САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

Кафедра вычислительной физики

В.А. Градусов, А.В. Цыганов

ПРАКТИКУМ ПО КОМПЬЮТЕРНЫМ СРЕДСТВАМ И СИСТЕМАМ, ЧАСТЬ 2

Учебно-методическое пособие

Санкт-Петербург
2019 г.

Рецензенты:
доцент кафедры вычислительной физики СПбГУ,
доктор физ.-мат. наук **В.А. Руднев**,
ведущий научный сотрудник кафедры радиофизики СПбГУ,
доктор физ.-мат. наук **М.А. Бисярин**.

Печатается по решению Ученого совета физического факультета СПбГУ.

Содержание одобрено Учебно-методической комиссией по УГСН 03.00.00 Физика и астрономия и УГСН 14.00.00 Ядерная энергетика и технологии.
Оригинал-макет подготовлен автором.

В.А. Градусов, А.В. Цыганов
ПРАКТИКУМ ПО КОМПЬЮТЕРНЫМ СРЕДСТВАМ И СИСТЕМАМ, ЧАСТЬ 2.—
СПб., 2019. — 77 с.

Учебно-методическое пособие содержит начальные сведения о ресурсах сети интернет для физиков, работе в ОС Linux, а также о таких разделах математики и Computer Science, как оптимизация функций, машинное обучение и обработка изображений. Этот материал предназначен для изучения студентами бакалавриата по направлениям «Физика», «Радиофизика» и «Прикладные математика и физика».

Оглавление

Работа № 1

Ресурсы сети интернет для физиков

Индексы цитирования. Индексы цитирования — это базы данных (БД), в которых индексируются научные публикации, в первую очередь статьи. Хранится следующая информация о научной статье: авторы, название, аннотация статьи; название, номера тома и страниц журнала, год выхода; ключевые слова, категория (область науки), к которой относится статья; некоторая другая информация. Самое главное, хранится список имеющихся в статье ссылок на другие научные публикации, часть которых также индексирована в индексе цитирования. Благодаря этому образуется сеть публикаций, в которой они связаны ссылками друг на друга. Современные индексы цитирования имеют веб-интерфейсы, которые позволяют осуществлять поиск по публикациям. Самыми известными международными платформами, предоставляющими доступ к индексам цитирования, являются Web of Science и Scopus, российской — elibrary.ru.

Работу с индексами цитирования рассмотрим на примере платформы Web of Science. Доступ к ресурсам платформы является платным, для студентов и сотрудников СПбГУ доступ осуществляется через сайт Научной библиотеки СПбГУ: `library.spbu.ru`

1. На сайте нужно выбрать раздел “Студентам” или “Преподавателям” → в списке ссылок выбрать “Список ресурсов по типу изданий” → “Базы данных” → в списке баз данных найти “Web of Science Core Collection” и нажать на кнопку “Вход на ресурс”. После этого будет предложено ввести университетские логин и пароль. При успешной авторизации Вы окажетесь на странице поиска. Рекомендуется перед осуществлением поиска войти в систему (это позволит сохранять подборки статей и т.п.).

2. Дальше нужно выбрать в выпадающем списке базу данных, по которой будет осуществляться поиск:
   - все базы данных
   - Web of Science Core Collection — это главная БД платформы Web of Science, в которой индексируются статьи из самых престижных международных журналов.

- Russian Science Scitation Index — недавно созданная БД, в которой индексируются статьи российских журналов (часть их входит и в Core Collection). Поиск можно вести как на английском, так и на русском языках.

Выберем индекс цитирования Web of Science Core Collection.

3. В поисковой строке введем “Coulomb scattering” и в выпадающем списке выберем поле поиска “Тема” (поиск будет идти по названиям, аннотациям и ключевым словам статей). Будет найдено порядка 10000 публикаций, поэтому дальше нужно заниматься уточнением результатов.

4. В левой колонке страницы с результатами поиска есть раздел “Категории Web of Science”, в котором приведены несколько категорий (областей науки), к которым относятся больше всего найденных статей. Чтобы просмотреть все категории, перейдем по ссылке “дополнительные параметры”. В списке выберем интересующие нас категории: “Physics Nuclear”, “Physics Multidisciplinary”, “Physics Atomic Molecular Chemical”, “Chemistry Physical”, и нажмем на кнопку “Уточнить”. Это сократит количество статей примерно в два раза, но их по прежнему очень много.

5. Продолжим уточнять результаты. Предположим, нас интересует обзор современного состояния исследований по теме. Поэтому выберем в разделе “Типы документов” вариант “Review” (обзорная статья) и уточним результаты.

6. Затем в разделе “Годы публикаций” перейдем по ссылке “дополнительные параметры” $\rightarrow$ над списком годов с помощью выпадающего списка выберем “Сортировать по Алфавиту” $\rightarrow$ выберем последние несколько годов (скажем, 2014–2017) и уточним результаты. Теперь список содержит всего несколько десятков публикаций и вполне обозрим.

7. Отсортируем еще результаты поиска по цитируемости, выбрав с помощью выпадающего списка над списком публикаций “Сортировать по Количество цитирований”.

8. Теперь можно работать со списком публикаций. В списке о каждой статье приведена минимальная информация, чтобы просмотреть более подробную информацию, нужно перейти по ссылке с названием статьи. Обратите внимание на кнопку с надписью “НБ@СПбГУ” рядом с каждой найденной статьей. Если эта статья доступна по подписке СПбГУ, то при нажатии на эту кнопку будет предоставлена ссылка, по которой можно скачать полный текст статьи.

Платформа Web of Science обладает и другими полезными инструментами, кроме поиска по БД, но мы их рассматривать не будем. Отметим только EndNote, который позволяет создавать и работать с персональными подборками публикаций.

Электронные журналы. Ресурсы, через которые можно получить доступ к полным текстам статей научных журналов:

- Сайт научной библиотеки СПбГУ → раздел “Студентам” или “Преподавателям” → “Электронные журналы” → выбрать в поисковой строке в выпадающем списке “начинается с” и ввести название журнала (например, “Computer Physics Communications”). Естественно, скачивать можно будет только статьи, доступные по подписке СПбГУ.

- Сайт `arxiv.org` — бесплатный доступ к самому большому архиву электронных препринтов. Многие ученые до отправки своих статей в рецензируемые научные журналы размещают их предварительные версии (препринты) в этом архиве. Поэтому если какая-то статья недоступна для скачивания, можно попробовать поискать препринт на `arxiv.org`.

- Сайт `mathnet.ru` — российский математический портал. Предоставляет доступ к архиву статей российских математических и некоторых физических журналов. Статьи становятся доступными для скачивания через три года после выхода, а статьи некоторых журналов (например, “Успехи физических наук”) доступны сразу же после выхода.

На сайте `mathnet.ru` и на сайтах с архивами статей зарубежных журналов можно подписаться на RSS каналы, оповещающие о выходе новых статей, что очень удобно.

Электронные книги и учебники. СПбГУ имеет доступ ко многим коллекциям электронных книг. На сайте Научной библиотеки СПбГУ имеется сервис поиска книг по всем коллекциям: в разделах сайта “Студентам” или “Преподавателям” в списке ссылок нужно выбрать “Электронные книги”. Искать книги можно как с помощью поисковой строки, так и по предметной области, уточняя поиск. Например, выберем предметную область “Q — Science” и уточним поиск: в списке “предлагаемые темы” выберем “Physics”, затем “Quantum physics”, затем “Theoretical, Mathematical and Computational Physics”.

СПбГУ также имеет доступ к коллекции электронных учебников на русском языке: на сайте Научной библиотеки в разделах “Студентам” или “Преподавателям” в списке ссылок выберем “Список ресурсов по типу изданий” → “Электронные учебники” → ЭБС издательства Лань. Здесь можно найти много классических учебников и задачников по физике и математике, например знаменитый десятитомник Ландау и Лифшица по теоретической физике.

Библиотеки программ.

- CPC Program Library — база данных программ, решающих задачи физики и численных методов. Описания этих программ публикуются в журнале Computer Physics Communications. Ссылка Subject Index в левой колонке на сайте `www.cpc.cs.qub.ac.uk` позволяет искать программы по областям физики.

- `netlib.org` — репозиторий библиотек численных методов в исходных кодах, в основном на языках Fortran и C++.

- `numerical.recipes` — электронная версия книги Numerical Recipes, содержащей реализации численных методов на языках C++ и Fortran. Доступ к книге и кодам платный, но разрешается бесплатно просматривать ограниченное число страниц.

Электронные справочники.

- `physics.nist.gov/cuu` — справочник по физическим константам и системам единиц.
- `dlmf.nist.gov` — справочник по специальным функциям.

MOOK. Массовые открытые онлайн курсы — это учебные курсы, преподаваемые через интернет. Самые известные зарубежные платформы с MOOK: `coursera.org` и `edx.org`, российская: `openedu.ru`. При выборе курса из каталога прежде всего нужно смотреть на такие разделы описания курса, как "Программа курса", "Требования" ("Рекомендуемая подготовка").

1.1. Задание

С помощью индекса цитирования Web of Science Core Collection (или любого другого) попробуйте найти в литературе формулу для оценки сверху (upper-bound estimate) матричной нормы (matrix norm) матрицы, обратной (inverse) к данной матрице с диагональным преобладанием (diagonally dominant).

Работа № 2

Оптимизация функций большого числа переменных

При решении многих физических задач возникает необходимость найти глобальный минимум некоторой функции большого числа переменных. Например, это может понадобиться для подбора параметров структуры вещества по данным эксперимента. Напомним, что минимизируемая функция называется целевой функцией, а процесс поиска глобального минимума — оптимизацией. Обычные итерационные методы поиска минимума функции, такие как наискорейший (градиентный) спуск, нельзя применять для поиска глобального минимума функции с несколькими локальными минимумами (итерации сходятся к одной из точек локального минимума).

В этом легко убедиться. Рассмотрим функцию Швефеля

$$F(\bar{r}) = -\frac{1}{d}\sum_{i=1}^{d} r_i \sin\sqrt{|r_i|}. \tag{2.1}$$

Это одна из стандартных функций тестирования алгоритмов оптимизации функций многих переменных, и мы будем использовать ее в данной работе. Функция (2.1) неограниченно убывает, поэтому ограничим ее значения, положив $F(\bar{r}) = 0$ при $r_i \notin [-500, 500]$, $i = 1, \ldots, d$. Ее график в случае $d = 2$ изображен на рис. 2.1. Легко понять, что при любом числе d аргументов функции ее глобальный минимум расположен в некоторой точке прямой $r_1 = r_2 = \ldots = r_d$, а именно, это точка с координатами $r_i \approx 420,9687$, $i = 1, \ldots, d$.

Попробуем найти его в случае $d = 2$ с помощью функции fminunc из библиотеки Optimization Toolbox системы MATLAB. В этой функции реализован итерационный метод, основанный на приближении целевой функции квадратичной в окрестности текущего приближения точки минимума. Прежде всего нужно реализовать целевую функцию в виде m-файла функции costFunction.m:

```
function fval = costFunction(r)
fval = 0;
if (any(r(:) <= -500) || any(r(:) >= 500)), return, end;
fval = -sum( r(:).*sin(sqrt(abs(r(:)))) ) / length(r);
```

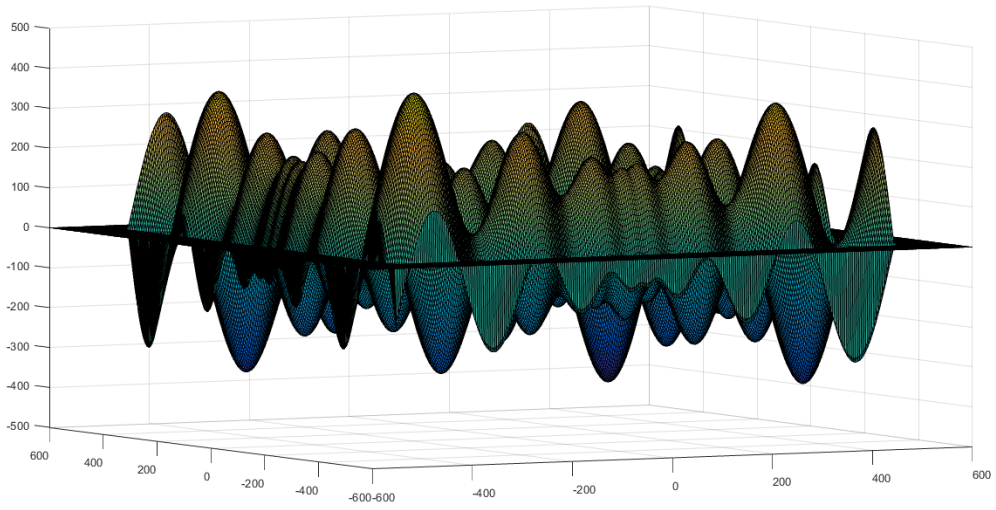


Рис. 2.1.: Функция Швефеля

Приведенный ниже сценарий позволяет задавать начальное приближение точки минимума $\bar{r}_0$ и вызывает функцию fminunc:

```
r0 = [150  200]

options = optimset('TolFun', 1e-8, 'MaxIter', 100);
[ropt, fopt, err] = fminunc(@costFunction, r0, options);

fprintf('ropt = (%10.5f,%10.5f)\n', ropt(1), ropt(2));
```

Меняя начальные приближения, легко убедиться в том, что итерации сходятся к точке одного из ближайших локальных минимумов.

Поэтому для поиска глобального минимума функции применяются специальные методы. Их можно разделить на две группы: детерминированные и стохастические (основанные на случайности). Многие из этих методов реализованы в функциях библиотеки Global Optimization Toolbox системы MATLAB. Примерами детерминированных методов являются алгоритмы, реализованные в функциях GlobalSearch и MultiStart. Эти алгоритмы основаны на следующей простой идее: запуская обычные итерационные методы с разными начальными приближениями точки минимума, определить набор точек локальных минимумов и выбрать из них точку, в которой функция минимальна. Однако для оптимизации функций большого числа переменных более эффективными оказываются стохастические методы: генетический алгоритм, метод отжига, метод роя частиц. Некоторые из них мы и рассмотрим в этой работе.

Генетический алгоритм. Кратко опишем шаги генетического алгоритма:

1. Генерируется начальная популяция особей. Особи — это точки d-мерного про-

странства, представленные в виде векторов.

2. Вычисляется пригодность каждой особи популяции. Чем меньше значение целевой функции в точке, тем выше пригодность особи.

3. Выбираются особи-родители (в соответствии с пригодностями особей популяции).

4. Часть особей-потомков создается кроссинговером. Часть генов потомка (координат точки) берется от первого родителя, оставшиеся — от второго.

5. Оставшаяся часть потомков создается мутацией особей-родителей.

6. Предыдущее поколение заменяется новым, которое состоит из лучших особей предыдущего поколения и особей-потомков.

7. Генерируются новые поколения (повторяются шаги 2–6), пока не будет достигнут критерий окончания процесса.

Генетический алгоритм реализован в виде функции ga в библиотеке Global Optimization Toolbox системы MATLAB. Для вызова этой функции удобно воспользоваться приложением optimtool с графическим пользовательским интерфейсом из библиотеки Optimization Toolbox. Для его запуска надо выбрать вкладку APPS и в группе элементов управления APPS в выпадающем списке выбрать Optimization.

В окне приложения в левом столбце выберем в выпадающем списке “Решатель” (Solver): ga, укажем в поле “Целевая функция” (Fitness function): @costFunction (дескриптор функции, реализующей функцию Швефеля) и количество переменных: 15. Остальные поля левого столбца относятся к задаче условной оптимизации, когда область изменения аргументов целевой функции ограничена набором неравенств. Мы ограничимся рассмотрением задачи безусловной оптимизации.

В правом столбце укажем:

- Количество особей в популяции (Population size): 100.

- Creation function: Constraint dependent. В этом случае способ создания начальной популяции зависит от того, является задача оптимизации безусловной или условной. В случае задачи безусловной оптимизации используется способ Uniform: особи случайным образом равномерно распределяются в области, задаваемой пунктом “Начальный отрезок”.

- Начальный отрезок (Initial range): [-500; 500].

