В. И. Зенкин

КУРС МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

© В. Зенкин. Калининград, 2015

© В. Зенкин. Калининград, 2015

Вёрстка текста и иллюстрации автора.

Оглавление

Введение

Математическое и компьютерное моделирование в самых различных областях науки и техники является сейчас одним из основных способов получения новых научных знаний и технологических решений. Это — наиболее гибкие и универсальные методы исследования реальных объектов, процессов и явлений, с успехом применяемые в физике, астрономии, химии, биологии, медицине, экономике, военном деле, технических и социальных науках и многих других областях.

Для применения методов математического и компьютерного моделирования исследователь, независимо от его специальности, должен разбираться в алгоритмах вычислительной математики и владеть способами их программной реализации. Эти знания и навыки необходимы даже при использовании готовых пакетов программ, иначе будут затруднительны или вообще невозможны анализ работы компьютерной модели, планирование и проведение вычислительных экспериментов и интерпретация их результатов. Хотя в настоящее время имеются несколько мощных программных пакетов для моделирования, таких как Simulink Matlab, Model Vision Studium и др., но они достаточно сложны в изучении, дороги и, что самое главное, при работе с ними исследователь вынужден полностью полагаться на правильность работы этих программ, что никто гарантировать не может.

Основное внимание в книге уделено *практическим* методам построения математических и компьютерных моделей, весь необходимый теоретический материал даётся в краткой форме и снабжён примерами. Для понимания материала необходимы базовые знания языков Object Pascal (Delphi), C/C++. Приведён список задач, достаточный для организации лабораторных и практических работ по данному курсу.

Структура книги. В первой главе сформулированы все необходимые для практической работы понятия, относящиеся к математическому моделированию. Приведены простейшие примеры построения математических моделей. Вторая глава посвящена разработке компьютерных моделей. На примере языка Object Pascal (Delphi) показаны способы программной реализации математических моделей средствами объектно-ориентированного метода программирования. В третьей главе собраны примеры моделей из физики, биологии и физиологии, военного дела, социологии и политологии и информатики. Для всех моделей подробно указаны методы построения, программной реализации и анализа. Дополнительно сформулированы задачи, рекомендуемые для самостоятельного решения, а также вопросы, решение которых приведено в четвертой главе книги. В Приложении собран весь необходимый для решения задач теоретический материал.

Почта автора открыта для отзывов, замечаний и критики книги.

I. МАТЕМАТИЧЕСКИЕ МОДЕЛИ

1. Математическое моделирование

Науки не пытаются объяснить, вряд ли они даже стараются интерпретировать — они в основном создают модели. Под моделью понимается математическая конструкция, которая при добавлении некоторых словесных объяснений описывает изучаемый феномен. Оправданием для такой математической конструкции служит единственное обстоятельство: ожидается, что она сработает.

Дж. фон Нейман, цит. по [17]

Может показаться, что написанные нами уравнения сами содержат структуру реального мира, которым можно управлять, манипулируя этими уравнениями.

Л. Купер, [3]

1.1. Модели. Метод моделирования

Моделью[1] какого-либо объекта (явления, феномена, процесса) называют другой объект, реальный или формальный, некоторые свойства которого *частично* совпадают со свойствами моделируемого объекта. Ввиду сложности реального мира при исследовании его явлений, процессов или объектов их обычно в той или иной мере упрощают, выделяя те свойства, которые считают основными для рассматриваемого объекта или явления, и отвлекаясь от несущественных или малосущественных деталей. Такое упрощение неизбежно при любом исследовании хотя бы по той причине, что любой реальный объект имеет бесконечно много различных свойств и характеристик и, следовательно, даже перечислить их все, а тем более изучить, нет никакой возможности.

В некотором смысле можно утверждать, что вся научная деятельность — можно даже сказать, значительная часть *интеллектуальной* деятельности — сводится к построению и анализу моделей физических, биологических, химических, технических, экономических, социальных, политических и других процессов, явлений и объектов. На это есть причины: способность к моделированию, в частности — с целью *предсказания* развития событий, была и остается необходимым условием выживания для людей. «По всей видимости, человек стал человеком не тогда, когда сделал палку или камень орудием труда, не тогда, когда освоил членораздельную речь, а тогда, когда научился моделировать окружающий мир и помещать в эту модель себя самого» [5].

Метод моделирования обычно применяют для изучения исходных объектов тогда, когда непосредственное их изучение либо по каким-то

[1] От лат. modulus — мера, аналог, образец.

причинам неудобно (очень дорого[2], требует слишком много времени[3] или опасно[4]), либо вообще невозможно (моделируемый объект может не существовать в реальности[5] или его прямое натурное исследование неизбежно приведёт к катастрофе[6]).

Основными целями, ради которых создаются модели, являются:

- Подтверждение или опровержение различных теорий и гипотез.
- Выявление зависимостей различных параметров модели, характера их взаимодействия во времени и пространстве, нахождение оптимальных соотношений этих параметров.
- Прогнозирование поведения объектов моделирования, чтобы, в частности, получить возможность ими управлять.
- Применение в качестве систем виртуальной реальности или тренажёров при подготовке персонала к работе на смоделированных устройствах (системы реального времени).

Следует ясно понимать, что не бывает модели без упрощений и, следовательно, любая модель не может быть тождественна оригиналу. Для этого существуют, по крайней мере, две причины. Во-первых, модель, в которой «для реализма», присутствует много параметров (а любой реальный объект имеет бесконечно много параметров, его характеризующих), может оказаться практически необозримой. Действительно, если некоторая модель имеет всего десять входных параметров, каждый из которых независимо от других может принимать десять различных значений, то только для тестирования модели в полном объёме (т. е. при всевозможных значениях параметров) понадобится $10^{10} = 10\,000\,000\,000$ прогонов. Не трудно посчитать, что если на один

---

[2] Например, модель процессов в переходной экономике нашей страны начала 90-х годов XX в, см. [1, стр. 302–306].

[3] Например, моделирование игры в шахматы посредством прямого перебора всех возможных ходов. По оценкам К. Шеннона число возможных ситуаций в шахматной партии равно 10^{43}. Исследование *всех* ситуаций пока лежит за пределами возможностей любого суперкомпьютера [4, стр. 105]. Для шашек оценка существенно ниже — $5 \cdot 10^{20}$ различных вариантов. Канадские специалисты заявили (2007 г), что создали компьютерную программу, названную ими Chinook, для игры в шашки (точнее, в их разновидность, именуемую «Checkers»), которую невозможно обыграть. Для разработки алгоритма потребовалось 50 компьютеров и почти 20 лет времени. Испытать программу в деле можно на http://webdocs.cs.ualberta.ca/~chinook/play/.

[4] Например, модели гонки вооружений, см. [1, С. 173–175], боевых действий — пример на стр. 70, эпидемий — стр. 53.

[5] Например, проект космического корабля на фотонной тяге, «математическая реставрация» Тунгусского феномена [1, стр. 288–291], «зомби-апокалипсис» — стр. 57.

[6] Например, модель климатических последствий ядерного конфликта — «ядерная зима», см. [1, стр. 192–296].

прогон модели затратить только одну минуту, то такое «тестирование» продлится более 19 000 лет. Во-вторых, модель, полностью совпадающая с оригиналом, также бесполезна, как географическая карта в масштабе $1:1$, поскольку в этом случае отсутствует наиболее полезная черта моделей — абстрактность[7].

Поскольку модель никогда полностью не совпадает по своим свойствам с оригиналом, встает проблема допустимой степени их различия. С одной стороны, она должна отражать все свойства исходного объекта или явления, которые существенны для него, иначе модель бесполезна. С другой стороны, необходимо, чтобы модель была как можно более простой, иначе её исследование будет затруднительно или невозможно.

Для сложных явлений и объектов часто очень трудно совместить эти противоречивые требования[8]. Таким образом, при построении модели основной и наиболее трудной задачей является вопрос о мере соответствия модели моделируемому объекту (адекватности) и, следовательно, проблема определения степени «существенности» параметров объекта. Причём, проблема осложняется тем, что какие свойства считать основными, а какие несущественными зависит как от моделируемого объекта, так и от целей исследования. Поскольку очевидно, что одному и тому же явлению могут соответствовать несколько различных моделей, то, при прочих равных условиях, предпочтительнее та из них, которая в каком-то смысле проще других.

Метод моделирования, по существу, основан на рассуждении по *нестрогой аналогии:* если некоторый объект (оригинал) с какими-либо свойствами имеет сходство с другим объектом (моделью), обладающим некоторыми из свойств первого, то по *поведению и параметрам модели* делают вывод об аналогичном *поведении и соответствующих параметрах оригинала* (см. рис. I.1).

Например, если объект A имеет свойства x, y, z, а объект B обладает характеристиками y, z, то отсюда заключают, что последний *вероятно* имеет также и свойство x. Считается, что чем больше у объектов общих свойств, тем эта вероятность выше и, следовательно, аналогия более «надежна»[9].

---

[7] Льюис Керрол, математик и известный писатель, правда в шутку, предлагал пользоваться именно такой картой, так как её достаточно разложить на земле, чтобы в любой момент знать, где находишься: нужно просто прочитать надпись, на которой стоишь. А «отец кибернетики» Норберт Винер с присущей ему ироничной афористичностью писал: «Наилучшей моделью кота является другой кот, а еще лучше — тот же самый кот».

[8] «Ученый, подобно Паломнику, должен идти прямой и узкой тропой между западнями Переупрощения и Болотом Переусложнения» — Р. Белман. Динамическое программирование. М., 1960, стр. 11.

[9] Примеры: 1) если A — умная (x), красивая (y) блондинка (z), то красивая (y) блондинка (z) B также умна; 2) если A — богатая (x), де-

Очевидно, что такой подход не совсем надёжен, даёт только вероятное знание и, следовательно, *любая* модель нуждается в проверках, практических[10] (сравнение в пределах изначально задаваемой точности результатов, следующих из модели, и реальных, полученных в натурном эксперименте) и иных (внутренняя логическая непротиворечивость модели, сравнение с другими моделями, соответствие фундаментальным законам природы и другие).

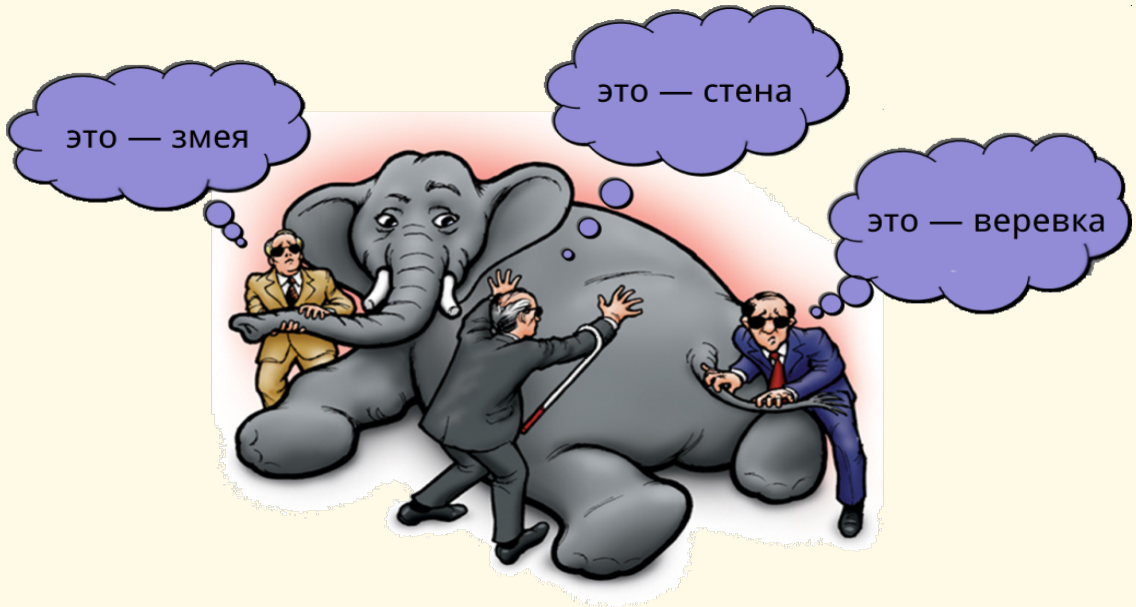


Рис. I.1. *Процесс моделирования, метод аналогий: модель змеи, зловеще спускающейся по веревке со стены*

Специально отметим: никакое количество успешных проверок модели логически *не доказывают* её истинность, а лишь отчасти *подтверждают* её адекватность. В то же время, даже один единственный тест модели, выявивший её существенное отличие от реального объекта-прототипа, полностью *опровергает* модель или, по крайней мере, сокращает область её применимости. Если модель не соответствует оригиналу больше, чем это изначально предусматривалось при её построении, то, в зависимости от степени несоответствия, нужно либо уточнить модель, либо полностью её пересмотреть. Модели, которые невозможно проверить практически[11], должны, как минимум, не противоречить известным фундаментальным принципам и законам природы.

мократическая (y) Западная Европа (z), то демократическая (y) Россия B также будет богатой. Вторая аналогия, конечно, чуть менее обоснована, чем первая, т. к. общих свойств у объектов A и B меньше, но какое дело до этого блондинкам, особенно при переходе от «тоталитаризма» к «демократии».

[10] Практика — критерий истины. «Практика выше (теоретического) познания, ибо она имеет не только достоинство всеобщности, но и непосредственной действительности» — В. И. Ленин. Конспект книги Гегеля «Наука логики». ПСС, изд. 5, т. 29, стр. 195.

[11] Например, модель Большого взрыва при образовании Вселенной.

1.2. Математические модели

Все модели можно разделить на два класса: *физические* и *абстрактные*. Первые обычно представляют собой упрощённую реальную копию объекта (например, макеты самолётов или морских судов). Абстрактные модели представляют собой описание объектов при помощи некоторых символов, схем, средств формальных языков и т. д. (например, географическая карта, электрическая схема какого-либо устройства, системы разностных или дифференциальных уравнений).

Определение 1.1. *Математической моделью называют абстрактную модель, в которой реальные объекты исследования заменяются идеальными и описываются при помощи математических соотношений или различных алгоритмических схем.*

Главными и наиболее характерными чертами математических моделей являются их:

- *абстрактность* (реальные объекты всегда заменены их абстракными идеализированными аналогами, к примеру, физическое тело — материальной точкой) и
- *общность* (например, механические и электрические колебания описываются в самом общем виде одними и теми же дифференциальными уравнениями, см. рис. I.2).

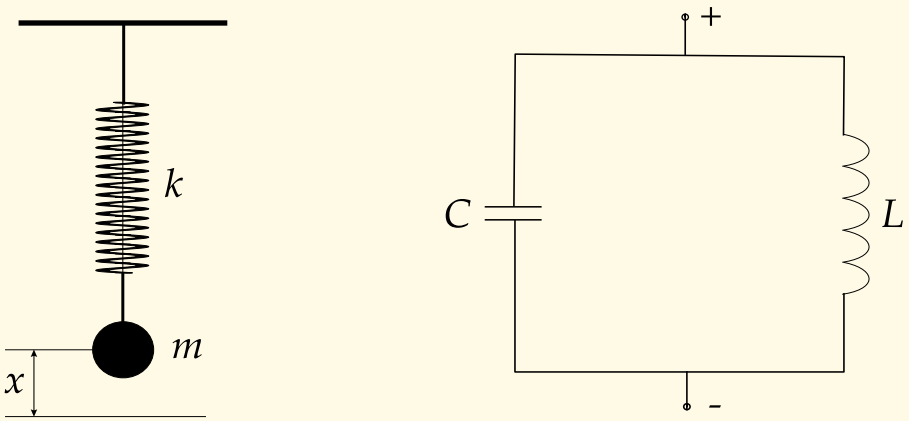


Рис. I.2. *Два разных физических явления описываются одной математической моделью*

В частности, абстрактность позволяет построить математическую модель так, как создаются любые математические теории, на основе *аксиоматического метода*, то есть вначале выделяются основные (неопределяемые) понятия, формулируются аксиомы теории, а затем все остальные утверждения (теоремы, леммы) выводятся из них чисто логическим путём.

Например, классическая механика основана на понятиях инерции, силы и законах (аксиомах) Ньютона и всемирного тяготения. Класси-

ческая электродинамика построена на уравнениях Максвелла — Лоренца, математически выражающих законы (аксиомы) электромагнитного поля — законы Гаусса, Фарадея и др.

Таким образом, вместо исследования реального явления или процесса изучается его математическая модель. Единственное отличие по сравнению с чисто математической теорией — следствия аксиом не должны противоречить эмпирическим фактам. Философский вопрос о причинах успешной применимости математических теорем к реальному миру — почему *абстрактные* понятия и теории позволяют описывать *реальный мир*? — оставим здесь без внимания[12].

Общность. Математическая модель, задаваемая дифференциальным уравнением (см. рис. I.2)

$$\ddot{y} = -\alpha y, \qquad \alpha = \text{const},$$

описывает два физически разных явления: движение груза (материальной точки) на пружине и электромагнитные колебания. Различие лишь в физической интерпретации функции $y = y(t)$, где t — время, и коэффициента α. В первом случае $y = x$, $\alpha = k/m$, где k — жесткость пружины, m — масса, во втором y — сила тока в цепи, $\alpha = 1/LC$, L — индуктивность, C — емкость конденсатора. Помимо замены физического тела материальной точкой, в модели присутствуют такие идеализации, как: нерастяжимая и невесомая нить, на которой подвешено тело, полное отсутствие активного сопротивления в цепи колебательного контура.

Именно эти особенности математических моделей дают возможность их строгого исследования, что особенно важно при рассмотрении сложных систем, когда словесные описания становятся очень запутанными и, в следствие этого, не могут гарантировать правильность выводов.

Примерами блестящих математических моделей являются: гелиоцентрическая система Коперника — Кеплера, механика Ньютона, модель электромагнитного поля Максвелла, теория относительности, теория информации, теория игр и многие другие.

1.3. Признаки хороших моделей

Из вышесказанного видимо уже понятно, что, вообще говоря, построение более-менее нетривиальной математической модели — процесс творческий, а не шаблонный, скорее искусство, чем наука. Поэтому, как принято в искусстве, при создании моделей можно ориентироваться на имеющиеся источники вдохновения и образцы — [1], [6], [16], [22].

Таким образом, можно сформулировать задачу моделирования как процесс построения образцовой «хорошей» модели, обладающей определенными присущими ей признаками.

---

[12] «...невероятная эффективность математики в естественных науках есть нечто граничащее с мистикой, ибо никакого рационального объяснения этому факту нет». — Е. Вигнер. Непостижимая эффективность математики в естественных науках. УФН, т 94, вып. 3, 1968.

Хорошая математическая модель имеет следующие свойства:

1. **Адекватна** (т. е. в данном случае способна описать моделируемое явление с требуемой численной точностью, не превосходящей, конечно, точности экспериментальных измерений параметров исходного явления) моделируемому объекту или явлению;
2. **Позволяет получить новые сведения,** неизвестные до построения модели, или уточняющие известные сведения;
3. **Проста** настолько, насколько это возможно (имеет меньшее число параметров, менее сложное математическое описание и т. д.).

Адекватность, очевидно, — абсолютно необходима для любой модели. Проверка соответствия результатов расчетов на модели поведению реального объекта осуществляется при помощи *вычислительного эксперимента*. Вычислительный эксперимент состоит в получении результатов для какого-либо конкретного набора значений параметров математической модели. Если, для примера, ограничиться рассмотрением моделей детерминированных механических систем и процессов, то тест адекватности таких моделей сводится к проверке их *точности* и *непротиворечивости*. Под точностью понимается отклонение результатов численного эксперимента от значений натурного эксперимента не более чем на наперед заданную величину допустимой погрешности. Непротиворечивость здесь означает одинаковый характер поведения соответствующих модельных и найденных эмпирически параметров, т. е. идентичный вид их основных функциональных зависимостей, как-то: возрастание, убывание, экстремумы, неотрицательность, ограниченность и т. п. Для проектируемых систем, когда её реальные объекты не существуют, тесты сводятся к одной лишь проверке на непротиворечивость.

Крайне желательно и второе требование к модели. В качестве наиболее ярких примеров здесь можно привести открытие в 1841 г. Леверье и Адамсом «на кончике пера» планеты Нептун, предсказание существования позитрона, сделанное в 1932 г. Дираком, исходя из теоретических представлений, открытие в 70-х годах XX в. т. н. «Т-слоя», сделанное в Институте прикладной математики АН СССР, и множество других.

Требование простоты для математических моделей является, по существу, следствием уверенности учёных в рациональном устройстве мира. Например, гелиоцентрическая система считается предпочтительнее геоцентрической именно по причине своей большей простоты, хотя обе эти теории позволяют одинаково точно описать поведение и параметры моделируемой ими Солнечной системы.

Кроме того, как отмечалось выше, наличие большого количества параметров у модели, может сделать её практически недоступной для исследования, и часто не увеличивает, а наоборот уменьшает адекватность модели[13]. «Появление и широкое внедрение компьютеров породило ил-

[13] «Сложные модели редко бывают полезными (разве что для диссертан-

люзию, что „чем больше учтем, тем лучше“. (Это сродни мнению, бытующему среди некоторых исторических школ, что „все существенно“.) При этом построение модели сложного явления часто сравнивали со складыванием мозаики. Провал нескольких крупных исследовательских проектов показал, что так действовать нельзя. Например, американский проект „Биосфера“, связанный с моделированием экологических процессов, в котором участвовало около 700 ведущих специалистов, „складывающих мозаику“, привел к результатам, не допускающим какой-либо разумной интерпретации» [6].

2. Примеры простых моделей

Традиционно математические модели чаще всего описываются дифференциальными, см. разд. 22, [10], а также разностными уравнениями, разд. 33, [15]. Дифференциальные уравнения получают из условий, связывающих параметры моделируемой системы, пользуясь физическим смыслом производной: если $x = x(t)$ — некоторый параметр модели, зависящий от времени t, то его производная $\dot{x} = dx/dt$ является *скоростью изменения* x, соответственно вторая производная $\ddot{x}$ есть скорость скорости, т. е. *ускорение*. Очевидное ограничение, накладываемое на функцию x: она должна быть дифференцируема. Разностные уравнения связывают дискретные значения параметров модели.

Уравнения модели в той или иной мере выражают *законы*[1] тех областей науки, с которыми они связаны, например, в механике — законы Ньютона или принцип Даламбера — Лагранжа, в теории электрических цепей — законы Кирхгофа, в химии — закон действия масс и т. д. В случаях, когда такие законы неизвестны, прибегают к различным *гипотезам* относительно характера моделируемых явлений или процессов. В частности, предположения делаются относительно протекания процесса при *малых изменениях* некоторых его параметров, а затем предельным переходом получают дифференциальные уравнения.

«Как и при составлении алгебраических уравнений, при решении прикладных задач по дифференциальным уравнениям многое зависит от навыков, приобретаемых упражнением. Однако здесь еще в большей степени требуется изобретательность и глубокое понимание сути изучаемых процессов» [10].

Пример 2.1. *Естественный радиоактивный распад подчиняется эмпирически установленному Ф. Содди и Э. Резерфордом (1903 г) закону: число распадов атомов за один интервал времени в произвольном веществе пропорционально количеству имеющихся в образце радиоактивных атомов данного типа. Средняя вероятность λ распада ра-*

тов)» — В. И. Арнольд. О преподавании математики.

[1] Или, по меньшей мере, им не противоречат.

диоактивных ядер в каждую следующую единицу времени не зависит от времени, прошедшего с начала наблюдения, и от количества ядер, оставшихся в образце, а также, в большинстве случаев, практически не зависит от окружающих условий — температуры, давления и т. д.

Пусть $m = m(t)$ — масса радиоактивного вещества, t — время, m_0 — масса вещества в начальный момент времени $t = 0$, тогда, учитывая, что масса пропорциональна количеству атомов вещества, закон Содди — Резерфорда математически можно записать в виде

$$\lambda = -\frac{\dfrac{\Delta m}{m}}{\Delta t}, \qquad m(0) = m_0,$$

что в словесной формулировке означает: за *малый* интервал времени Δt относительное изменение (точнее — уменьшение, т. к. перед дробью стоит знак минус) количества вещества $\Delta m/m$ постоянно[2]. Заменяя приращения дифференциалами (и, тем самым, следовательно, дополнительно предполагая дифференцируемость функции $m(t)$), получим

$$\lambda = -\frac{\dfrac{\mathrm{d}m}{m}}{\mathrm{d}t}, \qquad m(0) = m_0,$$

или

$$\dot{m} = -\lambda m, \qquad m(0) = m_0. \tag{1}$$

Из решения задачи Коши (1)

$$m = m_0 e^{-\lambda t} \tag{2}$$

ясно, что радиоактивный распад идёт по экспоненте и тем быстрее, чем больше величина λ. При $\lambda = 0$ из (2) следует, что $m(t) = m_0 = \mathrm{const}$, т. е. распада вещества не происходит.

Другую модель получим, если будем считать, что измерения массы проводятся через некоторый постоянный интервал времени Δt. Пусть $m_0 = m(0)$, $m_1 = m(\Delta t)$, $m_2 = m(2\Delta t)$, $\ldots$, $m_n = m(t)$ — полученная последовательность. Тогда закон Содди — Резерфорда выражается соотношением

$$\lambda = -\frac{\dfrac{m_{i+1} - m_i}{m_i}}{\Delta t}, \qquad i = 0, \ldots, n-1,$$

[2] «Следует отметить, что закон превращений одинаков для всех радиоэлементов, являясь очень простым и в то же время практически необъяснимым. Этот закон имеет вероятностную природу. Его можно представить в виде духа разрушения, который в каждый данный момент наугад расщепляет определенное количество существующих атомов, не заботясь об отборе тех из них, которые близки к своему распаду. Эта доля атомов называется постоянной распада и обычно обозначается символом λ». — Ф. Содди. История атомной энергии. М.: Атомиздат, 1979, стр. 110.

откуда получаем линейное разностное уравнение

$$m_{i+1} = m_i(1 - \lambda\Delta t), \qquad \Delta t = \frac{t}{n}, \tag{3}$$

имеющее решение (см. разд. 3.1)

$$m_n = m_0\,(1 - \lambda\Delta t)^n\,. \tag{4}$$

Отметим, что в модели (3) нам не понадобилось дополнительное требование о дифференцируемости функции $m(t)$. Непрерывная модель (1), как не трудно проверить, получается из дискретной модели (3) предельным переходом при $\Delta t \to 0$ ($n \to \infty$).

Пример для иллюстрации закона Содди — Резерфорда: временная динамика распада изотопа ксенона $^{133}\mathrm{Xe}$ в соответствии с непрерывной и дискретной моделями представлены на рис. I.3 (время отсчитывается в днях, изначальная масса вещества m_0 принята равной 1).

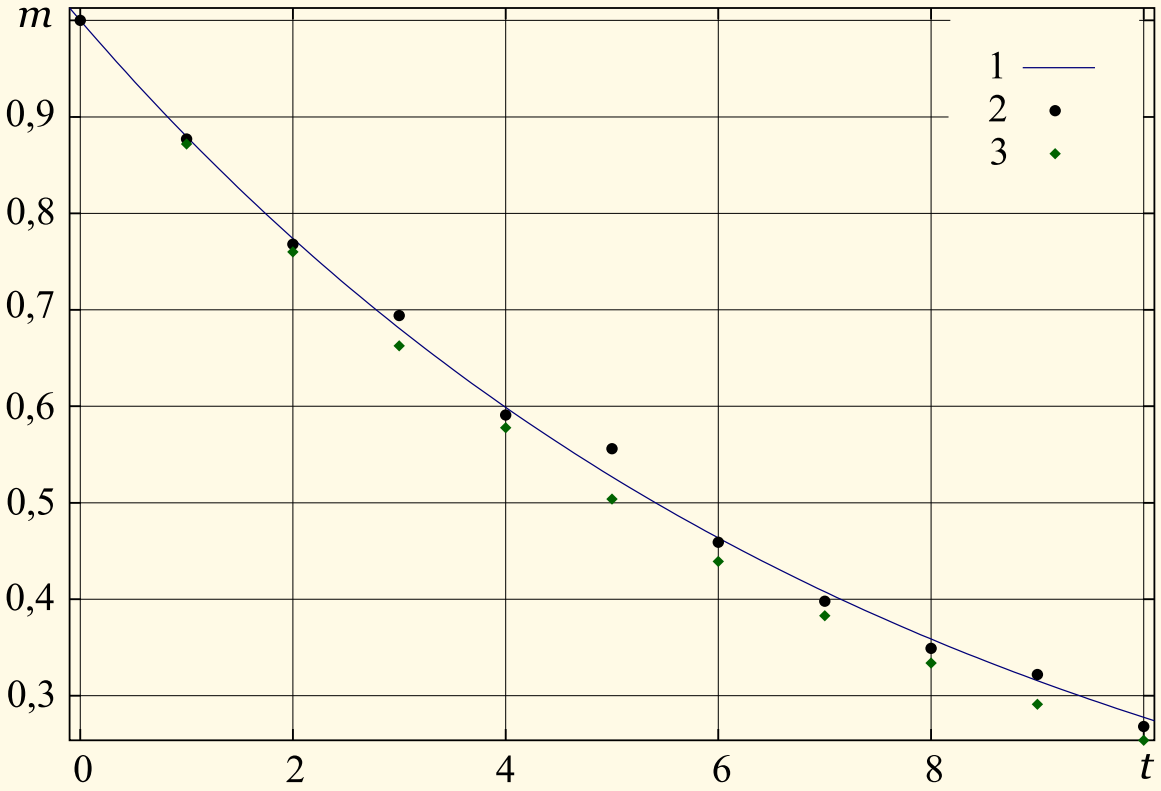


Рис. I.3. *Модели радиоактивного распада:* 1 — *непрерывная модель* (1), 2 — *эмпирические данные,* 3 — *дискретная модель* (3)

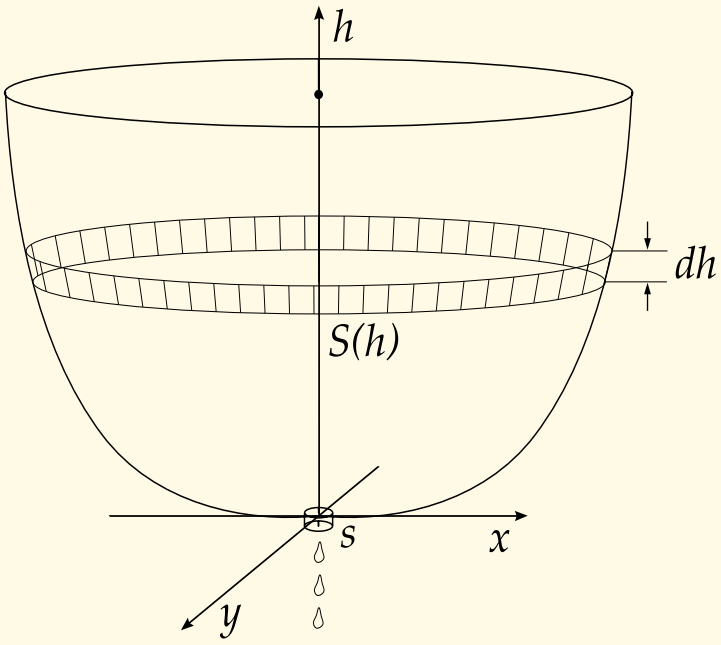
Значение коэффициента $\lambda = 0{,}128$ определялось с помощью аппроксимации эмпирических данных

t	0	1	2	3	4	5	6	7	8	9
m	1	0,877	0,768	0,674	0,591	0,578	0,452	0,398	0,349	0,302

функцией $e^{-\lambda t}$ по методу наименьших квадратов, см. стр. 179.

Пример 2.2. *Водяные часы (клепсидра) в виде сосуда с истекающей струей воды известны со времен Древнего Египта. У древних греков и римлян ими, в частности, определялась длина речей ораторов в суде. Промежуток времени измерялся количеством воды, вытекшей из малого отверстия, сделанного в дне сосуда*[3]*. Задача: какова должна быть форма сосуда, чтобы вода вытекала вполне равномерно*[4]*.*

Будем считать, что внутреннее трение (вязкость) у воды отсутствует и площадь отверстия слива много меньше площади поперечного сечения сосуда. Тогда скоростью изменения уровня жидкости в клепсидре можно пренебречь и из закона сохранения энергии имеем



$$mgh = \frac{1}{2}\,mv^2,$$

где m — масса воды, вытекшей из сосуда при понижении её уровня на величину h, со скоростью истечения, равной v. Отсюда, при принятых допущениях, получаем, что скорость истечения воды зависит только от изменения уровня жидкости в сосуде:

$$v(h) = \sqrt{2gh}. \qquad (5)$$

Пусть в момент времени t высота уровня воды в сосуде равна h, площадь поперечного сечения сосуда равна $S = S(h)$, площадь сливного отверстия — s. Тогда объем dV вытекшей воды за *малый* промежуток времени dt, за который уровень воды упадет на dh, можно определить двумя способами:

$$dV = -S(h)dh = sv(h)dt.$$

Оба выражения определяют объемы цилиндров, первый — с площадью основания $S(h)$ и высотой dh (знак минус — из-за убывания высоты со временем), второй — с площадью s и высотой, равной $v(h)dt$.

Отсюда имеем дифференциальное уравнение

$$dt = -\frac{S(h)dh}{sv(h)} \overset{(5)}{=} -\frac{S(h)dh}{s\sqrt{2gh}},$$

[3] Отсюда происходят выражения: «время истекло», «с тех пор много воды утекло» и т. д.

[4] Этой задачей, среди прочих, занимались Галилей, Вариньон и Бернулли.

которое, если переписать его в виде

$$\frac{dh}{dt} = -\frac{s\sqrt{2gh}}{S(h)},$$

имеет вполне ясный физический смысл: скорость падения уровня воды dh/dt прямо пропорциональна площади сливного отверстия s и скорости истечения $\sqrt{2gh}$, и обратно пропорциональна площади поперечного сечения сосуда $S = S(h)$ на высоте h.

Из последнего соотношения следует

$$\sqrt{h} = -\frac{S(h)}{s\sqrt{2g}}\,\frac{dh}{dt}, \tag{6}$$

откуда находим такую форму сосуда, при которой уровень воды убывал бы с постоянной скоростью. В уравнении (6) положим $dh/dt = \mathrm{const}$. Клепсидра, в силу её очевидной симметрии, должна быть поверхностью вращения, т. е. $S(h) = \beta(x^2 + y^2)$, где $\beta = \mathrm{const}$. Поэтому из (6) следует, что

$$h = \gamma\left(x^2 + y^2\right)^2,$$

где $\gamma = \mathrm{const}$, т. е. форма чаши водяных часов получается вращением кривой $h = \gamma x^4$ вокруг оси h.

Пример 2.3. *Отладка компьютерной программы (debugging) состоит из нескольких последовательных итераций, включающих операции тестирования программного кода и исправления найденных при этом ошибок (см. стр. 36). Одновременно при правке кода к уже имеющимся добавляются новые ошибки. Построим модель процесса отладки для определения количества ошибок в зависимости от числа итераций.*

Исходя из модели нужно получить оценку требующихся усилий (число шагов отладки) для достижения приемлемого процента ошибок в программе (предполагается, что приближенно начальное количество ошибок известно).

Известно, что средняя вероятность наличия *необнаруженных* в программе ошибок прямо пропорциональна числу ошибок, уже найденных ранее [62, стр. 30][5]. Вместе с исправлением ошибок на каждом шаге отладки при редактировании кода отлаживающим программистом вносятся свои собственные новые ошибки[6].

Пусть e_n — количество ошибок, имеющихся в программе на n-ой итерации. Тогда, если обозначить a долю обнаруженных и исправленных

[5] «Принцип, не согласующийся с интуитивным представлением», но, тем не менее, подтвержденный опытом.

[6] «В большой программе, рассчитанной на широкое применение, каждая шестая вновь обнаруженная ошибка может быть допущена при предшествующем внесении изменений в программу [62].»

на предыдущем шаге ошибок, то e_n равна $(1-a)e_{n-1}$ плюс некоторое количество добавленных программистом ошибок, которое пропорционально e_{n-1}. Будем считать, что коэффициент a постепенно увеличивается с каждой итерацией отладки вследствие накопления тестировщиком опыта и лучшего понимания алгоритмов работы программы, примем $a = \alpha n$, где $\alpha = \mathrm{const}$. Тогда модель будет описываться разностным уравнением

$$e_n = (1 - \alpha n)e_{n-1} + \beta e_{n-1},$$

где β — доля добавленных при отладке ошибок, которую будем считать постоянной на всех итерациях отладки, предполагая тем самым, что время и опыт полученный при отладке слишком малы, чтобы повлиять на собственную «криворукость» тестировщика. Количество изначально присутствующих в программе ошибок e_0 считаем заданным[7].

Таким образом, имеем следующее описание модели

$$e_n = (1 - \alpha n + \beta)e_{n-1}, \qquad 0 < \alpha, \beta < 1, \quad n < \frac{1+\beta}{\alpha}. \tag{7}$$

Последнее неравенство, по смыслу величин в (7), следует из положительности коэффициента $1 - \alpha n + \beta$.

Учитывая его, решение (7) получаем стандартным для линейных разностных уравнений способом (см. стр. 169):

$$e_n = e_0 \prod_{1 \leqslant k < \frac{1+\beta}{\alpha}} (1 - \alpha k + \beta). \tag{8}$$

Оценим количество итераций отладки, требующихся для того, чтобы уменьшить количество ошибок в программе до относительной доли r (полное избавление от ошибок на практике часто нереально). Другими словами, из соотношения

$$\frac{e_n}{e_0} < r$$

нужно найти оценку n.

Из решения (8) имеем

$$\frac{e_n}{e_0} = \prod_{1 \leqslant k < \frac{1+\beta}{\alpha}} (1 - \alpha k + \beta) \geqslant \alpha^n \prod_{1 \leqslant k < \frac{1+\beta}{\alpha}} \left(\left[\frac{1+\beta}{\alpha} \right] - k \right) =$$

$$= \alpha^n \left(\left[\frac{1+\beta}{\alpha} \right] - 1 \right)!,$$

где $[x]$ — целая часть числа x, откуда

$$n < \frac{1}{\ln \alpha} \ln \frac{r}{\left(\left[\frac{1+\beta}{\alpha} \right] - 1 \right)!}.$$

[7] «. . . число ошибок, которое существует в типичных программах ко времени завершения кодирования, составляет примерно 4–8 на 100 операторов программы [62, стр. 148].»

Например, пусть $\alpha = 0{,}1$, $\beta = 0{,}01$ и задано число $r = 0{,}01$ (программа после отладки должна содержать не более 1% от первоначального количества ошибок), тогда получим $n \approx 7{,}56$, т. е. что достаточно семи итераций.

На рис. I.4 показано поведение модели при различных входных параметрах.

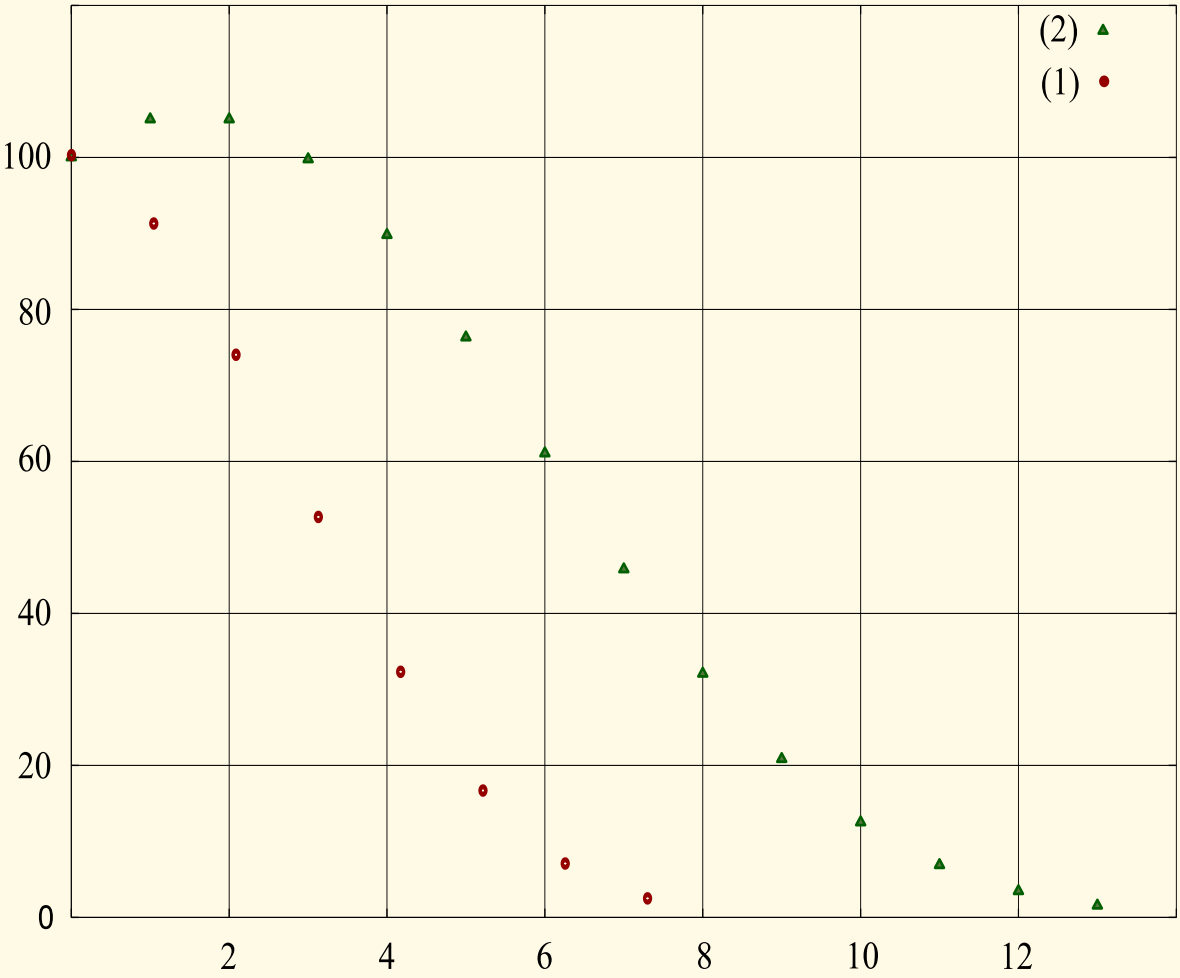


Рис. I.4. *Модель отладки компьютерной программы* (7), *параметры:* $e_0 = 100$; *(1)* $\alpha = 0{,}1$, $\beta = 0{,}01$; *(2)* $\alpha = 0{,}05$, $\beta = 0{,}1$

II. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ

1. Проектирование структуры программы

Для программирования необходима паранойя (в разумных дозах) и твёрдая вера в справедливость законов Мерфи.
Д. Тейлор и др., [58]

Лучший способ в чём-то разобраться до конца — это попробовать научить этому компьютер.
Дональд Эрвин Кнут

Характерная черта компьютерных программ, соответствующих современным математическим моделям реальных явлений или систем некоторых объектов, — *сложность*. Поскольку сложность программного обеспечения обусловлена сложностью реального мира, она неизбежна: с ней можно справиться при помощи различных приёмов, но полностью избавиться от неё нельзя.

В следствие ограниченности человеческих мыслительных способностей — человек может разом охватить программу из десятков, но не из десятков тысяч строк — возникает необходимость разбиения сложной программной системы на всё более и более простые подсистемы (декомпозиция). Такой способ управления сложными системами известен с древности — divide et impera (разделяй и властвуй). Он полезен не только в программировании, но и вообще при решении любых сложных задач, см., например, [66]. Различают два основных вида такой декомпозиции:

- *объектно-ориентированная;*
- *алгоритмическая* (или *структурная*).

При структурной декомпозиции производится разделение *алгоритмов*, где каждый программный блок системы выполняет один из этапов общего процесса. При объектной декомпозиции разбиение производится по *объектам* системы. При разделении по алгоритмам внимание концентрируется на порядке происходящих событий, а при разделении по объектам особое значение придается агентам, на которые направлены или которые сами вызывают какие-то действия. Легко видеть, что эти два подхода, по меньшей мере, неразумно применять одновременно.

Алгоритмическая декомпозиция является основой для структурного проектирования в программировании. В языках, где используется такой подход, основной базовой единицей является *подпрограмма*, а структуру программы в целом можно представить в форме «дерева», в котором одни подпрограммы в процессе работы вызывают другие подпрограммы.

Структурный подход реализован в таких языках высокого уровня, как, например, FORTRAN или COBOL.

На идее объектно-ориентированной декомпозиции основан метод объектно-ориентированого проектирования. В его основе лежит представление о том, что программную систему необходимо проектировать как совокупность *взаимодействующих друг с другом объектов*, рассматривая каждый объект как экземпляр определенного *класса* (типа), причем классы образуют определённую структуру (иерархию). Объектно-ориентированный подход является основой последнего поколения языков высокого уровня, таких как Smalltalk, Object Pascal, C++, Java.

Какой именно подход выбрать в данном конкретном случае зависит, главным образом, от решаемой задачи, а также от имеющихся в наличии программных средств (готовых модулей, библиотек и т. п.). Преимущества объектно-ориентированной декомпозиции проявляются, главным образом, в больших проектах со сложной структурой.

Структурное проектирование можно условно подразделить на *процедурное*, когда основным «строительным блоком» при проектировании является подпрограмма и *модульное*, при котором основное внимание уделяется организации данных, а не алгоритмов, по которым эти данные обрабатываются. Модулем в последнем случае называют множество взаимосвязанных процедур вместе с данными, которые эти процедуры обрабатывают. Эти два направления не исключают, а взаимно дополняют друг друга.

При структурном проектировании декомпозиция достигается с помощью метода пошаговой детализации, который иногда называют *технологией программирования сверху вниз*. Идея метода состоит в том, что программа не пишется сразу на языке высокого уровня, а придумываются некие воображаемые *исполнители*, которые «решают» поставленные задачи. Если исполнителя невозможно или затруднительно сразу реализовать программно, то придумывается другой исполнитель, конкретизирующий действия первого, для второго исполнителя могут оказаться нужными другие исполнители и т. д., пока исполнители самого нижнего уровня не удастся записать на программном языке.

Практически, при таком проектировании удобно записать программу, считая, что подзадачи, на которые она разбита, уже решены. После чего можно для каждой подзадачи писать отдельную программу (такие программы называют *подпрограммами*), в случае, если это возможно. Если же непосредственная программная реализация подзадачи невозможна или по каким-то причинам неудобна, то эта подзадача снова разбивается на другие подзадачи и т.п. Чем сложнее исходная задача, тем больше получится уровней иерархии больше будет подпрограмм.

Практически вначале вместо настоящих подпрограмм для исполнителей, в код вставляются фиктивные части — *заглушки*, которые не делают ничего, кроме, быть может, выдачи сообщения о вызове. После

того, как программист убедится, что общая структура программы верна, подпрограммы-заглушки последовательно заменяются на реально работающие. Для каждой подпрограммы процедура повторяется до тех пор, когда не останется ни одной заглушки. Подробнее см. [53], [54], [55].

1.1. Объектно-ориентированная декомпозиция

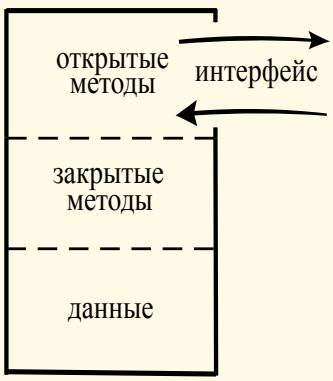
Наиболее естественно, просто и наглядно программная реализация математической модели в общем случае строится при использовании объектно-ориентированного подхода. К тому же, структурная декомпозиция становится затруднительной при программировании больших, сложных моделей. Основными преимуществами объектно-ориентированного метода по сравнению со структурным программированием являются:

— лучшая применимость к моделированию реальных физических объектов и явлений;
— *правильно выполненные* объектно-ориентированные проекты гораздо легче кодировать и сопровождать;
— возможность повторного использования программного кода.

К недостаткам можно отнести необходимость более тщательного проектирования и, как правило, большее время, затрачиваемое на разработку программ.

Базовыми понятиями объектно-ориентированного проектирования и программирования (ООП) являются *объекты* (они являются электронными аналогами моделируемых физических объектов) и *классы*. Объект в ООП реально существует в памяти компьютера на этапе выполнения программы, а класс — это абстракция, которая задаёт общую структуру и поведение сходных объектов. Другими словами, класс — это абстрактное множество объектов, имеющих общую структуру и поведение, а объект представляет собой экземпляр, представителя, имеющий *тип* данного класса. Объект определяется своим *состоянием*, *поведением* и *именем* (идентификатором), позволяющим отличить его от объектов того же класса или других классов. Например, класс — `Homo_Sapiense`, объекты этого класса — `Ivanoff`, `Petroff`.

У объекта можно выделить две важнейшие характеристики: интерфейс и реализацию. *Интерфейс* определяет как объект «общается» с внешней средой (с пользователем, другими объектами, операционной системой). *Реализация* — это внутренние особенности объекта. Интерфейс и реализация объекта должны быть максимально независимы друг от друга в том смысле, что изменение внутренних функций не должно изменять соответствующего интерфейса. Можно условно представлять себе объект как «чёрный ящик» с окном, через которое объект взаимодействует с внешним миром (интерфейс). Это взаимодействие осуществляется через



открытые («видимые» в окне) методы, доступа к другим, закрытым, методам нет — они предназначены исключительно для внутренней реализации объекта («не видны» в окне). Также не должно быть прямого доступа к данным, хотя многие языки программирования допускают нарушение этого принципа.

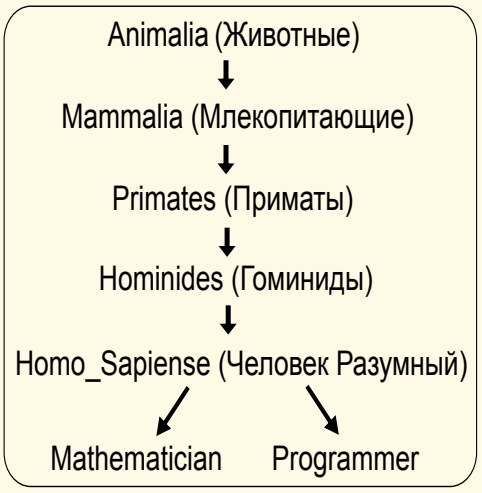
Сам интерфейс организован посредством пересылки сообщений к объекту от «окружающей среды» и от объекта к «внешнему» миру. Основными базовыми принципами ООП являются:

- *инкапсуляция* (encapsulation);
- *наследование* (inheritance);
- *полиморфизм* (polimorphism);
- *пересылка сообщений*.

Инкапсуляция — объединение в классе в единое целое данных (переменных, констант различных типов) и методов (процедур и функций), обрабатывающих эти данные. Данные класса это то, что объект этого класса «знает», а методы — то, что он «умеет». Например, для объектов класса `Homo_Sapiense` данными могут быть: рост, цвет глаз, возраст, знание русского языка и арифметики, а методами — умение читать, писать и считать.

Наследование. Любой класс может быть порождён от другого класса (*базового* или *родительского*). Порождённый класс (*потомок*, *производный класс*) автоматически получает доступ к данным и методам родителя, т. е. наследует его структуру, и, кроме того, имеет возможности переопределять («улучшать») методы и дополняться новыми данными и методами.

Например, классу `Programmer`, наследнику класса `Homo_Sapiense`, можно добавить свойство знания языка программирования C++ и метод — умение работать в IDE (интегрированных средах разработки), а способность считать усовершенствовать до владения калькулятором. Точно так же, порождённый от `Homo_Sapiense` класс `Mathematician` можно наделить «знанием», скажем, интегрального исчисления или топологии и добавить метод — умение интегрировать. Все свойства и методы базового класса `Homo_Sapiense` будут также присущи любому объекту классов `Programmer` и `Mathematician`. Класс-потомок может, в свою очередь, быть базовым для другого класса. Таким образом, совокупность классов, связанных отношениями наследования, может образовывать иерархию или «дерево» классов любой сложности. У каждого класса может быть сколько угодно потомков. Что касается предков, то в некоторых языках программирования (например, C++) их



может быть несколько, а в других (Object Pascal, C#) множественное наследование не допускается.

Полиморфизм даёт возможность решать сходные задачи различными способами. Для изменения метода родительского класса в классе-потомке его можно переопределить. Наследование позволяет различным типам данных совместно использовать один и тот же код, что приводит к уменьшению его размера и повышению функциональности, а полиморфизм перекраивает этот общий код так, чтобы удовлетворить конкретным особенностям отдельных типов данных. Практически полиморфизм часто реализуется через механизм т. н. *виртуальных* функций.

Виртуальные функции позволяют объекту производного класса изменять поведение, определённое на уровне базового класса или обеспечить какие-либо возможности базового класса, которые он не может осуществить из-за того, что нужная для этого информация объявляется на уровне производного класса. Виртуальные методы позволяют объявить класс общего назначения, не требуя для этого знания конкретных особенностей, которые могут быть доступны только в производных классах. Например, поскольку для работы с различными IDE (Integrated Development Environment — интегрированная среда разработки) требуются различные навыки, то соотвествующий метод в `Programmer` можно объявить виртуальным, а в классах-наследниках, которые назовём, к примеру, `Borland_Programmer` и `Microsoft_Programmer`, должным образом его переопределить, чтобы наш программист обрёл способность работать в Borland C++ Builder и Visual C++ соответственно.

Другим важным понятием и принципом ООП является *пересылка сообщений*, которая содержательно выражает основную методологию построения объектно-ориентированных программ. Любой объект может отправлять сообщения другим объектам и принимать сообщения от них. При получении сообщения объекты выполняют определённые действия, причём разные объекты приняв одно и то же сообщение, вообще говоря, могут выполнять разные действия. Другими словами, поведение и реакция, инициируемые сообщением, зависят от объекта-получателя. Практически пересылка сообщений осуществляется вызовом соответствующего метода, обрабатывающего это сообщение. Сообщения это то, при помощи чего объекты «общаются» друг с другом. Например, для объекта класса `Programmer` можно предусмотреть отправку сообщения объекту класса `Mathematician` с запросом на вычисление некоторого интеграла.

Таким образом, можно сформулировать следующее

Определение 1.1. *ООП — метод построения программ в виде множества взаимодействующих посредством передачи сообщений объектов, структура и поведение которых заданы соответствующими классами, и все эти классы принадлежат иерархии классов, выражающей отношение наследования.*

Не трудно заметить аналогию между методами ООП и инженерным проектированием. Действительно, техническая система в самом общем виде является совокупностью некоторых физических объектов и устройств (их электронные аналоги — объекты в ООП) различного типа (класса). Эти устройства обладают (инкапсулируют) некоторыми определёнными характеристиками (данные) и функциональностью (методы) и взаимодействуют между собой посредством каких-либо механических, электрических или иных связей (пересылка сообщений). Кроме этого, такую систему можно модифицировать, изменив свойства некоторых её объектов или добавив к ней новые (наследование и полиморфизм).

1.2. Объектно-ориентированное проектирование

Сейчас не существует единого и универсального способа объектно-ориентированного проектирования. К настоящему времени не разработаны строгие методы классификации и нет правил, позволяющих выделять классы и объекты. Можно, однако, *рекомендовать* следующий порядок действий:

1. Разработку программы следует начать с анализа функционирования моделируемой системы, так как её поведение обычно известно задолго до всех остальных свойств. Затем следует спроектировать объекты и сообщения, которыми они будут обмениваться. *После*[1] определения того, как объекты будут «общаться» друг с другом, можно приступать к проектированию классов.
2. Проектирование иерархии классов. Обработчики всех *общих* сообщений объединяйте в один базовый класс. Например, если имеется объект, получающий сообщения A, B, C и другой, получающий сообщения B, C, D, E, то нужно создать базовый класс, обрабатывающий сообщения B, C и два производных от него класса, один из которых будет содержать метод, обрабатывающий A, а другой — методы, работающие с D и E. Если продолжать этот процесс, пока не будут исчерпаны все объекты, то в результате получится требуемая иерархия классов, в «вершине» которой будет располагаться самый общий класс, от которого наследуют все другие классы. Любой базовый класс должен иметь не менеее двух потомков. Возможности базового класса должны использоваться всеми его производными классами. Все методы, обрабатывающие сообщения объявите от-

[1] Подчеркнём, сначала проектируются объекты и система обмена сообщениями между ними, потом — иерархия классов. Иначе, не зная как объекты взаимодействуют друг с другом, трудно или даже невозможно заранее определить какие методы будут нужны в каждом классе. Следует чётко понимать, что на этапе выполнения программы нет никаких классов, а существуют только объекты, структура и взаимодействие которых определены на основе иерархии классов.

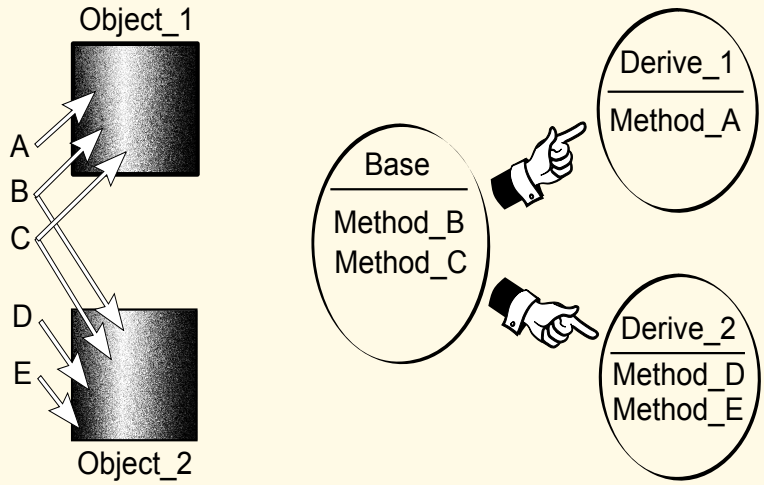


Рис. II.1. *Процесс проектирования классов*

крытыми (public), а все другие, по возможности, должны быть закрытыми (private, protected).

3. Данные добавляются в последнюю очередь. Для поддержки работы с данными на этом этапе также добавляются методы, которые должны быть *закрытыми*, как и сами данные. Обращение к данным должно быть реализовано *только с помощью методов класса*, т. е. интерфейс объекта с его окружением должен полностью определяться его методами, чтобы к его состоянию не было другого доступа извне, кроме как через методы [57], [2].

Может также оказаться полезным метод неформального описания модели, предложенный в книге [57]. В словесном описании модели, программная реализация которой строится, выделяются существительные и глаголы. Существительные рассматриваются как кандидаты для образования классов, а глаголы — кандидаты в операции над классами, т. е. для сообщений.

