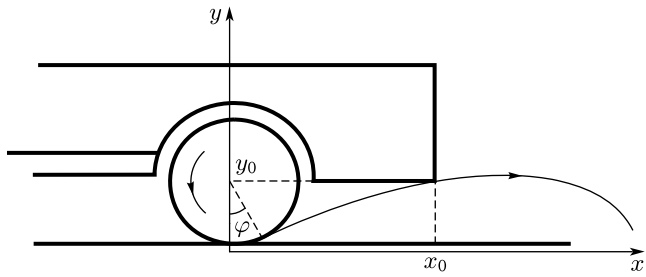

Рис. 1.1.3

Для решения задачи можно составить уравнение траектории движения частиц в координатах, изображенных на рис. 1.1.3:

$$y = y_1 + (x - x_1)\operatorname{tg}\varphi - \frac{g}{2(\omega R\cos\varphi)^2}(x - x_1)^2, \qquad (1.1.7)$$

где координаты точки отрыва частицы от колеса x_1 и y_1 определяются равенствами:

$$x_1 = R\sin\varphi, \quad y_1 = R(1 - \cos\varphi).$$

Дальность полета частиц при заданном угле φ и заданной угловой скорости ω равна координате x, при которой правая часть равенства (1.1.7) обращается в нуль. Задача численного расчета сводится, таким образом, к нахождению максимального значения дальности при учете только тех частиц, траектории которых не будут лежать выше точки с координатами (x_0, y_0). Проверка численного решения может быть проведена путем сравнения с аналитическим решением, полученным в отсутствие ограничений на движение частиц. Если задача запрограммирована на решение с произвольными значениями (x_0, y_0), то достаточно, например, сделать проверку, положив $x_0 = R$, $y_0 = 2R$.

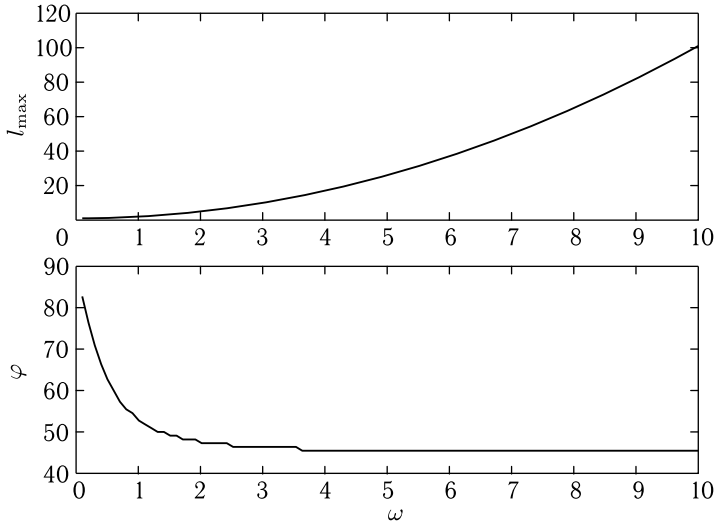

Рис. 1.1.4

Результаты расчета приведены на рис. 1.1.4 (отсутствие ограничения) и рис. 1.1.5 (значения $x_0 = 3R$, $y_0 = R$, что имеет разумный порядок значений для современных автомобилей). На верхних графиках показана зависимость дальности полета (в радиусах колеса) от угловой скорости (в единицах $\sqrt{gR}$). На нижних графиках — угол φ (в градусах), соответствующий углу

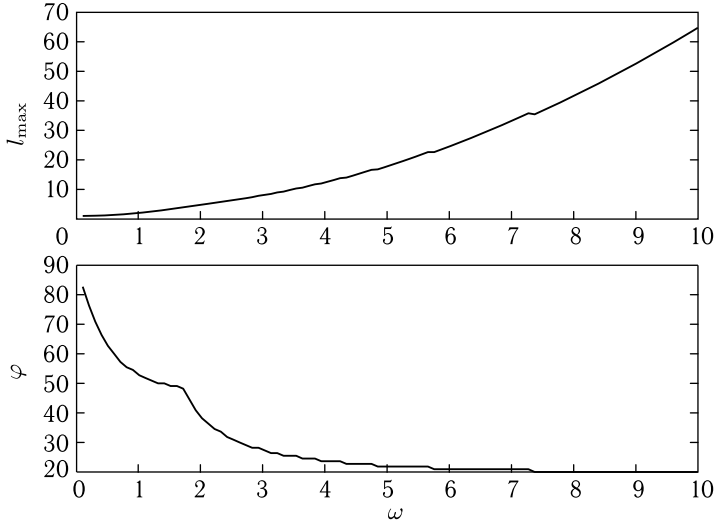

Рис. 1.1.5

отрыва частицы, улетающий на наибольшее расстояние. Как видно из графиков, ограничение конструкциями автомобиля снижает дальность полета грязи примерно в 1,5 раза.

Темы для дальнейших самостоятельных исследований

1. Исследуйте полет комочков грязи при различных формах крыла автомобиля.
2. Определите максимальную высоту подъема капель, не ограниченного крылом вращающегося колеса. В какой точке O колеса происходит отрыв комка, поднимающегося на максимальную высоту.
3. Найдите уравнение параболы, по которой летит комок грязи, если она касается границы сухой области в точке с координатами X, Y.

1.2. Фокусировка в однородном гравитационном поле

Условие

Рассмотрите движение нескольких частиц в однородном гравитационном поле, считая, что они имеют скорости, одинаковые по модулю и близкие по направлению. Покажите, что в некоторой области пространства эти частицы проходят практически через одну точку.

Решение

Объяснить существование такого, довольно неожиданного свойства можно с помощью простых рассуждений. Будем для определенности считать, что из ствола ружья, образующего угол α, с горизонтом вылетают дробинки. Отклонение направления начальной скорости каждой из них от направления оси ствола характеризуется углом β.

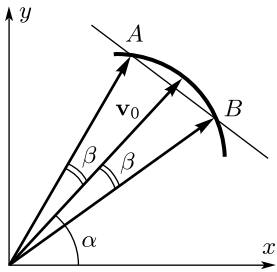

Рис. 1.2.1

Проследим мысленно за их полетом, предположив на минуту, что поле тяготения отсутствует. В этом случае дробинки полетят равномерно и прямолинейно, образуя веер (рис. 1.2.1).

В некоторый момент времени t все дробинки будут находиться на дуге AB окружности радиусом $v_0 t$ с центром в точке вылета O. Длина этой дуги, очевидно, равна $2v_0 t\beta$. При малых углах β дугу AB можно приближенно, с точностью до членов порядка β^2, заменить хордой AB.

В поле тяжести все дробинки движутся с одинаковым ускорением g, поэтому отрезок AB, на котором они находятся, перемещается параллельно самому себе с ускорением g, монотонно увеличиваясь в длине с течением времени. Совершенно очевидно, что фокусировка пучка окажется возможной, только если весь отрезок AB (т. е. все находящиеся на нем дробинки) пройдет через одну точку. Для этого в некоторый момент времени скорости всех дробинок должны оказаться направленными вдоль отрезка AB, тогда одна за другой все дробинки пройдут через одну и ту же точку-фокус. А поскольку на самом деле дробинки находятся не на хорде AB, а на дуге, то сразу становится ясно, что в действительности дробинки пройдут не через одну точку, а через некоторую малую область.

