

Санкт-Петербургский государственный университет

А. Н. Пепелышев

**Статистические вычисления
в пакете MATLAB**

У ч е б н о е п о с о б и е

Санкт-Петербург
2006

УДК 519.24
ББК 22.18
П25

Рецензенты: проф. М. К. Чирков (С.-Петербург. гос. ун-т),
доц. Н. Э. Голяндина (С.-Петербург. гос. ун-т)

*Печатается по постановлению
Редакционно-издательского совета
С.-Петербургского государственного университета*

Пепелышев А. Н.

П25 Статистические вычисления в пакете MATLAB: Учеб.
пособие. — СПб., 2006. — 64 с.

В учебном пособии изложены основы математического пакета MATLAB и ряд статистических задач с алгоритмами их решения. В основу пособия положен курс лекций, читаемых на математико-механическом факультете СПбГУ.

Пособие рассчитано на студентов и аспирантов, специализирующихся в области математического моделирования.

ББК 22.18

© А. Н. Пепелышев, 2006

© С.-Петербургский
государственный
университет, 2006

Предисловие

Учебное пособие посвящено исследованию ряда статистических задач в математическом пакете MATLAB (версия 6.x).

С развитием науки математику при исследовании задачи все чаще приходится обращаться к компьютеру. Компьютер может использоваться для двух целей: либо для проверки некоторого утверждения на ряде примеров, либо для получения гипотезы, основанной на большом количестве численных примеров. Математический пакет MATLAB позволяет записывать алгоритмы, связанные с техническими вычислениями, в несколько раз быстрее, чем при написании программ в других средах. Язык MATLAB является простым для изучения и применения. MATLAB основан на принципе непосредственного исполнения и ориентирован на работу с матрицами, что делает его удобным и естественным инструментом обработки данных.

Целью учебного пособия является познакомить читателя с использованием MATLAB на ряде примеров решения статистических задач для того, чтобы он мог эффективно исследовать свою научную задачу.

В основу пособия положен курс лекций, читаемых на математико-механическом факультете СПбГУ.

Пособие состоит из двух разделов. В первом разделе излагаются основы программирования в пакете MATLAB. Во втором разделе кратко формулируется ряд статистических задач и обсуждается их реализация в пакете MATLAB. В пособии рассматриваются такие задачи, как вычисление стационарного распределения марковской цепи, вычисление интеграла методом Монте-Карло, имитационное моделирование, нахождение оценок максимального правдоподобия, спектральный анализ и фильтрация, анализ временных рядов методом «Гусеница»-SSA, построение оптимальных планов регрессионных экспериментов, задача классификации изображений и построение нейронных сетей.

1. Основы программирования в MATLAB

Математический пакет MATLAB — это интерактивная система, в которой основным элементом данных является матрица. Это позволяет записывать алгоритмы, связанные с техническими вычислениями, в несколько раз быстрее, чем при написании программ на C или Pascal. Слово MATLAB произошло от matrix laboratory. В MATLAB важная роль отводится специализированным наборам функций, называемых *toolboxes*. Они позволяют решать частные классы задач, в частности, статистической обработки данных, решения дифференциальных уравнений, анализа нейронных сетей, вейвлетов, нечеткой логики, обработки сигналов и т.д.

Основное окно программы MATLAB содержит несколько под-окон:

- командное окно,
- окно со списком предыдущих команд,
- окно для выбора рабочей папки,
- окно с переменными из рабочего пространства.

В командном окне знак `>` означает приглашение ввести новую команду. При нажатии на Enter команда начинает выполняться и ее результат выводится на экран. Если в конце команды поставить точку с запятой, то результат не будет выведен. Клавиша `↑` используется для перехода к предыдущим командам. Если набрать несколько символов, то клавиша `↑` найдет предыдущую команду, которая начинается с этих символов.

Командное окно можно использовать как обычный калькулятор, так и для вызова сценариев (файла со списком команд, которые последовательно выполняются).

Команда `help` — это самый основной способ определения синтаксиса и поведения отдельных функций. Для вывода помощи по некоторой функции надо набрать `help` с именем функции. Команда `helpwin` открывает окно помощи. Команда `lookfor` позволяет искать функции по заданному ключевому слову.

Рабочее пространство — это область памяти, доступная из командной строки и сценариев. Здесь хранятся переменные, созданные в текущем сеансе. Команда `save` сохраняет содержание рабочего пространства в файл с расширением `mat`, который может быть прочитан командой `load` в последующих сеансах работы MATLAB.

По умолчанию данные сохраняются в двоичном формате. Если вы хотите использовать данные вне MATLAB, то данные можно сохранить в текстовом виде, указав параметр `-ascii`.

Рабочая папка является первой в маршруте поиска, который представляет собой упорядоченный список папок. Когда вы вызываете функцию, MATLAB исполняет первый файл на своем пути, который имеет заданное имя. Отредактировать маршрут поиска можно через меню "Set Path".

1.1. Выражения

Переменные

В программе MATLAB нет необходимости в определении типа переменных и их размерности. Когда MATLAB встречается новое имя переменной, он автоматически создает переменную и выделяет соответствующий объем памяти. Если переменной присваивается новое значение, то MATLAB изменяет ее состав и выделяет дополнительную память по необходимости. Команда `clear` с именем переменной удаляет ее из памяти, а команда `clear all` удаляет все переменные.

Имена переменных состоят из букв, цифр и символов подчеркивания. MATLAB чувствителен к регистрам, различает строчные и заглавные буквы. Чтобы увидеть значение переменной, надо ввести ее имя в командной строке.

Числа

MATLAB хранит числа в машинном представлении и использует принятую десятичную систему счисления. Мнимые числа используют `i` или `j` как суффикс. Вот некоторые примеры:

3 -99 0.1250 1.4506e-10 4-3i 0.2101+0.3454j

Команда `format` управляет форматом вывода чисел на экран. MATLAB имеет несколько специальных констант:

`pi` — 3.14159265...;
`realmin` — наименьшее число с плавающей точкой;
`realmax` — наибольшее число с плавающей точкой;
`Inf` — бесконечность;
`NaN` — не число.

Бесконечность появляется при делении на нуль или при переполнении, а NaN при вычислении выражений типа $0/0$ или $\infty - \infty$.

Операторы

В выражениях используются обычные арифметические операции и правила старшинства:

- `-`, `+` — вычитание, сложение;
- `*`, `/` — умножение, деление;
- `\` — левое деление;
- `^` — возведение в степень;
- `'` — комплексно сопряженное транспонирование.

Для матриц эти операции определены естественным образом. Если перемножаются матрицы неподходящих размеров, MATLAB выдаст ошибку. Умножение и деление матриц на число осуществляется по математическим правилам. При сложении (вычитании) числа с матрицей, это число складывается (вычитается) со всеми элементами матрицы. Для матриц определены также поэлементные операции:

- `.*`, `./` — поэлементное умножение, деление;
- `.\` — поэлементное левое деление;
- `.^` — поэлементное возведение в степень;
- `.'` — несопряженное матричное транспонирование.

1.2. Работа с матрицами

Для эффективной работы в MATLAB надо сначала научиться работать с матрицами. Матрица в MATLAB — это таблица чисел, строки и столбцы которой нумеруются с 1. Особое значение придается матрицам 1×1 , которые являются скалярами, пустой матрице и матрицам-векторам. MATLAB организован так, чтобы команды над матрицами были как можно более естественными, и позволяет так же легко оперировать с целыми матрицами, как и с простыми числами.

Матрицу можно создать перечислением ее элементов, загрузить из внешнего файла и получить как результат функции. При указании списка элементов матрицы надо соблюдать следующие условия: 1) отделять элементы пробелами или запятыми, 2) использовать точку с запятой для обозначения окончания строки, 3) окружать весь список элементов квадратными скобками. Например, в результате команды

`A = [1 3 5; 2 4 6]`

MATLAB создаст переменную **A** размерности 2×3 и отобразит ее следующим образом:

```
A=
    1    3    5
    2    4    6
```

Выражение **B=[]** сохраняет пустую матрицу в переменную **B**. Элемент в строке *i* и столбце *j* матрицы **A** обозначается **A(i,j)**. Если попытаться использовать значение элемента с индексами за пределами размерности (например, **z=A(3,2)**), MATLAB выдаст ошибку. С другой стороны, если вы сохраняете значение вне матрицы, то размер матрицы увеличивается до подходящих размеров и новые элементы обнуляются. Например, в результате **A(3,5)=9** получится

```
A=
    1    3    5    0    0
    2    4    6    0    0
    0    0    0    0    9
```

Для создания матриц имеются следующие функции:

- ones(n)** — матрица из 1 размера $n \times n$;
- ones(n,m)** — матрица из 1 размера $n \times m$;
- zeros(n,m)** — матрица из нулей;
- eye(n,m)** — единичная матрица;
- rand(n,m)** — матрица со случайными числами р.р. на $[0, 1]$;
- randn(n,m)** — матрица со случайными числами из $N(0, 1)$.

Если имеется текстовый файл с именем **mytab.dat**, сформированный в виде прямоугольной таблицы чисел, отделенных пробелами, с равным количеством элементов в каждой строке, то команда **load mytab.dat** создаст переменную **mytab**, в которой будет храниться матрица, прочитанная из файла.

Двоеточие

Двоеточие — один из наиболее важных операторов. Он служит для создания списка значений и имеет две формы: однократное и двукратное двоеточие. Выражение **1:4** дает вектор-строку

```
1 2 3 4
```

Для получения обратного интервала следует описать приращение, например, **50:-4:35** дает

```
50 46 42 38
```

Выражение **0:pi/4:pi** дает

0 0.7854 1.5708 2.3562 3.1416

Если $r < s$, то выражения $s:r$ и $r:-0.1:s$ дают пустой вектор. Работа с подматрицами осуществляется следующим образом:

$b = A(:,j)$ — выбор j -го столбца;
 $b = A([4\ 3\ 2],:)$ — выбор строк с номерами 4, 3 и 2;
 $C = [A,B]$ — склеивание по горизонтали;
 $C = [A;B]$ — склеивание по вертикали;
 $A(:,1) = b$ — замена 1-го столбца на вектор b ;
 $A(3,[1:4]) = 0$ — обнуление элементов в 3-й строке с 1-го по 4-й столбец;
 $A(:,3) = []$ — удаление 3-го столбца.

где матрицы в выражениях должны иметь подходящие размеры. Кроме того, для индекса последнего элемента в некоторой размерности имеется ключевое слово `end`:

$b = A(:,[end-2:end])$ — выбор последних трех столбцов;
 $A(end,:) = []$ — удаление последней строки.

Элементы матрицы могут быть пронумерованы одним индексом: сначала нумеруются элементы 1-го столбца, затем 2-го и т.д.:

$b = A(:)$ — записывание всех элементов матрицы в виде столбца;
 $s = A(6)$ — выбор 6-го элемента матрицы, рассматриваемой в форме столбца.