- Selection function: Roulette. Выбор особей-родителей будет осуществляться методом рулетки: особи популяции изображаются сегментами колеса рулетки, длины сегментов пропорциональны пригодностям особей. Другой метод — турнирный отбор (Tournament) — заключается в том, что особи случайным образом отбираются в группу из нескольких особей, особью-родителем становится самая пригодная особь в группе. Этот процесс повторяется нужное количество раз.

- Количество "элитных" особей (Elite count): оставим значение по умолчанию 0.05*PopulationSize. 5% лучших особей популяции перейдут в новую популяцию.

- Доля особей-потомков, создаваемых кроссинговером (Crossover fraction): оставим значение по умолчанию 0.8. 80% потомков будут созданы кроссинговером, оставшиеся 20% — мутацией.

- Mutation function: Constraint dependent, Crossover function: Constraint dependent. Выбор функций, осуществляющих мутацию особи и кроссинговер особей, будет зависеть от того, является задача оптимизации безусловной или условной. В случае безусловной оптимизации мутация особи осуществляется функцией Gaussian — добавлением гауссовского шума к координатам вектора, а кроссинговер функцией Scattered — случайно выбранные гены (координаты точки) берутся от первого родителя, остальные от второго.

- Количество поколений (Generations): 1000. Это один из пунктов раздела "Критерии остановки алгоритма" (Stopping criteria). Остановка алгоритма произойдет раньше 1000-го поколения, если в течение количества поколений, указанного в пункте Stall generations, среднее изменение значений целевой функции окажется меньше величины, указанной в пункте Function tolerance.

- В разделе Plot functions: Best fitness, Best individual, Distance. В процессе выполнения функции ga будут выводиться графики значений целевой функции и среднего расстояния между особями в зависимости от числа поколений, а также график значений координат лучшей на текущий момент особи.

Далее нажимаем на кнопку "Start". По окончанию выполнения алгоритма будут выведены координаты точки минимума и минимальное значение целевой функции.

Метод отжига. Это стохастический алгоритм, имитирующий технологический процесс отжига (нагрев металла до определенной температуры с последующим медленным остыванием, в результате которого пропадают дефекты структуры металла). На каждом шаге алгоритма совершается шаг в сторону от точки текущего приближения точки минимума, в результате чего определяется новая пробная точка. Если значение целевой функции в этой новой точке меньше, чем в точке текущего приближения, она становится новой точкой текущего приближения. Если значение больше, пробная точка все же может стать точкой текущего приближения, но с некоторой вероятностью, зависящей от "температуры" T — параметра алгоритма, значение которого постепенно уменьшается. Вероятность может определяться, например, выражением

$$e^{-\Delta/T}$$

(распределение Больцмана). Здесь Δ — разница значений целевой функции в пробной точке и точке текущего приближения. Чем меньше эта разница и больше температура, тем больше вероятность. Ненулевая вероятность перехода в точку с бОльшим значением целевой функции обеспечивает возможность "скачков" между "по-

тенциальными ямами" локальных минимумов. С понижением температуры вероятность таких "скачков" уменьшается.

Метод отжига реализован в виде функции simulannealbnd библиотеки Global Optimization Toolbox системы MATLAB. В левом столбце окна приложения Optimization Toolbox в выпадающем списке "Решатель" (Solver) выберем simulannealbnd. В поле "Целевая функция" (Fitness function) введем @costFunction, в поле "Начальная точка" (Start point) укажем: zeros(1, 5) (начало координат пространства размерности 5).

В правом столбце укажем:

- Annealing function: Fast annealing. Этот пункт задает способ выбора новой пробной точки. Она выбирается в случайном направлении, длина шага определяется текущим значением температуры при выборе Fast annealing и корнем из температуры при выборе Boltzmann annealing.

- Temperature update function: Exponential temperature update. Задает зависимость температуры от номера итерации k_i, связанного с i-й координатой, в виде $T_i = T_{0i} \cdot 0.95^{k_i}$. Здесь T_i — температура, связанная с координатой i.

- Reannealing interval: укажем значение 20. После каждого 20-го перехода в новую точку, номера текущих итераций k_i будут уменьшены, в результате чего увеличатся зависящие от них температуры.

- Начальные температуры (Initial temperature) T_{0i} возьмем равными 150.

- Acceptance probability function: Simulated annealing acceptance. Устанавливает вероятность принятия пробной точки в качестве новой точки текущего приближения равной
$$1/\left(1 + e^{\Delta/\max_i T_i}\right).$$

Далее нажимаем на кнопку "Start". По окончанию выполнения алгоритма будут выведены координаты точки минимума и минимальное значение целевой функции.

Описание задачи и параметры метода можно сохранить в структуре MATLAB. Для этого нужно в строке меню приложения optimtool выбрать пункт File, затем Export to Workspace..., поставить галочку напротив пункта Export problem and options to a MATLAB structure named: optimproblem и нажать OK. Теперь для запуска алгоритма достаточно выполнить команду

```
simulannealbnd(optimproblem)
```

2.1. Задания

В задачах требуется подобрать параметры алгоритма так, чтобы процесс сходился к точке глобального минимума 3-4 раза из 5.

1. Подберите параметры генетического алгоритма для нахождения глобального минимума функции

$$-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} r_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi r_i)\right) + 20 + \exp(1)$$

при $d = 100$. Генерируйте начальную популяцию на отрезке $[-5; 5]$, используйте не более 1000 поколений, количество особей в популяции 100. Использовать параметры условной оптимизации (Constraints) и задавать начальную популяцию вручную нельзя. Выводите на экран графики Best fitness, Best individual, Distance. Ответом считаются .m-файл функция costFunction1.m с целевой функцией и файл optimproblem1.mat, содержащий структуру optimproblem1 с описанием задачи и подобранными параметрами генетического алгоритма.

2. Подберите параметры метода отжига для нахождения глобального минимума функции

$$10d + \sum_{i=1}^{d}\left(r_i^2 - 10\cos(2\pi r_i)\right)$$

при $d = 2$. Используйте в качестве начальной точку (-5 -5), максимальное число итераций 300, параметры условной оптимизации (Constraints) использовать нельзя. Выводите на экран графики Best function value, Best point, Temperature plot, Current point. Ответом считаются .m-файл функция costFunction2.m с целевой функцией и файл optimproblem2.mat, содержащий структуру optimproblem2 с описанием задачи и подобранными параметрами метода отжига.

3. Подберите параметры генетического алгоритма для нахождения глобального минимума функции

$$100\sqrt{|r_2 - 0.01r_1^2|} + |r_1 + 10|.$$

Генерируйте начальную популяцию на отрезке $[-15\ \ -1.5; -5\ \ 1.5]$, используйте не более 1000 поколений, количество особей в популяции 100. Использовать параметры условной оптимизации (Constraints) и задавать начальную популяцию вручную нельзя. Выводите на экран графики Best fitness, Best individual, Distance. Ответом считаются .m-файл функция costFunction3.m с целевой функцией и файл optimproblem3.mat, содержащий структуру optimproblem3 с описанием задачи и подобранными параметрами генетического алгоритма.

Работа № 3

Задача классификации

Эта и несколько следующих работ посвящены темам, которые сейчас включают в раздел Computer Science, который называется “Машинное обучение”.

Немного теории. Начнем с простого примера задачи классификации: пусть на плоскости расположено некоторое количество крестиков и ноликов (рис. 3.1). Добавим точку с координатами (x_1, x_2). Требуется определить, будет это крестик или нолик. В терминах задачи классификации вещественные координаты (x_1, x_2) называются признаками (их может быть любое число), крестики и нолики обозначают классы $\{0, 1\}$. Таким образом, решением задачи классификации называется отображение

$$(x_1, x_2) \to y \in \{0, 1\}, \tag{3.1}$$

которое по набору признаков определяет принадлежность одному из двух классов. Задача классификации похожа на задачу аппроксимации функции, только функция принимает не все возможные вещественные значения, а значения из дискретного набора $\{0, 1\}$. Отображение (3.1) строится с помощью веществennозначной функции

$$F(\bar{x}) = g\left(\sum_j c_j \varphi_j(\bar{x})\right), \tag{3.2}$$

где $\bar{x}$ - вектор признаков, суммирование ведется по заданному набору функций φ_j (например, $\varphi_j(\bar{x}) = x_j$), коэффициенты c_j требуется определить (они называются моделью, а процесс их определения — обучением модели). Функция

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

называется сигмоида, ее график представлен на рис. (3.2). Чтобы определить отображение (3.1) условимся, что если $F(\bar{x}) \geq 1/2$ ($< 1/2$), то набор признаков $\bar{x}$ определяет принадлежность классу 1 (0). Значение F можно интерпретировать как вероятность принадлежности классу 1. Величина $1/2$ называется порогом, который может быть любым вещественным числом из интервала $[0, 1]$ (чем выше порог, тем больше должно быть значение вероятности F для установления факта принадлежности классу 1). Если порог выбран равным $1/2$, то из графика сигмоиды легко видеть, что условия $F(\bar{x}) \geq 1/2$ ($< 1/2$) эквивалентны условиям $\sum_j c_j \varphi_j(\bar{x}) \geq 0$ (< 0). Поэтому уравнение

$$\sum_j c_j \varphi_j(\bar{x}) = 0 \tag{3.4}$$

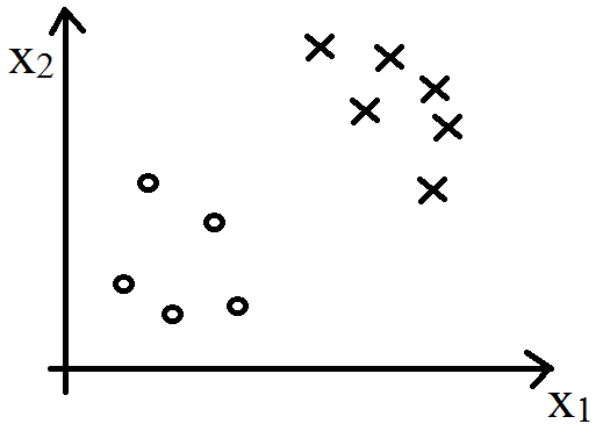


Рис. 3.1.: Пример задачи классификации

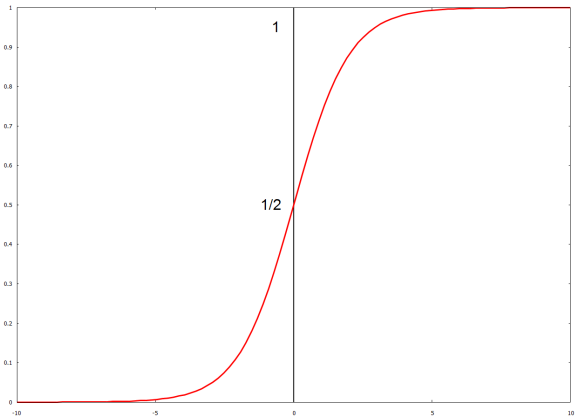


Рис. 3.2.: Сигмоида

определяет гиперповерхность в пространстве признаков, которая называется границей решений и разделяет точки, принадлежащие разным классам. В нашем примере с крестиками и ноликами при выборе $\varphi_j(\bar{x}) = x_j$ граница решений является прямой (см. рис. 3.3).

Как обучить модель, т.е. найти набор коэффициентов c_j? Для этого нужно иметь какое-то количество m примеров

$$(\bar{x}^{(i)}, y^{(i)}), \quad 1 \leq i \leq m$$

(наборы признаков, про которые известна принадлежность их одному из классов). Дальше можно было бы действовать по методу наименьших квадратов (МНК): ввести ценовую функцию

$$J(\bar{c}) = \sum_{i=1}^{m} \left(F(\bar{x}^{(i)}) - y^{(i)}\right)^2 \tag{3.5}$$

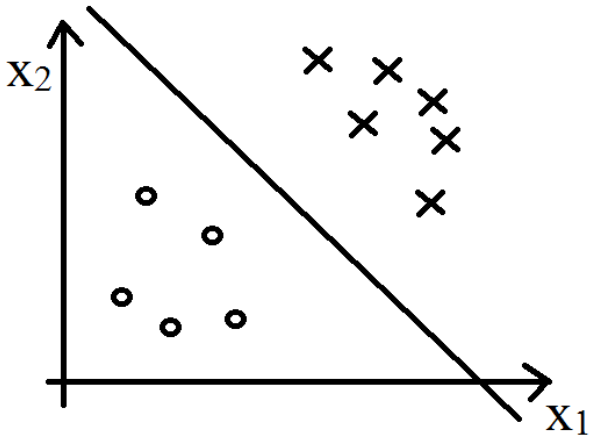


Рис. 3.3.: Граница решений

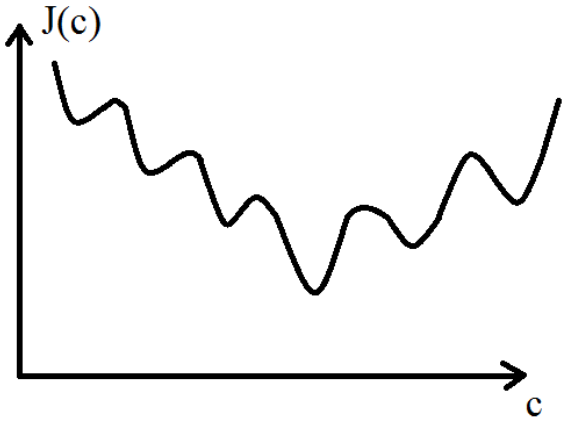


Рис. 3.4.: Пример невыпуклой ценовой функции

и минимизацией ее по вектору коэффициентов $\bar{c}$ определить модель. Недостаток подхода МНК в данном случае состоит в том, что ценовая функция (3.5) не является выпуклой, что затрудняет поиск ее глобального минимума (в качестве иллюстрации см. рис. 3.4). Поэтому минимизируется другая ценовая функция

$$J(\bar{c}) = -\sum_{i=1}^{m} \left[y^{(i)} \log F(\bar{x}^{(i)}) + (1 - y^{(i)}) \log(1 - F(\bar{x}^{(i)})) \right]. \tag{3.6}$$

Можно показать, что она является выпуклой. Графики функций $-\log z$ и $-\log(1-z)$ приведены на рис. 3.5. Из них видно, что для минимизации функции (3.6) надо, чтобы при $y^{(i)} = 1$ (0) значение $F(\bar{x}^{(i)})$ стремилось к 1 (0), чего мы и добиваемся. Минимизировать функцию (3.6) можно одним из хорошо известных методов минимизации, например, методом наискорейшего (градиентного) спуска. На каждой итерации этого метода обновление вектора коэффициентов осуществляется по

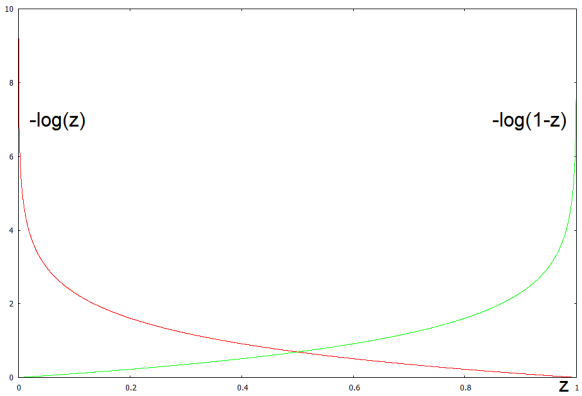


Рис. 3.5.: Графики функций $-\log z$ и $-\log(1-z)$

правилу

$$c_j \leftarrow c_j - \alpha \frac{\partial}{\partial c_j} J(\bar{c}), \tag{3.7}$$

где $\alpha \in (0, 1]$ — константа метода наискорейшего спуска, которая позволяет подбирать оптимальный шаг. Легко проверить, что в случае функции (3.6) имеем

$$\frac{\partial}{\partial c_j} J(\bar{c}) = \sum_i \varphi_k(\bar{x}^{(i)}) \left(F(\bar{x}^{(i)}) - y^{(i)} \right). \tag{3.8}$$

До сих пор мы предполагали существование только двух классов $\{0, 1\}$. Классов может быть и больше, тогда говорят о задаче мультиклассовой классификации. Простейшим способом решения такой задачи путем сведения ее к уже рассмотренной задаче с двумя классами является метод 1-vs-all. Классы делятся на два типа: один выделенный класс 0 и все остальные:

$$y \in \left\{ 0, \underbrace{1, 2, \ldots, n}_{1} \right\}$$

После этого решается задача классификации с двумя выделенными типами. Затем объединенные ранее классы снова делятся на два типа: класс 1 и все оставшиеся $2, 3, \ldots, n$, решается задача классификации с двумя новыми типами и т.д.