Не трудно видеть, что принципы структурного проектирования, основанного, естественно, на структурном анализе, никак не соответствуют принципам объектно-ориентированного стиля разработки программ. Ни один из этих подходов не является «лучшим» или «более правильным» во всех ситуациях и задачах, но их не рекомендуется использовать одновременно в одном проекте. Опыт программирования показывает, что применение элементов структурного анализа в процессе объектно-ориентированного проектирования очень часто приводит к значительным трудностям при кодировании и, особенно, сопровождении программного продукта, или даже полному провалу проекта.

Для более детального изучения различных аспектов разработки объектно-ориентированных проектов рекомендуются книги [57] и [56], последняя содержит много примеров программ на различных языках программирования, а также [60], [58].

1.3. ООП в Delphi

Delphi — это мощная, прекрасно организованная среда разработки для визуального программирования, основной компонент которой, Object Pascal, является объектно-ориентированным языком.

Рассмотрим реализацию принципов ООП в среде Delphi. Формат объявления класса

```
type  Tclass_name = class( Tbase_class )
                    // поля, методы и свойства
                    ...
                    end;
```

Здесь Tclass_name — имя класса (имена классов в Delphi принято начинать с буквы T, но это не обязательно), Tbase_class — имя родительского для Tclass_na- me класса, *полями* называют инкапсулированные в классе данные, *методы* — процедуры и функции, *свойства* (*property*) — поля специального вида, регулирующие доступ к обычным полям. Класс может быть объявлен только в интерфейсной части модуля или самом начале области реализации, но не в разделе описаний подпрограмм.

Дерево классов Object Pascal, имея в корне TObject, постепенно разрастается, включая всё новые и новые классы, каждый из которых обладает, кроме своих собственных, всеми возможностями своих предков. Из любого из этих классов программист может вывести свой собственный класс, добавив в него новые свойства и методы или переопределить родительские методы.

Для реализации графического интерфейса с пользователем Delphi использует библиотеку VCL (Visual Component Library — библиотека визуальных компонентов), которая содержит большое количество разнообразных классов, поддерживающих форму (окно) и различные её компоненты (командные кнопки, поля редактирования, меню и т. д.). Во время конструирования формы, по мере добавления на форму компонентов, Delphi автоматически вписывает в текст программы необходимые объявления.

При создании классов доступ к его данным извне ограничивается. Хотя в Object Pascal можно в некоторых случаях напрямую обратиться к полям объекта, это считается отступлением от принципов ООП, согласно которым это обращение должно осуществляться только с помощью методов класса и никак иначе. Для ограничения доступа к данным и методам задаются различные *области видимости*.

В Delphi класс может содержать четыре секции, которые задают следующие области видимости:

1. *public* (общедоступные) — поля и методы этой секции доступны из любого модуля программы;

2. *published* (декларированные) — так же, как *public*, не ограничивает области видимости, используется только при разработке пользовательских компонент VCL Delphi, свойства, объявленные в этой секции, будут доступны в окне Object Inspector;
3. *protected* (защищённые) — поля и методы доступны только методам самого класса и его потомков, независимо от того в каком модуле они находятся;
4. *private* (личные) — минимально возможная область видимости, приватные элементы доступны только потомкам класса, причём только тем, которые размещены в том же модуле.

Внутри каждой секции вначале объявляются поля, а затем методы. Порядок следования секций произволен. По умолчанию, т.е. без явного задания ключевого слова, секция считается *published*.

Класс может содержать два специальных метода: *constructor* (конструктор) и *destructor* (деструктор). В любом конструкторе вначале вызывается конструктор родителя с объявлением inherited, а затем, обычно, выполняются некоторые инициализирующие действия. Конструктор размещает объект в динамической памяти (т. е. до вызова конструктора объект не существует!) и автоматически объявляет указатель Self на выделенную объекту память. Деструктор уничтожает объект, возвращает память обратно в heap. Действия с объектом возможны только после вызова конструктора.

Методы базового класса можно изменять (перекрывать) в потомках, причём это можно сделать как *статически*, так и *динамически*. При статическом изменении объявляется процедура с таким же именем, что и в родительском классе, но выполняющяя другие действия. При динамическом замещении метод родительского класса должен объявляться с директивой *virtual* или *dynamic*, а в классе-потомке — с директивой *override*.

Динамические методы могут вообще не выполнять никаких действий и лишь перекрываться в потомках. Такие методы называют *абстрактными*. Они объявляются с директивой *abstract*. Классы, содержащие такие методы, также называют абстрактными. Такие классы инкапсулируют методы, реализация которых откладывается до объявления соответствующего класса-потомка. Конечно, невозможно создать объект абстрактного класса и вызвать неперекрытый абстрактный метод.

Пример 1.1. *Создадим заготовку для модели шахматной игры. Объекты шахматные фигуры должны взаимодействовать с объектом шахматная доска согласно правилам.*

Объекты шахматные фигуры, очевидно, имеют общие свойства: цвет, координаты положения на доске и др. В соответствии с этим создадим базовый класс

```
// файл figure.pas
unit figure;
 interface
uses board;
//-------------------------------------
// класс Шахматная фигура
//-------------------------------------
Type Tfigure = class
                protected
                ID       : Tid;          // идентификатор фигуры;
                board    : Tboard;       // используется board;
                 function  can_move(     // правильность хода;
                                    to_position : Tposition
                                  ): Boolean; virtual; abstract;
                public
                procedure   move ( to_position : Tposition );
                 constructor Init ( ident      : Tid;
                                    chessBoard : Tboard );
                 destructor  delete;
                end;
```

Этот абстрактный класс содержит поле ID — идентификатор фигуры, включающий её номер (№0 — фигура отсутствует на доске, №1 — король, №2 — ферзь, №3 — ладья и т. д.), координаты на шахматной доске, цвет, а также объект board — «шахматная доска» (объявления класса Tboard и всех типов вынесены в отдельный модуль board.pas, см. ниже). Методы класса

```
implementation
// Ход фигуры в позицию to_position
//-------------------------------------
procedure  Tfigure.move ( to_position : Tposition );
var tempID : Tid;
begin
  if ( can_move( to_position ) )          // если ход правиль-
  then  begin                             // ный, запись фигуры
        tempID := ID;                     // удаляется с "доски"
        tempID.number := 0;               // (№0 — нет фигуры),
        board.entry_fig_board ( tempID ); // фигура записыается
        ID.pos :=  to_position;           // в другой позиции;
        board.entry_fig_board ( ID );
        end;
 board.refresh;
end;
```

```
//-------------------------------------------
// инициализация
//-------------------------------------------
 constructor   Tfigure.Init ( ident : Tid; chessBoard : Tboard );
begin
ID    := ident;
board := chessBoard;
end;
//-------------------------------------------
// фигура удаляется с доски
//-------------------------------------------
 destructor    Tfigure.delete;
begin
ID.number := 0;
 board.entry_fig_board ( ID );
 board.refresh;
end;
end.
// конец файла  figure.pas
```

Класс Tboard предназначен для поддержки графики и интерфейса с шахматными фигурами и, помимо прочего, содержит поле fig_board — массив 8×8 идентификаторов фигур.

```
// файл board.pas
 unit  board;
 interface
uses  Forms,  Graphics,  Types;

Type  Tletter    = ( _a, _b, _c, _d, _e, _f, _g, _h );
Type  Tposition  = record
                     x : Tletter;          // координаты шахматных
                     y : 1..8;             // фигур на доске;
                     end;
type  Tfigcolor  = ( black, white );       // цвет фигур;
Type  Tid        = record                  // идентификатор фигуры;
                   pos    : Tposition;     // положение;
                   color  : Tfigcolor;     // цвет фигуры;
                   number: 0..16;          // № фигуры
                   // 0 — отсутствует, 1 — король, 2 — ферзь,...
                   end;
```

Фигура «делает ход», посылая сообщение объекту board отметить её положение в этом массиве. Поэтому, кроме конструктора и деструктора этот класс имеет метод *move*, который осуществляет «ход» фигуры —

посылает сообщение стереть запись своего текущего положения на доске и записать новые координаты в массив fig_board, предварительно проверив допустимость данного хода.

Правильность хода проверяет логическая функция can_move. Поскольку эта функция индивидуальна для каждой фигуры, она объявлена абстрактной и будет впоследствии перекрываться в соответствующих классах.

Далее приводятся только заголовки класса Tboard и его методов, назначение всех процедур и функций должно быть вполне понятным из комментариев.

```
Type Tboard = class
               protected
               Forma      : TForm;  // форма, куда всё рисуется;
               whiteCell  : Tcolor; // цвета клетки
               blackCell  : Tcolor; // шахматной доски;
               hCell      : word;   // длина стороны клетки;
               // доска как массив для идентефикаторов фигур:
               fig_board  : array[1..8, 1..8] of Tid;
               function     get_color_cell ( i, j: word ): boolean;
               procedure    draw;
               // по позиции фигуры определяет клетку доски:
               function     search_rect ( pos: Tposition ): TRect;
               procedure    show_figures;
               // рисование фигур:
               procedure    draw_queen( pos   : TPosition;
                                        color : Tfigcolor );
               procedure    draw_rook ( pos   : TPosition;
                                        color : Tfigcolor );
               // здесь могут быть другие фигуры...
               public
               // по координатам доски определяет позицию:
               function     search_pos ( X, Y: word ): Tposition;
               // запись ID фигуры в массив fig_board:
               procedure    entry_fig_board ( ID: Tid );
               procedure    refresh;
               constructor  init ( white_cell, black_cell : Tcolor;
                                   h_cell : word; form : TForm );
               end;
```

Из абстрактного класса Tfigure теперь можно выводить классы-потомки для конкретных фигур, в частности определим производный класс Tqueen (ферзь).

Ограничимся листингом лишь для класса Tqueen, для других программный код пишется аналогично.

```
// файл queen.pas
unit queen;
 interface
uses figure, board;
//-------------------------------------
Type Tqueen = class( Tfigure ) // класс Ферзь
                protected
                function can_move(
                                    to_position : Tposition
                                  ): Boolean; override;
                public
                 constructor Init ( ident : Tid; chessBoard : Tboard );
                 destructor delete;
                end;
```

Раздел реализации содержит, кроме конструктора и деструктора, только функцию проверки правильности хода ферзя:

```
implementation
//-------------------------------------
//  может ли ферзь идти в to_position
//-------------------------------------
function Tqueen.can_move( to_position : Tposition ): Boolean;
begin
result := ( ID.pos.x = to_position.x )
            OR ( ID.pos.y = to_position.y )
               OR ( Abs( ID.pos.y - to_position.y ) =
                     Abs( ord( ID.pos.x ) - ord( to_position.x ) ) );
end;
//-------------------------------------
constructor Tqueen.Init ( ident : Tid; chessBoard : Tboard );
begin
inherited Init ( ident, chessBoard );
end;
//-------------------------------------
destructor Tqueen.delete;
begin
inherited delete;
end;
// конец файла queen.pas
```

Теперь, при желании, легко можно продолжить добавлять другие фигуры, снова наследуя от базового класса Tfigure [2].

2. Оптимизация и отладка программы

Отладка кода вдвое сложнее, чем его написание. Так что если вы пишете код настолько умно, насколько можете, то вы по определению недостаточно сообразительны, чтобы его отлаживать.

Брайан Керниган

Всегда пишите код так, будто сопровождать его будет склонный к насилию психопат, который знает, где вы живете.

Мартин Голдинг

2.1. Оптимизация программ по времени выполнения

Для подавляющего большинства программ основная часть времени их исполнения расходуется на выполнение небольших участков кода, который называют *критичным* или *внутренним* циклом. Именно с этих программных блоков нужно начинать оптимизацию (и, в большинстве случаев, ими можно и ограничиться). Большая часть кода, обычно до 80–90%, не нуждается ни в какой оптимизации.

Как правило, этот код действительно является циклом в смысле синтаксиса языков программирования. Если программа содержит несколько вложенных друг в друга циклов, то внутренний — это цикл, принадлежащий всем циклам. Если имеется несколько циклов, не содержащих вложенных циклов, то нужно оценить время их работы, подсчитав количество повторений каждого из них.

Пример. Рассмотрим следующий блок операторов

```
for j := 1 to N do // цикл 1;
begin
    for k := 1 to M do
    begin
    // тело цикла 2;
    end;
    for l := N downto j do
    begin
    // тело цикла 3;
    end;
end;                    // конец цикла 1;
```

Время работы программы, помимо всего прочего, зависит от характеристик компьютера («железа»), на котором программа выполняется. Поэтому заранее точно установить время работы того или иного алгоритма практически невозможно и обычно используют приближённые оценки.

Пусть время одного прогона тела циклов 2 и 3 приближённо равно, соответственно, T_2 и T_3 (если это время не является постоянным, например, в теле цикла имеется условный переход или вызов подпрограммы,

то нужно взять среднее время). Операторы тела цикла 2 выполняются MN раз, а цикла 3 — всего $N-j+1$ раз при каждом прогоне цикла 1. Следовательно, общее количество итераций цикла 3 равно

$$\sum_{j=1}^{N}(N-j+1)=\frac{1}{2}N(N-1).$$

А время работы всего блока

$$MNT_2+\frac{1}{2}N(N+1)T_3.$$

Какой из циклов в данном случае будет критичным зависит от конкретных значений величин M, N, T_2, T_3. В более сложных случаях можно воспользоваться специально предназначенными для этих целей программами — профайлерами (profiler) [63].

После того, как определены критические участки кода, можно приступать к оптимизации. Следующие советы могут помочь повысить быстродействие:

- Никогда не полагайтесь на автоматическую оптимизацию кода компилятором[1].
- По возможности, старайтесь исключить операции с плавающей точкой, так как они выполняются много медленнее, чем манипуляции с целыми величинами.
- Целочисленное умножение (деление) лучше заменить сложением (вычитанием), или, ещё лучше, *сдвигами*. Например, вместо: k := k∗11; можно использовать k := k *shl* 3 + k *shl* 1 + k;, поскольку $11=2^3+2+1$.
- Так как перед каждым вызовом процедуры или функции её параметры помещаются в стек, а затем изымаются оттуда, то поцедуры лучше спроектировать так, чтобы они имели как можно меньше параметров.
- Если есть возможность не выполнять какие-либо расчёты в реальном времени, можно их предварительно подготовить.
- В самых критических случаях можно использовать язык Assembler, программирование на котором более трудоёмко по сравнению с языками высокого уровня, но обеспечивает максимальную скорость выполнения программы.

[1] «Вера в могущество компиляторов в своем корне абсолютно безосновательна. Хороший оптимизирующий компилятор по большому счету может похвастаться лишь своим умением эффективно транслировать грамотно спроектированный код, т. е. если он не сильно ухудшает исходную программу, то уже только за это его разработчикам следует сказать „спасибо“. Изначально „кривой“ код не исправит никакой компилятор, и оптимизирующий — в том числе» [63].

Следует отметить, что самым лучшим способом оптимизации является *выбор более эффективного алгоритма* — пузырьковая сортировка будет всегда идти медленно, несмотря на все программисткие трюки и ухищрения [61].

2.2. Тестирование и отладка программ

Ошибки в программах могут быть трёх видов:

- Синтаксические ошибки, вызванные тем, что программист нарушил правила языка программирования.
- Ошибки периода выполнения программы, когда синтаксически правильная программа работает не верно (или вообще не работает), например, ошибка «деление на 0», или «бесконечный цикл».
- Логические ошибки, связанные с тем, что неправильно запрограммирован алгоритм, или сам алгоритм неверен.

Синтаксические ошибки *автоматически* обнаруживаются при трансляции, причём компилятор выдаёт соответствующее сообщение о том, где произошла ошибка и возможных её причинах.

Ошибки двух других типов более неприятны и коварны, так как они могут проявляться не при всех наборах входных данных, а только при некоторых, или при возникновении каких-то конкретных обстоятельств. Эти ошибки устраняются при помощи отладки и тестирования программ.

В настоящее время не существует теории тестирования и отладки, применение которой гарантировало бы выявление всех возможных ошибок. Отсюда следует, что любая более или менее сложная программа *почти неизбежно* содержит ошибки. Некоторые из них можно устранить посредством отладки, при этом, быть может, внеся в программу новые ошибки. В связи с этим очевидна необходимость написания программного продукта с учётом последующего *сопровождения*.

Отладка и тестирование программы — искусство, где нет строго определённых правил и где реально наиболее ярко проявляется квалификация программиста.

Следующие три простых совета, возможно, помогут значительно облегчить отладку:

- Лучше сначала отладить процедуры, функции, объекты и т. д. по отдельности, а затем уже проверить их взаимодействие в программе в целом.
- При отладке особое внимание нужно уделить наиболее потенциально уязвимым участкам кода, таким, как *циклы* и *ветвления*, *рекурсивные функции*, а также всем программным объектам, связанным с работой с динамической памятью, *указателями*, пользовательским вводом данных и т. п.
- При необходимости можно воспользоваться отладчиком (debugger), который сейчас имется в составе почти любой IDE, и который позволяет пошагово выполнять программу, от оператора к оператору,

расставлять точки прерывания и т. д. Причём при отладке дебаггером на каждом шаге выполнения будет доступна вся информация о текущем состоянии всех элементов программы.

Под *тестированием* понимается выполнение программы с набором таких входных данных, при которых результат работы программы известен заранее. Цель тестирования — определение логических ошибок в программе. Конечно, прохождение набора тестов *не гарантирует* логическую правильность в достаточно сложных программных проектах. С другой стороны, провал любого теста *всегда* означает, что программный код нуждается в исправлении. Логическую правильность программы обычно удаётся доказать только в относительно простых случаях.

При тестировании рекомендуются следующие правила:

- Составление тестов лучше проводить параллельно с разработкой программы.
- Сравнение эталонных и полученных значений предпочтительнее проводить в ходе самого теста.
- Программа должна не только правильно работать при корректных входных данных, но и уметь обрабатывать недопустимые для данной программы входные параметры.
- Все тесты нужно тщательно анализировать. При существенном изменении програмного кода, скорее всего, понадобится модифицировать и набор тестов.

Рассмотрим в качестве простого примера вычисление выражения

$$\frac{e^x - 1}{x}.$$

Тестирование функции

```
function exp_div_x( x: double ): double;
begin
 result := ( exp( x ) - 1 )/x;
end;
```

даст следующие результаты

№ теста	значение x	результат вызова функции
1	1.0	1.71828182845905 E+0000
2	0.0	Invalid floating point operation
3	-1.0	6.32120558828558 E-0001
4	1000.0	Floating point overflow
5	1.0 E-3	1.00050016670834 E+0000
6	1.0 E-1000	Invalid floating point operation

Причина неправильной работы функции в тесте №2 — деление на нуль, которое согласно стандарту формата представления чисел с плавающей точкой IEEE 754 должно приводить к нечисловым значениям

+INF или -INF, но, в зависимости от языка программирования, может также сгенерировать исключение, сообщение об ошибке, остановку выполняемой программы или привести к специальному нечисловому значению NaN (Not-a-Number).

Поскольку

$$\lim_{x\to 0}\frac{e^x-1}{x}=1,$$

то эту логическую ошибку можно исправить так:

```
function exp_div_x( x: double ): double;
begin
    if ( x = 0.0 ) then result := 1.0
    else result := ( exp( x ) - 1 )/x;
end;
```

Сбои в тестах №4 и №6 вызваны выходом за допустимый диапазон изменения величины типа double, который в Delphi составляет

$$5{,}0\times 10^{-324}\ldots 1{,}7\times 10^{308}.$$

Для решения таких проблем можно, например, воспользоваться механизмом *обработки исключительных ситуаций* (exception handling), который применяется в Object Pascal, C++, Java и других языках.

```
function exp_div_x( x: double ): double;
begin
    if ( x = 0.0 ) then result := 1.0
    else begin
            try
            // если при выполнении
             result := ( exp( x ) - 1 )/x;
            // возникла исключительная ситуация, то
            except
                On EOverFlow do
                begin
                // если используется GUI Windows, то
                // можно использовать процедуру
                // ShowMessage('result is too large !');
                // из модуля Dialogs,
                // а для консольных приложений:
                 write('result is too large !');
                 result := 0.0;
                end;
            end;
        end;
end;
```

Теперь в случае вещественного переполнения будет выдано сообщение: «result is too large!» («результат вычисления слишком велик!») и функция вернёт $0{,}0$. Возвращаемое значение сознательно выбрано невозможным для данной функции (она обращается в нуль только при $x \to -\infty$ и этого, конечно, не может произойти при вычислениях), чтобы по этой величине можно было впоследствии в вызывающем функцию блоке соответствующим образом обработать эту ситуацию. Применение программных исключений особенно удобно для глубоко вложенных блоков операторов, но следует учитывать, как следствие, неизбежное снижение скорости работы программы и усложнение её кода.

С приобретением определённого опыта у программиста появится возможность в простых и средней сложности программных блоках обходиться без тестирования или, по крайней мере, свести его к некоторому необходимому минимуму.

2.3. Работа с вещественными числами

Основная трудность при работе с вещественными числами является следствием того факта, что любая переменная в памяти компьютера может принимать только конечное число значений, тогда как даже на конечном отрезке вещественной прямой содержится бесконечно много действительных чисел.

Вещественные числа в компьютере представляются в т. н. *формате с плавающей точкой* — отдельно хранится *мантисса* M, $|M| \leqslant 1$, и *порядок* p. Число вычисляется по формуле

$$Ma^p.$$

Для записи мантиссы всегда используется фиксированное количество цифр, диапазон изменения порядка также ограничен. Поэтому машинное представление вещественных чисел имеет следующие, важные для практического программирования, особенности:

a. в компьютере невозможно представить очень большие и очень малые по абсолютной величине действительные числа;

b. вещественное число, даже и попадающее в допустимый диапазон, может быть записано с некоторой погрешностью.

Из свойства **a** непосредственно следует, что существует величина, называемая *машинным нулём*, т. е. такое число ε, что в компьютерных расчётах для всех чисел x, таких что $0 < x < \varepsilon$, выполняется $1.0 + x = 1.0$. Другими словами, все вещественные числа, меньшие ε, компьютер будет «воспринимать» как нуль. Величина машинного нуля зависит также от типа x.

Из свойства **b** вытекает неизбежность ошибок округления, что можно показать на простом примере:

```
program real_error;
{$APPTYPE CONSOLE}
uses SysUtils;
```

```
const   x = 17.0;
begin
 writeln (' ERROR = ', x - sqrt(x)*sqrt(x) );
 readln;
end.
```

Кроме этого, нужно всегда помнить, что погрешность при суммировании чисел складывается из погрешностей слагаемых и погрешности выполнения операции сложения. Если вначале складывать большие по модулю числа, то можно получить неправильный результат.

Рассмотрим пример суммирования первых N членов ряда, задающего ζ-функцию от 2

$$\zeta(2) = \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}.$$

При суммировании в различном порядке можно получить различные результаты:

```
program   floatPrj_1;
{$APPTYPE CONSOLE}
uses   SysUtils;

const  N                = 100000;
       dzeta2           = PI*PI/6;     // "точная"  dzeta(2);
var    k                 : integer;
       S_to, S_downto    :  real;

begin
S_to := 0;
    for   k := 1 to N do                  // сумма по возрастанию k;
    S_to := S_to + 1/k/k;
S_downto := 1/N/N;
    for   k := N - 1 downto 1 do          // сумма по убыванию k;
    S_downto := S_downto + 1/k/k;
 writeln ( 'dzeta(2)   = ',  dzeta2     );
 writeln ( 'S_downto   = ',  S_downto );
 writeln ( 'S_to       = ',  S_to       );
 readln;
end.
```

III. ПРИМЕРЫ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ

1. Физика

Когда вы знаете, о чем идет речь, знаете, что одни символы означают силы, другие - массы, инерцию и т. д., вы можете обратиться за помощью к здравому смыслу, к интуиции. Вы видели разные вещи и более или менее знаете, как будут происходить разные явления. Несчастный математик переводит все это на язык уравнений, и, поскольку символы для него ничего не означают, у него лишь один компас — математическая строгость и тщательность в доказательствах.

Ричард Фейнман. Характер физических законов

Практика рождается из тесного соединения физики и математики.

Роджер Бэкон

1.1. Виброгаситель

При работе многих технических устройств возникают крайне нежелательные, а иногда и чрезвычайно опасные[1], вибрации.

Вибрация может возникнуть, например, из-за движения неуравновешенных масс двигателя, неточно изготовленных деталей, резонансных эффектов, крутильных колебаний и прочего.

Одним из распространенных методов борьбы с вибрацией является *виброгашение*. Суть метода заключается в присоединении к защищаемому объекту дополнительных систем, реакции которых уменьшают вибрации самого объекта

Рассмотрим упрощенную модель виброгасителя. Пусть на тело массы m_1, действует периодическая сила $f = A\sin\omega t$, вызывающая вибрации; константы ω, A заданы. К вибрирующему телу на пружине жесткости k присоединено тело массы m_2. На основе данной модели покажем, что по известным параметрам A, m_1, ω, можно так подобрать величины всего лишь двух параметров k и m_2, что вибрации будут погашены. Трением пренебрегаем.

[1] Пример: авария на Саяно-Шушенской ГЭС 17 августа 2009 года. Из-за многократного превышения амплитуды вибрации подшипника турбины разрушилась и была вырвана из бетонного гнезда ее крышка весом в полторы тысячи тонн, разрушен машинный зал, уничтожено три гидроагрегата ГЭС и повреждены остальные семь, погибли 75 человек.

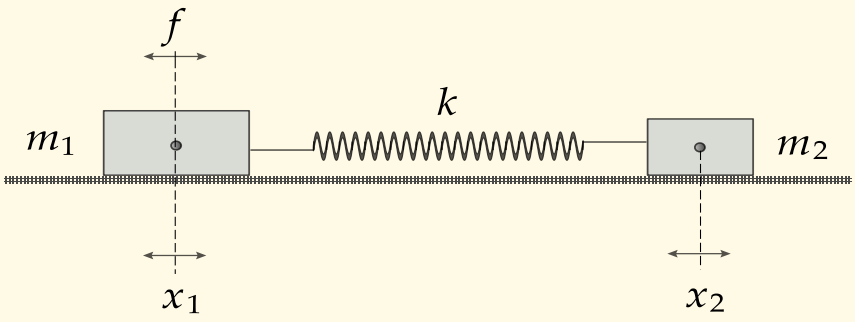


Рис. III.1. *Виброгаситель*

По закону Гука сила упругости F пропорциональна удлинению (сжатию) x пружины:

$$F = -kx,$$

где $k = \text{const}$ — жёсткость пружины. Поэтому, пользуясь вторым законом Ньютона, модель можно описать системой дифференциальных уравнений

$$m_1\ddot{x}_1 + m_2\ddot{x}_2 = A\sin\omega t, \tag{1}$$
$$m_2\ddot{x}_2 + k(x_2 - x_1) = 0. \tag{2}$$

Требуется найти такое решение, при котором $x_1(t) \equiv 0$. Дважды интегрируя первое уравнение, имеем

$$m_1x_1 + m_2x_2 = -\frac{A}{\omega^2}\sin\omega t, \tag{3}$$

отсюда

$$x_2 = -\frac{A}{m_2\omega^2}\sin\omega t.$$

Подставляя это значение во второе уравнение системы вместе с $x_1 = 0$, получим

$$-1 + \frac{k}{\omega^2 m_2} = 0.$$

Таким образом, для подавления вибраций величины k, m_2 нужно выбрать так, чтобы выполнялось соотношение

$$\omega = \sqrt{\frac{k}{m_2}}. \tag{4}$$

Вопрос III.1. Найти аналитическое решение системы (1)–(2) и графически проиллюстрировать решения, получающиеся при выполнении соотношения (4) и при отклонении от него величины ω.

1.2. Реалистичное освещение. Рейтрессинг

Рассмотрим модель реалистичного освещения трёхмерной сцены, основанную на *трассировке* лучей света. Прежде чем сформулировать задачу приведём все необходимые определения, относящиеся к трёхмерной компьютерной графике и трассировке лучей. Поскольку русская

терминология в этой области в настоящее время недостаточно установилась, все названия ниже будут сопровождаться своими английскими аналогами.

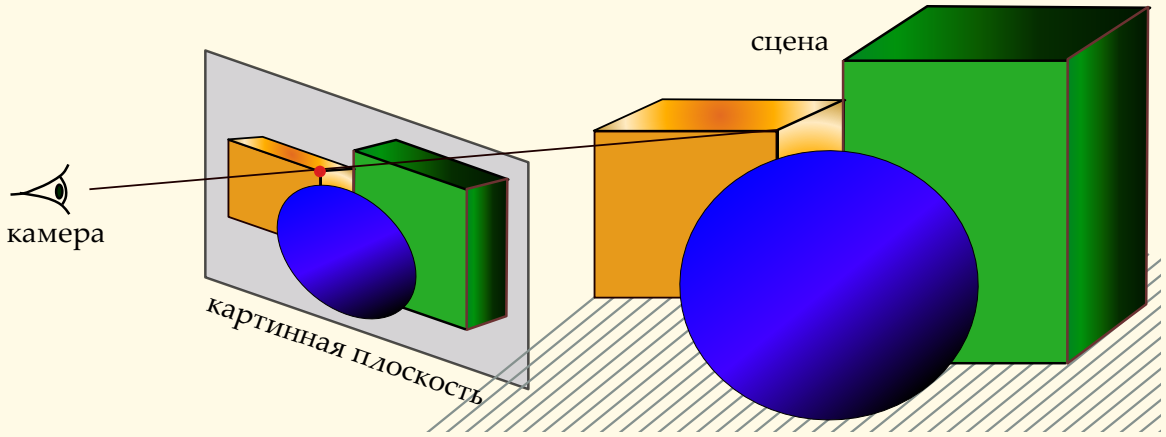


Рис. III.2. *Камера, картинная плоскость и сцена*

Все объекты в совокупности, которые будут отрисованы программой называются частями *сцены* или *мира*. В компьютерной графике точка, из которой осуществляется наблюдение называется, *камерой* (camera). По аналогии с фотокамерой, на плёнку которой проецируется и записывается сцена, в графике мы имеем *картинную плоскость* (view window), на которую проецируется трёхмерная сцена и затем отрисовывается.

Каждый пиксел (pixel — точка растрового изображения) полученной картинки является следствием программной имитации светового луча, который попадает на картинную плоскость по пути в камеру. Задача получения изображения сцены заключается, таким образом, в определении цвета каждой точки картинной плоскости.

Для решения этой задачи в компьютерной графике применяются самые различные методы. Одним из таких методов, потенциально позволяющий добиться максимально возможного *реализма изображения*, является метод *трассировки лучей* света — *рейтрессинг* (ray tracing). Рейтрессинг называется так[2], потому что при использовании этого метода отслеживается путь, которым лучи света распостраняются на сцене — *трассируют* сцену. Целью трассировки является определение цвета каждого луча, который попадает на картинную плоскость после всех отражений и преломлений от объектов сцены, перед тем тем как он достигнет камеры.

Если отслеживать все лучи с исходной точки источника света (light source) на всём протяжении пути до камеры, то такой наиболее точный способ окажется слишком трудоёмким из-за полномасштабных численных расчётов. К тому же, при этом многие лучи, исходящие от осветителя, очевидно, вообще не попадут в камеру. Поэтому вместо трассировки

[2] Tracing — прослеживание, запись регистрирующего прибора.

от источника света, обычно применяют метод *обратной трассировки* — лучи трассируют в обратном порядке, начиная от камеры. Таким образом, метод обратной трассировки делает то же самое, что и оригинальный способ, за исключением того, что понапрасну не тратятся усилия на лучи, которые никогда не достигают камеры.

Определение цвета объекта в точке пересечения луча с его поверхностью зависит от выбранной *модели освещённости*. Опишем одну из самых распространённых — *модель Фонга* (Phong). В этой модели общая интенсивность (которая определяется *энергией* световой волны, обычно принимается, что величина интенсивности меняется от 0 до 1) освещения складывается из следующих трёх компонент:

- интенсивности *рассеянного* (ambient) освещения;
- интенсивности *диффузного* (diffuse) освещения;
- интенсивности *зеркального* (specular) освещения.

Интенсивность рассеянного (фонового) освещения I_a обусловлена множественными отражениями света от всех объектов сцены, она считается постоянной в любой точке пространства

$$I_a = k_a I_p,$$

где I_p — интенсивность источника света, коэффициент $k_a \in [0; 1]$.

Интенсивность диффузного освещения I_d можно рассчитать на основе *закона Ламберта*:

$$I_d = k_d I_p \cos\Theta,$$

где Θ — угол падения луча на поверхность, k_d — коэффициент диффузного отражения, $k_d \in [0; 1]$, определяет меру «шершавости» или «зернистости» поверхности объекта.

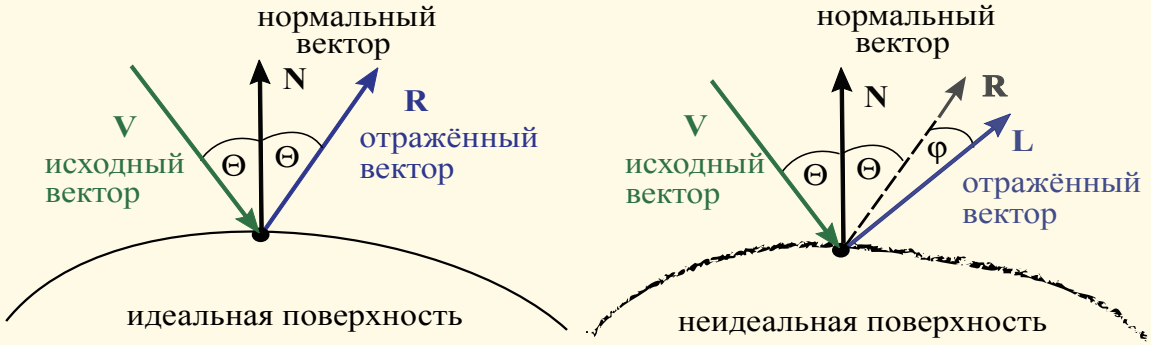


Рис. III.3. *Зеркальное отражение*

Интенсивность I_s зеркально отражённого от поверхности света вычисляется в зависимости от степени отклонения от направления отражённого луча в идеальном случае (зеркало).

$$I_s = k_s I_p \cos^n \varphi,$$

где k_s — коэффициент зеркальности $k_s \in [0; 1]$, n определяет степень «зеркальности» поверхности (для зеркала $n \to \infty$).

Общая интенсивность света I по Фонгу в каждой точке при наличии m источников света определяется соотношением

$$I = I_a + \sum_{k=1}^{m} \left(I_{d,k} + I_{s,k} \right).$$

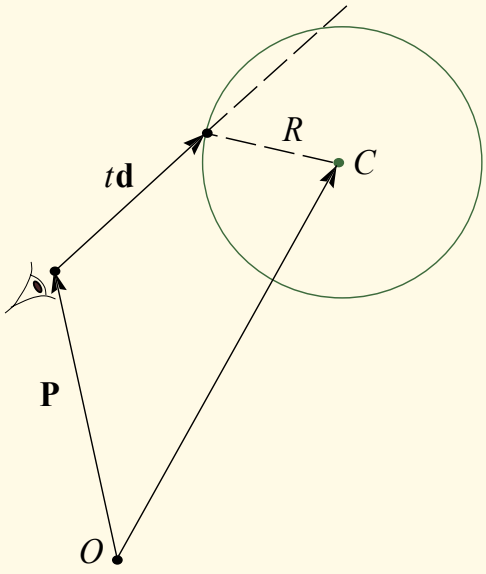
Итак, *метод трассировки лучей состоит в следующем*: для каждого пиксела на картинной плоскости, определяется луч, проходящий от камеры к этой точке. Этот луч прослеживается по всей сцене при всех его отражениях от различных объектов и преломлениях в соответствии с законами физики. Окончательный цвет луча (и, следовательно, соответствующего пиксела картинной плоскости) определяется цветами объектов, на которые падает луч при прохождении через сцену, и цветом источников света.

Теперь сформулируем задачу

> *Построить математическую и компьютерную модели освещения пространственной трёхмерной сцены на основе метода трассировки лучей света. Цвет объектов определять на основе модели Фонга. Ограничиться одним источником света и только полностью непрозрачными объектами-сферами. Ослабление интенсивности света с расстоянием не учитывать.*

Перечислим упрощения, которые будут присутствовать в модели. Поскольку все объекты сцены полностью непрозрачны, то преломление света в данной модели не учитывается, учитывается только отражение.

Закон отражения света: отражённый луч лежит в плоскости падения и угол отражения равен углу падения, или, обозначая $\mathbf{v}$, $\mathbf{n}$, $\mathbf{r}$ направления соответственно падающего на поверхность луча, нормали к поверхности и отражённого луча (все векторы нормированы), получим соотношение



$$\mathbf{r} = \mathbf{v} - 2(\mathbf{n}, \mathbf{v})\mathbf{r}, \qquad (5)$$

справедливость которого легко проверить подставив выражение для $\mathbf{r}$ в формулу, определяющую закон отражения света: $(\mathbf{n}, \mathbf{r}) = -(\mathbf{n}, \mathbf{v})$. Для того, чтобы определить факт пересечения луча со сферой радиуса R, обозначим $\mathbf{d}$ направление луча из камеры на точку пересечения, вектор $\mathbf{d}$ выберем нормированным, то есть $|\mathbf{d}| = 1$. Тогда из уравнения

$$\left| -\overrightarrow{OC} + \mathbf{P} + t\mathbf{d} \right| = R,$$

обозначив $\mathbf{V} = \mathbf{P} - \overrightarrow{OC}$, получим

$$(\mathbf{V} + t\mathbf{d}, \mathbf{V} + t\mathbf{d}) = R^2,$$

откуда находим

$$t_{1,2} = \frac{-(\mathbf{V}, \mathbf{d}) \pm \sqrt{(\mathbf{V}, \mathbf{d})^2 - \mathbf{d}^2(\mathbf{V}^2 - R^2)}}{\mathbf{d}^2}. \qquad (6)$$

Отсюда следует, что при отрицательном значении величины, стоящей в формуле (6) под знаком радикала, луч не пересекает сферу, в противном случае, беря меньший *положительный* корень, получим радиус-вектор точки пересечения: $\mathbf{P} + t\mathbf{d}$.

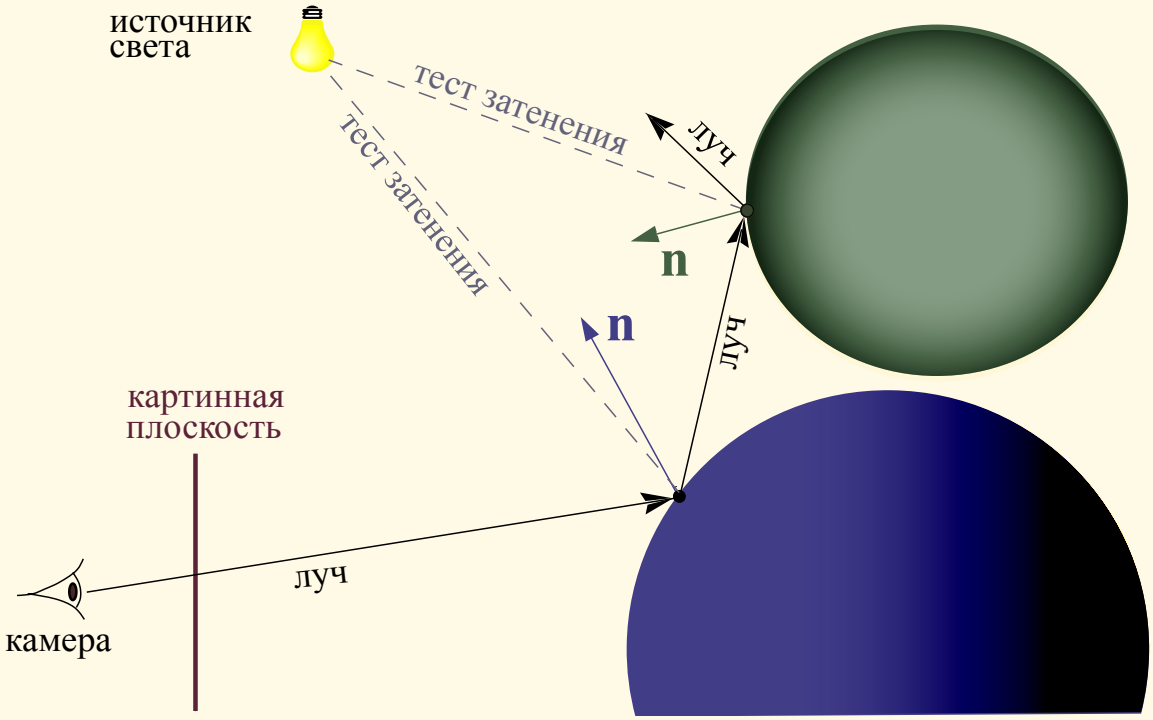


Рис. III.4. *Алгоритм трассировки*

Таким образом, *алгоритм модели* таков. Луч «выстреливается» в заданном направлении, чтобы определить имеется ли там что-нибудь. По формуле (6) определяются точки пересечения луча со всеми сферами сцены. После того, как получим все точки пересечения, выбираем из них ближайшую и вычисляем освещённость объекта в данной точке. Для этого нужно:

- выполнить *тест затенения*, определяющий освещает источник света точку пересечения или нет (см. рис. III.4);
- найти вектор нормали к поверхности объекта в точке пересечения. Он определяет диффузный компонент освещения, а также направление отражённого луча;
- найти отражённый луч, который определяет зеркальный компонент освещения и, конечно, цвет отражённого света (если объект отражает свет).

После чего цвет точки определяется на основе модели Фонга. После отражения луча эта процедура повторяется и т. д.

Выше была рассмотрена сильно упрощенная модель трассировки лучей, создание *фотореалистических* изображений (особенно — в реальном времени, что необходимо, в частности, для компьютерных игр) требует значительных вычислительных ресурсов[3]. Прекрасным примером реализации идей рейтрессинга является программа POV-Ray (Persistence of Vision Raytracer). Эта программа свободно распространяется, причём с исходными кодами (на языке C++), снабжена отличной справочной документацией. На сайте разработчиков [64] можно найти множество материалов по рейтрессингу и соответствующему программному обеспечению.

Вопрос III.2. На языке высокого уровня реализовать описанный выше алгоритм трассировки лучей. Цвет объектов определять на основе модели Фонга. Ограничиться одним источником света и только объектами-сферами, которые считать полностью непрозрачными. Предусмотреть возможность расширения в дальнейшем набора объектов сцены, т. е. добавления кубов, торов, эллипсоидов и пр. Ослабление интенсивности света с расстоянием не учитывать.

Входные параметры модели: количество сфер, их радиусы, цвета, коэффициенты диффузии и отражения, координаты источника света, его цвет, координаты положения камеры.

Вопрос III.3. Из общего принципа Ферма

> *Из всех возможных путей «свет выбирает» тот путь, на который требуется наименьшее время*

получить следующие законы геометрической оптики:

1. Угол падения луча света равен углу отражения (углы откладываются от нормали к поверхности падения).
2. Закон преломления Снелла (Снеллиуса, Snel van Royen):
$$\frac{\sin\alpha_1}{v_1} = \frac{\sin\alpha_2}{v_2},$$
где v_1, v_2 — скорость распространения света в соответствующих средах, α_1, α_2 — углы падения и преломления (см. рис. III.5).

[3] Генеральный менеджер по продуктам компании Nvidia GeForce Дрю Хенри (Drew Henry): «Трассировка лучей (Ray Tracing) — очень серьезная вычислительная проблема. Производительность, необходимая для фотореалистичной имитации мира, во столько раз выше доступных сегодня (2012 г, В. З.) решений, что только этот вызов сам по себе еще не один год будет стимулировать развитие индустрии GPU.» GPU — Graphics Processing Unit, графический процессор компьютера или игровой приставки. Тим Суини (Tim Sweeney) из компании Epic Games считает, что для обсчёта эффектов, заметных на разрешении человеческого глаза, производительность GPU должна вырасти в 2000 раз (по состоянию на 2012 г).

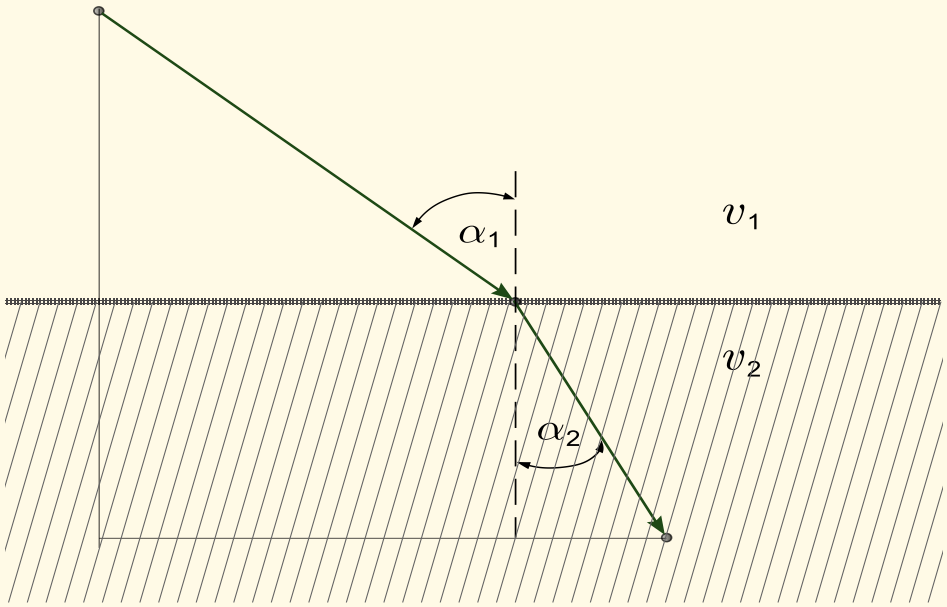
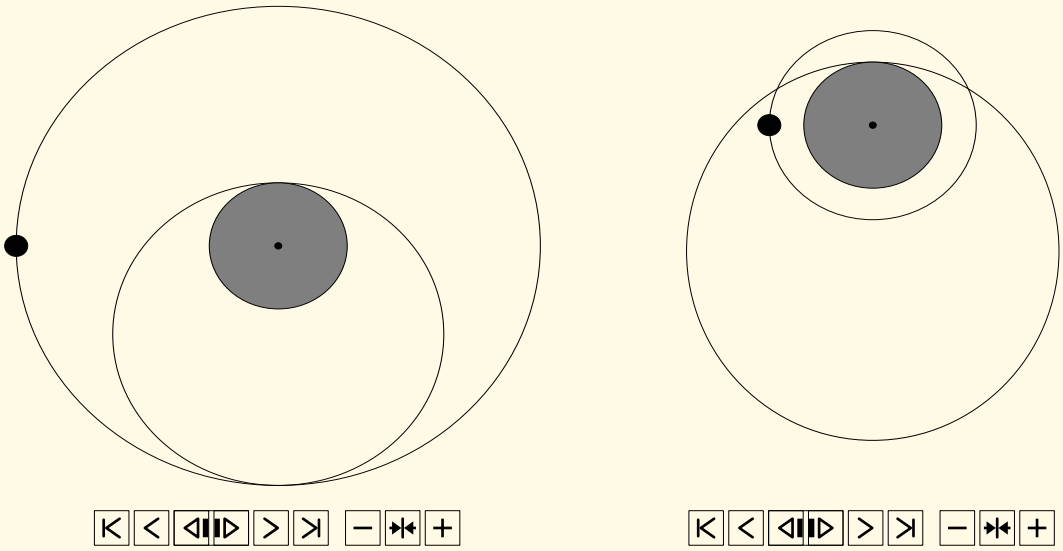


Рис. III.5. *К закону Снелла*

1.3. Возвращение спутника с орбиты

Космический корабль (спутник) движется по круговой орбите Земли радиуса r со скоростью v_0. Для перехода на траекторию приземления кораблю сообщают дополнительную скорость Δv, включая на короткое время тормозные двигатели. Нужно рассмотреть два способа приземления:

1. дополнительная скорость сообщается в направлении, противоположном орбитальной скорости;
2. дополнительная скорость сообщается вертикально вниз, по направлению к центру Земли. Определить дополнительную скорость, которую необходимо сообщить кораблю в обоих случаях для схода с орбиты и приземления [65].



В нашей модели космический корабль и даже Земля будут фигурировать в виде материальных точек, что вполне достаточно для ответа на поставленный вопрос и даст возможность применить *законы Кеплера* и закон сохранения энергии в наиболее простой форме.

При сообщении кораблю дополнительной скорости Δv его орбита с круговой переходит на эллиптическую, один из фокусов которой, в соответствии с *первым законом Кеплера*, находится в центре Земли. Очевидно, что при любом способе торможения величина скорости будет наименьшей, если эллипс только коснётся границы плотных слоёв атмосферы.

Имеются два способа[4] приземления: когда кораблю придаётся скорость Δv противоположно v_0 и когда Δv направляется к центру Земли, как показано на анимациях. Для определения дополнительной величины скорости Δv в каждом из случаев используем закон сохранения энергии, момента импульса и *второй закон Кеплера*[5], в соответствии с которыми при движении по орбите после придания дополнительной скорости секторная скорость остаётся постоянной.

Для первого способа имеем

$$\frac{1}{2}mv^2 - \frac{mgR^2}{r} = \frac{1}{2}mu^2 - mgR, \tag{7}$$

$$rv = Ru, \tag{8}$$

где $v = v_0 - \Delta v$ — скорость в апогее (в самой нижней точке эллиптической орбиты), R — радиус Земли, u — скорость в точке приземления. Из уравнений (7), (8) имеем

$$v = \sqrt{\frac{2gR^2}{r}}\frac{1}{\sqrt{1 + r/R}}. \tag{9}$$

[4]Вообще говоря, существует ещё способ торможения с использованием силы сопротивления атмосферы — *аэродинамическое торможение*. Поскольку аэродинамическое торможение не требует затрат топлива, этот способ выгодно применять при спуске на планету, обладающую достаточно плотной атмосферой. Однако, свободный спуск с орбиты за счет торможения в разреженной атмосфере вызывает трудности при прогнозировании времени и места приземления. Задача еще больше осложняется тем, что нужно учитывать ограничение при перегрузках, допустимых для экипажа и приборов, а также конструкции спускаемого аппарата.

[5]Второй закон Кеплера (закон площадей): Каждая планета движется в плоскости, проходящей через центр Солнца, так что за равные промежутки времени радиус-вектор, соединяющий Солнце и планету, описывает равные площади.

Учитывая, что $\sqrt{gR^2/r}$ — скорость корабля на круговой орбите v_0, получаем

$$\Delta v = v_0 \left(1 - \sqrt{\frac{2}{1 + r/R}}\right). \tag{10}$$

Для второго способа при сообщении кораблю дополнительной скорости Δv, направленной к центру Земли, его секторная скорость не меняется. Для точки приземления это условие даёт

$$rv_0 = Ru. \tag{11}$$

Отсюда при учёте закона сохранения

$$\frac{1}{2}\, m \left(v_0^2 + \Delta v^2\right) - \frac{mgR^2}{r} = \frac{1}{2}\, mu^2 - mgR, \tag{12}$$

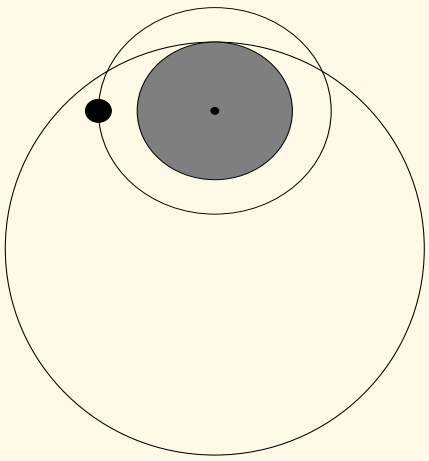
аналогично рассмотренному случаю, после несложных преобразований получаем

$$\Delta v^2 = v_0^2 \left(\frac{r^2}{R^2} - 2\,\frac{r}{R} + 1\right), \tag{13}$$

откуда находим

$$\Delta v = v_0 \Big(\frac{r}{R} - 1\Big). \tag{14}$$

Заметим также, что уравнение (13) имеет и ещё одно решение, совпадающее с найденным по абсолютной величине и противоположное по знаку. Простой анализ показывает, что этому решению соответствует такой способ схода с орбиты, при котором дополнительная скорость будет направлена не к центру Земли, а в *противоположном* направлении, т. е. вверх. Так что в этом случае при получении этой дополнительной скорости корабль вначале станет удаляться от Земли по эллиптической орбите, а затем, двигаясь по ней всё равно попадёт в прежнюю точку приземления. Таким образом, построенная нами простая модель, кроме ответа на поставленный вопрос, «подсказывает» и ещё один способ для схода с орбиты.



Получение на основе модели новых или «неочевидных» сведений является характерным признаком полезной модели.

1.4. Задачи

III.1. Морская волна. Пусть функция $u(x,t)$ задаёт профиль волны в момент времени t. Тогда уравнение распространения волны имеет вид

$$\frac{\partial u}{\partial t} - c(u)\frac{\partial u}{\partial x} = 0, \qquad u(x,0) = u_0(x).$$

Функции $u_0(x)$ (возмущение в начальный момент времени) и $c(u)$ — «скорость распространения» возмущения волны, заданы. Сделать гра-

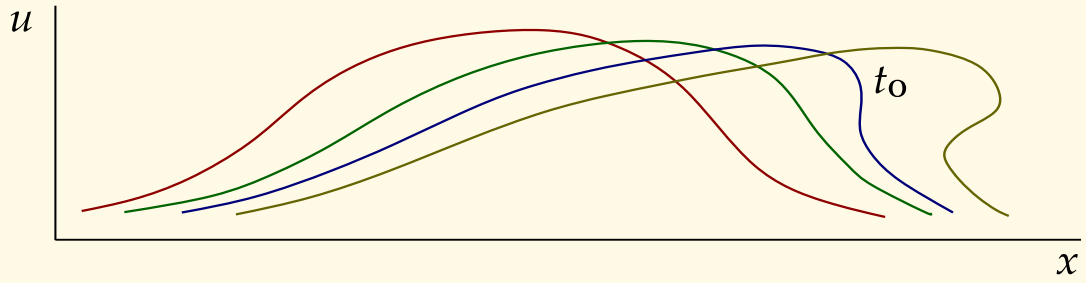


Рис. III.6. *Опрокидывающаяся волна*

фическую анимированную иллюстрацию модели, подобрав подходящий масштаб времени τ. Входные параметры: τ, $u_0(x)$, $c(u)$ (подобрать 2–3 варианта функций, $c'(u) > 0$).

Для каждого варианта определять момент t_O опрокидывания волны. Литература: [24, стр. 189], [21, стр. 23–31].

III.2. Три точечные массы. Три точечные массы m закреплены на струне так, что расстояние между ними и от крайних масс до закреплённых концов струны равны L. В начальный момент времени все массы находятся в состоянии равновесия, а средней массе сообщается скорость v_0. Пусть $x_1(t)$, $x_2(t)$, $x_3(t)$ — отклонения масс от положения равновесия, тогда движение системы описывается системой уравнений

$$\begin{aligned} \ddot{x}_1 + k(2x_1 - x_2) &= 0; \\ \ddot{x}_2 + k(2x_2 - x_1 - x_3) &= 0; \\ x_3 &= x_1, \end{aligned}$$

где $k = P/(mL)$, P — константа, зависящая от материала струны. Начальные условия: $x_i = 0$, $\dot{x}_1 = 0$, $\dot{x}_3 = 0$, $\dot{x}_2 = v_0$. Сделать графическую

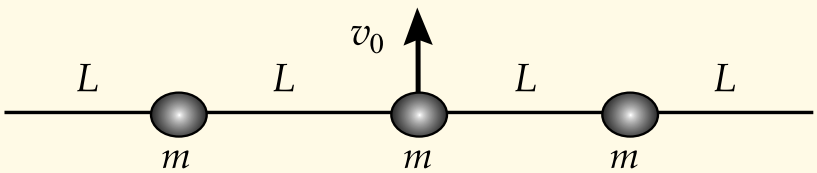
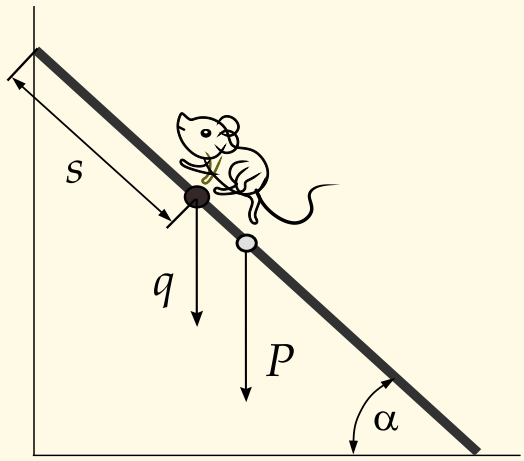


Рис. III.7. *Три точечные массы на струне*

анимированную иллюстрацию модели, подобрав подходящий масштаб времени τ. Входные параметры: m, v_0, L, P, τ. Литература: [26].

III.3. Задача И. Е. Жуковского. Балка длины $2L$, вес которой сосредоточен в середине и равен P, упирается концами в идеально гладкие стену и пол, образуя с последним угол α. По балке бегает животное, вес которого равен q. Найти, как должно двигаться животное, чтобы балка не падала. Ответ даётся уравнением



$$\ddot{s} = -\frac{gs}{2L\sin\alpha} + \frac{(P+2q)g}{2q\sin\alpha},$$
$$s(0) = s_0, \dot{s}(0) = v_0,$$

где g — ускорение свободного падения. Сделать графическую анимированную иллюстрацию модели, подобрав подходящий масштаб времени τ. Входные параметры: L, P, q, τ, v_0, α. Литература: [22, стр. 152–153].

III.4. Маятник на наклонной плоскости. Точка подвеса маятника, состоящего из материальной точки, висящей на нерастяжимой нити длины L, движется по заданному закону $S = S(t)$ по наклонной плоскости, образующей угол α с горизонталью. Трением пренебрегаем.

Движение описывается дифференциальным уравнением

$$\ddot{x}L^2 - \ddot{S}L\cos(x-\alpha) + gL\sin x = 0,$$
$$x(0) = x_0, \dot{x}(0) = 0,$$

где g — ускорение свободного падения.