Для работы с матрицами имеются следующие основные функции:

A' — транспонирование;
 $\text{sum}(A)$ — сумма элементов (по столбцам);
 $\text{sum}(A,\text{dim})$ — сумма вдоль указанной размерности (1 — по столбцам, 2 — по строкам);
 $\text{cumsum}(A)$ — накопленные суммы элементов (по столбцам);
 $\text{prod}(A)$ — произведение (по столбцам);
 $\text{mean}(A)$ — среднее (по столбцам);
 $\text{std}(A)$ — квадратичное отклонение (по столбцам);
 $\text{cov}(A)$ — ковариационная матрица;
 $\text{corrcoef}(A)$ — корреляционная матрица;
 $\text{size}(A)$ — вектор, содержащий размеры матрицы по каждой размерности;
 $\text{length}(v)$ — длина вектора;

`diag(A)` — вектор-столбец из элементов, стоящих на диагонали;
`diag(v)` — диагональная матрица с элементами вектора на диагонали;
`eig(A)` — матрица собственных векторов и диагональная матрица собственных чисел;
`svd(A)` — вычисление SVD-разложения.

Примеры использования этих функций даны в разделе 2.

В пакете MATLAB есть полный набор тригонометрических и сопутствующих функций (`sin`, `cos`, `asin`, `acos`, `exp`, `log`). В частности, имеются следующие функции:

`fix` — округление к 0;
`floor` — округление к $-\infty$;
`ceil` — округление к $+\infty$;
`round` — округление к ближайшему целому;
`mod` — знаковый остаток от деления;
`rem` — остаток от деления;
`sign` — знак;
`abs` — модуль;
`finite` — истина, если число конечно;
`isprime` — истина, если число простое.

Если в качестве аргумента у этих функций указать матрицу, то результатом будет матрица из значений функций для каждого элемента. В MATLAB истине соответствует 1, а значению ложь — 0.

Для работы с матрицами, рассматриваемыми как множества значений, имеются следующие функции:

`intersect(A,B)` — вектор общих элементов, упорядоченный по возрастанию;
`ismember(b,A)` — вектор той же длины, что и `b`, содержащий 1 на местах тех элементов `b`, которые принадлежат множеству `A`, и 0 иначе;
`union(A,B)` — вектор объединенных значений без повторяющихся элементов;
`unique(A)` — вектор значений, которые не повторяются;
`isequal(A,B)` — истина, если матрицы совпадают;
`any(A)` — истина, если существует элемент не равный нулю;

`all(A)` — истина, если все значения не равны нулю;
`isempty(A)` — истина, если матрица пустая.

При работе с матрицами имеется возможность обращения к подмножеству своих элементов с помощью так называемой логической индексации. Логические вектора, созданные с помощью логических операторов и операторов сравнения, могут быть использованы для ссылки на подмассивы. Предположим, что X — обыкновенная матрица и L — матрица такого же размера, но содержащая логические значения. Тогда $X(L)$ задает элементы X , в которых элементы L ненулевые. Этот вид индексации может быть осуществлен за один шаг указанием логической операции. Отметим, что логические значения выглядят при выводе как 1 и 0, но не являются числами. Например, в результате `x = x(finite(x))` из вектора `x` будут убраны элементы со значением `NaN` и `Inf`, команда

```
x = x( abs(x - mean(x)) <= 3*std(x) )
```

приведет к удалению тех значений, которые отклоняются от среднего более, чем на три стандартных отклонения, а выражение

```
A(~isprime(A)) = 0
```

приведет к обнулению всех чисел матрицы A , которые не обладают свойством простоты. Функция `find` определяет индексы массива элементов, удовлетворяющих заданным логическим условиям (для матриц используется одномерное индексирование). Например, результатом команд

```
A = [1 3 7; 2 4 9];  
k = find(isprime(A))'
```

будет

```
k =    2    3    5
```

1.3. Циклы и условные выражения

Логический оператор имеет вид

```
if логическое условие  
    операторы  
elseif логическое условие  
    операторы  
else  
    операторы  
end
```

Оператор выбора имеет вид

```

switch переменная
case значение1
    операторы
case значение2
    операторы
otherwise
    операторы
end

```

Выполняется только первый соответствующий случай. Таким образом, нет необходимости в использовании оператора `break`.

Цикл `for` повторяет группу операторов фиксированное, предопределенное число раз:

```

for переменная = вектор значений
    операторы
break
операторы
end

```

Цикл `while` выполняет группу операторов, пока выполняется логическое условие

```

while логическое условие
    операторы
end

```

Оператор `break` позволяет досрочно выходит из циклов. Например, следующие команды приводят к решению уравнения $x^3 - 2x - 5 = 0$ при $x \in [0, 3]$:

```

a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > 1e-10
    x = (a+b)/2;
    fx = x^3-2*x-5;
    if fx == 0
        break
    elseif sign(fx) == sign(fa)
        a = x; fa = fx;
    else
        b = x; fb = fx;
    end
end
x

```

1.4. Строки, массивы и структуры

Строки в MATLAB — это матрицы символов. В основе представления символов в строках лежит их кодирование: каждому символу соответствует некоторый код со значением от 0 до 255. Для формирования строки используется одинарный апостроф, `s='abc'`. Переменная `s` есть по сути вектор с кодами символов. Для работы со строками, в частности, имеются следующие функции

<code>char(b)</code>	— преобразование массива чисел целых положительных чисел в массив символов;
<code>double(s)</code>	— преобразование строки в вектор кодов;
<code>s=blanks(n)</code>	— формирование строки из <code>n</code> пробелов;
<code>deblank(s)</code>	— возвращает строку с удаленными из конца пробелами;
<code>ischar(s)</code>	— истина, если <code>s</code> есть символьная переменная;
<code>findstr(s1,s2)</code>	— вектор индексов, с которых начинается подстрока <code>s2</code> в строке <code>s1</code> ;
<code>strcmp(s1,s2)</code>	— истина, если две строки равны;
<code>strrep(s1,s2,s3)</code>	— замена всех подстрок <code>s2</code> на <code>s3</code> внутри <code>s1</code> ;
<code>int2str(A)</code>	— массив символов, коды которых есть округления элементов матрицы <code>A</code> ;
<code>mat2str(A,n)</code>	— преобразует матрицу в строку, используя <code>n</code> цифр после десятичной точки, расставляя точки с запятой для обозначений конца строк и обрамляя квадратными скобками;
<code>eval(s)</code>	— обратное к <code>mat2str</code> преобразование;
<code>num2str(A,n)</code>	— преобразует матрицу в вектор строк, используя <code>n</code> цифр после десятичной точки;
<code>str2double(s)</code>	— преобразование строки в вещественное число.

Строки можно склеивать так же, как и матрицы, например, `s=[s1, s2]` для конкатенации двух строк, `s=[s1; s2]` для объединения строк друг над другом. Функция `s=sprintf(format,...)` записывает форматированные данные в виде строки. Параметры форматирования аналогичны C++. Функция `A=sscanf(s,format)` анализирует строку с учетом формата.

Многомерные матрицы в MATLAB — это матрицы более чем с двумя индексами. Они могут быть созданы вызовом функций `zeros`, `ones`, `rand`, `randn` с более чем двумя аргументами. Например,

в результате команд:

```
n = 4;
A = eye(n);
p = perms(n);
M = zeros(n, n, size(p, 1));
for k = 1 : size(p,1)
    M(:, :, k) = A(:, p(k, :));
end
```

матрица **M** будет хранить последовательность перестановок (столбцов) единичной матрицы.

Массивы ячеек в MATLAB — это многомерные массивы, элементы которых являются данными разных типов и размерностей. Массивы создаются путем заключения разнообразной группы объектов в фигурные скобки. Фигурные скобки также используются с индексами для получения доступа к содержанию различных ячеек. Функция `cells` создает массив пустых элементов. Например, в результате

```
M = cells(8,1);
for k = 1 : 8
    M{n} = randn(k);
end
A = M{5}
```

будет создан массив, содержащий последовательность квадратных матриц разного размера.

Структуры — это многомерные массивы с элементами, доступ к которым осуществляется через поля. Например:

```
S.name = 'Tom';
S.score = 83;
S.height = 15;
```

создает скалярную структуру с тремя полями. Так как структуры по сути являются массивами, вы можете добавлять в них элементы. Поля могут добавляться либо по одному:

```
S(2).name = 'Nemo';
S(2).score = 79;
S(2).height = 24;
```

либо полностью:

```
S(3) = struct('name', 'Jerry', 'score', 88, 'height', 19);
```

Есть несколько способов передать различные поля в другие массивы MATLAB. Например, переменная `scores=[S.score]` будет вектором, содержащим указанное поле всех элементов структуры.

1.5. Графика

MATLAB имеет широкие возможности для графических изображений векторов и матриц. Особенностью MATLAB является то, что надо явно создать матрицу значений функции в определенных пользователем точках и затем построить график по этой матрице с помощью некоторой функции рисования.

Для изображения скалярной функции на плоскости MATLAB имеет функцию `plot`, которая имеет различные аргументы. Команда `plot(y)` создает кусочно-линейный график зависимости элементов вектора y от индексов. Команда `plot(x,y)` создает график зависимости вектора y от вектора x , их длины должны совпадать. Например, для построения графика функции $\sin(x)$ при $x \in [0, 2\pi]$ сделаем следующее:

```
t = 0 : pi/100 : 2*pi;
y = sin(t);
plot(t, y)
```

Вызов функции `plot` с многочисленными парами x - y создает несколько ломанных на одном рисунке, например:

```
y2 = sin(t - 0.25);
y3 = sin(t - 0.5);
plot(t, y, t, y2, t, y3)
```

Возможно изменение цвета, стиля линий и маркеров следующим образом:

```
plot(t, y, 'цвет_маркер_стиль')
```

Третий аргумент должен быть строкой из не более трех символов: первый символ определяет цвет, второй — маркер, третий — тип линии. Например, `plot(x, y, 'y+ :')` строит желтую пунктирную ломанную и помещает маркер $+$ в каждую точку данных. Другие возможности можно узнать, если выполнить команду `help plot`. Команда `plot(z)` для вектора z комплексных чисел строит график зависимости действительной части от мнимой.

Функция `plot` автоматически открывает новое окно, если до этого его не было на экране. Каждое окно изображения имеет свой номер, который указывает в заголовке окна. Команда `figure(n)` открывает или активизирует окно с данным номером, в которое будут выводиться результаты последующих команд рисования. Команда `hold on` позволяет добавлять новые кривые на существующий график, при этом изменяются оси, если это необходимо. Для рисования

нескольких графиков отдельно в одном окне MATLAB имеет команду `subplot(n,m,p)`. Эта команда разбивает окно на матрицу n на m подграфиков и выбирает p -й график текущим, например:

```
figure(1)
subplot(2,2,1); hist(t)
subplot(2,2,2); plot(t, y, 'r')
subplot(2,2,3); plot(y, t, 'r')
subplot(2,2,4); hist(y)
```

Здесь команда `hist` строит гистограмму, рассматривая вектор как выборку.