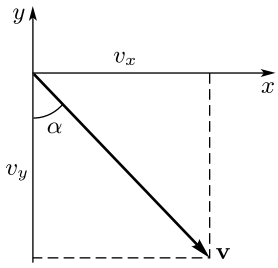

Рис. 1.2.2

Теперь нетрудно найти положение фокуса. Для этого, прежде всего, определим, в какой момент времени t_1 направление скорости дробинки, вылетевшей из ружья под углом α к горизонту, окажется параллельным отрезку AB, ориентация которого не меняется со временем. Из рис. 1.2.2 видно, что это произойдет при выполнении условия

$$v_y v_x = -\operatorname{ctg}\alpha. \tag{1.2.1}$$

Горизонтальная и вертикальная проекции скорости дробинки определяются выражениями

$$v_x = v_0\cos\alpha, \quad v_y = v_0\sin\alpha - gt.$$

Отсюда, используя (1.2.1), находим $t_1 = v_0/(g\sin\alpha)$.

Подставив полученное значение t_1 в уравнения движения дробинки в поле тяжести,

$$x = v_0\cos\alpha t, \quad y = v_0\sin\alpha t - \frac{gt^2}{2},$$

определим координаты x_1 и y_1 фокуса:

$$x_1 = \frac{v_0^2\operatorname{ctg}\alpha}{g}, \quad y_1 = \frac{v_0^2}{2g}(1 - \operatorname{ctg}^2\alpha). \tag{1.2.2}$$

Еще один полезный вывод можно сделать, рассматривая обратную задачу нахождения начальной скорости дробинок $\mathbf{v}_0$ (т. е. значений v_0 и α) по заданному положению фокуса (т. е. по его координатам x_1 и y_1). С этой целью с помощью (1.2.2) найдем

уравнение для $\operatorname{tg}\alpha$:

$$v_0^2 = gx_1 \operatorname{tg}\alpha, \quad \operatorname{tg}^2\alpha - 2\frac{y_1}{x_1}\operatorname{tg}\alpha - 1 = 0.$$

Отсюда видно, что угол α зависит не от координат фокуса x_1 и y_1, а только от их отношения y_1/x_1.

Приведенные рассуждения показывают, что эффект фокусировки пучка дробинок фактически определяется не условием пересечения различных траекторий между собой, а условием пересечения траекторий с линией MN (рис. 1.2.3), перпендикулярной к центральной траектории пучка. Точки пересечения для траекторий, отличающихся от центральной начальным углом на величину β, отстоят на прямой MN друг от друга на расстояние порядка β^2.

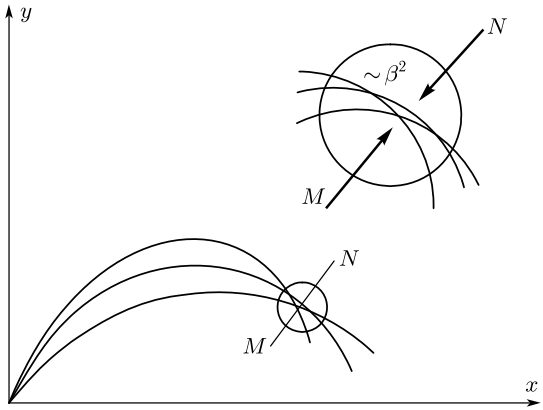

Рис. 1.2.3

Вопрос о координатах фокуса тесно связан с вопросом о границе достижимых целей при выстреле из данного ружья. Уравнение траектории дробинки получается при исключении времени из уравнений движения в гравитационном поле:

$$y = x\operatorname{tg}\alpha - \frac{gx^2}{2v_0^2}(1 + \operatorname{tg}^2\alpha). \tag{1.2.3}$$

Рассмотрим теперь цели, находящиеся на одной вертикали, отстоящей от ружья на расстоянии x, и найдем на ней самую высокую точку, в которую еще может попасть дробинка. Эта точка, очевидно, будет принадлежать искомой границе. Таким образом, задача сводится к нахождению максимума y, т. е. правой части (1.2.3), рассматриваемой как функция угла α, под которым дробинки вылетают из ствола. Анализируемая правая часть

представляет собой квадратный трехчлен относительно $\operatorname{tg}\alpha$ и имеет максимум при

$$\operatorname{tg}\alpha = \frac{v_0^2}{gx}. \tag{1.2.4}$$

Соответствующее максимуму значение y получается подстановкой (1.2.4) в (1.2.3):

$$y = \frac{v_0^2}{2g} - \frac{gx^2}{2v_0^2}. \tag{1.2.5}$$

Это и есть уравнение границы достижимых целей. Видно, что (1.2.5) — уравнение параболы с вершиной при $x = 0$ и $y = v_0^2/(2g)$. Ветви параболы направлены вниз и пересекают горизонтальную ось в точках $x = \pm v_0^2/g$. Найденная парабола обычно называется «параболой безопасности».

Укажем еще на один способ получения уравнения (1.2.5). Именно такого вида уравнение получится, если исключить $\operatorname{ctg}\alpha$ из выражений (1.2.2), определяющих координаты фокуса. Таким образом, фокусы, т. е. точки пересечения траекторий, соответствующих близким начальным углам вылета дробинок, лежат на границе достижимых целей.

Программа в среде Matlab

> Есть шестьдесят девять способов сочинять песни племен, и каждый из них правильный.
>
> *Р. Киплинг*

Занимаясь программированием, полезно помнить приведенное выше высказывание Киплинга. Почему мы выбрали среду Matlab? У нее много достоинств, которые мы попытаемся далее продемонстрировать. Однако, если кто-то хорошо освоил программирование в другой среде, то посмотри снова на эпиграф. Почему в приведенных ниже программах мы поступаем именно так, а не иначе? Есть определенные правила программирования, которые надо выполнять, но если ваша программа не противоречит правилам и хорошо работает, то для целей, которые ставятся в нашем учебнике, этого вполне достаточно, даже если программа не совершенна. Скорость работы программы, конечно, зависит от того, как она сконструирована. Однако, для подавляющего числа задач, предлагаемых в данном пособии, составленные программы, на современных персональных компьютерах выполняются за доли секунды, или, максимум, за десятки секунд, поэтому требования по скорости работы программы не существенны.

Прежде всего, поставим задачу на языке математики. Наша задача — визуализировать результаты приведенных выше расче-

тов. Для этого необходимо построить серию графиков функции

$$y(x,\alpha) = x\,\mathrm{tg}\,\alpha - \frac{gx^2}{2v_0^2}(1+\mathrm{tg}^2\,\alpha). \qquad (1.2.6)$$

По горизонтальной оси следует отложить значение x, по вертикальной — значение y и сделать это для нескольких близких значений α.

С точки зрения математики функция y — это непрерывная функция переменных x и α. Компьютер работает с дискретными величинами. Поэтому вместо непрерывной функции образуется массив:

$$y_{ij} = y(x_i, \alpha_j). \qquad (1.2.7)$$

С точки зрения программирования — это двумерный массив. В большинстве языков программирования не принципиально, в каком порядке располагать индексы, нужно только помнить и не менять этот порядок. В среде Matlab каждый двумерный массив — это матрица и порядок расположения индексов важен. Пусть, например, индекс i в выражении (1.2.7) пробегает 10 значений (10 значений переменной x), а индекс j — пять значений. Тогда матрица y_{ij} — таблица, у которой 10 строк и 5 столбцов. Если же сконструировать другую матрицу,

$$Y_{ji} = y(x_i, \alpha_j),$$

то таблица будет иметь 5 строк и 10 столбцов. Важность понимания того, с какой матрицей вы имеете дело, обусловлена тем, что в среде Matlab к матрицам можно обращаться, не выписывая индексов, и автоматически выводить на экран данные в виде таблицы.