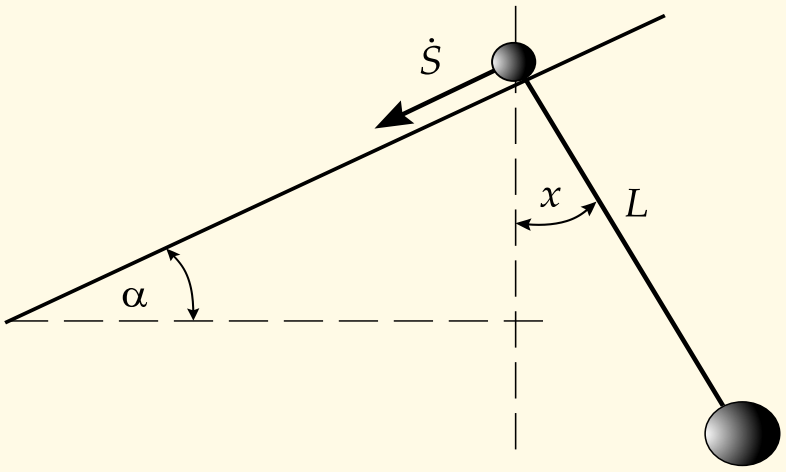


Рис. III.8. *Маятник на наклонной плоскости*

Построить графическую анимированную иллюстрацию модели, подобрав подходящий масштаб времени τ. Входные параметры модели: L, τ, $S(t)$, α, x_0.

Рассмотреть случаи: $S(t) = t^2$, $S(t) = t^3$. Литература: [22].

2. Биология и физиология

. . . во многих случаях очень трудно получить действительно удовлетворительное количественное определение. Меньше всего затруднений в этом плане возникает в таких точных науках, как физика и химия, больше всего — в области искусства и этики. Биология и медицина находятся где-то посредине . . .

Н. Бейли [69]

Иногда с виду очень простые уравнения приводят к самым неожиданным результатам и к новому пониманию сущности биологических явлений.

Дж. Торнли [68]

Биологические исследования в настоящее время часто приводят к необходимости *количественного описания* и *предсказания поведения* биологических объектов, то есть к построению и исследованию математических и компьютерных моделей.

2.1. Модель эпидемии SIR

История человечества неразрывно связана с чередой эпидемий[1] и пандемий[2], которые часто влияли на ход мировой истории, оказывая воздействие на экономику, психологию, культуру, религию и даже генетический состав населения [71], [72], [73]. Борьба с эпидемиями, начиная, по крайней мере, с Гиппократа (ок. 460–377 гг. до н. э.), в числе работ которого были и «Семь книг об эпидемиях», и до настоящего времени, привела к множеству блестящих побед и сокрушительных провалов. Ценой научных и административных поражений в этой борьбе неоднократно были миллионы жизней. Например, пандемия бубонной чумы, вошедшая в историю как «Чёрная смерть», в середине XIV в. привела к гибели трети населения Европы (более 30 миллионов человек), испанский грипп,

[1]Эпидемия (греч. επιδημια — повальная болезнь) — широкое распространение инфекционных болезней среди людей, значительно превышающее обычно регистрируемый уровень заболеваемости.

[2]Пандемия (греч. πανδημια — весь народ) — эпидемия, распространенная на территории нескольких стран или континентов.

«Испанка», в 1918–1919 гг. унес жизни 40–50 миллионов человек, азиатский грипп 1957–1958 гг. убил около 2 000 000, гонконгский грипп 1968–1969 гг. — около 1 000 000 человек.

Модель, называемая SIR, была предложена У. Кермаком и А. Маккендриком (W. Kermack, A. McKendrick) в 1927 г. Она подходит для описания эпидемий, типа гриппа/ОРВИ (Острых Респираторных Вирусных Инфекций), кори, краснухи, компьютерных вирусов и т. п. Популяцию в данной модели разбивают на три группы:

- $S = S(t)$ — susceptible, чувствительные, количество неинфицированных к моменту времени t;
- $I = I(t)$ — infected, инфицированные.
- $R = R(t)$ — removed (удаленные) — изолированные от остальных членов популяции, или — recovered, излечившиеся (имунные[3]) от болезни.

Предполагается, что

a. В модели не принимается во внимание демографические изменения: рождение и смерть, по причинам не связанным с рассматриваемым заболеванием, миграции и т. д. Таким образом, размер популяции считается постоянным на временном отрезке моделирования

$$S(t) + I(t) + R(t) = \text{const} \tag{15}$$

b. Все индивидумы популяции имеют одинаковую вероятность α заражения при контакте с инфицированными. Болезнь передаётся только при общении с инфицированным. Контакт с заболевшим вызывает заражение, если контактирующий не имеет приобретенного иммунитета к болезни. Врождённый иммунитет отсутствует. Инкубационный период заболевания пренебрежимо мал, т. е. заболевание в данной модели происходит мгновенно.

c. Доля выздоровевших β после болезни и изолированных больных постоянна. Иммунитет всегда приобретается в результате болезни. Заболевание никогда не приводит к смерти.

Из этих условий следует

$$\dot{S} = -\alpha SI, \tag{16}$$

$$\dot{I} = \alpha SI - \beta I, \tag{17}$$

$$\dot{R} = \beta I. \tag{18}$$

$$S(0) = S_0, I(0) = I_0, R(0) = R_0, \qquad 0 \leqslant \alpha, \beta \leqslant 1. \tag{19}$$

Подробнее. Уравнение (16) следует из предположений **a** и **b**: количество контактирующих особей из групп в составе S и I членов равно величине SI, умноженной на вероятность контакта/заболевания α, и

[3]Иммунитет (лат. immunitas — освобождение, избавление) — невосприимчивость, сопротивляемость организма к инфекциям.

пропорционально скорости $\dot{S}$ изменения численности; количество S восприимчивых к инфекции индивидуумов убывает со временем, поэтому в правой части (16) стоит знак минус. Соотношение (17) вытекает из предположений **b** и **c**: скорость заражения прямо противоположна скорости $\dot{S}$ без скорости $\dot{R}$ образования не подверженных болезни (по причине обретения иммунитета или изоляции), т. е. интенсивности убывания группы *восприимчивых* к инфекции особей.

Почленное суммирование (16), (17) и (18) даёт

$$\dot{S} + \dot{I} + \dot{R} = 0,$$

что соответствует соотношению (15) и, тем самым, условию **a**.

На графиках III.9 и III.10 изображены зависимости численности групп S, I и R от времени при различных входных параметрах.

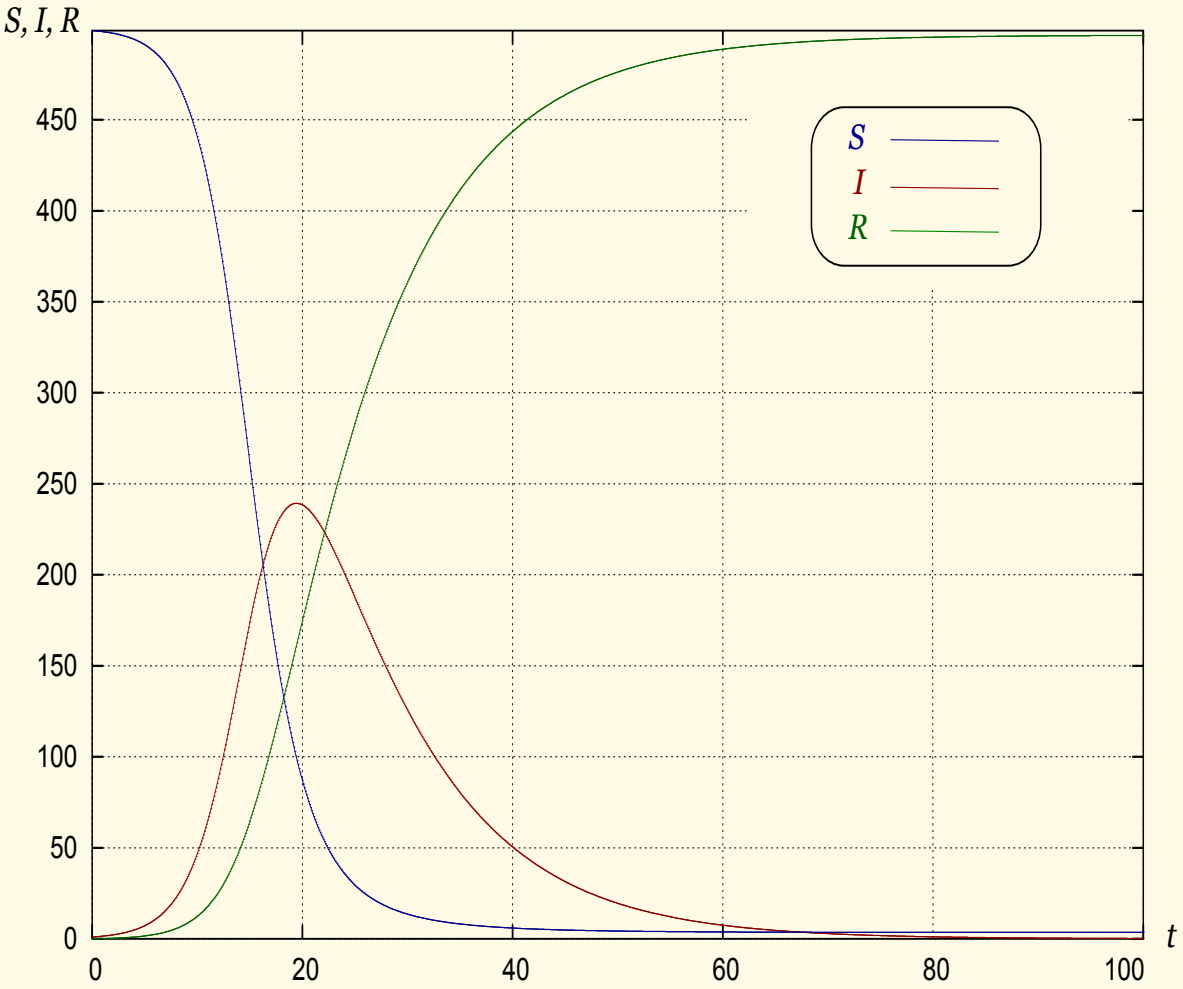


Рис. III.9. *SIR-модель,* $\alpha = 0{,}001$, $\beta = 0{,}1$, $I_0 = 1$, $S_0 = 499$

Борьба с эпидемиями включает комплекс санитарно-гигиенических, лечебно-профилактических и административных мер, направленных на локализацию инфекции в очаге заражения, недопущению заражения здоровых лиц и ликвидации очага заражения. В частности, противоэпидемические мероприятия, которые принимаются в рамках борьбы с

эпидемией *гриппа*, по большей части носят в основном ограничения административные, нежели врачебные. А именно: ограничения и запреты различных культурно-массовых и спортивных мероприятий, приостановка занятий в школах и детских садах, пропаганда среди населениям методов профилактики и защиты от гриппа и т. д. В построенной модели все эти меры, очевидно, приводят к *уменьшению коэффициента* α. Соответственно, своевременное выявление, изоляция и лечение заболевших позволяет *увеличить величину параметра* β.

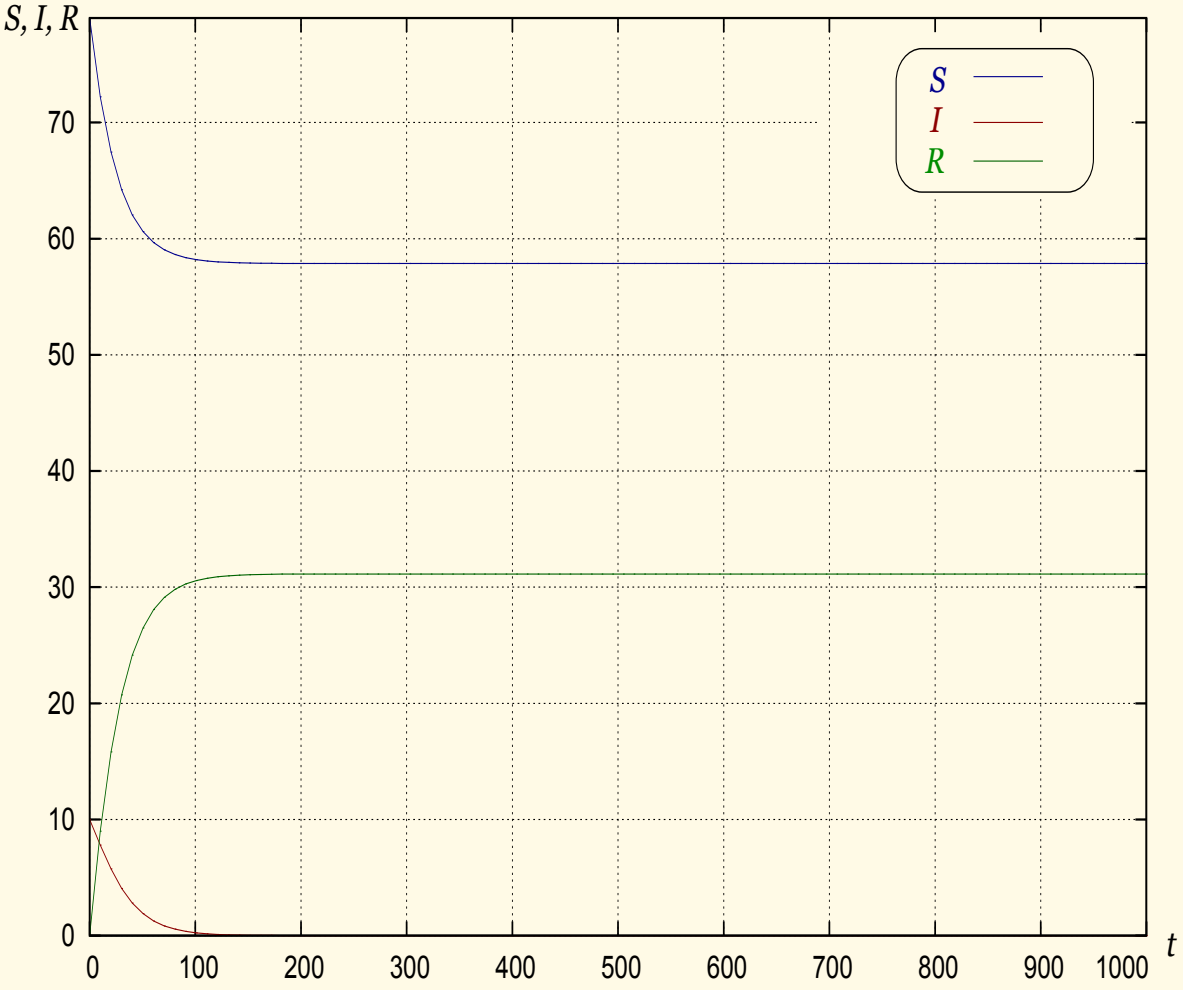


Рис. III.10. *SIR-модель,* $\alpha = 0{,}001$, $\beta = 0{,}1$, $I_0 = 10$, $S_0 = 80$

Вопрос III.4. При каких значениях параметров α, β и начальных условиях SIR-модели эпидемия не начнется, т. е. количество инфицированных не будет увеличиваться[4]?

Вопрос III.5. Очевидно, что *при прочих равных условиях* болезнь — а, тем более, эпидемию — всегда лучше предупредить, чем лечить. На основе анализа SIR-модели (16)–(19) показать, что параметры α, β в

[4] В реальности началом эпидемии считается момент, когда количество заболевших превышает т. н. *эпидемический порог*, величину которого для каждого заболевания определяют по специальным методикам расчета.

одинаковой степени влияют на её поведение и, следовательно, исходя из модели, профилактика лучше лечения.

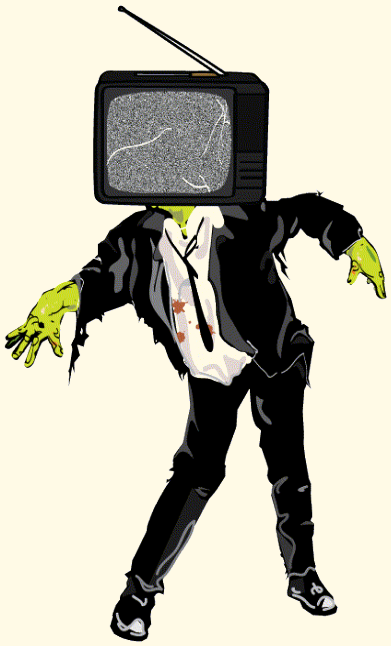
Вопрос III.6. На основе следующих данных[5]

% \ месяц	октябрь	ноябрь	декабрь	январь	февраль
S	92,753	84,517	81,883	81,223	80,565
I	0,033	0,073	0,084	0,088	0,091
R	7,213	15,410	18,033	18,689	19,344

произвести калибровку SIR-модели, т. е. подобрать соответствующие величины α, β, и затем провести вычислительный эксперимент, на основе результатов которого предсказать дальнейший ход эпидемии. Аппроксимацию данных проводить с абсолютной погрешностью не более $0{,}02$.

2.2. Зомби-апокалипсис, SZR-модель

В современной поп-культуре под зомби (англ. zombie) понимают мифический персонаж, оживший мертвец, зараженный неким вирусом. Зомби широко распространены в массовой культуре, особенно начиная с



фильма режиссёра Джорджа Ромеро (George Romero) «Ночь живых мертвецов» («Night of the Living Dead», 1968 г). С тех пор на эту актуальную тему снято огромное количество художественных фильмов и телевизионных сериалов, выпущено множество компьютерных игр, книг и комиксов [74], в настоящее время уже образующих самостоятельный поджанр фильмов/литературы ужасов (horror literature, horror fiction). Сейчас понятие «зомби» часто используют для обозначения человека, который *под некоторым внешним воздействием*, как правило, — неявным, теряет способность самостоятельно мыслить и действовать, находится под сильным влиянием каких-либо идей, убеждений или увлечений. Ученые всё чаще говорят о *зомбировании* с использованием специальных методов *социального программирования* и *информационно-психологической войны:* реклама, пропаганда, манипулирование массовым сознанием и др. [75], [76].

В произведениях искусства встречается большое разнообразие поведения, скорости передвижения, степени разумности и других характеристик зомби. Для построения модели [77] будем считать, что:

[5] Эти данные соответствуют реальной пандемии гриппа H1N1 2009–2010 гг. в США (в СМИ её называли «свиной грипп»), приведены Центром по контролю и профилактике заболеваний (Center for Disease Control and Prevention, CDC), http://www.cdc.gov/h1n1flu/estimates_2009_h1n1.htm.

- зомби не имеют памяти и разума;
- зомби действуют большой неорганизованной толпой, единственное стремление которой — пожирать живую человеческую плоть, «своих» зомби не трогают и, обычно вообще игнорируют;
- зомби передвигаются медленно, слабы, и представляют опасность только в составе большой группы или при неожиданном нападении;
- единственный надежный способ уничтожения зомби — поражение головы, при всех других травмах зомби может «воскреснуть»;
- чтобы человек превратился в зомби, достаточно одного укуса последнего.

Главное отличие данной модели от моделей распространения эпидемии, где заболевшие могут умереть только один раз, в том, что живые мертвецы после гибели способны возвращаться «к жизни». Также эта модель учитывает, что в зомби превращаются мертвецы, а люди постоянно нападают на носителей зомби-вируса, пытаясь их уничтожить.

Интересно отметить, что рассматриваемая модель имеет много общего с феноменом распространения в обществе *популярных идей*, которые выступают в роли зомби-вируса. В частности в такой интерпретации, уничтожение зомби можно трактовать как отказ от идеи *под воздействием*[6] общества и/или государства, однако индивидуум может впоследствии передумать и снова вернуться к ней — «воскрешение» зомби.

Переходя к описанию модели, популяцию разобъем на три группы:

- $S = S(t)$ — susceptible, чувствительные, количество незомбированных людей к моменту времени t;
- $Z = Z(t)$ — zombies, зомби.
- $R = R(t)$ — removed (удаленные) — уничтоженные зомби, некоторые из которых могут «воскреснуть».

Перечислим явно основные предположения и упрощения, принятые в данной модели:

a. В модели не принимается во внимание демографические изменения: рождение и смерть, по причинам не связанным с рассматриваемым зомби-вирусом, миграции и т. д., таким образом, на временном отрезке моделирования

$$S(t) + Z(t) + R(t) = \text{const} \tag{20}$$

b. Все здоровые индивидумы S популяции имеют одинаковую вероятность α заражения при контакте с зомби. Людям вирус передаётся

[6]Как известно, цивилизация накопила большой исторический опыт в борьбе с «вредоносными идеями», для чего создала ряд эффективных институтов, от Инквизиции до Гестапо и современных СМИ. Устранение идеи вместе с её носителем — также давняя историческая традиция и, следовательно, не противоречит такой интерпретации модели.

только при укусе зомби. Инкубационный период заболевания пренебрежимо мал, т. е. инфицирование зомби-вирусом в данной модели происходит мгновенно.

c. Люди, атакуя зомби, уничтожают их с постоянной вероятностью β.

d. Доля «возрождённых» зомби γ постоянна.

Из этих условий следуют уравнения

$$\dot{S} = -\alpha SZ, \tag{21}$$

$$\dot{Z} = \alpha SZ + \gamma R - \beta SZ, \tag{22}$$

$$\dot{R} = \beta SZ - \gamma R. \tag{23}$$

$$S(0) = S_0, Z(0) = Z_0, R(0) = R_0, \qquad 0 \leqslant \alpha, \beta, \gamma \leqslant 1. \tag{24}$$

Подробнее. Уравнение (21) следует из предположений **a** и **b**: количество контактирующих особей из групп в составе S и Z членов равно величине SZ, умноженной на вероятность зомбирования α, и пропорционально скорости $\dot{S}$ изменения численности; количество S восприимчивых к инфекции индивидуумов убывает со временем, поэтому в правой части (21) стоит знак минус. Соотношение (22) вытекает из предположений **b**, **c** и **d**: скорость зомбирования прямо противоположна скорости $\dot{S}$ без скорости $\dot{R}$ уничтожения зомби. Уравнение (23) выражает факт равенства скорости «убиения» зомби количеству контактирующих особей из групп в составе S и Z членов, умноженной на вероятность уничтожения зомби β, за вычетом доли «воскресших» γR.

На графиках III.11, III.12, III.13 изображены зависимости численности групп S, Z и R от времени при различных входных параметрах модели.

Почленное суммирование (21), (22) и (23) приводит к равенству

$$\dot{S} + \dot{Z} + \dot{R} = 0,$$

что соответствует соотношению (20) и, тем самым, условию **a** — постоянство количества членов популяции на временном отрезке моделирования.

Вычислительные эксперименты на модели (21)–(23) приводят к выводу о почти неизбежном всеобщем зомбировании (рис. III.11, III.12). Однако при некоторых наборах входных параметров (рис. III.13) для людей имеется возможность выживания. Не трудно показать, что решение системы уравнений модели имеет два состояния равновесия. Действительно, из системы уравнений

$$\begin{aligned} -\alpha SZ &= 0, \\ \alpha SZ + \gamma R - \beta SZ &= 0, \\ \beta SZ - \gamma R &= 0. \end{aligned}$$

следует, что либо S, либо Z равны нулю. Следовательно, имеются две точки покоя $(0, \overline{Z}, 0)$ (зомби-апокалипсис) и $(\overline{S}, 0, 0)$ (выживание людей), но последняя неустойчива.

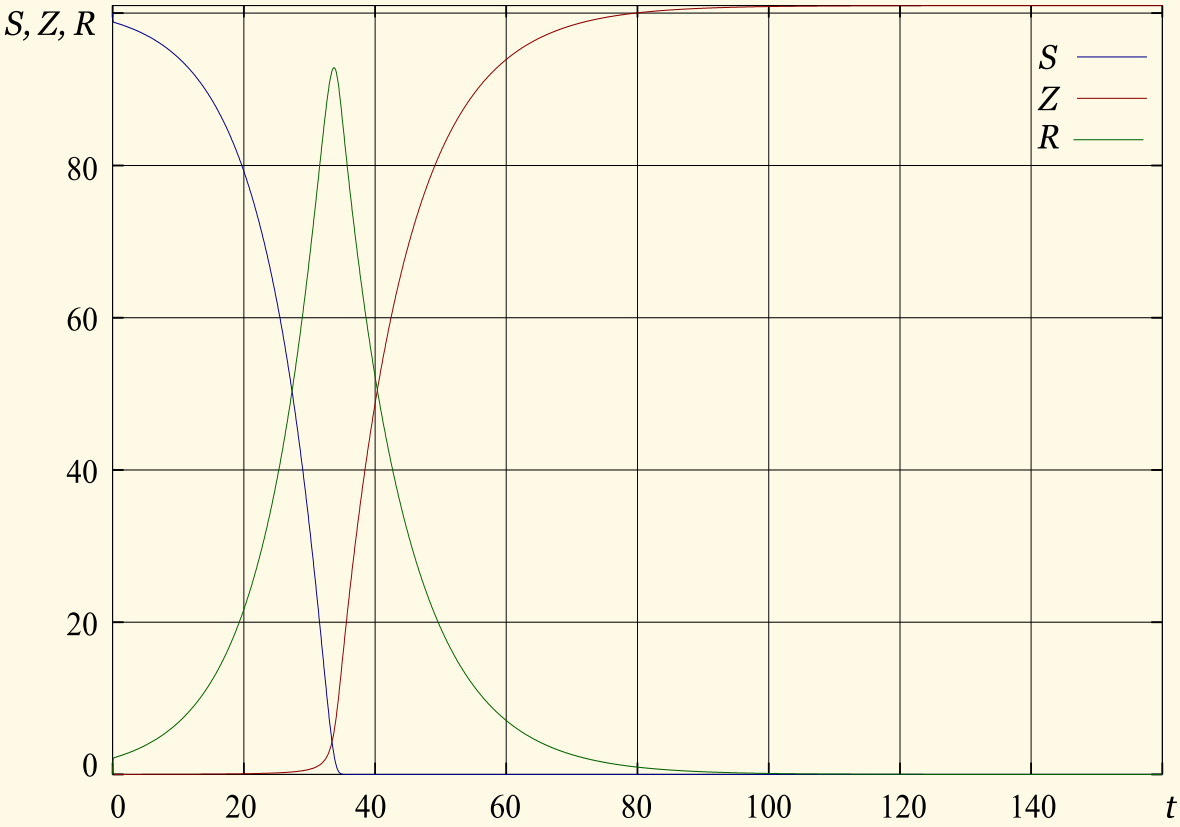


Рис. III.11. *SZR-модель с параметрами:* $\alpha = 0{,}35$, $\beta = 0{,}65$, $\gamma = 0{,}1$, $S_0 = 100$, $Z_0 = 1$, $R_0 = 0$

Авторы модели [77] рассмотрели её различные модификации и пришли к выводу: *единственный надежный* способ избежать зомби-апокалипсиса — тотальное уничтожение всех зомби. Любые другие меры, типа карантина или медикаментозного лечения, не спасут от «живых мертвецов».

Вопрос III.7. Исследовать устойчивость по первому приближению состояния $(0, \overline{Z}, 0)$ и $(\overline{S}, 0, 0)$ системы (21)–(23).

2.3. Цикады и простые числа

Известно, что некоторые цикады рода Magicicada имеют жизненные циклы, периоды которых выражаются *простыми числами*, — цикады появляются каждые 7, 13 или 17 лет, что трудно объяснить с чисто биологической точки зрения. Личинки периодических цикад живут под землей, питаясь соками корней растений. Они проходят через несколько стадий развития и выходят на поверхность весной 13-го или 17-го года своей жизни, появляясь сразу в большом числе и практически одновременно. Их массовое появление — способ выживания большинства индивидуумов рода посредством «пресыщения хищников», которые ими питаются. Есть предположение [79], [78], что циклы появления цикад, периоды которых равны простым числам, также являются

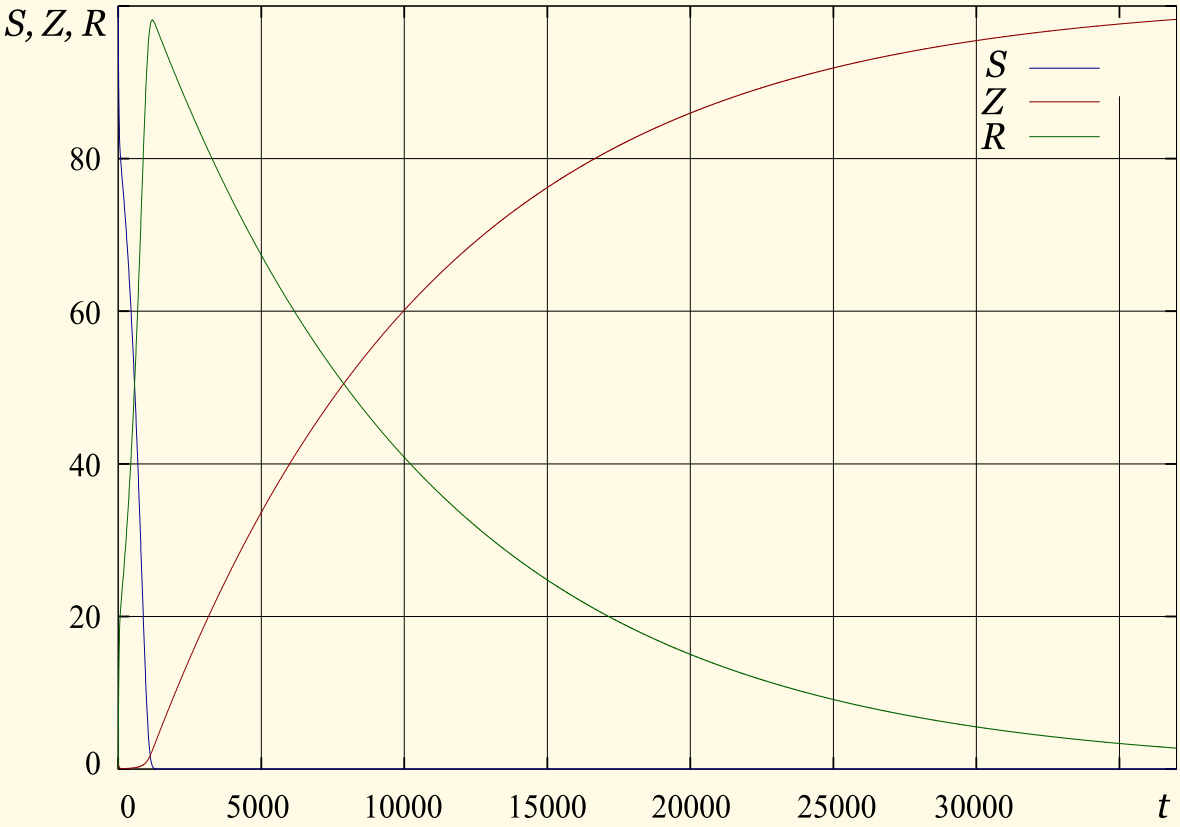


Рис. III.12. *SZR-модель с параметрами:* $\alpha = 0{,}0095$, $\beta = 0{,}01$, $\gamma = 0{,}0001$, $S_0 = 100$, $Z_0 = 1$, $R_0 = 0$

частью «стратегии» выживания. Рассмотрим математическую модель, показывающую возможный механизм[7] такой стратегии выживания. В модели взаимодействуют две популяции: хищники и жертвы. Предположим, следуя [79], что приспособляемость популяций как хищников так и их жертв выражается в выборе периодов количества лет, проходящих между их появлениями. Пусть хищник появляется каждые X лет, а жертва — Y лет. Определим «функцию выигрыша» хищника в год t следующим образом

$$f_X(t) = \begin{cases} 0, & \text{если хищник не появился в год } t; \\ 1, & \text{если хищник и жертва появились в год } t; \\ -1, & \text{если хищник появился, а жертва нет.} \end{cases}$$

Аналогично, для жертвы

$$f_Y(t) = \begin{cases} 0, & \text{если жертва не появилась в год } t; \\ -1, & \text{если хищник и жертва появились в год } t; \\ 1, & \text{если жертва появилась, а хищник нет.} \end{cases}$$

Определим величину средней приспособленности хищника F_X. За время XY хищник появится $XY/X = Y$ раз, т. е. его популяция за это

[7] Стремясь к реализму, сходу отвергаем предположение о том, что цикады знают теорию чисел.

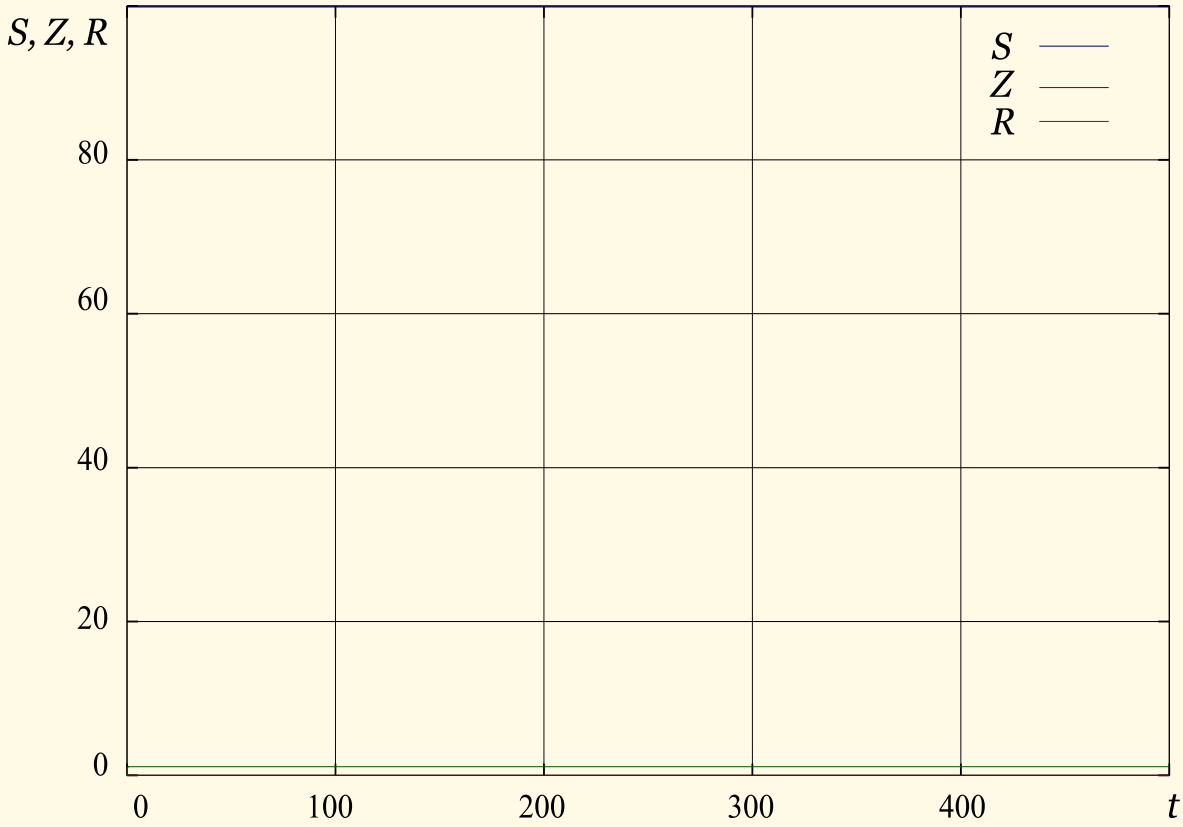


Рис. III.13. *SZR-модель с параметрами:* $\alpha = 0{,}0095$, $\beta = 0{,}09$, $\gamma = 0{,}0001$, $S_0 = 100$, $Z_0 = 1$, $R_0 = 0$

время имеет Y поколений, поэтому примем

$$F_X = \frac{1}{Y} \sum_{t=0}^{XY-1} f_X(t). \tag{25}$$

Для жертвы, соответственно

$$F_Y = \frac{1}{X} \sum_{t=0}^{XY-1} f_Y(t). \tag{26}$$

Предположим, что периоды появлений, как жертв, так и хищников, зависят от случайных мутаций, а «целью эволюции» обеих популяций является наилучшая приспособленность, т. е. максимизация функций (25), (26) за счёт выборов наиболее подходящих значений величин периодов X, Y. Именно, переход популяции хищников от периода X к периоду X' происходит тогда и только тогда, когда это улучшает их приспособленность:

$$F_{X'} > F_X. \tag{27}$$

Аналогично, для жертв условие перехода от периода Y к периоду Y':

$$F_{Y'} > F_Y. \tag{28}$$

Будем также, для простоты, считать, что «взаимодействие» популяций всегда имело место в начальный момент времени $t = 0$.

Анализ модели

Выразим F_X, F_Y в явном виде. За XY лет хищник появится Y раз, а обе популяции $XY/\text{НОК}\big(X,Y\big)$ раз. Следовательно, «выигрыш» хищника за время XY составит

$$\frac{XY}{\text{НОК}\big(X,Y\big)} - \left(Y - \frac{XY}{\text{НОК}\big(X,Y\big)}\right) = 2\,\text{НОД}\big(X,Y\big) - Y.$$

Отсюда, в соответствии с (25) и учитывая известное тождество

$$\text{НОК}\big(X,Y\big)\text{НОД}\big(X,Y\big) = XY,$$

получим

$$F_X = \frac{2}{Y}\,\text{НОД}\big(X,Y\big) - 1 \tag{29}$$

и, точно также, для жертвы

$$F_Y = 1 - \frac{2}{X}\,\text{НОД}\big(X,Y\big). \tag{30}$$

Зададим ограничения на величины периодов прямоугольником

$$H: \quad 2 \leqslant X \leqslant \frac{L}{2} + 1, \quad \frac{L}{2} + 2 \leqslant Y \leqslant L \tag{31}$$

и покажем, что при этих условиях случайная последовательность периодов жертвы сведётся к периоду, длина которого выражается простым числом. Точнее, докажем, что если Y составное число, то найдётся период который, в соответствии с (28), изменит Y (для лучшей приспособленности), а если Y — простое, то период не изменится.

Предположим, что Y_c — составное число и X^* — период, «выбранный» хищниками при данном периоде жертв. Тогда найдётся такое Y', что

$$F_{Y'} > F_{Y_c},$$

или, ввиду соотношения (30)

$$d \stackrel{\text{def}}{=} \text{НОД}\big(X^*,Y'\big) < \text{НОД}\big(X^*,Y_c\big).$$

Очевидно, что X^* есть наибольший из делителей Y_c — формула (29), удовлетворяющий ограничениям

$$2 \leqslant X^* \leqslant L/2 + 1; \tag{32}$$

$$2 \leqslant X^* \leqslant Y_c - 1. \tag{33}$$

Следовательно, $X^* = \text{НОД}\big(X^*,Y_c\big)$ и достаточно указать такое Y', что

$$d < X^*.$$

Таковыми значениями будут, например,

$$Y' = Y_c \pm 1.$$

Действительно, так как $X^* = \text{НОД}\big(X^*,Y_c\big)$, то $Y_c = X^*Y_1$ и, поскольку $d = \text{НОД}\big(X^*,Y_c \pm 1\big)$, то $Y_c \pm 1 = dY_2$, $X^* = dX_2$. Из

последнего соотношения следует, что

$$d < X^* \iff X_2 > 1.$$

И, если предположить, что $X_2 = 1$, то получим $d = X^*$ и

$$Y_c = dY_2 \mp 1 = X^*Y_1 = dY_1,$$

откуда

$$X^*(Y_2 - Y_1) = \pm 1$$

и, следовательно, $X^* = \pm 1$, что противоречит (32).

Если же Y_c — простое число, то при условии (33) X^* взаимно просто с Y_c и, следовательно, $\text{НОД}\big(X^*, Y_c\big)$ уже имеет наименьшее возможное значение, а F_{Y_c}, соответственно, максимально.

Вопрос III.8. На языке высокого уровня реализовать описанную выше модель. Входные параметры модели: L.

2.4. Модель отрезвления

«Среди множества продуктов, созданных и потребляемых человечеством, водка, или, говоря более общим термином, „хлебное вино“, занимает совершенно особое и значительное положение по своему разнообразному влиянию на человеческое общество, на отношения людей и на возникающие общественные проблемы. При этом весьма важно подчеркнуть, что такие три магистральных направления проблем, поставленных возникновением спиртных напитков, как фискальная, производственная, социальная, веками не умирают, а, наоборот, имеют тенденцию возрастать и усложняться в связи с развитием жизни человеческого общества» (В. В. Похлебкин [80]).

Водка — водно-спиртовой раствор, крепость которого по российским стандартам составляет обычно 40%. Основной компонент водки и других алкогольных напитков — депрессант, угнетающий центральную нервную систему человека, — *этиловый спирт* C_2H_5OH, или *этанол*, *алкоголь*.

Этанол воздействует на многие системы организма. В частности, среди основных последствий *умеренного* приема спиртных напитков можно выделить усиленную выработку в гипоталамусе *эндорфинов*, «гормонов удовольствия», повышение уровня которых сопровождается улучшением психофизиологического статуса, повышением настроения, снижением утомляемости и т. д. Однако с другой стороны, как многим хорошо известно, после употребления *чрезмерного* количества алкоголя, как правило, на следующее утро наступает своеобразное болезненное состояние, называемое *алкогольный абстинентный синдром* или, проще говоря, по-

хмелье. Поэтому тема количественного описания процесса отрезвления[8], безусловно, важна и актуальна.

Построим простую модель процесса удаления алкоголя из крови. Будем считать, что концентрация этанола в крови идет с постоянной скоростью v‰[9] в час. Поскольку большая часть алкоголя разлагается печенью, возможности которой ограничены, скорость удаления зададим формулой Михаэлиса — Ментена (Michaelis — Menten) [81]

$$\frac{c}{k/x+1} = \frac{cx}{k+x},$$

где x — концентрация спирта в крови, c, k — константы.

Таким образом, процесс описывается задачами Коши

$$\dot{x} = v - \frac{cx}{k+x}, \qquad x(0) = x_0, \qquad 0 \leqslant t \leqslant T, \tag{34}$$

$$\dot{x} = -\frac{cx}{k+x}, \qquad x(T) = x_T, \qquad t \geqslant T, \tag{35}$$

где T — время окончания приема алкоголя.

Известно, что средняя скорость выведения этанола из крови равна $\beta \approx 0{,}15$‰ в час и, следовательно, концентрацию алкоголя можно приближенно описать линейной функцией (см. рис. III.14)

$$w(t) = x_T + \beta(T - t).$$

Психофизиологический эффект существенно зависит от концентрации этанола в крови [82]:

Уровень алкоголя (‰)	Поведение
0,5–1,0	Наблюдается чувство эйфории.
1,0–1,5	Расстройство координации движений от слабого до среднего уровней.
1,5–2,0	Полное отсутствие координации движений, невнятность речи.
2,0–2,5	Потеря памяти.
более 2,5	Индуцированный сон или потеря сознания.

Пример. Как показано на рис. III.14, индивидуум, начав с состояния эйфории, непрерывно и умеренно употребляя алкоголь, повышает его концентрацию со скоростью 0,18‰/час и достигает за 3 часа некоторого

[8] Моделирование процесса пития — задача более сложная, поскольку этот процесс сильно зависит от субъективных факторов, выражающихся в сентенциях: «Водка в малых дозах полезна в любом количестве», «Между первой и второй промежуток небольшой», «Я свою норму знаю: упал — хватит!», «Сколько водки не бери, все равно в магазин бежать» и т. д.

[9] Промилле (лат. per mille — на тысячу, обозначение: ‰) — одна тысячная доля, $1/10$ процента. Содержание алкоголя в крови 1‰ означает, что в одном литре крови находится 1 грамм чистого алкоголя.

расстройства координации движений. Затем, благодаря отличной работе печени[10], отрезвление наступает примерно через 9 часов.

Концентрация алкоголя в крови зависит от массы и пола пациента. Шведским химиком Э. Видмарком (E. Widmark) предложена формула

$$x = \frac{A}{mr},$$

где x — концентрация алкоголя в крови (‰), A — масса принятого алкоголя[11] в граммах, m — масса человека (кг), r — коэффициент Видмарка: $r = 0{,}7$ для мужчин, $r = 0{,}6$ для женщин. Для более точного результата учитывают, что не весь выпитый алкоголь попадает в кровь. Для этого из массы выпитого алкоголя вычитают 10%.

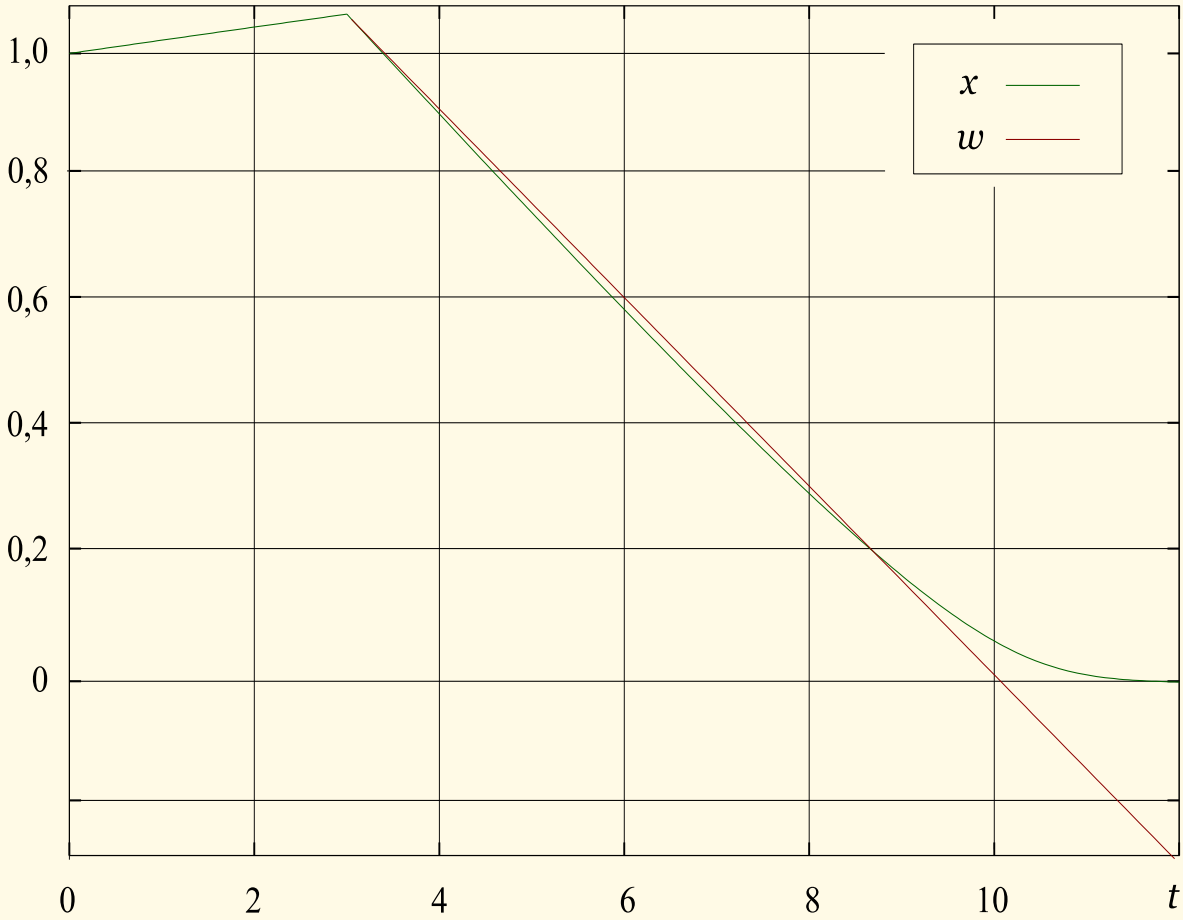


Рис. III.14. *Концентрация алкоголя в крови, параметры модели:* $v = 0{,}18$, $c = 0{,}17$, $k = 0{,}07$, $x_0 = 1{,}0$, $T = 3$, $w = x_T + \beta(T - t)$

Вопрос III.9. Пусть пациент, чьё поведение описывается моделью (34)–(35) с параметрами, приведенными на рис. III.14, — мужчина весом 80 кг. Найти выпитое им общее количество водки.

[10] Алкоголь в печени окисляется до токсичного вещества *альдегида*, который позднее, при попадании в кровь, и вызывает похмелье.

[11] Плотность спирта равна $0{,}7893$ г/см^3.

2.5. Задачи

III.5. Волки и медведи. Имеются две биологические популяции — волки и медведи, численностью N и M соответственно. Их развитие описывается системой уравнений

$$\dot{M} = M\big(e_M - g_M F(M,N)\big),$$
$$\dot{N} = N\big(e_N - g_N F(M,N)\big),$$
$$M(0) = M_0, \quad N(0) = N_0,$$

где положительные правильные дроби e_M, e_N — коэффициенты прироста и g_M, g_N — «коэффициенты прожорливости» соответствующих популяций. $F(M,N)$ — количество пищи, поедаемой обеими популяциями в единицу времени, задаётся одним из соотношений

$$F(M,N) = pM + qN; \tag{36}$$
$$F(M,N) = qN, \tag{37}$$

здесь p, q — некоторые положительные константы. (36) соответствует случаю, когда обе популяции активны, а (37) — когда медведи впадают в спячку. Переключение между режимами происходит автоматически, причём переход с режима (37) на (36) через 3 месяца, а с (36) на (37) через 9 месяцев. Сделать графическую анимированную иллюстрацию модели (графики зависимости M, N от времени), подобрав подходящий масштаб времени τ.

Входные параметры: M_0, N_0, e_M, e_N, g_M, g_N, p, q, τ. Литература: [25, стр. 436–437].

III.6. Модель обучения. В данной модели обучения предполагается, что выполнены следующие условия:

- уровень обученности характеризуется средней вероятностью p_n *правильных ответов* на некоторые вопросы (тесты) после прохождения $n-$ого этапа (часа) обучения;
- уровень обученности постоянно возрастает, но состояние полного обучения никогда не достигается;
- вероятность неправильного ответа $q_n = 1 - p_n$ линейно убывает, причём коэффициент убывания α постоянен на всех этапах;
- сохранившаяся в памяти информация соответствует *кривой забывания* Эббингауза (Hermann Ebbinghaus), построенной исходя из данных натурных экспериментов

время, час	0,33	1	8,8	24	48	6×24	31×24
сохранилось, %	58,2	44,2	35,8	33,7	27,8	25,4	21,1

Построить и исследовать дискретную модель обучения, шаг дискретизации равен одному часу. Кривую забывания аппроксимировать функцией

$$f(t) = ae^{-bt} + c,$$

где t — время, a, b, c — константы, с относительной погрешностью не более $0{,}03$. Уровень начальной обученности — p_0.

Входные параметры: α, p_0. Литература: [32].

III.7. Фармакокинетическая модель. Камера содержит некоторый объём жидкости, неизменный в течение времени. Заданный объём лекарственного препарата всасывается в камеру пропорционально своей

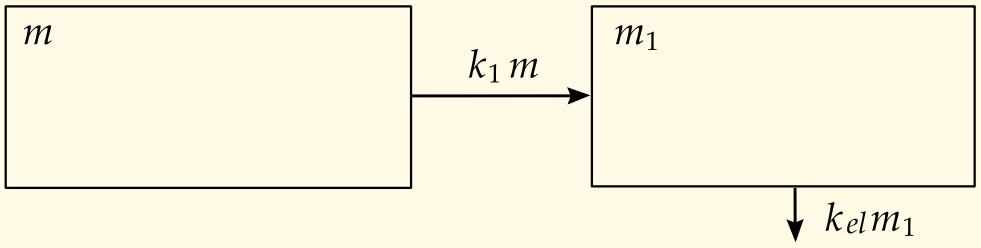


Рис. III.15. *Схема фармакокинетической модели*

массе согласно уравнению

$$\dot{m} = -k_1 m,$$

где m — масса лекарственного препарата, k_1 — константа скорости поступления препарата в камеру.

Масса лекарственного препарата в месте введения в начальный момент времени равна m_0, а в самой камере в начальный момент препарата нет. Масса m_1 лекарственного препарата в камере меняется в соответствии с уравнением

$$\dot{m}_1 = k_1 m - k_{el} m_1,$$

где k_{el} — константа выведения препарата из камеры. Как только масса лекарственного препарата становится в месте введения меньше порогового значения ε, засекается отрезок времени T, по истечении которого в месте введения уничтожаются все остатки прежней дозы препарата и вводится новая доза m_0. Сделать графическую анимированную иллюстрацию модели, подобрав подходящий масштаб времени τ.

Входные параметры τ, k_1, k_{el}, m_0, T, ε. Литература: [25].

III.8. Эффективное лечение. Доктор не может поставить точный диагноз больному, но уверен, что болезнь может быть вызвана одним из четырех видов бактерий a, b, c, d. У врача есть 3 вида лекарств $(1, 2, 3)$, причем первое из них на 50% уничтожает бактерии a, b, c и никак не действует против d, второе — на 100% эффективно против a и бесполезно против всех остальных бактерий, третье — полностью лечит b, d, наполовину уничтожает c и не действует против a.

Рассматривая модель как матричную игру двух лиц, найти наилучшую процедуру лечения.

3. Военное дело

Кто — ещё до сражения — побеждает предварительным расчетом, у того шансов много; кто — ещё до сражения — не побеждает расчетом, у того шансов мало. У кого шансов много — побеждает; у кого шансов мало — не побеждает; тем более же тот, у кого шансов нет вовсе.

Сунь-цзы. Искусство войны

Хотя война и противоречит здравому смыслу, так как является средством решения вопросов силой, когда переговоры не приводят к положительному результату, однако ведение войны должно контролироваться разумом, если хотят, чтобы цель войны была достигнута.

Лиддел Гарт. Стратегия непрямых действий

Подсчитано [83], что за последние 3400 лет человечество прожило в мире только 268, т. е. около 8% своей письменной истории. В войнах XX в погибло не менее 108 миллионов человек, а общее количество убитых на протяжении всей истории оценивается от 150 миллионов до миллиарда. Поэтому ясно, что к такому важному государственному делу, как война и массовые убийства, вид Homo sapiens[1] из рода Люди,

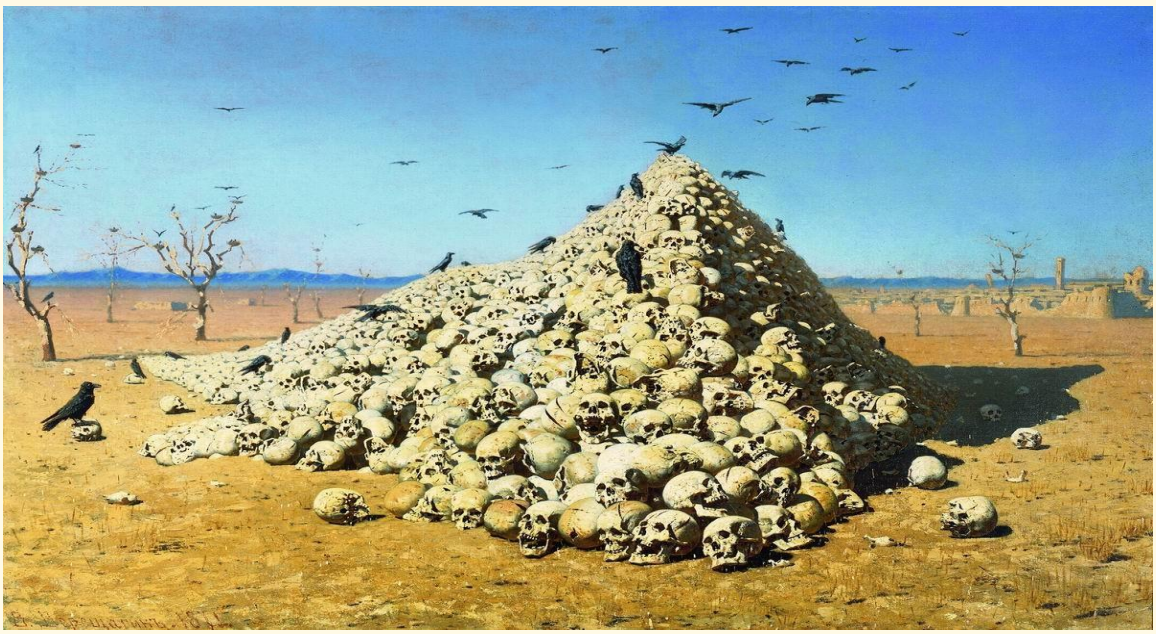


Рис. III.16. *Василий Верещагин. Апофеоз войны*

семейства гоминид, отряда приматов, класса млекопитающих, должен относиться со всей ответственностью.

[1]Homo Sapiens — лат. человек разумный. Это мы *сами* себя так назвали, благо возразить некому.

Поскольку homo sapiens'ы отличаются от прочих млекопитающих несколько большей развитостью *абстрактного мышления*, военное дело они издавна пытались поставить на научные основы.

3.1. Модель боевых действий Осипова — Ланчестера

Эта модель[2] применяется для описания динамики военных сражений. Пусть $B(t)$ — численность армии «синих» в момент времени t, $R(t)$ — численность армии «красных». Средняя эффективность стрельбы «синих» и «красных» задаётся константами b и r соответственно. Например, если $r = 0{,}7$, то в среднем из каждых 10 выстрелов «красных» 7 поражают солдат противника. Таким образом, параметры b и r характеризуют технические характеристики оружия и обученность солдат, его использующих, а таже тактическую грамотность и опыт командиров.

Построим модель боевых действий между армиями «синих» и «красных» с учетом следующих упрощающих предположений:

— противники несут потери только от вражеского огня;
— подкреплений не поступает ни к одной из сторон;
— «победа» означает полное уничтожение противника при ненулевой численности собственной армии;
— возможна патовая ситуация — «ничья», положение, приводящее к взаимному уничтожению, когда $B(t) \to 0$, $R(t) \to 0$.

Модель, с учётом имеющихся упрощений, описывается системой дифференциальных уравнений

$$\begin{aligned} \dot{R} &= -bB; \\ \dot{B} &= -rR; \\ B(0) &= B_0, \quad R(0) = R_0. \end{aligned}$$

Решение имеет вид

$$\begin{aligned} B &= C_1 e^{t\sqrt{br}} + C_2 e^{-t\sqrt{br}}; \\ R &= -C_1\sqrt{\frac{b}{r}}\, e^{t\sqrt{br}} + C_2\sqrt{\frac{b}{r}}\, e^{-t\sqrt{br}}. \end{aligned}$$

Константы интегрирования определяются из начальных условий:

$$C_1 = \frac{1}{2}\left(B_0 - R_0\sqrt{\frac{r}{b}}\right), \qquad C_2 = \frac{1}{2}\left(R_0\sqrt{\frac{r}{b}} + B_0\right).$$

Отсюда, в патовой ситуации $C_1 = 0$ и, следовательно, выполняется *закон Ланчестера:*

$$B_0 = R_0\sqrt{\frac{r}{b}}.$$

[2]Модель была предложена в 1915 году русским офицером М. П. Осиповым и, независимо, в 1916 году английским ученым и инженером Ф. Ланчестером (F. Lanchester).