Для визуализации функции двух переменных необходимо создать матрицы X и Y , состоящие из одинаковых строк и столбцов соответственно. Функция `meshgrid` преобразует область определения, заданную через вектора x и y , в матрицы X и Y для использования при вычислении функции двух переменных. Строки матрицы X дублируют вектор x , а столбцы Y — вектор y . Это сделано для удобства, например, для построения графика функции $f(x, y) = \sin(\sqrt{x^2 + y^2})/\sqrt{x^2 + y^2}$ напомним:

```
x = [-8 : 0.4 : 8];
y = [-9 : 0.5 : 9];
[X,Y] = meshgrid(x,y);
R = sqrt(X.^2+Y.^2);
Z = sin(R)/(R+1e-50);
mesh(X,Y,Z)
```

Добавление очень маленького числа позволяет избежать неопределенности $0/0$. Функции `mesh` и `surface` отображают поверхность в трех измерениях. При этом `mesh` создает каркасную поверхность, где цветные линии соединяют точки в соседних узлах, а функция `surface` вместе с линиями отображает в цвете и саму поверхность. Также можно указывать вектора значений по осям x и y , а именно можно писать так: `surf(x,y,Z)`.

Матрицы могут отображаться как изображения. Элементы матрицы X — целые числа от 1 до 128 — определяют индекс в палитре `map`. Эта палитра задается матрицей размера 128×3 , строка которой хранит RGB цвет, например:

```
image(X)
colormap(map)
```

где X и `map` — заранее созданные матрицы.

Для плоского изображения графиков двумерных функций существуют функции `contour` и `pcolor`. Функция `contour(X,Y,Z,n)` создает n контурных линий. Функция `pcolor(X,Y,Z)` создает псевдоцветной график из закрашенных прямоугольников. Например, команды

```
contour(X,Y,Z,20)
hold on
pcolor(X,Y,Z)
shading interp
```

приводят к построению контурных линий на сглаженном цветном фоне.

Команда `axis` имеет несколько возможностей для настройки масштаба, ориентации и коэффициента сжатия. Например,

```
axis([xmin xmax ymin ymax zmin zmax])
```

устанавливает предельные значения. Для установки предельных значений также имеются команды `xlim([min max])`, `ylim([min max])`, `zlim([min max])`. Команда `grid on` включает сетку, а команда `grid off` выключает ее. Функции `xlabel`, `ylabel`, `zlabel` добавляют подписи к соответствующим осям. Функция `title` добавляет заголовок в верхнюю часть окна. Функция `text` добавляет текст в любое место графика.

В подписях можно применять простейшие команды $\TeX$:

```
xlabel('0 \leq t \leq 2\pi')
text(0,1,'точка (0,1)')
```

1.6. Сценарии и функции

Файлы, которые содержат код на языке MATLAB, имеют расширение *m*. MATLAB имеет текстовый редактор для написания и отладки *m*-файлов. Существует два вида *m*-файлов: сценарии и функции. По умолчанию новые *m*-файлы сохраняются в текущей папке. Если вы разработали много *m*-файлов, то следует их сгруппировать в отдельные папки по смыслу, в собственные *toolboxes*, и добавить эти папки в маршрут поиска через пункт меню "Set Path".

Сценарии не имеют входных и выходных аргументов. Когда вы вызываете сценарий, указав имя файла без расширения в командной строке, MATLAB просто вызывает команды, содержащиеся в файле. Сценарии могут оперировать существующими данными из рабочего пространства и могут сами создавать эти данные. Все переменные, созданные во время работы сценария, остаются в рабочем пространстве для использования в последующих вычислениях. Например, сценарий

```

n = 32;
% заранее выделяем память для увеличения быстродействия
r = zeros(1,32);
for k = 3:n
    r(k) = rank(magic(k));
end
r % вывод вектора на экран
bar(r)

```

вычисляет ранг магических квадратов и строит столбиковую диаграмму. Если бы не было второй строки, то в каждой итерации цикла размер вектора увеличивался бы на 1, что привело бы к выделению новой памяти и копированию элементов вектора. Такое **предварительное выделение** ведет к существенному увеличению быстродействия программы.

Функции имеют входные и выходные аргументы и оперируют с локальными переменными, а файл начинается с зарезервированного слова `function`. Имя файла и имя функции должно быть одним и тем же.

Если вы хотите завести **глобальные переменные**, чтобы в нескольких функциях и в сценариях использовать одну и ту же переменную, ее нужно объявить как `global`. Определение `global` должно быть сделано до использования самой переменной. Например, файл *fib.m*

```

function f = fib(n, v1, v2)
f = zeros(1,n);
if (nargin == 1)
    f(1) = 1; f(2) = 1;
elseif (nargin == 2)
    f(1) = v1; f(2) = 1;
else
    f(1) = v1; f(2) = v2;
end
for k = 3:n
    f(k) = f(k-1)+f(k-2);
end
f = f(1:n);

```

содержит функцию, возвращающую вектор, содержащий первые n чисел ряда Фибоначи с заданными начальными условиями `v1` и `v2`.

Этот пример показывает особенность функций MATLAB — переменное число аргументов. Эта функция может быть использована в различных формах:

```
x = fib(9,2,3); y = fib(9,2); z = fib(9);
```

Внутри тела функции всегда присутствуют две переменные `nargin` и `nargout`, которые равны числу входных и выходных аргументов при каждом вызове функции. Например, следующая функция имеет несколько выходных аргументов:

```
function [f, g, status] = myfun(x,y)
    status = ...
    if (nargout==1)
        f = x^3+x*y+y^2+x*exp(y);
    else
        f = x^3+x*y;
        g = y^2+x*exp(y);
    end;
```

Допускаются следующие вызовы такой функции:

```
nx = myfun(x,y); [nx, ny] = myfun(x,y); [nx, ny, s] = myfun(x,y);
```

В файле-функции может быть объявлено несколько функций, причем имя первой функции должно совпадать с именем файла. Другие функции (кроме первой) доступны для использования только в функциях из этого файла.

Векторизация — один из подходов к программированию на языке MATLAB. Девиз этого подхода — «жизнь слишком коротка, чтобы тратить время на запись циклов». Простым примером является создание таблицы значений функции:

```
n = 1001; x = 0; y = zeros(1,n);
for k = 1:n
    y(k) = log(x);
    x = x+0.01;
end
```

Векторизованная версия этого кода выглядит следующим образом:

```
x = 0 : 0.01 : 10;
y = log(x);
```

Векторизация также относится к особой записи собственных функций (со скалярными аргументами), так чтобы при вызове функции от матрицы получилась бы матрица значений. Это, в частности, необходимо при использовании функций интегрирования. Например, функция `myf`

```
function f = myf(x)
f = x^2+x*exp(x);
```

не векторизована. Ее векторизованная версия имеет вид

```
function f = myf(x)
f = x.^2+x.*exp(x);
```

Для более сложных программ возможность векторизации не так очевидна.

1.7. Функции от функций

В этом разделе будут рассмотрены функции, которым в качестве аргументов надо передавать другие функции. Такими функциями, в частности, являются функции, реализующие алгоритмы нахождения корней, оптимизации, интегрирования, решения дифференциального уравнения.

Функция `fzero('fun_name',x0)` ищет корень уравнения, где `x0` есть начальная точка для итерационного алгоритма поиска корня. Например, в результате `x=fzero('sin',2.9)` найдем, что `x=3.1416`. Функция `fsolve` (из `toolbox optim`) ищет решение системы уравнений $F(x) = 0$.

Функции `quad('f_name',a,b)` и `quad8 (quadl в MATLAB 7.x)` вычисляют интеграл методами низкого и высокого порядков. Например, результат команды `I=quad('sin',0,pi)` будет таким `I=2.0000166`.

Функция `fmins('f_name',x0)` в MATLAB 5.x (`fminsearch` в версиях MATLAB 6.x и выше) ищет минимум функции многих переменных с начальной точкой `x0`. Например, определим функцию трех переменных следующим образом:

```
function v = myfun(x)
v = x(1)*(x(1)+1)+(x(2)-1)^2-cos(x(3));
```

Тогда в результате `x=fmins('myfun',[0.1, 0.1, 0.1])` получим, что переменная `x` равна `[-0.5, 1.0, 0.0]`. Задачи на максимум надо сводить к задачам на минимум.

1.8. Работа с полиномами

Полином в MATLAB задается вектором коэффициентов, начиная с коэффициента при старшей степени. Например, полиному $x^3 - 2x + 4$ соответствует вектор `[1, 0, -2, 4]`. Для работы с полиномами существуют следующие команды, где `a`, `b`, `d`, `p`, `q` — полиномы, `r` — вектор корней, `S` — массив точек, `M` — квадратная матрица:

<code>poly(r)</code>	— задание полинома по корням;
<code>polyval(p,S)</code>	— вычисление значений полинома в точках <code>S</code> ;
<code>polyvalm(p,M)</code>	— вычисление значений полинома для матричного аргумента;
<code>polyder(p)</code>	— дифференцирование полинома;
<code>polyder(a,b)</code>	— дифференцирование произведения $a \cdot b$;
<code>[d,p]=polyder(a,b)</code>	— дифференцирование дроби a/b ;
<code>conv(a,b)</code>	— умножение полиномов;
<code>[q,p]=deconv(a,b)</code>	— деление полиномов;
<code>residue(a,b)</code>	— разложение a/b на простые дроби;
<code>roots(p)</code>	— вычисление корней полинома.

При работе с полиномами высоких степеней могут возникать ошибки вычислений. Например, при нахождении корней полиномов с чисто вещественными корнями могут получаться комплексные числа, `roots(poly([1:21]))`, и при вычислении значений полинома в малой окрестности кратного корня, `polyval((x - 1)^7, 0.99:1e-4:1.01)`.

1.9. Решение дифференциальных уравнений

MATLAB имеет ряд функций, в частности, `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, позволяющие решать системы обыкновенных дифференциальных уравнений (ОДУ) первого порядка $y' = F(t, y)$ с независимой переменной t и вектором неизвестных функций $y = y(t)$ методами Рунге-Кутты разных порядков и другими методами. Обращения ко всем функциям одинаково:

`[T,Y] = ode23('funF', [T0 T1], Y0)`

Здесь `[T0 T1]` — отрезок, на котором интегрируется система ОДУ, `Y0` — вектор начальных условий $y(T0)$. Результаты решения размещаются в матрице `Y`, каждая строка которой есть значение функции $y(t)$ в моменты времени t , расположенные в столбце `T`. Имя функции `funF` задает правую часть системы ОДУ. MATLAB имеет дополнительные возможности для решения дифференциальных уравнений, которые можно найти в окне помощи. В частности, MATLAB имеет в своем составе пакет PDE для решения уравнений в частных производных.