Какое количество функций φ_j (тем самым коэффициентов модели) надо брать? Довольно очевидно, что выбор слишком малого количества функций приведет к плохим результатам, если граница решений нелинейна (эта ситуация называется “недообучение” и проиллюстрирована на рис. 3.6). Возможно, менее очевидно, что и выбор слишком большого количества функций приводит к плохой ситуации, которая называется “переобучение”. Дело в том, что в этом случае граница решений полученной модели слишком хорошо описывает примеры, по которым производилось обучение, и плохо обобщается на новые примеры, см. рис. 3.7.

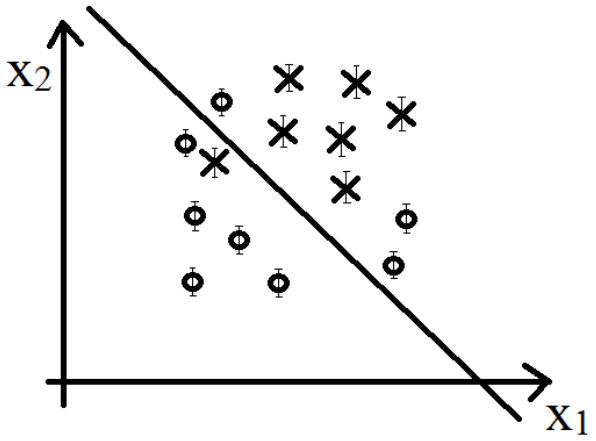


Рис. 3.6.: Недообучение. Изображены погрешности, с которыми известны данные. Граница решений недостаточно хорошо разделяет классы.

Практическая часть. Изложенной выше теории достаточно, чтобы реализовать на одном из языков программирования свой собственный простейший классификатор. Но во многих случаях лучше воспользоваться библиотеками программ, в которых реализованы более сложные современные методы решения задачи классификации. Мы воспользуемся библиотекой Statistics and Machine Learning Toolbox системы MATLAB. Эта библиотека содержит приложение Classification Learner с графическим пользовательским интерфейсом.

1. Для запуска Classification Learner надо выбрать вкладку APPS и в группе элементов управления APPS в выпадающем списке выбрать Classification Learner.

2. Загрузим данные, набрав в командной строке MATLAB

```
load ionosphere
```

Это данные с фазированной антенной решетки с излучающими элементами, меняющей направление излучения за счет изменения амплитуд и фаз токов на элементах. Двумерные массивы X и Y содержат 351 пример, которые будем использовать для обучения модели. Удобно объединить их содержимое в таблицу:

```
ionosphere=array2table(X);
ionosphere.Signal=Y;
```

Каждая строка таблицы ionosphere содержит 34 признака, задающих направление излучения антенны. Последний столбец определяет принадлежность одному из двух классов 'g' или 'b'. 'g' означает, что сигнал отразился от препятствия и вернулся, 'b' означает, что сигнал ушел через ионосферу.

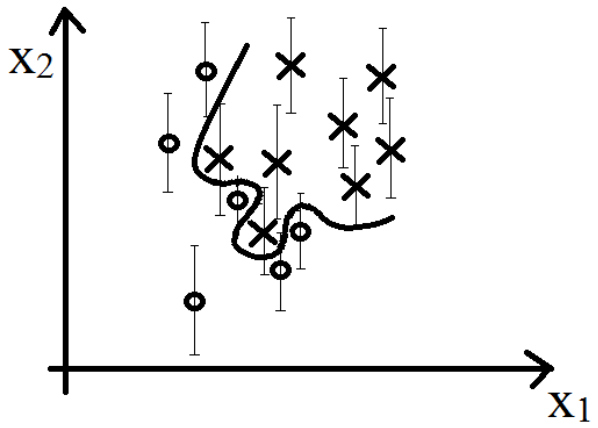


Рис. 3.7.: Переобучение. Изображены погрешности, с которыми известны данные. Граница решений имеет особенности, не соответствующие характерной величине погрешностей.

3. По имеющимся данным обучим модель, определяющую по заданному направлению в пространстве наличие объекта, отражающего сигналы. В окне приложения Classification Learner нажмем на кнопку New Session, в появившемся окне в левом столбце выберем данные — таблицу ionosphere. Средний столбец позволяет разделить их на признаки и классы (Predictor и Response).

4. В правом столбце можно выбрать метод проверки обобщающей способности (на новые примеры) обучаемой модели. Это нужно для защиты от переобучения и недообучения. Основная идея состоит в том, что имеющиеся примеры делятся на обучающую и тестовую выборки. Обучение модели производится по примерам из обучающей выборки, затем обученная модель проверяется на примерах из тестовой выборки (важно, что примеры из этой выборки в обучении модели не участвуют!). В случае выбора метода Holdout Validation просто указывается процент примеров, которые войдут в тестовую выборку. У нас количество примеров небольшое, поэтому лучше выбрать метод Cross Validation. Укажем количество групп 5. В этом случае примеры разделяются на 5 групп, и каждая по очереди становится тестовой, а обучение производится по оставшимся группам. Ошибки предсказания модели усредняются, итоговое же обучение модели производится по всем примерам.

   Нажмем на кнопку Start Session.

5. В окне Classification Learner в группе CLASSIFIER в выпадающем списке надо выбрать алгоритм классификации: логистическая регрессия (описана в теоретической части), дискриминантный анализ, SVM (Support Vector Machines, более сложный и требовательный к ресурсам алгоритм) и др.

6. Для обучения модели надо нажать на кнопку Train в группе TRAINING. После этого можно выбрать другой алгоритм классификации и обучить новую

модель. Лучше использовать несколько алгоритмов. Список обученных моделей отображается в левом окне Data Browser в выпадающем списке History. Теперь нужно выбрать лучшую модель из полученных.

7. Рядом с каждой моделью отображается процент точных предсказаний модели на тестовой выборке. Означает ли высокий процент точных предсказаний высокое качество модели? Следует четко понимать, что нет, не означает! Приведем такой пример: пусть в задаче классификации с двумя классами $y \in \{0, 1\}$ только в 1% случаев $y = 1$. Тогда, очевидно, выгодно взять модель, которая по любым признакам $\bar{x}$ будет предсказывать $y = 0$. Эта очевидно плохая модель даст очень высокий процент точных предсказаний на тестовой выборке 99%!

8. Более адекватным показателем качества модели является матрица ошибок. Для ее отображения нужно выделить интересующую модель в списке и нажать на кнопку Confusion Matrix в группе PLOTS. Матрица ошибок имеет вид:

истина \ предсказано	1	0
1	TP	FN
0	FP	TN

Например, FP (False Positive) — это число примеров из тестовой выборки, принадлежащих классу 0, про которые было предсказано, что они принадлежат классу 1. По элементам матрицы ошибок вычисляются

$$\text{точность} = \frac{TP}{TP + FP}, \quad \text{полнота} = \frac{TP}{TP + FN}.$$

Точность — это процент точных предсказаний принадлежности классу 1, полнота — процент примеров, принадлежащих классу 1, правильно предсказанных моделью. Аналогично точность и полноту можно вычислить для класса 0. Заметим, что в нашем примере, в котором мы выбирали всегда $y = 0$, полнота будет в точности равна 0. Точность и полнота отображаются в MATLAB в процентах справа и снизу от матрицы ошибок (надо в списке Summarize левее матрицы ошибок выбирать элементы Per true/predicted class).

9. Еще одним способом оценить качество модели является график ROC Curve. Надо выбрать модель из списка и нажать на кнопку ROC Curve в группе PLOTS. На графике отображается взаимозависимость величин

$$FPR = \frac{FP}{FP + TN}, \quad TPR = \frac{TP}{TP + FN}.$$

Это проценты неправильно предсказанных примеров класса 0 и правильно предсказанных примеров класса 1 (полнота) соответственно. Вдоль кривой

меняется порог (в теоретической части он у нас равнялся 1/2). У оптимальной модели форма кривой близка к ⌜ (в этом случае существует порог, точно разделяющий примеры из тестовой выборки, принадлежащие разным классам). У самой плохой модели кривая близка к ／ (примеры, принадлежащие разным классам, перемешаны — любое изменение порога, приводящее к увеличению числа правильно предсказанных примеров класса 1 обязательно ведет к увеличению числа неправильно предсказанных примеров класса 0). Если кривая имеет форму ⌣ , т.е. лежит ниже кривой ／ , лучшие результаты покажет модель, предсказывающая все наоборот.

10. Чтобы определить, какие признаки больше (меньше) остальных влияют на принадлежность к разным классам или приводят к ошибкам предсказания, можно построить график Parallel Coordinates Plot: кнопка Parallel Coordinates Plot в группе PLOTS. На графике каждая линия относится к одному примеру и соединяет значения признаков этого примера, отложенные по оси ординат. Сплошные линии обозначают правильно классифицированные примеры, штрихованные — неправильно. Лучше включить нормировку значений признаков (Scaling: Standardization в левом столбце). Мало влияющие на принадлежность к классам или приводящие к ошибкам предсказания признаки можно исключить с помощью кнопки Feature Selection в группе FEATURES и обучить новую модель.

11. Наконец, когда модель уже выбрана, нужно ее сохранить. Для этого нужно выбрать Export Compact Model в выпадающем списке кнопки Export Model из группы EXPORT. Модель сохраняется в виде структуры, которую можно использовать для предсказаний по новым данным:

```
X = 2*rand(10, 34)-1;
X_tab = array2table(X);
Y = trainedClassifier.predictFcn(X_tab)
```

3.1. Задания

Для выполнения заданий этой и двух следующих работ нужно скачать по ссылке файл с данными и считать данные из него в переменные MATLAB. Для этого поместите файл в рабочую папку MATLAB. Затем в MATLAB в окне Current Folder щелкните правой кнопкой мыши по названию файла и в контекстном меню выберите пункт “Import Data...”. В появившемся окне при необходимости в группе элементов управления DELIMITERS выберите разделитель полей данных в строке (например, Comma — запятую). Поменяйте типы данных в столбцах таблицы (NUMBER, TEXT), если они были неправильно определены. Затем выделите нужные ячейки, выберите структуру данных в группе IMPORTED DATA и нажмите на кнопку Import Selection в группе IMPORT.

При решении задач классификации с помощью методов из Classification Learner даные нужно сохранить в виде таблицы (Table).

1. По ссылке

   `https://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes`

   доступны описание и файл с данными pop_failures.dat задачи классификации, в которой требуется по значениям параметров математической модели климата научиться предсказывать, будет ли моделирование успешным (1) или приведет к останову (0).

   Задачу классификации требуется решить с помощью одного из методов, доступных в приложении Classification Learner из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. Обратите внимание на то, что значения в первых двух столбцах определяют номера вычислительных экспериментов и не являются признаками. Ответом считается .mat файл, содержащий структуру climate с полученной моделью.

2. По ссылке

   `https://archive.ics.uci.edu/ml/datasets/Musk+%28Version+1%29`

   доступны описание и файл с данными clean1.data задачи классификации, в которой требуется по значениям физических параметров (межатомные расстояния) различных конформаций молекул научиться предсказывать, является ли молекула разновидностью мускуса (1) или нет (0).

   Задачу классификации требуется решить с помощью одного из методов, доступных в приложении Classification Learner из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. Обратите внимание на то, что данные в первых двух столбцах являются названиями молекулы и ее конформации и признаками не являются. Ответом считается .mat файл, содержащий структуру musk с полученной моделью.

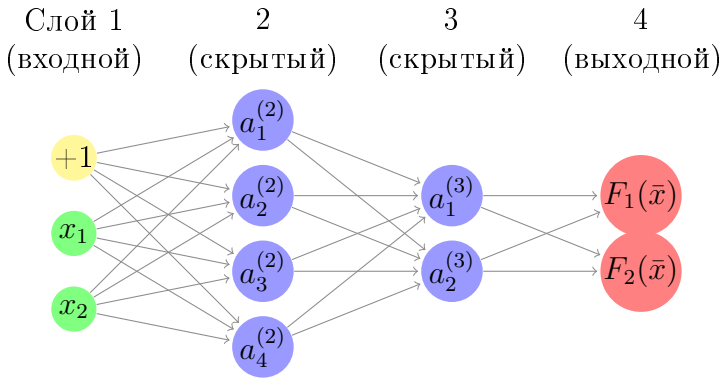
Работа № 4

Решение задачи классификации с помощью нейронных сетей

Немного теории. Граница решений в задаче классификации может быть сильно нелинейной — см. рис. 4.1. Добавление нелинейности по признакам в задачу классификации приводит к резкому увеличению количества признаков:

$$x_1^2, x_2^2, x_1x_2, x_1^3, x_1^2x_2, \ldots$$

что приводит к резкому увеличению требовательности задачи к ресурсам. Эффективным способом добавления нелинейности является использование в качестве функции $F(\bar{x})$ из предыдущей работы нейронной сети. Рассмотрим пример нейронной сети:



Нейроны изображаются кружочками, стрелочки показывают связи между нейронами. На выходе нейронов входного слоя имеем значения признаков. Величины $a_i^{(k)}$ на выходе нейронов k-го слоя называются скрытыми признаками. На входном и скрытых слоях может присутствовать специальный нейрон, который называется смещением (bias), значение на выходе которого равно $+1$. Пересчет значений признаков осуществляется со слоя на слой слева направо. Значения на выходе первого

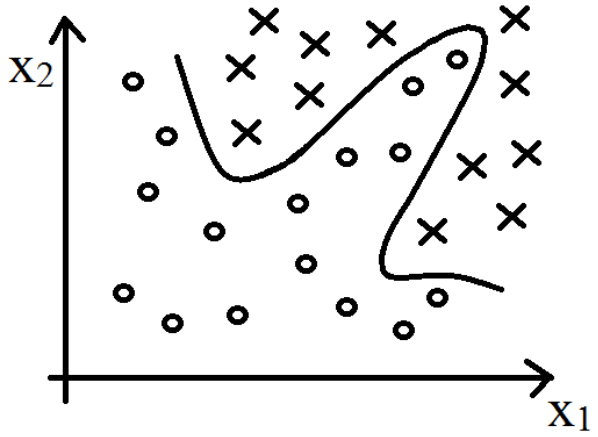


Рис. 4.1.: Задача классификации с нелинейной границей решений

скрытого слоя вычисляются по формулам:

$$a_1^{(2)} = g(c_{10}^{(1)} + \sum_{j=1}^{2} c_{1j}^{(1)} x_j), \quad a_2^{(2)} = g(c_{20}^{(1)} + \sum_{j=1}^{2} c_{2j}^{(1)} x_j), \quad \dots$$

значения на выходе следующих слоев:

$$a_i^{(k)} = g(c_{i0}^{(k-1)} + \sum_{j=1} c_{ij}^{(k-1)} a_j^{(k-1)}).$$

Говоря просто, значение на выходе одного из нейронов следующего слоя получается из значений на выходе предыдущего слоя комбинацией линейного преобразования и некоторой нелинейной функции. Эта функция называется функцией активации (i-го скрытого признака в слое k). В наших формулах это сигмоида, определенная в предыдущей работе формулой (3.2), но используются и другие функции. Коэффициенты c_{ij}^k — элементы матриц весов c^k, $k = 1, 2, \dots$ — являются параметрами модели, которые требуется определить. Коэффициенты c_{i0}^k присутствуют в формулах только тогда, когда на k-м слое есть нейрон смещения.

Рассмотрим в качестве примера такую задачу: представить в виде нейронной сети логическую функцию “исключающее ИЛИ”. Таблица истинности этой функции имеет вид:

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0

Эта задача примерно соответствует решению задачи классификации с непрерывно меняющимися признаками x_1 и x_2, изображенной на рис. 4.2. Видно, что граница

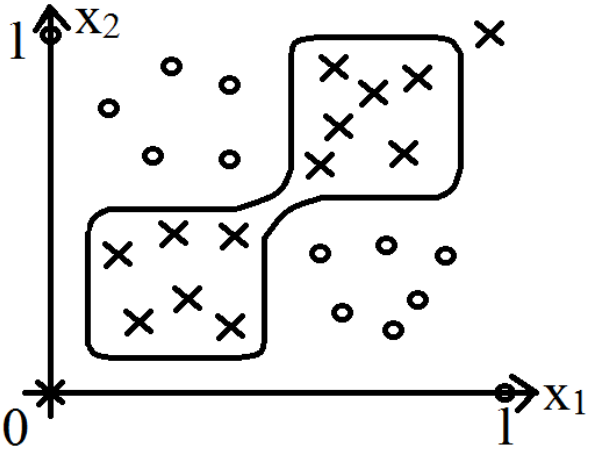
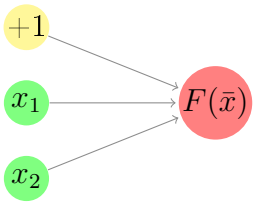


Рис. 4.2.: Задача классификации, соответствующая задаче представления “исключающего ИЛИ” в виде нейронной сети

решений является нелинейной.