Например, чтобы поставить противника, имеющего 3-хкратный численный перевес, в патовую ситуацию нужно иметь в 9 раз более эффективное оружие (и/или лучше обученный личный состав).

Таким образом, получаем, что

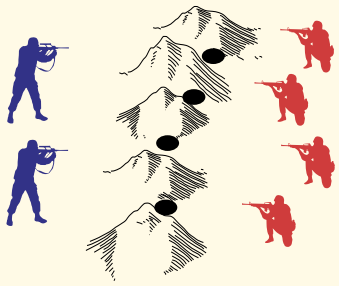
$$B = C_2\, e^{-t\sqrt{br}}, \qquad R = C_2\sqrt{\frac{b}{r}}\, e^{-t\sqrt{br}}$$

есть убывающие величины одного порядка.

Вопрос III.10. Пользуясь моделью Ланчестера — Осипова проанализировать бой группы сильнее бронированных, но менее скорострельных танков (советские ИС-2) против менее защищенных, но более скорострельных танков (немецкие Тигры PzVI). Принять скорострельность ИС равной 3 выстрела в минуту, Тигра — 9 выстрелов в минуту, вероятность поражения ИС равной 0,1, Тигра — 0,9, начальное количество равно 20 машинам у каждого противника.

3.2. Военная игра, оборона перевала

«Красные» обороняют четыре горных перевала силами 4-х батальонов. «Синие», имеющие 2 батальона, должны захватить *хотя-бы один* из перевалов. По условию игры перевал считается захваченным, если у «Синих» будет в этом пункте численный перевес (в количестве батальонов). Требуется определить оптимальные планы действий противников — их оптимальные *стратегии*, рассматривая данную модель как конечную игру двух лиц с нулевой суммой [33], [34], [35]. Решение начнем с перечисления всех возможных действий противоборствующих сторон. Стратегии «Синих»:



1. Батальоны направить на различные перевалы (всё равно какие);
2. Оба батальона штурмуют один перевал (любой из четырёх).

Стратегии «Красных»:

1. Поставить по 1-му батальону на каждый из перевалов;
2. Направить по 2 батальона на два перевала, остальные оставить незащищёнными;
3. На один из перевалов поставить 2 батальона, ещё каждый из двух перевалов защищать силами 1-го батальона и один перевал оставить без защиты.

Остальные возможные стратегии «Красных» явно абсурдны и потому далее не рассматриваются.

Проведём количественную оценку выигрышей во всех возможных ситуациях игры. В ситуации $(1,1)$ (1-я стратегия «Синих» и 1-я стратегия «Красных») выигрыш «Синих» примем равным 0. В ситуации $(1,2)$ у «Синих» имеются $6 = C_4^2$ равновероятных исходов, из которых только один проигрышный, следовательно, выигрыш «Синих» равен $5/6$.

$(1,3)$ — выигрыш «Синих» $= 1/2$ (всего есть $6 = C_4^2$ различных вариантов для «синих», из которых проигрышных $3 = C_3^2$). В остальных ситуациях вероятности победы «Синих» вполне очевидны. Полученные величины сведём в платёжную матрицу

	Красные 1	Красные 2	Красные 3
Синие 1	0	5/6	1/2
Синие 2	1	1/2	3/4,

или, если для удобства все элементы умножить на 12, то получим

	Красные 1	Красные 2	Красные 3
Синие 1	0	10	6
Синие 2	12	6	9.

Решение игры находим поэтапно, см. стр. 176:

- Матрица игры не имеет доминируемых строк/столбцов, следовательно, размерность игры невозможно понизить, см. стр. 174.
- Платёжная матрица не имеет седловой точки, поэтому, решение следует искать в смешанных стратегиях, 172.
- Для нахождения оптимальных стратегий сведём задачу к задаче линейного программирования, стр. 174.

Решим игру симплекс-методом (стр. 177), используя систему Maxima.

```
    /*  стратегии Красных */
--> load( "simplex" ) $
    L   : y1 + y2 + y3              /* целевая функция */ $
    eq1 :  0*y1 + 10*y2 + 6*y3 <= 1 $
    eq2 : 12*y1 + 6*y2 + 9*y3 <= 1 $
    eqs : [ eq1, eq2 ]               $
    max : maximize_lp( L, eqs ),  nonegative_lp =true      $
    v   : 1/max[1]                  /* цена игры       */ $
    q1  : rhs( max[2][1] )*v        /* оптимальные     */ $
    q2  : rhs( max[2][3] )*v        /* стратегии       */ $
    q3  : rhs( max[2][2] )*v        /* Красных         */ $
     print ( "оптимальные стратегии Красных ",
                                  q1, ", ", q2, ", ", q3 ) $
    /* стратегии Синих */
--> L : x1 + x2                     /* целевая функция */ $
    eq1 : 0*x1  + 12*x2 >= 1 $
    eq2 : 10*x1 +  6*x2 >= 1 $
    eq3 : 6*x1  +  9*x2 >= 1 $
    eqs : [ eq1, eq2, eq3 ]  $
    min : minimize_lp( L, eqs ),  nonegative_lp =true      $
    p1  : rhs( min[2][1] )*v        /* оптимальные     */ $
    p2  : rhs( min[2][2] )*v        /* стратегии Синих */ $
     print ( "оптим. стратегии Синих ", p1, ", ", p2 )     $
```

Расчеты дают: $\mathbf{p}^* = (3/8, 5/8)$, $\mathbf{q}^* = (1/4, 3/4, 0)$.

Следовательно, «Синие» в 3-х случаях из 8-ми должны использовать свою первую стратегию — атаковать любые два перевала силами одного батальона каждый, а в остальных 5-ти случаях штурмовать один из перевалов двумя батальонами. «Красные» должны с 25% вероятностью применять свою 1-ю стратегию (оборона каждого перевала одним батальоном) и с 75% вероятностью удерживать два перевала силами двух батальонов каждый, а третью стратегию (один перевал удерживать двумя батальонами, два перевала защищать одним батальоном каждый и один перевал оставить без защиты) не использовать.

Вопрос III.11. *Борьба на коммуникациях*. Военно-морские группировки «Синих» (4 боевых корабля и транспорт) и «Красных» (3 боевых корабля и транспорт) ведут борьбу на коммуникациях. Каждая из сторон может атаковать транспорт и/или боевые корабли противника, а также защищать свой транспорт, произвольно распределяя имеющиеся у нее силы.

Если количество атакующих кораблей больше числа подвергнувшихся нападению, то атакующий игрок сохраняет все свои силы и уничтожает всю группу противника, получая по очку за каждый боевой корабль и одно очко за транспорт, если последний был в составе группы. Если силы атакующей стороны меньше, он теряет все корабли, участвующие в нападении, а противник получает сумму, равную количеству нападавших кораблей и сохраняет все свои силы. При количественном равенстве атакующих и обороняющихся боевых кораблей никто ничего не выигрывает и не проигрывает.

Построить игровую модель, найти решение игры. Литература: [37, стр. 93–97].

Вопрос III.12. *Десант в Нормандии*. По мнению некоторых военных историков операция вторжения Союзников в Нормандию во время Второй мировой войны (1944 г) моделируется игрой двух лиц (Германия — стратегии: $A, \ldots, F$, Союзники — стратегии: $1, \ldots, 6$) с платежной матрицей

	A	B	C	D	E	F
1)	13	29	8	12	16	23
2)	18	22	21	22	29	31
3)	18	22	31	31	27	37.
4)	11	22	12	21	21	26
5)	18	16	19	14	19	28
6)	23	22	19	23	30	34

Исторически противники выбрали решение $(1, B)$. Найти правильное решение и оценить ущерб, понесенный сторонами, при таком выборе.

Литература: W. Drakert. Normandy: Game and Reality. Moves, No. 6 (1972) — http://strategyandtacticspress.com/library-files/Moves-%20Issue06.pdf.

3.3. Задачи

III.9. Партизанская война — военные действия регулярной армии против противника, базирующегося в природной среде (чаще всего в горной или лесистой местности) или городских условиях и пользующегося поддержкой местного населения или его значительной части. Партизаны, как правило, вооружены хуже правительственных войск, но менее их уязвимы, так как действуют скрытно, зачастую оставаясь невидимыми для врага[3], избегают открытых и крупных столкновений с противником. Построить модель боевых действий регулярной армии против партизан. Предполагается, что правительственные войска (численностью в момент времени t равной $g(t)$) несут потери только от партизанских действий — атаки блок-постов, диверсии и т. д. Их потери пропорциональны численности партизан $p(t)$. Считать, что регулярная армия, не имея достоверной информации, «работает по площадям» — бомбардировки, артиллерийские обстрелы и т. п., нанося повстанцам потери, пропорциональные численности как партизан, так и своей собственной. На основе анализа модели определить условие победы партизан (армии), если:

a. подкреплений к обоим противникам не поступает.

b. партизаны вербуют новых сторонников со скоростью, пропорциональной темпу убывания живой силы правительственных войск.

По аналогии с моделью Осипова — Ланчестера 3.1 получим при выполнении условия **a**

$$\begin{aligned}
&\dot g = -\beta p,\\
&\dot p = -\alpha p g,\\
&p(0) = p_0, \quad g(0) = g_0,\\
&\alpha, \beta = \text{const} > 0.
\end{aligned}$$

Для случая **b**

$$\begin{aligned}
&\dot g = -\beta p,\\
&\dot p = -\alpha p g + \gamma \dot g,\\
&p(0) = p_0, \quad g(0) = g_0,\\
&\alpha, \beta, \gamma = \text{const} > 0.
\end{aligned}$$

[3] «„Укусит и убежит“ — так в пренебрежительном тоне нередко отзываются о действиях партизанского отряда. Да, именно так он действует: укусит, убежит, ждет, подстерегает, снова кусает и снова бежит, не давая покоя врагу» — Че Гевара. Партизанская война. М.: Издательство иностранной литературы, 1961.

4. Социология и политология

Человеческая цивилизация похожа на корабль, построенный без плана. Постройка удалась на диво. Цивилизация создала мощные двигатели и освоила недра своего корабля — неравномерно, правда, но это-то поправимо. Однако у корабля нет кормчего. Цивилизации недостает знания, которое позволило бы выбрать определенный курс из многих возможных, вместо того чтобы дрейфовать в потоках случайных открытий.

С. Лем. Сумма технологий

Модели общественных процессов содержат огромное количество всякого рода неопределенностей. Поэтому их анализ не дает окончательных однозначных выводов, которые мы привыкли извлекать при исследовании хорошо поставленной физической или технической задачи. Существует мнение, что законы в общественных науках имеют характер тенденций[1]*. С этим следует согласиться.*

Н. Н. Моисеев. Математика в социальных науках

. . . преклонение перед математикой в начале XX в. превратилось в своеобразный культ, отвлекший много сил у естественников и гуманитариев.

Л. Н. Гумилев [48]

С древних времён многие мыслители, видя несовершенство социально-политического устройства общества и государства, предлагали свои планы их улучшения. Платон, Аристотель, Томмазо Кампанелла, Томас Мор, Томас Гоббс, Жан-Жак Руссо, Анри Сен-Симон, Карл Маркс, Жозеф Прудон, П. А. Кропоткин, Карл Поппер и др. выдвигали свои теории и модели общественных и государственных структур, которые благодарное человечество в той или иной мере старалось реализоввть на практике (см. рис. III.17). Когда стало понятно, что «традиционные методы» исследования для прогресса наук оказались исчерпанными, началась математизация общественных дисциплин.

В эпоху Просвещения «успех ньютоновской системы просто кружил головы мыслителям нового времени: Сен-Симон и Фурье обещали построить „универсальную“ систему общества то по „принципу тяготения“,

[1] Тенденция (от ср. век. лат. tendentia — направленность) — 1) возможность тех или иных событий развиваться в данном направлении, 2) замысел, идея какого-либо изложения, 3) предвзятая, односторонняя, навязываемая мысль.

то по „принципу гиперболы или эллипса“. Ученики Сен-Симона и в особенности Огюст Конт настаивали на возможности рассчитать все — и организмы и социальную жизнь, включая мораль и религию „как в политехнической школе учат рассчитывать мосты“. И они были уверены, что придут к куда более совершенным результатам, чем такой плохой математик как Природа. При этом, как замечает фон Хайек, им, видимо, ни разу не пришло в голову, что человеческий мозг, который они наделяют такими способностями, создан той же несовершенной Природой!» [39]. Математизация наук, в том числе общественных, однако, продолжалась и к настоящему времени стало ясно[2], что «принципиально не математических» научных дисциплин вообще не существует.

Рис. III.17. *Питер Брейгель Старший. Притча о слепых*

Математическое описание общественных явлений сталкивается со значительными трудностями в силу необходимости учитывать субъективные стороны деятельности людей (их цели, волю, интересы, ценностные ориентировки, мотивации и т. д.); усугубляет вопрос то, что и сами задачи часто формулируются субъективно[3], в неопределённых, «размытых» терминах. Математическое моделирование в общественных науках, в отличие от естественных, вызывает также проблемы, которые в основном связаны с отсутствием *экспериментальной наглядности* и возможности *воспроизводства* моделируемого явления или процесса.

[2]Ясно, разумеется, не всем. Некоторая часть «гуманитариев по жизни» с особо богатым внутренним миром, имеют прямо противоположное мнение, с негодованием отвергая любые попытки «поверить алгеброй гармонию», но эта книга — не для них.

[3]«Памятуя афоризм Найвена: „Нет такого благородного дела, к которому не пристали бы дураки“, — к использованию математических моделей следует подходить с определенной осторожностью» [40].

4.1. Демократия как скрытая форма диктатуры

Одной из основ политического режима, называемого *демократия*[4], как известно, является назначение должностных лиц государства на основе *честных* и состязательных выборов. Рассмотрим математическую модель избирательной системы.

Пусть имеются конечные множества, E («избиратели», «электорат», «эксперты») и C («кандидаты», «альтернативы»). Каждый из избирателей имеет относительно любого из кандидатов определённое мнение, выражаемое в *предпочтениях* одних кандидатов по сравнению с другими, т. е. избиратели *ранжируют* (упорядочивают) множество альтернатив C — создают свой *профиль предпочтений*. Отношение предпочтения избирателем e кандидата c_1 по сравнению с кандидатом c_2 запишем в виде

$$c_1 \succ_e c_2,$$

что можно читать: «c_1 лучше c_2», или «c_1 предпочтительнее c_2», или «c_1 выше c_2» (с точки зрения e). Очевидно, что определённое нами отношение препочтения обладает свойством *транзитивности:*

$$\big((a \succ b)\,\&\,(b \succ c)\big) \Longrightarrow a \succ c.$$

Наша задача заключается в том, чтобы по заданным индивидуальным профилям предпочтений для каждого избирателя построить профиль *общественного предпочтения*, т. е. определить ранжировку кандидатов, «справедливую» для всего множества избирателей. Например, пусть из кандидатов $C = \{$*Хренов*, *Редькин*, *Морковьев*$\}$ избиратели $E = \{1, 2, 3\}$[5] выбирают лучшего и профили их предпочтений таковы:

$$\begin{aligned} &\textit{Хренов} && \succ_1 \textit{Редькин} && \succ_1 \textit{Морковьев},\\ &\textit{Редькин} && \succ_2 \textit{Морковьев} && \succ_2 \textit{Хренов}\ ,\\ &\textit{Редькин} && \succ_3 \textit{Морковьев} && \succ_3 \textit{Хренов}\ . \end{aligned}$$

Тогда профиль общественного предпочтения (итоговая ранжировка множества кандидатов C), определённый по большинству голосов:

$$\textit{Редькин} \succ_E \textit{Морковьев} \succ_E \textit{Хренов}.$$

Сразу заметим, что даже для таких простых случаев, когда выбирают из трёх кандидатов три избирателя простым большинством голосов, могут возникнуть проблемы. Рассмотрим, к примеру, такую ранжировку кандидатов:

$$\begin{aligned} &\textit{Хренов} && \succ_1 \textit{Редькин} && \succ_1 \textit{Морковьев},\\ &\textit{Редькин} && \succ_2 \textit{Морковьев} && \succ_2 \textit{Хренов}\ ,\\ &\textit{Морковьев} && \succ_3 \textit{Хренов} && \succ_3 \textit{Редькин}\ , \end{aligned} \tag{38}$$

[4] От греч. δημοκρατια — власть народа.

[5] Автор здесь с трудом удерживается от того, чтобы избирателям также присвоить «говорящие» фамилии.

тогда, большинством в два голоса против одного, имеем

$$\textit{Редькин} \succ_E \textit{Морковьев}, \textit{Морковьев} \succ_E \textit{Хренов}, \textit{Хренов} \succ_E \textit{Редькин},$$

т. е. получаются противоречащие друг другу условия и профиль общественного предпочтения в этом случае не существует. Рассмотренный пример — простейшая иллюстрация парадокса Кондорсе (Marie Jean Antoine Nicolas de Caritat, marquis de Condorcet), открытого в 1785 г.

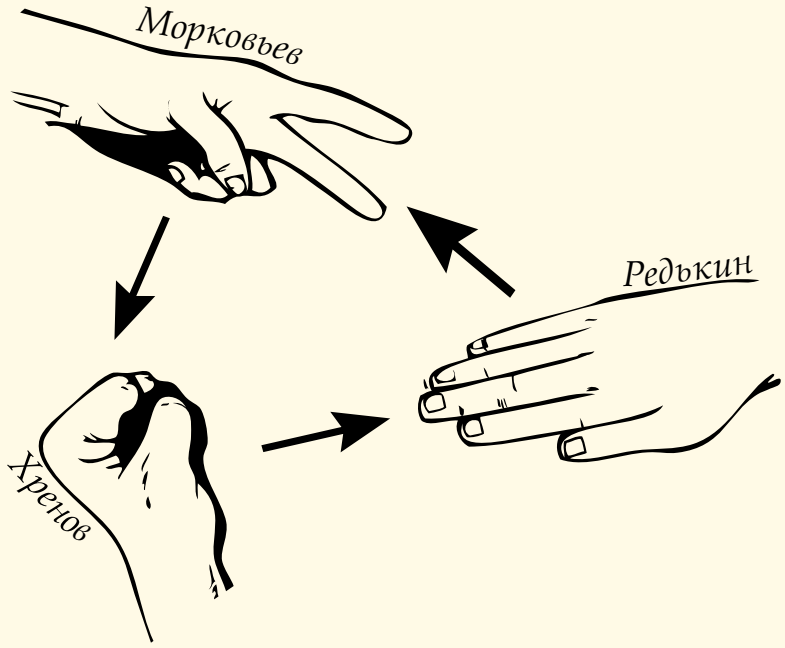


Рис. III.18. *Парадокс Кондорсе: при наличии более двух альтернатив и более двух избирателей коллективная ранжировка альтернатив может быть цикличной (не транзитивна), даже если ранжировки всех избирателей не являются цикличными*

Все известные правила «демократического» голосования имеют существенные недостатки. В качестве иллюстрации приведём два простых примера. Наиболее распространённое правило *простого большинства* голосов, когда побеждает тот кандидат, который набрал наибольшее количество голосов. Тогда в ситуации, когда кандидат Хренов получил 40% голосов, а Редькин и Морковьев — по 30%, победит Хренов, несмотря на то, что большинство избирателей в 60% высказалось против него и, быть может, от всей души ненавидят этого кандидата. Другое, также очень распространённое, правило *голосования в два тура:* если при первом голосовании никто из кандидатов не набрал более 50% голосов, то два претендента, получившие максимальное число голосов, проходят во второй тур голосования, где победитель определяется по правилу простого большинства. Рассмотрим случай, когда кандидаты в первом туре получили ранжировку, как в (38), но при этом вместо одного избирателя голосуют их коалиции, имеющие в своих составах 1-я — 40%, 2-я — $29{,}9\%$ и 3-я — $30{,}1\%$ избирателей. Тогда во второй тур пройдут *Хренов* и *Морковьев*, причём победит последний, собрав 60% голосов (считаем,

что предпочтения избирателей не меняются за всё время выборов и подсчёт голосов абсолютно честный). Однако, если исключить из списка кандидатов *Хренова*, вроде бы, не имеющего шансов на победу, то выиграет *Редькин*. Ясно, что это открывает большие возможности для манипуляций, даже при самом честном подсчёте голосов избирателей.

В связи со сказанным возникает вопрос: возможна ли, хотя бы теоретически, идеальная («справедливая», приемлемая для всех избирателей) избирательная система? Для ответа нужно, во-первых, явно и недвусмысленно указать *необходимые требования* к такой системе. Американский математик К. Эрроу (Kenneth J. Arrow) сформулировал такие требования в виде следующих аксиом [41]:

1. **Универсальность**, universality, unrestricted domain. Для любых кандидатов a, b и любых индивидуальных их ранжировок общественное предпочтение устанавливает либо $a \succ_E b$, либо $b \succ_E a$, либо $a =_E b$.
2. **Единогласие**, unanimity, weak Pareto principle. Если все избиратели считают, что a лучше b, то и в общественном предпочтении a выше b.
3. **Независимость** от посторонних альтернатив, independence of irrelevant alter-natives. Положение любых двух кандидатов зависит только от их положения в индивидуальных профилях предпочтений и не зависит от расположения других кандидатов.
4. **Отсутствие диктатора**, non-dictatorship. Не существует такого избирателя $d \in E$, который мог бы навязать свой выбор всему электорату:

$$x \succ_d y \implies x \succ_E y \qquad \text{для любых } x, y \in C.$$

К. Эрроу доказал (1951 г.), что эта система аксиом *противоречива*, когда $|C| \geqslant 3$ [41]. Следовательно, *идеальная демократическая избирательная система невозможна, даже теоретически*. Особенно печально для «демократов» выглядит (это будет показано ниже), что отказавшись от четвёртой аксиомы мы получим логически непротиворечивую систему требований, что можно трактовать так: демократия — скрытая форма диктатуры.

Для доказательства теоремы Эрроу достаточно построить специальные профили предпочтений, т. к., ввиду аксиомы Универсальности, избирательная система должна «работать» при любых индивидуальных ранжировках. Само доказательство технически не сложно — будем в основном следовать его наиболее простому варианту [42]. Введём определение

Определение 4.1. *Коалицию избирателей D, $D \subseteq E$, назовём решающей для кандидатов a, b, если*

$$\big((a \succ_D b)\,\&\,(b \succ_{E\setminus D} a)\big) \implies a \succ_E b. \tag{39}$$

Другими словами, если избиратели из D считают, что кандидат a лучше b, а все остальные избиратели придерживаются противоположного мнения, то в итоговой ранжировке будет так, как решили члены коалиции D.

В случае, когда соотношение (39) *выполняется для любых кандидатов, коалицию D назовём просто — решающей.*

Заметим, что решающая коалиция, в силу аксиомы Единогласия, во-первых, существует (например, таково всё множество E) и, во-вторых, не может быть пустой (если никто не ставит a выше b, то и общественном предпочтении такого быть не может). Переходя непосредственно к доказательству, обозначим M *минимальную по количеству* избирателей решающую коалицию и покажем, что:

a. Минимальная решающая коалиция M состоит из одного избирателя d.

b. Множество $\{d\}$ — решающая коалиция.

c. Избиратель d — диктатор.

Рис. III.19. *«Что такое демократия?» — «Демократия — это свобода выбирать своих собственных диктаторов»*

a. Минимальная решающая коалиция M состоит из одного избирателя d. Пусть коалиция M является минимальной решающей относительно кандидатов a, b и избиратель $d \in M$. Покажем, что либо коалиция $\{d\}$, либо коалиция $M \setminus \{d\}$ являются решающими и, значит (ввиду минимальности M), $M = \{d\}$. Для этого рассмотрим следующие профили предпочтений

$$\begin{array}{llll} a & \succ_{\{d\}} & b & \succ_{\{d\}} \quad c, \\ c & \succ_{M\setminus\{d\}} & a & \succ_{M\setminus\{d\}} \quad b, \\ b & \succ_{E\setminus M} & c & \succ_{E\setminus M} \quad a. \end{array} \tag{40}$$

Тогда в общественном предпочтении $a \succ_E b$, поскольку так высказались все члены M, а остальные избиратели — наоборот, и расположение кандидата c по аксиоме Независимости никак на это не влияет.

Далее, могут быть две возможности:

— $b \succ_E c$, тогда, в силу $a \succ_E b$, выводим (транзитивность), что $a \succ_E c$. Это значит (см. (40)), что коалиция $\{d\}$ — решающая для a, c;

— $c \succ_E b$, тогда коалиция $M \setminus \{d\}$ — решающая (см. (40)), что противоречит минимальности M.

b. $\{d\}$ — решающая коалиция. Рассмотрим профили

$$\begin{aligned} a &\succ_{\{d\}} b \succ_{\{d\}} x, \\ b &\succ_{E\setminus\{d\}} x \succ_{E\setminus\{d\}} a. \end{aligned}$$

Т. к. $\{d\}$ — решающая коалиция, то $a \succ_E b$, и, в соответствии с аксиомой Единогласия $b \succ_E x$, следовательно (транзитивность), $a \succ_E x$. В силу Независимости полученный результат не зависит от b и, поэтому, справедлив при произвольном x. Не трудно аналогичным образом построить соответствующие профили так, чтобы для произвольного кандидата y было $y \succ_E a$. Следовательно, если $\{d\}$ образует решающую коалицию для какой-либо пары, то эта одноэлементная коалиция будет решающей и для любой пары, в которой один из её кандидатов заменён другим. Отсюда уже прямо вытекает, что $\{d\}$ — решающая коалиция.

Таким образом, избиратель d может навязать электорату своё мнение, если все остальные избиратели голосуют *в точности наоборот*. Для доказательства того, что d является диктатором[6], теперь достаточно показать его независимость от мнения остальных избирателей, то есть, что мнение d относительно ранжировки, скажем, кандидатов a, b всегда будет достаточным для того, чтобы и в коллективном мнении было точно так же.

c. Избиратель d — диктатор. Пусть избиратели, голосующие так же (относительно предпочтения a по сравнению с b), как d, образуют множество P, все несогласные — множество Q. Множества P, Q могут быть пустыми. Рассмотрим профили предпочтений

$$\begin{aligned} a &\succ_{\{d\}} c \succ_{\{d\}} b, \\ c &\succ_P a \succ_P b, \\ c &\succ_Q b \succ_Q a. \end{aligned} \tag{41}$$

Тогда в коллективном предпочтении $a \succ_E c$ в силу того, что $\{d\}$ — решающая коалиция, и $c \succ_E b$ (Единогласие). Поэтому (транзитивность) $a \succ_E b$ — именно так, как высказался d. Следовательно, d — диктатор и, таким образом, теорема Эрроу полностью доказана.

[6]Подчеркнем, что в данном случае диктатор имеет возможность навязывать свое мнение всем остальным, «оставаясь в рамках закона», то есть не нарушая никакой из «демократических аксиом» — стр. 79.

4.2. Модель коррупции

Давняя неотъемлемая черта любого государства — коррупция[7] — является следствием неразрешимого логического противоречия: наделением чиновников властью и стремлением ограничить их доход лишь законным, фиксированным жалованием. Государству для реализации политических решений правящей верхушки всегда[8] был необходим аппарат профессиональных администраторов, наделенных той или иной долей

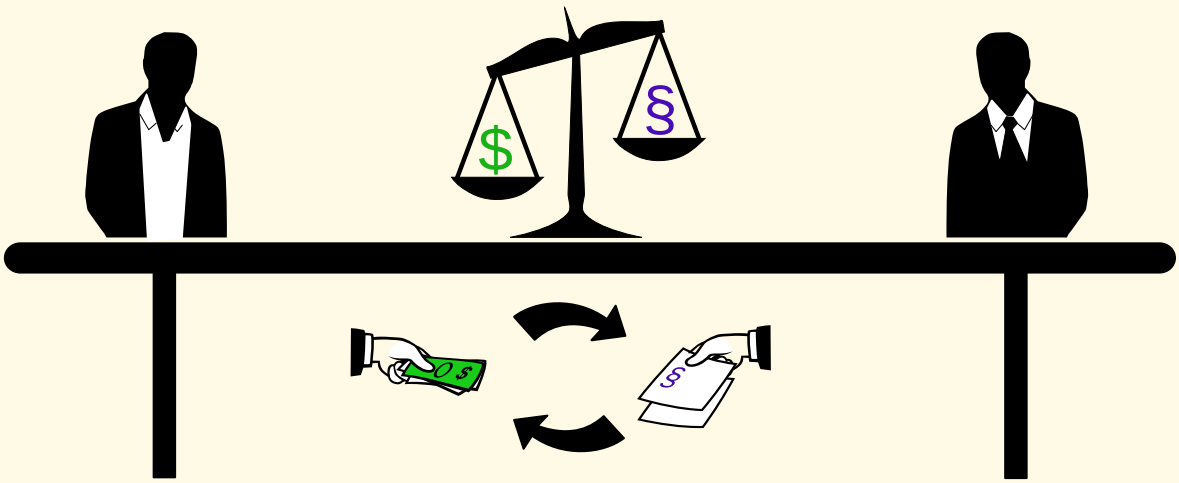


Рис. III.20. *Эффективная схема смазки государственного механизма: не подмажешь — не поедешь*

дискреционной власти[9], имеющих возможность по своему усмотрению определять трактовку законов и принимаемых постановлений правительства. Необходимость в предоставлении таких широких полномочий объясняется тем, что в противном случае должностные лица не смогут принимать адекватные решения в приемлемые сроки, особенно — в сложных, быстро меняющихся и трудно предсказуемых условиях.

Коррупционная сделка является услугой: коррупционер, используя данные ему властные полномочия, совершает (не совершает) некоторые действия, получая взамен *ренту*[10] — деньги, имущество или другие

[7] От лат. corruptio — подкуп, порча. В связи с бурным расцветом демократии, общечеловеческих ценностей, разгулом толерантности и невиданным ранее ростом гуманизма и социальной справедливости в нашей стране с начала 90-х годов XX века, излишне приводить *определение* коррупции.

[8] Самые ранние упоминания о коррупции появились с зарождением письменности и встречаются уже в архивах древнего Вавилона, согласно которым царь шумерского города-государства Лагаша Урукагина приблизительно 4,5 тыс. лет тому назад реформировал государственное управление, пытаясь пресечь злоупотребления чиновников и судей [44, стр. 34].

[9] От фр. discretionnaire — зависящий от личного усмотрения.

[10] Рента (фр. rente, от лат. reddere — возвращать, отдавать) — доход с капитала, облигаций, имущества, земли. Здесь рента — незаконный доход с занимаемой должности.

услуги. Стратегия коррупционного поведения достаточно быстро принимается большинством администраторов: если лишь один из них вопреки остальным примет решение никогда не вступать в коррупционное соглашение, то он поставит себя в крайне невыгодные условия. В результате, как хорошо известно, такая система становится достаточно устойчивой и постепенно общество (а не только бюрократия) в условиях высокого уровня коррупции закрепляет ее в качестве традиции, что сильно снижает эффективность любых антикоррупционных мер.

Природа коррупции, ее причины и последствия, антикоррупционные меры в настоящее время являются предметом многочисленных дискуссий. Рассмотрим сильно упрощенную модель коррупционного механизма, учитывающую только три параметра: *популярность* правительства, *уровень коррумпированности* его должностных лиц и *уровень противодействия* коррупции со стороны государства и общества.

Опишем, отчасти следуя [47], явление коррупции как динамическую модель с тремя переменными:

- $x = x(t)$ — политическая поддержка (рейтинг, популярность) правительства в момент времени t;
- $y = y(t)$ — коррупционная рента (мзда, величина взяток), которой продажные чиновники владеют в момент t;
- $z = z(t)$ — величина усилий по разоблачению коррупции в момент t, т. е. средняя суммарная активность различных институтов: полициии, судов, прессы, общественных организаций, оппозиционных политических партий и др.

Для этих переменных можно использовать подходящую размерность. Например, x может выражаться в процентах поддержки правящей партии, y — денежная величина взяток (считаем, что взятка услугой также имеет определенное денежное выражение), z — количество условных лиц, вовлеченных в расследование лихоимств.

Построим модель при следующих предположениях:

a. Государство вообще не оказывает давления на антикоррупционные расследования.

b. Изменение популярности $\dot{x}$ политиков зависит от их действий и ожидаемого населением эффекта от этих акций, в совокупности описываемых *ограниченной* функцией, вида

$$\frac{\alpha x}{\beta + x}, \qquad \alpha, \beta = \mathrm{const}\,.$$

c. Количественно разоблачение продажных чиновников пропорционально усилиям по расследованию и величине взяток (согласно пословице: «Плуту да вору — честь по разбору»): $\gamma y z \stackrel{\mathrm{def}}{=} A$, где γ — «антикоррупционный (репрессивный) коэффициент».

d. Изменение величины взяток $\dot{y}$ прямо пропорционально как уже имеющимся нелегальным доходам (Деньги к деньгам льнут, Дай

вору целковых гору — воровать не перестанет), так и популярности политиков (Алтынного вора вешают, а полтинного чествуют), т. е. значению δxy, где δ — «коэффициент жадности» коррупционера, и величине A.

d. На изменение усилий по противодействию коррупции $\dot z$ влияет уровень разоблачений (Легки взятки — тяжелы отдатки) и популярность чиновников, т. е. оно равно $\varepsilon A - \zeta xz$, где ε — «коэффициент злопамятности», ζ — «коэффициент забывания (прощения)» относительно действий продажных чиновников.

Из этих условий следует система уравнений

$$\dot x = \frac{\alpha x}{\beta + x} - \gamma xyz, \tag{42}$$

$$\dot y = \delta xy - \gamma yz, \tag{43}$$

$$\dot z = \varepsilon\gamma yz - \zeta xz. \tag{44}$$

$$x(0) = x_0, y(0) = y_0, z(0) = z_0, \quad \alpha, \beta, \gamma, \delta, \varepsilon, \zeta \geqslant 0. \tag{45}$$

Полученная модель существенно нелинейна, что делает невозможным аналитическое решение (42)–(44) и сильно затрудняет анализ. Поэтому проведем численные эксперименты. На рис. III.21 показано, как эффек-

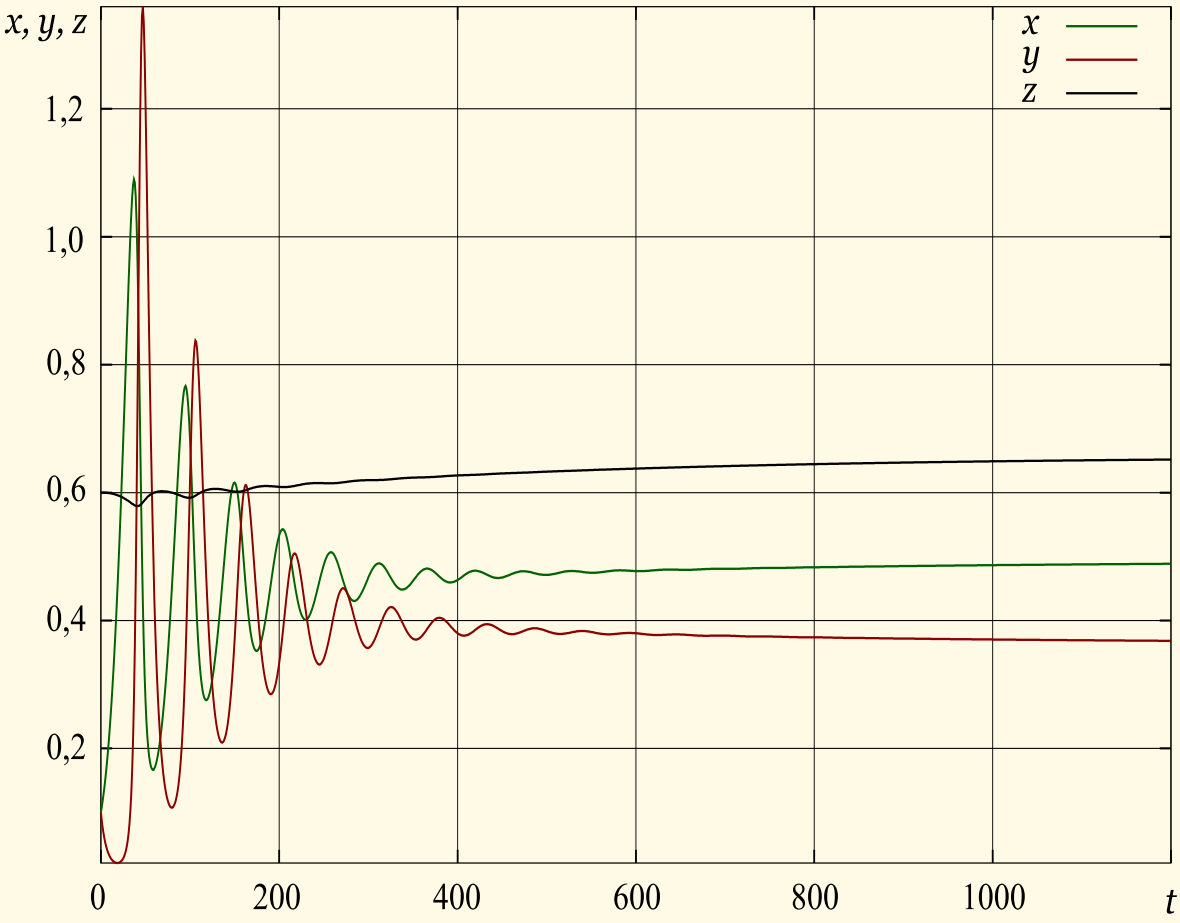


Рис. III.21. *Модель коррупции,* $\alpha = 0{,}1$, $\beta = 0{,}9$, $\gamma = 0{,}3$, $\delta = 0{,}4$, $\varepsilon = 0{,}009$, $\zeta = 0{,}02$, $x_0 = 0{,}1$, $y_0 = 0{,}01$, $z_0 = 0{,}6$

тивная борьба (антикоррупционный коэффициент $\gamma = 0{,}3$) гасит значительные вначале по амплитуде волны коррупции (коэффициент жадности $\delta = 0{,}4$), постепенно снижая её уровень и мало-помалу увеличивая популярность властей. В случае отсутствия противодействия коррупции ($\gamma = 0$), она неограниченно и очень быстро растет — см рис. III.22, несмотря на довольно низкий её изначальный уровень ($y_0 = 0{,}01$).

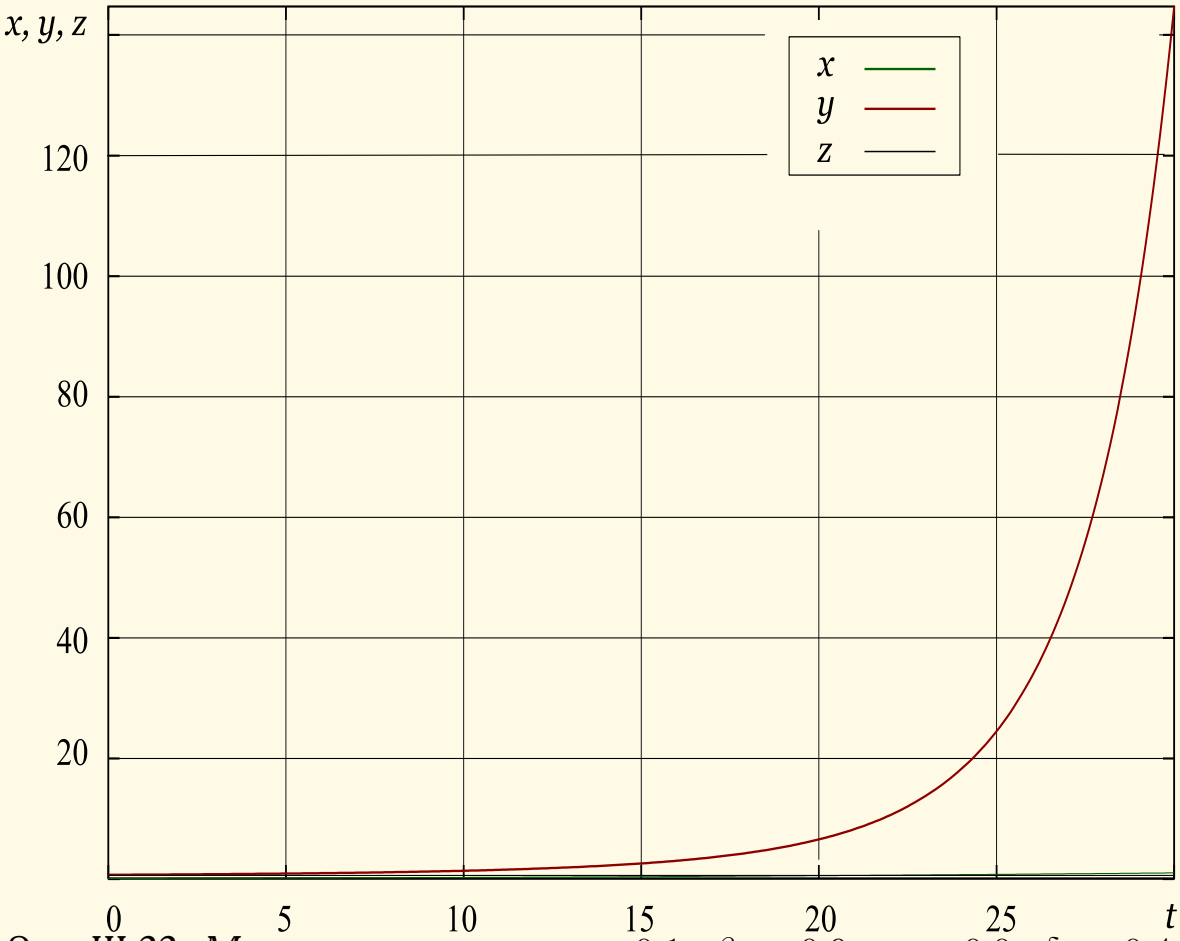


Рис. III.22. *Модель коррупции,* $\alpha = 0{,}1$, $\beta = 0{,}9$, $\gamma = 0{,}0$, $\delta = 0{,}4$, $\varepsilon = 0{,}009$, $\zeta = 0{,}02$, $x_0 = 0{,}1$, $y_0 = 0{,}01$, $z_0 = 0{,}6$

Заметим, что если, с целью *предупреждения коррупции* попытаться формализовать любые действия чиновника, то она, конечно, заметно сократится, но возрастет неадекватность управления и придется увеличить затраты на контролирующий аппарат. Поэтому полная ликвидация коррупции невыгодна для государства и, по-видимому, разумно поддерживать некий *оптимальный* уровень коррупции, соответствующий наименьшим суммарным потерям и признаваемый обществом допустимым [45]. Если в нашей модели уровень коррупции взять постоянным ($\dot{y} = 0$, $y(0) = y_0$, см рис. III.23), то уравнения (42)–(44) примут вид

$$\dot{x} = \frac{\alpha x}{\beta + x} - \gamma y_0 x z, \tag{46}$$

$$\dot{z} = \varepsilon\gamma y_0 z - \zeta x z. \tag{47}$$

$$x(0) = x_0, y(0) = y_0, z(0) = z_0, \quad \alpha, \beta, \gamma, \delta, \varepsilon, \zeta \geqslant 0. \tag{48}$$

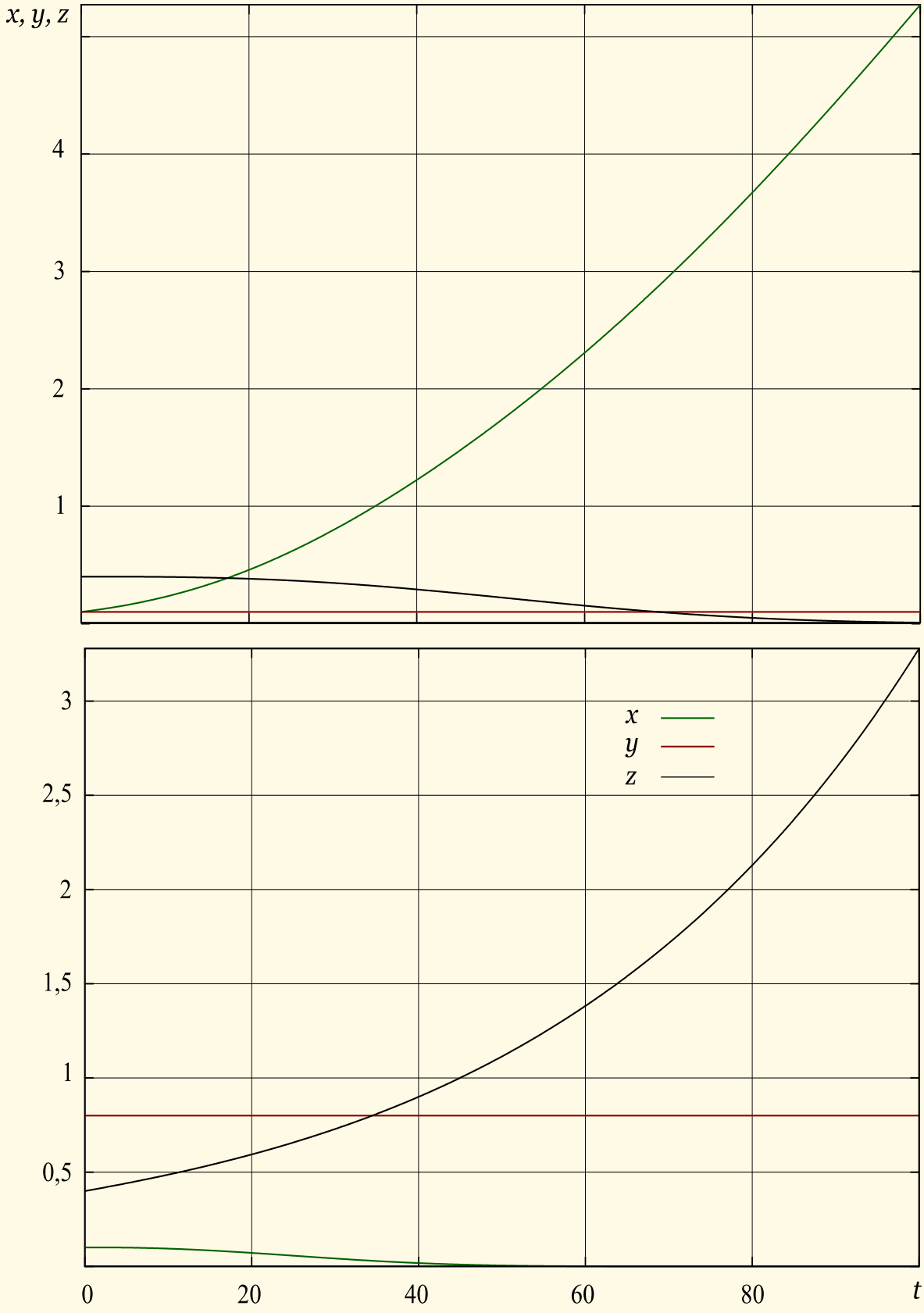


Рис. III.23. *Модель с постоянным уровнем коррупции,* $\alpha = 0{,}1$, $\beta = 0{,}9$, $\gamma = 0{,}3$, $\varepsilon = 0{,}09$, $\zeta = 0{,}02$, $x_0 = 0{,}1$, $y_0 = 0{,}1$ *(верхний рис.),* $y_0 = 0{,}8$ *(нижний рис.),* $z_0 = 0{,}4$

Таким образом, при постоянном уровне коррупции как видно из рис. III.23 и численного эсперимента на модели (46), (47), (48), популярность правительства возрастает при относительно низком уровне коррупции ($y_0 = 0{,}1$) и стремительно падает при высоком уровне ($y_0 = 0{,}8$). Соотвественно, «накал разоблачений» z в первом случае быстро сходит на нет, и растет во втором случае.

Как показано на рис. III.24, усиление репрессий ($\gamma = 0{,}9$) позволяет обуздать и значительную ($y_0 = 1{,}0$, $\delta = 0{,}7$) непостоянную коррупцию.

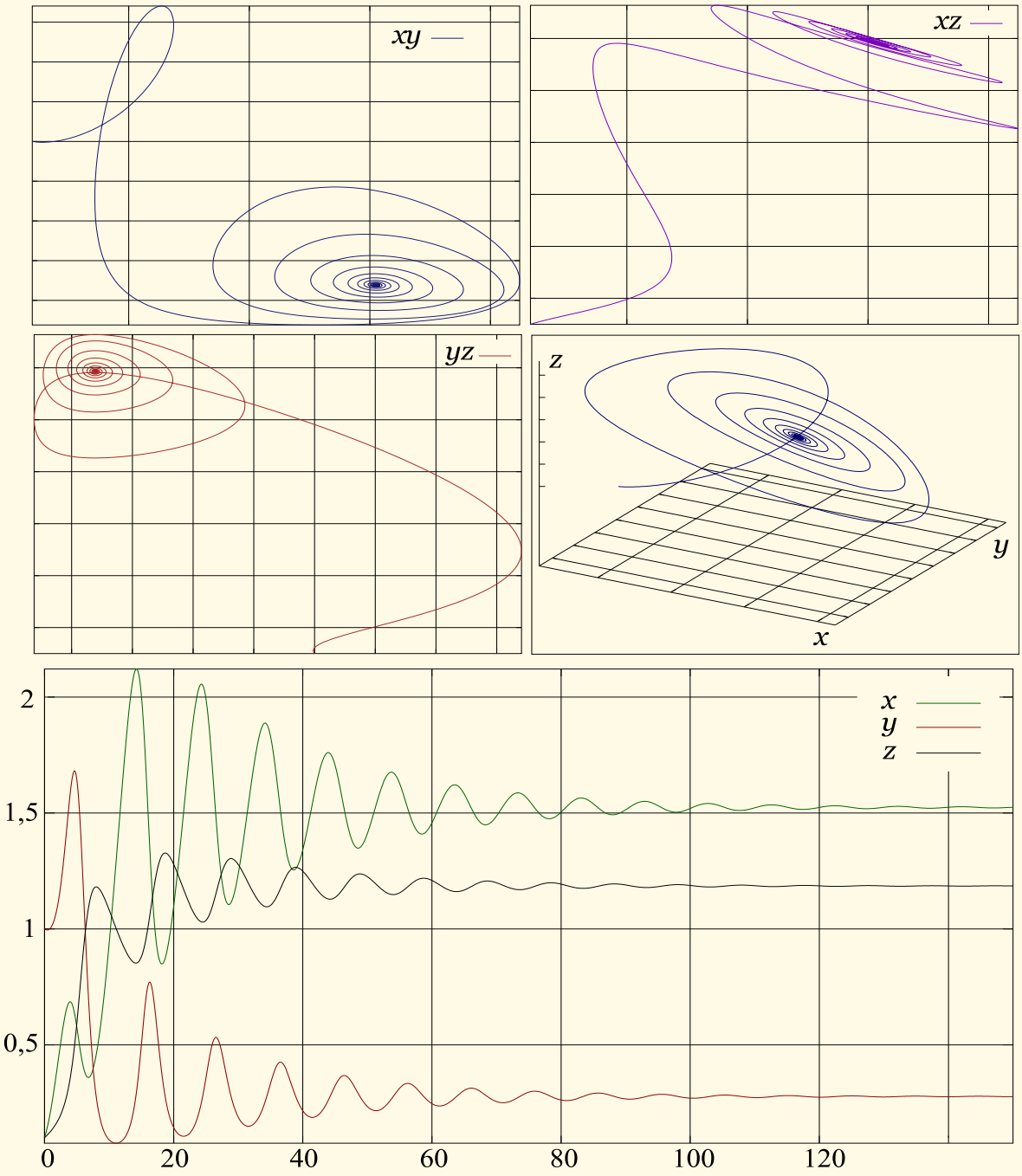


Рис. III.24. *Фазовые портреты и графики динамической системы* (42)–(44), $\alpha = 0{,}6$, $\beta = 0{,}5$, $\gamma = 0{,}9$, $\delta = 0{,}7$, $\varepsilon = 0{,}4$, $\zeta = 0{,}1$, $x_0 = 0{,}1$, $y_0 = 1{,}0$, $z_0 = 0{,}1$

Таким образом, стабилизация уровня коррупции на низком уровне ($y \approx 0{,}2$) при начальном значении $y_0 \approx 1{,}0$ и высокой «жадности» чиновников-взяточников ($\delta = 0{,}7$), приводящая к стабильно высокой популярности правительства ($x \approx 1{,}5$), может достигаться при жестком подавлении коррупции ($\gamma = 0{,}9$) и поддержании стабильно больших усилий по её разоблачению ($x \approx 1{,}2$) — см. рис. III.24.

Вопрос III.13. Рассматривая упрощение модели (42)–(45) вида

$$\dot{x} = \frac{\alpha x}{\beta + x} - \gamma y_0 z_0 x, \qquad x(0) = x_0, \tag{49}$$

т. е. считая постоянными как величину коррупции ($y(t) = y_0 = \text{const}$), так и уровень противодействия ($z(t) = z_0 = \text{const}$), найти условия роста/падения популярности правительства.

4.3. Модель территориальной динамики государства

Как известно, одной из важнейших внешних функций государства является *оборона*, т. е. защита своей страны и, по мере возможности, грабеж и аннексия чужих территорий. В этой связи ряд ученых интересует вопрос: почему некоторые государства начинают успешно расширяться и становятся империями, а другие государства разрушаются?

Историки и социологи предлагают различные ответы на этот вопрос, от конкретных объяснений, учитывающих уникальные характеристики определенного государства, до обобщенных теорий социальной или этнической динамики [48], [49]. В последнее время в попытках решения этой и других исторических задач постепенно начинают применяться математические методы [46], [50], [51].

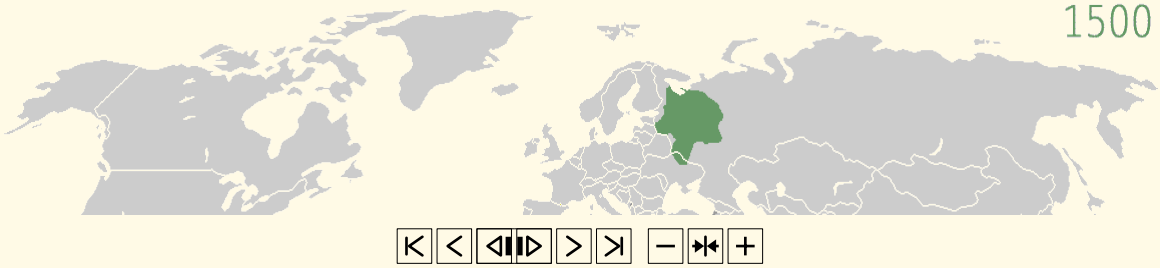


Рис. III.25. *Территориальная динамика России/СССР, 1500–2014 гг*

Построим модель [50], сосредоточившись на динамике размеров государства, поскольку значительная часть исторических источников посвящена территориальной экспансии одних стран против других, обычно связанной с войнами, как правило, достаточно подробно описанными в летописях и хрониках, что способствует оценке адекватности модели. Вначале введем следующие переменные:

- $A = A(t)$ — размер (площадь) территории, которую занимает государство (страна, империя) в момент времени t;
- $R = R(t)$ — геополитические ресурсы (население и другой расходный материал) государства;
- $W = W(t)$ — военная мощь (потенциал) страны.

И будем предполагать, что модельные переменные связаны между собой следующими условиями:

a. Скорость изменения территории $\dot{A}$ государства прямо пропорциональна его военной мощи

$$\dot{A} = c_1 A, \qquad c_1 = \text{const} \geqslant 0.$$

b. Так как главный ресурс любой страны — люди, и упрощённо принимая, что большая площадь страны всегда соответствует большей численности населения, платящего налоги и поставляющего рекрутов для армии, считаем, что ресурсы прямо пропорциональны площади территории

$$R = c_2 A, \qquad c_2 = \text{const} \geqslant 0.$$

c. Государству противостоят вражеские державы с *постоянной* военной мощью c_4. Потенциал страны линейно связан с количеством ресурсов и давлением извне

$$W = c_3 R - c_4, \qquad c_3, c_4 = \text{const} \geqslant 0.$$

d. Значительное расширение государства вызывает трудности, связанные с ослаблением влияния его «центра» на больших расстояниях, — тыловая нагрузка, «бремя империи». Примем, что влияние падает по экспоненте:

$$\exp(-A/h), \qquad h = \text{const} > 0.$$

Предположения **a**–**d** можно записать в виде одного уравнения

$$\dot{A} = cA \exp\left(-\frac{A}{h}\right) - a, \qquad a, c, h = \text{const} > 0,$$

откуда упростив, используя аппроксимацию $e^x \approx 1 + x$, получим

$$\dot{A} = cA\left(1 - \frac{A}{h}\right) - a, \quad A(0) = A_0, \quad a, c, h > 0, \tag{50}$$

где c — коэффициент преобразования ресурсов государства в его военно-политическую мощь, $c = c_1 c_2 c_3$, h — коэффициент «проецирования силы центра на расстояние», a — геополитическое давление враждебных государств, $a = c_1 c_4$.

Наша модель пока не учитывает одного важного фактора — «внутренней силы» государства, которая основана на сплоченности его граждан, их способности действовать совместно в интересах страны, их *коллективной* солидарности. Действительно, множество исторических примеров подтверждают, что если население страны не способно к совместным действиям (каковые часто требуется организовывать *в ущерб личным, эгоистическим интересам*), то такое государство обречено на поражение, независимо от имеющихся ресурсов. С другой стороны, в истории зафиксированы противоположные случаи, когда высокосплоченные этносы с поначалу незначительными ресурсами значительно расширяли свои территории. Данный фактор коллективной солидарности

называют *асабией*[11], обозначим его $S = S(t)$ и будем предполагать, что коэффициент c в уравнении (50) является линейной функции асабии

$$c = c_0 S, \qquad c_0 = \text{const} > 0, \quad 0 \leqslant S \leqslant 1.$$

Значение асабии $S = 0$ означает полную неспособность граждан к совместным действиям, $S = 1$ — максимальный уровень солидарности.

В качестве закона изменения асабии выберем логистический:

$$\dot{S} = r(A)S(1 - S),$$

где коэффициент r линейно зависит от размера территории. Асабия минимальна в центре и максимальна в приграничной полосе, отделяющей страну от враждебных держав [50], поэтому положим

$$r = r_0 \left(1 - \frac{A}{2b}\right),$$

где b — ширина зоны границы.