Итак, математически задача сводится к следующему: необходимо задать массивы x_i и α_j, по формуле (1.2.6) вычислить массив y_{ij} и вывести соответствующие точки на график.

Теперь самые необходимые сведения о среде Matlab. Подробные сведения и способы программирования можно найти в книгах [1, 2]. Среда представляет собой совокупность нескольких окон, которые можно располагать по-разному, в том числе и закрывать некоторые из них. Главное командное окно присутствует всегда, хотя бы в свернутом виде. В этом окне можно производить различные математические действия и даже писать простейшие программы. Один из простейших примеров приводится ниже (далее все, что касается программ, мы будем располагать в таблицах, верхняя строка которых служит для нумерации, а нижняя соответствует тому, что имеется в окнах программы).

Листинг 1

```
>> A=rand(3,5)

A =

  0.9501  0.4860  0.4565  0.4447  0.9218
  0.2311  0.8913  0.0185  0.6154  0.7382
  0.6068  0.7621  0.8214  0.7919  0.1763

>> size(A)

ans =

   3  5

>> A(2,3)

ans =

  0.0185

>>
```

Здесь символы >> есть приглашение к вводу, и все, что идет на экране после этих символов, — есть набранное с клавиатуры. В данном случае первым действием мы организовали построение матрицы размером в 3 строки и 5 столбцов из случайных чисел в диапазоне от 0 до 1, которую среда тут же вывела на экран. Естественно, что, если вы повторите на своем компьютере действия, то получите другие числа. Вторым действием мы определили размер матрицы. Этот способ бывает очень полезен при отладке программ. Третьим действием вы вывели матричный элемент, стоящий во второй строке, в третьем столбце. В конце снова появляется приглашение к вводу (далее мы, приводя текст из командного окна, будем эти символы для краткости опускать).

Заметим, что, если бы мы немного изменили первую команду, то на экране получили бы другое:

```
>> A=rand(3,5);
>> size(A)

ans =

   3  5

>> A(2,3)

ans =

 0.8132
```

Разделителем — точкой с запятой мы подавили вывод матрицы на экран, однако в памяти среды матрица имеется, и второе действие вывело ее размеры. Третье действие, как и прежде, вывело матричный элемент, стоящий во второй строке и в третьем столбце, однако его значение уже другое, поскольку каждый раз, вызывая команду rand, мы получаем новые случайные числа.

В командном окне можно делать несложные вычисления, но для выполнения полноценных программ необходимо создать файл с программой, в Matlab они называются m-файлы и имеют расширение m. Эти файлы создаются в окне редактора файлов. Если такое окно закрыто, то оно открывается при нажатии на значок «открыть файл» или значок «создать файл». Язык программирования в среде Matlab похож на Бейсик или Паскаль, но это язык не просто высокого уровня, а сверхвысокого уровня. Смысл этого высказывания будет понятен далее.

Все m-файлы в Matlab подразделяются на так называемые скрипт-файлы и файлы-функции. Между ними есть три существенных различия (несущественные мы опускаем).

1. После окончания работы скрипт-файлы оставляют в памяти среды значения всех переменных, которые были использованы в процессе работы. Эти значения могут быть вызваны из среды и после окончания работы программы, записанной в скрипт-файле. При запуске программы из файла-функции после окончания работы программы из памяти среды удаляются все переменные, которые использовались программой. Это свойство скрипт-файлов является преимуществом при отладке программ и недостатком, когда вы работаете со многими программами.

2. В файлах обоих типов можно обращаться к подпрограммам, написанным пользователем. Однако если обращение идет из программы, написанной в виде скрипт-файла, то подпрограмма пользователя должна быть оформлена в виде отдельного файла-функции. В программе, написанной в виде файла-функции, подпрограммы пользователя могут находиться в исходном файле, как составляющие. Это свойство не существенно для начинающего программиста, но, если вы пишете много программ, то, создавать множество файлов-подпрограмм нецелесообразно, поэтому следует использовать файлы-функции.

3. Файл-функция может вызываться любой другой программой, причем при обращении к нему можно передать значения необходимых переменных. Скрипт-файл можно вызывать только из командного окна.

Текст файла-функции отличается от текста скрипт-файла тем, что он начинается словом function, за которым следует имя файла-функции.

Мы покажем вначале, как создать программу в виде скрипт-файла, а затем переделаем ее в файл-функцию.

Любой программный файл целесообразно начать с комментария. В Matlab комментарий идет после символа % до конца строки. Комментарий, как и фрагменты текста, при выводе на экран можно писать как латинскими, так и русскими буквами, если Вы пользуетесь русифицированной версией. Однако, при переносе программ с одного компьютера на другой Вы можете столкнуться с проблемами, которые в лучшем случае сведутся к нечитаемости русского текста. В худшем варианте компьютер может зависнуть. Поэтому мы рекомендуем печатать комментарии и текстовый вывод делать латинскими буквами.

После комментария в скрипт-файл целесообразно вставить оператор clear, очищающий память среды. В противном случае при отладке программы Вы можете столкнуться с тем, что имеющиеся в памяти переменные могут Вам «помешать».

Следующий этап — ввод данных, который может осуществляться операторами присваивания, либо операторами ввода с клавиатуры в процессе работы программы. Например, введем число элементов массива α — na, число элементов массива x — nx, угол β, характеризующий угол разброса начальных скоростей — b, а угол α_0, характеризующий среднее значение угла α будем вводить с клавиатуры (как и в большинстве языков программирования, для идентификаторов можно использовать только латинские буквы, цифры и некоторые другие символы). Листинг программы будет иметь следующий вид.

Листинг 2

```
% focusing of trajectories
clear;
nx=10;
na=5;
b=0.1;
a0=input('alfa=');
```

Сохраним эту программу под именем focus. Теперь можно вызвать эту программу в командном окне, просто набрав ее имя и нажав клавишу Enter. На экране появится

```
>> focus
alfa=
```

это означает приглашение к вводу. Далее нужно набрать число и снова нажать Enter, после чего программа продолжит работу.

В программе нам потребуется вычислять тригонометрические функции. Значения аргументов среда воспринимает в радианах. Ввод часто удобно делать в градусах. Поэтому добавим команду пересчета: a0=a0*pi/180. Идентификатор pi зарезервирован для числа π, хотя Вы можете использовать его и для своей переменной.

Следующий этап — организация массивов α_j и x_i. Пусть элементы массива будут разделены равными промежутками значений. Тогда первый элемент массива α_j будет иметь значение $\alpha_0 - \beta/2$, шаг массива $d\alpha = b/(na-1)$, а конечный элемент массива $\alpha_0 + \beta/2$. В Matlab массив с известными начальным и конечными значениями и шагом задается простой командой, в данном случае: a=a0−b/2:da:a0+b/2. Вставим соответствующие операторы в программу.

Листинг 3

```
% focusing of trajectories
clear;
nx=10;
na=5;
b=0.1;
a0=input('alfa=');
a0=a0*pi/180;
da=b/(na-1);
a=a0-b/2:da:a0+b/2
```

Далее вызовем программу в командном окне, и после приглашения ввести alfa, введем 45. В командном окне получим следующее:

Листинг 4

```
>> focus
alfa=45

a =

   0.7354   0.7604   0.7854   0.8104   0.8354
```

Поскольку после оператора, определяющего массив a, не был поставлен разделитель, этот массив автоматически вывелся на экран. Такие действия полезны при отладке программ, но далее разделитель можно поставить. Значения массива a, естественно, получились в радианах. Массив вывелся в виде строки, именно в таком виде он вводится последней командой и находится в памяти.