Рассмотрим сначала представление логических функций “И” и “ИЛИ”. Легко понять, что границы решений в этих случаях линейны. Как мы сейчас увидим, это приводит к тому, что обе функции можно представить в виде нейронных сетей без скрытых слоев (а вот функцию “исключающее ИЛИ” нельзя). Действительно, попробуем представить их в виде нейронной сети



Это соответствует функции $F(\bar{x}) = g(d_0 + d_1 \cdot x_1 + d_2 \cdot x_2)$, коэффициенты которой надо подобрать так, чтобы при определенных x_1 и x_2 значения функции были больше $1/2$ (тогда на выходе будет значение 1) или меньше (значение 0) в соответствии с таблицами истинности. Легко видеть, что подойдут, например, коэффициенты:

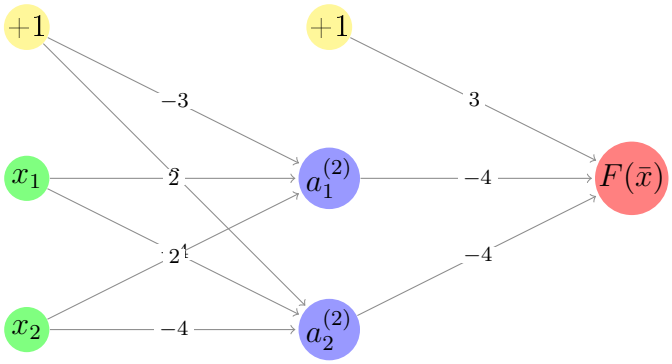
d_0	d_1	d_2	
-3	+2	+2	И
-3	+4	+4	ИЛИ
+3	-4	-4	НЕ ИЛИ

Для представления функции “исключающее ИЛИ” введем скрытый слой с двумя нейронами и возьмем такие коэффициенты пересчета с входного слоя из таблицы выше, чтобы на выходе первого из этих нейронов получилась функция “И”, а на

выходе второго “НЕ ИЛИ”. По значениям $a_1^{(2)} = \text{И}(x_1, x_2)$ и $a_2^{(2)} = \text{НЕ ИЛИ}(x_1, x_2)$ на выходе нейронов скрытого слоя, вновь используя коэффициенты пересчета для функции “НЕ ИЛИ”, на выходном слое получаем значения
$\text{НЕ ИЛИ}(a_1^{(2)}, a_2^{(2)}) = \text{исключающее ИЛИ}(x_1, x_2)$:

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$F(\bar{x})$
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

Получается такая нейронная сеть (над стрелками надписаны элементы матриц весов):



Вернемся к задаче классификации. Для обучения модели требуется иметь m примеров $(\bar{x}^{(i)}, \bar{y}^{(i)})$. Принадлежность классу s обозначается вектором

$$\bar{y}^{(i)T} = (0 \quad 0 \quad \dots \quad 0 \quad \underset{\substack{\uparrow \\ s}}{1} \quad 0 \quad \dots \quad 0).$$

Величины на выходе нейронов выходного слоя являются значениями функции $\bar{F}(\bar{x}) \in \mathbb{R}^n$. Мы хотим подобрать такие элементы матриц весов $c_{ij}^{(k)}$, чтобы в случае принадлежности i-го примера классу s вектор $\bar{F}(\bar{x}^{(i)})$ был как можно ближе к вектору $\bar{y}^{(i)}$. Это достигается минимизацией ценовой функции

$$J(\bar{c}) = -\sum_{i=1}^{m}\sum_{s=1}^{n}\left\{y_s^i \log F_s(\bar{x}^{(i)}) + (1 - y_s^{(i)}) \log(1 - F_s(\bar{x}^{(i)}))\right\}. \tag{4.1}$$

Здесь за вектор $\bar{c}$ обозначена совокупность всех элементов всех матриц весов. Вычисление производных $\partial J(\bar{c})/\partial c_{ij}^k$ для применения метода наискорейшего спуска приводит к формулам метода обратного распространения ошибки, которые мы здесь выписывать не будем.

Практическая часть. Воспользуемся приложением с графическим пользовательским интерфейсом Neural Net Pattern Recognition из библиотеки Neural Network Toolbox системы MATLAB.

1. Для запуска Neural Net Pattern Recognition надо выбрать вкладку APPS и в группе элементов управления APPS в выпадающем списке нажать на кнопку Neural Net Pattern Recognition.

2. Дальше надо нажать на кнопку Next и на следующем экране на Load Example Data Set. В левом столбце выбираем Types of Glass и нажимаем на Import. Будут загружены массивы glassInputs и glassTargets, содержащие 214 примеров. По этим примерам попытаемся обучить нейронную сеть определять, является стекло оконным (вектор $(1\ 0)^T$ в массиве glassTargets) или нет (вектор $(0\ 1)^T$) по 9 признакам — характеристикам стекла (показатель преломления и содержание 8 элементов), содержащимся в glassInputs. После выбора данных нажимаем на Next.

3. На следующем экране примеры делятся на обучающую, проверочную и тестовую выборки. Примеры из обучающей выборки используются непосредственно для обучения модели. Проверочная выборка используется для проверки обобщающей способности модели в процессе обучения (тем самым косвенно влияет на параметры модели). Тестовая выборка никак не участвует в процессе обучения и используется только для оценки обобщающей способности полученной модели. Согласимся с предложенным делением на выборки 70–15–15% и нажмем Next.

4. На следующем экране предлагается выбрать конфигурацию сети, которая содержит один скрытый слой, на котором в качестве функции активации используется сигмоида — эти параметры менять нельзя. Задать можно лишь число нейронов на скрытом слое. Оставим значение по умолчанию 10, нажмем на Next и на следующем экране в левом столбце на Train для начала обучения сети.

5. После окончания обучения можно нажать на кнопки Plot Confusion и Plot ROC в правом столбце. MATLAB покажет матрицы ошибок и кривые ROC, описанные в предыдущей работе, для каждой из трех выборок (тренировочной, проверочной и тестовой) в отдельности и для всех примеров вместе. В нижнем правом квадрате каждой матрицы ошибок отображается процент ошибок предсказаний модели на данной выборке. Однако оценивать качество модели правильнее по таким характеристикам, как точность и полнота, что подробно обсуждалось в предыдущей работе.

6. Если процент ошибок на тестовой выборке оказался большим, можно изменить количество нейронов на скрытом слое, нажав на кнопку Next и на следующем экране на Adjust Network Size в левом столбце. Затем переобучить сеть и снова оценить модель.

7. Когда будет получена хорошая модель, надо нажать на Next и на следующем экране нажатием на кнопку MATLAB Matrix-Only Function заставляем MATLAB сгенерировать m-файл функцию модели, которую можно использовать для предсказаний по новым данным.

8. Можно также нажать далее на Next и на последнем экране нажать на Save Results. Нейронная сеть будет сохранена в структуре, которую также можно использовать для предсказаний по новым данным:

```
X = [1.52; 13; 4; 1.2; 72; 0.5; 8; 0; 0];
Y = sim(net, X) %или
Y = myNeuralNetworkFunction(X)
```

4.1. Задания

При решении задачи классификации с помощью нейронных сетей признаки нужно сохранить в виде матрицы (Numeric Matrix). Столбец с названиями классов нужно сохранить в виде массива Cell Array, который затем нужно преобразовать в матрицу из единичных векторов. Это можно сделать с помощью следующего кода (массив sonar содержит названия классов R и M):

```
sonar_arr = cell2mat(sonar);
sonar_y = zeros(size(sonar_arr));
sonar_y(find(sonar_arr=='R')) = 1;
sonar_y(find(sonar_arr=='M')) = 2;
sonar_y = full(ind2vec(sonar_y'));
```

1. По ссылке

   `https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope`

   доступны описание и файл с данными magic04.data задачи классификации, в которой требуется по параметрам полученных с помощью черенковского телескопа MAGIC изображений научиться определять, зафиксирован ли на изображении сигнал космических гамма-лучей высоких энергий (g, вектор [1; 0]) или фоновое излучение (h, вектор [0; 1]).

   Задачу классификации требуется решить с помощью нейронных сетей, используя библиотеку Neural Network Toolbox системы MATLAB. Ответом считается .mat файл, содержащий обученную нейронную сеть magic.

2. По ссылке

   `https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29`

доступны описание и файл с данными sonar.all-data задачи классификации, в которой требуется по энергетическим характеристикам отраженного от объекта сигнала сонара научиться определять, является ли объект металлическим цилиндром (миной, M, вектор [0; 1]) или примерно имеющей форму цилиндра скалой (R, вектор [1; 0]).

Задачу классификации требуется решить с помощью нейронных сетей, используя библиотеку Neural Network Toolbox системы MATLAB. Ответом считается .mat файл, содержащий обученную нейронную сеть sonar.

Работа № 5

Задача кластеризации. Метод K-средних

Немного теории. Поясним суть задачи кластеризации на простом примере: пусть на плоскости расположено некоторое количество точек $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$. Требуется разбить эти точки на кластеры, т.е. множества "похожих" точек. В отличие от постановки задачи классификации точки никак не помечены, т.е. не отнесены к каким-либо классам. Координаты точек называются признаками, множество m точек называется обучающей выборкой. Решением задачи классификации будем называть набор номеров кластеров $\{c^{(1)}, c^{(2)}, \ldots, c^{(m)}\}$, к которым отнесены точки.

Для определения "похожести" точек требуется ввести меру расстояния между ними $d(x^{(k)}, x^{(l)})$. В кластерном анализе используются

- евклидово расстояние $d(x^{(k)}, x^{(l)}) = \sqrt{\sum_i \left(x_i^{(k)} - x_i^{(l)}\right)^2}$.
- квадрат евклидова расстояния $d(x^{(k)}, x^{(l)}) = \sum_i \left(x_i^{(k)} - x_i^{(l)}\right)^2$
- расстояние городских кварталов ("манхэттенское расстояние") $d(x^{(k)}, x^{(l)}) = \sum_i |x_i^{(k)} - x_i^{(l)}|$ и др.

Одним из методов решения задачи классификации является метод K-средних. Он основан на подходе, в котором решением задачи классификации считаются номера кластеров, минимизирующих ценовую функцию

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \sum_{i=1}^{m} d(x^{(i)}, \mu_{c^{(i)}}). \tag{5.1}$$

Функция минимизируется по всем своим аргументам, число кластеров K считается фиксированным. Точка μ_k имеет смысл центра k-го кластера (см. рис. 5.1). Метод K-средних — это итерационный процесс минимизации ценовой функции (5.1), на каждом шаге которого функция минимизируется сначала по набору аргументов $c^{(1)}, \ldots, c^{(m)}$ (при фиксированных $\mu_1, \ldots, \mu_K$), затем наоборот по аргументам $\mu_1, \ldots, \mu_K$ при фиксированных $c^{(1)}, \ldots, c^{(m)}$. Первая минимизация соответствует поиску центра кластера, ближайшего к данной точке $x^{(i)}$, $i = 1, \ldots, m$, вторая минимизация — поиску таких координат центра кластера μ_k, $k = 1, \ldots, K$, при которых сумма расстояний от центра кластера до точек, входящих в него, минимальна.

Рис. 5.1.: Точки, отнесенные к разным кластерам, и центры этих кластеров

В случае использования в качестве меры расстояния d квадрата евклидова расстояния это приводит к следующему алгоритму:

1. Случайным образом выбрать начальные K центров кластеров.

2. Определить $c^{(i)}$, $i = 1, \ldots, m$: отнести каждую точку $x^{(i)}$ к кластеру, центр которого является ближайшим к этой точке.

3. Определить новые координаты центров кластеров μ_k, $k = 1, \ldots, K$: новая точка μ_k является "центром масс" (средним) точек, входящих в k-й кластер.

4. Если критерий остановки алгоритма не удовлетворен, вернуться к п. 2.

Практическая часть. Воспользуемся функцией kmeans из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. В этой функции реализован алгоритм, имеющий две стадии работы.

Первая стадия — это алгоритм K-средних, описанный выше. В качестве начальных K центров кластеров случайным образом выбираются точки из обучающей выборки. По умолчанию это делается с помощью простого стохастического алгоритма kmeans++, целью которого является выбор достаточно удаленных друг от друга точек. Затем выполняются другие шаги алгоритма.

На второй стадии на каждой итерации проходятся все точки обучающей выборки. Если точка отнесена к другому кластеру, сразу же производится пересчет центров кластеров. В результате выполнения второй стадии гарантированно достигается локальный минимум ценовой функции, но итерации выполняются дольше, поэтому для больших обучающих выборок эту стадию можно отменить.

Работа алгоритма останавливается, если на очередной итерации не было объектов, переместившихся из кластера в кластер. По умолчанию в качестве меры расстояния используется квадрат евклидова расстояния.

Функция kmeans имеет интерфейс
[c, mu, sumds] = kmeans(x, K, ...)
Матрица x размера $m \times n$ содержит обучающую выборку — m n-мерных векторов, расположенных по строкам; с помощью K указывается число кластеров. Остальные входные аргументы — это необязательные опции (см. ниже). Выходной вектор с длины m содержит полученные в результате выполнения итераций номера кластеров, матрица mu размера $K \times n$ — координаты центров кластеров, вектор sumds длины K — суммы расстояний от точек кластера до его центра для каждого кластера.

1. Создадим множество трехмерных точек, генерируя точки, нормально распределенные вокруг одного из 6 центров, выбираемых случайно.

```
means = [0 0 0; -0.5 -0.3 -0.7; -0.6 0.4 0.6; ...
         0.5 -0.4 0.4; 0.4 0.6 0.5; -0.5 0.5 -0.4];
stdev = 0.16; K = size(means, 1);
m = 1000; x = zeros(m, 3);
for im = 1:m
        x(im, :) = normrnd(means( randi(K), : ), stdev);
end
scatter3(x(:, 1), x(:, 2), x(:, 3))
```

   Как видно из графика, мы фактически создали 6 кластеров точек.

2. Предположим, что мы не знаем количества кластеров. Попробуем разделить точки на 4 кластера, используя метод K-средних:

```
[c, mu] = kmeans(x, 4, 'Display', 'iter');
hold on
scatter3(x(:, 1), x(:, 2), x(:, 3), [], c)
scatter3(mu(:, 1), mu(:, 2), mu(:, 3), 100, 'rx')
hold off
```

   Если Вы попробуете выполнить приведенный выше фрагмент кода несколько раз, Вы будете получать разные результаты. В частности, будет меняться сумма расстояний от точек кластеров до их центров, т.е. значение ценовой функции (5.1). Это связано с тем, что в результате выполнения алгоритма мы попадаем в разные локальные минимумы ценовой функции, в зависимости от выбора начальных центров кластеров, меняющихся от запуска к запуску. Поэтому для получения хорошего результата следует вызывать kmeans несколько раз и среди полученных моделей выбирать лучшую.

3. Задать количество запусков алгоритма можно с помощью опции “Replicates”:

```
[c, mu] = kmeans(x, 4, 'Replicates', 5, ...
                       'Display', 'final');
```

Среди полученных 5 моделей функция выбирает модель с наименьшим значением суммы квадратов и выдает ее параметры в качестве выходных аргументов. Использование опции “Replicates” при вызове функции kmeans считается хорошей практикой.

4. Как же все-таки определить оптимальное количество кластеров? Применим метод K-средних несколько раз, указав разное количество кластеров K, и построим график суммы расстояний до центров кластеров в зависимости от K:

```
kmax = 10; sumd = zeros(1, kmax);
for k = 1:kmax
    [~, ~, sumds] = kmeans(x, k, 'Replicates', 5);
    sumd(k) = sum(sumds);
end
plot(1:kmax, sumd, 'b-x')
```

Как видно из графика, сумма расстояний убывает с увеличением K, но при $K = 6$ график имеет характерный излом, после которого убывание замедляется. Этот излом указывает оптимальное количество кластеров. Это называется “метод локтя”.