Таким образом, получаем описание модели системой уравнений

$$\dot{A} = c_0 A S \left(1 - \frac{A}{h}\right) - a, \tag{51}$$

$$\dot{S} = r_0 \left(1 - \frac{A}{2b}\right) S(1 - S) \tag{52}$$

$$A(0) = A_0, S(0) = S_0, \quad A > 0, \quad 0 \leqslant S \leqslant 1. \tag{53}$$

Анализ модели. Точками покоя системы (51)–(53) являются

$$\overline{A} = 2b, \quad \overline{S} = \frac{ah}{2bc_0(h - 2b)}; \tag{54}$$

$$\overline{A} = \frac{c_0 h \pm \sqrt{c_0 h(c_0 h - 4a)}}{2c_0}, \quad \overline{S} = 1. \tag{55}$$

Для первой стационарной точки якобиан системы (51)–(53)

$$\mathbf{J} = \begin{bmatrix} c_0 \overline{S} \left(1 - \dfrac{4b}{h}\right) & 2c_0 b \left(1 - \dfrac{2b}{h}\right) \\ \dfrac{r_0}{2b} \overline{S} \left(1 - \overline{S}\right) & 0 \end{bmatrix}.$$

Характеристическое уравнение

$$\det\left(\mathbf{J} - \lambda \mathbf{I}\right) = -\lambda \left(c_0 \overline{S} \left(1 - \frac{4b}{h}\right) - \lambda\right) + c_0 r_0 \left(1 - \frac{2b}{h}\right) \overline{S} \left(1 - \overline{S}\right) =$$

$$= \lambda^2 - c_0 \overline{S} \left(1 - \frac{4b}{h}\right) \lambda + c_0 r_0 \left(1 - \frac{2b}{h}\right) \overline{S} \left(1 - \overline{S}\right) = 0$$

[11] Термин введен арабским историком и политическим деятелем IV в Ибн Халдуном.

имеет положительные коэффициенты и, следовательно, решение в точке (54) устойчиво (см. стр. 159), если

$$\left(1-\frac{4b}{h}\right)<0, \quad \left(1-\frac{2b}{h}\right)>0,$$

или

$$\frac{h}{4}<b<\frac{h}{2}. \tag{56}$$

При $b<h/4$ равновесие неустойчиво. Динамика территории состоит из одного цикла взлета/упадка (см. рис. III.26). Первоначально и A, и S увеличиваются, но когда $A>2b$, S начинает снижаться. Территориальная экспансия останавливается, потому что империя стал-

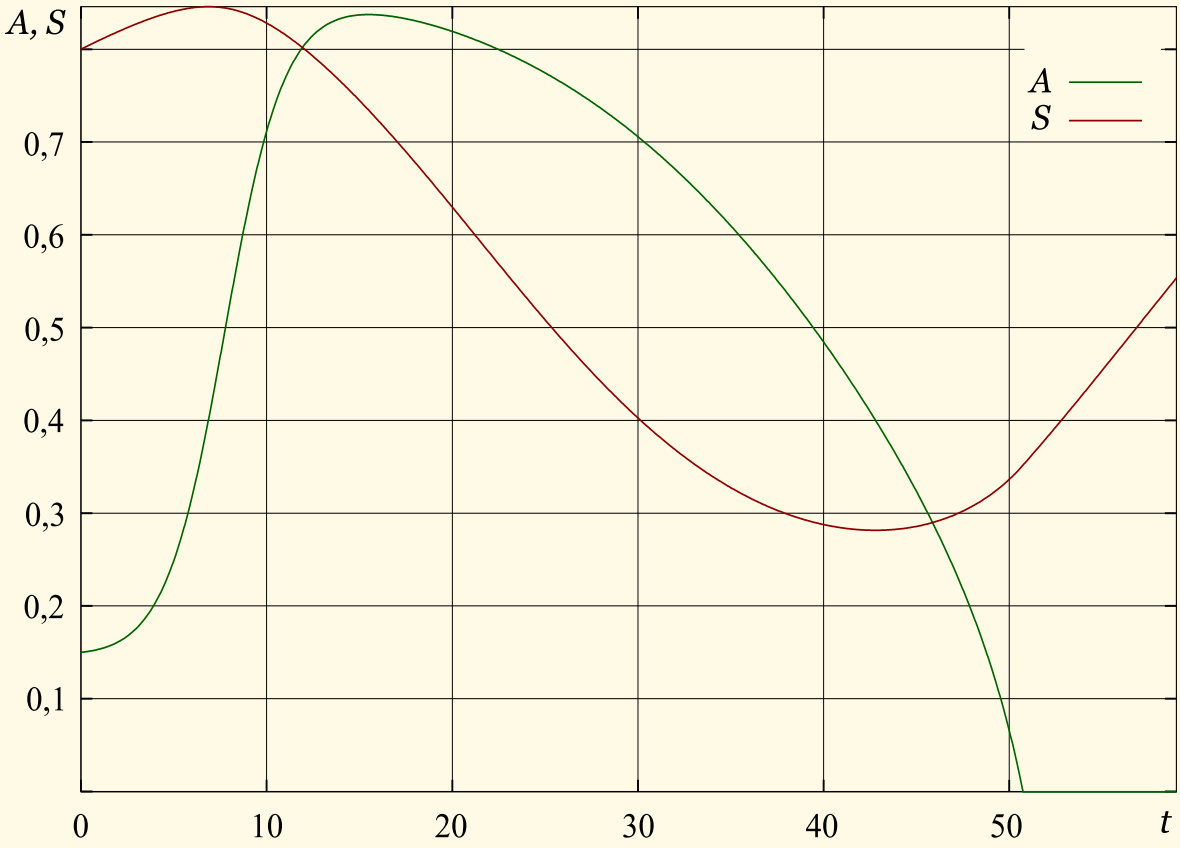


Рис. III.26. *Модель территориальной динамики,* $a=0{,}1$, $b=0{,}2$, $c_0=1$, $r_0=0{,}1$, $h=1$. *Неустойчивое решение:* $b<h/4$

кивается с тыловыми ограничениями, и асабия S продолжает уменьшаться, так как центральные области империи доминируют над периферией. Малая величина S приводит к тому, что империя не может противостоять геополитическому давлению извне. В результате территория A начинает сжиматься, сначала медленно, а затем все быстрее, так как ресурсы государства все более сокращаются. В некоторой точке асабия снова начинает увеличиваться (когда A падает ниже $2b$), но это происходит слишком поздно: уменьшение территории A, а с ней и ресурсов, подавляет любое увеличение S и империя рушится ($A\to 0$).

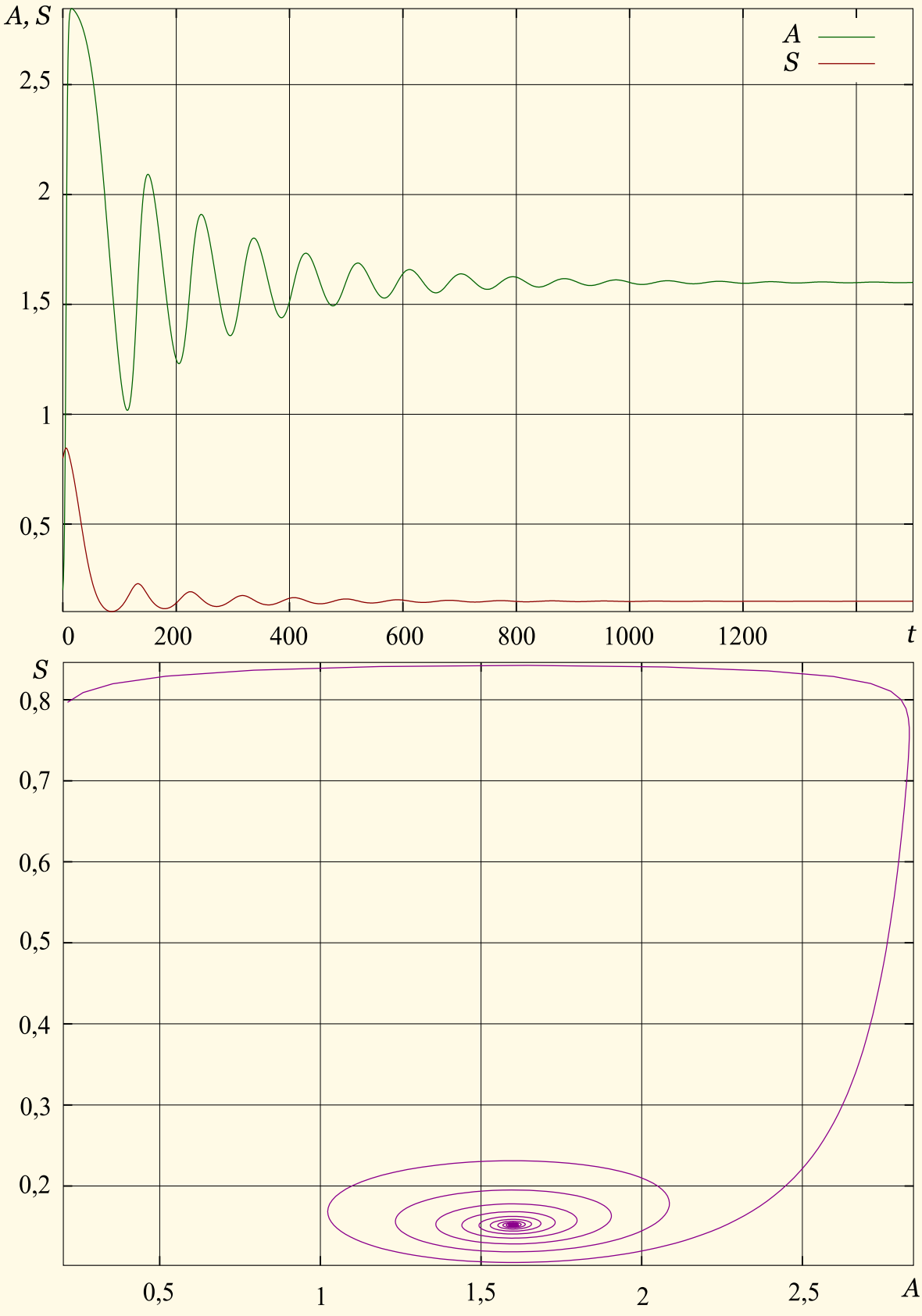


Рис. III.27. *Модель территориальной динамики,* $a = 0{,}1$, $b = 0{,}8$, $c_0 = 0{,}9$, $r_0 = 0{,}09$, $h = 3$. *Устойчивое решение:* $h/4 < b < h/2$

Если $h/2 > b > h/4$, то равновесие устойчиво к малым возмущениям (см. рис. III.27). Рост A увеличивает и область в центре, заставляя значение асабии снижаться, что оттесняет границу империи назад. С другой стороны, уменьшение A увеличивает S, что толкает границу вперед. Если $b > h/2$, то равновесие системы неустойчиво: незначительное усиление внешнего давления ($a = 0{,}105$ вместо $a = 0{,}104$) ведет к катастрофически быстрой утрате территории (см. рис. III.28).

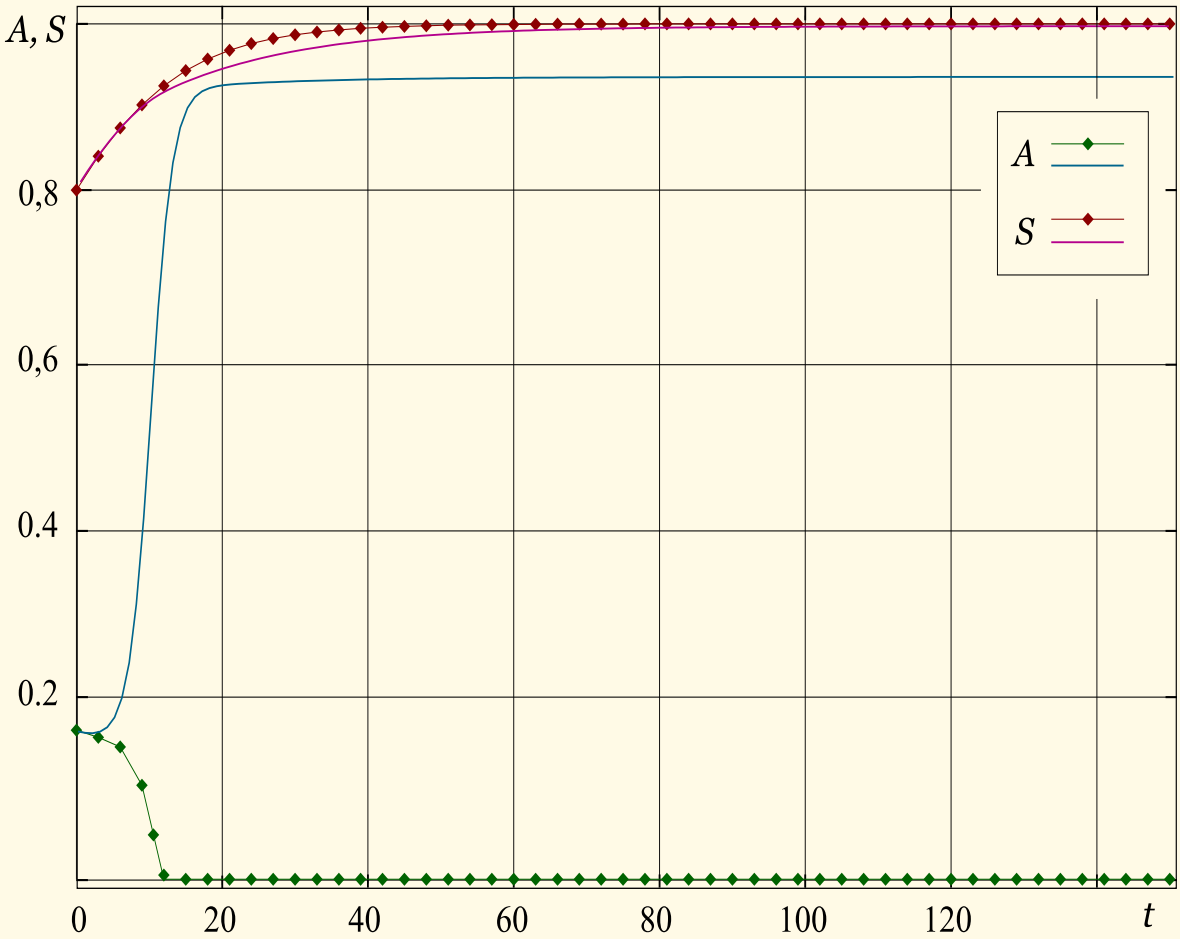


Рис. III.28. *Модель территориальной динамики,* $a = 0{,}105$ *(ромбы) и* $a = 0{,}104$ *(линии без ромбов),* $b = 0{,}99$, $c_0 = 1$, $r_0 = 0{,}1$, $h = 1$. *Неустойчивое решение:* $b > h/2$

Для стационарной точки (55), как легко проверить, характеристическое уравнение

$$\lambda^2 - c_0\left(1 - \frac{2\overline{A}}{h}\right)\lambda = 0$$

имеет один нулевой корень и, следовательно (см. стр. 159), в этом случае решение (для точки, где перед радикалом стоит плюс) лежит на границе устойчивости при $\overline{A} > h/2$.

5. Информатика

Информатика дала нам право на полную и безопасную манию величия.

Бернард Вербер. Революция муравьев

Многие не сведущие в математике люди думают, что поскольку назначение аналитической машины Бэббиджа — выдавать результаты в численном виде, то природа происходящих в ней процессов должна быть арифметической и численной, а не алгебраической и аналитической. Но они ошибаются.

Августа Ада, графиня Лавлейс

5.1. L-системы

L-системы (Lindenmayer-Systems) обычно употребляются для моделирования различных процессов, как правило, связанных с эволюцией каких-либо биологических и иных природных объектов: кристаллов, раковин моллюсков, пчелиных сот и т. д. L-система определяется специальным очень простым формальным языком (см. определение на стр. 183), который представляет собой набор правил, задаваемых текстовой записью, и предназначенных для графического представления объекта на плоскости. Отличительной особенностью языка L-систем является возможность представления визуально довольно сложных систем посредством простых команд [19].

Можно считать, что графические построения L-систем реализуются при помощи специального *исполнителя*, который в каждый момент времени «знает» свои координаты, направление движения и длину отрезка (длина отрезка постоянна, её можно выбрать произвольно на каждой итерации), на величину которого он может переместиться. Управляется исполнитель следующими командами:

F — рисовать заданный отрезок в заданном направлении;
f — передвинуться в заданном направлении на заданный отрезок расстояния без рисования;
\+ — изменить текущее направление против часовой стрелки на заданный угол;
− — изменить текущее направление по часовой стрелке на заданный угол;
| — реверс (поворот на 180°);
[— поместить в стек[1] текущие координаты и направление;
] — вытолкнуть из стека координаты и направление и сделать их текущими;
{ — начало записи L-системы;
} — конец записи L-системы;
; — комментарий до конца строки.

Спецификация входных данных для описания L-систем представляет собой простой текстовый файл. В начале, до символа «{», принято задавать имя объекта. После символа начала записи задаётся текущий угол — «Angle n», угол равен $360°/n$. Затем — т. н. *инициатор*, аксиома «Axiom S», где S — начальное значение строки команд, включающей один или несколько текстовых символов, которая далее будет содержать команды исполнителю. Символ «=» означает подстановку правой части вместо левой. Сама операция подстановки выполняется столько раз, сколько задаст пользователь (параметр Order). Начальное направление по умолчанию равно $0°$.

Начальным состоянием L-системы является её аксиома. При дальнейшем развитии строка S, описывающая состояние, будет меняться. Развитие L-системы происходит циклически Order раз. В каждом цикле строка просматривается от начала к концу, символ за символом, и для каждого из них выполняется соответствующая подстановка, если он задан в аксиоме. Символы, не входящие в S, не меняются.

Пример 1. Кривая Коха.

```
koch {
      Angle  6          ;угол  = PI/Angle
      Axiom  F          ;стартовый  символ
      F= F+F--F+F
      }
```

При Order = 1 строка команд имеет вид F+F−−F+F, при Order = 2 F+F−−F+F + F+F−−F+F −− F+F−−F+F + F+F−−F+F и т. д. Соответственно будет меняться изображение кривой Коха, как показано на рисунке

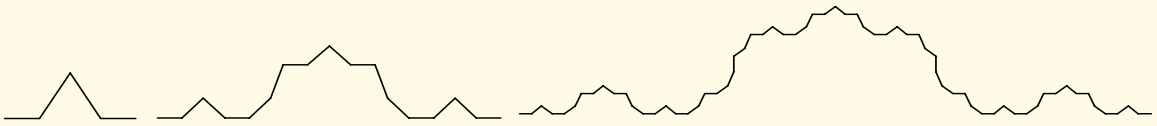


Рис. III.29. *Кривая Коха*

Кривая Коха является классическим примером геометрического фрактала и обладает множеством замечательных свойств. Алгоритм построения: исходный единичный отрезок разделяется на три равные части и средний интервал заменяется равносторонним треугольником без основания. В результате получится ломаная, состоящая из четырех звеньев

[1] Стеком (англ. stack — стопка, пачка) называют структуру, в которой накапливают некоторые элементы, причём обязательно условие: элементы из стека можно доставать только в порядке обратном порядку их помещения в стек — т. н. принцип LIFO (англ. Last In — First Out, «последним пришёл — первым вышел»). На языках высокого уровня стек обычно реализуется в виде односвязного или двусвязного списка.

длины $1/3$. На следующем шаге операция повторяется для каждого из четырёх получившихся звеньев и т. д. Предельная кривая, после бесконечного числа шагов, является кривой Коха.

Пример 2. «Дерево» (Order = 1, 2, 3, 4).

```
tree {
    Angle  16                    ;угол  = PI/Angle
    Axiom  F                     ;стартовый символ
    F= FF-[-F+F+F]+[+F-F-F]
    }
```

Рис. III.30. *L-дерево*

Пример 3. «Цветок» (Order = 4).

```
flower {
        Angle  6
        Axiom  F
        F= F+F-F+F
        }
```

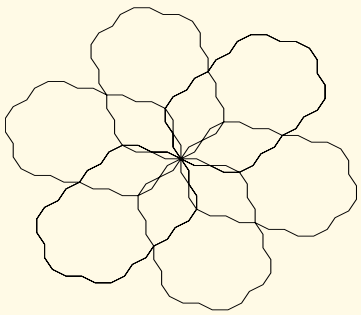


Рис. III.31. *L-цветок*

Пример 4. «Солнце» (Order = 4, 5, 6).

```
sun{
    Angle  18
    Axiom  f
    f=f+[+FF−FF]−−[FF+FF]f
    }
```

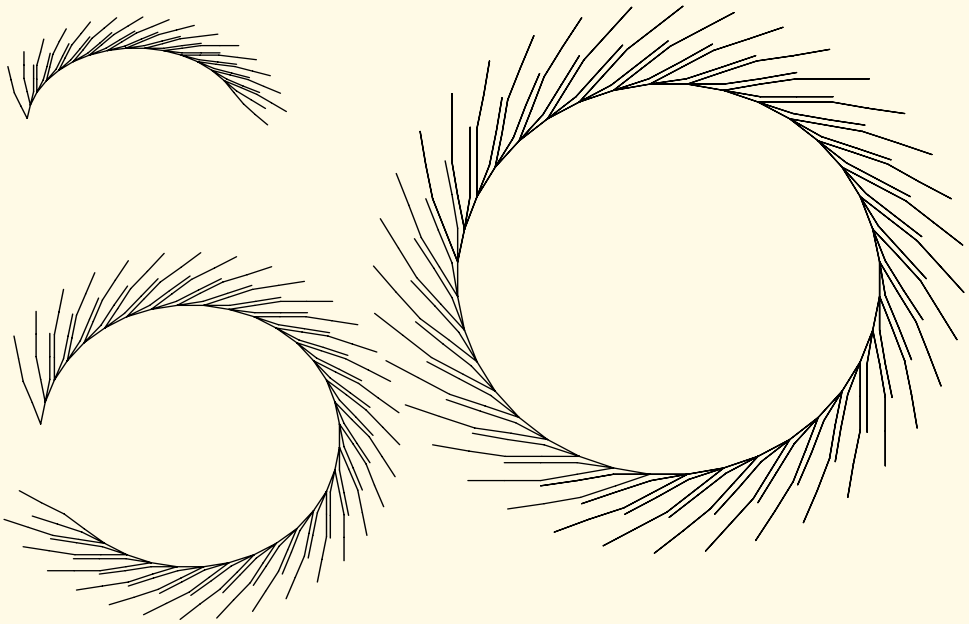


Рис. III.32. *L-солнце*

Пример 5. «Орнамент» (Order = 4).

```
ornament{
        Angle  4
        Axiom  F
        F=−FF−−−FF+
        }
```

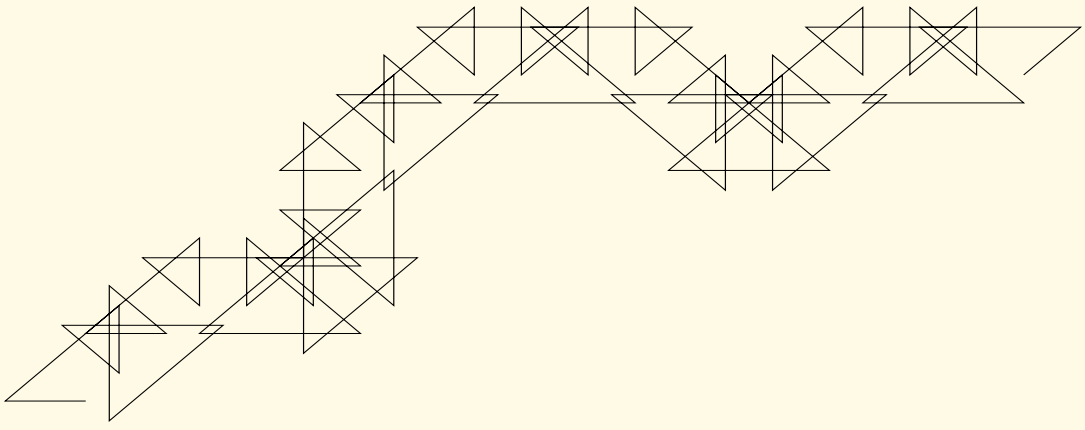


Рис. III.33. *L-орнамент*

Пример 6. «Куст» (Order = 2, 3, 4).

```
bush{
      Angle  9
      Axiom  F
      F= F[+F]F[−F]+F
      }
```

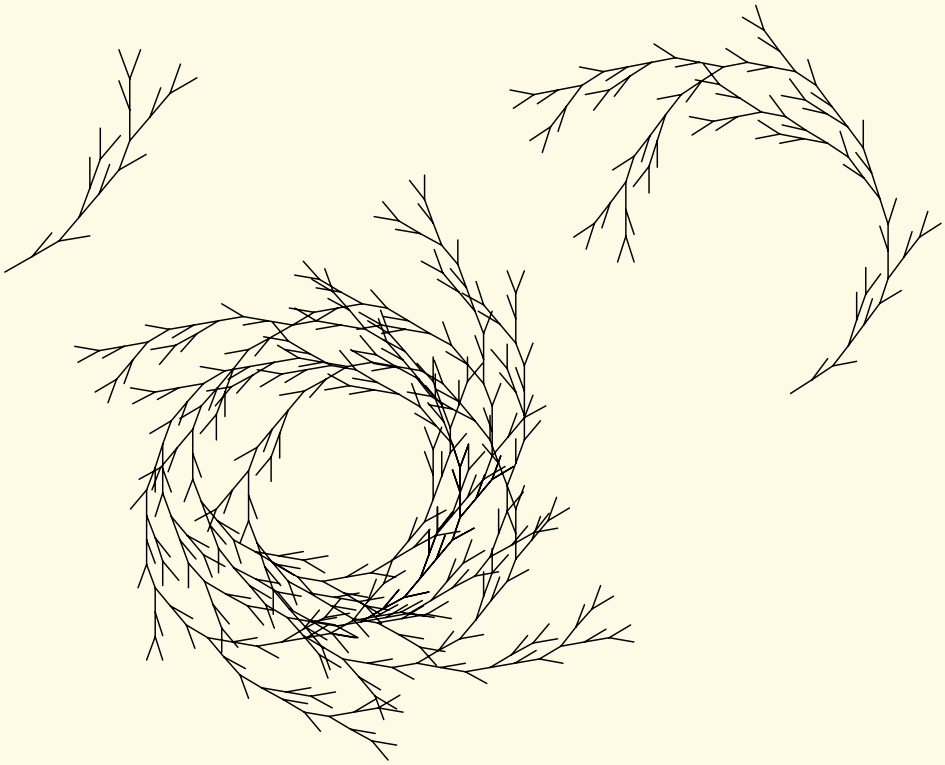


Рис. III.34. *L-куст*

5.2. Транслятор с языка L-систем на PostScript

Для качественного визуального представления и хранения изображений, построенных на основе L-систем, необходимо перевести их в подходящий графический формат. В связи со спецификой языка L-систем эту задачу можно решить используя *векторный* формат, например, EPS (Encapsulated PostScript, разновидность PostScript). В данном случае особенно важны преимущества векторных форматов графики над растровыми, такие как: простота масштабирования изображения, относительно малый размер файлов и др.

Основы языка PostScript. Язык PostScript был разработан фирмой Adobe Systems в 1982 г и создавался в качестве средства описания вида текста, чертежей и изображений на печатаемой странице так, чтобы это описание не зависело от устройства, на котором страница будет воспроизведена.

Язык PostScript содержит около 250 операторов. Около трети средств языка предназначены для графики, остальная часть — это процедурный

язык программирования [86]. Файлы PostScript (PS, EPS) представляют собой обычный текст, который одновременно является исходной программой для интерпретатора и одним из форматов *векторной* графики. Интерпретатор PostScript способен принять на вход этот файл с описанием страницы и преобразовать его в *растровую* форму, которая затем выводится на печать или на экран монитора. В настоящее время наиболее распространён интерпретатор Ghostscript ([87], портирован на платформы MS Windows, OS/2, GNU Linux).

Формат EPS является разновидностью PostScript. Главные отличия состоят в том, что EPS-файлы могут иметь только одну страницу и предназначаются в основном для хранения векторных изображений. В соответствии с [86] минимальный EPS-файл имеет вид

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox:  minX minY maxX maxY
%%Creator: L2eps
%%Title: визуализация L-системы
%%Pages: 1
%%EndComments
newpath
%операторы
 stroke
 closepath
%%Page: 1 1
%%EOF
```

где minX, minY, maxX, maxY — минимальные и максимальные координаты изображения.

В связи со спецификой языка описания L-систем из всего многообразия возможностей PostScript нам понадобится только следующие два оператора, задающие соответственно перемещение «пера» в точку с координатами (x,y) и рисование отрезка от текущего положения пера до точки с координатами (x,y):

```
   x   y    moveto
   x   y     lineto
```

Транслятор (англ. translator — переводчик) — специальная программа, преобразующая исходную программу, написанную на одном алгоритмическом языке, в программу на другом языке. Транслятор может быть двух типов:

Компилятор (англ. compile — составлять, собирать, накапливать) — программа, переводящяя исходную программу в эквивалентную ей объектную программу, записанную, как правило, на некотором машинном языке.

Интерпретатор (англ. interpreter — переводчик, истолкователь) отличается от компилятора тем, что вместо перевода исходной программы в объектный код, сразу выполняет инструкции, записанные на входном языке. Наряду с тем достоинством, что интерпретатор непосредственно выполняет все команды, он имеет по сравнению с компилятором существенный недостаток: интерпретируемые программы выполняются во много раз медленнее программ, представленных в машинных кодах.

Транслятору необходимо произвести анализ исходной программы, лексический[2], синтаксический[3] и семантический[4], и затем сгенерировать объектную программу. Характерным свойством трансляции является то, что этот процесс не является линейным, т. е. преобразование исходной программы не последовательно. Трансляция обычно включает несколько фаз анализа исходной программы.

Лексический анализатор (*сканер*) — самая простая часть транслятора. Сканер читает символы из входного потока и строит из них лексемы[5]: идентификаторы, константы, служебные слова, знаки операций и т. п., которые затем передаются на обработку синтаксическому анализатору. Кроме этого, сканеру также можно поручить простейшие действия, не требующие анализа исходной программы, как-то: удаление комментариев, занесение идентификаторов в информационные таблицы и др. Фактически лексический анализатор представляет собой конечный автомат, см. стр. 187.

Синтаксический и *семантический анализаторы* выполняют наиболее сложную работу и являются основными компонентами при трансляции. Когда синтаксический анализатор получает от сканера лексему и «узнаёт» её как конструкцию языка, он вызывает соответствующую семантическую процедуру. Существуют программы для *автоматической* генерации синтаксических анализаторов, одну из которых — Bison — мы используем далее.

---

[2] *Лексика* (от греч. λέξισ — слово, речь, выражение) — словарный состав какого-либо языка, совокупность слов, входящих в состав языка. В данном случае лексика — правила, по которым слова (лексемы) составляются из отдельных символов.

[3] См. сноску на стр. 185.

[4] *Семантика* (греч. σημαντικόσ — обозначающий) — смысл, который имеет каждая конструкция языка.

[5] *Лексема* (от греч. λέξισ — слово, речь, выражение) — единица лексического уровня языка, представляющая собой слово во всей совокупности его значений. Например, все формы слова «логика» и различные значения этих форм в различных сочетаниях: «формальная логика», «математическая логика», «женская логика» и т. д. тождественны как выразители одной и той же лексемы. В данном случае лексемы — слова в соответствии с определениями языка.

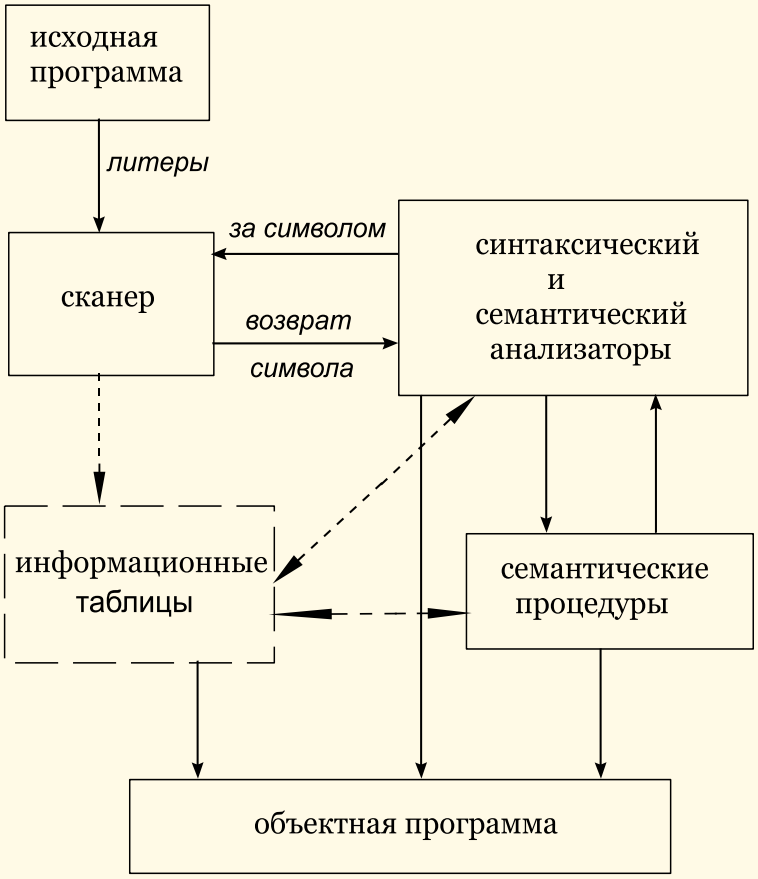


Рис. III.35. *Схема трансляции в простейшем случае [92], пунктирами изображены информационные потоки*

Генератор синтаксических анализаторов Bison преобразует описание контекстно-свободной LALR(1)[6] грамматики в программу на языке C для разбора этой грамматики [92]. Bison написан Робертом Корбеттом (Robert Corbett) при участии Ричарда Столлмена (Richard Stallman), Вильфреда Хансена (Wilfred Hansen) и др. Bison обратно совместим с другим известным генератором синтаксических анализаторов Yacc.

Bison обеспечивает общее средство для придания определенной структуры входным данным компьютерных программ. Пользователь готовит спецификацию процесса ввода; она включает в себя правила, описывающие структуру входных данных, код, вызываемый, когда распознаны эти правила, процедуру для ввода данных и лексический анализатор — функция yylex(). Затем Bison генерирует функцию yyparse() (синтаксический анализатор), контролирующую процесс ввода. Синтаксический анализатор вызывает yylex() для извлечения элементарных

[6] «Look Ahead Left Recursive» — леворекурсивный «с заглядыванием», предпросмотром. Метод грамматического разбора, считается менее мощным, чем «чистый» LR, но нуждается в таблицах меньшего размера. LR(k) разбор осуществляет восходящий грамматический анализ с правосторонним выводом, используя для определения следующего действия не более k лексических единиц. Алгоритм LALR(1) описан в статье [93].

объектов (лексем) из входного потока. Эти лексемы должны быть организованы в соответсвии с правилами структуризации входных данных, называемых *грамматическими правилами*. Когда одно из этих правил распознается, вызывается пользовательский код для этого правила, или *действие*; действия способны возвращать значения и использовать значения других действий.

Входные данные для Bison. Ядром спецификаций ввода является система грамматических правил, записываемых в виде близком к нормальной форме Бэкуса — Наура (см. стр. 184). Каждое правило описывает допустимую структуру и дает ей имя. Например, грамматическое правило может выглядеть так:

```
date : month_name day ',' year;
```

Здесь *date*, month_name, day и year представляют собой структуры для процесса ввода; предполагается, что month_name, day и year определены где-либо еще. Запятая «,» заключена в одиночные кавычки; это значит, что она должна появиться в тексте буквально. Двоеточие и точка с запятой служат просто как знаки препинания в правиле и не имеют никакого значения при контроле входных данных. Таким образом, при должных определениях ввод

```
October 25, 1917
```

будет соответствовать вышеуказанному правилу.

Входные данные могут не соответствовать спецификациям. Эти ошибки данных определяются настолько рано, насколько возможно при чтении слева-направо; таким образом, не только в значительной степени понижается вероятность чтения и работы с неправильными данными, но и неверные данные могут быть быстро обнаружены.

Каждый файл спецификаций состоит из трех секций: объявления, грамматические правила и программы. Секции разделяются символами двойного процента «%%». (Символ процента «%» в основном используется в Bison как Esc-символ.)

Другими словами, полный файл спецификации выглядит как

```
описания
%%
правила
%%
программы
```

Секция описаний может быть пуста. Более того, если секция программ опущена, то вторая метка %% также может быть опущена. Таким образом, минимальная разрешенная спецификация для Bison есть

```
%%
правила
```

Пробелы, табуляции и новые строки игнорируются, за исключением случаев, когда они появляются в именах или многосимвольных зарезервированных словах. Комментарии могут появляться везде, где разрешено имя; они заключаются в /* . . . */, как в языке C.

Секция правил построена из одного или более грамматических правил, имеющих вид:

```
A : BODY ;
```

A представляет собой нетерминальное имя, а BODY представляет собой последовательность нуля или более имен и литералов. Двоеточие и точка с запятой — пунктуация Bison.

Имена допускаются произвольной длины и могут состоять из букв, точек «.», подчерков «_» и неначальных цифр. Заглавные и строчные буквы различаются. Имена, используемые в теле грамматических правил могут представлять собой токены и нетерминальные символы. Литерал состоит из символов, заключенных в одиночные кавычки «'».

Каждый нетерминальный символ должен появляться в левой части как минимум одного правила. Среди всех нетерминальных символов один, называемый *стартовым* символом, имеет особенную важность. Парсер разработан так, чтобы распознавать стартовый символ; таким образом, этот символ представляет самую большую, самую общую структуру, описанную грамматическими правилами. По умолчанию стартовый символ берется из левой части первого грамматического правила в секции правил. Возможно и даже желательно, явно объявить стартовый символ в секции объявлений, используя ключевое слово %start:

```
%start keyword
```

Действия. Каждому грамматическому правилу пользователь может приписать *действия*, исполняемые каждый раз, когда распознается правило во входных данных. Эти действия могут возвращать значения, и получать значения, возвращенные предыдущими действиями. Более того, лексический анализатор может возвращать значения токенов, если это необходимо.

Действие — это произвольный C-оператор, и поэтому может производить ввод и вывод, вызывать подпрограммы, изменять внешние структуры и переменные. Действие обозначается одним или более операторами, заключенными в фигурные скобки «{» и «}». Например,

```
A : '(' B ')' { hello ( 1, "abc" ); }
```

и

```
XXX : YYY ZZZ { printf ( "a message\n" ); flag = 25; }
```

являются грамматическими правилами с действиями.

Чтобы вернуть значение, действие обычно устанавливает псевдопеременную «$$» в некоторое значение. Например, действие, которое ничего не делает, а только возвращает значение 1, это

```
{ $$ = 1 }
```

Чтобы получить значения, возвращенные предыдущими действиями и лексическим анализатором, действие также может использовать псевдопеременные $1, $2, ..., которые относятся к значениям, возвращенным компонентами в правой части правила, при его чтении слева направо. Таким образом, если правило, скажем, такое

```
A     : B C D ;
```

то $2 имеет значение, возвращенное C, а $3 — значение, возвращенное D.

Как более конкретный пример, представим правило

```
expr  :  '('  expr  ')'  ;
```

Значение, возвращенное этим правилом — это обычное значение expr в скобках. Это может быть выражено так:

```
expr  :  '('  expr  ')'   { $$ = $2 ; }
```

По умолчанию значение правила — это значение первого элемента в нем ($1). Таким образом, грамматическое правило вида

```
A :  B ;
```

часто не требует явного действия.

Лексический и синтаксический анализ. Пользователь должен предоставить лексический анализатор для чтения входных данных и передавать токены (со значениями, если надо) парсеру. Лексический анализатор — это функция, называемая yylex(). Функция возвращает целое число, номер токена, представляющий вид прочитанного токена. Если с этим токеном связано *значение*, оно должно быть присвоено внешней переменной yylval.По историческим причинам, маркер конца должен иметь номером токена 0 или отрицательное число. Этот номер токена не может быть переопределен пользователем; таким образом, *все* лексические анализаторы должны быть написаны так, чтобы возвращать 0 или отрицательное число при достижении конца своих входных данных.

Парсер, созданный Bison, является конечным автоматом со стеком (см. стр. 189). Парсер также способен читать и запоминать следующий входной токен (называемый *lookahead token*). Текущее состояние — всегда на вершине стека. Состояниям конечного автомата присваиваются небольшие целые метки; изначально автомат находится в состоянии 0, и не прочитано ни одного *lookahead token*'а.

Автомату доступно только четыре действия, называемые *сдвиг (shift)*, свертка *(reduce)*, *принятие (accept)* и *ошибка (error)*. Каждый шаг разбора происходит следующим образом:

1. Основываясь на текущем состоянии парсер определяет, нужен ли ему *lookahead token* для решения, какое действие нужно произвести; если ему требуется *lookahead token* и он его не имеет, то вызывает yylex() для получения следующего токена.
2. Используя текущее состояние и, если необходимо, *lookahead token* парсер принимает решение о следующем действии и производит его. В результате состояния могут быть записаны в стек или прочитаны из него или *lookahead token* обработан или оставлен неизменным.

Действие сдвига — самое частое действие, которое производит парсер. Когда исполняется действие сдвига, всегда есть *lookahead token*. Например, в состоянии 56 может быть действие:

```
IF shift 34
```

которое означает, в состоянии 56, если *lookahead token* есть `IF`, текущее состояние (56) продвигается дальше в стеке, и состояние 34 становится текущим (на вершине стека). *lookahead token* очищается.

Действие свертки предохраняет стек от неограниченного возрастания. Свертки соответствуют тому, что парсер «увидел» правую часть правила и готовится объявить, что он увидел случай, подчиняющийся правилу, заменяя его правую часть на левую. Действие свертки также важно при обработке пользовательских действий и значений. Когда происходит свертка, код, связанный с правилом исполняется перед тем как стек выравнивается. В дополнение к стеку, содержащему правила, параллельно ему работает еще один, содержащий значения, возвращенные лексическим анализатором и действиями. Когда происходит сдвиг, внешняя переменная yylval копируется на стек значений. После возврата из пользовательского кода происходит свертка. Когда происходит действие перехода, внешняя переменная yylval копируется на стек значений. Псевдопеременные $1, $2 и т. д. ссылаются на стек значений.

Остальные два действия парсера значительно проще. Действие принятия сигнализирует о том, что просмотрены все входные данные и они соответсвуют спецификации, это происходит только когда *lookahead token* есть маркер конца и сигнализирует, что парсер успешно завершил работу. Действие ошибки обозначает место, с которого парсер не может продолжать разбор в соответствии со спецификацией. За входным токеном, вместе с *lookahead token* не может следовать ничего, что является верными данными. Парсер сообщает об ошибке и пытается продолжить разбор, если это возможно.

Транслятор L2eps напишем на языке C для операционной системы Windows, см. листинги на стр.136. Его портирование на другие платформы не представит никаких трудностей.

Порядок компиляции и сборки следующий:

— исходные файлы L2eps.y, Lsyst.h, Lsyst.c, EPS.h, EPS.c поместить в отдельную директорию;

— запустить Bison

bison.exe −lvg L2eps.y,

после чего Bison создаст головной файл L2eps.tab.c, содержащий синтаксический анализатор и, кроме того, файл L2eps.output — разбор грамматики и L2eps.vcg — «дерево» разбора в специальном формате для визуализации графов VCG;

— скомпилировать и собрать проект.

Транслятор работает из коммандной строки, формат вызова:

L2eps.exe L_файл.l EPS_файл.eps Order нач_угол

Последний параметр не обязателен, начальный угол по умолчанию равен 0. В соответствии с описанием L-системы, содержащемся в L_файл.l, будет сгенерирован файл EPS_файл.eps.

Изображения III.29, III.30, III.31, III.32, III.33, III.34, приведенные выше, построены программой L2eps по соответствующим L-кодам.

5.3. Простейшие клеточные автоматы

Стивен Вольфрам, известный разработкой системы компьютерной алгебры Mathematica и системы извлечения знаний WolframAlpha, предложил новый метод моделирования сложных систем [96], основываясь на предположении, что любая система состоит из многих более или менее идентичных простых элементов с ограниченным числом возможных состояний. Состояние элементов зависит от состояний расположенных рядом элементов и правил, определяющих эту зависимость. Таким образом, данные простые элементы можно трактовать как клеточные автоматы, см. стр. 187.

Наиболее простым и, в то же время, нетривиальным клеточным автоматом является *одномерный* (решетка автомата представляет собой одну строку клеток) клеточный автомат с *двумя возможными состояниями* (0, 1 или, скажем, черный и белый цвета), при котором соседями клетки считаются две смежные с ней в строке клетки. Таким образом, три клетки (текущая клетка с соседями) могут принимать $2^3 = 8$ различных состояний. Правило перехода, т. е. будет ли на следующем шаге центральная клетка тройки «белой» или «чёрной», задаётся исходя из состояния тройки, например, как показано на рис. III.36.

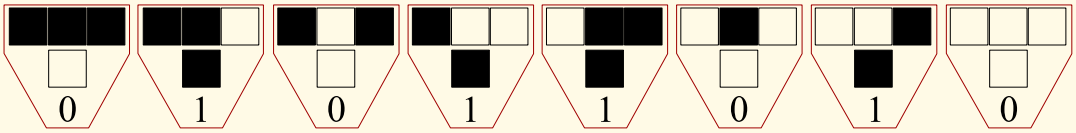


Рис. III.36. *Правило перехода* 90 ($90 = 01011010_2$)

Нумерация правил определяется переводом бинарной последовательности, определяющей правило, в десятичное представление, — например на рис. III.36 задано правило 90. Последовательное применение данного правила приводит к генерации фрактала, известного как треугольник Серпинского, или «салфетка» Серпинского.

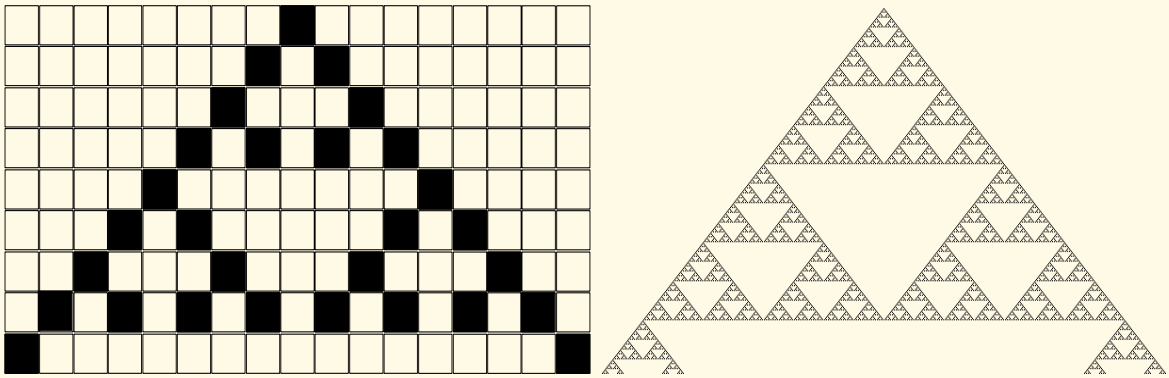


Рис. III.37. *Элементарный клеточный автомат с правилом* 90 *генерирует «салфетку» Серпинского*

Правило под номером 30 порождает последовательность, кажущуюся случайной, при неслучайных начальных условиях — Стивен Вольфрам использует его для генерации псевдослучайных целых чисел в пакете Mathematica. Эволюция системы по правилу 30 демонстрирует апериодическое, хаотическое поведение. Вольфрам считает, что, наряду с другими, правило 30 — ключ к пониманию того, как простые правила могут порождать сложные структуры и различное сложное поведение разных природных объектов.

Правило 184 используется в качестве простой модели транспортного потока на однополосном шоссе: каждая клетка автомата соответствует транспортному средству, которое движется вперед, если она имеет открытое пространство перед ней — на каждом шаге клетка в состоянии 1, справа от которой находится клетка в состоянии 0 («свободное место»), перемещается вправо, освобождая занятое пространство (см. рис. III.38). Поэтому это правило называют иногда «дорожным правилом». В физике аэрозолей правило 184 применяется для моделирования осаждения частиц на нерегулярную поверхность.

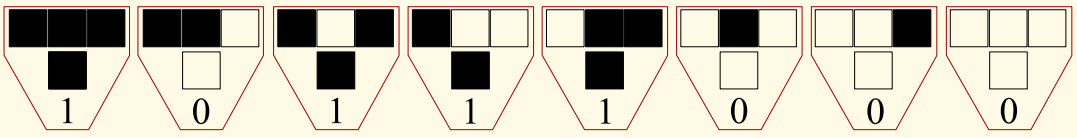


Рис. III.38. *Правило перехода* 184

Для программной реализации простейшего клеточного автомата дополнительно примем, что последняя ячейка каждой строки совпадает с первой. См. листинги на стр. 148.

IV. РЕШЕНИЕ ВОПРОСОВ

1. Физика

1.1. Виброгаситель

Вопрос III.1. Из уравнения (3) получаем

$$x_1 = -\frac{m_2}{m_1}\,x_2 - \frac{A}{\omega^2 m_1}\sin\omega t.$$

Подставляя это значение в (1), получим дифференциальное уравнение

$$m_2\ddot{x}_2 + k\left(1+\frac{m_2}{m_1}\right)x_2 + \frac{kA}{\omega^2 m_1}\sin\omega t = 0,$$

частное[1] решение которого не трудно найти подбором:

$$x_2 = \frac{kA}{\omega^2\bigl(m_1 m_2\omega^2 - k(m_1+m_2)\bigr)}\sin\omega t.$$

На графиках IV.1 показаны поведение решений системы уравнений (1)–(2) при различных значениях параметра ω.

1.2. Реалистичное освещение. Рейтрессинг

Вопрос III.2. Язык программирования — Object Pascal (Delphi). Будем использовать объектно-ориентированнвй метод проектирования [2].

Для того, чтобы иметь возможность работать с лучами света и задавать координаты точек в пространстве нужны *векторы* — соответствующие процедуры и функции сосредоточим в классе Tvector (см. стр. 117). Для работы с *цветами* нам понадобятся соответствующие процедуры и функции смешивания цветов — модуль colorUnit (см. стр. 120). Объект Луч, который нужен для работы с источником света, по существу является вектором, имеющим цвет, поэтому выведем соответствующий класс Tray из класса Tvector.

```
Type Tray = class( Tvector )
 protected
 color                    : Tcvet;
 public
 function      get_source  : Tvector;
 function      get_color   : Tcvet;
 constructor   birth ( x0, y0, z0: double;
                       RR, GG, BB: integer ); overload;
 constructor   birth ( p0 :        Tpoint3d;
                       CC :        Tcvet     ); overload;
end;
```

[1] По условиям вопроса нет необходимости находить *общее* решение системы (1)–(2), отыскание которого, впрочем, не представляет никаких сложностей.

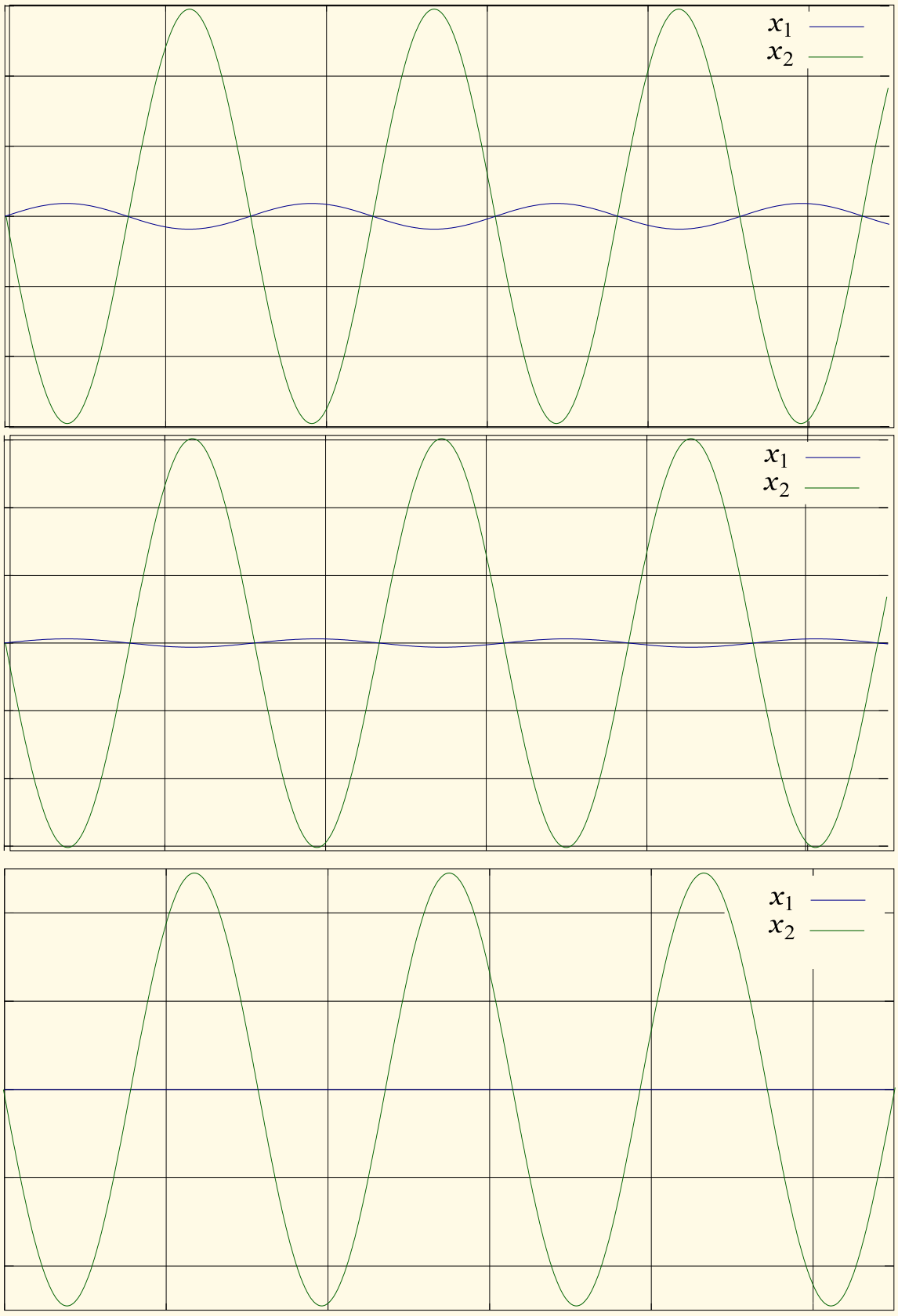


Рис. IV.1. *Виброгаситель, отклонения от величины ω, определяемой соотношением* (4)*, сверху-вниз: 3%, 1%, точное значение*

Несмотря на то, что в нашей модели только один тип тел — сфера, учитывая возможные дальнейшие модификации программы, когда к сферам на сцену могут добавиться и другие виды объектов, сосредоточим общие для них методы в базовом классе Tfigure (см. стр. 121). Этот класс содержит методы для расчёта отражённого луча, вычисления коэффициентов диффузии и отражения в заданной точке, для которых нужно знать *нормаль* к данной точке.

Поскольку нормаль можно определить только для конкретного тела, то метод set_normal объявлен виртуальным и абстрактным, с тем, чтобы в дальнейшем иметь возможность его переопределения в производных классах.

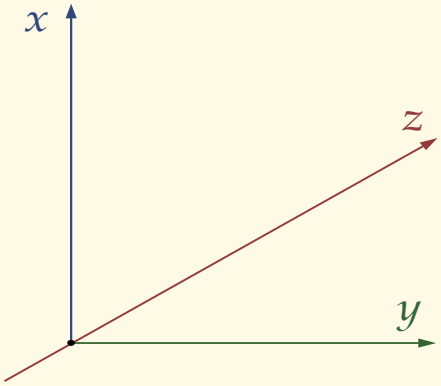
```
type  Tfigure = class
protected
center      : Tvector;              // центр фигуры;
normal      : Tvector;              // нормаль;
reflected   : Tvector;              // отражённый луч;
color       : Tcvet;                // цвет;
n_spec      : double;               // показатель отражения;
v_spec      : double;               // коэффициент отражения;
procedure  set_normal ( point : Tvector ); virtual; abstract;
procedure  set_reflected ( incident, p: Tvector );
public
function   get_reflected ( incident, p: Tvector ): Tvector;
function   get_diff ( p, light_direction : Tvector ): double;
function   get_spec ( p, ray_direction : Tvector;
                      light_direction : Tvector ): double;
function   get_value_spec : double;
function   get_color : Tcvet;
constructor  birth ( center0 : Tvector; color0 : Tcvet;
                     n_spec0, v_spec0 : double          );
end;
```

Из класса Tfigure выведем класс Tsphere (сфера, стр. 123). Объект этого класса должен иметь возможность рассчитывать вектор нормали к любой точке своей поверхности, определять факт пересечения трассирующего луча с поверхностью и, кроме того, определять расстояние до камеры. Для этого в классе Tsphere заданы соттветствующие методы set_normal и intersection , последний из которых возвращает множитель t, на который нужно умножить направляющий вектор трассирующего луча, чтобы этот луч попал на сферу. Если луч не попадает на сферу, то метод возвращает $-1{,}0$.