Теперь следует ввести массив x_i. Начальное значение массива, очевидно, нуль. Конечное значение массива различное для траекторий с разными значениями углов. Поскольку для наглядного изображения эффекта фокусировки графики для различных

углов удобно совместить, то в качестве конечного значения целесообразно выбрать максимально возможное значение x. Это значение соответствует углу 45° и равно v_0^2/g. Значения v_0 и g введем с помощью оператора присваивания. Для g возьмем значение 10, а для v_0 — значение 1 (при необходимости его легко изменить). Операция возведения в степень в Matlab выполняется с помощью символа ^. Для отладки не будем подавлять вывод массива x на экран. Мы получим программу:

Листинг 5

```
% focusing of trajectories
clear;
nx=10;
na=5;
b=0.1;
a0=input('alfa=');
a0=a0*pi/180;
da=b/(na-1);
a=a0-b/2:da:a0+b/2;
g=10;
v0=1;
xmax=v0^2/g;
dx=xmax/nx;
x=0:dx:xmax
```

Вызвав ее в командном окне и набрав в качестве alfa число 45, получим результат.

Листинг 6

```
>> focus
alfa=45

x =

  0  0.0100  0.0200  0.0300  0.0400  0.0500 \
     0.0600  0.0700  0.0800  0.0900  0.1000
```

Таким образом вы ввели оба массива, не обращаясь к оператору цикла. Теперь нужно запрограммировать вычисление массива y. Если Вы знакомы с языками программирования, но не знакомы с Matlab, проще всего сделать при помощи операторов цикла. Если цикл выполняется известное число раз, то проще всего для него использовать оператор for, после которого идет тело цикла. Тело цикла представляет собой последовательность любых операторов. Закрывается цикл всегда оператором end. В данном случае следует организовать два цикла, один из которых вложен в другой. Мы получим следующую программу:

Листинг 7

```
% focusing of trajectories
clear;
nx=10;
na=5;
b=0.1;
a0=input('alfa=');
a0=a0*pi/180;
da=b/(na-1);
a=a0-b/2:da:a0+b/2;
g=10;
v0=1;
xmax=v0^2/g;
dx=xmax/(nx-1);
x=0:dx:xmax;
for i=1:na
  for j=1:nx
    y(i,j)=x(j)*tan(a(i))-g*(x(j)/v0)^2/2*(1+(tan(a(i))^2));
  end
end
y=y
```

Дадим некоторый комментарий. Табуляцию редактор программ делает автоматически при вводе. Это делает программу легко читаемой и позволяет избежать ошибок. Кроме того, редактор выделяет цветом операторы цикла и условные операторы, т. е. те, которые необходимо закрывать оператором end. Другим цветом выделяются комментарии. Приоритет операций такой же, как в большинстве языков программирования. Последняя строка вставлена для отладки, чтобы вывести массив y. Заметим, что при попытке вывести массив, убрав разделитель в строке, соответствующей телу цикла, этот массив выводился бы $5 \times 10 = 50$ раз, т. е. каждый раз при дополнении массива.

Вызовем теперь эту программу из командного окна с введением alfa=45.

Листинг 8

```
y =

    0   0.0089   0.0156   0.0201   0.0222   0.0222 \
        0.0199   0.0154   0.0086  -0.0005
    0   0.0094   0.0164   0.0211   0.0235   0.0234 \
        0.0211   0.0164   0.0093  -0.0001
    0   0.0099   0.0173   0.0222   0.0247   0.0247 \
        0.0222   0.0173   0.0099  -0.0000
    0   0.0104   0.0182   0.0233   0.0259   0.0259 \
        0.0233   0.0181   0.0103  -0.0001
```

```
    0   0.0109   0.0191   0.0245   0.0272   0.0271 \
        0.0243   0.0188   0.0105  -0.0006
```

Анализ выведенной матрицы показывает, что мы получили разумный результат. Последнее значение в средней строке, соответствующей углу 45° равно нулю. Для других значений оно немного меньше нуля.

Теперь строку с выводом массива y можно стереть, или (вдруг еще потребуется) подавить, поставив перед ней символ комментария. Далее нужно вывести график. Вот здесь проявляются существенные преимущества среды Matlab. Вместо того, чтобы организовывать два цикла по индексам массива y, достаточно набрать простую команду: plot(x,y). Первый аргумент здесь тот, что будет отложен по горизонтальной оси, второй — тот, что будет отложен по вертикальной оси. Поскольку y — на самом деле двумерный массив, среда автоматически выводит несколько графиков, раскрашивая их разными цветами. Для значения alfa=60 получим график, приведенный на рис. 1.2.4.

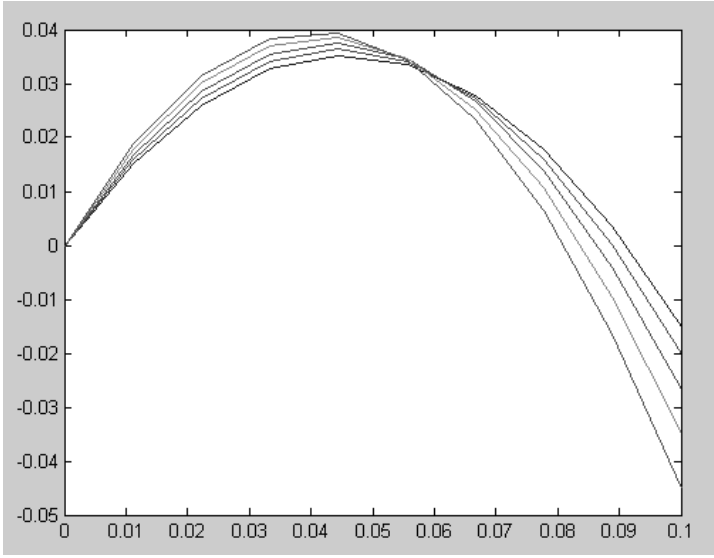

Рис. 1.2.4

Некоторые комментарии. По умолчанию среда Matlab соединила точки прямыми линиями, так что получились ломаные линии. Существуют способы подавить эти линии и выводить точки. Целесообразно, однако, сделать линии гладкими, для чего достаточно увеличить число точек по оси x с 10 до 100. Грани-

цы графика по вертикали среда также вывела по умолчанию. Естественно, мы можем в программе их изменить. Поскольку нам не нужен график при $y < 0$, хотелось бы сделать нижнюю границу графика по вертикали нулем. Для этого необходимо изменить параметры, которые среда создает по умолчанию при построении графика. Вызвать эти параметры можно, например, командой: ax=axis, поставив ее в конец программы. Вызвав программу и введя то же самое значение для alfa, Вы, помимо окна с графиком, получите в командном окне

Листинг 9

```
> focus
alfa=60

ax =

     0   0.1000   -0.0500   0.0400
```

Мы создали переменную ax, в которую занесли массив, определяющий границы графика. Массив представляет вектор-строку из 4 элементов. Первые два элемента — это границы по горизонтали, два вторые элемента границы по вертикали. Нам нужно изменить третий элемент массива, что можно сделать командой: ax(3)=0. Теперь нужно сделать этот массив определяющим границы графика, что производится командой: axis(ax). Новая программа имеет следующий вид.

Листинг 9

```
% focusing of trajectories
clear;
nx=100;
na=5;
b=0.1;
a0=input('alfa=');
a0=a0*pi/180;da=b/(na-1);
a=a0-b/2:da:a0+b/2;
g=10;
v0=1;xmax=v0^2/g;
dx=xmax/(nx-1);
x=0:dx:xmax;
for i=1:na
  for j=1:nx
    y(i,j)=x(j)*tan(a(i))-g*(x(j)/v0)^2/2*(1+(tan(a(i))^2));
  end
end
plot(x,y)
ax=axis;
```

```
ax(3)=0;
axis(ax);
```

Вызовем ее снова в командном окне и введем alfa=60. Мы получим следующий график (рис. 1.2.5).