5. Итак, мы построили модель с 6 кластерами:

```
[c, mu] = kmeans(x, 6, 'Replicates', 5);
```

Пусть мы имеем 10 новых точек

```
xnew = 2*rand(10, 3)-1;
```

Как определить принадлежность этих точек к кластерам, используя нашу модель? Для этого совершим одну итерацию алгоритма, указав с помощью опции “Start” найденные в результате обучения модели центры кластеров в качестве начальных центров:

```
kmeans(xnew, 6, 'MaxIter', 1, 'Start', mu)
```

Новые точки будут отнесены к ближайшим центрам кластеров.

6. Еще один способ найти оптимальное количество кластеров — использовать специальную меру близости точки к точкам из ее кластера по сравнению с точками из других кластеров, которая называется Silhouette. Она определяется следующей формулой:

$$Silhouette(x^{(i)}) = \frac{b(x^{(i)}) - a(x^{(i)})}{\max(a(x^{(i)}), b(x^{(i)}))},$$

где $a(x^{(i)})$ — среднее значение расстояний от точки $x^{(i)}$ до других точек из ее кластера, $b(x^{(i)})$ — минимум (по другим кластерам) средних значений расстояний от точки $x^{(i)}$ до точек из другого кластера. Легко видеть, что значения

$Silhouette(x^{(i)})$ принадлежат интервалу $[-1, 1]$. Чем ближе значение к 1, тем лучше точка $x^{(i)}$ отделена от других кластеров. Отрицательное значение означает то, что точку $x^{(i)}$ следовало бы отнести к другому кластеру.

Загрузим новые данные

```
clear; load kmeansdata
```

Это набор из 560 точек из $\mathbb{R}^4$, которые мы хотим разделить на неизвестное число K кластеров. Применим метод K-средних со значениями K от 2 до 5 и расстоянием городских кварталов, и для каждого значения с помощью функции silhouette построим диаграммы значений $Silhouette(x^{(i)})$ для всех точек:

```
for k = 2:5
    subplot(2, 2, k-1)
    c = kmeans(X, k, 'Distance', 'cityblock', ...
                     'Replicates', 5);
    [s, h] = silhouette(X, c, 'cityblock');
    fprintf('K = %d, average silhouette value = %f\n', ...
                     k, mean(s))
end
```

Для каждого K мы также выводим усредненное по всем точкам значение $Silhouette(x^{(i)})$. Из этих значений и графиков видно, что оптимальным значением количества кластеров является $K = 4$. В частности, из графиков для $K = 3$ и $K = 5$ видно, что имеются точки с отрицательным значением $Silhouette(x^{(i)})$, т.е. их следовало бы отнести к другим кластерам.

5.1. Задания

При решении задачи кластеризации данные нужно сохранить в виде матрицы (Numeric Matrix).

1. По ссылке

   `https://archive.ics.uci.edu/ml/datasets/Water+Treatment+Plant`

   доступны описание и файл с данными water-treatment.data задачи кластеризации, в которой требуется по данным сенсоров городского водоочистного сооружения (входные и выходные кислотность, содержание элементов, потоки воды и т.д.), полученным в разные дни, определить возможные режимы работы сооружения (например: нормальный режим, перегрузка, ...).

   Задачу кластеризации требуется решить методом K-средних, используя соответствующую функцию из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. Обратите внимание на то, что данные в первом столбце

являются датой и признаком не являются. Кроме того, имеются примеры, в которых отсутствуют значения некоторых признаков (вместо них стоят знаки вопроса). Эти примеры будут проигнорированы функцией kmeans. Ответом считается функция [c, mu] = water(watertreatment), которая по матрице примеров watertreatment (полученной из файла water-treatment.data) выдает номера кластеров и координаты их центров.

2. По ссылке

   `https://cs.joensuu.fi/sipu/datasets/yeast.txt`

   доступен файл с данными задачи кластеризации, в которой требуется по параметрам, описывающим структуры белков, определить возможные места их локализации в клетке (например: цитоплазма, мембрана, ...).

   Задачу кластеризации требуется решить методом K-средних, используя соответствующую функцию из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. Ответом считается функция [c, mu] = yeast(yeast_mat), которая по матрице примеров yeast_mat (полученной из файла yeast.txt) выдает номера кластеров и координаты их центров.

3. По ссылке

   `https://cs.joensuu.fi/sipu/datasets/glass.txt`

   доступен файл с данными задачи кластеризации, в которой требуется по характеристикам стекла (показатель преломления, содержание химических элементов), определить тип стекла (например: оконное, автомобильное, посуда). Это нужно для того, чтобы научиться распознавать стекла, оставленные на месте преступления!

   Задачу кластеризации требуется решить методом K-средних, используя соответствующую функцию из библиотеки Statistics and Machine Learning Toolbox системы MATLAB. Ответом считается функция [c, mu] = glass(glass_mat), которая по матрице примеров glass_mat (полученной из файла glass.txt) выдает номера кластеров и координаты их центров.

Работа № 6

Обработка изображений

Немного теории. Рассмотрим операции морфологического анализа изображений. Изображение будем считать набором единичных (белых) и нулевых (черных) пикселов определенного размера. Результатом операции морфологического анализа является другое (выходное) изображение того же размера. Операции определяются с помощью структурного элемента: это набор пикселов, образующих определенную фигуру. Центром структурного элемента называется его центральный пиксел. Для примеров возьмем изображение и структурный элемент, представленные на рис. 6.1. Центром этого структурного элемента будем считать правый нижний пиксел.

1. Наращивание (Dilation). В выходном изображении единичными являются те пиксели, при перемещении в которые центра (отраженного относительно своего центра) структурного элемента он пересекается с единичными пикселами исходного изображения. В некоторых определениях наращивания речь идет о самом структурном элементе, а не отраженном относительно центра, поэтому эти слова взяты в скобки. В нашем примере получаются отраженный структурный элемент и выходное изображение, представленные на рис. 6.2.

2. Эрозия (Erosion). В выходном изображении единичными являются пиксели, при перемещении в которые центра структурного элемента он всеми своими пикселами пересекается с единичными пикселами исходного изображения. В нашем случае получим изображение, представленное на рис. 6.3. Эрозия удаляет с изображения малые объекты, неровности границ и разъединяет объекты, соединенные тонкими линиями. Но у этой операции есть вредное свойство: остающиеся на изображении объекты уменьшаются в размере. Поэтому вводится операция размыкания.

3. Размыкание (Opening). Это эрозия, а затем наращивание с тем же структурным элементом. Размыкание делает все то же самое, что и эрозия, не уменьшая размеры объектов на изображении. Результат размыкания показан на рис. 6.4.

4. Замыкание (Closing). Это наращивание, а затем эрозия с тем же структурным элементом. Замыкание избавляет изображение от малых дыр и щелей без увеличения контуров объектов на изображении. Результат показан на рис. 6.5.

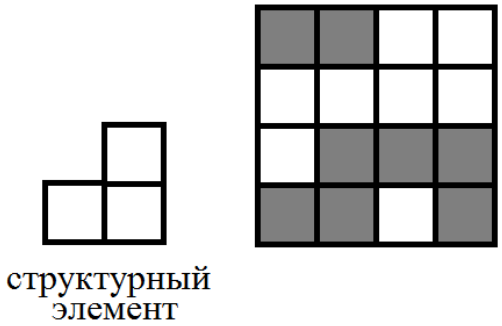


Рис. 6.1.: Структурный элемент и исходное изображение

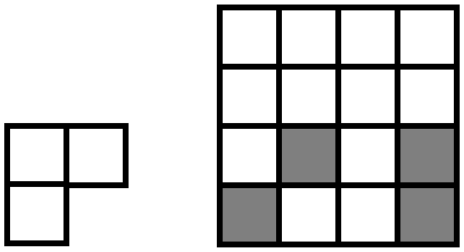


Рис. 6.2.: Результат наращивания

Важный класс задач обработки изображений — обнаружение простых геометрических фигур (прямые, окружности, эллипсы) на изображении. Они решаются с помощью преобразований Хафа. Кратко рассмотрим преобразование Хафа для прямых. Для каждого единичного пиксела с координатами (x_0, y_0) исследуемого изображения строим кривую в пространстве Хафа (r, θ), точки которой параметризуют все прямые, проходящие через точку (x_0, y_0):

$$r = x_0 \cos\theta + y_0 \sin\theta$$

(получается синусоида). Смысл параметров r и θ поясняется рисунком 6.6. Прямые на исследуемом изображении соответствуют пересечению синусоид в одной точке пространства Хафа.

Другой важный класс задач обработки изображений — выделение границ объектов на изображении. Два известных метода выделения границ:

- метод Собеля

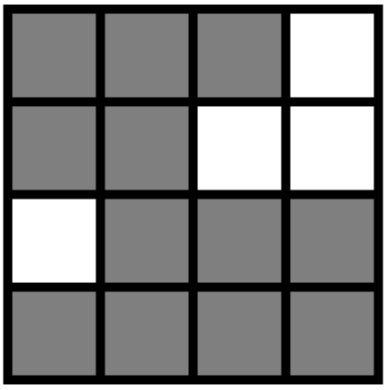


Рис. 6.3.: Результат эрозии

- метод Канни

Мы не будем их здесь подробно описывать. Оба метода основаны на поиске максимумов градиентов интенсивности пикселов изображения. Метод Канни является более эффективным, он дополнительно использует свертку с функцией Гаусса (фильтр Гаусса) для сглаживания шумов и нахождения градиентов. Также используются два порога чувствительности для определения максимумов градиентов, в результате чего определяются слабые и сильные границы; слабые границы выделяются как границы только если они соединены с сильными.

Практическая часть. Мы будем использовать для обработки изображений библиотеку Image Processing Toolbox системы MATLAB. Обработаем изображение золотых наночастиц, полученных с помощью сканирующего электронного микроскопа. Это изображение доступно для скачивания по ссылке:
`https://www.nist.gov/image/07msel018semlrjpg`
Наша конечная цель — найти на изображении наночастицы и определить их геометрические характеристики. Ниже приводится m-файл сценарий goldnano.m, который решает эту задачу. Замечание: лучше выполнять сценарии этой работы построчно, чтобы лучше разобраться в том, что происходит. Для этого проще всего выделять очередную строчку сценария и в контекстном меню нажимать на Evaluate Selection или использовать горячую клавишу F9.

```
%goldnano.m
img = imread('07msel018_sem_lr.jpg');
imshow(img)
img2 = rgb2gray(img); %переводит изображение в оттенки серого
figure; imshow(img2)
figure; imhist(img2) %гистограмма интенсивности пикселов
img2 = imadjust(img2); %делает гистограмму интенсивности более
```

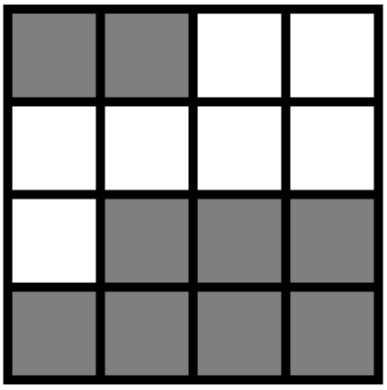


Рис. 6.4.: Результат размыкания

```
                                %равномерно распределенной, что означает
                                %увеличение контрастности
figure; imshow(img2)
figure; imhist(img2)
[img3, thresh] = edge(img2, 'sobel');
                                %выделяет границы методом Собеля
                                %выходной аргумент thresh содержит
                                %порог чувствительности определения
                                %максимума градиента
figure; imshow(img3)
[img3, thresh] = edge(img2, 'sobel', 1.2*thresh);
                                %немного увеличиваем порог
figure; imshow(img3) %изображение стало черно-белым!
img3 = imfill(img3, 'holes'); %морфологическая операция,
                                 %заполняет внутренние области
                                 %объектов, изолированные от границы
figure; imshow(img3)
img3 = imclearborder(img3); %еще одна морфологическая операция,
                  %убирает соединенные с границей части  изображения
                  %светлее окружающих их пикселей (убираем
                  %частицы, не поместившиеся целиком на изображение)
figure; imshow(img3)
%попробуем найти на изображении частицы как круги
imdistline; %помещает на изображение линейку
              %для измерения размеров объектов вручную
[centers, radii] = imfindcircles(img3, [7  15])
```

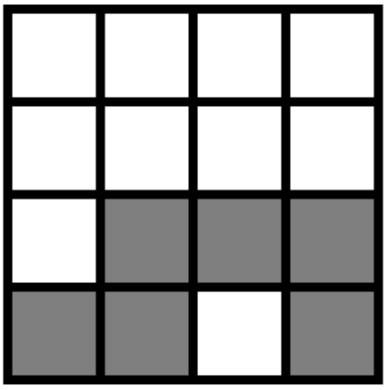


Рис. 6.5.: Результат замыкания

```
                %ищет окружности с помощью преобразования Хафа
                %указываем диапазон радиусов окружностей 7-15 пикселов
                %выходные аргументы содержат их xy-координаты и радиусы
[centers, radii] = imfindcircles(img3, [7 12], ...
                                        'Sensitivity', 0.97)
                %чтобы вытянутые наночастицы были определены как окружности
                %увеличиваем порог чувствительности детектирования окружностей
                %(по умолчанию равен 0.85)
viscircles(centers,radii); length(radii)
                  %отрисовка найденных 28 окружностей
%попробуем найти частицы как объекты произвольной формы
[B, L] = bwboundaries(img3);
                %ищет границы на черно-белом изображении
                %(bw в названии функции означает black and white)
                %B - массив (cell array) координат границ
                %каждого из объектов и дыр
                %L - матрица, в которой области, занимаемые объектами
                %и дырами, помечены их номерами
stats = regionprops(L, 'Area', 'Centroid', 'Perimeter');
                %по матрице L выдает перечисленные характеристики объектов
                %в данном случае площадь, координаты центра масс и периметр
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), '-m')
                              %отрисовка границ объектов
```

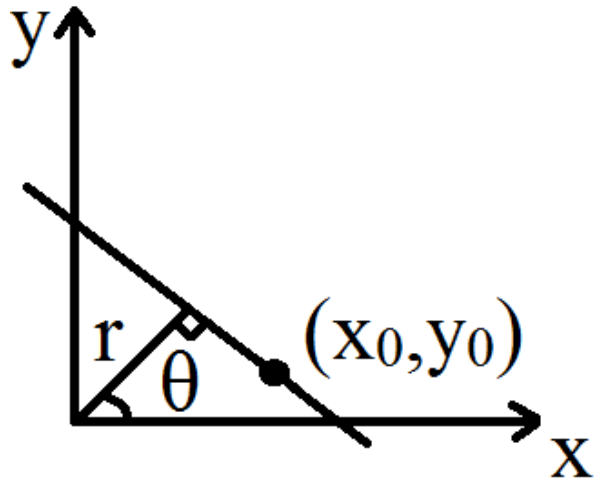


Рис. 6.6.: Параметризация прямой параметрами r и θ

```
    %-----------
    area = stats(k).Area;
    perimeter = stats(k).Perimeter;
    metric = 4*pi*area/perimeter^2;
                    %в качестве меры степени округлости объекта
                    %используем отношение площади к квадрату периметра
    text(boundary(1, 2),boundary(1, 1)+12, ...
        num2str(metric, 3), 'Color', 'y', 'FontSize', 8);
                        %выводим значение отношения рядом с объектом
end
hold off
```