Пусть, например, требуется решить систему

$$\begin{cases} y_1' = y_1 + 2y_2y_3, \\ y_2' = y_2 - t(y_1 + y_3), \\ y_3' = y_1 + y_2 \end{cases}$$

на отрезке $[0, 4]$ с начальным условием $y_1(0) = 1, y_2(0) = 0, y_3(0) = 2$.
Для решения напишем файл *myDE.m*, в котором напишем:

```
function dy = myDE(t,y)
dy(1) = y(1) + 2*y(2)*y(3);
dy(2) = y(2) - t*(y(1)+y(3));
dy(3) = y(1) + y(2);
```

В сценарии теперь напишем:

```
[T,Y] = ode45('myDE', [0,4], [1,0,2]);
% Теперь визуализируем решение
plot(T,Y(:,1), T,Y(:,2), T,Y(:,3)) % графики функций  $y_i(t)$ 
plot3(Y(:,1), Y(:,2), Y(:,3)) % фазовая траектория в  $\mathbf{R}^3$ 
comet(Y(:,1), Y(:,3)) % подвижная фазовая траектория в  $\mathbf{R}^2$ 
```

1.10. Работа с файлами

Быстрыми и удобными в работе являются команда `save` для сохранения информации во внешний файл и команда `load` для ее считывания. Информация может сохраняться как в двоичном коде, так и в текстовом виде. Например:

```
save data a b % запись  $a, b$  в файл data.mat в двоичном коде
save out.txt -ascii a % запись  $a$  в текстовый файл out.txt
for i = 1 : 30
    a = rand(i); % запись в разные файлы разных данных
    save(['date', int2str(i)], '-ascii', 'a')
end
```

MATLAB также имеет следующие более гибкие операции ввода-вывода, напоминающие стандартные функции языка C:

```
fopen — открытие файла w/r/a, t/b;
fclose — закрытие файла;
fread — считывание бинарных данных из файла;
fwrite — запись данных в бинарном виде;
fscanf — считывание форматированных данных;
fprintf — запись данных в файл;
fgetl — считывание строки без служебных символов;
fgets — считывание строки со служебных символов;
dlmread — чтение данных с разделителями из текстового файла;
txtread — чтение форматированных данных из текстового файла.
```

Для считывания таблиц Excel есть команда `xlsread`.

Для импорта-экспорта графических файлов предназначены команды `imread` и `imwrite`:

```
A = imread('pic.jpg','jpg')
imwrite(A,'pic.bmp','bmp')
```

MATLAB умеет работать с файлами, в частности, в форматах `jpg`, `bmp`, `tif`, `pcx`. Матрица содержит числа типа `uint8` со значениями от 0 до 255. В большинстве случаев сразу после открытия файла следует писать `dA=double(A)` для изменения типа чисел и работать с матрицей `dA` из вещественных чисел. Размер матрицы `A` равен $n \times m$ для черно-белых изображений и равен $n \times m \times 3$ для цветных изображений. Команда `imfinfo` позволяет узнать параметры графического файла.

Для импорта-экспорта звуковых файлов имеются команды

```
[Y, fs, bits] = wavread('sound.wav',[n1 n2])
wavwrite(Y, fs, bits, 'sound.wav')
```

Массив `Y` содержит числа от -1 до 1 и имеет размерность длина выборки на число каналов. Если не указан второй аргумент в вызове функции, то считывается вся выборка (все отсчеты), а не указанная подвыборка (отсчеты). Переменная `fs` хранит частоту дискретизации (в герцах). Переменная `bits` указывает число бит, используемое при оцифровке сигнала по амплитуде. Результатом команды `sz=wavread('snd.wav', 'size')` будет вектор, который содержит объем выборки и число каналов. Команда `sound(Y, fs)` воспроизводит звуковой сигнал.

2. Примеры решения задач

2.1. Марковские цепи

Пусть P — переходная матрица марковской цепи. Стационарное распределение удовлетворяет уравнению $\pi P = \pi$. Из определения следует, что переходная матрица P имеет собственное число, равное 1, и π пропорционален соответствующему собственному вектору. Кроме того, существует $\lim_n P^n = P^*$ и каждая строка P^* равна π .

Задача. Найти стационарное распределение и построить график зависимости нескольких норм отклонений матриц P^n от предельной.

Для решения задачи напомним следующий сценарий:

```

m = 6; % размер матрицы P
P = rand(m); % создадим случайную матрицу P
% нормализуем так, чтобы сумма по каждой строке = 1
P = diag(1./sum(P,2))*P
[Vecs, Eigs] = eig(P'); %находим с.ч. и с.в. переходной матрицы
for i = 1 : m
    if (abs(Eigs(i,i)-1)<1e-9)
        StatDistr = Vecs(:,i)'; % ищем решение уравнения P'v = v
    end
end
StatDistr = StatDistr/sum(StatDistr) % нормализуем
n = [2:7]; % будем считать нормы отклонений для этих n
limitP = ones(m)*diag(StatDistr);
norms = zeros(4,length(n)); % выделим память
for j = 1 : length(n)
    M = P^n(j) - limitP;
    norms(1,j) = sqrt(sum(sum(M.^2)));
    norms(2,j) = norm(M,Inf); % max(sum(abs(M')))
    norms(3,j) = norm(M,1); % max(sum(abs(M)))
    norms(4,j) = norm(M); % max(sqrt(eig(M*M')))
end
figure(1); clf % открываем окно с номером 1 и очищаем его
CurvesPrm = {'r-', 'g-', 'b-', 'k-'};
for i = 1 : 4
    hold on
    plot(n, norms(i,:), CurvesPrm{i})
end
title('different norms')
xlabel('n')

```

2.2. Вычисление интегралов методом Монте-Карло

Вычисление интеграла J по методу Монте-Карло состоит в построении случайной величины ξ такой, что $\mathbf{E}\xi = J$ и $\mathbf{D}\xi = \sigma^2 < \infty$, и использовании закона больших чисел (ЗБЧ). А именно промоделируем $\xi_1, \dots, \xi_N$. Положим $J_N = \frac{1}{N} \sum_{i=1}^N \xi_i$. Тогда $J_N \rightarrow J$ по вероятности при $N \rightarrow \infty$ (по ЗБЧ) и $\mathbf{P}((J_N - J) \frac{\sqrt{N}}{\sigma} < x) \rightarrow \Phi(x)$ (по центральной предельной теореме (ЦПТ), $\Phi(x)$ — функция распределения нормального закона $N(0, 1)$). Доверительный интервал

для истинного значения интеграла с надежностью $1 - \gamma$ равен

$$(J_N - \frac{x_\gamma \sigma}{\sqrt{N}}, J_N + \frac{x_\gamma \sigma}{\sqrt{N}}),$$

где γ и x_γ связаны соотношением $\Phi(x_\gamma) = (1 + \gamma)/2$. Для 95% надежности $x_{5\%} \approx 1.96$. Положим

$$s_N^2 = \frac{1}{N} \sum_{i=1}^N (\xi_i - J_N)^2 = \frac{1}{N} \sum_{i=1}^N \xi_i^2 - J_N^2.$$

Тогда $s_N \rightarrow \sigma$ по ЗБЧ.

Задача. Для интеграла $J = \int_0^1 \sin(x + e^{-x}) dx$ вычислить J_N при разных N для случая зависимых и независимых оценок и построить график J_N с доверительными интервалами.

Для вычисления интеграла построим случайную величину $\xi = \sin(\alpha + e^{-\alpha})$, где α равномерно распределенная случайная величина на $[0, 1]$. Для построения независимых оценок J_{N_1} и J_{N_2} используются разные реализации случайной величины ξ . Для построения зависимых оценок J_{N_1} и J_{N_2} используются $\min\{N_1, N_2\}$ одних и тех же реализаций случайной величины ξ .

2.3. Фракталы

Задача. Изобразить точки на плоскости, получаемые итеративно через систему уравнений. Фрактал Mira curve задается следующей системой:

$$\begin{aligned} x_{k+1} &= by_k + F(x_k), \\ y_{k+1} &= -x_k + F(x_{k+1}), \end{aligned}$$

где $F(x) = ax + 2(1 - a)x^2/(1 + x^2)$, $|a| < 1$, $b \approx 1$. Для построения использовать следующие начальные данные и сравнить траектории при $b = b_1$ и при $b = b_2$, $n = 10000$:

a	b_1	b_2	x_1	y_1
0.5	1	0.9998	3	0
-0.99	1	0.98	4	0
0.99	1	0.9998	0	12
-0.48	1	0.99	4	0
0.25	1	0.99	3	0
0.7	1	0.9998	0	12.1

В качестве упражнения изобразите фрактал Henon's model, который задается следующими уравнениями:

$$\begin{aligned}x_{k+1} &= ax_k - b(y_k - x_k^2), \\y_{k+1} &= bx_k + a(y_k - x_k^2),\end{aligned}$$

где $|a| \leq 1$, $b = \sqrt{1 - a^2}$. $k = 1, \dots, n$. Для построения используйте следующие начальные данные:

$n = 200$, $a = 0.24$, $(x_1, y_1) = (0.5696, 1622)$ и $(x_1, y_1) = (0.565, 165)$,
 $n = 2000$, $a = 0.24$, (x_1, y_1) – 40 значений р.р. на $[-1, 1]^2$.

2.4. Имитационное моделирование

В ряде задач требуется найти закон распределения некоторой случайной величины, которая задана посредством сложной алгоритмической формулы. Найти явный вид распределения не всегда удается. В этом случае единственным средством является моделирование этой случайной величины. Выборочное распределение будет сходиться к искомому.

Задача. Рассмотрим урновую схему. Вначале в урне находится x_1 черных шаров и y_1 белых шаров. Затем итеративно повторяется следующий процесс: достаем шар, смотрим на цвет, возвращаем шар обратно, кладем шар такого же цвета, $k = 1, \dots, n$. Нарисовать (выборочное) распределение $\eta_n = x_n / (x_n + y_n)$, при $n = 100$, $n = 300$. Нарисовать 20 траекторий (x_k, y_k) , $k = 1, \dots, 300$, на одном рисунке.

На одном рисунке будут изображены ломаные, разбегающиеся из левого нижнего угла в случайном направлении. На другом рисунке будет изображена выборочная плотность случайной величины η_n . Известно, что при $n \rightarrow \infty$ распределение случайной величины η_n стремится к бета распределению.

2.5. Оценки максимального правдоподобия

Пусть $(x_1, \dots, x_N)$ есть выборка из генеральной совокупности с плотностью распределения $f(x, \theta)$, где θ есть вектор параметров распределения. Оценка максимального правдоподобия (ОПМ) параметра θ максимизирует функцию правдоподобия, задаваемую формулой $L(\theta) = \prod_{j=1}^N f(x_j, \theta)$. Для дискретного распределения

$L(\theta) = \prod_{j=1}^N \mathbf{P}_\xi(\xi = x_j)$. Для нахождения ОМП обычно решают систему из необходимых условий экстремума $\partial \ln L(\theta) / \partial \theta = 0$.