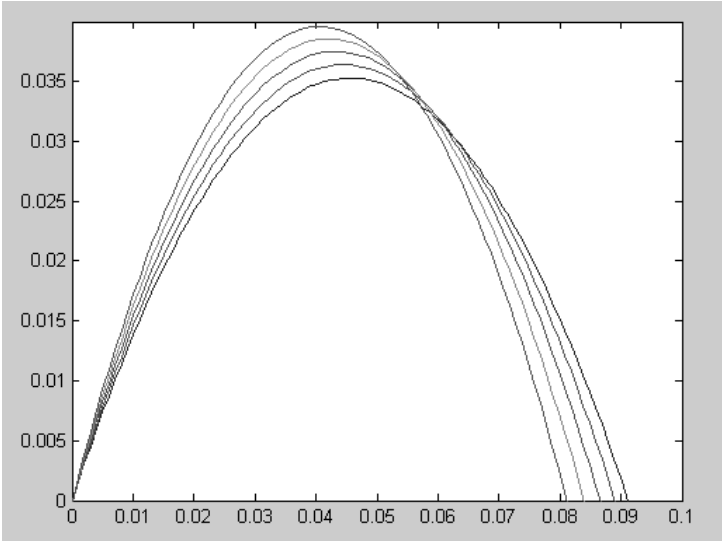

Рис. 1.2.5

На первый взгляд график получился вполне приличным, но при внимательном изучении видно, что масштабы по осям различаются. Изменяя курсором размеры окна, где выведен график, можно добиться равенства масштабов. Но лучше запрограммировать вывод так, чтобы масштабы сразу же были равными и не изменялись при изменении параметров окна. Для этого достаточно вставить сразу же после команды plot команду: axis equal.

Неплохо также вывести надписи того, что откладывается по осям. Это можно сделать командами: xlabel('x'); ylabel('y'). Между апострофами может быть любой текст, даже на русском языке, но при использовании русского текста могут возникнуть проблемы, о которых мы уже говорили.

Теперь мы получим следующий график (рис. 1.2.6). Даже при изменении размеров окна курсором равенство масштабов по осям сохраняется.

Программа готова и работает. Однако с точки зрения программирования в Matlab она не совершенна. По сравнению с программой, написанной, например, на языке Pascal, она содержит

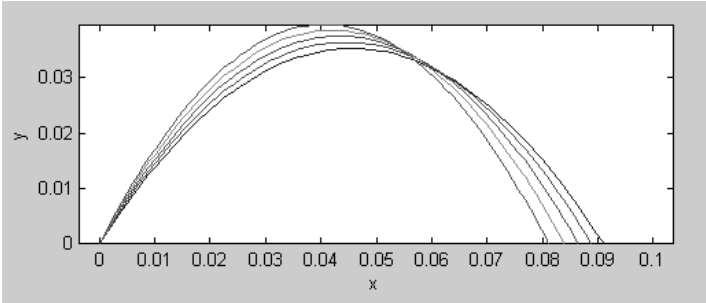

Рис. 1.2.6

на два цикла меньше (ввод элементов массивов). Может быть, попытаться убрать и другие циклы.

Уберем, вначале цикл по индексам переменной x. Для этого вместо строчек

```
for j=1:nx
  y(i,j)=x(j)*tan(a(i))-g*(x(j)/v0)^2/2*(1+(tan(a(i))^2));
end
```

вставим строку

```
y(i,:)=x*tan(a(i))-g*(x/v0).^2/2*(1+(tan(a(i))^2));
```

Поясним, что дает эта строка. Идентификатор y(i,:) определяет массив, а именно вектор-строку, соответствующий i-й строке массива y. Таким образом, присваивание значений происходит сразу же целой строке. В правой части мы по-прежнему используем индексы для массива a, однако массив x задаем целиком. Первое слагаемое, x*tan(a(i)), умножает вектор-строку на скаляр. Здесь достаточно опустить индекс j у массива x. Второе слагаемое сложнее, нам нужно вначале создать массив, элементами которого являются $(x(j))^2$. Это достигается тем же оператором возведения в степень, но с добавлением точки перед символом ^. Если эту точку опустить, то компилятор выдаст сообщение об ошибке, поскольку Вы запрограммируете перемножение двух матриц, каждая из которых является вектором-строкой. Это некорректная операция. Убедитесь, что новая программа успешно работает и дает тот же результат.

Если Вы попробуете аналогичным образом избавиться от цикла по индексам массива a, то компилятор выдаст сообщение об ошибке. Вы опять неправильно перемножаете матрицы, делая ту же ошибку, о которой говорилось выше. Чтобы понять, что нужно сделать, поработаем прямо в командном окне. Введем две матрицы A и B, представляющие векторы-строки, и попытаемся их перемножить.

Листинг 10

```
>> A=[1 2]

A =

  1  2

>> B=[3 4 5]

B =

  3  4  5

>> A*B

??? Error using ==> *
Inner matrix dimensions must agree.

>> B*A
??? Error using ==> *
Inner matrix dimensions must agree.
```

Как видите, при любой последовательности матриц в произведении компилятор выдает сообщение об ошибке. Введем теперь вместо вектора-строки B вектор-столбец Bt с теми же значениями. Для этого достаточно значения элементов разделять не пробелами, а точкой с запятой. Вот что мы получим.

Листинг 11

```
> Bt=[3;4;5]

Bt =

   3
   4
   5

>> A*Bt
??? Error using ==> *
Inner matrix dimensions must agree.

>> Bt*A

ans =

   3  6
   4  8
   5  10
```

Если число элементов в векторе-строке и векторе-столбце различно, то перемножить вектор-строку на вектор-столбец мы не можем. А вот перемножение вектора-столбца на вектор-строку вполне возможно. При этом получается матрицы с числом строк

таким же, как число элементов в векторе-столбце и с числом столбцов, таким же, как число элементов в векторе-строке. Каждый элемент матрицы, являющейся произведением, как несложно убедиться, является произведением соответствующих элементов вектора-столбца на вектор строку.

Именно это нам и нужно для программы. Из вектора-строки a следует создать вектор-столбец с теми же элементами. Это делается просто помещением символа апострофа после соответствующего идентификатора. На языке матриц такая операция называется транспонированием. Таким образом, цикл

```
for i=1:na
 y(i,:)=x*tan(a(i))-g*(x/v0).^2/2*(1+(tan(a(i))^2));
end
```

мы заменили одной строкой

```
y=tan(a')*x-g*(1+(tan(a').^2))*(x/v0).^2/2;
```

Конечно же, пришлось поменять порядок сомножителей и не забыть про точку при возведении тангенса в квадрат. «Умный» компилятор даже сделал из скаляра 1 массив из единиц с размерностью той же что у вектора a', перед тем как выполнить операцию сложения. Новая программа, листинг которой мы приводим, содержит всего 21 строку, не содержит циклов и прекрасно работает.