Второй пример — фотография квантовых точек в клетках. Изображение доступно по ссылке
`https://www.nist.gov/image/13mml003hydrogel72nmcompositescalelrjpg`
Здесь наша цель — обвести контуры объектов, целиком помещающихся на изображение. Для этого был написан m-файл сценарий composite.m:

```
%composite.m
img = imread('13mml003_hydrogel_72nmcomposite_scale_lr.jpg');
imshow(img)
img2 = rgb2gray(img);
[~, thresh] = edge(img2, 'sobel');
img3 = edge(img2, 'sobel', 1.5*thresh);
figure; imshow(img3)
                %метод Собеля дает плохие результаты
[img3, thresh] = edge(img2, 'canny');
figure; imshow(img3)
```

```
[img3, ~] = edge(img2, 'canny', 0.8*thresh, 3.0);
figure; imshow(img3)
                %значительно лучше сложные границы выделяются
                %методом Канни
img3 = bwareaopen(img3, 3); %простая морфологическая операция,
                            %удаление маленьких объектов
                            %состоящих менее чем из 3 пикселов
figure; imshow(img3)
len = 3;
se = [strel('line', len, 90) strel('line', len, 0)];
img4 = imdilate(img3, se);
img4 = imdilate(img4, se);
img4 = imdilate(img4, se);
          %чтобы соединить части границ
          %увеличиваем границы объектов со всех сторон с помощью
          %операций морфологического наращивания
          %со структурным элементом крестом с длиной стороны 3
figure; imshow(img4)
img5 = imfill(img4, 'holes');
figure; imshow(img5)
img6 = imclearborder(img5, 4);
figure; imshow(img6)
img_borders = bwperim(img6); %оставляет на изображении
                             %только граничные пиксели объектов
                             %(т.е. единичные пиксели соседние с нулевыми)
tmp = img(:, :, 1); tmp(img_borders) = 255; img(:, :, 1) =
    tmp;
tmp = img(:, :, 2); tmp(img_borders) = 0; img(:, :, 2) = tmp;
tmp = img(:, :, 3); tmp(img_borders) = 0; img(:, :, 3) = tmp;
figure; imshow(img)
```

В качестве третьего примера используем фотографию спиральных галактик, доступную для скачивания по ссылкам
`http://hubblesite.org/image/4021/gallery`
`http://imgsrc.hubblesite.org/hvi/uploads/image_file/image_attachment/30041/STSCI-H-p-1714c-m-2000x2000.png`
Мы хотим определить наиболее часто встречающиеся размеры галактик на изображении. Эта сделано в m-файле сценарии galaxies.m:

```
%galaxies.m
img = rgb2gray(imread('STSCI-H-p-1714c-m-2000x2000.png'));
imshow(img);
imdistline; %линейкой примерно определяем размеры
            %самых больших галактик
radius_range = 1:90;
```

```
intensity_area = zeros(size(radius_range));
for radius = radius_range
    remain = imopen(img, strel('disk', radius));
    %в цикле совершаем операции морфологического размыкания
    %используя в качестве структурных элементов круги
    %радиусов от 1 до 90
    intensity_area(radius) = sum(remain(:));
    %записываем сумму оставшихся единичных пикселов
    %это площадь оставшихся на изображении галактик
end
figure; plot(intensity_area, 'm-*')
    %площадь оставшихся на изображении галактик убывает
    %при увеличении радиуса структурного элемента
    %т.к. большие структурные элементы
    %приводят к удалению большего числа галактик
intensity_area_diff = diff(intensity_area);
    %приближенная производная площади как функции радиуса
figure; semilogy(intensity_area_diff, 'c-*')
    %минимум производной при некотором радиусе
    %означает наличие большого числа галактик
    %у которых меньший из линейных размеров
    %совпадает с этим радиусом
    %т.к. структурный элемент такого радиуса
    %удаляет эти галактики с изображения при размыкании
    %например, на графике производной имеется минимум при радиусе 82
img82 = imopen(img, strel('disk', 82));
img83 = imopen(img, strel('disk', 83));
    %выходные изображения операций размыкания с радиусами 82 и 83
    %на изображении img83 пропадают галактики размера 82
img2 = imsubtract(img82, img83); %попиксельное вычитание
figure; imshow(img2, []) %галактики размера 82
```

6.1. Задания

1. Напишите .m-файл сценарий, который находит и отрисовывает на исходном изображении

   `https://www.nist.gov/image/ecolimicroscopyspotsjpg`

   границу самого большого кластера молекул, полностью помещающегося на фотографию.

2. Напишите .m-файл сценарий, который находит и отрисовывает на исходном изображении

`http://hubblesite.org/image/4053/gallery`

границу самого вытянутого объекта, полностью помещающегося на фотографию. Мерой вытянутости объекта считайте отношение квадрата его периметра к площади.

3. Напишите .m-файл сценарий, который находит и отрисовывает на исходном изображении

   `https://www.nist.gov/image/qglassjpg`

   границы круглых вкраплений в сплав. Сценарий также должен выводить в стандартный вывод количество найденных вкраплений.

Работа № 7

Основы работы и файловая система ОС Linux

Для работы на ресурсах современных вычислительных центров необходимо знание операционной системы (ОС) Linux. Как правило, пользователь ресурса получает логин и пароль, а также ip-адрес (или доменное имя) для удаленного доступа на машину. С помощью свободно распространяемого пакета программ putty можно войти на удаленную машину и обмениваться с ней файлами. Этот пакет можно использовать в ОС семейства Windows без установки: скачать архив putty.zip, распаковать в папку putty и поместить в свою домашнюю папку.

Вход на удаленную машину с помощью putty:

1. Запустите putty.exe
2. В поле Host Name введите iiivanov@xxx.cc.spbu.ru, где “iiivanov” — Ваш логин, “xxx.cc.spbu.ru” — доменное имя машины
3. Нажмите Open и при необходимости ответьте Да на вопрос о доверии хосту
4. Введите пароль. При вводе ничего отображаться не будет!

Копирование файла на удаленную машину:

1. Поместите файл в папку putty
2. Откройте эту папку в Проводнике, в адресную строку введите cmd (рис. 7.1) и нажмите Enter, в результате чего запустится командный процессор cmd.exe
3. В командном процессоре введите
   pscp.exe my_file.txt iiivanov@xxx.cc.spbu.ru:
   где «iiivanov» — Ваш логин, my_file.txt — название копируемого файла

Работа с интерпретатором команд.

1. При успешном входе на удаленную машину начинается сеанс работы, запускается интерпретатор команд (оболочка) bash (или аналогичный) и он выводит приглашение (как правило завершающееся символом $) для ввода команд. Текущим каталогом по умолчанию оказывается домашний каталог пользователя “/home/iiivanov”.

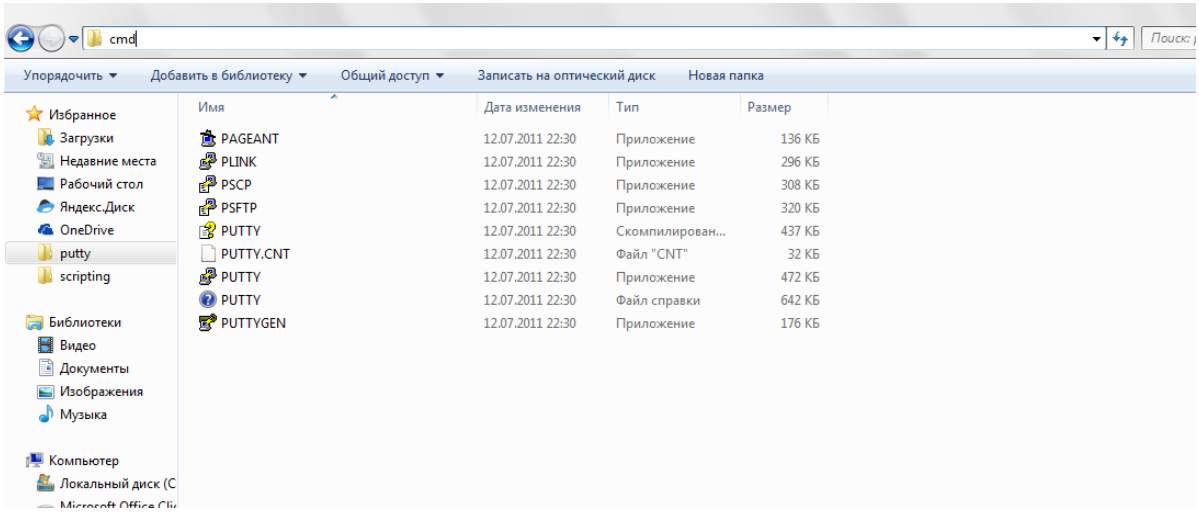


Рис. 7.1.: Запуск командного процессора cmd.exe

2. В целях безопасности рекомендуется первым делом сменить выданный пароль, для чего надо выполнить команду

```
passwd
```

   и ввести старый и новый пароли.

3. Команды могут иметь ключи и аргументы. Ключи указываются после знака “минус” (или двух знаков “минус” если ключ длинный) и изменяют режим выполнения команды:

```
ls
ls -a
ls -l
ls -al
ls -al /bin
```

   Команда ls предназначена для просмотра содержимого текущего каталога. Команда ls с ключом a показывает также и скрытые файлы (их имена начинаются с символа “.”; напомним, что файлами в ОС Linux называются обычные файлы, каталоги и некоторые другие типы файлов). Использование ключа l приводит к выводу дополнительной информации о содержимом каталога. Ключи a и l могут использоваться совместно. Указание в качестве аргумента “/bin” приводит к выводу содержимого каталога /bin.

4. Справку по команде можно получить с помощью утилиты man:

```
man ls
```

Для просмотра справки используется программа для просмотра больших текстовых файлов less. Перемещение по тексту осуществляется клавишами ↑↓ (построчное), клавишами пробел, Page Down и Page Up (постраничное), выход — клавиша q. Попробуйте найти в справке информацию о том, зачем нужны важные ключи -h и --color.

5. При работе с интерпретатором bash удобно использовать конвейеры, в которых вывод одной команды подается на ввод другой с помощью символа "|":

```
ls -l /usr/bin | less
```

   В данном примере мы просматриваем очень длинный вывод команды ls с помощью утилиты less. Конвейеры могут состоять и более чем из двух команд.

6. Интерпретатор bash позволяет перемещаться по истории команд (они записываются в специальный файл ".bash_history" в домашнем каталоге) с помощью стрелок ↑↓ на клавиатуре. Это очень удобно, т.к. позволяет вернуться к ранее введенной длинной команде и при необходимости отредактировать ее, а не вводить заново. Еще один способ извлечь команду из истории команд: нажать на клавиатуре комбинацию клавиш Ctrl + r и набрать несколько начальных символов команды, они будут дополнены до последней команды из истории команд, начинающейся с тех же символов. Попробуйте, например, нажать Ctrl + r и набрать l.

7. В интерпретаторе bash можно также пользоваться автодополнением имен файлов и имен программ (напомним, что большинство команд также являются программами, их исполняемые файлы находятся в каталоге /bin): набрать несколько начальных букв названия файла или программы и нажать клавишу Tab. Если начальные буквы можно единственным образом продолжить до полного имени, это будет сделано. Иначе ничего не произойдет, и тогда можно повторно нажать Tab и bash выведет все доступные варианты. Попробуйте, например, сделать автодополнение начальных букв pas и pass.

8. В интерпретаторе bash можно определять переменные: локальные переменные оболочки и переменные окружения. Переменные окружения, в отличие от локальных, передаются вызываемым программам и могут влиять на их работу. В именах локальных переменных оболочки принято использовать маленькие, переменных окружения — большие буквы. Пусть, например, требуется передавать в программу количество процессорных ядер, на которых она будет выполняться. Тогда определим

```
NUM_CORE=2
```

   (без пробелов!). Пока что NUM_CORE является локальной переменной оболочки. Чтобы сделать ее переменной окружения, нужно подать команду

```
export NUM_CORE
```

Можно было, конечно, сразу набрать

```
export NUM_CORE=2
```

Чтобы подставить в командную строку значение переменной, нужно написать $имя. Вывод значения переменной с помощью команды echo:

```
echo $NUM_CORE
```

9. Некоторые переменные окружения являются стандартными, например PATH

```
echo $PATH
```

Она содержит путь поиска программ — список каталогов, в которых bash ищет исполняемые файлы. Добавим в путь поиска каталог "~/myscripts" (символ "~" bash интерпретирует как значение переменной HOME, в которой хранится имя домашнего каталога):

```
export PATH=~/myscripts:$PATH
echo $PATH
```

Вывод всех переменных окружения осуществляется командой export без аргументов.

10. Заметим, что все определения и изменения переменных действительны только в течение работы текущего экземпляра интерпретатора bash. Если Вы хотите, чтобы какие-то переменные автоматически определялись или изменялись в начале каждого нового сеанса работы, нужно вписать соответствующие команды в специальный файл ".bashrc" в домашнем каталоге.

Работа с файловой системой.

1. Чтобы в домашнем каталоге появились какие-то файлы, создадим три текстовых файла. Простой способ сделать это — воспользоваться утилитой конкатенации файлов cat:

```
cat > file1
```

Дальше нужно ввести несколько строчек содержимого файла и нажать комбинацию клавиш Ctrl + d. Создайте еще два файла file2, file3. Напомним, что в ОС Linux довольно часто имена файлов не имеют расширений — пользователь сам определяет, какой программой открывать файл.

2. Команда

```
pwd
```

выводит имя текущего каталога.

3. Выполните команды:

```
mkdir my_dir
cd my_dir
mkdir new_dir
```

Мы создали каталог “my_dir”, перешли в него и создали в нем каталог “new_dir”.

4. Перейдите в каталог “new_dir” и подайте команду

```
cd ../../
```

Мы воспользовались тем, что в ОС Linux в любом каталоге есть два скрытых файла “.” и “..”: это ссылки на сам каталог и на каталог уровнем выше (каталог, в котором находится данный каталог). Т.о. в нашей команде первые две точки означают каталог “my_dir”, в котором находится ссылка (следующие две точки) на каталог уровнем выше — домашний каталог, в котором мы и оказались.

5. До сих пор в качестве аргумента мы указывали относительный путь (путь к каталогу относительно текущего каталога). Можно указывать абсолютный путь:

```
cd /home/iiivanov/my_dir
cd
```

Вместо iiivanov, конечно, нужно подставить Ваше имя пользователя. Команда cd без аргументов осуществляет переход в домашний каталог пользователя.

6. Попробуем удалить каталог “my_dir”:

```
rmdir my_dir
```

Не получилось! Команда удаления каталогов rmdir удаляет только пустые каталоги (чтобы пользователь случайно не удалил содержащиеся там файлы с результатами работы за три года — корзина в Linux не предусмотрена!).

7. Рассмотрим дальше команды копирования и перемещения файлов:

```
cp file3 my_dir
cp file3 a
mv a my_dir
```

Мы скопировали файл file3 в каталог “my_dir”, затем создали копию того же файла с именем a в текущем каталоге и переместили файл a в каталог “my_dir”.

8. Перейдите в каталог “my_dir” и подайте команды

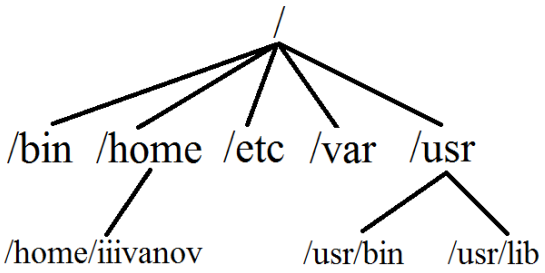


Рис. 7.2.: Дерево каталогов Linux

```
cp ../file2 .
mv a b
```

Зачем нужна последняя команда? Это способ переименовывать файлы.

9. Команда rm предназначена для удаления файлов и каталогов:

```
rm b file3
cd
rm -r my_dir
```

Так что удалять непустые каталоги все-таки можно с помощью команды rm с ключом r.

10. Команда

```
df
```

выводит размер используемого и доступного дисковых пространств для всех смонтированных файловых систем. Напомним, что в ОС Linux есть единое дерево каталогов (см. рис. 7.2), разные части которого располагаются на разных файловых системах (на разных логических разделах жестких дисков). Команду df лучше использовать с ключом h.

11. Права пользователя на совершение операций с файлами определяются атрибутами файлов, которые называются правами доступа. Они выводятся командой ls с ключом l и имеют вид

$\underbrace{-}_{\text{d}}\ \underbrace{\text{rwx}}_{\text{владелец}}\ \underbrace{\text{r-x}}_{\text{группа}}\ \underbrace{\text{r-x}}_{\text{остальные}}$

Первый символ к правам доступа не относится (он определяет, является ли файл каталогом). Буквы r, w, x означают наличие прав на чтение файла, запись в файл и исполнение файла одной из трех категорий пользователей: владелец файла, члены группы-владельца файла и все остальные пользователи. Владелец файла и группа-владелец файла являются другими атрибутами

файла; они также выводятся командой ls с ключом l. Напомним, что в ОС Linux пользователи объединяются в группы для удобства администрирования системы (например, группу students разумно максимально ограничить в правах доступа к важным файлам).