```
type  Tsphere = class ( Tfigure )
protected
radius  : double;
```

```
procedure set_normal ( point : Tvector ); override;
public
function intersection ( ray_origin : Tvector;
                        ray_direction : Tvector ): double;
constructor birth ( center0 : Tvector; color0 : Tcvet;
                    n_spec0, v_spec0, radius0 : double );
end;
```

В модуле traceUnit (стр. 125) содержится процедура инициализации сцены init_scene (используется левая координатная система — см. рисунок, считаем, что картинная плоскость фиксирована и совпадает с плоскостью XY), задающая положение камеры, источника света, количество и параметры сфер и т. д., и *трассировщик* — рекурсивная функция trace , которая собственно и выполняет всю основную работу по трассировке лучей. Головной модуль mainUnit содержит процедуру отрисовки изображения, средства для поддержки управления программой и для сохранения трассированного изображения в файл. Пример трассировки показан на рис. IV.2.



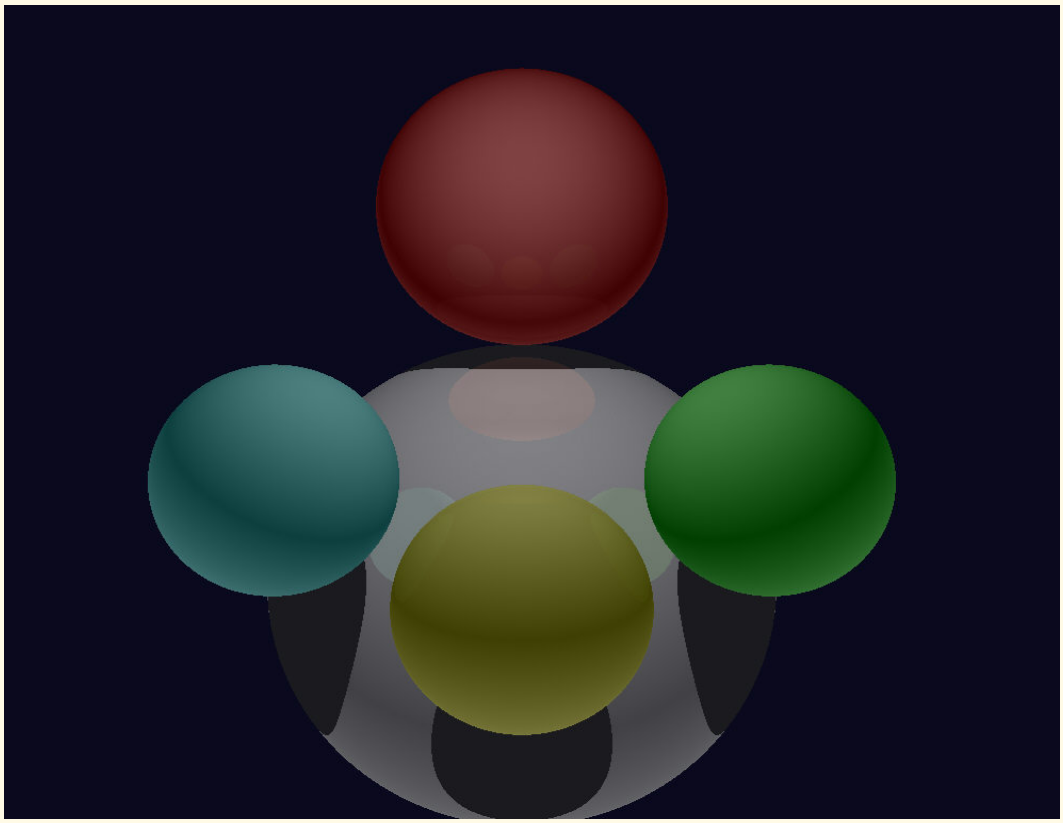


Рис. IV.2. *Рейтрессинг сфер*

Использованный нами объектно-ориентированный метод построения программы даёт при желании возможность легко и относительно просто

дополнить сцену новыми объектами — любыми трёхмерными телами[2]. Подробное описание программы трассировки см. в [2], её листинги — на стр. 117.

Вопрос III.3. Решение начнем с вывода закона преломления Снелла. Пусть луч света из среды, в которой он имеет скорость v_1 попадает в среду, где его скорость равна v_2. Найдём, применяя принцип Ферма, как связаны величины v_1, v_2 и α_1, α_2 (см. рис. IV.3). Время, необходимое для прохождения луча света от т. A до т. B, равно

$$T(x) = \frac{\sqrt{a^2 + x^2}}{v_1} + \frac{\sqrt{b^2 + (c - x)^2}}{v_2}.$$

В соответствии с принципом Ферма нужно найти минимум функции

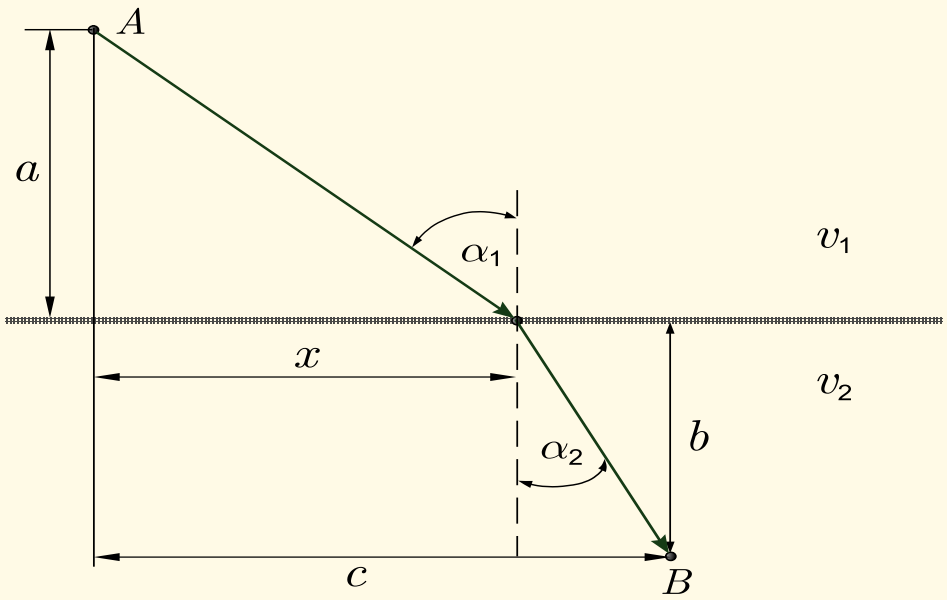


Рис. IV.3. *К выводу закона преломления Снелла*

$T(x)$, для чего, в частности, требуется решить уравнение

$$\frac{dT}{dx} = 0,$$

откуда получим

$$\frac{x}{v_1\sqrt{a^2 + x^2}} = \frac{c - x}{v_2\sqrt{b^2 + (c - x)^2}},$$

или, что то же самое,

$$\frac{\sin\alpha_1}{v_1} = \frac{\sin\alpha_2}{v_2}$$

Равенство углов падения и отражения лучей света сразу следует из последнего соотношения, если положить $v_1 = v_2$.

[2]Подробнее об основах объектно-ориентированного проектирования см. [56], [57], [58].

2. Биология и физиология

2.1. Модель эпидемии SIR

Вопрос III.4. Эпидемия не начнется, если скорость инфицирования не будет положительной — $\dot{I} \leqslant 0$, то есть в соответствии с (17),

$$\alpha S_0 I_0 - \beta I_0 \leqslant 0.$$

Ответ: эпидемия не начнется, если изначально выполняется соотношение

$$S_0 \leqslant \frac{\beta}{\alpha}.$$

Пример, когда это условие выполняется, показан на рис. III.10.

Вопрос III.5. Поделив почленно уравнение (17) на (16) получим

$$\frac{\dot{I}}{\dot{S}} = \frac{dI}{dS} = -1 + \frac{\beta}{\alpha}\frac{1}{S},$$

откуда, после интегрирования, имеем

$$I = -S + \frac{\beta}{\alpha}\ln S + C,$$

константа C определяется из начальных условий (19)

$$C = I_0 + S_0 - \frac{\beta}{\alpha}\ln S_0.$$

И, следовательно, так как параметры α, β в последние соотношения входят в одних и тех же степенях, то они одинаково влияют на поведение модели. А поскольку, при прочих равных условиях, профилактика всегда лучше лечения, исходя из SIR-модели заключаем, что эпидемию действительно предпочтительнее предупредить.

Вопрос III.6. Аппроксимация табличных данных III.6 методом наименьших квадратов с абсолютной погрешностью $\varepsilon \leqslant 0{,}02$ дает следующие результаты

$$\begin{aligned}
S_{app} &= -0{,}4667t^3 + 5{,}4236t^2 - 21{,}1218t + 108{,}8942, && \varepsilon = 0{,}00173;\\
I_{app} &= 0{,}4644t^3 - 5{,}3977t^2 + 21{,}0219t - 8{,}8522, && \varepsilon = 0{,}00779;\\
R_{app} &= 0{,}0023t^3 - 0{,}0268t^2 + 0{,}1028t - 0{,}0452, && \varepsilon = 0{,}01633.
\end{aligned}$$

Коэффициенты α, β определим как среднее арифметическое величин

$$\beta = \frac{\dot{R}_{app}}{I_{app}} \approx 0{,}0020, \quad \alpha = -\frac{\dot{S}_{app}}{S_{app}I_{app}} \approx 0{,}0047$$

Поведение модели и сравнение с табличными данными показаны на графиках IV.4.

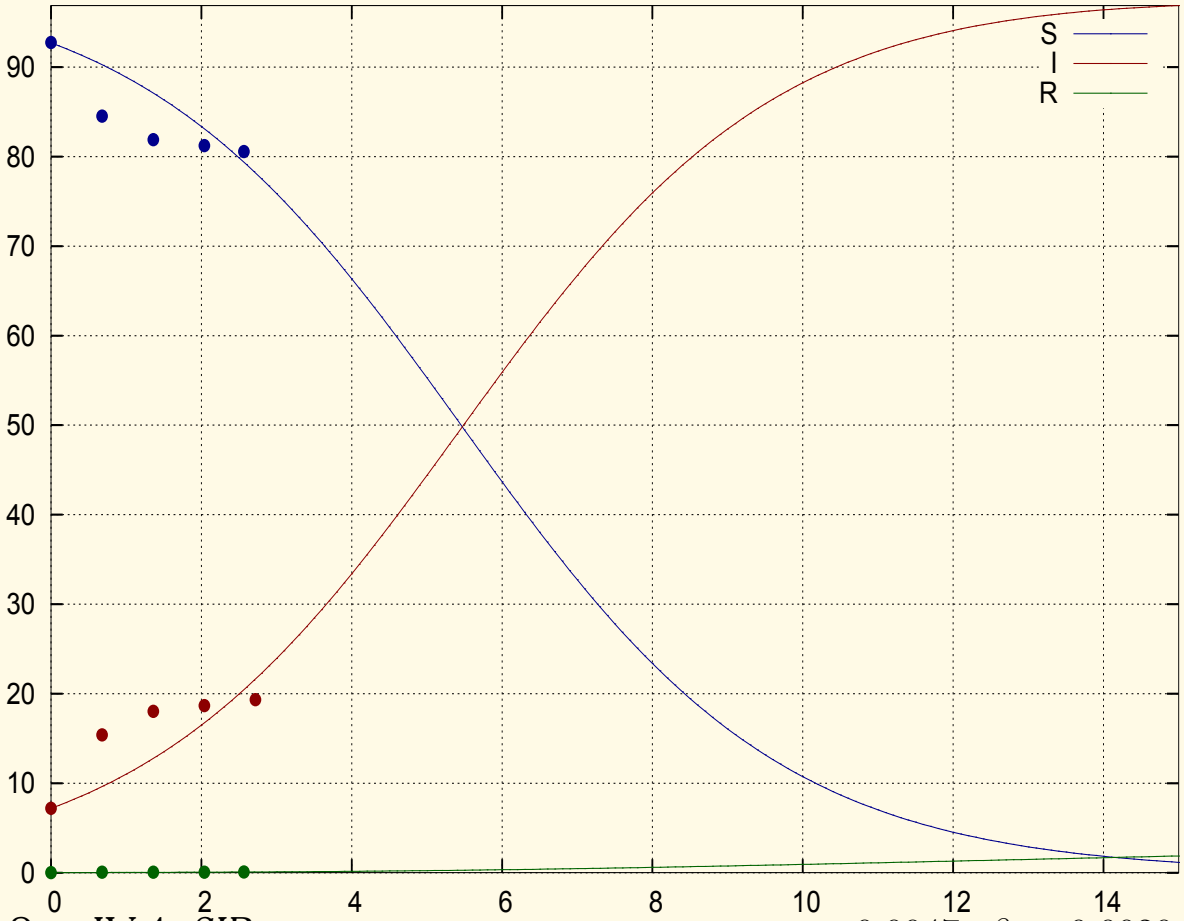


Рис. IV.4. *SIR-модель, значения параметров:* $\alpha = 0{,}0047$, $\beta = 0{,}0020$, $I_0 = 7{,}2$, $S_0 = 92{,}7$, $R_0 = 0{,}03$. *Точками обозначены табличные данные.*

2.2. Модель зомби-эпидемии

Вопрос III.7. Якобиан матрицы системы уравнений (21)–(23)

$$\mathbf{J} = \begin{bmatrix} -\alpha Z & -\alpha S & 0 \\ (\alpha-\beta)Z & (\alpha-\beta)S & \gamma \\ \beta Z & \beta S & -\gamma \end{bmatrix}$$

Для случая $(0, \overline{Z}, 0)$ (зомби-апокалипсис) имеем

$$\mathbf{J} = \begin{bmatrix} -\alpha\overline{Z} & 0 & 0 \\ (\alpha-\beta)\overline{Z} & 0 & \gamma \\ \beta\overline{Z} & 0 & -\gamma \end{bmatrix}.$$

Откуда, поскольку характеристическое уравнение

$$\det(\mathbf{J} - \lambda\mathbf{I}) = -\left(\lambda^3 + \left(\alpha\overline{Z} + \gamma\right)\lambda^2 + \alpha\gamma\overline{Z}\lambda\right),$$

где $\mathbf{I}$ — единичная матрица, по критерию Гурвица (стр. 159) имеет один нулевой корень и остальные корни с отрицательными действительными частями, заключаем, что система находится на границе устойчивости.

Для точки $(\overline{S},0,0)$

$$\mathbf{J}=\begin{bmatrix}0 & -\alpha\overline{S} & 0\\ 0 & (\alpha-\beta)\overline{S} & \gamma\\ 0 & \beta\overline{S} & -\gamma\end{bmatrix}$$

и поскольку характеристическое уравнение

$$\det\left(\mathbf{J}-\lambda\mathbf{I}\right)=-\left(\lambda^3-\left((\alpha-\beta)\overline{S}-\gamma\right)\lambda^2-\beta\gamma\overline{S}\lambda\right)$$

имеет отрицательный коэффициент, это решение неустойчиво.

2.3. Цикады и простые числа

Вопрос III.8. Язык программирования — C++, см. стр. 132.

2.4. Модель отрезвления

Вопрос III.9. Ответ: приблизительно 250 миллилитров.

3. Военное дело

3.1. Модель Ланчестера — Осипова

Вопрос III.10. Прилагая модель к условиям боя танков ИС-2 («красные») с Тиграми («синие»), получаем, что коэффициенты равны $b=9\cdot0{,}2=1{,}8$, $r=3\cdot0{,}9=2{,}7$.

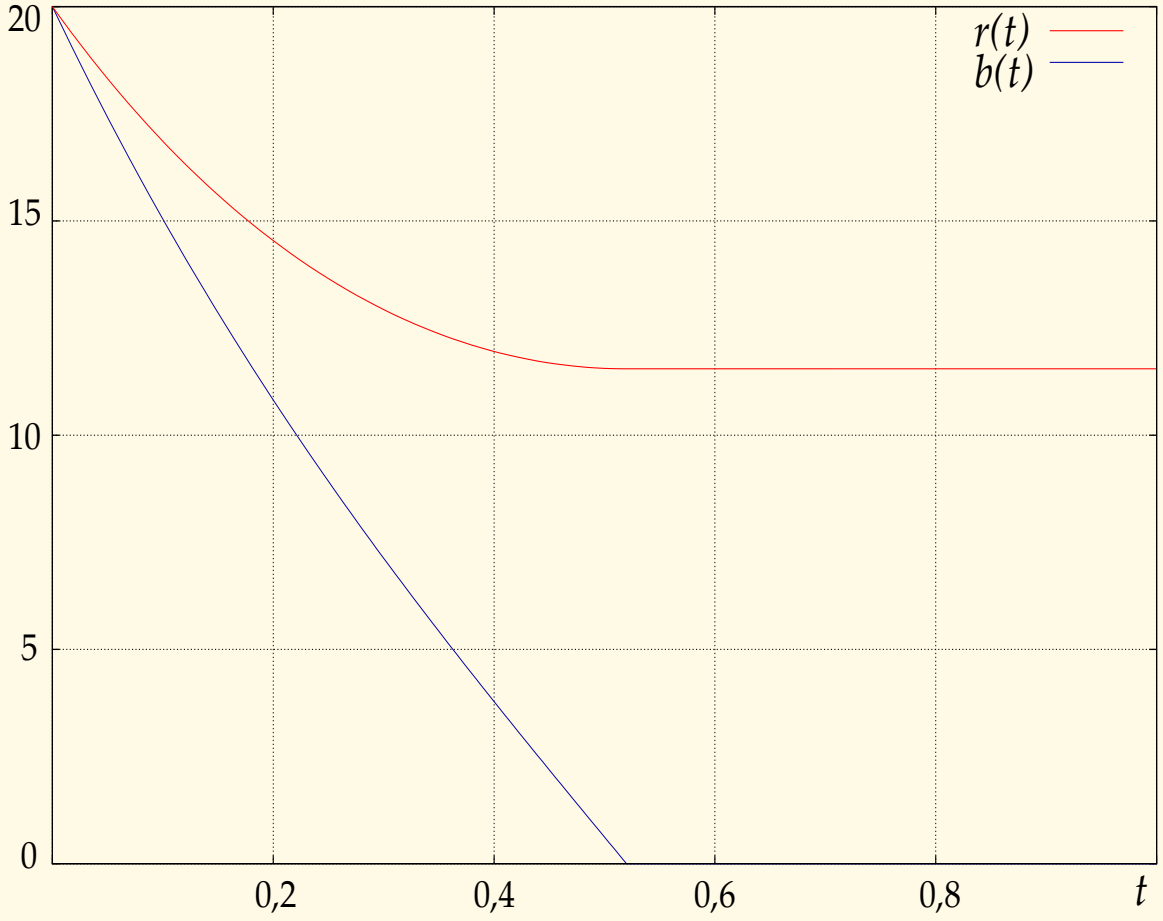


Рис. IV.5. *Модель боя ИС-2 с Тиграми*

Расчеты в Maxima (см. 5.3, стр. 135) с указанными параметрами приводят к следующим результатам:

```
"Синие уничтожены через 0.51993406116865 мин."
"У Красных осталось 12 танков."
```

Ход сражения иллюстрируется графиком IV.5.

3.2. Военная игра

Вопрос III.11. Стратегия (k, m) означает: «направить m кораблей для охраны транспорта». Платёжная матрица игры

	$(3,0)$	$(0,3)$	$(2,1)$	$(1,2)$
$(4,0)$	4	0	2	1
$(0,4)$	0	4	1	2
$(3,1)$	1	-1	3	0
$(1,3)$	-1	1	0	3
$(2,2)$	-2	-2	2	2

Решение:

$$\mathbf{X}^* = \left(\frac{4}{9}, \frac{4}{9}, 0, 0, \frac{1}{9}\right), \qquad \mathbf{Y}^* = \left(\frac{1}{18}, \frac{1}{18}, \frac{4}{9}, \frac{4}{9}\right), \qquad v = \frac{14}{9}.$$

Вопрос III.12. После удаления доминируемых строк/столбцов матрица игры примет следующий вид

	A	B	C
1)	13	29	8
3)	18	22	31
6)	23	22	19

Решение игры:

$$\mathbf{X}^* = \left(0, \frac{4}{17}, \frac{13}{17}\right), \qquad \mathbf{Y}^* = \left(\frac{12}{17}, 0, \frac{5}{17}\right), \qquad v = \frac{371}{17} \approx 21,82.$$

Ущерб Союзников равен $29 - 21{,}82 = 7{,}12$. Другими словами, исторически выбранная стратегия отлична от оптимальной примерно на 32%.

4. Социология и политология

4.1. Модель коррупции

Вопрос III.13. Рост популярности правительства

$$\dot{x} = \frac{\alpha x}{\beta + x} - \gamma y_0 z_0 x > 0 \tag{1}$$

при $x \geqslant 0$, ввиду неравенства

$$\frac{\alpha}{\beta + x} < \frac{\alpha}{\beta + x_0},$$

обеспечивается при выполнении соотношения

$$\frac{\alpha}{\beta + x_0} - \gamma y_0 z_0 > 0.$$

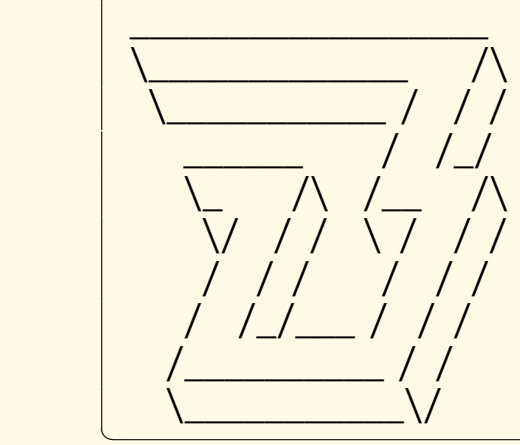
5. Листинги

Болтовня ничего не стоит. Покажите мне код.

Линус Торвальдс

Для сокращения размера программного кода почти не применялась оптимизация. Эффективность всюду принесена в жертву читаемости. Все необходимые пояснения содержатся в комментариях.

Коды программ поставляются в том виде как есть, автор не несет ответственности в случае, если использование или неправильное использование программы повлекло нарушение функционирования каких-либо компьютерных систем.



Copyleft : В.Зенкин, (v_zenk@mail.ru). Разрешается любое использование программных кодов, приведенных в данной книге, без ограничений.

5.1. Реалистичное освещение. Рейтрессинг

Базовый класс Tvector (вектор) и векторные операции

```
// файл vectorUnit.pas
unit vectorUnit;
interface
//-------------------------------------
type Tpoint3D = record
                x, y, z : double;
                end;
//-------------------------------------
type Tvector = class
protected
coordinate : Tpoint3D;
public
function      get_coord   : Tpoint3D;
function      get_module  : double;
procedure     normalization;
constructor   birth ( xx, yy, zz: double ); overload;
constructor   birth ( p: Tpoint3d );        overload;
end;
//-- векторные операции: ---------------------
```

```
 function   summ_vectors( v_1, v_2: Tvector ): Tvector;
 function   dot_product ( v_1, v_2: Tvector ) : double;
 function   mult_on_scal ( k: double; v: Tvector ): Tvector;
//=====================================
 implementation
//-------------------------------------
 function   Tvector.get_coord : Tpoint3D;
begin
 result := coordinate;
end;
//-------------------------------------
 function   Tvector.get_module : double;
begin
 result := sqrt (     coordinate.x * coordinate.x
                    + coordinate.y * coordinate.y
                    + coordinate.z * coordinate.z  );
end;
//-------------------------------------
 procedure    Tvector.normalization;
var  module: double;
begin
module := get_module;
    if  ( module > 0.0 ) then
    begin
     coordinate.x  :=  coordinate.x /module;
     coordinate.y  :=  coordinate.y /module;
     coordinate.z  :=  coordinate.z /module;
    end;
end;
//-------------------------------------
 constructor  Tvector.birth ( xx, yy, zz: double );
begin
 coordinate.x  := xx;
 coordinate.y  := yy;
 coordinate.z  := zz;
end;
//-------------------------------------
 constructor  Tvector.birth ( p: Tpoint3d );
begin
 coordinate.x  := p.x;
 coordinate.y  := p.y;
 coordinate.z  := p.z;
end;
//=====================================
```

```
 function  summ_vectors( v_1, v_2: Tvector ): Tvector;
 var  c1, c2: Tpoint3D;
 begin
 c1      := v_1.get_coord;
 c2      := v_2.get_coord;
  result  := Tvector.birth (c1.x + c2.x, c1.y + c2.y, c1.z + c2.z);
 end;
 //-------------------------------------
  function   dot_product ( v_1, v_2: Tvector ): double;
 var  c1, c2: Tpoint3D;
 begin
 c1      := v_1.get_coord;
 c2      := v_2.get_coord;
  result  := c1.x*c2.x + c1.y*c2.y + c1.z* c2.z;
 end;
 //-------------------------------------
  function   mult_on_scal ( k: double; v: Tvector ): Tvector;
 var  c: Tpoint3D;
 begin
 c       := v.get_coord;
  result  := Tvector.birth ( c.x*k, c.y*k, c.z*k );
 end;
 //-------------------------------------
 end. // конец файла vectorUnit.pas
```

Класс Tray (луч)

```
// файл rayUnit.pas
 unit  rayUnit;
  interface
 uses  vectorUnit,  colorUnit;
 //-------------------------------------
 Type Tray = class( Tvector )
  protected
 color                  : Tcvet;
 public
  function   get_source  : Tvector;
  function   get_color   : Tcvet;
  constructor   birth ( x0, y0, z0: double;
                        RR, GG, BB: integer  ); overload;
  constructor   birth ( p0 :          Tpoint3d;
                        CC :          Tcvet     ); overload;
 end;
 //-------------------------------------
```

```
implementation
//-------------------------------------
function    Tray.get_source :  Tvector;
begin
 result  :=  Tvector.birth (   coordinate.x,
                               coordinate.y,
                               coordinate.z    );
end;
//-------------------------------------
function    Tray.get_color :  Tcvet;
begin
 result  :=  color;
end;
//-------------------------------------
constructor    Tray.birth ( x0, y0, z0: double;
                            RR, GG, BB: integer );
begin
 inherited  birth ( x0, y0, z0 );
color.R  := RR; color.G := GG; color.B := BB;
end;
//-------------------------------------
constructor    Tray.birth ( ρ0 : Tpoint3d; CC: Tcvet );
begin
 inherited  birth ( ρ0 );
color  := CC;
end;
//-------------------------------------
end. // конец файла rayUnit.pas
```

Операции с цветом

```
// файл colorUnit.pas
unit  colorUnit;
 interface
uses  math;
//-------------------------------------
type  Tcvet =  record                 // цвет как тройка
               R, G, B :  integer;    // чисел Red, Green,
               end;                   // Blue;
//== смешивание и «умножение цветов»: ===========
function    mix_colors ( c_1, c_2: Tcvet ): Tcvet;
function    mult_colors ( color : Tcvet; k: double ): Есмуе;
//-------------------------------------
implementation
```

```
//-----------------------------------------
 function   mix_colors( c_1, c_2: Tcvet ): Tcvet;
 var  c: Tcvet;
 begin
c.R     := ( c_1.R + c_2.R ) div 2 mod 256;
c.G     := ( c_1.G + c_2.G ) div 2 mod 256;
c.B     := ( c_1.B + c_2.B ) div 2 mod 256;
 result  := c;
end;
//-----------------------------------------
 function   mult_colors ( color : Tcvet; k: double ): Tcvet;
 begin
 color.R  :=   floor ( abs( k*color.R ) ) mod 256;
 color.G  :=   floor ( abs( k*color.G ) ) mod 256;
 color.B  :=   floor ( abs( k*color.B ) ) mod 256;
 result   :=   color;
 end;
 end. // конец файла colorUnit.pas
```

Базовый класс Tfigure

```
// файл figureUnit.pas
 unit  figureUnit;
  interface
 uses  vectorUnit,  colorUnit;
//-----------------------------------------
 type  Tfigure = class
  protected
 center      : Tvector;    // центр фигуры;
normal       : Tvector;    // нормаль;
 reflected   : Tvector;    // отражённый луч;
 color       : Tcvet;      // цвет;
n_spec       : double;     // показатель отражения;
v_spec       : double;     // коэффициент отражения;
 procedure  set_normal( point : Tvector ); virtual;  abstract;
 procedure   set_reflected ( incident, p: Tvector );
 public
 function    get_reflected ( incident, p: Tvector ): Tvector;
 function    get_diff ( p,  light_direction : Tvector ): double;
 function    get_spec ( p,  ray_direction : Tvector;
                        light_direction : Tvector ): double;
 function    get_value_spec : double;
 function    get_color : Tcvet;
 constructor    birth ( center0 : Tvector; color0 : Tcvet;
```

```
                        n_spec0, v_spec0: double          );
end;
//-----------------------------------------------
 implementation
//-----------------------------------------------
// определяет отражённый в т. ρ луч, incident – падающий луч
//-----------------------------------------------
 procedure  Tfigure.set_reflected ( incident, ρ: Tvector);
 var  scalar : double;
      tmρ    : Tvector;
 begin
 set_normal ( ρ );
 scalar := dot_product ( normal, incident );
tmρ := mult_on_scal ( -2.0* scalar, normal );
 reflected := summ_vectors( incident, tmρ );
 tmρ.Free;
end;
//-----------------------------------------------
// возвращает отражённый в т. ρ луч, incident – падающий луч
//-----------------------------------------------
 function  Tfigure.get_reflected ( incident, ρ: Tvector ): Tvector;
 begin
  set_reflected ( incident, ρ );
 result := reflected;
end;
//-----------------------------------------------
// возвращает коэффициент диффузии в т. ρ при направлении
// на осветитель  light_direction
//-----------------------------------------------
 function  Tfigure.get_diff (ρ, light_direction : Tvector ): double;
 begin
 set_normal ( ρ );
 result := dot_product ( light_direction, normal );
end;
//-----------------------------------------------
// возвращает коэффициент отражения в т. ρ при направлениях
// на осветитель  light_direction, из камеры -- ray_direction
//-----------------------------------------------
 function  Tfigure.get_spec ( ρ              : Tvector;
                              ray_direction  : Tvector;
                              light_direction : Tvector ): double;
 var  scalar : double;
 begin
 scalar := dot_product ( reflected, light_direction );
```

```
    if  ( scalar < 0.0 )
    then  result := 0.0
    else  result := v_spec*exp( n_spec*ln( scalar ) );
end;
//-----------------------------------------------
function  Tfigure.get_value_spec : double;
begin
result := v_spec;
end;
//-----------------------------------------------
function  Tfigure.get_color : Tcvet;
begin
result := color;
end;
//-----------------------------------------------
constructor  Tfigure.birth ( center0 : Tvector; color0 : Tcvet;
                             n_spec0, v_spec0 : double );
begin
center  := center0;
color   := color0;
n_spec  := n_spec0;
v_spec  := v_spec0;
normal  := Tvector.birth ( 0, 0, 0 );
reflected := Tvector.birth ( 0, 0, 0 );
end;
end.  // конец файла figureUnit.pas
```

Класс Tsphere (сфера)

```
// файл sphereUnit.pas
unit sphereUnit;
interface
uses  figureUnit,  vectorUnit,  colorUnit;
//-----------------------------------------------
type  Tsphere = class( Tfigure )
protected
radius : double;
procedure set_normal( point : Tvector ); override;
public
function  intersection ( ray_origin : Tvector;
                         ray_direction : Tvector ): double;
constructor  birth ( center0 : Tvector; color0 : Tcvet;
                     n_spec0, v_spec0, radius0 : double );
end;
```

```
//-------------------------------------
 implementation
//-------------------------------------
// определяет нормаль к сфере в т. point :
// N = OPoint - OCenter, normal = N/|N|
//-------------------------------------
 procedure   Tsphere.set_normal ( point : Tvector );
 var  tmp :  Tvector;
 begin
tmp :=  mult_on_scal ( -1.0,  center  );
 normal.Free;
normal :=  summ_vectors(  point,  tmp );
 tmp.Free;
  normal.normalization;
end;
//-------------------------------------
// Возвращает множитель t, на который нужно умножить на-
// правляющий вектор луча  ray_direction,  чтобы луч попал
// на сферу, если пересечения нет, то возвращает -1.0.
// Множитель t определяется из векторного уравнения:
// | ray_origin  + t* ray_direction  - Center_sphere | = radius,
//-------------------------------------
 function     Tsphere.intersection  (  ray_origin      :  Tvector;
                                       ray_direction   :  Tvector
                                    ):  double;
 var   Center_sphere,  V: Tvector;
       t,  scalar_dV,  d_sqr,  v_sqr,  discriminant  :  double;
 begin
 Center_sphere  :=  mult_on_scal ( -1.0,  center  );
V               :=  summ_vectors(  ray_origin,  Center_sphere  );
scalar_dV       :=  dot_product (  ray_direction,  V );
d_sqr           :=  sqr(  ray_direction.get_module  );
V_sqr           :=  sqr(  V.get_module );
 discriminant   :=  sqr(scalar_dV) - d_sqr*(V_sqr - sqr(radius));
    if  ( discriminant  < 0 )  then  t := -1.0
    else  t := -scalar_dV - sqrt(  discriminant  )/d_sqr;
    if  ( abs(  t  ) < 1.0E-20 ) then  t := -1;
 Center_sphere.Free;
V.Free;
 result  :=  t;
end;
//-------------------------------------
 constructor  Tsphere.birth (  center0 :  Tvector;  color0 :  Tcvet;
                               n_spec0,  v_spec0,  radius0 :  double );
```

```
begin
 inherited  birth ( center0,  color0,  n_spec0,  v_spec0 );
radius  :=  radius0;
end;
//-------------------------------------------------
end. // конец файла sphereUnit.pas
```

Трассировщик

```
// файл traceUnit.pas, инициализация и трассировка сцены
unit  traceUnit;
 interface
uses  vectorUnit,  figureUnit,  rayUnit,  colorUnit,  sphereUnit;

const numb_spheres     = 5;          // количество сфер;
      depth_recursion  = 12;         // глубина рекурсии;
      max_dist         = 1000.0;     // макс. длина трассировки;

var   backgr_color     : Tcvet;      // цвет фона;
      eye_position     : Tpoint3D;   // положение камеры;
      lamp             : Tray;       // источник света;
      sphere           : array [1 ..numb_spheres ] of Tsphere;
      recursion        :  integer;   // № рекурсии;

procedure  init_scene;
function   trace ( ray_source,  ray_direction : Tvector ): Tcvet;
procedure  delete_scene;

implementation
//-------------------------------------------------
procedure  init_scene;                 // эти данные лучше хранить
var  center :  Tvector;                // в файле специального
     color  :  Tcvet;                  // формата описания сцены;
begin
backgr_color.R  := 10;  backgr_color.G  := 10;
backgr_color.B  := 30;
lamp       :=  Tray.birth ( 0, 4, 2, 255, 255, 255 );
center     :=  Tvector.birth ( 0, -3, 10 );
color.R    := 255;  color.G  := 255;  color.B  := 255;
sphere [1] :=  Tsphere.birth ( center,  color,  5, 0.4,  4.0  );
center     :=  Tvector.birth ( 3.0, -1.0, 5 );
color.R  := 5;  color.G  := 255;  color.B  := 5;
sphere [2] :=  Tsphere.birth ( center,  color,  1, 0.01,  1.5 );
center     :=  Tvector.birth ( -3.0, -1.0, 5 );
```

```
color.R := 55; color.G := 255; color.B := 255;
sphere[3] := Tsphere.birth ( center, color, 5, 0.01, 1.5 );
center    := Tvector.birth ( 0, 3.0, 8 );
color.R := 255; color.G := 10; color.B := 10;
sphere[4] := Tsphere.birth ( center, color, 2, 0.1, 2.1 );
center    := Tvector.birth ( 0, -2.5, 4 );
color.R := 255; color.G := 255; color.B := 10;
sphere[5] := Tsphere.birth ( center, color, 1, 0.08, 1.5 );
end;
//-------------------------------------
//  трассировщик
//-------------------------------------
function trace ( ray_source, ray_direction : Tvector ): Tcvet;

var t, tt, diff_koeff, spec_koeff, min_t, spec : double;
    color, diff_color, spec_color, plus_color : Tcvet;
    SHADOWED : boolean;
    trace_point, light_direction, reflect, tmp,tm : Tvector;
    i, id : integer;

begin
inc( recursion );
color := backgr_color;
id    := numb_spheres + 1;
min_t := max_dist;
  for i := 1 to numb_spheres do
  begin
  t := sphere[i].intersection ( ray_source, ray_direction );
    // ближайшее пересечение сферы с индексом
    // id с лучом на расстоянии min_t:
    if ( t >= 0 ) AND ( t < min_t ) then
    begin
    min_t := t;
    id    := i;
    color := sphere[id].get_color;
    end;
  end;
  if ( id = numb_spheres + 1 ) // луч никуда не попал
  then color := backgr_color   // -- вернуть цвет фона,
  else                         // иначе -- расчёт цвета
  begin                        // точки пересечения:
  tmp := mult_on_scal ( min_t, ray_direction );
  trace_point := summ_vectors( ray_source, tmp );
  tmp.Free;
```

```
tmp := mult_on_scal( -1, trace_point );
tm := lamp.get_source;
 light_direction := summ_vectors( tmp, tm );
tmp.Free; tm.Free;
  light_direction.normalization;
// отражённый луч:
 reflect := sphere[id].get_reflected ( ray_direction,
                                       trace_point );
// освещение - затенённость другой сферой:
SHADOWED := FALSE;
    for i := 1 to numb_spheres do
    begin
    tt := sphere[i].intersection ( trace_point,
                                   light_direction );
       if ( ( tt > 0 ) AND ( i <> id ) ) then
       begin
       SHADOWED := TRUE;
        color := mult_colors ( sphere[id].get_color, 0.2 );
        break;
       end;
    end; // конец теста затенённости;
    if ( SHADOWED = FALSE ) then
    begin
    // освещение - диффузное освещение:
     diff_koeff := sphere[id].get_diff ( trace_point,
                                         light_direction );
     diff_color := mult_colors ( lamp.get_color, diff_koeff );
    color := mix_colors ( color, diff_color );
    // освещение - specular-ое освещение:
       if ( sphere[id].get_value_spec > 0 ) then
       begin
        spec_koeff := sphere[id].get_spec ( trace_point,
                                            ray_direction,
                                            light_direction );
          if ( ( spec_koeff > diff_koeff )
                AND ( diff_koeff > 0 ) ) then
          begin
           spec_color := mult_colors ( lamp.get_color, spec_koeff );
           spec_color := mix_colors ( spec_color,
                                      sphere[id].get_color );
           color := mix_colors ( color, spec_color );
          end;
       end;
     end; // if ( SHADOWED = FALSE )...
```

```
        // отражение (рекурсивные вызовы):
        if ( recursion <= depth_recursion ) then
        begin
        spec := sphere[id].get_value_spec;
            if ( spec > 0.0 ) then
            begin
             plus_color := trace( trace_point, reflect );
             plus_color := mult_colors( plus_color, spec );
             color := mix_colors( color, plus_color );
            end;
          end;
       trace_point.Free;
        light_direction.Free;
       reflect.Free;
     end; // else if( id = numb_spheres + 1 ) ...
 result := color;
end;
//-------------------------------------------
procedure delete_scene;
var i: integer;
begin
lamp.Free;
    for i := 1 to numb_spheres do
    sphere[i].Free;
end;
//-------------------------------------------
end. // конец файла traceUnit.pas
```

Головной модуль

```
// головной файл mainUnit.pas
unit mainUnit;
 interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, vectorUnit, figureUnit, rayUnit,
  colorUnit, ComCtrls, ExtCtrls, StdCtrls, traceUnit, Menus,
 ExtDlgs;
//-------------------------------------------
type TmainForm = class( TForm )
Panel          : TPanel;
StatusBar      : TStatusBar;
renderButton   : TButton;
cameraBox1     : TGroupBox;
```

```
cameraEdit_x    : TEdit;
cameraEdit_y    : TEdit;
cameraEdit_z    : TEdit;
Label1          : TLabel;
Label2          : TLabel;
Label3          : TLabel;
MainMenu1       : TMainMenu;
menu_file       : TMenuItem;
menu_exit       : TMenuItem;
menu_help       : TMenuItem;
menu_about      : TMenuItem;
SavePictDialog  : TSavePictureDialog;
save_picture    : TMenuItem;
procedure  FormPaint( Sender: TObject );
procedure  renderButtonClick ( Sender: TObject );
procedure  menu_exitClick (Sender: TObject);
procedure  menu_aboutClick(Sender: TObject);
procedure  save_pictureClick (Sender: TObject);
procedure  FormCreate(Sender: TObject);
procedure  FormCanResize(      Sender    : TObject;
                           var NewWidth,
                               NewHeight: Integer;
                           var Resize    : Boolean   );
private
{ Private declarations }
public
{ Public declarations }
end;
//-------------------------------------------------
var  mainForm                  : TmainForm;
     ImageWidth, ImageHeight   : integer;
     bitmap                    : TBitmap;
     EmptyBMP                  : boolean;
implementation
{$R *.dfm}
//-------------------------------------------------
// обновление холста при перерисовке формы
//-------------------------------------------------
procedure TmainForm.FormPaint( Sender: TObject );
begin
 canvas.Brush.Color  := mainForm.Color;
 canvas.FillRect ( rect ( 0, 0, mainForm.ClientWidth,
                                mainForm.ClientHeight ) );
ImageWidth := mainForm.ClientWidth - panel.Width - 20;
```

```
ImageHeight := mainForm.ClientHeight - StatusBar.Height - 20;
Canvas.Rectangle( 10, 10, ImageWidth + 10, ImageHeight + 10 );
StatusBar.SimpleText := ' ';
    if ( EmptyBMP = FALSE ) then Canvas.Draw( 10, 10, bitmap );
end;
//-------------------------------------
// РЕНДЕР
//-------------------------------------
procedure TmainForm.renderButtonClick( Sender: TObject );
var  i, j          : integer;
     x, y          : double;
     ray           : Tvector;
     ray_direction : Tvector;
     R_G_B         : Tcvet;
begin
     try     // обработка ошибок ввода данных:
     begin   // установка камеры:
      eye_position.x := StrToFloat( cameraEdit_x.Text );
      eye_position.y := StrToFloat( cameraEdit_y.Text );
      eye_position.z := StrToFloat( cameraEdit_z.Text );
     end
     except on EConvertError
     do  begin
         ShowMessage('Неверный ввод данных' + #13#10 +
                     'проверьте десятичную ./,');
         RenderButton.Enabled := TRUE;
         exit;
         end;
     end;    // конец обработки ошибок ввода данных;
 init_scene;
RenderButton.Enabled := FALSE;
bitmap          := TBitmap.Create;
bitmap.Width    := ImageWidth;
bitmap.Height   := ImageHeight;
ray := Tvector.birth ( eye_position );
    for  i := 10 to ImageHeight + 10 do
    begin
    y := -(i/( ImageHeight - 1 )*2 - 1 );
     StatusBar.SimpleText := ' трассируется строка '+IntToStr(i)
                             +'/'+IntToStr( ImageHeight ) ;
       for  j := 10 to ImageWidth + 10 do
       begin
       x:=(j/(ImageWidth - 1) - 0.5)*2*ImageWidth/ImageHeight;
        ray_direction := Tvector.birth ( x, y, 3 );
```

```
          ray_direction.normalization;
         recursion  := 1;
        R_G_B := trace( ray,   ray_direction   );
         ray_direction.Free;
        Canvas.Pixels [ j,  i ]:=RGB( R_G_B.R, R_G_B.G, R_G_B.B );
        bitmap.Canvas.Pixels [j-10, i-10] := RGB( R_G_B.R,
                                                  R_G_B.G, R_G_B.B );
       end;
     end;
 ray.Free;
 delete_scene;
EmptyBMP := FALSE;
RenderButton.Enabled  := TRUE;
 StatusBar.SimpleText  := 'конец трассировки';
end;
//-------------------------------------
procedure  TmainForm.menu_exitClick( Sender: TObject );
begin
 close;
end;
//-------------------------------------
procedure  TmainForm.menu_aboutClick(Sender: TObject);
begin
ShowMessage( 'Трассировка лучей' + #13#10 + 'учебный пример'
                + #13#10 + 'Зенкин В.И., 2005 г.');
end;
//-------------------------------------
procedure   TmainForm.save_pictureClick ( Sender: TObject );
begin
    if  ( EmptyBMP ) then ShowMessage('нет картинки!')
    else
        if   (  SavePictDialog.Execute   )
        then   bitmap.SaveToFile (  SavePictDialog.FileName  );
end;
//-------------------------------------
procedure  TmainForm.FormCreate( Sender: TObject );
begin
EmptyBMP := TRUE;
end;
//-------------------------------------
// перерисовка картинки при изменении размера формы
//-------------------------------------
procedure  TmainForm.FormCanResize(     Sender      : TObject;
                                    var  NewWidth,
```

```
                                        NewHeight :  Integer;
                                    var Resize     : Boolean  );
begin
  if ( EmptyBMP = FALSE ) then Canvas.Draw( 10, 10, bitmap );
end;
end. // конец файла mainUnit.pas
```

5.2. Цикады и простые числа

Базовый класс «Особь»

```
// Файл thing.h. Базовый класс "Особь"
//-------------------------------------
#ifndef  _THING_H_
#define  _THING_H_
typedef   unsigned long long int    Lint;
//-------------------------------------
class   thing
{
 protected:
 Lint       term;            // период появления особи;
 Lint       min_bound;       // границы выбираемого особью
 Lint       max_bound;       // периода, далее зависят от L;
 Lint       HOD( Lint a,
                 Lint b );  // Н.О.Д. (a,b);
 // приспособленность особи:
  virtual   double   fitness ( const Lint &p, const Lint &q ) = 0;
  // выбор наилучшего периода:
 void       best_period ( const Lint &enemy_term );
 public:
 Lint       period ();          // возвр. выбранный период;
            thing ( const Lint &enemy_term );
  virtual  ~thing() {};
};
#endif
```

```
// Файл thing.cpp. Базовый класс "Особь"
//-------------------------------------
#include "thing.h"
#include <iostream>
//-------------------------------------
// Наибольший общий делитель чисел a, b.
//-------------------------------------
Lint   thing::HOD( Lint a, Lint  b  )
{
```

```
    while ( ( a > 0 ) && ( b > 0 ) )
        if   ( a > b )  a = a - b;
        else            b = b - a;
return ( a + b );
}
//-----------------------------------------
// Подбор наилучшего периода для особи
//-----------------------------------------
void    thing :: best_period ( const Lint &enemy_term )
{
 term = min_bound;
   for ( Lint p = min_bound + 1; p <= max_bound; ++p )
     if ( fitness (p, enemy_term) > fitness (term, enemy_term) )
      term = p;
}
//-----------------------------------------
Lint   thing :: period ()
{
 return   term;
}
//-----------------------------------------
thing :: thing ( const Lint &enemy_term )
{
}
```

Класс «Хищник»

```
// Файл predator.h. Класс "Хищник"
//-----------------------------------------
#ifndef  _PREDATOR_H_
#define  _PREDATOR_H_
#include  "thing.h"
//-----------------------------------------
class  predator : public  thing
{
  virtual  double  fitness ( const Lint &p,   // приспособлен-
                             const Lint &q ); // ность Хищника;
 public :
            predator ( const Lint &L, const Lint &enemy_term );
  virtual  ~predator ()  {};
};
#endif
```

```
// Файл predator.cpp. Класс "Хищник"
```

```
//-----------------------------------------
#include "predator.h"
//-----------------------------------------
// приспособленность Хищника
//-----------------------------------------
double predator::fitness( const Lint &p, const Lint &q )
{
 return ( 2.0*HOD( p, q )/q - 1.0 );
}
//-----------------------------------------
predator::predator( const Lint &L, const Lint &enemy_term )
          : thing( enemy_term )
{
 min_bound = 2;
 max_bound = L/2 + 1;
  best_period( enemy_term );
}
```

Класс «Жертва»

```
// Файл prey.h. Класс "Жертва"
//-----------------------------------------
#ifndef _PREY_H_
#define _PREY_H_
#include "thing.h"
//-----------------------------------------
class prey: public thing
{
  virtual double fitness( const Lint &p,    // приспособлен-
                          const Lint &q );  // ность Жертвы;
 public:
           prey( const Lint &L, const Lint &enemy_term );
  virtual ~prey() {};
};
#endif
```

```
// Файл prey.cpp. Класс "Жертва"
//-----------------------------------------
#include "prey.h"
//-----------------------------------------
// приспособленность Жертвы
//-----------------------------------------
double prey::fitness( const Lint &p, const Lint &q )
{
```

```
  return   ( 1.0 - 2.0*HOD( p, q )/q );
}
//-----------------------------------------
prey :: prey( const Lint &L, const Lint &enemy_term )
       : thing ( enemy_term )
{
 min_bound = L/2 + 2;
 max_bound = L;
  best_period ( enemy_term );
}
```

Головной файл

```
// Файл bioprm_main.cpp
// Биологическая модель типа "хищник-жертва"
// с простыми периодами
//-----------------------------------------
#include <iostream>
#include <stdio.h>
#include "predator.h"
#include "prey.h"
int main( int argc, char *argv[] )
{
  const Lint L      = 16;
  Lint pred_period  = 2;
  Lint prey_period  = L/2 + 2;
  prey *pprey       = new prey( L, pred_period );
  prey_period       = pprey->period();
  predator *ppred   = new predator( L, prey_period );
  pred_period       = ppred->period();
  std :: cout << "prey period = " << prey_period
              << ", predator = "  << pred_period << std::endl;
  delete   pprey;
  delete   ppred;
  std :: cout << std::endl << "Game over, press Enter";
  getchar ();
  return   0;
}
```

5.3. ИС-2 против PzVI Tiger

```
--> eq1 : ' diff( r(t), t ) = -9*0.2*b(t) $
--> eq2 : ' diff( b(t), t ) = -3*0.9*r(t) $
/*   начальные условия:                    */
--> t0 : 0  $
```

```
--> r0 : 20 $
--> b0 : 20 $
--> tN : 1  $
--> atvalue( r(t), t = t0, r0 ) /* r(t0) = r0 */ $
--> atvalue( b(t), t = t0, b0 ) /* b(t0) = b0 */ $
/*    решение системы дифф. уравнений:                    */
--> d : desolve( [ eq1, eq2 ], [ r(t), b(t) ] );
--> rr : rhs( d[1] ) $
--> bb : rhs( d[2] ) $
/*    Результат боя - кто и когда победил. Синие          */
/*    уничтожены? Решение ур. b(t) = 0 методом Ньютона:   */
--> tр : - 100       /* момент победы или поражения */ $
    load( "mnewton" )$
--> newton_b : mnewton( [bb = 0], [t], [t0] ) $
--> if newton_b # []
    then block( tр : rhs( newton_b[1][1] ),
                print ("Синие уничтожены через ",tр," мин."),
                print ("У Красных осталось ",
                round( ev( rr, t = tр, float )),"танков." )
              );
/*    Красные уничтожены? Решение b(t) = 0 мет. Ньютона: */
--> newton_r : mnewton( [rr = 0], [t], [t0] ) $
    if newton_r # []
    then block( tр : rhs( newton_r[1][1] ),
                print ("Красные уничтожены через ",tр,"мин."),
                print ("У Синих осталось ",
                round(ev( bb, t = tр, float )),"танков.")
              );
--> B(t) := if t <= tр then bb else ev( bb, t = tр ) $
--> R(t) := if t <= tр then rr else ev( rr, t = tр ) $
--> plot2d( [ R(t), B(t) ], [ t, t0, tN ],
           [ gnuplot_preamble, "set grid ;"],
           [ color, red, blue, gray ],
           [ xlabel, "t"], [ legend, "r(t)", "b(t)"] )$
```

5.4. Транслятор L2eps с L-языка на PS

L2eps.y (грамматика и действия)

```
/*****************************************************
 * Файл Lsystem.y.
 * Грамматика и соответствующие действия для трансляции с
 * языка L-систем на EPS. После обработки Bison'ом сгенериру-
 * ется головной файл программы.
 *****************************************************/
```

```
%{
 #include    <stdio.h>
 #include    <ctype.h>
 #include    "Lsyst.h"
 #include    "EPS.h"

  int       numb_symb = -1;   /* № символа в комм. строке */
  stack   * top_stack  = NULL;
  const     double    PI = 3.14159265358979323846;
%}
%%
str  :    '['       str       ']'
     |    str       '['       str      ']'
     |    symb
     |    str       symb
     ;

symb :    'F'              { EPS_lineto();                              }
     |    '+'              { initL.curr_Angle  += initL.Angle;           }
     |    '-'              { initL.curr_Angle  -= initL.Angle;           }
     |    '|'              { initL.curr_Angle  = -initL.curr_Angle;      }
     |    'f'              { EPS_moveto();                              }
     ;
%%
/*-------------------------------------
 * Лексический анализатор
 *-------------------------------------*/
yylex()
{
++numb_symb;
int  c = initL.command[ numb_symb ];
   /* текущие значения координат и угла толкаем в стек: */
    if  (  initL.command[ numb_symb ] == '[' )
    {
     struct   CURR_DATA curr_data;
     curr_data.x     = eps.currX;
     curr_data.y     = eps.currY;
     curr_data.ugol  = initL.curr_Angle;
     top_stack  = new_link(  top_stack ,  curr_data  );
    }
   /* значения координат и угла выталкиваем из стека     *
    * и делаем их текущими:                              */
    if  (  initL.command[ numb_symb ] == ']')
    {
```

```
        eps.currX          = top_stack->data.x;
        eps.currY          = top_stack->data.y;
        EPS_movetoCurr();
        initL.curr_Angle   = top_stack->data.ugol;
        top_stack          = delete_link ( top_stack );
        }
        if ( numb_symb == strlen( initL.command ) )
        return   0;      /* дошли до конца --- успешный разбор */
        else
        return   c;
}
/*-------------------------------------------------------------*/
int main ( int argc, char* argv[] )
{
        if ( argc < 4 )
        {
        printf ( "неверные аргументы, нужно:\n" );
        printf ( "L2eps.exe имя_L-файла.l имя_EPS-файла.eps " );
        printf ( "Order Angle0\n" );
        printf ( "прессуй <Enter>");
        getchar ();
        exit ( 1 );
        }
char*   Lfile_name     = argv[1];
char*   EPSfile_name   = argv[2];
int     Order          = atoi ( argv[3] );
        if  ( argc == 5 )
        {
        double  ugol_0          = atof ( argv[4] );
                if   ( ugol_0 )  initL.curr_Angle = PI/ugol_0;
                else             initL.curr_Angle = 0;
        }
        else    initL.curr_Angle = 0;
top_stack          = (stack*) malloc( sizeof ( stack ) );
top_stack->link    = NULL;
        if ( !read_Lsyst ( Lfile_name, Order ) )
        {
        printf ( "\nсчитан файл %s\n", Lfile_name          );
        printf ( "Order    = %d\n",    Order               );
        printf ( "Angle0   = %f\n",    initL.curr_Angle    );
        printf ( "Angle    = %f\n",    initL.Angle         );
        printf ( "Axiom    = '%c'\n",  initL.Axiom         );
        printf ( "command  = '%s'\n",  initL.command       );
          if ( ! substitute ( initL.Axiom, Order ) )
```

```
            {
             printf ("подстановка %d раз(а)\n", Order              );
             printf ("command = '%s'\n",          initL .command );
            }
            else
            {
             printf ( "сбой при подстановке в ком. строке" );
             exit ( 1 );
            }
        }
        else
        {
         printf ( "сбой при чтении файла %s\n", Lfile_name );
        exit ( 1 );
        }
  printf ( "начинаю запись EPS файла\n" );
 EPS_begin( EPSfile_name );
  printf ( "начинаю разбор команд\n" );
        if ( !yyparse() )    printf ( "разбор прошёл успешно\n" );
        else                 printf ( "синтакс. ошибка\n"       );
 EPS_end();
  printf ( "OK с EPS-файлом %s,\nпрессуй <Enter>",
            EPSfile_name );
  getchar ();
  return   0;
  }
/*-----------------------------------
 * вызывается yyparse в случае ошибки
*-----------------------------------*/
  yyerror ( mes )
  char   *mes;
  {
   printf ( "%s\n", mes );
  }
/** конец файла Lsystem.y. ********************************/
```

Lsyst.h (L-системы, заголовки функций)

```
/*****************************************************
 * Файл Lsyst.h. Структура Lsystem и заголовки функций для
 * чтения файла с описанием L-system. Читают входной L-файл,
 * начиная с символа "{" до символа "}" и пропуская все
 * комментарии (от ";" до конца строки). В случае успеха
```

```
 * опеделяются параметры Angle, Axiom и командная строка, в
 * которой произвоится подстановка символа Axiom Order раз.
 ************************************************************/
 #include   <stdio.h>
 #include   <ctype.h>
 #include   <stdlib.h>
 #include   <string.h>
 #include   <math.h>

 #define   array_size   50000
  struct    Lsystem
            {
            double     Angle;
            double     curr_Angle;
            char       Axiom;
            char       command[array_size];
            }  initL;
 typedef    struct    CURR_DATA
            {
            double     ugol;
            double     x;
            double     y;
            }  curr_data;
 typedef    struct    STACK        /* стек данных для L-системы */
            {
             struct    CURR_DATA data;
             struct    STACK     *link;
            }  stack;
/*-----------------------------------------
 * Сообщение об ошибке.
 *----------------------------------------*/
  inline   void   noOK( char *message, char* str );
/*-----------------------------------------
 * Анализ строк вх. L-файла и заполнение полей Lsystem. При
 * обнаружении ошибки возвращает 1, иначе 0. Вызывается
 *  read_Lsyst.
 *----------------------------------------*/
  static   int    init_Lsystem( char *str );
/*-----------------------------------------
 * В строке InitL.command символ X заменяется на InitL.command
 * Order раз. В случае успеха возвращает 0, иначе -- 1.
 *----------------------------------------*/
 int     substitute ( char  X, int  Order );
/*-----------------------------------------
```

```
 * Чтение файла с описанием L-system. Читает входной файл,
 * начиная с символа "{" до символа "}" и пропуская ком-
 * ментарии (от ";" до конца строки), пробелы подавляются.
 * Считанные из файла "правильные" строки передаются на ана-
 * лиз init_Lsystem . При обнаружении ошибки возвращает 1,
 * иначе 0.
 *-------------------------------------*/
 int    read_Lsyst( char  *file_name , int Order );
/*-------------------------------------
 * Добавляет новое звено с полем данных v в стек.
 *-------------------------------------*/
 struct STACK* new_link( struct STACK     *ptr ,
                         struct CURR_DATA v   );
/*-------------------------------------
 * Уничтожает звено стека, на которое указывает ptr и прод-
 * вигает указатель к основанию стека.
 *-------------------------------------*/
 struct STACK* delete_link( struct STACK *ptr );

/* конец файла Lsyst.h*********************************/
```

Lsyst.c (L-системы, функции)

```
/***********************************************
 * Файл Lsyst.c. Функции чтения файла с описанием L-system.
 * Читают входной L-файл, начиная с символа "{" до символа "}"
 * и пропуская все комментарии (от ";" до конца строки). В
 * случае успеха определяются параметры Angle, Axiom и команд-
 * ная строка, в которой произвоится подстановка символа Axiom
 * Order раз.
 ***********************************************/
 #include "Lsyst.h"
 #include <math.h>
 const  double   pi = 3.14159265358979323846;
/*-------------------------------------
 * Сообщение об ошибке.
 *-------------------------------------*/
 inline  void  noOK( char *message, char* str )
 {
  printf ( "ОШИБКА: %s '%s'\nпресс <Enter>\n", message, str );
  getchar ();
 }
/*-------------------------------------
 * Анализ строки вх. файла и заполнение полей Lsystem. При
```