Листинг 12

```
1  % focusing of trajectories
2  clear;
3  nx=100;
4  na=5;
5  b=0.1;
6  a0=input('alfa=');
7  a0=a0*pi/180;
8  da=b/(na-1);
9  a=a0-b/2:da:a0+b/2;
10 g=10;
11 v0=1;
12 xmax=v0^2/g;
13 dx=xmax/(nx-1);
14 x=0:dx:xmax;
15 y=tan(a')*x-g*(1+(tan(a').^2))*(x/v0).^2/2;
16 plot(x,y) axis equal
17 ax=axis;
18 ax(3)=0;
19 axis(ax);
20 xlabel('x');
21 ylabel('y');
```

Чтобы понять, что такое язык сверхвысокого уровня, достаточно сравнить эту программу с аналогичной, написанной на языке Basic [3], не помещающейся на одной странице книги.

Теперь можно заняться некоторым дополнением. Вычислим точку, в которой теория предсказывает фокусировку, и вокруг этой точки нарисуем окружность. Положим радиус окружности равным 1/10 от максимальной высоты, которой достигают траектории, изображенные на графике. Для этого нам нужно добавить несколько строк. Выпишем эти строки отдельно, не приводя весь листинг программы:

```
hold on x1=v0^2*cot(a0)/g y1=v0^2/2/g*(1-(cot(a0))^2)
t=0:pi/30:2*pi;
ymax=max(max(y)) r=ymax/10;
x2=x1+r*cos(t);
y2=y1+r*sin(t);
plot(x2,y2)
```

Первая строка нужна для того, чтобы окружность выводилась на прежний график, и не стирала предыдущий. Две следующие строки вычисляют точку фокусировки по формулам (1.2.2). Отсутствие разделителя (точки с запятой) в конце строк означает, что эти значения будут автоматически выводится на экран в процессе работы программы. Следующая строка определяет массив вспомогательной переменной t для того, чтобы параметрически задать окружность. Пятая строка определяет максимум, достигаемый траекториями на графике. Заметим, что y — матрица, т. е. с точки зрения языков программирования двумерный массив. Для нахождения максимума в обычном языке программирования необходим двойной цикл с использованием условных операторов. Здесь одна команда max формирует строку, каждый элемент которой содержит максимальный элемент из соответствующего столбца. Применение команды max вторично дает максимальный элемент в этой строке. Далее мы определяем радиус окружности, которую мы хотим построить, и задаем ее параметрически. После чего просто добавляем вновь уже знакомую команду plot. Результат при $\alpha = 60^\circ$ приведен на рис. 1.2.7.

Написанная программа представляет собой скрипт-файл. Как уже говорилось, после окончания работы программы в памяти среды остаются значения переменных, что можно проверить, вызвав массив a.

Листинг 13

```
>> focus
alfa=60
```

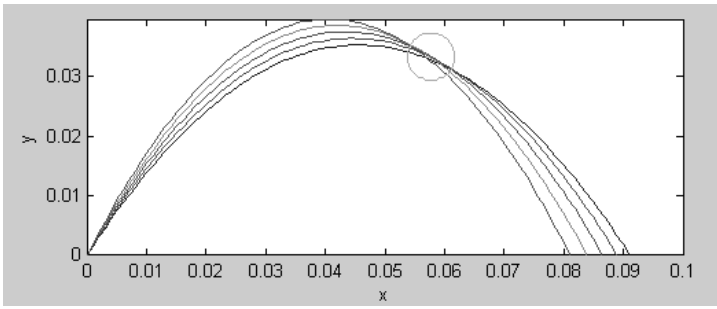

Рис. 1.2.7

```
>> a
a =

   0.9972   1.0222   1.0472   1.0722   1.0972
```

При отладке это удобно, однако после отладки это может даже помешать. Переделать программу в файл-функцию совсем просто, для этого достаточно заменить оператор clear, который перестает быть нужным на строку: function focus;. Очистите теперь память командой clear в командном окне и снова запустите программу. Вы получите:

Листинг 13

```
> clear
>> focus
alfa=60
>> a
??? Undefined function or variable 'a'.
```

При этом, как и прежде выведется правильный график. Однако в памяти среды уже нет переменной a, как и других переменных, используемых программой, написанной в файле-функции. Компилятор выдает сообщение об ошибке. Более того, если в памяти среды и были какие-то переменные, то при работе программы она игнорирует эти переменные и создает свои.

Этим, однако, не исчерпываются достоинства программы в виде файла-функции. Ее можно вызвать из другой программы, передав ей параметры. Параметры передаются при обращении, если после имени программы в скобках набрать эти параметры. Допустим, нам нужна программа, которая выдаст серию графиков при различных значениях α_0, если при обращении к программе мы введем соответствующие аргументы, и потребует вводить alfa, если мы просто набираем имя программы без аргументов. Нам не

придется сильно изменять программу. Листинг новой программы имеет следующий вид.

Листинг 14

```
% focusing of trajectories
function focus(alfa);
nx=100;
na=5;
b=0.1;
if nargin<1
  alfa=input('alfa=');
end
n=max(size(alfa));
da=b/(na-1);
g=10;
v0=1;
xmax=v0^2/g;
dx=xmax/(nx-1);
x=0:dx:xmax;
for i=1:n
  a0=alfa(i)*pi/180
  a=a0-b/2:da:a0+b/2;
  y=tan(a')*x-g*(1+(tan(a').^2))*(x/v0).^2/2;
  figure
  plot(x,y)
  axis equal
  ax=axis;
  ax(3)=0;
  ax(1)=0;
  ax(2)=xmax;
  axis(ax);
  xlabel('x');
  ylabel('y');
  hold on
  x1=v0^2*cot(a0)/g
  y1=v0^2/2/g*(1-(cot(a0))^2)
  t=0:pi/30:2*pi;
  ymax=max(max(y))
  r=ymax/10;
  x2=x1+r*cos(t);
  y2=y1+r*sin(t);
  plot(x2,y2)
end
```

Небольшой комментарий к программе. Идентификатор nargin зарезервирован для числа аргументов, с которыми вызывается программа. Если Вы вызываете программу просто именем focus, то nargin=0, и программа запрашивает ввод alfa, для чего ста-

вится конструкция из условного оператора. Если же вы вводите аргумент в виде числа, программа работает, как и прежде. Если же аргумент представляет собой вектор-строку или вектор столбец, то командой n=max(size(alfa)) определяется число элементов в столбце, после чего выполняется цикл. В цикле, помимо тех операторов, о которых уже говорилось, есть оператор figure. Этот оператор открывает новое окно для каждого из графиков. В его отсутствие все графики для различных значений α_0 будут выводиться в одно окно, стирая предыдущие, так что вы на них не успеете посмотреть.

Вызвав теперь программу строкой focus([50 60 70]), вы получите три окна с тремя графиками, подобными графику на рис. 1.2.7.

Темы для дальнейших самостоятельных исследований

1. Исследуйте, как изменяется размер области d на прямой MN, через которую проходят дробинки при изменении угла α, образуемого осью ствола с горизонтом.
2. Средствами, имеющимися в вычислительных средах, создайте анимацию процесса движения нескольких дробинок, выпущенных из ружья в один и тот же момент времени.
3. Рассмотрите как влияет на эффект фокусировки сопротивление воздуха при движении дробинок. Считайте, что сила сопротивления воздуха пропорциональна скорости дробинки.

1.3. Соударения трех шаров

Условие

Рассмотрите задачу о соударении трех шаров, насаженных на горизонтально расположенный стержень, по которому они могут скользить без трения. Массы крайних шаров равны m_1, m_2, масса среднего шара равна m, причем $m < m_1$ и $m < m_2$. В начальный момент крайние шары покоятся, а средний шар движется в сторону массы m_1 со скоростью v. Соударения шаров считаем абсолютно упругими. Исследуйте движение шаров при различных значениях параметров системы. При каком условии средний шар столкнется с крайними не менее n (например, $n = 50$) раз?