12. Владелец файла может устанавливать права доступа к файлам с помощью команды chmod:

```
chmod   u-r   file1
        g+w
        o  x
        a  s
```

Символы u, g, o, a задают категории пользователей (владелец, группа-владелец, остальные или все пользователи). Символы + и - определяют, добавляем мы или лишаем права доступа. Символы r, w, x, s определяют тип права доступа. Символ s устанавливает атрибут SUID. Этот атрибут предназначен для исполняемых файлов. Если он установлен, эффективный идентификатор пользователя процесса (см. следующую работу), порожденного выполнением этого исполняемого файла, устанавливается равным идентификатору владельца файла (а не пользователя, запустившего программу). В результате процесс будет обладать теми же правами доступа к ресурсам, что и владелец файла (эти права могут быть большими, чем права пользователя).

Например, лишим самих себя права на чтение каталога “my_dir”:

```
chmod u-r my_dir
ls -l my_dir
```

13. В ОС Linux программы-архиваторы (создают архивы) и программы-компрессоры (сжимают файлы) разделены. Однако архиватор tar имеет ключи z и j, которые позволяют одной командой создать архив “arch.tar” (ключ c) и сжать его соответственно утилитами gzip и bzip2:

```
tar -czvf arch.tar.gz file*
tar -cjvf arch.tar.bz2 file*
```

Здесь мы впервые использовали маску имен файлов “file*”. Это шаблон имен файлов, в котором символ “*” означает любой набор символов (в том числе ни одного символа). Все имена файлов, подпадающие под шаблон, будут подставлены интерпретатором bash в строку команды вместо маски. В нашем случае вместо “file*” будет подставлено “file1 file2 file3”. Эти файлы окажутся в сжатом архиве “arch.tar.gz”, в чем можно убедиться, просмотрев его содержимое или распаковав:

```
tar -tf arch.tar.gz
tar -xf arch.tar.gz
```

Для завершения сеанса работы нужно подать команду

```
logout
```

или завершить работу интерпретатора с помощью команды exit или комбинации клавиш Ctrl + d.

7.1. Задания

1. Создайте в своем домашнем каталоге каталог Data. Сделайте так, чтобы все пользователи, кроме вас, могли записывать в него файлы, но не могли бы просматривать его содержимое.

2. Напишите команду для создания при помощи утилиты tar сжатого архивного файла config.tar.gz, содержащего все файлы из каталога /etc/, имеющие расширение.

3. Изучите команду find с помощью справочной системы man, особенно полезен будет раздел EXAMPLES (или посмотрите описание и примеры в интернете). Создайте папку settings в своем домашнем каталоге и ОДНОЙ командой find скопируйте все обычные файлы из каталога /etc/, в названиях которых содержится слово bash, в папку settings.

Работа № 8

Работа с текстовыми файлами и процессы в ОС Linux

В данной работе нам понадобится файл hello.c с программой на языке С:

```
#include <stdio.h>

#define TICS 1000000000

void infty_loop(void) {
        int i = 0;
        while(1) i++;
}

void echo(void) {
        int i = 0, j = 0;
        char str[50];
        while(i++<15) {
                if(scanf("%49s",str) == 1) {
                        for(j = 0; j < TICS; j++);
                        printf("%s\n", str);
                }
        }
}

int main() {
//echo();
infty_loop();
}
```

Проще всего создать этот файл в текстовом редакторе и скопировать на машину с ОС Linux с помощью утилиты pscp.exe из пакета putty.

Утилиты для работы с текстовыми файлами.

1. Утилита конкатенации файлов cat предназначена для объединения текстовых файлов:

```
cat file1 file2 file3
```

2. Утилиты tail и head выводят 10 соответственно последних и первых строчек текстового файла:

```
head ~/.bash_history
tail -n 15 ~/.bash_history
```

   С помощью ключа n можно изменять количество выводимых строчек.

3. Перечислим далее некоторые полезные фильтры (утилиты, преобразующая некоторым образом входные данные в выходные). Команда grep осуществляет поиск в файлах по образцу:

```
grep "cpu cores" /proc/cpuinfo
```

   grep копирует в стандартный вывод те строки поданного ей на вход файла (или вывода другой команды, или стандартного ввода), которые содержат заданное выражение.

   Команда sort используется для сортировки строк файлов. Ключ n заставляет команду сортировать строки, рассматривая первое слово строки как число.

```
ls -l ~ | sort -nk5
```

   Мы отсортировали файлы из домашнего каталога по размерам (сортировка шла по 5 столбцу вывода команды ls за счет ключа k).

   Команда wc предназначена для подсчета числа строк (ключ l), слов (w) и символов (c) в текстовом файле. Команда nl нумерует строки:

```
wc -lw hello.c
nl hello.c
```

   Команда diff осуществляет построчное сравнение файлов:

```
diff file1 file2
```

Текстовый редактор vi. Открыть файл в текстовом редакторе vi можно командой

```
vi hello.c
```

После запуска редактор работает в командном режиме. В этом режиме пользователь может подавать команды. Ввод команд, начинающихся с символов “:” и “/”, должен завершаться нажатием клавиши Enter. Примеры команд:

- 10↑ — перемещение курсора на 10 строк вверх. Перемещение курсора осуществляется стрелками на клавиатуре; напечатанное перед подачей команды число задает, сколько раз будет выполнена команда.
- :10 — перемещение курсора на 10 строку файла.
- u — отмена предыдущего действия.
- yy — копирование строки, в которой находится курсор, в буфер обмена.
- 4yy — копирование строки и трех следующих за ней строк.
- dd — вырезать строку.
- p — вставка содержимого буфера обмена справа от курсора.
- Shift + p — вставка слева от курсора.
- x — вырезать, r — заменить (на введенный далее) символ под курсором
- выделение блока текста для копирования в буфер обмена: нажимаем клавишу v (или Ctrl + v если выделяем прямоугольный блок текста), перемещаем курсор, нажимаем y.
- /void — поиск первого после позиции курсора вхождения слова “void”. Чтобы находить следующие вхождения, достаточно печатать / и нажимать Enter. Если вместо символа / использовать ?, поиск будет осуществляться в обратную сторону.
- :%s/void/int/g — замена строки “void” строкой “int” во всем файле.
- Insert или i — переход в режим вставки, еще одно нажатие Insert — в режим замены.

В режимах вставки и замены можно перемещать курсор по тексту стрелками на клавиатуре и набирать текст (в режиме замены вводимые символы будут замещать символы под курсором). Возвращение в командный режим — клавиша Esc.

Для выхода из vi нужно подать одну из команд:

- :q — выход при условии, что редактируемый файл не изменился с момента последнего сохранения.
- :q! — выход с отменой всех изменений файла.
- :wq — сохранение файла и выход. NB! Не сохраняйте измененный файл “hello.c” — он нам понадобится дальше.

Работа с процессами. При запуске программы на исполнение создаются один или несколько процессов. Можно условно написать, что
процесс = программа (машинный код + данные) + среда выполнения процесса (атрибуты)
К атрибутам процесса относятся: стандартные файлы ввода, вывода и ошибок, текущий каталог, переменные окружения и др. Новый процесс может быть порожден только другим, уже работающим в системе процессом (родительский процесс). Большинство атрибутов процесса наследуются от родительского процесса. Помимо пользовательских процессов в системе работают системные процессы, которые в ОС Linux называются демонами.

Для наблюдения над процессами и управления ими предназначены утилиты ps и top. Утилита top, в отличие от ps, является интерактивной и отображает информацию о состоянии системы и процессах в реальном времени, и мы рассмотрим именно ее. При запуске

```
top
```

в нижней части экрана отображается список работающих в системе процессов, отсортированных по колонке %CPU (загрузка ядер). Этот список периодически обновляется. В нем отображается следующая информация о процессах в колонках:

- PID — уникальный идентификатор процесса.
- USER — эффективный идентификатор пользователя. Реальный и эффективный идентификаторы пользователя — это атрибуты процесса. Реальный идентификатор совпадает с идентификатором пользователя, который осуществлял запуск на исполнение файла, породивший данный процесс (или родительский процесс, по цепочке процессов породивший данный процесс). Если у этого исполняемого файла не установлен атрибут SUID, эффективный идентификатор совпадает с реальным идентификатором пользователя. Если SUID установлен, эффективный идентификатор пользователя совпадает с идентификатором владельца файла. Именно эффективный идентификатор определяет права доступа процесса к ресурсам.
- PR, NI — приоритет и относительный приоритет (nice) процесса. Меньшим значениям в колонках PR и NI соответствуют более высокие приоритеты. Процессы конкурируют между собой за предоставление им процессорного времени. Чем выше приоритет процесса, тем чаще и в большем количестве процессорное время выделяется этому процессу. Приоритет процесса является динамической величиной, на значение которой пользователь может повлиять только через относительный приоритет. Это целое число в диапазоне от -20 до 20, по умолчанию процесс имеет нормальный относительный приоритет 0. Чем выше относительный приоритет, тем выше будет и приоритет процесса. Обычный пользователь может только уменьшать относительный приоритет. Увеличивать относительный приоритет процессов может лишь суперпользователь (root, администратор машины).

- S — состояние процесса:
  - R (running) — процесс выполняется или стоит в очереди на получение кванта процессорного времени.
  - S (sleeping) — спящий. Процесс ждет окончания системной операции (ввод/вывод, истечение заданного интервала времени, завершение дочернего процесса).
  - T (stopped) — остановленный.
  - Z (zombie) — зомби (завершившийся процесс, родительский процесс которого еще выполняется, поэтому запись о нем надо хранить в таблице процессов).

- VIRT, RES, %MEM — объемы используемой процессом памяти. VIRT и RES — виртуальный размер и размер резидентной части процесса, по умолчанию выводятся в Кбайт. Они связаны формулой VIRT = SWAP + RES, где SWAP — размер части процесса, выгруженной на жесткий диск в файл свопинга. Соответственно резидентная часть процесса — часть, расположенная в оперативной памяти машины. %MEM — это отношение

$$\frac{\text{резидентная часть процесса}}{\text{общий объем оперативной памяти}}$$

  в процентах.

- %CPU — загрузка ядер. Это отношение

$$\frac{\text{время, которое процесс исполнялся на всех ядрах}}{\text{время работы одного ядра}}$$

  в процентах. Время отсчитывается с момента последнего обновления утилиты top. На многоядерных машинах %CPU может быть больше 100%!

- TIME+ — время, которое процесс исполнялся на всех ядрах с момента запуска.

В верхней части экрана отображаются строки с информацией о состоянии машины:

- Cpu(s) — средняя загрузка ядер пользовательскими процессами с нормальным относительным приоритетом (us), с пониженным относительным приоритетом (ni), системными процессами (sy).

- Mem — оперативная память,
  полный объем (total) = используемая (used) + свободная (free).

- Swap — то же для файла свопинга.

Полезные интерактивные команды:

- u, ввести имя пользователя — в списке процессов будут отображаться только процессы указанного пользователя.
- Shift + o — появится экран выбора колонки, по которой производится сортировка процессов. Нужно нажать клавишу с соответствующей буквой и нажать клавишу Esc.
- k и r — аналоги комманд kill и renice (см. ниже).
- 1 — отображать в верхней части экрана загрузку ядер по отдельности.
- q — выход.

Иногда утилиту top требуется запустить в неинтерактивном режиме. Для этого предназначен ключ b. Например, вывести список процессов пользователя iiivanov можно командой

```
top -u iiivanov -bn 1
```

Альтернативно, можно воспользоваться утилитой ps:

```
ps -u iiivanov
```

1. Скомпилируем программу hello.c с помощью компилятора gcc, который имеется на большинстве машин с ОС Linux:

   ```
   gcc hello.c
   ```

   В результате компиляции в текущем каталоге должен появиться исполняемый файл “a.out”.

2. Для запуска этого файла на исполнение нужно подать команду

   ```
   ./a.out
   ```

   Программа будет запущена в оперативном режиме. Это означает, что до завершения работы программы нельзя будет выполнять новые команды. Наша программа не завершится никогда, поскольку в ней исполняется бесконечный цикл. В такой ситуации завершим программу вручную комбинацией клавиш Ctrl + c.

3. Теперь запустим нашу программу в фоновом режиме:

   ```
   ./a.out &
   ```

   На экран будет выведен идентификатор процесса. После запуска программы в фоновом режиме можно продолжать работать с интерпретатором команд и выполнять новые команды. В том, что программа действительно исполняется, можно убедиться с помощью утилит ps или top. Программы, запущенные в фоновом режиме, продолжат исполняться и после завершения пользователем сеанса работы.

4. Однако любая попытка ввода/вывода с/на экран фоновым процессом приведет к его останову. Чтобы в этом убедиться, измените программу в файле hello.c, закомментировав вызов функции “infty_loop” и раскомментировав вызов “echo” в функции “main”. Скомпилируйте новый вариант программы. Теперь программа занимается тем, что дублирует на экране введенные пользователем фразы. Запустите ее в оперативном и фоновом режимах и убедитесь, что в последнем случае она сразу же будет остановлена.

5. Чтобы все-таки запустить программу в фоновом режиме, нужно средствами bash направить связанные с программой потоки ввода, вывода и ошибок с терминала в файлы:

```
./a.out < file1 >> file2 2>&1 &
```

   Конструкцией “2>&1” мы объединяем поток ошибок (2) и вывода(1). Теперь ввод будет осуществляться из файла file1, вывод и ошибки будут добавляться в конец файла file2 (если для направления вывода в файл использовать “>”, перед записью старое содержимое файла будет удалено).

6. Для запуска программ в фоновом режиме удобно пользоваться утилитой nohup:

```
nohup ./a.out < file1 &
```

   Эта утилита объединяет потоки вывода и ошибок и добавляет вывод в конец файла nohup.out.

7. Управление работающими процессами осуществляется с помощью сигналов. Команда

```
kill -l
```

   выводит список сигналов. Пользователь может посылать сигналы процессам с помощью команды kill. Пусть, например, требуется завершить процесс с идентификатором 1448:

```
kill -15 1448
```

   Мы послали сигнал с номером 15 SIGTERM (завершение с предварительным уведомлением). Это безопасный и наиболее предпочтительный способ завершения процессов (процесс перед завершением совершает необходимые действия — сохраняет данные и т.п.), но он не всегда работает. Нужно немного подождать, затем попробовать еще раз послать сигнал SIGTERM. Если несколько попыток не привели к успеху, можно завершить процесс с помощью сигнала с номером 9 SIGKILL (безусловное завершение):

```
kill -9 1448
```

8. Для изменения относительного приоритета процесса предназначена команда renice. Например, уменьшить относительный приоритет процесса с идентификатором 3416 до значения 15 можно командой:

```
renice 15 3416
```

Перед завершением сеанса не забудьте завершить все запущенные в процессе выполнения работы фоновые процессы!

8.1. Задания

1. Выведите на экран: идентификатор процесса, эффективный идентификатор пользователя, степень загрузки процессора, приоритет, относительный приоритет (nice), имя программы ТОЛЬКО для процессов, порожденных запуском на исполнение программ, в названия которых входит сочетание букв ssh. Воспользуйтесь утилитой grep.

2. Запустите на исполнение скрипт (≈ программа) new.sh

```
#!/bin/bash
while :
do
read line1
if [ $? -eq 0 ]
then
        echo $line1
else
        echo "No more lines ..."
fi
sleep 10
done
```

с помощью команды nohup в фоновом режиме (предварительно убедитесь, что файл new.sh является исполняемым, если нет — измените соответствующий атрибут). Ввод должен производиться из файла new.sh. Завершите работу скрипта с помощью сигнала SIGTERM и вывод скопируйте к себе на компьютер.

Работа № 9

Создание скриптов на Python

Скрипты (сценарии) — это исполняемые файлы, содержащие последовательности команд, которые интерпретируются и исполняются одна за другой. Скрипты неизбежно приходится создавать при работе на машинах современных вычислительных центров, чтобы избавить себя от многократного повторения рутинных операций по запуску задач и обработке полученных результатов. Интерпретатор команд bash ОС Linux обладает достаточно развитыми средствами программирования для создания скриптов. Однако по мнению авторов гораздо эффективнее создавать скрипты на очень популярном в настоящее время языке программирования Python. Он проще и современнее языка bash. Среда Python, в отличие от bash, обладает развитыми библиотеками для проведения численных расчетов (сравнимыми с библиотеками среды MATLAB). Кроме того, скрипты на Python при грамотном написании получаются кроссплатформенными: их можно исполнять как на машинах с ОС Linux, так и с Windows. Среда Python является свободно распространяемой и как правило имеется на машинах вычислительных центров. Для ОС Windows удачной средой является Enthought Canopy.