Задача. Найти оценки максимального правдоподобия для выборки из сложно биномиального распределения. Случайная величина ξ имеет сложно биномиальное распределение с параметрами (n_1, p_1, n_2, p_2) , если производящая функция имеет вид $h(t) = ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2}$, а

$$\mathbf{P}(\xi = j) = b_2(j|n_1, p_1, n_2, p_2) = \sum_{i=\lceil j/n_1 \rceil}^{n_2} b(i|n_2, p_2) b(j|in_1, p_1),$$

$j = 0, 1, \dots, n_1 n_2$, где выражение $b(j|n, p) = C_n^j p^j q^{n-j}$, $q = 1 - p$, определяет вероятность обычного биномиального распределения.

В качестве упражнения найдите ОМП для выборки из отрицательно биномиального распределения с параметрами (n, p) , вероятности которого задаются формулой $\mathbf{P}(\xi = j) = b_-(j|k, p) = C_{j+k}^k p^k q^j$, $q = 1 - p$, $j = 0, 1, \dots$.

Задача. Рассмотреть функцию правдоподобия как функцию двух переменных $f(p_1, p_2) = L(p_1, p_2, n_1, n_2)$. Параметры n_1, n_2 считать известными и равными истинным значениям. Требуется вычислить 30 итераций метода покоординатного подъема с начальной точкой $(0.7, 0.3)$ к максимуму функции и изобразить итерации на плоском изображении функции.

В качестве упражнения для отрицательно биномиального распределения рассмотрите $f(n_1, p_1) = L(n_1, p_1)$ при нецелом n_1 (при этом факториал в определении вероятностей заменяется на Γ -функцию).

Доверительные интервалы для ОМП

Из теории известно (Рао, 1968), что при некоторых условиях асимптотическая ковариационная матрица B оценок максимального правдоподобия $\hat{\theta}$ имеет вид $B = R^{-1}$, где R — ковариационная матрица случайной величины $g(\xi)$,

$$g(x) = \left(\frac{\partial}{\partial \theta_1} \ln f(x, \theta), \dots, \frac{\partial}{\partial \theta_m} \ln f(x, \theta) \right)^T, \theta = (\theta_1, \dots, \theta_m)^T,$$

выборка взята из генеральной совокупности с плотностью распределения $f(x, \theta)$ и случайная величина ξ имеет плотность $f(x, \theta)$.

Отметим, что математическое ожидание $g(\xi)$ равно нулю, и, следовательно, $R = \mathbf{E}g(\xi)g^T(\xi)$. Диагональные элементы матрицы B определяют длину доверительных интервалов.

Построим доверительные интервалы для параметров отрицательно биномиального распределения. При нецелом параметре n распределение имеет вид

$$b_{-}(j|k, p) = \frac{\Gamma(k+j)}{\Gamma(k)\Gamma(j+1)} p^k (1-p)^j, \quad j = 0, 1, \dots$$

Эти вероятности можно вычислять, не используя Γ -функцию:

$$b_{-}(j|k, p) = \frac{(k+j-1)(k+j-2)\cdots k}{j!} p^k (1-p)^j.$$

Частные производные имеют вид

$$\begin{aligned} \frac{\partial}{\partial p} \ln b_{-}(j|k, p) &= \frac{k}{p} - \frac{j}{1-p}, \\ \frac{\partial}{\partial k} \ln b_{-}(j|k, p) &= \sum_{i=0}^{j-1} \frac{1}{k+i} + \ln p. \end{aligned}$$

Сначала рассмотрим случай, когда параметр k известен. Тогда мы имеем

$$r_{22} = \mathbf{E} \left(\frac{\partial}{\partial p} \ln b_{-}(\xi|k, p) \right)^2 = \mathbf{D} \frac{\partial}{\partial p} \ln b_{-}(\xi|k, p) = \frac{\mathbf{D}\xi}{(1-p)^2} = \frac{k}{p^2(1-p)}.$$

Отсюда получаем γ -доверительный интервал для p при известном k

$$\left(\hat{p} - \frac{x_{\gamma}}{\sqrt{N}} \sqrt{\frac{\hat{p}^2(1-\hat{p})}{k}}, \hat{p} + \frac{x_{\gamma}}{\sqrt{N}} \sqrt{\frac{\hat{p}^2(1-\hat{p})}{k}} \right),$$

где $\Phi(x_{\gamma}) = 1 - \gamma/2$ и $\Phi(x)$ есть функция распределения стандартного нормального закона.

Рассмотрим теперь случай, когда параметр p известен. Тогда, используя определение математического ожидания и равенство $g(\xi) = 0$, имеем

$$r_{11} = \mathbf{E} \left(\sum_{i=0}^{\xi-1} \frac{1}{k+i} + \ln p \right)^2 = \sum_{j=0}^{\infty} \left(\sum_{i=0}^{j-1} \frac{1}{k+i} + \ln p \right)^2 b_{-}(j|k, p) =$$

$$= \sum_{i=0}^{\infty} \left(\sum_{i=0}^{j-1} \frac{1}{k+i} \right)^2 - \ln^2 p.$$

Отсюда получаем γ -доверительный интервал для k при известном p

$$\left(\hat{k} - \frac{x_\gamma}{\sqrt{\hat{r}_{11}N}}, \hat{k} + \frac{x_\gamma}{\sqrt{\hat{r}_{11}N}} \right).$$

Перейдем теперь к случаю обоих неизвестных параметров. Отметим, что диагональные элементы матрицы $R = (r_{ij})$ уже найдены. Можно показать, что $r_{12} = -1/p$. Следовательно, матрица $B = R^{-1}$ принимает вид

$$B = \frac{1}{r_{11}k - (1-p)} \begin{pmatrix} k & p(1-p) \\ p(1-p) & r_{11}p^2(1-p) \end{pmatrix}.$$

Отсюда получаем γ -доверительные интервалы для p и k

$$\left(\hat{p} - \frac{x_\gamma}{\sqrt{N}} \sqrt{\frac{\hat{p}^2(1-\hat{p})}{\hat{k} - (1-\hat{p})/\hat{r}_{11}}}, \hat{p} + \frac{x_\gamma}{\sqrt{N}} \sqrt{\frac{\hat{p}^2(1-\hat{p})}{\hat{k} - (1-\hat{p})/\hat{r}_{11}}} \right),$$

$$\left(\hat{k} - \frac{x_\gamma}{\sqrt{(\hat{r}_{11} - (1-\hat{p})/\hat{k})N}}, \hat{k} + \frac{x_\gamma}{\sqrt{(\hat{r}_{11} - (1-\hat{p})/\hat{k})N}} \right),$$

Заметим, что эти доверительные интервалы шире, чем интервалы, полученные в предположении того, что другой параметр известен.

Рассмотрим теперь вопрос получения доверительных интервалов для сложно биномиального распределения. Непосредственное дифференцирование $\ln b_2(j|n_1, p_1, n_2, p_2)$ по параметрам приводит к следующим формулам:

$$\frac{\partial}{\partial p_1} \ln b_2(j|\theta) = \frac{1}{b_2(j|\theta)} \sum_{i=\lceil j/n_1 \rceil}^{n_2} \frac{n_2!}{i!(n_2-i)!} \frac{(in_1)!}{j!(in_1-j)!} \times$$

$$\times p_2^i (1-p_2)^{n_2-i} (j p_1^{j-1} (1-p_1)^{in_1-j} - (in_1-j) p_1^j (1-p_1)^{in_1-j-1}),$$

$$\frac{\partial}{\partial p_2} \ln b_2(j|\theta) = \frac{1}{b_2(j|\theta)} \sum_{i=\lceil j/n_1 \rceil}^{n_2} \frac{n_2!}{i!(n_2-i)!} \frac{(in_1)!}{j!(in_1-j)!} \times$$

$$\times (i p_2^{i-1} (1-p_2)^{n_2-i} - (n_2-i) p_2^i (1-p_2)^{n_2-i-1}) p_1^j (1-p_1)^{in_1-j}.$$

Эти значения можно получить, используя свойства производящей функции. Дифференцируя

$$\sum_{j=0}^{n_1 n_2} b_2(j|\theta) t^j = h(t) = ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2}$$

по некоторому параметру ($\theta = (n_1, p_1, n_2, p_2)$), получаем

$$\sum_{j=0}^{n_1 n_2} \frac{\partial}{\partial \theta_i} b_2(j|\theta) t^j = \frac{\partial}{\partial \theta_i} ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2}.$$

В результате имеем

$$\frac{\partial}{\partial \theta_i} \ln b_2(j|\theta) = \frac{1}{b_2(j|\theta)} \frac{\partial}{\partial \theta_i} b_2(j|\theta).$$

Например:

$$\begin{aligned} \frac{\partial}{\partial p_1} h(t) &= n_2 ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2-1} p_2 n_1 (p_1 t + q_1)^{n_1-1} (t-1), \\ \frac{\partial}{\partial p_2} h(t) &= n_2 ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2-1} ((p_1 t + q_1)^{n_1-1} - 1), \\ \frac{\partial}{\partial n_2} h(t) &= n_2 ((p_1 t + q_1)^{n_1} p_2 + q_2)^{n_2-1} \ln((p_1 t + q_1)^{n_1-1} p_2 + q_2). \end{aligned}$$

Хотя производные по n_1, n_2 содержат логарифм, ряды Тейлора этих функций содержат конечное число ненулевых коэффициентов.

В частности, доверительный интервал для p_1 при известных n_1, n_2, p_2 имеет вид

$$\left(\hat{p}_1 - \frac{x_\gamma}{\sqrt{\hat{r}N}}, \hat{p}_1 + \frac{x_\gamma}{\sqrt{\hat{r}N}} \right),$$

где

$$\begin{aligned} \hat{r} &= \sum_{j=0}^{n_1 n_2} \frac{1}{b_2(j|\hat{\theta})} \left(\frac{\partial}{\partial p_1} b_2(j|\hat{\theta}) \right)^2 \approx \\ &\approx \sum_{j=0}^{n_1 n_2} \left(\frac{\ln b_2(j|n_1, p_1 + \varepsilon, n_2, p_2) - \ln b_2(j|n_1, p_1, n_2, p_2)}{\varepsilon} \right)^2 b_2(j|\hat{\theta}), \end{aligned}$$

последнее приближение удобно для грубых численных расчетов.