Решение

Соотношения, определяющие скорости двух шаров m_1 и m_2 после их абсолютно упругого соударения, легко получить с помощью законов сохранения энергии и импульса. Обозначим через

v_1 и v_2 проекции скоростей шаров на ось, направленную вдоль стержня, до соударения, а через v_1' и v_2' — после соударения. Тогда справедливы соотношения:

$$v_1' = \frac{(m_1 - m_2)v_1 + 2m_2v_2}{m_1 + m_2}, \tag{1.3.1}$$

$$v_2' = \frac{(m_2 - m_1)v_2 + 2m_1v_1}{m_1 + m_2}. \tag{1.3.2}$$

Теперь не представляет труда записать выражения для скоростей шаров в рассматриваемой задаче. Будем считать, что шар массой m_1 расположен на стержне правее шара массой m, и направим ось x вправо.

После первого соударения шара массой m с шаром массой m_1 имеем:

$$v' = \frac{(m - m_1)v + 2m_1v_1}{m + m_1}. \tag{1.3.3}$$

В этой формуле $v > 0$, так как ось x направлена вправо. Скорость шара массой m до первого соударения по условию равна нулю, но нам удобнее сразу записать формулу (1.3.3) в общем виде, так как дальше придется рассматривать много ударов. Легко видеть, что шар массой m после удара будет двигаться в противоположную сторону, в направлении шара массой m_2, только если $m < m_1$.

Скорость шара массой m_1 после первого соударения с шаром массой m есть

$$v_1' = \frac{(m_1 - m)v_1 + 2mv}{m + m_1}. \tag{1.3.4}$$

С шаром массой m_2 шар массой m столкнется в первый раз, имея проекцию скорости v', определяемую формулой (1.3.3). Поэтому выражения для проекций скоростей этих шаров после первого соударения можно записать в виде

$$v'' = \frac{(m - m_2)v' + 2m_2v_2}{m + m_2}, \tag{1.3.5}$$

$$v_2' = \frac{(m_2 - m)v_2 + 2mv'}{m_1 + m_2}. \tag{1.3.6}$$

Как и в случае столкновения с шаром массой m_1, скорость второго шара равна нулю до первого столкновения. Если $m < m_2$, то шар массой m изменит после удара направление своего движения. Если при этом проекция его скорости v'' окажется больше проекции скорости первого шара v_1', то шар массой m догонит первый шар и столкнется с ним еще раз. Для скоростей этих шаров после их второго соударения v''' и v_1'' можно воспользоваться

формулами (1.3.3) и (1.3.4), в которые вместо v следует подставить v'', а вместо v_1 подставить v_1'. Если окажется, что после удара $v''' < 0$, то шар массой m опять начнет двигаться в направлении шара массой m_2. Если при этом $v''' < v_2'$ (что соответствует условию $|v'''| > |v_2'|$), то шар массой m вторично догонит шар массой m_2, столкнется с ним и т. д.

Таким образом, рассматривая вопрос о полном числе соударений, нам придется вычислять скорости шаров после каждого удара и проверять, сможет ли шар массой m в очередной раз догнать шар массой m_1 или m_2. Очевидно, что эту работу можно поручить компьютеру. Только для этого нужно составить соответствующую программу.

Прежде всего, перепишем формулы (1.3.3)–(1.3.6) в более удобном виде, поделив числители и знаменатели их правых частей на массу среднего шара m. Это эквивалентно измерению масс крайних шаров в единицах массы среднего шара:

$$v' = \frac{(1 - m_1)v + 2m_1v_1}{1 + m_1}, \tag{1.3.7}$$

$$v_1' = \frac{(m_1 - 1)v_1 + 2v}{1 + m_1}, \tag{1.3.8}$$

$$v'' = \frac{(1 - m_2)v' + 2m_2v_2}{1 + m_2}, \tag{1.3.9}$$

$$v_2' = \frac{(m_2 - 1)v_2 + 2v'}{1 + m_2}. \tag{1.3.10}$$

В этих формулах отношение m_i/m $(i = 1{,}2)$ снова обозначено через m_i. Теперь 1 — это масса среднего шара, а m_1 и m_2 — безразмерные массы крайних шаров, измеренные в единицах массы среднего шара. Этой заменой мы уменьшаем число независимых параметров системы от трех до двух. Однако для наглядности программа составлена так, что можно вводить произвольные значения масс для каждого шара.

Остается ввести формулы (1.3.3)–(1.3.6) в программу, полностью повторив в ее логике приведенные рассуждения.

Теперь можно проводить вычислительный эксперимент, меняя различные параметры рассматриваемой системы. В среде Matlab можно, используя графический интерфейс пользователя (GUIDE), составить программу, результат выполнения которой отображается в графическом окне. Изменяя параметры можно выводить графики траекторий движения шаров и число состоявшихся соударений. На рис. 1.3.1 приведены результаты расчетов для случая: $m_1 = m_2 = 1$, $m = 0{,}5$, $v = 1$. Верхняя линия

на графике соответствует движению правого шара массой m_1, нижняя — движению левого шара массой m_2, а промежуточная — движению среднего шара. Как видно из этого рисунка, число соударений в данном случае равно 2.

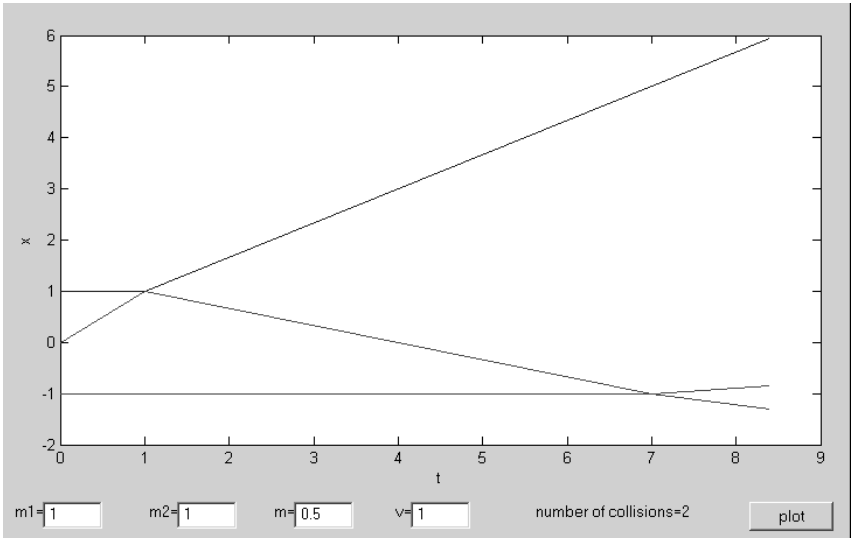

Рис. 1.3.1

Уменьшая значение массы m, мы можем увеличить число соударений. Вычислительный эксперимент показывает, что 30 соударений происходит при значении $m = 0{,}0014$ (рис. 1.3.2). Основное число соударений приходится на начальный участок графика и визуально они трудно различимы. Используя логарифмический масштаб, можно было бы «растянуть» этот участок и сделать соударения легко наблюдаемыми на графике.