Рассмотрим несколько примеров скриптов. Начнем с очень простого скрипта, который по традиции назовем helloworld.py. Он вычисляет и выводит на экран гиперболический тангенс своего аргумента:

```
1 #!/usr/bin/env python
2 import sys, math
3
4 x = float(sys.argv[1])
5 v = math.tanh(x)
6 print "th(" + str(x) + ") = " + str(v)
```

Чтобы скрипт выполнился, нужно подать команду

```
python helloworld.py 4.2
```

или (на машине с ОС Linux) просто

```
./helloworld.py 4.2
```

но для этого пользователь должен иметь право на исполнение файла. Имя программы для интерпретации скрипта задано в комментарии в строке 1. В строке 2

мы импортируем модули system и math, т.е. предоставляем скрипту доступ к функциям и структурам данных этих модулей. Например, в строке 4 мы используем список argv из модуля system. Первый элемент (с индексом 0) этого списка содержит название скрипта, следующие элементы — аргументы командной строки. Язык Python является языком программирования с динамической типизацией. Это означает, что тип переменной может меняться и определяется в момент присваивания ей значения. В строке 4 перед присваиванием мы делаем явное преобразование типа (строка в вещественное число с плавающей запятой двойной точности), поскольку в следующей строке переменная x передается в функцию tanh (из модуля math), аргументом которой должно быть вещественное число. В строке 6 осуществляется конкатенация строк и вывод результата на экран.

Следующий скрипт extractLowest.py используется для поиска минимального числа из набора чисел, записанных в файл. Такой файл получается, например, на выходе программы bound.f90 (приведена в Приложении А), которая ищет уровни энергии квантовой частицы в поле потенциала (в таком случае скрипт находит энергию основного состояния частицы). Имя файла передается в качестве аргумента скрипта.

```
1  #!/usr/bin/env python
2  import sys
3
4  try:
5      outfilename = sys.argv[1]
6  except:
7      print "Usage:", sys.argv[0], "outfile"; sys.exit(1)
8
9  f = open(outfilename, 'r'); lines = f.readlines(); f.close()
10 e = []
11 for line in lines:
12     e.append(float(line))
13 if len(e) == 0:
14     print 'No bound states'
15 else:
16     print min(e)
```

В скрипте есть несколько управляющих конструкций: оператор цикла for, условный оператор if, блок try-except. Особенностью языка Python является то, что блок кода (например, тело оператора for) выделяется отступом строки. В блоке try-except (строки 4–7) обрабатывается ситуация запуска скрипта на исполнение без аргументов (попробуйте!). В строке 9 файл открывается функцией open для чтения (аргумент 'r' — чтение, 'w' — запись) и с помощью функции readlines строки файла считываются и сохраняются в списке lines. В строке 10 создаем пустой список e. Затем следует цикл for, в котором переменная line последовательно принимает значения элементов списка lines. В теле цикла каждую строку line преобразуем в вещественное число, которое затем с помощью функции append добавляем в спи-

сок e. После этого в случае непустого списка e с помощью встроенной функции min находим минимальный элемент списка и выводим его.

Аргументы еще одного скрипта shiftData.py — имена входного и выходного файлов. Этот скрипт считывает данные из входного файла с тремя столбцами значений (энергия, фазовый сдвиг, сечение рассеяния). Эти же данные записываются в выходной файл, но значения энергий сдвигаются на определенную фиксированную величину. Кроме того, скрипт находит и выдает коэффициенты прямой, аппроксимирующей зависимость сечения рассеяния от энергии.

```
#!/usr/bin/env python
import sys
import numpy as np


try:
    infilename = sys.argv[1]; outfilename = sys.argv[2];
except:
    print "Usage:", sys.argv[0], "infile outfile"; sys.exit(1)

def shift(x):
    return x+0.4997286

ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
for line in ifile:
    triple = line.split()
    e, delta, sigma = map(float, triple)
    #e = float(triple[0]); delta = float(triple[1]); sigma =
        float(triple[2])
    e_sh = shift(e)
    ofile.write('%10.8f %10.8f %10.8f\n' % (e_sh, delta,
        sigma))

ifile.close(); ofile.close()

##################################################################

ifile = open(infilename, 'r')

lines = ifile.readlines()
e = []; delta = []; sigma = []
for line in lines:
    eval_, deltaval, sigmaval = line.split()
    e.append(float(eval_)); delta.append(float(deltaval));
```

```
            sigma.append(float(sigmaval))
34  e = np.array(e); delta = np.array(delta); sigma =
        np.array(sigma)
35  a, b = np.polyfit(e, sigma, 1)
36  print 'E = %f * sigma + %f' % (a, b)
37
38  ifile.close()
```

В строках 11, 12 скрипта определяется вспомогательная функция shift, которая сдвигает свой аргумент на фиксированную величину (обратите внимание на отступ строки, выделяющий тело функции). Функция, естественно, должна определяться до использования. Обратите внимание на считывание строк входного файла прямо в заголовке цикла for в строке 16. Функция split в строке 17 разбивает строку на слова и записывает их в список triple. В строке 18 функция float поэлементно применяется к этому списку с помощью функции map; элементы преобразованного списка записываются в переменные e, delta, sigma. В строке 21 осуществляется форматированный вывод в выходной файл. Аппроксимация данных прямой линией производится функцией polyfit из пакета numpy. Этот пакет реализует расширение языка, которое называется Numerical Python и добавляет в Python многомерные массивы (array); он также содержит набор модулей и функций, реализующих некоторые численные методы (в частности, численные методы линейной алгебры в модуле numpy.linalg). В строке 34 мы создаем массивы из списков с помощью функции array, чтобы затем передать их в функцию polyfit.

Последний скрипт test.py работает с программой расчета уровней энергии квантовой частицы в поле потенциала. Эта программа на языке Фортран приведена в приложении А. Надо скопировать ее в файл bound.f90. Для создания исполняемого файла в ОС Linux понадобятся имеющиеся на многих машинах компилятор gfortran и библиотека LAPACK:

```
gfortran -o bound -llapack bound.f90
```

Входной файл parameters.inp имеет вид:

```
1  #system: mu, l
2  1.0, 0
3  #potential parameter: r0
4  2.0
5  #discretization: R, N
6  50.0, 100
```

В нем в строке 2 задаются параметры системы (масса и орбитальный момент частицы), в строке 4 параметр потенциала, в строке 6 параметры дискретизации задачи (R — размер области, в которой ищется решение, N — количество точек сетки). Скрипт test.py делает следующее: несколько раз запускает программу расчета с разными входными файлами, в которых меняет один из параметров дискретизации. Какой именно из параметров (R или N), в каком диапазоне и с каким шагом

меняется, задается в строках 8–11 скрипта. Затем скрипт обрабатывает полученные результаты: применяет к выходным файлам, в которые записаны выводы программы (наборы значений энергии), скрипт extractLowest.py, который находит минимальное значение энергии (энергию основного состояния), полученное в каждом из расчетов. Результатом работы скрипта является файл energies.dat, в который записаны пары значений “параметр дискретизации”–“энергия основного состояния”. По этим данным можно построить график и подобрать значение исследуемого параметра. Все выходные файлы записываются в каталог с названием “res#”, где # — номер запуска скрипта.

```
1  #!/usr/bin/env python
2  import os, shutil
3  import math
4  import sys
5  from subprocess import Popen, PIPE
6
7  #Task: R, N
8  task = 'N'
9  from_ = 50
10 to = 100
11 step = 10
12 #Executable
13 exe = 'bound'
14 paramfname = 'parameters.inp'
15
16 def read_param_file(filename):
17     global params
18     f = open(filename, 'r'); lines = f.readlines(); f.close();
19     pair = lines[1].split(',')
20     params['mu'] = float(pair[0]); params['l'] = int(pair[1])
21     params['r0'] = float(lines[3])
22     pair = lines[5].split(',')
23     params['R'] = float(pair[0]); params['N'] = int(pair[1])
24
25 def write_param_file(filename):
26     global params
27     f = open(filename, 'w')
28     f.write('#system: mu, l\n')
29     f.write('%.1f, %d\n' % (params['mu'], params['l']))
30     f.write('#potential parameter: r0\n')
31     f.write('%.1f\n' % params['r0'])
32     f.write('#discretization: R, N\n')
33     f.write('%.1f, %d\n' % (params['R'], params['N']))
34     f.close
```

```
35
36 params = {}
37 read_param_file(paramfname)
38
39 outdir = 'res'
40 a = 1
41 while os.path.isdir(outdir):
42     outdir = 'res' + str(a)
43     a = a+1
44 os.mkdir(outdir)
45
46 tmpdir = 'tmpdir'
47 if os.path.isdir(tmpdir):
48     shutil.rmtree(tmpdir)
49 os.mkdir(tmpdir)
50 tmp_exe = os.path.join(tmpdir, exe)
51 shutil.copy(exe, tmp_exe)
52 tmp_paramfname = os.path.join(tmpdir, paramfname)
53 shutil.copy(paramfname, tmp_paramfname)
54
55 b = 0
56 val = from_
57 while val <= to:
58     #Write new parameters.inp file
59     params[task] = val
60     write_param_file(tmp_paramfname)
61     #Execute program in tmpdir
62     os.chdir(tmpdir)
63     cmd = exe + ' 1> out.res 2> err.res'
64     failure = os.system(cmd);
65     if failure:
66         print 'Running the bound code failed:\n%s' % (cmd)
67         sys.exit(1)
68     os.chdir('..')
69     #Copy from tmpdir to outdir
70     os.mkdir(os.path.join(outdir, str(b)))
71     shutil.copy(os.path.join(tmpdir, 'out.res'),
          os.path.join(outdir, str(b)))
72     shutil.copy(os.path.join(tmpdir, 'err.res'),
          os.path.join(outdir, str(b)))
73     shutil.copy(tmp_paramfname, os.path.join(outdir, str(b)))
74     val += step
75     b+=1
76
```

```
77 shutil.rmtree(tmpdir)
78
79 #Extract lowest energies and write to file
80 f = open(os.path.join(outdir, 'energies.dat'), 'w')
81 val = from_
82 for c in range(b):
83     cmd = 'python extractLowest.py ' + os.path.join(outdir,
          str(c), 'out.res')
84     p = Popen(cmd, shell=True, stdout=PIPE)
85     output, errors = p.communicate()
86     if 'No' in output:
87         continue
88     f.write(str(val) + ' ' + output.rstrip() + '\n')
89     val += step
90 f.close()
```

В скрипте используются функции для работы с файловой системой из модулей os и shutil, такие как path.join (создает полное имя файла объединением своих аргументов), path.isdir (определяет, является ли файл каталогом), mkdir, rmtree, chdir (создание и удаление каталога, переход в каталог), copy (копирование файла), system (выполнение комманд оболочки ОС). Модуль subprocess также используется для выполнения комманд оболочки, он позволяет сохранить потоки вывода и ошибок в переменных (переменные output, errors в строке 85). В скрипте используется словарь (массив, элементы которого индексируются строками) params.

Работа № A

Программа расчета уровней энергии квантовой частицы в поле потенциала

```
program bound
implicit none

character(len = 50), parameter :: fileEF = 'eigenfunc.txt'
character(len = 50), parameter :: fileP = 'parameters.inp'
integer :: N, l
integer :: i, j, info, lwork
real, parameter :: PI = 3.1415927
real :: R, mu, h
real :: c1, c2
real :: F
real, allocatable :: A(:,:), C(:,:), alpha(:), V(:,:)
real, allocatable :: work(:), lambda(:), ilambda(:)
real, allocatable :: vl(:,:), vr(:,:)
real :: r0
common /coeff/ r0

open(24, file = fileP)
read(24, *); read(24, *) mu, l
read(24, *); read(24, *) r0
read(24, *); read(24, *) R, N
close(24)
if (N > 600) then
  N = 600
  print *, "Warning: N must be < 600, set to 600"
endif

allocate(A(N, N))
h = R/(N+1)
```

```
! fill matrices
c1 = 5.0*h*h/6.0
c2 = h*h/12.0
A(:, :) = 0.0
A(1, 1) = 2.0 + c1*F(h, l)*2*mu
A(1, 2) = -1.0 + c2*F(2*h, l)*2*mu
do i = 2, N-1
  A(i, i-1) = -1.0 + c2*F((i-1)*h, l)*2*mu
  A(i, i) = 2.0 + c1 *F(i*h, l)*2*mu
  A(i, i+1) = -1.0 + c2*F((i+1)*h, l)*2*mu
enddo
A(N, N-1) = -1.0 + c2*F((N-1)*h, l)*2*mu
A(N, N) = 2.0 + c1 *F(N*h, l)*2*mu

allocate(alpha(N), V(N, N))

do i = 1, N
  alpha(i) = 10.0 + 2*cos(i*PI/(N+1))
  do j = 1, N
    V(j,i) = sin(i*j*PI/(N+1))
  enddo
enddo
V(:,:) = sqrt(2.0/(N+1))*V(:,:)

allocate(C(N,N))
C(:, :) = 0.0
call SGEMM('N', 'N', N, N, N, 1.0, A, N, V, N, 0.0, C, N)
call SGEMM('T', 'N', N, N, N, 1.0, V, N, C, N, 0.0, A, N)
if (allocated(C)) deallocate(C)

do i = 1, N
  A(i, :) = A(i, :)/alpha(i)
enddo

allocate(lambda(N), ilambda(N), vl(1,N), vr(N,N), work(1))
call SGEEV('N', 'V', N, A, N, lambda, ilambda, &
           vl, 1, vr, N, work, -1, info)
if (info /= 0) then
  write(*, *) 'Stop. SGEEV workspace query failed'
  stop
endif
lwork = work(1)
if (allocated(work)) deallocate(work)
```

```
allocate(work(lwork))
call SGEEV('N', 'V', N, A, N, lambda, ilambda, &
           vl, 1, vr, N, work, lwork, info)
if (info /= 0) then
  write(*, *) 'Stop. SGEEV failed'
  stop
endif

if (allocated(work)) deallocate(work)
if (allocated(vl)) deallocate(vl)

lambda(:) = 0.5*lambda(:)/mu
lambda(:) = lambda(:)/c2
allocate(C(N,N))
C(:, :) = 0.0
call SGEMM('N', 'N', N, N, N, 1.0, V, N, vr, N, 0.0, C, N)
vr(:, :) = C(:, :)
if (allocated(C)) deallocate(C)
open(23, file = fileEF)
do i = 1, N
  if ( (lambda(i) < 0.0) .and. (ilambda(i) == 0.0) ) then
    write(*, '(f10.5)') lambda(i)
    do j = 1, N
      write(23, '(f10.5,f15.10)') j*h, vr(j, i)
    enddo
    write(23, *) '_____________________________________________'
    write(23, *) '_____________________________________________'
  endif
enddo
close(23)
if (allocated(lambda)) deallocate(lambda)
if (allocated(ilambda)) deallocate(ilambda)
if (allocated(vr)) deallocate(vr)

if (allocated(V)) deallocate(V)
if (allocated(alpha)) deallocate(alpha)
if (allocated(A)) deallocate(A)

end program bound

real function F(r, l)
implicit none

integer, intent(IN) :: l
```

```
real, intent(IN) :: r

F = V(r) + l*(l+1)/(r*r)

contains
  real function V(r)
  implicit none
  real, intent(IN) :: r
  real, parameter :: nu = 2.01d0
  real :: r0
  common /coeff/ r0

V = -1.0d0*( 1 - 2.d0/( 1.d0 + exp( (r/r0)**nu/(1.d0) ) ) )/r

  end function V

end function F
```

Литература

[1] Материалы курса Стэнфордского университета “Машинное обучение” [online]. URL: `www.coursera.org/course/ml`.

[2] А. Комолкин С. Немнюгин, М. Чаунин. *Эффективная работа: UNIX*. Питер, 2003.

[3] *Global Optimization Toolbox User's Guide*.

[4] H. P. Langtangen. *Python Scripting for Computational Science*. Springer, 3 edition, 2009.