Полученные доверительные интервалы являются асимптотическими и верны при больших объемах выборки N . Для того чтобы

получить доверительный интервал при маленьких N и найти то значение N , начиная с которого верна асимптотическая формула, необходимо воспользоваться методом статистического моделирования. Оценка параметров $\hat{\theta} = \hat{\theta}(x_1, \dots, x_N)$ по выборке из генеральной совокупности с плотностью распределения $f(x|\theta)$ имеет распределение $\mathcal{P} = \mathcal{P}_{N,\theta} = \mathcal{L}(\hat{\theta}(\xi_1, \dots, \xi_N))$, где ξ_i — независимые случайные величины с плотностью распределения $f(x|\theta)$. Найти явный и простой вид $\mathcal{P}$ очень трудно. Моделируя, мы можем получить приближение к распределению $\mathcal{P}$. А именно, моделируя несколько выборок $(x_1^{(j)}, \dots, x_N^{(j)})$, по каждой вычислим оценки $\hat{\theta}_{(j)}$, $j = 1, \dots, L$. Эти оценки, тоже образуя выборку, характеризуют распределение $\mathcal{P}$. В частности, $\gamma\%$ -доверительный интервал образован $(1 - \gamma)/2$ и $(1 + \gamma)/2$ квантилями выборочной функции распределения, построенной по $(\hat{\theta}_{(1)}, \dots, \hat{\theta}_{(L)})$.

Задача. Получить выборку. Построить 90%-ный и 95%-ный доверительные интервалы для параметров. Вычислить и изобразить (выборочные) распределения оценок параметров. Вычислить 0.05, 0.1, 0.5, 0.9 и 0.95 квантили у полученных распределений.

2.6. Спектральный анализ и фильтрация

В этом разделе мы рассмотрим некоторые классические инструменты исследования стационарных процессов.

Пусть ξ_n — дискретный стационарный процесс со средним 0. Функция $R(n) = E\xi_{k+n}\xi_k$ называется ковариационной функцией процесса. Из теории процессов известно, что для любой стационарной последовательности существует мера $F(\cdot)$ такая, что

$$R(n) = \int_{-\pi}^{\pi} e^{in\lambda} F(d\lambda),$$

и ортогональная стохастическая мера $\mu(\cdot)$ такая, что

$$\xi_n = \int_{-\pi}^{\pi} e^{in\lambda} \mu(d\lambda)$$

и $F(d\lambda) = E|\mu(d\lambda)|^2$. Эти равенства называются спектральными разложениями. Из них вытекает, что стационарный процесс можно рассматривать как сумму элементарных тригонометрических функций со случайными амплитудами. Спектральное разложение

устанавливает связь между временным и частотным описанием одного и того же объекта. Для непрерывного процесса также верно спектральное разложение, только область интегрирования заменяется на прямую. Если $F(d\lambda) = f(\lambda)d\lambda$, то в силу того, что разложение представляет собой преобразование Фурье, справедливо выражение для спектральной плотности $f(\lambda)$

$$f(\lambda) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} R(n)e^{-in\lambda}.$$

Оценки среднего и ковариационной функции по реализации $(x_1, \dots, x_N)$ имеют вид

$$\bar{x} = \frac{1}{N} \sum_{j=1}^N x_j, \quad \hat{R}(n) = \frac{1}{N-n} \sum_{j=1}^{N-n} x_j x_{j+n}.$$

Оценка спектральной плотности

$$\hat{f}(\lambda) = \frac{1}{2\pi N} \left| \sum_{j=1}^N x_j e^{-i\lambda j} \right|^2$$

является несостоятельной. Для исправления оценки используется спектральное окно $w(\nu)$, которое применяется следующим образом:

$$f_w(\lambda) = \int f(\lambda - \nu) w(\nu) d\nu.$$

Спектральное окно есть некоторая функция $w(\nu)$, для которой выполняются условия: 1) $\int w(\nu) d\nu = 1$, 2) $w(\nu) = w(-\nu)$, 3) походить на дельта функцию, и которое выбирается так, чтобы оценка $\hat{f}_w(\lambda)$ была состоятельной.

Фильтрацией называется вычисление ряда η_n по ряду ξ_n по формуле

$$\eta_n = \sum_{j=-\infty}^{+\infty} b_j x_{n+j},$$

которая представляет собой линейное преобразование. Числа b_j называются коэффициентами фильтра. Фильтрация применяется в случаях, когда из ряда требуется выделить слагаемое с заданной

частотной полосой, или устранить заданную частоту, или сгладить ряд. Введем передаточную функцию фильтра $b(\lambda) = \sum_{j=-\infty}^{+\infty} b_j e^{-i\lambda j}$. Такое название дано в силу важного равенства между спектральными плотностями исходного и фильтрованного процесса

$$f_\eta(\lambda) = |b(\lambda)|^2 f_\xi(\lambda).$$

Это равенство есть, на самом деле, теорема о свертке. Таким образом, многократное трудоемкое прямое и обратное преобразование Фурье может быть заменено на простое линейное преобразование с заранее найденными коэффициентами фильтра с подходящей передаточной характеристикой.

При выделении частотной полосы $[\lambda_1, \lambda_2]$ фильтр с передаточной характеристикой $1_{[\lambda_1, \lambda_2]}(\lambda)$ имеет бесконечное число ненулевых коэффициентов. На практике для фильтрации рядов строят фильтры с конечным числом коэффициентов таких, чтобы передаточная характеристика была близка к идеальной.

Для анализа рядов MATLAB, в частности, имеет следующие функции:

fft(x,N) — быстрое преобразование Фурье;
 ifft(p) — обратное преобразование Фурье;
 conv(x,y) — свертка;
 filter(b,a,x) — фильтрация.

Рассмотрим пример анализа временного ряда, полученного суммированием двух периодических составляющих и случайной ошибки:

```
FS = 1000; % частота дискретизации
t = 0:1/FS:0.7; % точки, в которых вычисляется функция
N = length(t); % длина ряда
x = sin(2*pi*t*50)+sin(2*pi*t*120)+rand(size(t));
px = fft(x); % делаем преобразование Фурье
f = FS*[1:N/2]/N; % значения по оси x для периодограммы
pgx = px.*conj(px)/N; % вычисляем периодограмму
subplot(2,1,1)
plot(t,x)
title('series')
subplot(2,1,2)
plot(f,pgx([1:N/2]))
title('periodogram')
```

Можно видеть, что амплитуда спектра максимальна в 3 точках: 0, 50 и 120. Отметим, что для вещественных рядов спектральная плотность симметрична. Поэтому мы изобразили только половину вектора `pgx`. Кроме того, `px(1)` содержит сумму исходных данных, а `px(N/2+1)` при четном `N` соответствует частоте Найквиста. А именно для обнаружения частоты f_0 частота дискретизации должна быть не менее $2f_0$ (т.е. расстояние между соседними точками не более $1/(2f_0)$).

Для проверки теоремы о свертке векторов `x` и `b` напишем:

```
px = fft([x, zeros(1,length(a)-1)]);
pb = fft([b, zeros(1,length(x)-1)]);
y1 = conv(b,x)
y2 = ifft(px.*pb)
```

Векторы `y1` и `y2` будут совпадать.

Известно, что скользящее среднее приводит к сглаживанию, т.е. к удалению высокочастотных составляющих. Например, воспользуемся фильтром с одинаковыми коэффициентами для реализации скользящего среднего:

```
x = sin(2*pi*t*20)+rand(size(t));
y = filter(ones(1,11)/11,1,x);
plot(t,x,t,y)
```

Однако фильтрация приводит к запаздыванию во времени. Фильтрация с помощью функции `filtfilt(b,a,x)` из пакета `signal` исправляет этот недостаток.

В пакете `signal` имеется огромное количество функций для обработки рядов. Для вычисления данных для построения периодограммы имеется функция `periodogram`. Например:

```
[pgx, f] = periodogram(x,[],N,FS) % по N точкам вектора x;
periodogram(x,[],N,FS) % построение графика.
```

Функция `freqz` вычисляет передаточную характеристику (ПХ) фильтра, задаваемую формулой

$$a_1 y_k = b_1 x_k + \dots + b_{p+1} x_{k-p} - a_2 y_{k-1} - \dots - a_{m+1} y_{k-m},$$

и равную $|b(\lambda)|^2/|a(\lambda)|^2$:

```
[h, f] = freqz(b,a,n,FS) % длина векторов h,f равна n
freqz(b,a,n,FS) % построение графика ПХ
```

С помощью `FilterDesignToolbox` можно визуально задавать передаточную характеристику фильтра. Функция `fir1` вычисляет коэффициенты фильтра с заданными характеристиками

```

b = fir1(n,q,'type') % длина a равна n
b = fir1(32,[0.4 0.6]) % полосовой фильтр  $1_{[0.4,0.6]}$ 
b = fir1(32,0.3,'low') % фильтр низких частот  $1_{[0,0.3]}$ 
b = fir1(32,0.8,'high') % фильтр верхних частот  $1_{[0.8,1]}$ 

```

Чем больше длина фильтра, тем ближе его передаточная характеристика к идеальной, но хуже получается начало отфильтрованного ряда и медленнее реакция на изменение структуры ряда. Проверить характеристики фильтра можно следующим образом:

```

figure(3) % рисуется преобразование Фурье в разных шкалах
freqz(b,1)
[h,f] = freqz(b,1,50,FS)
figure(4)
plot(f,abs(h))

```

Задача. Прочитать wav-файл, убрать шум и записать результат в файл.

Следующий сценарий позволяет убрать только «шипение».

```

fname = 'win.wav'
[Y, FS, BITS] = wavread(fname);
pass = [300/(FS/2) 3000/(FS/2)];
% пропускаем частоты от 300Гц до 3кГц
b = fir1(round(FS/100),pass);
Y = filter(b,1,Y);
sound(Y,FS) % озвучивание данных
wavwrite(Y,FS,BITS,'out.wav')

```

2.7. Метод «Гусеница»-SSA

Для анализа временных рядов предложено в литературе множество методов. Особенностью метода «Гусеница»-SSA является то, что этот метод является непараметрическим. Далее мы приведем основные понятия базового алгоритма. Полное описание метода «Гусеница»-SSA можно найти в [2], [3].