Программирование в Matlab

На примере данной задачи мы покажем, каким образом можно использовать средства графического интерфейса пользователя (GUI), предоставляемого средой Matlab для создания приложения в виде программы, при запуске которой возникает графическое окно, и вся дальнейшая работа идет с использованием элементов в этом графическом окне. Такого рода программы оказываются удобными для конструирования приложений, позволяющих ставить различные вычислительные эксперименты. Мы не ставим своей целью сколько-нибудь подробное изучение программирования подобных приложений, предлагая ознакомиться

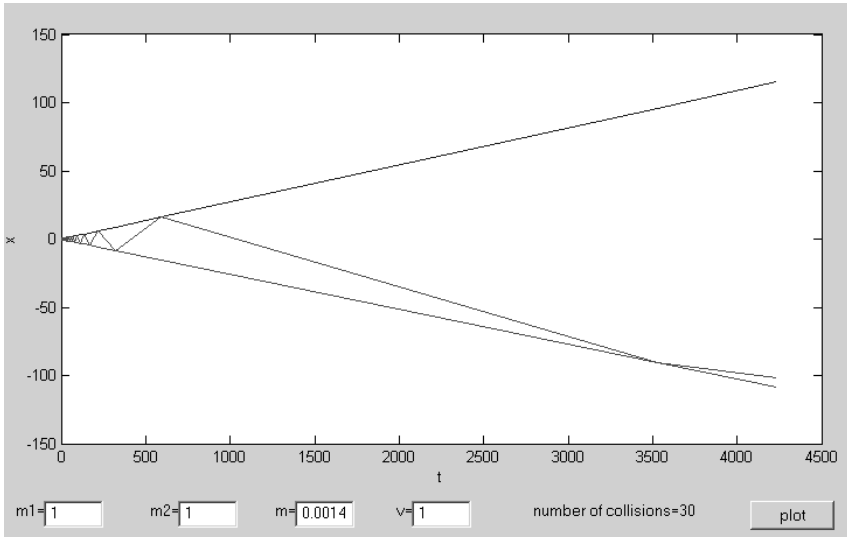

Рис. 1.3.2

с соответствующей литературой, а хотим показать возможности среды применительно к решению физических задач.

Работа по программированию подобных приложений сводится к работе в трех окнах плюс окно, создаваемое пользователем. Первое окно — командное, в котором будет вызываться программа. Второе окно — редактор программ. Третье окно, вызываемое из командного окна командой GUIDE, представляет собой окно среды GUIDE, предназначенной для разработки графического окна-программы.

После перехода в среду GUIDE можно либо создавать новое графическое окно, либо редактировать уже имеющееся. При создании нового окна появляется заготовка окна и панель инструментов с элементами, которыми можно заполнять создаваемое графическое окно. Мы перечислим эти элементы, останавливаясь подробно лишь на тех, которые нам потребуются в данной задаче. Заметим, что все, о чем идет речь, существует в версии Matlab 6.x. В более ранних версиях каких-то элементов может не быть.

1. Кнопка (Push Button). Элемент, по нажатию на который (кнопкой мыши) выполняются некоторые программные действия. В процессе работы программы кнопка может быть активной или пассивной, когда ее «нажать невозможно».

2. Кнопка-переключатель (Toggle Button).

3. Переключатель (Radio Button).

4. Флаг (Checkbox).

5. Область ввода текста (Edit Text). Используя этот элемент, можно вводить какие-либо данные для работы программы, например, значения некоторых переменных.

6. Текстовая область (Static Text). Этот элемент служит для вывода какого-либо текста. Слово «Static» не означает статичность. Выводимый текст может формироваться при работе программы.

7. Полоса скроллинга (Slider).

8. Рамка (Frame).

9. Список (Listbox).

10. Раскрывающийся список (Popup Menu).

11. Оси (Axes). Область для вывода графиков. Таких областей в окне может быть несколько. Заметим, что значения по осям среда может делать по умолчанию, но это же можно делать программно. Надписи по осям делаются программно.

Для нашей программы (рис. 1.3.2) мы используем один элемент «Оси», один элемент «Кнопка», пять элементов «Текстовая область» и четыре элемента «Область ввода текста». Все эти элементы размещаются в окне-заготовке при помощи мыши. При этом можно произвольно (в определенных ограничениях) изменять их размеры.

Каждому вновь размещенному элементу среда придает свойства. Ряд свойств — размеры и местоположение элемента — создаются, исходя из того, какими элементы были созданы. Ряд свойств — цвет, размер шрифта и т. д. можно изменять, используя вызываемое в меню среды окно редактора свойств (Property Inspector). В частности, одним из свойств является тег (Tag) — имя, по которому этот элемент будут вызываться в программе. По умолчанию среда присваивает теги, нумеруя их по мере создания. Например, при создании нескольких областей ввода текста возникнут элементы с тегами: edit1, edit2, … Чтобы впоследствии не запутаться в номерах, полезно переименовать теги в соответствии с обозначениями вашей задачи. Другие свойства, которые сразу же полезно изменить, — это текст, который будет на кнопках, и выводимый текст, который будет появляться сразу же после запуска программы.

Есть еще одно полезное свойство, которое следует изменить сразу же при создании окна, если результаты работы программы предназначены для использования в каких-либо других программах, например, при создании документа Word. Это свойство присваивается всему создаваемому окну и называется Menu Bar. Оно представляет собой меню создаваемого графического окна.

По умолчанию среда определяет это свойство как «none», что означает отсутствие меню. Если Вам захочется скопировать созданное окно в другую программу, то следует «none» изменить на «figure». Тогда у окна будет стандартное меню, и в частности, команда копирования окна в буфер.

Все действия по созданию графического окна не требуют особых комментариев, с ними справится любой пользователь, умеющий работать в среде Windows.

После создания графического окна требуется сохранить его под каким-либо именем, по которому он будет вызываться из командного окна. Среда присваивает этому файлу расширение fig. При сохранении среда автоматически создает файл-программу под тем же именем, но со стандартным расширением .m. Автоматически этот файл помещается в окно редактора программ, где и предстоит дальнейшая работа. Однако если потребуется дополнить окно какими-то элементами или изменить свойства уже существующих, можно опять вернуться к окну среды GUIDE, автоматически оно не закрывается.

Запустить программу с созданным окном можно сразу же, до редактирования программы из командного окна. В нашем случае возникнет окно, представленное на рис. 1.3.3.

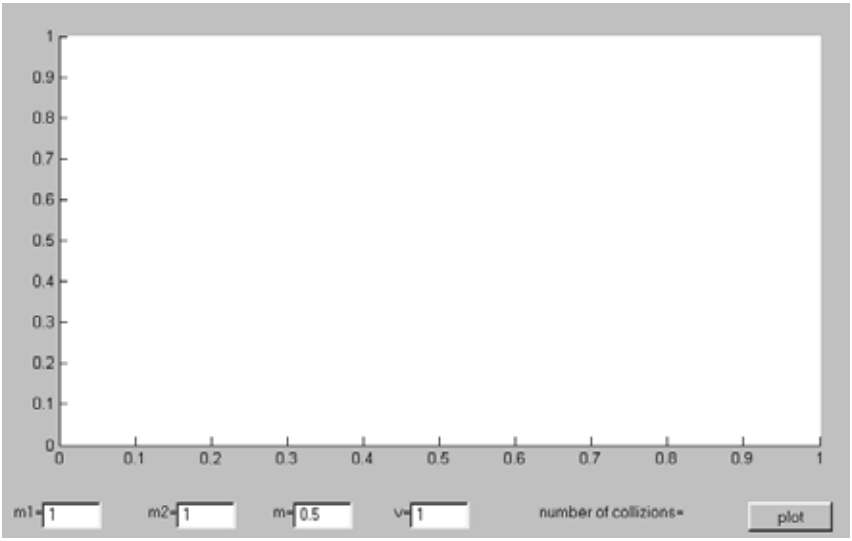

Рис. 1.3.3

Заметим, что название кнопки и строки в области текста и вводимого текста мы создали уже при конструировании окна. Естественно, программа никаких действий не выполняет, хотя