```
* обнаружении ошибки возвращает 1, иначе 0. Вызывается
*  read_Lsyst .
*-------------------------------------------------------*/
 static   int    init_Lsystem ( char  *str  )
{
char   *Axiom                = "Axiom";
char   *Angle                = "Angle";
char    tmp_str[ array_size ] = "";
int      right_str           = 1; /* флаг 'неправильной' стр.  */
      /* поиск в строке  str  и определение параметра Angle:   */
       if  (  strstr  (  str ,  Angle  ) != NULL  )
       {
       int    i  =  strlen ( Angle  )  -  1;
       tmp_str[0]  =  '\0';
            while  ( ++i <= strlen(  str  ) )
            {
            if  (    iscntrl  (  str[i]  )  )
            continue ;
                 if   (  isdigit  (  str[i]  )  ||  str[i]  ==  '.'  )
                  sprintf  (  tmp_str ,  "%s%c", tmp_str,  str[i]  );
                 else
                 {
                 noOK( "недопустимый символ в строке", str  );
                  return    1;
                 }
            }
        right_str   = 0;
       double   a  =  strtod ( tmp_str ,  NULL );
            if     ( a )   initL .Angle = pi/a;
            else           initL .Angle = 0;
       }/* Angle определён (если он правилен, иначе == 0). */
        /* поиск в строке  str  и определение символа Axiom: */
       if  (  strstr  (  str ,  Axiom  ) != NULL  )
       {
       int    i  =  strlen ( Axiom );
             if   (  isalpha  (  str[i]  )  )
                    initL .Axiom = str[i];
             else   {
                    noOK( "недопустимый символ в строке", str  );
                     return    1;
                    }
             while  ( ++i <= strlen(  str  ) )
             {
                  if   (  iscntrl  (  str[i]  )  )
```

```
                    continue ;
                else {
                    initL .Axiom = ' ';
                    noOK("недопустимый символ в строке ", str );
                    return   1;
                    }
            }
        right_str  = 0;
       }/* Axiom определён (если он правилен, иначе = ' '). */
       /* поиск в строке  str  командной строки символов:      */
       if  (  strstr  (  str ,  "=" )  && initL.Axiom != ' ' )
       {
            if  (  initL .Axiom != str [0]  ||  (  str [1])  !=  '=' )
            {
            noOK( "недопустимый символ в строке ",  str  );
            return   1;
            }
            else
            {
            int  j  = 1;
                while ( ++j <= strlen(  str  ) )
                {
                    if  (  ! iscntrl  (  str [j ] ) )
                    sprintf  (  initL .command, "%s%c",
                                initL .command, str [j ]  );
                }
            }
            if  (  strlen  (  initL .command ) < 1 )
            {
            noOK( "пустая коммандная строка ",  str  );
            return   1;
            }
        right_str  = 0;
       }
       if  (  right_str  && strlen(  str  ) > 1 )
       {
       noOK( "недопустимая строка символов ",  str  );
       return   1;
       }
 return   0;
 }
/*-------------------------------------------------
 * В строке  InitL .command символ X заменяется на InitL.command
 * Order раз. В случае успеха возвращает 0, иначе -- 1.
```

```
*-----------------------------------*/
int     substitute ( char   X, int   Order )
{
int     i                           = -1;
char   command0[ array_size  ] = "";
     /* начальное значение коммандной строки */
     /* сохраняем в command0:                */
     while  ( ++i <= strlen(  initL .command ) )
     command0[i] = initL.command[i];
     /* вставка символа или строки вместо X: */
     while  ( --Order > 0 )
     {
     int    i = -1;
     char    tmp_str[   array_size   ] = "";
          while  ( ++i <= strlen(  initL .command ) )
          {
          char   symb = initL.command[i];
               if   ( symb == X )
               {
                    if   (  strlen (tmp_str) +  strlen ( initL .command)
                         >  array_size   )
                    return    1;
                sprintf (  tmp_str ,  "%s%s", tmp_str,  command0 );
               }
               else
               {
                    if  (  strlen (  tmp_str  ) + 1 >  array_size   )
                    return    1;
                sprintf (  tmp_str ,  "%s%c", tmp_str,  symb );
               }
          } /*  ... while  ( ++i <= strlen(  str   )  )...* /
       int    j  = -1;
          while( ++j <= strlen(  tmp_str  ) )
          initL .command[j] = tmp_str[ j ];
       } /*  ... while  ( --Order >= 0 )... */
 return    0;
}
/*-----------------------------------
 * Чтение файла с описанием L-system. Читает входной файл,
 * начиная с символа "{" до символа "}" и пропуская ком-
 * ментарии (от ";" до конца строки), пробелы подавляются.
 * Считанные из файла "правильные" строки передаются на ана-
 * лиз  init_Lsystem .  При обнаружении ошибки возвращает 1,
 * иначе  0.
```

```
*-------------------------------------------------------------------*/
int    read_Lsyst( char   *file_name,  int   Order )
{
initL.Angle                    = 0.0;     /* значения     */
initL.Axiom                    = '';      /* по умол-     */
initL.command[0]               = '\0';    /* чанию        */
char    symb                   = '';
char     curr_line [ array_size ] = "";
FILE  * file ;
     if  ( !(  file  = fopen(  file_name,  "r" ) ) )
     {
     noOK( "Не могу открыть входной L-файл ", file_name );
     exit ( 1 );
     }
     /* пропустить всё до начального символа '{'          */
     while  ( ( symb != '{' ) && ( symb != EOF ) )
     symb = fgetc(  file  );
     if  ( symb == EOF )
     {
     noOK( "Нет начального символа ",  "'{'"  );
     return   1;
     }
     else    /* обнаружено начало:   '{'   */
     symb   = fgetc(  file  );
    /* читаем L-файл по одному символу, сформированные *
     * строки отдаём на анализ функции init_Lsystem:    */
     while  ( symb != EOF )
     {
          switch  ( symb )
          {
          case   ';'  :  /* пропустить комментарии  *
                         * от  ';'  до конца строки: */
                         {
                              while  ( symb != '\n' )
                              symb = fgetc(  file  );
                         ungetc( symb,  file  );
                         break;
                         }
          case   '{'  :  {
                         noOK(  "'{...{'",   " "  );
                         return   1;
                         }
          case   '}'  :  return   0;
          case   '\n':   {
```

```
                        if ( strlen ( curr_line ) > 0 )
                        {
                            if ( init_Lsystem ( curr_line ) == 1)
                            return 1;
                        }
                    curr_line [0] = '\0';
                    break;
                    }
                    /* считанные символы накапливаются *
                    * в строке curr_line , пробелы *
                    * пропускаются: */
        default : {
                        if ( symb != ' ' )
                        sprintf ( curr_line , "%s%c",
                                curr_line , symb );
                    break;
                    }
            } /*... switch ( symb )... */
        symb = fgetc( file );
            if ( symb == EOF )
            {
            noOK( "не закрыта скобка ", "'{'" );
            return 1;
            }
        } /*... while ( symb != EOF )... */
        if ( fclose ( file ) )
        {
        noOK( "сбой при закрытии L-файла ", file_name );
        abort ();
        }
    return 0;
    }
/*-----------------------------------
 * Добавляет новое звено с полем данных v в стек.
 *-----------------------------------*/
 struct STACK* new_link( struct STACK *ptr ,
                         struct CURR_DATA v )
 {
 stack *temp = (stack*) malloc( sizeof ( stack ) );
   if ( !temp )
   {
    printf ( "не удалось выделить память!" );
   return NULL;
   }
```

```
 temp->data = v;
 temp->link = ptr;
 return   temp;
 }
/*-------------------------------------
 * Уничтожает звено стека, на которое указывает ptr и прод-
 * вигает указатель к основанию стека.
 *-------------------------------------*/
 struct  STACK* delete_link( struct  STACK *ptr )
 {
 stack   *temp = ptr;
 stack   *p    = ptr->link;
 free ( temp );
 return   p;
 }
/* конец файла Lsyst.c ***********************************/
```

EPS.h (EPS-формат, заголовки функций)

```
/*****************************************************
 * Файл EPS.h, Заголовки функций для работы с EPS файлом.
 *****************************************************/
 struct   EPS
          {
          FILE*    file ;
          char*    file_name ;
          char*     discription  ;
          int      maxX;          /* экстремальные          */
          int      maxY;          /* координаты             */
          int      minX;          /* размеров               */
          int      minY;          /* изображения            */
          double   currX;         /* текущие                */
          double   currY;         /* координаты             */
          double   segment;       /* длина отрезка рисования */
          } eps;
/*-------------------------------------
 * Заголовочные комментарии в начале EPS-файла.
 *-------------------------------------*/
 static  void  EPS_head_comments();
/*-------------------------------------
 * Заголовочные комментарии в конце EPS-файла.
 *-------------------------------------*/
 static  void  EPS_end_comment();
/*-------------------------------------
```

```
 * передвижение "пера" в текущие координаты.
 *-----------------------------------------*/
 void   EPS_movetoCurr();
/*-----------------------------------------
 * передвижение от текущей точки "пера" на расстояние segment
 * в направлении initL.curr_Angle.
 *-----------------------------------------*/
 extern   void   EPS_moveto();
/*-----------------------------------------
 * Рисует отрезок длины segment от текущей точки в направле-
 * нии initL.curr_Angle.
 *-----------------------------------------*/
 extern   void   EPS_lineto();
/*-----------------------------------------
 * Определение крайних значений ширины и высоты картинки.
 *-----------------------------------------*/
 inline   void   min_max( double  x, double  y );
/*-----------------------------------------
 * Открывает EPS файл name_file для записи, пишет в него нача-
 * льный блок комментариев. Инициализирует поля eps.
 *-----------------------------------------*/
 extern   void   EPS_begin( char* name_file );
/*-----------------------------------------
 * Записывает в конец EPS файла заключительный блок коммен-
 * тариев, закрывает файл, снова открывает для замены на-
 * чального блока комментариев с новыми размерами картинки.
 *-----------------------------------------*/
 extern   void   EPS_end();
/** конец файла EPS.h ***********************************/
```

5.5. Простейший клеточный автомат

```
#include <iostream>
#include <stdio.h>
#include "cell_bits.h"
using namespace std;
//-----------------------------------------
int  main( int argc, char *argv[] )
{
  const int    rule_number = 90;
  const Lint   number      = 1ULL << 31;
  const int    dimension   = 62;
  int          iteration   = 32;
  cout << endl << " Rule " << rule_number << endl << endl;
```

```
  cell_auto    *CA = new cell_auto ( rule_number,
                                     number,
                                     dimension    );
       if ( CA->get_error() )
       {
        cout << "Input error!" << endl;
        delete CA;
        getchar ();
        return 1;
       }
  boolvect * init   = CA->get_init_generation ();
       for ( int i = init->size() - 1; i >= 0; --i )
       {
        //cout << (*init)[ i ];
             if ( (* init )[ i ] ) cout << (char)254;
             else                  cout << " ";
       }
  cout << "   = " << CA->decimal( init ) << endl;
       for ( int i = 1; i <= iteration ; ++i )
       {
        CA->next_generation();
        boolvect *gen    = CA->get_curr_generation();
          // вывод в двоичном виде:
          for ( int i = gen->size() - 1; i >= 0; --i )
          {
           //cout << (*gen)[ i ];
                if ((*gen)[ i ] )   cout << (char)254;
                else                cout << " ";
          }
        // вывод в десятичном виде:
        cout << "   = " << CA->decimal( gen ) << endl;
       }
  delete CA;
 //getchar ();
  return 0;
}
```

Класс Клеточный автомат

```
//-------------------------------------
// Файл cell_bits.h.
//-------------------------------------
#ifndef   CELLS_BITS_H_
#define   CELLS_BITS_H_
```

```
#include <vector>
typedef unsigned long long int Lint;
typedef std::vector<bool> boolvect;
//-------------------------------------
class cell_auto
{
  boolvect *init_generation; // указ. на исх. поколение;
  boolvect *curr_generation; // указ. на текущ. поколение;
  const int L; // число клеток в строке;
  const int rule_number;
  int ERROR;
  public:
  bool rule(int i);
  void next_generation();
  Lint decimal(boolvect *bit_vector);
  boolvect *get_init_generation();
  boolvect *get_curr_generation();
  int get_error();
  cell_auto(const int &rulenumber,
            const Lint &number,
            const int &dimension_automata);
  cell_auto(const cell_auto &s);
  cell_auto &operator=(const cell_auto &s);
  virtual ~cell_auto();
};
#endif
//-- конец файла cell_bits.h ----------
```

```
//-------------------------------------
// файл cell_bits.cpp
//-------------------------------------
#include "cell_bits.h"
//-------------------------------------
cell_auto::cell_auto(const int &rulenumber,
                     const Lint &number,
                     const int &dimension_automata)
        : L(dimension_automata), rule_number(rulenumber%256)
{
  init_generation = new boolvect(L, 0); // <- 000...0;
  curr_generation = new boolvect(L, 0); // <- 000...0;
 ERROR = 0;
 // max число, которое влезет в строку, 2^63 -1:
 const Lint max_number = (1ULL << L) - 1;
     if (number > max_number)
```

```
        {
         ERROR = 1;
        }
        else
        {
             for ( int  i = L - 1; i >= 0; i-- )
             {
              (* init_generation )[ i ]= ((number & (1ULL<<i))>>i);
             }
          curr_generation = init_generation ;
        }
}
//-------------------------------------------------
 cell_auto :: cell_auto ( const cell_auto &s )
            : L( s.L ), rule_number( s.rule_number % 256 )
{
   init_generation = s. init_generation ;
  curr_generation = s. curr_generation ;
}
//-------------------------------------------------
 cell_auto & cell_auto :: operator =( const cell_auto &s )
{
        if ( this != &s )
        {
          init_generation = s. init_generation ;
         curr_generation = s. curr_generation ;
        }
  return * this ;
}
//-------------------------------------------------
 cell_auto ::~ cell_auto ()
{
        if ( init_generation ) delete init_generation ;
        if ( curr_generation ) delete curr_generation ;
}
//-------------------------------------------------
// Правила клеточного автомата ( rule update function ):
//-------------------------------------------------
bool cell_auto :: rule ( int i )
{
 bool          next_generation_i = 0;
 unsigned int  left_i            = i + 1;
 unsigned int  right_i           = i - 1;
        if ( i == L - 1 ) left_i = 0;          // краевые случаи,
```

```
      if ( i == 0      )   right_i  = L - 1;     // замыкание ленты;
  unsigned  int  prod_3cell  = 4*(* curr_generation )  [ left_i ]
                             + 2*(* curr_generation )[ i ]
                             + (* curr_generation )  [ right_i ];
      switch  (  prod_3cell  )
      {
       case  7:  next_generation_i =((rule_number  & (1<<7))>>7);
                 break;
       case  6:  next_generation_i =((rule_number  & (1<<6))>>6);
                 break;
       case  5:  next_generation_i =((rule_number  & (1<<5))>>5);
                 break;
       case  4:  next_generation_i =((rule_number  & (1<<4))>>4);
                 break;
       case  3:  next_generation_i =((rule_number  & (1<<3))>>3);
                 break;
       case  2:  next_generation_i =((rule_number  & (1<<2))>>2);
                 break;
       case  1:  next_generation_i =((rule_number  & (1<<1))>>1);
                 break;
       case  0:  next_generation_i =( rule_number  & 1  );
                 break;
        default :  ERROR = 1;
      }
  return     next_generation_i  ;
}
//-------------------------------------------
void    cell_auto  ::  next_generation  ()
{
  boolvect  * temp_generation  = new  boolvect (L,  0);   // <- 0...0;
      for  (  int    i  = 0;  i  < L;  i++ )
      {
       (* temp_generation )[ i ]  =  rule ( i );
      }
   curr_generation   =  temp_generation ;
}
//-------------------------------------------
boolvect *    cell_auto  ::   get_init_generation   ()
{
  return     init_generation  ;
}
//-------------------------------------------
boolvect *    cell_auto  ::  get_curr_generation  ()
{
```

```
  return     curr_generation ;
}
//--------------------------------------------------
Lint    cell_auto :: decimal( boolvect* bit_vector )
{
 Lint   decimal_number = 0;
       for ( int i = 0; i < L; ++i )
       {
            if ((* bit_vector )[ i ]) decimal_number += (1ULL << i);
       }
  return   decimal_number;
}
//--------------------------------------------------
int    cell_auto :: get_error ()
{
  return   ERROR;
}
// конец файла cell_bits .cpp ---------------------
```

V. ПРИЛОЖЕНИЯ

1. Программное обеспечение

Для программной реализации математических моделей — как учебных проектов, так и с учетом их возможного последующего развития, — необходимы следующие требования к программному обеспечению:

— бесплатность (по крайней мере, для некоммерческого использования);
— открытые исходные коды.

Дополнительно, желательна кроссплатформенность. Открытые исходные коды дают возможность проверить корректность и детально контролировать работу любого алгоритма, и, если нужно, изменить его код.

Для решения задач компьютерного моделирования данного сборника можно *рекомендовать* следующее программное обеспечение:

- Компиляторы, интегрированные среды разработки (IDE — Integrated Development Environment):
  - для Linux: C++ или любой другой компилятор из пакета GNU Compiler Collection (gcc.gnu.org); для Windows: MinGW (Minimalist GNU for Windows, mingw.org) с IDE Dev-C++ (bloodshed.net) или IDE Code::Blocks (codeblocks.org), wxDev-C++ (wxdsgn.sourceforge.net);
  - Free Pascal Compiler с IDE Lazarus (lazarus.freepascal.org) — свободный аналог проприетарной Delphi.
- Система компьютерной алгебры Maxima (maxima.sourceforge.net). Позволяет проводить аналитические и численные вычисления, строить графики. По набору возможностей система близка к таким коммерческим системам как Maple и Mathematica.
- Gnuplot (www.gnuplot.info). Программа для создания графиков, может работать интерактивно и выполнять скрипты, читаемые из файлов. Позволяет сохранять графики во множестве графических форматов, в том числе в векторных, EPS и SVG.
- Для вёрстки математических текстов можно использовать:
  - текстовый процессор LaTeX. Для платформы Windows рекомендуется дистрибутив MiKTeX (miktex.org) с IDE WinShell (winshell.org) или Led (latexeditor.org). Многие дистрибутивы операционной системы Linux изначально включают пакет LaTeX с IDE и различными редакторами .
  - пакет OpenOffice (openoffice.org/ru) — бесплатный аналог коммерческого Microsoft Office.

  Использование LaTeX позволяет добиться наиболее качественной вёрстки математических документов любой сложности.

2. Решения и исследования дифференциальных уравнений

Традиционно многие математические модели описываются *дифференциальными уравнениями* — соотношениями, связывающими независимую переменную, неизвестную функцию этой переменной и ее производные или дифференциалы до некоторого порядка включительно. Порядок наивысшей производной, входящей в дифференциальные уравнения, называется *порядком* этого дифференциального уравнения. *Решением* (*интегралом*) дифференциального уравнения называется функция, удовлетворяющая этому уравнению.

Следует специально отметить, что в теории дифференциальных уравнений под решением понимается не только явный вид неизвестной функции, но и её выражение *в квадратурах* — в виде интеграла от комбинации известных («элементарных», «стандартных») функций, который не обязательно выражается в элементарных функциях. Например, уравнение

$$xy' = e^x,$$

имеет решение в квадратурах

$$y = \int \frac{e^x}{x}\,\mathrm{d}x + C,$$

где C — константа, а интеграл в правой части равенства не выражается через элементарные функции («неберущийся» интеграл).

Такое решение, очевидно, не годится для целей математического моделирования, так как модель, как правило, должна описываться конкретными *числовыми* величинами. В таких случаях лучше применить численные методы решения.

Вообще, при математическом моделировании более-менее сложных объектов аналитическое решение соответствующих дифференциальных уравнений — скорее исключение, а не правило. Тем не менее, наличие общего решения дифференциального уравнения в замкнутом виде во многих случаях даёт значительные преимущества по сравнению с численными решениями. Перечислим некоторые случаи, когда аналитическое решение обыкновенных дифференциальных уравнений возможно, детально известные решения перечислены в справочниках [11], [12].

2.1. Уравнения с разделяющимися переменными

$$a(x)b(y)\mathrm{d}x + c(x)d(y)\mathrm{d}y = 0. \tag{1}$$

После почленного деления на $b(y)c(x)$ получим

$$\frac{a(x)}{c(x)}\,\mathrm{d}x + \frac{d(y)}{b(y)}\,\mathrm{d}y = 0,$$

разделение переменных: коэффицент при dx зависит только от x, а

коэффицент при dy — только от y. Решение находится из соотношения

$$\int \frac{a(x)}{c(x)}\,\mathrm{d}x + \int \frac{d(y)}{b(y)}\,\mathrm{d}y = 0,$$

и, кроме того, нужно учесть решения уравнения

$$b(y)c(x) = 0,$$

которые могли быть «потеряны» при решении (1).

2.2. Однородные дифференциальные уравнения

$$a(x,y)\mathrm{d}x + b(x,y)\mathrm{d}y = 0, \tag{2}$$

где функции $a(x,y)$, $b(x,y)$ — однородные, степени k, то есть удовлетворяют тождеству

$$f(\lambda x, \lambda y) = \lambda^k f(x,y).$$

Подстановка $y = tx$ сводит однородное уравнение к уравнению с разделяющимися переменными.

2.3. Линейные дифференциальные уравнения первого порядка

$$y' + p(x)y = q(x) \tag{3}$$

имеют решение

$$y = e^{-\int p(x)\,\mathrm{d}x}\left(\int q(x)e^{\int p(x)\,\mathrm{d}x}\mathrm{d}x + C\right)$$

2.4. Линейные однородные уравнения с постоянными коэффициентами n-го порядка

Здесь под однородностью уравнения понимается отсутствие свободного члена — слагаемого, не зависящего от неизвестной функции:

$$y^{(n)} + a_1 y^{(n-1)} + \ldots a_{n-1}y' + a_n y = 0, \tag{4}$$

где $a_1, \ldots, a_n$ — вещественные константы. Решение уравнения (4) связано с решениями *характеристического уравнения*

$$\lambda^n + a_1\lambda^{n-1} + \ldots + a_{n-1}\lambda + a_n = 0,$$

имеющего вследствие основной теоремы алгебры ровно n корней, среди которых могут быть и комплексные, которые в этом случае входят в множество решений парами комплексно сопряженных чисел[1]. Общее решение (4) является суммой, слагаемые которой определяются типом корней характеристического уравнения, а именно:

- каждому действительному простому корню λ в общем решении соответствует слагаемое $Ce^{\lambda x}$, где C — константа;
- каждому вещественному корню λ кратности k в общем решении соответствует слагаемое $(C_1 + C_2 x + \ldots + C_k x^{k-1})e^{\lambda x}$, C_i — константы;
- каждой паре комплексных сопряженных простых корней

$$\alpha \pm \beta i$$

[1] Так как коэффициенты характеристического уравнения действительны.

в общем решении соответствует слагаемое вида

$$e^{\alpha x}(C_1 \cos \beta x + C_2 \sin \beta x);$$

- каждой паре комплексно сопряженных корней кратности k соответствует слагаемое

$$e^{\alpha x}\Big(\big(A_1 + A_2 x + \ldots + A_{k-1}x^{k-1}\big)\cos \beta x + \\ + \big(B_1 + B_2 x + \ldots + B_{k-1}x^{k-1}\big)\sin \beta x\Big), \quad (5)$$

где A_i, B_i — константы.

2.5. Неоднородные линейные уравнения

$$y^{(n)} + a_1(x)y^{(n-1)} + \ldots a_{n-1}(x)y' + a_n(x)y = f(x). \quad (6)$$

Общее решение неоднородного линейного уравнения равно сумме *любого* его частного решения и общего решения соответствующего однородного уравнения. Частные решения в некоторых случаях можно найти подбором, а в общем случае — *методом вариации постоянных* Лагранжа: в общем решении однородного уравнения, соответствующего (6)

$$C_1 y_1 + \ldots + C_n y_n, \quad (7)$$

константы интегрирования $C_1, \ldots, C_n$ считают неизвестными функциями от x, которые определяют подставив (7) в уравнение (6). Например, для уравнения с постоянными коэффициентами

$$y'' + a_1 y' + a_2 y = f(x)$$

функции $C_1(x), C_2(x)$ подбираются так, чтобы они удовлетворяли соотношениям

$$\begin{aligned} C_1'(x)y_1 + C_2'(x)y_2 &= 0, \\ C_1'(x)y_1' + C_2'(x)y_2' &= f(x). \end{aligned}$$

2.6. Устойчивость решений

Если не известно аналитическое решение дифференциального уравнения, численные решения получают последовательно задавая значения входных параметров — начальных или краевых условий, констант и т. д. Однако часто бывают нужны данные о поведении модели при *всех* входных параметрах. В частности, не редко возникает потребность в информации об *устойчивости* решений в окрестности некоторой точки или области фазового пространства. Под устойчивостью понимается свойство решений *не сильно* отклоняться друг от друга при *малом* изменении входных параметров. Например, по отношению к механическим системам это означает, что *малые* воздействия на систему не выведут её из равновесия.

Придание точного смысла понятиям, обусловливающим устойчивость, — «не сильно», «мало» и т. п. — приводят к различным её определениям: устойчивость по Ляпунову, асимптотическая устойчивость, эквиасимптотическая устойчивость и т.д. [84], [85].

Пусть для системы дифференциальных уравнений

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) \qquad \mathbf{x}(t_0) = \mathbf{x_0} \tag{8}$$

справедливы условия существования и единственности на множестве $(t, \mathbf{x})$, где $\alpha < t < +\infty$, $\mathbf{x} \in C$, C — открытое множество в пространстве переменного $\mathbf{x}$.

Определение 2.1. *Решение* $\mathbf{x} = \varphi(t)$ *(*$t \geqslant t_0$*) системы* (8) *называют устойчивым по Ляпунову, если для любого* $\varepsilon > 0$ *найдется такое* $\delta > 0$*, что для любого вектора* $\mathbf{x_0}$ *с условием*

$$|\varphi(t_0) - \mathbf{x_0}| < \delta,$$

решение $\mathbf{x} = \psi(t)$ *с начальным условием* $\psi(t_0) = \mathbf{x_0}$ *определено при всех* $t \geqslant t_0$ *и выполняется*

$$|\varphi(t) - \psi(t)| < \varepsilon, \qquad t \geqslant t_0.$$

Если, сверх того,

$$\lim_{t \to +\infty} |\varphi(t) - \psi(t)| = 0,$$

то решение $\mathbf{x} = \varphi(t)$ *называют асимптотически устойчивым.*

Легко видеть, что исследование устойчивости решения $\mathbf{x} = \varphi(t)$ системы (8) равносильно проверке устойчивости решения, тождественно равного нулю. Действительно, положим

$$\mathbf{y} = \mathbf{x} - \varphi(t),$$

тогда устойчивость $\mathbf{x} = \varphi(t)$ эквивалентна устойчивости решения $\mathbf{y} = \mathbf{0}$.

Устойчивость решений линейной системы с постоянными коэффициентами

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \tag{9}$$

где $\mathbf{A}$ — постоянная действительная матрица.

Для того, чтобы положение равновесия $\mathbf{y} = \mathbf{0}$ системы (9) было

— *асимптотически устойчиво,* необходимо и достаточно, чтобы все собственные значения матрицы $\mathbf{A}$ имели *отрицательные действительные части;*

— *устойчиво по Ляпунову,* необходимо (но не достаточно), чтобы все собственные значения матрицы $\mathbf{A}$ имели *неположительные действительные части;*

Устойчивость решений системы в первом приближении

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}). \tag{10}$$

Пусть правая часть (10) удовлетворяет условиям существования и единственности решения и имеет вид

$$\mathbf{f}(t, \mathbf{x}) = \mathbf{A}(t)\mathbf{x} + \mathbf{F}(t, \mathbf{x})$$

с условием

$$\lim_{|\mathbf{x}|\to 0}\frac{|\mathbf{F}(t,\mathbf{x})|}{|\mathbf{x}|}=0.$$

Определение 2.2. *Линейную однородную систему*

$$\dot{\mathbf{x}}=\mathbf{A}(t)\mathbf{x},\qquad \mathbf{A}(t)=\left[\frac{\partial f_i}{\partial x_j}(t,0)\right]$$

называют первым приближением (линеаризацией) системы (10).

В частности, для случая, когда матрица $\mathbf{A}$ постоянна, вопрос об устойчивости решения системы

$$\dot{\mathbf{x}}=\mathbf{A}\mathbf{x}+\mathbf{F}(t,\mathbf{x}),\qquad \mathbf{F}(t,\mathbf{0})=\mathbf{0} \tag{11}$$

определяется теоремами Ляпунова:

- Если все собственные значения матрицы $\mathbf{A}$ имеют *отрицательные действительные части* и
$$|\mathbf{F}(t,\mathbf{x})|\leqslant M|\mathbf{x}|^{1+\alpha}$$
при $t\geqslant t_0$, достаточно малом $|\mathbf{x}|$ и некоторых положительных константах α, M, то положение равновесия $\mathbf{x}=\mathbf{0}$ системы (11) *асимптотически устойчиво.*
- Если среди собственных значений матрицы $\mathbf{A}$ есть *хотя бы одно с положительной действительной частью* и
$$|\mathbf{F}(t,\mathbf{x})|\leqslant M|\mathbf{x}|^{1+\alpha}$$
при $t\geqslant t_0$, достаточно малом $|\mathbf{x}|$ и некоторых положительных константах α, M, то положение равновесия $\mathbf{x}=\mathbf{0}$ системы (11) *неустойчиво.*

Определение 2.3. *Говорят, что система находится на границе устойчивости, если хотя бы одно собственное значение матрицы* $\mathbf{A}$ *имеет нулевую действительную часть, а действительные части всех остальных собственных значений отрицательны.*

При исследовании устойчивости по первому приближению необходимо определить знаки действительных частей собственных значений матрицы, т. е. корни характеристического многочлена. Эта задача может быть решена применением *критерия Рауса — Гурвица* [85]. В частности, для характеристических многочленов второй и третьей степеней

$$\lambda^2+a_1\lambda+a_2,$$
$$\lambda^3+b_1\lambda^2+b_2\lambda+b_3$$

этот критерий формулируется так: все корни имеют отрицательные действительные части, тогда и только тогда, когда, соответственно, выполняются соотношения

$$a_1>0,\quad a_2>0,$$
$$b_1>0,\quad b_1b_2-b_3>0,\quad b_3>0.$$

Если в последнем условии вместо $b_3 > 0$ будет $b_3 = 0$, то система находится на границе устойчивости.

Необходимым (но не достаточным, для степеней многочлена больших 2) условием устойчивости системы является положительность всех коэффициентов характеристического многочлена. Таким образом, если у характеристического полинома есть хотя бы один отрицательный коэффициент, то решение неустойчиво.

2.7. Численное решение дифференциальных уравнений

Аналитические решения дифференциальных уравнений удается получить только для относительно простых случаев. Поэтому с появлением быстродействующих коппьютеров численные методы решения стали одним из основных способов решения конкретных практических задач для обыкновенных дифференциальных уравнений.

Рассмотрим численные методы решения *задачи Коши*

$$y' = f(x, y), \quad y(x_0) = y_0. \tag{12}$$

Далее предполагается, что численные методы применяются к задаче Коши, имеющей единственное решение и, сверх того, это решение должно быть устойчивым по начальным данным, т. е. *малые изменения начальных условий приводят к достаточно малому изменению интегральных кривых* [24].

Считая, что решение задачи (12) существует и единственно, требуется на некотором отрезке $[x_0, x_n]$ найти её приближенное решение с заданной точностью ε (погрешностью):

$$\widehat{y}_i \approx y(x_i), \qquad x_i = x_0 + ih, \quad i = 0, 1, \ldots, n,$$

где константа h называется *шагом* интегрирования, последовательность $\Omega = \{x_0, x_1, \ldots, x_n\}$ называют равномерной (регулярной) *сеткой*.

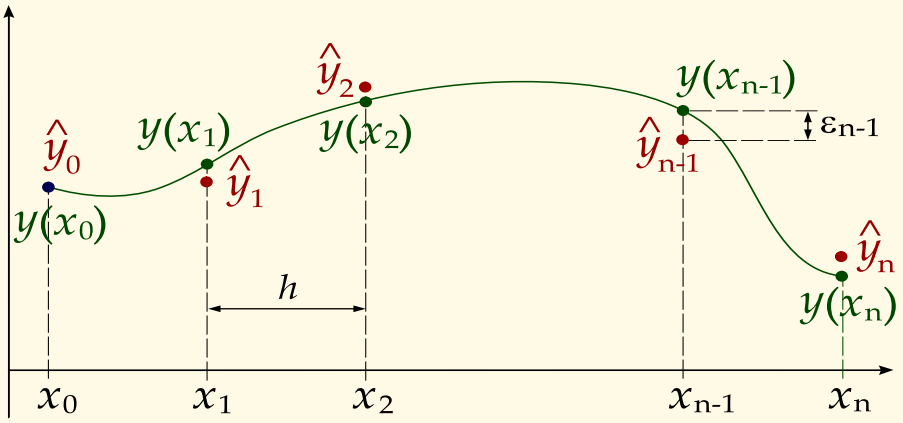


Рис. 1. *Регулярная сетка*

Локальной ошибкой численного метода на i-м шаге называется величина $\varepsilon_i = |\widehat{y}_i - y(x_i)|$. Погрешность определяется формулой

$$\varepsilon = \varepsilon(h) = \max_{i=0,1,\ldots,n} \varepsilon_i$$

Определение 2.4. *Число p называют* порядком *(*точностью*) метода, если*

$$\varepsilon = O(h^p).$$

Существует два типа схем решения дифференциальных уравнений:

Явные, когда для определения $\widehat{y}_i$ могут использоваться лишь значения вычислений на предыдущих шагах, т. е.

$$\widehat{y}_i = \Phi(x_0, x_1, \ldots, x_{i-1}, \widehat{y}_0, \widehat{y}_1, \ldots, \widehat{y}_{i-1});$$

Неявные, когда искомая величина $\widehat{y}_i$ входит одновременно и в левую, и в правую часть итерационного соотношения

$$\widehat{y}_i = \Phi(x_0, x_1, \ldots, x_i, \widehat{y}_0, \widehat{y}_1, \ldots, \widehat{y}_{i-1}, \widehat{y}_i);$$

Явные схемы проще, но зачастую неявные схемы предпочтительнее.

Метод Эйлера (метод ломаных)

Если заменить в (12) производную её разностным приближением

$$y' \approx \frac{y(x+h) - y(x)}{h} = \frac{y_{i+1} - y_i}{h},$$

то получим приближенное равенство[2]

$$y_{i+1} = y_i + hf(x_i, y_i), \qquad i = 0, 1, 2, \ldots, n. \tag{13}$$

Данная процедура называется *методом Эйлера*, она имеет первый порядок точности по h, если $f(x, y)$ ограничена вместе со своими первыми производными по обоим аргументам [30], [31],

$$\varepsilon = \max_{i=0,1,\ldots,n} |y_i - y(x_i)| = O(h).$$

Метод Эйлера очень прост в программной реализации, но вследствие своих недостатков:

- малой точности;
- накопления ошибок — погрешность на каждом новом шаге систематически возрастает,

для практического нахождения решений задачи Коши в настоящее время применяется редко.

Геометрический смысл схемы (13) заключается в приближении решения на отрезке $[x_i, x_{i+1}]$ отрезком касательной в точке x_i: если соединить точки (x_i, y_i) прямолинейными отрезками, получим *ломаную Эйлера*, каждое звено которой с началом в точке (x_i, y_i) имеет угловой коэффициент, равный $f(x_i, y_i)$. Фактически, на каждом шаге метода из-за погрешности приближенное решение переходит с одной интегральной кривой на другую и поэтому ломаная, как правило, сильно отклоняется от точного решения.

[2] Здесь и далее *приближенные* значения $\widehat{y}_i$ будем писать просто: y_i.

Точность метода Эйлера можно повысить, улучшив аппроксимацию функции $y(x)$, если при её разложении в ряд Тейлора учесть дополнительно слагаемое, содержащее h^2.

Подставляя аппроксимацию второй производой ее конечной разностью

$$y'' \approx \frac{y'(x_{i+1}) - y'(x_i)}{h}$$

в формулу Тейлора и отбрасывая члены ряда, начиная со слагаемого, содержащего h^3, запишем

$$y(x_{i+1}) = y(x_i) + \frac{h}{2}\big(y'(x_i) + y'(x_{i+1})\big),$$

откуда, после замены производных правыми частями по формуле (12), получим расчетную формулу *метода Эйлера с уточнением*

$$y_{i+1} = y_i + \frac{h}{2}\big(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\big), \qquad i = 0, 1, 2, \ldots, n. \tag{14}$$

Схема (14) — неявная, поскольку y_{i+1} присутствует одновременно в левой и правой её частях.

Поскольку неизвестная величина y_{i+1} в схеме (14) входит как аргумент в функцию f, для вычисления по неявной схеме, вообще говоря, придется разрешить уравнение (14) относительно y_{i+1}, что составляет дополнительную проблему.

Схему (14) можно превратить в явную, если дополнить ее, подставляя в правую часть (14) значение y_{i+1}, предварительно рассчитав его по формуле (13):

$$y_{i+1}^{(k)} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f\left(x_{i+1}, y_{i+1}^{(k-1)}\right)\right), \tag{15}$$

$$y_{i+1}^{(0)} = y_i + hf(x_i, y_i). \tag{16}$$

При этом предварительно вычисленное значение y_{i+1} называют *прогнозом*, а уточнение результата по формуле (14) — его *коррекцией*. Семейство численных алгоритмов, использующих методы прогноза и коррекции, называют схемами *предиктор-корректор* (англ. predictor — предсказатель, corrector — исправляющий).

Модифицированный метод Эйлера обеспечивает второй порядок точности. Повышение точности алгоритма достигается, однако, за счет дополнительных итераций при расчетах на каждом шаге.

Метод Рунге — Кутты

Наиболее широко применяется метод Рунге — Кутты четвертого порядка — так называемый *стандартный* или *классический* метод Рунге — Кутты:

$$y_{i+1} = y_i + \frac{1}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big), \tag{17}$$

$$
\begin{aligned}
k_1 &= hf(x_i, y_i),\\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right),\\
k_3 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right),\\
k_4 &= hf\big(x_i + h, y_i + k_3\big),
\end{aligned}
\tag{18}
$$

Этот метод имеет четвёртый порядок точности, то есть суммарная ошибка на всем интервале интегрирования имеет порядок $O(h^4)$ (ошибка на каждом шаге — $O(h^5)$).

Схемы Рунге — Кутты, выше четвертого порядка точности, как правило, оказываются громоздкими и неудобными для практического применения и, потому, используются редко.

Схемы Рунге-Кутта естественным образом обобщаются на случай систем уравнений первого порядка при помощи формальной замены функций $y(x)$ и $f(x;y)$ на вектор-функции $\mathbf{y}(x)$ и $\mathbf{f}(x;\mathbf{y})$

$$\mathbf{y}' = \mathbf{f}(x;\mathbf{y}), \qquad \mathbf{y}(0) = \mathbf{y_0},$$

а поскольку, как известно, любое дифференциальное уравнение n-го порядка эквивалентно системе из n уравнений первого порядка, то методы Рунге — Кутты можно применять для уравнений любого порядка.

Например, для системы двух дифференциальных уравнений, обозначив $\mathbf{y} = (y(x), z(x))^{\mathrm{T}}$, $\mathbf{f} = (f(x), g(x))^{\mathrm{T}}$ и, соответственно, $\mathbf{y_i} = (y_i, z_i)^{\mathrm{T}}$, $\mathbf{k_m} = (k_m, q_m)^{\mathrm{T}}$, метод Рунге — Кутты 4-го порядка принимает вид

$$
\mathbf{y_{i+1}} = \begin{pmatrix} y_{i+1} \\ z_{i+1} \end{pmatrix} = \begin{pmatrix} y_i + \big(k_1 + 2k_2 + 2k_3 + k_4\big)/6 \\ z_i + \big(q_1 + 2q_2 + 2q_3 + q_4\big)/6 \end{pmatrix},
$$

$$
\begin{aligned}
k_1 &= hf(x_i, y_i), \quad q_1 = hg(x_i, y_i),\\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{q_1}{2}\right),\\
q_2 &= hg\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{q_1}{2}\right),\\
k_3 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, z_i + \frac{q_2}{2}\right),\\
q_3 &= hg\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, z_i + \frac{q_2}{2}\right),\\
k_4 &= hf\big(x_i + h, y_i + k_3, z_i + q_3\big),\\
q_4 &= hg\big(x_i + h, y_i + k_3, z_i + q_3\big).
\end{aligned}
$$

Таким же образом расчетные формулы метода Рунге — Кутты обобщаются на случай системы из трех и более уравнений.

Правило Рунге

При численном решении задач Коши (12) обычно требуется найти их приближенное решение с некоторой заданной точностью ε, которая определяется условиями решаемой прикладной задачи. Для формул Рунге — Кутты такая погрешность пропорциональна величине выбираемого шага h [31]. Способ выбора такой величины шага интегрирования, чтобы обеспечивалась требуемая точность численного решения, дает правило Рунге.

Основная проблема состоит в том, что при численном решении обычно точное решение задачи не известно.

Функцию ошибки $\varphi_r(h)$ по формуле Тейлора можно представить в виде

$$\varphi_r(h) = \frac{\varphi_r^{(s+1)}(0)}{(s+1)!}\,h^{s+1} + \frac{\varphi_r^{(s+2)}(h)(\xi h)}{(s+2)!}\,h^{s+2}, \qquad 0 < \xi < 1.$$

Следовательно, если обозначить $y(x)$ — точное решение задачи (12) на отрезке $[x, x+h]$, а $\tilde{y}_{(h)}$ — приближенное решение, получаемое по соответствующей формуле Рунге — Кутты с шагом h, то

$$\tilde{y}_{(h)}(x+h) - y(x+h) \approx \frac{\varphi_r^{(s+1)}(0)}{(s+1)!}\,h^{s+1}$$

и, в силу малости h, погрешность на следующем шаге интегрирования будет иметь тот же главный член, то есть *после двух шагов* получим

$$\tilde{y}_{(h)}(x+2h) - y(x+2h) \approx 2\,\frac{\varphi_r^{(s+1)}(0)}{(s+1)!}\,h^{s+1}. \tag{19}$$

С другой стороны, если применить все ту же формулу Рунге — Кутты, но с удвоенным шагом $2h$, то

$$\tilde{y}_{(2h)}(x+2h) - y(x+2h) \approx \frac{\varphi_r^{(s+1)}(0)}{(s+1)!}\,(2h)^{s+1}. \tag{20}$$

Из соотношений (19) и (20) следует

$$\tilde{y}_{(h)}(x+2h) - y(x+2h) \approx \frac{\tilde{y}_{(2h)}(x+2h) - \tilde{y}_{(h)}(x+2h)}{2^s - 1}.$$

Таким образом, если выполняется

$$\frac{\left|\tilde{y}_{(2h)} - \tilde{y}_{(h)}\right|}{2^s - 1} \leqslant \varepsilon,$$

то и точное решение y уклоняется от приближенного $\tilde{y}_{(h)}$ не более, чем на величину ε.

Практически из последнего неравенства следует, что при отсутствии возможности сравнения приближенных и точных решений, при вычислениях для достижения заданной точности ε можно использовать *Метод двойного пересчета* (*правило Рунге*): по формулам Эйлера (13)

или Рунге — Кутты (17) на каждой итерации находится при заданном изначально шаге h приближенное решение $\tilde{y}_{(h)}$ и значение $\tilde{y}_{(h/2)}$, определяемое по тем же формулам при половинном шаге; если не выполняется неравенство

$$\frac{\left|\tilde{y}_{(h)} - \tilde{y}_{(h/2)}\right|}{2^s - 1} \leqslant \varepsilon, \tag{21}$$

где s — порядок точности метода ($s = 1$ для схемы Эйлера и $s = 4$ для метода Рунге — Кутты четвертого порядка), то шаг снова следует уменьшить вдвое и проверить истинность оценки (21) для полученных приближений.

Однако, следует помнить, что при измельчении шага начиная с некоторого значения h, точность вычислений начинает падать вследствии накапливания ошибок округления. Поэтому в общем случае, как правило, приходится применять *адаптивное изменение шага:* «простые» части траектории (небольшая и плавно меняющаяся правая часть) следует проходить с достаточно большим шагом (минимизируя ошибки округления), а на «сложных» участках (большая и резко меняющаяся правая часть) шаг нужно уменьшать для достижения нужной локальной точности.

2.8. Исследование систем дифференциальных уравнений

В практических приложениях дифференциальные уравнения как правило описывают некоторую математическую модель. После построения математической модели проводится ее аналитическое или численное исследование, позволяющее «проиграть» поведение моделируемого объекта при различных входных параметрах.

Математическая модель динамической системы включает задание параметров и эволюционный оператор, позволяющий указать значения этих параметров во времени: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ — набор чисел, описывающий состояние системы в момент времени t, эволюционный оператор определяется через скорость изменения (т. е. производную по времени) каждого параметра системы

$$\frac{dx_i}{dt} = F_i(x_1, x_2, \ldots, x_n), \quad i = 1, \ldots, n. \tag{22}$$

Точка $\mathbf{x}$ множества $\mathbb{R}^n$ принадлежит т. н. *фазовому пространству* системы (22). Дополнив (22) начальными условиями $\mathbf{x} = \mathbf{x_0}$ получим *задачу Коши*, решение которой образует *фазовую траекторию*. Множества фазовых траекторий, отвечающих различным начальным условиям, образуют *фазовый портрет* системы.

Если не известно аналитическое решение дифференциального уравнения, численные решения получают последовательно задавая значения входных параметров — начальных или краевых условий, констант и т. д. — *вычислительный эксперимент*. Если дифференциальные уравнения (22) нелинейны, то *аналитическое* решение этой системы, как правило, вызывает большие трудности и, чаще всего, невозможно.

К тому же, часто бывают нужны данные о поведении модели при *всех* входных параметрах, а не при их ограниченном иножестве.

Фазовые траектории дают наглядное представление о поведении динамической системы. Их вид часто можно определить не прибегая к решению, на основе качественного анализа системы.

Рассмотрим пример. *Математический маятник* представляет собой материальную точку, которая под действием силы тяжести движется по дуге окружности радиуса l, лежащей в вертикальной плоскости. Материальная точка массы m в начальный момент времени $t = 0$ получает угловую скорость v_0 и находится под действием только лишь силы тяжести (пренебрегаем трением и прочими факторами), направленной вертикально вниз. Таким образом, в соответствии со II-м законом Ньютона, уравнение движения маятника задаётся соотношением

$$ml\ddot{\varphi} = -mg\sin\varphi,$$

где φ — угол отклонения материальной точки от вертикали, g — ускорение свободного падения, откуда, учитывая начальные условия, получим

$$l\ddot{\varphi} = -g\sin\varphi, \qquad \varphi(0) = 0, \quad \dot{\varphi}(0) = v_0. \tag{23}$$

Если, для того чтобы привести уравнение (23) в соответствие с (22), положим $x_1 = \varphi$, $x_2 = \dot{\varphi}$, то получим задачу Коши для нелинейной системы двух дифференциальных уравнений

$$\dot{x}_1 = x_2, \tag{24}$$

$$\dot{x}_2 = -\frac{g}{l}\sin x_1 \tag{25}$$

$$x_1(0) = 0, \tag{26}$$

$$x_2(0) = v_0 \tag{27}$$

Динамические системы, в которых полная энергия сохраняется (как в рассмотренном примере) называются *консервативными*.

Полученная фазовая кривая (использовано численное решение) имеет следующий вид

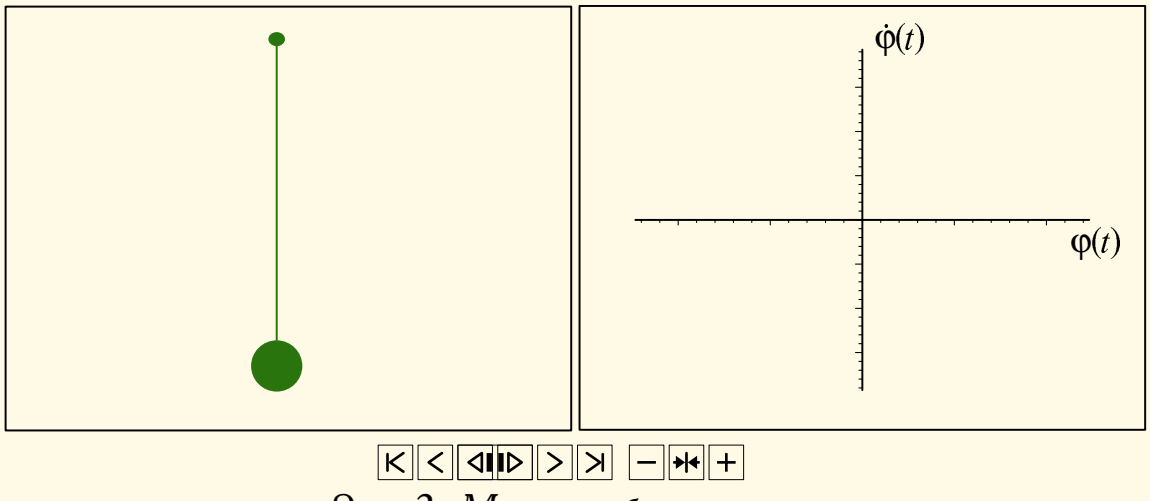


Рис. 2. *Маятник без трения*

Замкнутые фазовые кривые отвечают тем начальным значениям, при которых происходят колебания маятника, тем более близкие к *гармоническим*, чем ближе форма этих кривых к окружностям. Система совершает гармонические колебания при малых углах φ, тогда в уравнении (23) можно принять $\sin\varphi \approx \varphi$.

В реальных физических системах обычно происходят потери энергии (например, за счёт трения). Чтобы учесть трение в нашей модели, считая, что сила трения пропорциональна скорости $\dot\varphi$ движения маятника, преобразуем уравнение (23) к виду

$$l\ddot\varphi = -g\sin\varphi - a\dot\varphi, \qquad \varphi(0) = 0, \quad \dot\varphi(0) = v_0, \tag{28}$$

где a — константа. Фазовая траектория системы теперь будет такой

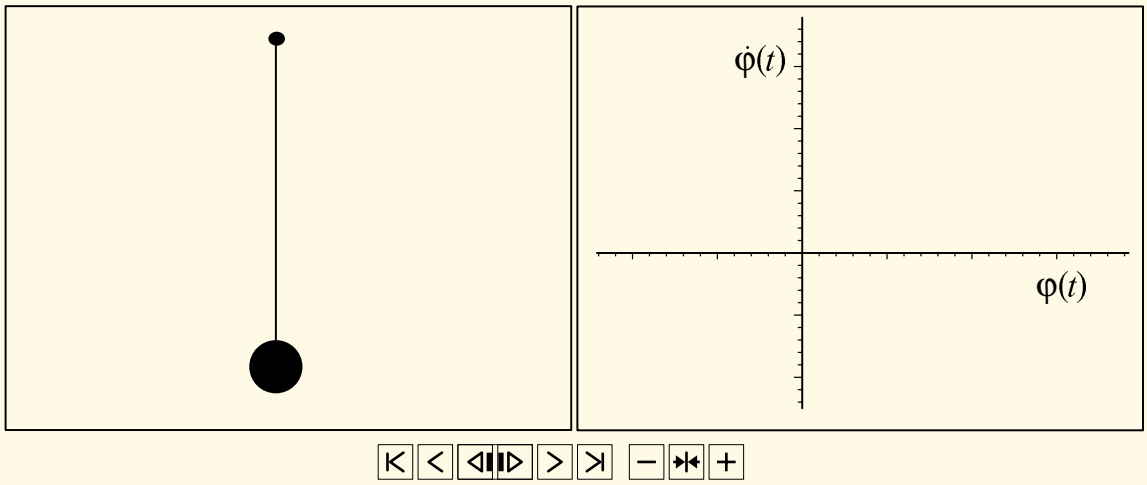


Рис. 3. *Маятник с трением*

В консервативных системах не существует *притягивающих множеств* (*аттракторов*, от англ. attract — притягивать, привлекать), т. е. таких подмножеств фазового пространства, к которым с течением времени стремятся траектории, начинающиеся в некоторой их окрестности. В диссипативных же системах они могут существовать. В примере с затухающими колебаниями аттрактор состоит из единственной точки — начала координат. В более общем случае потери энергии динамической системой из-за трения или, например, вязкости всегда приводят к тому, что орбиты притягиваются к некоторому подмножеству фазового пространства. Грубо говоря, аттрактор отвечает *установившемуся* поведению системы — тому, к которому она «стремится».

В частности, известны т. н. *странные аттрактор* — аттракторы, траектория которых не замыкается и режим функционирования неустойчив (малые отклонения от режима нарастают). Динамика странных аттракторов часто бывает хаотической: предсказание траектории, попавшей в аттрактор, затруднено, поскольку малая погрешность в начальных параметрах через некоторое время может привести к сильному расхождению прогноза с реальной траекторией. Непредсказуемость траектории в *детерминированных* динамических системах называют *динамическим хаосом* (эффектом бабочки) [17].

3. Решения разностных уравнений

Обычно математические модели описывают разностными уравнениями, если рассматриваемые в них величины регистрируют через некоторые (чаще всего одинаковые) промежутки времени. К разностным уравнениям приводят многие экологические задачи, модели популяционной динамики, экономические задачи (расчет сложных процентов, управление банковскими депозитами и т. п.), демографические модели (прогнозирование половозрастной структуры населения, построение демографических пирамид) [15].

3.1. Линейные уравнения с постоянными коэффициентами

Линейные разностные уравнения с постоянными коэффициентами порядка n имеют вид

$$y_{k+n} + a_1 y_{k+n-1} + \ldots + a_n y_k = f_k, \tag{29}$$

где $a_1, \ldots, a_n$ — заданные вещественные константы, $a_n \neq 0$, f_k — заданная числовая последовательность[1], y_k — неизвестная последовательность, подлежащая определению.

Уравнение

$$y_{k+n} + a_1 y_{k+n-1} + \ldots + a_n y_k = 0 \tag{30}$$

называют *однородным*. Его тривиальное решение: $y_k \equiv 0$. Нетривиальные решения (30) ищутся в виде $y_k = \lambda^k$, где λ определяется из *характеристического уравнения*

$$\lambda^n + a_1 \lambda^{n-1} + \ldots + a_n = 0. \tag{31}$$

Общее решение однородного уравнения (30) зависит от корней алгебраического уравнения (31):

- Если все корни $\lambda_1, \ldots, \lambda_n$ уравнения (31) вещественны и различны, то решение (30)
$$y_k = C_1 \lambda_1^k + C_2 \lambda_2^k + \ldots + C_n \lambda_n^k,$$
где $C_1, C_2, \ldots, C_n$ — произвольные константы, которые можно определить, если для уравнения заданы начальные условия.
- Если уравнение (31) имеет корни $\lambda_1, \ldots, \lambda_s$, $s \leqslant n$ кратностей $m_1, m_2, \ldots, m_s$ ($m_1 + m_2 + \ldots + m_s = n$), то общее решение (30)
$$y_k = \sum_{j=1}^{s} \left(C_{0,j} + C_{1,j} k + \ldots + C_{m_j - 1, j} k^{m_j - 1} \right) \lambda_j^k,$$
где $C_{i,j}$ — произвольные постоянные.
- Если уравнение (31) имеет комплексные корни, то каждый комплексный корень кратности m
$$\lambda = |\lambda|(\cos\varphi + i \sin\varphi), \qquad 0 \leqslant \varphi < 2\pi$$

[1] Последовательность — функция натурального аргумента.

и ему комплексно сопряженный корень

$$\overline{\lambda} = |\lambda|(\cos\varphi - i\sin\varphi), \qquad 0 \leqslant \varphi < 2\pi$$

соответствуют в общем решении уравнения (30) слагаемому в виде линейной комбинации функций

$$|\lambda|^k \cos k\varphi, k|\lambda|^k \cos k\varphi, \ldots, k^{m-1}|\lambda|^k \cos k\varphi;$$
$$|\lambda|^k \sin k\varphi, k|\lambda|^k \sin k\varphi, \ldots, k^{m-1}|\lambda|^k \sin k\varphi.$$

Решение уравнения (29) есть сумма общего решения соответствующего однородного уравнения (30) и любого частного решения (29). Последнее можно найти методом вариации постоянных[2] или, в простых случаях, исходя из вида функции f_k, подбором [13], [14].

3.2. Линейные уравнения первого порядка

Линейным разностным уравнением первого порядка называют

$$y_{k+1} + a_k y_k = f_k, \tag{32}$$

где a_k, f_k — заданные числовые последовательности, $a_k \not\equiv 0$, а последовательность y_k — искомая.

Его общее решение дается формулой

$$y_k = \left(C + \sum_{j=0}^{k-1} \frac{f_i}{A_{j+1}} \right) A_k, \qquad A_m = (-1)^m \prod_{j=0}^{m-1} a_j, \tag{33}$$

где C — произвольная константа, значение которой можно определить из начального условия, вида $y_0 = \alpha$.

3.3. Нелинейные разностные уравнения

Для нелинейных разностных уравнений, так же как для дифференциальных, алгебраических, интегральных уравнений, не существует *общих* методов решения. Аналитические решения нелинейных разностных уравнений получены только для отдельных, специальных случаев. Так иногда удается заменой переменных свести нелинейное разностное уравнение к одному или нескольким линейным [15, стр. 87–92].

В отличие от дифференциальных уравнений решения нелинейных разностных уравнений, даже довольно простого вида, одномерных, часто демонстрируют чрезвычайно сложное, хаотическое поведение. Например

$$y_{k+1} = \mu\left(1 - 2\left|\frac{1}{2} - y_k\right|\right), \qquad 0 \leqslant y_k \leqslant 1,$$

при $\mu \geqslant 1/2$, $k \to \infty$ [13]. У непрерывных моделей для хаоса обязательно наличие не менее трёх измерений.

---

[2] По аналогии с решением неоднородных линейных дифференциальных уравнений, см. стр. 157.

4. Решения матричных игр

Теория игр описывает и изучает математические модели *конфликтных* ситуаций [33], [34], [35], [36]. Как и в любой модели, в теории игр приняты различные упрощения по сравнению с реальными процессами, которые она описывает. В теории игр предполагается, что конфликт всегда протекает по чётко сформулированным и однозначно заданным правилам — упрощение, в реальности, как хорошо известно, далеко не каждый конфликт происходит по правилам.

Совокупность таких правил, а также сама процедура их применения, называется *игрой*. Участники игры называются *игроками*. В зависимости от моделируемого процесса игроками могут быть отдельные люди, военные подразделения, государства, коммерческие фирмы, природа и т. д. Другим существенным упрощением в теории игр является требование к каждому игроку иметь полный план всех своих действий *до* начала игры. Варианты всех возможных действий игрока называются его *стратегиями*[1].

Решение игры состоит в определении *оптимальных*[2] (в каком-то смысле, наиболее предпочтительных для данной игры) стратегий игроков. Очевидно, чтобы выбирать наилучшие стратегии, нужно иметь возможность их сравнивать. Поэтому будем предполагать, что правила игры дают возможность дать численную оценку каждой стратегии — *выигрыш*[3] $H(s)$ игрока при использовании данной стратегии s в данной ситуации. Таким образом:

> **Основная задача теории игр:** *как должен вести себя (какую стратегию выбрать) разумный игрок в конфликте (в игре) с разумным противником (противниками), чтобы обеспечить себе в среднем наибольший возможный выигрыш.*

Процесс игры состоит из одного или нескольких шагов (ходов), на каждом из которых игроки делают выбор, или лично выбирая стратегию, или делая это случайным образом. В результате в процессе игры на каждом ходу складывается некоторая система стратегий, называемая *ситуацией*. Множество всех ситуаций S является декартовым произведением множеств всех стратегий игроков.

Далее рассматриваются только бескоалиционные *антагонистические*[4] игры, т. е. игры двух игроков, значения функции выигрыша кото-

---

[1] От греч. στρατηγια — искусство ведения войны, общий план боевых действий.

[2] Лат. optimum — наилучшее.

[3] Это число может, в зависимости от моделируемого процесса, выражать количество выигранных денег, временной промежуток, «степень морального удовлетворения» и т. п.

[4] От др.-греч. ανταγωνισης — «противник».

рых равны по абсолютной величине и противоположны по знаку:

$$H_1(\mathbf{s}) = -H_2(\mathbf{s}), \qquad \forall \mathbf{s} \in S. \tag{34}$$

Следовательно, для антагонистических игр $H_1(\mathbf{s}) + H_2(\mathbf{s}) = 0$, $\forall \mathbf{s} \in S$, т. е. они являются играми с нулевой суммой: всё, что выигрывает один игрок, проигрывает его противник.

Определение 4.1. *Антагонистические игры, в которых каждый игрок имеет конечное множество стратегий называют матричными.*

Такую игру можно представить, задавая матрицу, в которой строки соответствуют стратегиям 1-го игрока, а столбцы — стратегиям 2-го.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{35}$$

Элементы матрицы a_{ij} равны выигрышу[5] 1-го игрока при выборе им i-ой стратегии, а 2-м — своей j-ой стратегии. Матрицу $\mathbf{A}$ называют *матрицей игры* (*платежной матрицей*, *матрицей выигрышей*).

Процесс «матричной игры» можно представить так:

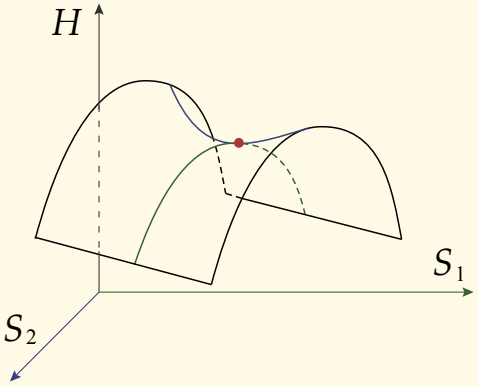
1. В $m \times n$ матрице (39), игрок 1 выбирает какую-либо её строку, а игрок 2 — столбец, причём выборы игроки делают независимо[6].
2. Игрок 1 получает выигрыш, стоящий на пересечении выбранных строки и столбца, игрок 2 — этот же выигрыш со знаком минус[7].

Определение 4.2. *Ситуация s называется приемлемой для игрока i, если для любой его стратегии s_i' выполняется*

$$H_i(s||s_i') \leqslant H_i(s),$$

где $s||s_i' = (s_1, \dots, s_{i-1}, s_i', s_{i+1}, \dots, s_n)$, т. е. стратегия s_i заменена s_i'.

Другими словами, ситуация для игрока приемлема, если он, изменяя свою стратегию, не сможет гарантированно получить больший выигрыш — «от добра добра не ищут». Ситуация, приемлемая *для всех игроков*, называется *равновесной*. Ситуацию равновесия называют также *седловой точкой* (см. рисунок). Очевидно, что при наличии седловой точки в игре, игрокам *разумно* придерживаться именно этих равновесных стратегий — теория игр всегда даёт наиболее безопасные, «перестраховочные» решения.



[5] Если $a_{ij} < 0$, то фактически речь идет о проигрыше.

[6] Т. е. игроки в момент выбора не знают о выборе противника.

[7] Т. о, платежи матрицы (39) записаны «с точки зрения первого игрока».

4.1. Седловые точки

Для матричных игр [34] существование седловых точек равносильно равенству

$$\max_i \min_j a_{ij} = \min_j \max_i a_{ij}, \tag{36}$$

где a_{ij} — элементы платёжной матрицы. Схема решения игры:

а) находим в каждой строке $\mathbf{A}$ min-ые элементы и из них — max-ый.

$$\left.\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}\right\} \begin{array}{l} \to \min a_{1j} \\ \to \min a_{2j} \\ \vdots \\ \to \min a_{mj} \end{array} \Big\} \to \max_i \min_j a_{ij}$$

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}}$$

$$\downarrow \quad \downarrow \quad \dots \quad \downarrow$$

$$\underbrace{\max a_{i1} \max a_{i2} \dots \max a_{in}}$$

$$\downarrow$$

$$\min_j \max_i a_{ij}$$

б) определяем в каждом столбце $\mathbf{A}$ максимальные элементы и из найденных выбираем минимальный.

Другими словами, оба игрока всегда выбирают *лучший* вариант действий из *худших*. Если найденные ими элементы совпадают, т. е. выполнено равенство (36), то найдена седловая точка, которая и является решением игры, т. к. она определяет *оптимальные* стратегии игроков. Таким образом, исход игры с седловой точкой фактически предрешён: у каждого игрока существует стратегия, следуя которой он может либо выиграть, если противник будет играть не лучшим образом, либо добиться «ничьей», если второй игрок будет играть самым лучшим образом.

4.2. Решение в смешанных стратегиях

В случае отсутствия седловой точки ни одна стратегия, сама по себе, не является предпочтительной. Однако, согласно соглашению, принятому в теории игр, игроки должны иметь полный план действий в любых возможных ситуациях. Поэтому, тот игрок, который *раскроет* план противника, приобретает явное преимущество. Следовательно, главной задачей в такой игре является *сокрытие* своих планов, что лучше всего можно сделать *выбирая стратегии случайным образом*. В таком случае все действия игрока полностью защищены от раскрытия противником, так как сам игрок наперёд не знает какой выбор он сделает.

Стратегии, выбираемые случайным образом, *с определёнными вероятностями*[8], называют *смешанными*. Первоначальные стратегии назовём *чистыми*. Т. о. смешанные стратегии игрока 1 задаются вектором

$$\mathbf{X} = (x_1, x_2, \dots, x_m), \qquad x_i \geqslant 0, \quad i = 1, \dots, m, \qquad \sum_{i=1}^{m} x_i = 1,$$

[8] Подчеркнём: стратегии выбираются случайно, но не как попало!

где x_i — вероятность, с которой игрок 1 выбирает свою i-ю чистую стратегию. Аналогично для 2-го игрока. Можно считать, что чистые стратегии также являются смешанными, взятыми с вероятностью 1. Поэтому переход к смешанным стратегиям называют *смешанным расширением игры*.

Определение 4.3. *Смешанным расширением матричной игры* $\mathbf{A}$ *называют игру*

$$\big\langle \mathbf{X}, \mathbf{Y}, H(\mathbf{X}, \mathbf{Y}) \big\rangle,$$

где

$$\mathbf{X} = (x_1, x_2, \ldots, x_m), \qquad \mathbf{Y} = (y_1, y_2, \ldots, y_n).$$

— смешанные стратегии игроков 1, 2, соответственно,

$$H(\mathbf{X}, \mathbf{Y}) = \mathbf{X}\mathbf{A}\mathbf{Y}^{\mathbf{T}} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_i y_j \tag{37}$$

— функция выигрыша игрока 1 (для 2-го игрока выигрыш противоположен по знаку).

Т. о. в смешанном расширении игры за выигрыш игрока принимается математическое ожидание выигрыша игрока в исходной игре, т. е. «среднее» значение выигрыша. Пару $(\mathbf{X}, \mathbf{Y})$ называют *ситуацией в смешанных стратегиях*.

Теорема 4.1. *В смешанном расширении любой матричной игры всегда существуют седловые точки [34], т. е. выполняется равенство*

$$\max_X \min_Y \mathbf{X}\mathbf{A}\mathbf{Y}^{\mathbf{T}} = \min_Y \max_X \mathbf{X}\mathbf{A}\mathbf{Y}^{\mathbf{T}} \stackrel{\text{def}}{=} v(\mathbf{A})$$

Число $v(\mathbf{A})$ называют *значением игры*.

Под *решением* смешанного расширения игры понимают наилучшие[9] взятые с определенными вероятностями чистые стратегии игроков.

4.3. Решение 2×2 матричных игр

В простейшем случае 2×2 игры $\mathbf{A}$, не имеющей седловой точки, нетрудно показать [35, стр. 50–51], что оптимальные смешанные стратегии игроков $\mathbf{X}^*$, $\mathbf{Y}^*$ и значение игры, соответственно, равны

$$\mathbf{X}^* = \frac{\mathbf{J}\mathbf{A}^\star}{\mathbf{J}\mathbf{A}^\star\mathbf{J}^{\mathbf{T}}}, \qquad \mathbf{Y}^* = \frac{\mathbf{A}^\star\mathbf{J}^{\mathbf{T}}}{\mathbf{J}\mathbf{A}^\star\mathbf{J}^{\mathbf{T}}}, \qquad v(\mathbf{A}) = \frac{|\mathbf{A}|}{\mathbf{J}\mathbf{A}^\star\mathbf{J}^{\mathbf{T}}}, \tag{38}$$

где $\mathbf{A}^\star$ — *присоединённая* к $\mathbf{A}$ матрица, её элементами являются алгебраические дополнения элементов матрицы $\mathbf{A}^{\mathbf{T}}$; $\mathbf{J} = (1 \;\; 1)$.

Общий алгоритм решения 2×2 матричных игр:

1. Определить седловые точки, как указано выше, стр. 172. Если их нет, перейти к п. 2, иначе — решение найдено.
2. Найти смешанные стратегии игроков, пользуясь формулами (38).

[9] Дающие наибольший выигрыш по формуле (37).

4.4. Решение $m \times n$ матричных игр

Решение матричной игры, т. е. нахождение оптимальных стратегий игроков, обычно тем труднее, чем больше стратегий имеется у игроков и, соответственно, выше размерность платежной матрицы игры

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{39}$$

В ряде случаев при моделировании игровых ситуаций можно отказаться от заведомо невыгодных стратегий, понизив тем самым размерность матрицы $\mathbf{A}$. Говорят, что чистая стратегия i' игрока 1 *доминирует*[10] его стратегию i'', если

$$a_{i'j} \geqslant a_{i''j} \qquad \forall j. \tag{40}$$

Стратегия j' игрока 2 доминирует его стратегию j'', если

$$a_{ij'} \leqslant a_{ij''} \qquad \forall i. \tag{41}$$

Отношение доминируемости относится только к стратегиям одного и того же игрока. Легко видеть, что игроки могут не употреблять своих *доминируемых* стратегий: игрок 1 может отказаться от i'', а 2 — от j''.

Например, в игре с матрицей

$$\begin{pmatrix} 2 & \mathbf{0} & 1 & 4 \\ 1 & \mathbf{2} & 5 & 3 \\ 4 & \mathbf{1} & 3 & 2 \end{pmatrix}$$

второй столбец доминирует четвертый, следовательно, игрок 2 не должен использовать свою четвертую стратегию и можно рассматривать матрицу

$$\begin{pmatrix} 2 & 0 & 1 \\ 1 & 2 & 5 \\ \mathbf{4} & \mathbf{1} & \mathbf{3} \end{pmatrix},$$

в которой третья строка доминирует первую. Удаляя ее, получим

$$\begin{pmatrix} 1 & \mathbf{2} & 5 \\ 4 & \mathbf{1} & 3 \end{pmatrix}.$$

И, наконец, удалив третий столбец, имеем

$$\begin{pmatrix} 1 & 2 \\ 4 & 1 \end{pmatrix}.$$

Т. о. размерность исходной матрицы с 3×4 удалось понизить до 2×2.

Известно, что решение матричной игры можно свести к решению задачи линейного программирования [34], [36] и решение матричной игры полностью эквивалентно решению задачи линейного программирования.

[10] от лат. dominary — преобладание

Пусть дана игра с платежной матрицей (39), тогда обозначим

$$\mathbf{p}^* = (p_1, p_2, \ldots, p_m), \qquad \mathbf{q}^* = (q_1, q_2, \ldots, q_n)$$

оптимальные смешанные стратегии игроков 1 и 2, соответственно, и v — цену игры. Получим, что для игрока 1 стратегия $\mathbf{p}^*$ гарантирует ему выигрыш не менее v, независимо от действий противника, т. е.

$$\begin{array}{l} a_{11}p_1 + a_{21}p_2 + \ \ldots \ + a_{m1}p_m \geqslant v, \\ a_{12}p_1 + a_{22}p_2 + \ \ldots \ + a_{m2}p_m \geqslant v, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{1n}p_1 + a_{2n}p_2 + \ \ldots \ + a_{mn}p_m \geqslant v, \end{array} \tag{42}$$

$$\sum_{i=1}^{m} p_i = 1, \qquad p_i \geqslant 0, \quad i = 1, \ldots, m.$$

Аналогично, для игрока 2

$$\begin{array}{l} a_{11}q_1 + a_{12}q_2 + \ \ldots \ + a_{1n}q_n \leqslant v, \\ a_{21}p_1 + a_{22}p_2 + \ \ldots \ + a_{2n}q_n \leqslant v, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}q_1 + a_{m2}q_2 + \ \ldots \ + a_{mn}q_n \leqslant v, \end{array} \tag{43}$$

где

$$\sum_{j=1}^{n} q_j = 1, \qquad q_j \geqslant 0, \quad j = 1, \ldots, n.$$

Поскольку элементы платёжной матрицы всегда можно сделать положительными (стратегическая эквивалентность), то можно считать, что $v > 0$. Поделим почленно уравнения (42), (43) на v и обозначим $p_i/v = x_i$, $q_j/v = y_j$. Получим для игрока 1

$$\begin{array}{l} a_{11}x_1 + a_{21}x_2 + \ \ldots \ + a_{m1}x_m \geqslant 1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{1n}x_1 + a_{2n}x_2 + \ \ldots \ + a_{mn}x_m \geqslant 1, \end{array} \tag{44}$$

$$\sum_{i=1}^{m} x_i = 1/v, \qquad x_i \geqslant 0, \quad i = 1, \ldots, m. \tag{45}$$

Для игрока 2 — аналогично:

$$\begin{array}{l} a_{11}y_1 + a_{12}y_2 + \ \ldots \ + a_{1n}y_n \leqslant 1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}y_1 + a_{m2}y_2 + \ \ldots \ + a_{mn}y_n \leqslant 1, \end{array} \tag{46}$$

где

$$\sum_{j=1}^{n} y_j = 1/v, \qquad y_j \geqslant 0, \quad j = 1, \ldots, n. \tag{47}$$

Таким образом, исходная матричная игра сведена к двум (*двойствен-*

ным) задачам[11] линейного программирования:

Для игрока 1 найти

$$\min(x_1 + x_2 + \ldots + x_m)$$

при ограничениях (44), (45).

Для игрока 2 найти

$$\max(y_1 + y_2 + \ldots + y_n)$$

при ограничениях (46), (47).

Найдя решения $\mathbf{x}^*$, $\mathbf{y}^*$ задачи линейного программирования (например, симлекс-методом, см. стр. 177), определим оптимальные смешанные стратегии

$$p_i = \frac{x_i^*}{\sum\limits_{k=1}^{m} x_k^*}, \qquad i = 1, \ldots, m;$$

$$q_j = \frac{y_j^*}{\sum\limits_{k=1}^{n} y_k^*}, \qquad j = 1. \ldots, n;$$

Таким образом, порядок построения и решения игровых моделей, описываемых матричными играми (нахождения оптимальных стратегий игроков) таков:

1. Исходя из модели перечислить все стратегии игроков и все возможные ситуации игры.
2. Из условия задачи определить для каждой ситуации значение функции выигрыша, составить платежную матрицу игры.
3. Исключить доминируемые строки и столбцы платежной матрицы, если таковые имеются.
4. Проверить платежную матрицу на наличие седловых точек; если седловых точек нет, искать решение в смешанных стратегиях, иначе — задача решена.
5. Найти смешанные стратегии игроков любым способом[12] (найти все оптимальные решения).

[11] Можно показать, что и обратно, пара двойственных задач линейного программирования соответствует некоторой матричной игре.

[12] Кроме симплекс-метода, описанного ниже, существуют и другие способы решения матричных игр, как-то: графический метод, итеративный метод Брауна — Робинсон (метод фиктивного разыгрывания), монотонный итеративный алгоритм [38] и др.

5. Симплекс-метод

Симплекс-метод[1] [52] применяется для решения *задачи линейного программирования:* найти неотрицательное решение $x_1, \ldots, x_n$ системы

$$\begin{cases} x_1 + a_{1,r+1}x_{r+1} + \ldots + a_{1,n}x_n = b_1, \\ x_2 + a_{2,r+1}x_{r+1} + \ldots + a_{2,n}x_n = b_2, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_r + a_{r,r+1}x_{r+1} + \ldots + a_{r,n}x_n = b_r, \end{cases} \tag{48}$$

где все свободные члены b_i неотрицательны, минимизирующее *целевую функцию:*

$$f = c_0 - (c_{r+1}x_{r+1} + \ldots + c_nx_n) \to \min.$$

Система ограничений (48) — система линейных уравнений, в которой количество неизвестных больше количества уравнений. Если ранг какой-либо системы линейных уравнений равен r, то, как известно, можно выбрать r *базисных* неизвестных, которые выражены через остальные *свободные* неизвестные. В нашем случае, для определенности, предполагается, что выбраны первые, идущие подряд, неизвестные $x_1, \ldots, x_r$. К такому виду можно привести любую совместную систему.

Область допустимых решений, задаваемая системой ограничений (48), геометрически представляется выпуклым многогранником в n-мерном пространстве и минимальное значение целевой функции достигается в одной или нескольких его вершинах. Решение задачи начинается с рассмотрений одной из вершин многогранника условий. Если исследуемая вершина не соответствует минимуму, то переходят к соседней, уменьшая значение целевой функции. Таким образом, переход от одной вершины к другой постепенно улучшает значение функции цели, а так как число вершин многогранника конечно, то за конечное число шагов гарантируется нахождение минимального значения или установление факта неразрешимости задачи.

5.1. Алгоритм симплекс-метода

Запишем все данные в виде таблицы

базис		x_1	$\ldots$	x_i	$\ldots$	x_r	x_{r+1}	$\ldots$	x_j	$\ldots$	x_n
x_1	b_1	1	$\ldots$	0	$\ldots$	0	$a_{1,r+1}$	$\ldots$	$a_{1,j}$	$\ldots$	$a_{1,n}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_i	b_i	0	$\ldots$	1	$\ldots$	0	$a_{i,r+1}$	$\ldots$	$a_{i,j}$	$\ldots$	$a_{i,n}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_r	b_r	0	$\ldots$	0	$\ldots$	1	$a_{r,r+1}$	$\ldots$	$a_{r,j}$	$\ldots$	$a_{r,n}$
f	c_0	0	$\ldots$	0	$\ldots$	0	c_{r+1}	$\ldots$	c_j	$\ldots$	c_n

[1] Симплекс (от лат. simplex — простой) — геометрическая фигура, n-мерное обобщение треугольника. Геометрически алгоритм метода соответствует перебору вершин выпуклого многогранника в многомерном пространстве. Симплекс-метод был разработан советским математиком Л. В. Канторовичем.

1. Определить имеются ли в последней строке таблицы (кроме c_0) положительные числа. Если все числа отрицательны, то процесс останавливается, оптимальное решение — $(b_1, b_2, \ldots, b_r, 0, 0, \ldots, 0)$, значение целевой функции: $f = c_0$. Если есть положительные числа, перейти к шагу 2.
2. Проверить столбцы, соответствующие положительным числам из последней строки, на наличие в них положительных чисел. Если ни в одном из таких столбцов нет положительных чисел, то процесс останавливается, оптимальное решение не существует. Если найден столбец, содержащий хотя бы один положительный элемент (если таких столбцов несколько, можно взять любой), отметить его (запомнить №) и перейти к п. 3.
3. Разделить свободные члены на соответствующие положительные элементы отмеченного столбца и выбрать наименьшее частное. Отметить строку, соответствующую наименьшему частному. Запомнить элемент таблицы, стоящий на пересечении выделенных столбца и строки (этот элемент называется *разрешающим*). Заменить базисное неизвестное, стоящее против разрешающего элемента на неизвестное, стоящее в столбце разрешающего элемента. Перейти к п. 4.
4. Разделить все элементы отмеченной строки на разрешающий элемент. Перейти к п. 5.
5. Все строки таблицы, кроме отмеченной, складываются поэлементно с отмеченной строкой, умноженной на такое число, чтобы в результате в отмеченном столбце появились нули. Результат сложения заменяет «старую» строку. Перейти к п. 1.

Пример

Пусть дана задача линейного программирования

$$\begin{cases} x_1 - x_4 + x_5 = 2, \\ x_2 + 2x_4 + 3x_5 = 7, \\ x_3 + x_4 - 2x_5 = 1, \end{cases}$$

$$f - x_4 + 2x_5 = 3.$$

базис		x_1	x_2	x_3	x_4	x_5
x_1	2	1	0	0	-1	1
x_2	7	0	1	0	2	3
x_3	1	0	0	1	1	-2
f	3	0	0	0	-1	2

Разрешающий элемент выделен красным цветом. В соответствии с пп. 4, 5 алгоритма получаем

базис		x_1	x_2	x_3	x_4	x_5
x_5	2	1	0	0	-1	1
x_2	1	-3	1	0	5	0
x_3	5	2	0	1	-1	0
f	-1	-2	0	0	1	0

Поскольку в последней строке есть положительный элемент, процесс не закончен. Снова, начиная с пункта 1, получим

базис		x_1	x_2	x_3	x_4	x_5
x_5	2,2	0,4	0,2	0	0	1
x_4	0,2	$-0,6$	0,2	0	1	0
x_3	5,2	1,4	0,2	1	0	0
f	$-1,2$	$-1,4$	$-0,2$	0	0	0

Решение: $(0; 0; 5{,}2; 0{,}2; 2{,}2)$.

6. Аппроксимация табличной функции

Задача приближения (аппроксимации[1]) возникает, когда требуется по набору дискретных данных, полученных, например, экспериментально, найти какую либо *простую приближенную функциональную зависимость* для этих данных. В более общем случае можно дать следующее

Определение 6.1. Аппроксимация — *замена одних математических объектов другими, более простыми и в каком-то смысле* близкими *к исходным. В частности — замена более сложных, «неудобных» функций на более простые, более «удобные» для определенных целей.*

Пусть задана таблица чисел

x_0	x_1	x_2	$\dots$	x_n
y_0	y_1	y_2	$\dots$	y_n

Приближение ищется в виде функции:

$$F(x, a_0, a_1, \dots, a_m) = \sum_{k=0}^{m} a_k \varphi_k(x),$$

где $\varphi_k(x)$ — заданные базисные функции, a_k — коэффициенты, подлежащие определению. Чаще всего используют аппроксимацию полиномами, $\varphi_k(x) = x^k$, тогда задача сводится к поиску полинома степени m, $(m < n)$, приближающего табличные значения:

$$F(x, a_0, a_1, \dots, a_m) = \sum_{k=0}^{m} a_k x^k. \tag{49}$$

[1] От лат. approximare — приближаться.

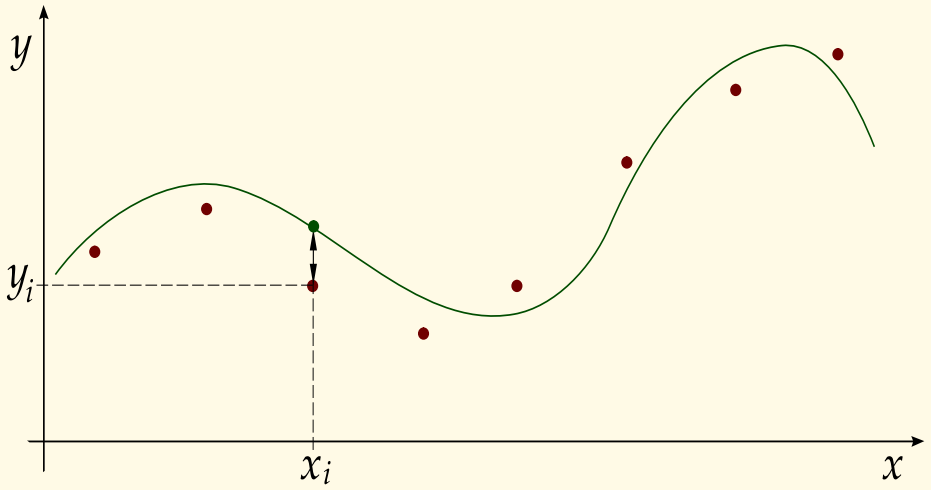


Рис. 4. *Аппроксимация*

6.1. Метод наименьших квадратов

Для определения коэффициентов a_k будем искать такой многочлен, отклонение значений которого $F(x_i, a_0, a_1, \ldots, a_m)$ от табличных значений y_i *минимально* «в среднем». В качестве меры уклонения («близости») выберем сумму квадратов разностей, т. е. значение функционала

$$\Delta(a_0, a_1, \ldots, a_m) = \sum_{i=0}^{n} \Big(F(x_i, a_0, a_1, \ldots, a_m) - y_i \Big)^2. \qquad (50)$$

Необходимым условием минимума функции многих переменных является равенство нулю ее частных производных первого порядка по всем независимым переменным. В функционале (50) такими независимыми переменными являются коэффициенты a_k.

$$\begin{aligned}
\frac{\partial \Delta}{\partial a_0} &= 2 \sum_{i=0}^{n} \Big(F(x_i, a_0, a_1, \ldots, a_m) - y_i \Big) = 0, \\
\frac{\partial \Delta}{\partial a_1} &= 2 \sum_{i=0}^{n} x_i \Big(F(x_i, a_0, a_1, \ldots, a_m) - y_i \Big) = 0, \\
\frac{\partial \Delta}{\partial a_2} &= 2 \sum_{i=0}^{n} x_i^2 \Big(F(x_i, a_0, a_1, \ldots, a_m) - y_i \Big) = 0, \qquad (51) \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\frac{\partial \Delta}{\partial a_m} &= 2 \sum_{i=0}^{n} x^m \Big(F(x_i, a_0, a_1, \ldots, a_m) - y_i \Big) = 0,
\end{aligned}$$

Система (51) представляет собой систему линейных алгебраических уравнений порядка $m+1$ относительно неизвестных a_i, $i = 0, 1, \ldots m$.

Поскольку второй дифференциал функции Δ, является положитель-

но определенной квадратичной формой:

$$\sum_{i,j=1}^{n} \frac{\partial^2 \Delta}{\partial a_i \partial a_j} da_i da_j > 0,$$

то решения a_i, $i = 0, 1, \dots m$ доставляют *минимум* функционалу Δ (достаточные условия минимума).

Решение СЛАУ (51) может быть осуществлено любым из известных методов. Подставляя найденные в результате решения СЛАУ значения в (49), получаем полином, наилучшим образом приближающий дискретную табличную функцию в «среднеквадратичном» смысле. Точность аппроксимации обычно оценивают величинами:

$$\max_{i=0,1,\dots n} |F(x_i, a_0, a_1, \dots, a_m) - y_i| \qquad \text{(максимальное уклонение)};$$

$$\frac{1}{n+1} \sum_{i=0}^{n} |F(x_i, a_0, a_1, \dots, a_m) - y_i| \qquad \text{(среднее)};$$

Пример. По таблице населения Земли

Дата (г)	1700	1800	1900	2000
Население (млн)	780	1 000	1 650	6 070

найти аппроксимацию функции народонаселения с относительным максимальным уклонением, не превосходящим $0{,}5$.

Решение. Перепишем табличные данные в виде

x_i	0	1	2	3
y_i	0,78	1,0	1,65	6,07

и применим сначала линейную аппроксимацию

$$F(x, a_0, a_1) = a_0 + a_1 x. \tag{52}$$

Система (51) запишется в виде

$$\frac{\partial \Delta}{\partial a_0} = 2 \sum_{i=0}^{n} \big(a_0 + a_1 x_i - y_i\big) = 0,$$

$$\frac{\partial \Delta}{\partial a_1} = 2 \sum_{i=0}^{n} x_i \big(a_0 + a_1 x_i - y_i\big) = 0,$$

$$a_0(n+1) + a_1 \sum_{i=0}^{n} x_i - \sum_{i=0}^{n} y_i = 0,$$

$$a_0 \sum_{i=0}^{n} x_i + a_1 \sum_{i=0}^{n} x_i^2 - \sum_{i=0}^{n} x_i y_i = 0.$$

Подставляя табличные значения, получаем систему двух линейных уравнений

$$4a_0 + 6a_1 - 9{,}5 = 0,$$
$$6a_0 + 14a_1 - 22{,}51 = 0,$$

решение которой: $a_0 = 0{,}103$, $a_1 = 1{,}652$. Многочлен

$$F = 0{,}103 + 1{,}652\,x$$

аппроксимирует табличные данные с относительным максимальным уклонением, равным 1.13, поэтому переходим к приближению полиномом третьей степени

$$F(x, a_0, a_1, a_2) = a_0 + a_1 x + a_2 x^2. \qquad (53)$$

Аналогично получим многочлен

$$F = 0{,}947 - 1{,}498\,x + 1{,}05\,x^2,$$

с относительным максимальным уклонением, равным $0{,}5$.

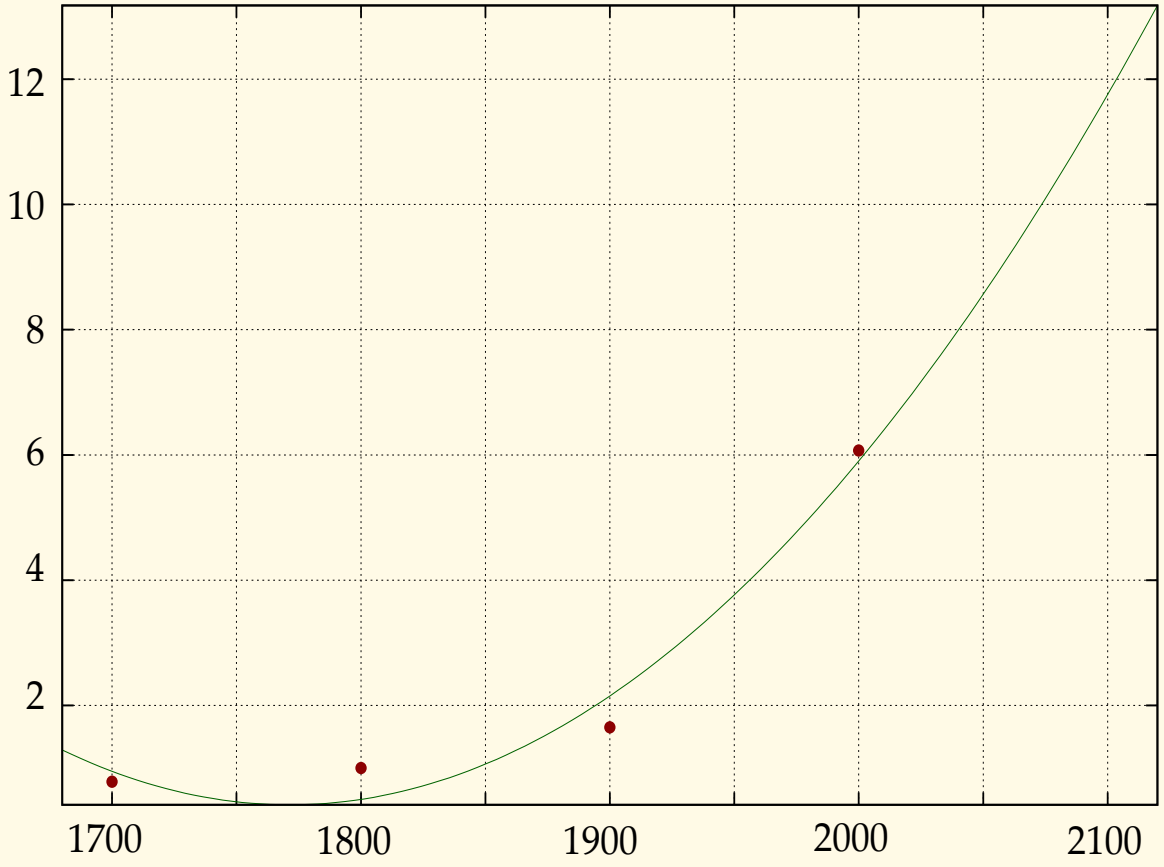


Рис. 5. *Аппроксимация данных о численности населения Земли*

На основе данной аппроксимации можно сделать прогноз численности населения Земли, скажем, на 2100 год: около 12 млрд. Этот прогноз, как и все подобные предсказания, очевидно, не очень надёжен, так как наша «модель» неявно предполагает, что условия роста народонаселения, известные до 2000 года, сохранятся и в будущем.

7. Формальные языки и грамматики

Чтобы иметь строгую математическую основу для выводов и утверждений относительно языка вначале необходимо определить это понятие абстрактно, как формальную систему [88], [90],[91].

Определение 7.1. *Произвольное непустое конечное множество A называют алфавитом. Элементы множества A называют символами (буквами) этого алфавита. Понятие символ является неопределяемым.*

Примеры алфавитов: $\{0,1\}$ — двоичный алфавит, $\{.,—\}$ — алфавит азбуки Морзе, {А, Б, В, ..., Я} — кириллица.

Определение 7.2. *Любую конечную последовательность символов (в т.ч. и пустую, обозначаемую далее ε) алфавита нызывают цепочкой (строкой, string, словом).*

Множество всех цепочек из алфавита A обозначим A^*. Например, если $A = \{0,1\}$ — двоичный алфавит, то

$$A^* = \{\varepsilon, 0, 1, 00, 11, 01, 10, 000, 001, 011, \dots\}.$$

Множество всех непустых цепочек обозначим A^+. Таким образом, имеем соотношение: $A^* = A^+ \bigcup \{\varepsilon\}$.

Определение 7.3. *Подмножество L_A множества A^* называют языком над A.*

Есть различные способы задания языка, т. е. методы определения всех цепочек, составляющих язык. Для достаточно сложных языков часто применяют не словесное описание, а *формальное*, с тем, чтобы принадлежность цепочки языку можно было установить (*распознать* язык), исходя только из определений. Наиболее распространен способ задания языка с помощью *порождающей грамматики*[1].

Определение 7.4. *Метаалфавитом для A называют алфавит M, не пересекающийся с A. Элементы M называют метасимволами*[2] *(нетерминалами, nonterminal).*

Определение 7.5. *Пусть M — метаалфавит для A. Грамматикой языка L_A называют конечную систему вырыжений (систему правил вывода, свёрток, reduce), вида*

$$\alpha \to \beta,$$

[1] Грамматика (греч. γραμμᾰτική — от γράμμα — буква, написание) строй языка, т. е. система языковых форм, способов словопроизводства, синтаксических конструкций.

[2] От греч. μετα — вслед за тем, после, переход к чему-либо другому.

где $\alpha \in (A \bigcup M)^+$, $\beta \in (A \bigcup M)^*$, *символ «*$\rightarrow$*» означает: «из* α *выводится* β*», вместо него употребляются также обозначения: «::=», «:». В грамматике выделяют один метасимвол, называемый главным (начальным, аксиомой). Содержательно любое правило вывода имеет смысл подстановки правой его части в левую.*

Языком, порожденным данной грамматикой, является множество цепочек A^**, выводимых (необязательно непосредственно, напрямую) из начального метасимвола.*

Цепочки из M^+, используемые в грамматике, составляют, таким образом, некоторый *метаязык*[3], т. е. язык, средствами которого производится описание исходного языка. Например, при изучении иностранного языка или, скажем, Pascal'я, его описание производится на родном языке.

Легко видеть, что если в цепочке имеется метасимвол, то ее можно преобразовать далее, если использовать какое-либо правило вывода с этим метасимволом в левой части правила. В противном случае цепочку уже нельзя преобразовывать. Поэтому символы из A^* называют *терминалами*[4], другое название — *токены* (tokens), а метасимволы называют *нетерминалами*. Метасимволы можно трактовать как названия понятий, которые вводятся для формального описания языка, поэтому часто для ясности при обозначении нетерминалов используют эти названия, заключенные в угловые скобки.

Пример №1. Грамматика простейших арифметических выражений (термов) задается правилами