Рассмотрим ряд $F = (f_1, \dots, f_N)$, где $f_j = f(j\Delta)$, $f(t)$ — некоторая функция, Δ — некоторый временной интервал. У метода есть целый параметр L , который называется длиной окна (длиной гусеницы). Обычно выбирают L такое, что $1 < L < N/2$. Ряду F сопоставим траекторную матрицу

$$\mathbb{X} = (x_{ij})_{i,j=1}^{L, N-L+1} = \begin{pmatrix} f_1 & f_2 & f_3 & \dots & f_{N-L+1} \\ f_2 & f_3 & f_4 & \dots & f_{N-L+2} \\ f_3 & f_4 & f_5 & \dots & f_{N-L+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_L & f_{L+1} & f_{L+2} & \dots & f_N \end{pmatrix}.$$

Далее построим сингулярное разложение (SVD) траекторной матрицы

$$\mathbb{X} = \sum_{j=1}^L \sqrt{\lambda_j} u_j v_j^T,$$

где λ_j и u_j есть собственные числа и ортонормированные собственные вектора ковариационной матрицы $S = \mathbb{X}\mathbb{X}^T$, $\lambda_j \geq \lambda_{j+1}$. Вектор v_j называют факторным вектором, а $\sqrt{\lambda_j} v_j$ называют j -м вектором главных компонент (ГК) по аналогии с терминологией факторного анализа. Векторы u_j и v_j также называются левыми и правыми сингулярными векторами матрицы $\mathbb{X}$. Доля $\lambda_k / \sum_j \lambda_j$ показывает вклад k -й компоненты в общее поведение ряда. Обозначим $\mathbb{X}_{(k)} = \sqrt{\lambda_k} u_k v_k^T$. Матрица $\mathbb{X}_{(k)}$ не является траекторной. Поэтому при обратном сопоставлении матрицы $\mathbb{X}_{(k)} = (x_{(k)i,j})$ ряду $F_{(k)} = (f_{(k)j})$ проведем усреднение вдоль диагоналей

$$f_{(k)j} = \frac{1}{L} \sum_{i=1}^L x_{(k)i,j-L+1}$$

для $L \leq j \leq N - L + 1$, для крайних элементов ряды усреднение происходит по меньшему числу элементов. В результате получаем, что исходный ряд F представляется в виде суммы рядов

$$f_j = \sum_{k=1}^L f_{(k)j}.$$

Оказывается, что описанная процедура приводит к тому, что отдельные слагаемые могут быть интерпретированы как составляющие тренда, периодической компоненты, или шума.

Из линейной алгебры следует, что если $r = \text{rank } \mathbb{X}$, то $r = \text{rank } \mathbb{X}\mathbb{X}^T$, SVD разложение траекторной матрицы $\mathbb{X}$ имеет ровно r ненулевых собственных чисел и r равно размерности линейного пространства, натянутого на столбцы матрицы $\mathbb{X}$.

Говорят, что ряд F имеет ранг r , если $r = \text{rank } \mathbb{X}$ не зависит от L и N ($L \geq r$, $N \geq L + r$).

Рассмотрим ряд примеров. Ранг экспоненциального ряда $f_j = a^j$ равен $r = 1$, так как j -й столбец $\mathbb{X}$ равен $a^j(1, a, \dots, a^{L-1})^T$. Ранг тригонометрического ряда $f_j = \cos(j2\pi\omega + a)$ равен 2 при $0 < \omega < 0.5$, так как с использованием формулы косинуса суммы аргументов, получается, что j -й столбец $\mathbb{X}$ равен линейной комбинации двух векторов с коэффициентами $\cos(i\omega)$ и $-\sin(i\omega)$. Ранг полиномиального ряда $f_j = q(j) = j^{p-1} + q_{p-2}j^{p-2} + \dots + q_1j + q_0$ равен p , так как j -й столбец $\mathbb{X}$ равен линейной комбинации p линейно независимых векторов $(0^k, 1^k, 2^k, \dots, (L-1)^k)^T$, $k = 0, \dots, p-1$. Ранг ряда $f_j = j^{p-1}a_j \cos(j\omega + a)$ не превосходит $2p$. Очевидно, что ранг суммы рядов не превосходит суммы рангов отдельных рядов.

Можно показать [3], что любой бесконечный ряд конечного ранга является суммой произведений экспоненты, косинуса и полинома. Кроме того, пусть исходный ряд есть сумма слагаемых вида $f_j = q(j)a_j \cos(j\omega + a)$. Тогда при выполнении условий разделимости (см. [2]) компоненты сингулярного разложения можно разбить на группы, соответствующие слагаемым исходного ряда.

Рассмотрим ряд конечного ранга r ($r < L$) с добавленным к нему небольшим шумом. В силу непрерывности сингулярное разложение будет отличаться незначительно от разложения незашумленного ряда. А именно мы получим r больших собственных чисел с интерпретируемыми собственными векторами в виде произведения экспоненты, косинуса и полинома, и $L - r$ маленьких собственных чисел.

Задача. Для ряда $f_j = 2 \exp(-0.01j) + \sin(2\pi j/7) + 1.5 \cos(2\pi j/17)$, $N = 120$, $L = 34$ с шумом провести анализ методом «Гусеница»-SSA. Изобразить собственные числа, первые 8 собственных векторов (отдельно и попарно) и вычислить отклонение восстановленного ряда по 5 ГК от незашумленного ряда.

Метод «Гусеница»-SSA для полей

Рассмотрим теперь модификацию метода «Гусеница»-SSA для обработки изображений (двумерных случайных полей). Пусть дана матрица Y размера $N_1 \times N_2$. Выберем некоторый прямоугольник размера $L_1 \times L_2$ ($1 < L_k < N_k/2$), который мы будем называть окном. Перенумеровав элементы этого окна построчно, получим вектор из матрицы. В начале окно разместим в левом верхнем углу матрицы Y . Будем двигать окно вправо. Когда доходим до правого края матрицы Y , спускаемся на одну позицию вниз и двигаемся

опять слева направо и т.д. Во время движения содержимое окон превращаем в столбцы траекторной матрицы $\mathbb{X}$. Далее проводим сингулярное разложение матрицы $\mathbb{X}$ и восстановление по нескольким первым главным компонентам.

Задача. Для картинки (100×100 пикс) построить восстановление по 5, 10, ..., 40 ГК. Результаты записать в один файл.

Классификация изображений на базе метода «Гусеница»-SSA

Пусть даны два изображения $Y1$ и $Y2$ — две матрицы одинаковых размеров. Фиксируем некоторое окно L . Построим для каждого изображения траекторные матрицы $\mathbb{X}_1$ и $\mathbb{X}_2$. Нормируем их на константы так, чтобы сумма собственных чисел матриц $S_i = \mathbb{X}_i \mathbb{X}_i^T$ была равна 1. Введем матрицу $\mathbb{X}$, составленную из матриц $\mathbb{X}_i$, стоящих друг над другом. Нетрудно видеть, что собственные числа матрицы

$$S = \mathbb{X} \mathbb{X}^T = \begin{pmatrix} S_1 & \mathbb{X}_1 \mathbb{X}_2^T \\ \mathbb{X}_2 \mathbb{X}_1^T & S_2 \end{pmatrix}$$

для одинаковых изображений в два раза больше собственных чисел матрицы S_1 . На этом свойстве собственных чисел мы введем меру близости двух изображений. А именно, пусть $\lambda_1^{(j)}, \dots, \lambda_r^{(j)}$ есть собственные числа S_j и $\lambda_1, \dots, \lambda_{2r}$ есть собственные числа S . Например, в качестве меры близости можно взять величину

$$d(Y1, Y2) = \sum_{i=1}^m \left(\lambda_i - \lambda_i^{(1)} - \lambda_i^{(2)} \right)^2$$

при некотором $m \leq r = L_1 L_2$. Будем считать, что два изображения одинаковы при $d(Y1, Y2) < c$ и различны при $d(Y1, Y2) > c$, где c — некоторая константа, которая должна быть определена на «обучающей» выборке изображений.

Задача. Построить по 8 разным черно-белым изображениям (12×12 пикс) букв А и С. На основе набора их зашумленных образов построить дискриминантную функцию на основе SSA, которая принимает два значения: одинаковые буквы или разные. Определить процент правильного распознавания для других зашумленных образов.

2.8. Оптимальные планы эксперимента

Пусть результаты эксперимента $y_1, \dots, y_N \in \mathbb{R}$ описываются уравнением

$$y_j = \eta(x_j, \beta) + \varepsilon_j, \quad j = 1, \dots, N,$$

где условия проведения эксперимента $x_1, \dots, x_N \in \mathcal{X}$ — точки некоторого множества планирования $\mathcal{X}$, $\eta(x, \beta)$ — известная вещественная функция, $\beta = (\beta_1, \dots, \beta_m)^T \in \Omega$ — вектор неизвестных параметров, N — общее число измерений, ошибки измерений $\varepsilon_1, \dots, \varepsilon_N$ — некоррелированные случайные величины такие, что $\mathbf{E}\varepsilon_j = 0$, $\mathbf{E}\varepsilon_j^2 = \sigma^2$, $j = 1, \dots, N$.

Под непрерывным планом эксперимента понимается дискретная вероятностная мера

$$\xi_N = \begin{pmatrix} x_1 & \dots & x_n \\ w_1 & \dots & w_n \end{pmatrix},$$

где $x_i \neq x_j$, $\sum_{j=1}^n w_j = 1$, $w_j \geq 0$, n — число различных точек в плане, x_j — условия проведения эксперимента, w_j — весовые коэффициенты. При практическом использовании непрерывных планов следует проводить приблизительно $w_j N$ измерений в точке x_j , $j = 1, \dots, n$.

Матрица

$$M(\xi) = M(\xi, \beta) = \int_{\mathcal{X}} f(x, \beta) f^T(x, \beta) \xi(dx)$$

называется информационной матрицей, где

$$f(x) = f(x, \beta) = \left(\frac{\partial \eta}{\partial \beta_1}(x, \beta), \dots, \frac{\partial \eta}{\partial \beta_m}(x, \beta) \right)^T$$

есть вектор регрессии. Известно, что асимптотическая дисперсионная матрица оценок параметров метода наименьших квадратов пропорциональна $M^{-1}(\xi)$. Более подробно о теории планирования эксперимента можно прочитать в [1, 7].

Локально D -оптимальный план $\xi^* = \xi^*(\beta)$ является решением задачи

$$(\det M(\xi))^{1/m} \rightarrow \sup_{\xi \in \Xi}.$$

Этот критерий оптимальности соответствует требованию минимизации объема доверительного эллипсоида. Для измерения качества применения некоторого плана ξ для значения параметров β есть понятие эффективности

$$\text{eff}_D(\xi, \beta) = \frac{(\det M(\xi, \beta))^{1/m}}{\sup_{\zeta} (\det M(\zeta, \beta))^{1/m}} = \frac{(\det M(\xi, \beta))^{1/m}}{(\det M(\xi^*(\beta), \beta))^{1/m}}.$$

Важным элементом теории оптимального планирования являются теоремы эквивалентности, которые устанавливают необходимые и достаточные условия оптимальности плана. Для D -критерия справедлива следующая теорема.

Теорема 2.1. *План ξ_D^* является D -оптимальным тогда и только тогда, когда функция $d(x, \xi_D^*) = f^T(x)M^{-1}(\xi_D^*)f(x)$ удовлетворяет неравенству*

$$\max d(x, \xi_D^*) \leq m.$$

Кроме того, $d(x_i^*, \xi_D^*) = m$, где x_i^* — точка плана ξ_D^* .

При практическом применении мы не знаем истинного значения β^* параметров, чтобы использовать план $\xi^*(\beta)$, а знаем лишь его некоторую оценку $\hat{\beta}$ (априорно или из предварительных экспериментов). Более прагматичным является использование максиминных планов. Пусть Ω есть некоторое множество возможных значений параметров. Максиминный D -оптимальный план является решением задачи

$$\min_{\beta \in \Omega} \text{eff}_D(\xi, \beta) \rightarrow \max_{\xi}.$$

Для максиминных планов также есть теоремы эквивалентности.