```
<терм>   ::=   x | y
             |  (<терм>)
             |  <терм> + <терм>
             |  <терм> − <терм>
             |  <терм> * <терм>
             |  <терм> / <терм>
```

Здесь «::=» обозначает подстановку правой части правила вместо левой, а «|» — логическое «или», «<терм>» — главный метасимвол. Такой способ записи называется *нормальной формой Бэкуса — Наура* (J. W. Backus, P. Naur).

При записи правил с одинаковыми правыми частями можно пользоваться сокращённой формой: вместо

```
лев_часть   ::=   пр_часть1
лев_часть   ::=   пр_часть2
лев_часть   ::=   пр_часть3
```

[3] См. сноску на стр. 183.

[4] Англ. terminal — заключительный, конечный, окончательный.

пишут

```
лев_часть   ::=  пр_часть1
               |  пр_часть2
               |  пр_часть3
```

(для лучшей читабельности альтернативы обычно принято располагать в отдельных строках). Как и в приведенном примере, грамматика часто задаётся без явного указания множеств теминалов и нетерминалов. Предполагается, что грамматика содержит только те символы, которые фигурируют в правилах вывода (в примере это единственный метасимвол <терм> и терминалы (,), +, −, ∗, /, x, y), главный метасимвол обычно записывается в левой части самого первого правила.

Пример №2. Калькулятор с постфиксной (польской) записью. Постфиксная запись означает, что знаки операций (действий) в такой нотации располагаются после операндов: вместо «2 + 5» пишут «2 5+», вместо «2 + 5 ∗ 3» — «2 5 3 ∗ +» и т. д.

```
<операнд> ::= NUMBER
            | <операнд> <операнд> +
            | <операнд> <операнд> −
            | <операнд> <операнд> ∗
            | <операнд> <операнд> /
```

Один и тот же язык, очевидно, можно описать при помощи разных наборов понятий и, следовательно, задать различными грамматиками.

Задача *синтаксического*[5] *анализа* — определить является ли данная цепочка правильной, т.е. принадлежащей языку, и построить для нее последовательность выводов («свёртку») к главному метасимволу. При этом, если правила грамматики (подстановки) применяются только к самому левому (правому) нетерминалу, то говорят, что вывод (разбор) является *левосторонним* (*правосторонним*). Кроме того, существуют два способа синтаксического разбора: *восходящий* и *нисходящий*. При нисходящем разборе вывод строится от главного метасимвола, при восходящем — в обратном направлении.

Произведем последовательный вывод (левосторонний и нисходящий) цепочки «(x + y)∗x» в вышеприведенной грамматике термов

```
<терм> ::= <терм>∗<терм> ::= (<терм>)∗<терм>
       ::= (<терм> + <терм>)∗<терм>::=(x + <терм>)∗<терм>
       ::= (x + y)∗<терм> ::= (x + y)∗x
```

Следовательно, эта цепочка «(x + y)∗x» принадлежит языку (является правильным «простейшим арифметическим выражением»), порожденному этой грамматикой, а, к примеру, цепочки «(y + x» и «−x∗y»

[5] *Синтаксис* (*греч.* σύνταξις — *конструкция, построение, порядок*) — совокупность правил языка, определяющих формирование его элементов.

не принадлежат, т.к. для них нельзя построить вывод на основе указанных правил. Вывод можно также представить в виде «дерева», для нашего случая:

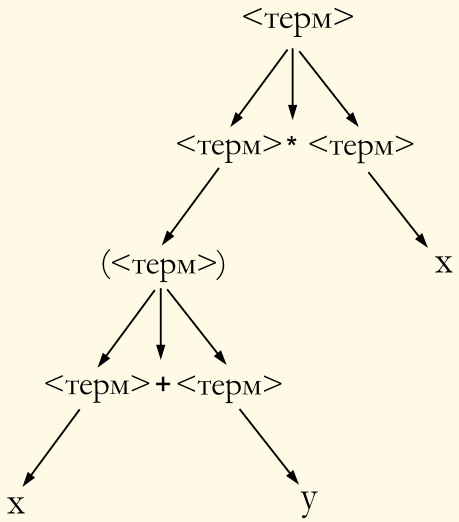


Рис. 6. *Дерево вывода*

Определение 7.6. *Грамматику называют неоднозначной, если она допускает цепочки, для вывода которых существуют два различных дерева. В противном случае грамматику называют однозначной.*

Известно, что проблема определения однозначности грамматики в общем случае неразрешима, т. е. не существует алгоритма, который может определить однозначность *любой* грамматики.

Грамматики различаются по виду их правил вывода. Нас в дальнейшем будут интересовать только те, которые в классификации Н. Хомского (N. Chomsky) называются «контекстно-свободными».

Определение 7.7. *Грамматика называется контекстно-свободной или КС-грамматикой (context-free grammar), если каждое её правило имеет вид*

$$a ::= b,$$

где $a \in M^+$, $b \in (A \bigcup M)^+$.

Другими словами, это грамматика, у которой левые части всех правил являются нетерминалами. Для практики наиболее важны именно контекстно-свободные грамматики, так как ими определяется достаточно широкий класс языков и хорошо изучены алгоритмы для распознавания языка, реализуемые *автоматами со стеком* (стр. 187). Именно КС-грамматиками задаётся большинство языков программирования.

Определение 7.8. *Язык, который может быть задан КС-грамматикой, называется контекстно-свободным языком.*

8. Конечные автоматы

Конечным автоматом[1] (КА) называют систему, которая в каждый момент времени может находиться в одном из конечного множества *состояний*. Каждый шаг (переключение) КА вызывается поступлением на его вход одного из определённых *сигналов* (воздействий) и заключается в переходе из одного состояния в другое. Далее будем считать, что сигналы представляют собой символы некоторого входного алфавита Σ. Первоначально автомат находится в состоянии q_0, затем, по мере чтения входных символов, он переключается между состояниями — элементами некоторого определённого множества Q. Если, прочитав входной символ, автомат оказывается в некотором состоянии из конечного множества $F, F \subseteq Q$, то говорят, что он *принял* ее. Более формально

Определение 8.1. *Конечным автоматом называют пятёрку* $(Q, \Sigma, \delta, q_0, F)$, *где* Q *— конечное непустое множество состояний;* Σ *— конечный входной алфавит;* q_0 *— начальное состояние,* $q_0 \in Q$, F *— множество конечных состояний,* $F \subseteq Q$, δ *— правила переключений:*

$$\delta : Q \times \Sigma \to Q$$

Т. о., запись вида $\delta(q_1, a) = q_2$ означает, что КА, находясь в состоянии q_1, считал символ a и перешёл в состояние q_2. Функцию δ можно естественным образом продолжить на множество $Q \times \Sigma^*$, используя соотношения

$$\delta(q, \varepsilon) = q,\ \delta(q, ab) = \delta(\delta(q, a), b),$$

тогда запись $\delta(q_1, x) = q_2$ будет означать, что КА, считав *цепочку* x, $x \in \Sigma^*$, окажется в состоянии q_2.

Определение 8.2. *Множество всех цепочек* $x \in \Sigma^*$, *принимаемых КА, называется языком, распознаваемым конечным автоматом.*

Наглядно КА удобно представлять как совокупность одинаковых *клеток*, соединенных между собой в так называемую, *решетку КА*. Решетки могут быть разных типов, отличаясь как по размерности, так и по форме клеток. Обычно используют квадратные клетки, образующие прямоугольную сеть. Каждая клетка является КА, состояния которого определяются обычно лишь состояниями соседних клеток и, возможно, ее собственным состоянием. Для практической реализации КА важны следующие свойства:

- Изменения состояний всех клеток происходят одновременно после вычисления нового состояния каждой клетки решетки.
- Решетка однородна, все её клетки «равноправны».

[1] От греч. αὐτόματος — самодействующий — устройство, способное выполнять какие-либо действия без непосредственного участия человека.

- Воздействия локальны. Лишь клетки окрестности (соседи) влияют на состояние данной клетки.
- Множество состояний клетки конечно.

КА в настоящее время широко применяются для моделирования в физике, биологии, экономике, социологии, информатике и т. д. [94], [95], [96]. «. . . клеточный автомат может мыслиться как стилизованный мир. Пространство представлено равномерной сеткой, каждая ячейка или клетка которой содержит несколько битов данных; время идет вперед дискретными шагами, а законы мира выражаются единственным набором правил, скажем, небольшой справочной таблицей, по которой любая клетка на каждом шаге вычисляет своё новое состояние по состояниям её близких соседей. Таким образом, законы системы являются локальными и повсюду одинаковыми. „Локальный“ означает, что для того, чтобы узнать, что произойдет здесь мгновение спустя, достаточно посмотреть на состояние ближайшего окружения: никакое дальнодействие не допускается. „Одинаковость“ означает, что законы везде одни и те же: я могу отличить одно место от другого только по форме ландшафта, а не по какой-то разнице в законах» [94].

Наиболее известным примером клеточного автомата, видимо, является игра «Жизнь», созданнная в 1970 году Дж. Конуэем (John Horton Conway). Возникающие в процессе игры ситуации имитируют в самом общем виде реальные процессы, происходящие при зарождении, развитии и гибели живых организмов. Правила «Жизни» определяются тремя законами:

1. Выживание. Каждая клетка, имеющая две или три соседние живые клетки, выживает и переходит в следующее поколение.
2. Гибель. Каждая клетка, у которой больше трёх соседей, погибает из-за перенаселённости. Каждая клетка, вокруг которой свободны все соседние клетки или же занята всего одна клетка, погибает от одиночества.
3. Рождение. Если число занятых клеток, с которыми граничит какая-нибудь пустая клетка, в точности равно трём, то на этой клетке происходит рождение нового организма.

При этом предполагается, что: а) у любой клетки — 8 соседей (все смежные клетки, включая соседей по диагонали); б) рождение и смерть организмов происходят *одновременно*, все вместе взятые клетки в течение одного цикла рождения-смерти образуют одно *поколение*.

Несмотря на простоту законов Конуэя поведение КА «Жизнь» может быть чрезвычайно сложным и разнообразным — существуют беспредельно развивающиеся популяции и такие, которые заканчивают своё развитие одним из следующих способов: полностью исчезают; переходят в устойчивые постоянные конфигурации, образуя бесконечный колебательный режим.

Если дополнить КА *стеком*[2], то получим т. н. *магазинный автомат* (МА, PDA — *pushdown automaton*). МА имеет дополнительно рабочую память — магазин (стек), в который записываются символы из еще одного алфавита — *алфавита магазинных символов*. Каждое переключение МА определяется в зависимости от текущего состояния, входного символа или независимо от него — т. н. *ε-переход*.

Одно переключение МА состоит в замещении верхнего символа магазина (на вершине стека) некоторой магазинной цепочкой, в частности она может быть пустой (стирание верхнего символа магазина), и переходе в новое состояние. При этом текущим входным символом становится следующий входной символ, если выполняется переход, зависящий от входного символа, либо текущий входной символ остается тем же самым, если выполняется ε-переход. Поскольку переключения зависят от верхнего символа магазина, то с самого начала в магазине должен находиться один символ — начальный символ магазина.

Входная цепочка считается принятой, если существует хотя бы одна последовательность выборов переключений, которая приводит автомат к *конечной конфигурации* — входная цепочка прочитана, текущее состояние — конечное или магазин уже пуст. В противном случае она не принимается. Множество всех цепочек, принимаемых данным магазинным автоматом, называется *языком, распознаваемым этим магазинным автоматом*.

Понятие магазинного автомата полностью эквивалентно классу КС-языков (см. стр. 186) в том смысле, что любой КС-язык распознается каким-нибудь МА , и любой МА распознает некоторый КС-язык.

[2]См. сноску на стр. 95.

Литература

1. А. А. Самарский, А. П. Михайлов. Математическое моделирование. М, 1997.
2. В. И. Зенкин. Практический курс математического и компьютерного моделирования. Учеб. пособие. Калининград: изд РГУ им. Канта, 2006.
3. Л. Купер. Физика для всех. М, 1973.
4. А. Эндрю. Искусственный интеллект. М, 1981.
5. В. В. Меншуткин. Искусство моделирования (экология, физиология, эволюция). Петрозаводск — СПб, 2010.
6. С. П. Капица, С. П. Курдюмов, Г. Г. Малинецкий. Синергетика и прогнозы будущего. М.: УРСС, 2003.
7. А. А. Самарский. Введение в теорию разностных схем. М, 1974.
8. А. А. Самарский. Теория разностных схем. М, 1977.
9. Н. Н. Калиткин. Численные методы. М, 1978.
10. К. К. Пономарев. Составление и решение дифференциальных уравнений инженерно-технических задач. М.: гос. учебно-педагогическое изд-тво Министерства просвещения РСФСР, 1962.
11. Э. Камке. Справочник по обыкновенным дифференциальным уравнениям. М.: Наука: Гл. ред. физ-мат. лит., 1971.
12. В. Ф. Зайцев, А. Д. Полянин. Справочник по обыкновенным дифференциальным уравнениям. М.: Физматлит, 2001.
13. В. К. Романко. Разностные уравнения. М.: БИНОМ, 2006.
14. В. Ш. Бурд. Дискретное операторное исчисление и линейные разностные уравнения: учеб. пособие. Ярославль: ЯрГУ, 2009.
15. Л. А. Мироновский. Моделирование разностных уравнений: учеб. пособие. СПб.: СПбГУАП, 2004.
16. И. Пригожин, И. Стенгерс. Порядок из хаоса, М, 1986.
17. Дж. Глейк. Хаос. Создание новой науки, СПб, 2001.
18. Г. А. Гальперин, А. Н. Земляков. Математические биллиарды, М, 1990.
19. P. Prusinkiewicz and J. Hanan. Lindenmayer Systems, Fractals, and Plants. Springer — Verlag, New York, 1989.
20. А. С. Галиулин. Аналитическая динамика. М, 1989.
21. Дж. Уизем. Линейные и нелинейные волны. М, 1977.
22. Н. Г. Четаев. Теоретическая механика. М, 1987.
23. А. Н. Тихонов, А. А. Самарский. Уравнения математической физики. М, 1972.
24. А. П. Карташев, Б. Л. Рождественский. Обыкновенные дифференциальные уравнения и основы вариационного исчисления. М, 1986.
25. Е. Бенькович, Ю. Колесов, Ю. Сениченков. Практическое моделирование динамических систем. Спб. 2002.

26. М. А. Лаврентьев, Б. В. Шабат. Методы теории функций комплексного переменного. М, 1958.
27. И. М. Воронков. Курс теоретической механики. М, 1966.
28. Г. Н. Дубошин. Небесная механика. Аналитические и качественные методы, М, 1978.
29. Н. Н. Воробьёв. Теория рядов. М, 1986.
30. А. А. Самарский. Введение в численные методы. Учебное пособие для вузов. 3-е изд., стер. СПб.: Издательство «Лань», 2005.
31. Н. С. Бахвалов. Численные методы. М.: Наука, 1975.
32. Ф. С. Робертс. Дискретные математические модели с приложениями к социальным, биологическим и экологическим задачам. М.: Наука, 1986. стр. 287–288.
33. Дж. фон Нейман, О. Моргенштерн. Теория игр и экономическое поведение. М.: Наука, 1970.
34. Н. Н. Воробьёв. Теория игр. Лекции для экономистов — кибернетиков. Л.: изд. ЛГУ, 1974.
35. Г. Оуэн. Теория игр. М.: Мир, 1971.
36. Л. А. Петросян и др. Теория игр. М.: Высшая школа, 1998.
37. В. Г. Суздаль. Теория игр для флота. М.: Воениздат, 1976.
38. В. З. Беленький. Итеративные методы в теории игр и программировании. М.: Наука, 1977.
39. И. Р. Шафаревич. Россия и мировая катастрофа. //Наш современник. 1993. № 1, стр. 100–129.
40. Дж. Б. Мангейм, Р. К. Рич. Политология. Методы исследования. М.: Весь Мир, 1997.
41. K. J. Arrow, Social Choice and Individual Values. John Wiley, New York, 1963.
42. В. Пахомов. Демократия с точки зрения математики// Квант, 1992, №№9–10.
43. В. Г. Гриб, Л. Е. Окс. Противодействие коррупции. М.: Московская финансово-промышленная академия, 2011.
44. Хрестоматия государства и права зарубежных стран. Древность и Средние века/сост. В. А. Томсинов. М.: Зерцало, 2004. стр. 34.
45. Д. А. Зенюк, Г. Г. Малинецкий, Д. С. Фаллер. Социальная модель коррупции в иерархических структурах //Препринты ИПМ им. М. В. Келдыша. 2013. № 87. http://library.keldysh.ru/preprint.asp?id=2013-87.
46. Г. Г. Малинецкий. Нелинейная динамика — ключ к теоретической истории?, Препринты ИПМ, 1995, 081. www.library.keldysh.ru/preprint.asp?id=1995-81&lg=r.
47. S. Rinaldi, G. Feichtinger, F. Wirl. Corruption Dynamics in Democratic Societies. Complexity, vol. 3, no. 5, 1998.
48. Л. Н. Гумилёв. Этногенез и биосфера Земли. М.: Эксмо, 2007.
49. А. Дж. Тойнби. Постижение истории. М.:Айрис-Пресс, 2002.

50. П. В. Турчин. Историческая динамика: На пути к теоретической истории. Изд. 2-е, М.: Издательство ЛКИ, 2010.
51. А.В. Коротаев и др. Законы истории. Математическое моделирование исторических макропроцессов. Демография, экономика, войны. М.: КомКнига, 2005.
52. Акулич И. Л. Математическое программирование в примерах и задачах. М.: Высшая школа, 1986.
53. У. Дал, Э. Дейкстра, Э. Хоор. Структурное программирование. М, 1975.
54. Н. Вирт. Алгоритмы и структуры данных. М, 1989.
55. А. Г. Кушниренко, Г. В. Лебедев. Программирование для математиков. М, 1988.
56. Т. Бадд. Объектно-ориентированное программирование в действии. СПб, 1998.
57. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.-СПб., 1998.
58. Д. Тейлор, Дж. Мишель, Дж. Пенман, Т. Гоггин, Дж. Шемитц. Delphi 3: библиотека программиста. Спб, 1996.
59. А. Голуб. C & C++. Правила программирования. М, 1996.
60. В. А. Скляров. Язык C++ и объектно-ориентированное программирование. Минск, 1997.
61. Д. Кнут. Искусство программирования для ЭВМ. М, Т. 2: Получисленные алгоритмы, 1977; Т. 3: Сортировка и поиск, 1978.
62. Г. Майерс. Искусство тестирования программ. М.: Финансы и статистика, 1982.
63. Крис Касперски. Техника оптимизации программ. Эффективное использование памяти. СПб.: БХВ-Петербург, 2003.
64. http://www.povray.org/.
65. Е. И. Бутиков и др. Физика в примерах и задачах. М.: Наука, 1983.
66. Д. Пойа. Математическое открытие. М, 1970.
67. Д. Пойа, Г. Сеге. Задачи и теоремы из анализа. М, 1978.
68. Дж. Торнли. Математические модели в физиологии растений. Киев: Наук. думка, 1982
69. Н. Бейли. Математика в биологии и медицине: Пер. с англ. М.: Мир, 1970.
70. Г. Ю. Ризниченко. Лекции по математическим моделям в биологии. Часть 1. Ижевск: НИЦ «Регулярная и хаотическая динамика», 2002.
71. К. Н. Токаревич, Т. И. Грекова. По следам минувших эпидемий. Л: Лениздат, 1986.
72. Д. М. Даймонд. Ружья, микробы и сталь. М: АСТ, 2009.
73. M. Oldstone. Viruses, plagues, and history: past, present, and future. Oxford University Press, 2010.

74. Игры, фильмы и книги про зомби: http://ozomby.ru/; http://www.ekranka.ru/?misc=zombie; http://kinozombi.ru/ .
75. С. Г. Кара-Мурза, С. В. Смирнов. Манипуляция сознанием-2. М.: Эксмо, Алгоритм, 2009.
76. Ноам Хомский. Десять стратегий манипулирования с помощью СМИ. http://psyfactor.org/lib/manipulation3.htm.
77. P. Munz, I. Hudea, J. Imad and R.J. Smith. When zombies attack!: Mathematical modelling of an outbreak of zombie infection. Infectious Disease Modelling Research Progress, Nova science publishers, 2010, pp. 133–150.
78. R.M. May. Periodical cicadas. Nature, 277, 1979, pp. 347–349.
79. Goles, E., Schulz, O. and M. Markus (2001). Prime number selection of cycles in a predator-prey model, Complexity 6(4): 33–38.
80. В. В. Похлебкин. История водки. М: Центрполиграф, 2005.
81. Alcohol, vol. 15, №2, pp. 147–160, 1998.
82. В. А. Яворский, П. П Григал. Основы количественной биологии. Методические указания к семинару. М: МФТИ, 2009.
83. Chris Hedges (July 6, 2003). What Every Person Should Know About War. The New York Times. http://www.nytimes.com.
84. И. Г. Малкин. Теория устойчивости движения. М.: Наука, 1966.
85. Н. Г. Четаев. Устойчивость движения. М.: Гостехиздат, 1955.
86. Encapsulated PostScript File Format Specification. http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf.
87. www.ghostscript.com
88. Б. К. Мартыненко. Языки и трансляции. Спб.: Изд-во С.-Петерб. ун-та, 2004.
89. A. V. Aho, S. C. Johnson. LR Parsing. Comp. Surveys Vol. 6(2) pp. 99-124 (June 1974).
90. А. Ахо, М. Лам, Р. Сети, Ульман Дж. Компиляторы: принципы, технологии и инструментарий. М.: «Вильямс», 2008.
91. Ф. В. Гладкий, И .А. Мельчук. Элементы математической лингвистики. М., 1969.
92. А. В. Костельцев. Построение интерпретаторов и компиляторов. Спб.: Наука и техника, 2001.
93. T.J. Pennello, F. DeRemer. Efficient Computation of LALR(1) Look-Ahead Sets. 615–649. TOPLAS vol 4, №4, 1982.
94. Т. Тоффоли, Н. Марголус. Машины клеточных автоматов. М.: Мир, 1991.
95. Г. Г. Малинецкий, М. Е. Степанцов. Клеточные автоматы для расчета некоторых газодинамических процессов. Ж. вычисл. матем. и матем. физ., 1996, том 36, №5, 137–145.
96. S. Wolfram. A New Kind of Science. Wolfram Media, Inc., May 14, 2002.
97. Р. Пенроуз. Новый ум короля. М. 2003.