Теорема 2.2. *План ξ^* является максиминным D -оптимальным тогда и только тогда, когда существует распределение π^* на множестве*

$$N(\xi^*) = \left\{ \tilde{\beta} \in \Omega \mid \text{eff}_D(\xi^*, \tilde{\beta}) = \min_{\beta \in \Omega} \text{eff}_D(\xi^*, \beta) \right\}$$

такое, что неравенство

$$\int_{N(\xi^*)} f^T(x, \beta)M^{-1}(\xi^*, \beta)f(x, \beta)d\pi^*(\beta) \leq m \quad (1)$$

выполняется для всех $x \in \mathcal{X}$. Кроме того, равенство в (1) достигается во всех точках максиминного D -оптимального плана.

Множество $N(\xi)$ — это множество тех значений параметров, на которых достигается минимум эффективности для плана ξ .

Задача. Построить максиминный D -оптимальный план для модели $\eta(x, \beta) = a_1 e^{-b_1 x} + a_2 e^{-b_2 x}$, $x \in \mathcal{X} = [0, \infty)$, и множества $\Omega = \{a_1 = 1, b_1 = 4, a_2 = 2, b_2 \in [1 - z, 1 + z]\}$ при $z = 0.2$ и выполнить проверку по теореме эквивалентности.

2.9. Нейронные сети

Под нейронными сетями (НС) подразумеваются вычислительные структуры, которые ассоциируются с простыми биологическими процессами мозга. Элементарным преобразователем в данных сетях является (искусственный) нейрон, названный так по аналогии с биологическим прототипом. В состав нейрона входят умножители (синапсы), сумматор и нелинейный преобразователь. Синапсы осуществляют связь между нейронами и умножают входной сигнал на число, характеризующее силу связи. Сумматор выполняет сложение сигналов, поступающих по синаптическим связям от других нейронов. Нелинейный преобразователь реализует нелинейную функцию одного аргумента — выхода сумматора. Эта функция называется "функцией активации". Математическая модель нейрона описывается соотношениями

$$y = f(s), \quad s = \sum_{i=1}^n w_i x_i + b,$$

где y — выходной сигнал нейрона, f — функция активации, s — результат суммирования, n — число входов нейрона, w_i — вес i -го синапса, b — смещение, x_i — значение на i -м входе. Наиболее употребительными являются функции из следующего перечня: пороговая $f(s) = \mathbf{1}_{[0, \infty)}(s)$, знаковая $f(s) = \text{sign}(s)$, логистическая $f(s) = 1/(1 + e^{-s})$, полулинейная $f(s) = s_+ = s \mathbf{1}_{[0, \infty)}(s)$, линейная $f(s) = s$, радиальная $f(s) = e^{-s^2}$, линейная с насыщением, сигмоидальная $f(s) = (e^s - e^{-s})/(e^s + e^{-s})$, треугольная $f(s) = (1 - |s|)_+$. В пакете MATLAB им соответствуют функции `hardlim`, `hardlims`, `logsig`, `poslin`, `purelin`, `raddbas`, `satlin`, `satlins`, `tansig`, `tribas`.

Нейронная сеть — это набор нейронов, соединенных между собой. Как правило, функции активации нейронов фиксируются, а веса и смещения являются параметрами НС. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы — как

внешние выходы сети. Подавая любые числа на входы сети, мы получаем какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора X в выходной вектор Y , причем это преобразование задается весами сети. Практически любую задачу можно свести к задаче, решаемой нейросетью. Построение НС состоит из двух этапов: выбор типа (топологии) сети и подбор весов (обучение) сети. На первом этапе следует определить сколько нейронов мы хотим использовать, какие функции активации взять, как соединить нейроны между собой. По топологии сети можно разделить на полносвязные, многослойные, слабосвязные сети, сети без обратных связей и с обратными связями. Обработка взаимодействия всех нейронов ведется послойно. В пакете MATLAB реализованы наиболее популярные и изученные архитектуры. Теоретической основой для построения НС является следующее утверждение.

Предложение 2.1. *Для любого множества пар входных-выходных векторов произвольной размерности $\{(X_k, Y_k)\}_{k=1}^N$ существует двухслойная однородная НС с последовательными связями, с сигмоидальной передаточной функцией и конечным числом нейронов, которая для каждого входного вектора X_k формирует выходной вектор Y_k .*

Другой результат говорит о том, что НС являются универсальными аппроксимирующими свойствами.

Предложение 2.2. *Любая непрерывная функция на замкнутом ограниченно множестве может быть равномерно приближена функциями, вычисляемыми нейронными сетями, если функция активации нейрона дважды непрерывно дифференцируема.*

На втором этапе построения НС нам следует обучить выбранную НС, т.е. подобрать такие значения ее весов, чтобы сеть работала нужным образом. Обучение НС производится следующим образом. Поочередно примеры из базы данных подаются на вход НС. Если полученный ответ отклоняется далеко от требуемого значения, то подстраиваются веса сети. Если ошибка мала, для всех примеров, мы говорим, что НС обучена. Оказывается, что после многократного предъявления примеров веса НС стабилизируются, причем НС дает правильные ответы на все (или почти все) примеры из базы данных. При решении ряда задач (в частности, для кластеризации)

обучение НС происходит «без учителя». Таким образом, обучение НС превращается в задачу многомерной оптимизации большой размерности, в общем случае это многоэкстремальная невыпуклая задача. Для решения этой задачи могут быть использованы алгоритмы локальной оптимизации с вычислением производных первого и/или второго порядка (`trainbfg`, `trainbr`, `traincgb`, `traincgf`, `traincgp`, `traingd`, `trainwb`, `traingdx`). Важно отметить, что качество обучения НС напрямую зависит от количества примеров в обучающей выборке, а также от того, насколько полно эти примеры описывают данную задачу.

Основными недостатками аппарата НС являются отсутствие строгой теории по выбору структуры НС и практическая невозможность извлечения приобретенных знаний из обученной НС. НС практически всегда есть «черный ящик» для исследователя.

Нейронные сети для аппроксимации

Для восстановления значений $y = \eta(x)$ в произвольных точках по набору пар (x_i, y_i) , $i = 1, \dots, n$, MATLAB имеет специальную функцию `newgrnn(P,T,spread)`, которая создает двухслойную нейронную сеть. Параметр `spread` задает отклонение; чем больше это значение, тем глаже будет аппроксимация. Например:

```
P = [-1 -0.8 -0.5 -0.2 0.1 0.3 0.7 0.9 1]
T = P.^2+0.2*randn(size(P))
net = newgrnn(P, T, 0.05);
nP = [-0.9 -0.3 0 0.4 0.8]
nY = sim(net,nP) % опрос НС в заданных точках
```

Функция `newrb(P,T,goal,spread)` создает двухслойную радиальную нейронную сеть. Аргумент `goal` задает среднеквадратичную ошибку на обучающей выборке. Например:

```
net = newrb(P,T,0.0.9);
nY = sim(net,nP)
```

Для построения линейной регрессии следует воспользоваться линейной НС, которая задается функцией `newlin(PR,S,LD,LR)`, где `PR` есть матрица минимальных и максимальных значений, `S` — число элементов в выходном векторе, `LD` — вектор входной задержки, `LR` — коэффициент обучения, Например:

```
P = sort(5*rand(1,10));
T = 2*P - 1+0.3*randn(size(P));
```

```

maxlr = maxlinlr(P,'bias'); % вычисл. коэфф. обучения
net = newlin(minmax(P),1,[0],maxlr);
net = train(net,P,T);
nP = [1:0.4:4];
plot(P,T,'b',nP,sim(net,nP),'r')

```

Нейронные сети для прогнозирования процесса

Для построения линейного прогноза по предыдущим L значениям воспользуемся функцией `newlind`, которая строит линейную НС:

```

t = 0:0.4:80;
N = length(t); N2 = ceil(N/2);
x = [sin(t([1:N2])) sin(t([N2+1:end])/2)/2];
L = 7;
P = zeros(L,N);
for i = 1 : L
    P(i,i+1:N) = x(1,1:N-i);
end
net = newlind(P,x);
plot(t,x,'b',t,sim(net,P),'r')

```

Для построения адаптивного линейного прогноза следует использовать линейную НС с адаптивной настройкой весов НС. Корректировка весов и смещений производится после предъявления очередного вектора из обучающей выборки. Адаптивная настройка удобна при работе в реальном времени. Например:

```

T = con2seq(x); % преобр. матрицы в массив ячеек
net = newlin(minmax(x),1,[1:L],0.1);
[net,y,resid] = adapt(net,T,T);
plot(t,x,'b',t,cat(2,y{:}),'r')

```

Нейронные сети для кластеризации

Для выявления центров кластеров служит слой Кохонена, который является однослойной НС, в которой выход каждого нейрона является суммой взвешенных элементов входных сигналов. Обучение НС происходит без учителя. Например, рассмотрим двумерный случай:

```

clusters = 5; % пусть будет 5 кластеров по 10 точек
points = 10;
sigma = 0.03;

```

```

C = rand(2, clusters);
P = [ ];
for i = 1 : points
    P = [P, C];
end
P = P + sigma * randn(size(P));
% создадим НС для выявления заданного числа кластеров
net = newc(minmax(P), clusters, 0.1);
net.trainParam.epochs = 500;
net = init(net); % инициализация значений весов и смещений
net = train(net, P); % обучаем НС
centres = net.LW{1};
plot(P(1,:), P(2,:), '+' ) % рисуем исх. данные
hold on % рисуем центры кластеров
plot(centres(:,1), centres(:,2), 'or')
y = sim(net, [0.5; 0.5]) % какому кластера принадлежит точка

```

Нейронные сети для распознавания

Рассмотрим следующую постановку задачи. Построить НС для распознавания изображений трех букв (12×12 пикс). Определить процент правильного распознавания для зашумленных изображений.

Для решения задачи напомним сценарий:

```

let = {'a', 'c', 'f'};
YP = [ ]; G = [ ];
for i = 1 : 7*3
    filename = [let{ceil(i/7)}, '0', num2str(1+mod(i-1,7)), '.jpg'];
    Y = imread(filename, 'jpg'); Y = Y(:,:,1);
    YP = [YP, double(Y(:))]; % добавляем изобр. как столбец
    G = [G, ceil(i/7)]; % добавляем номер буквы
end
YT = ind2vec(G);
[R, Q] = size(YP);
[S2, Q] = size(YT);
S1 = 10; % число нейронов в первом слое
sigma = 155;
net = newff([min(YP,[ ],2), max(YP,[ ],2)+sigma], [S1 S2], ...
    {'logsig', 'logsig'}, 'traincgp');
net.LW{2,1} = net.LW{2,1} * 0.01;
net.b{2} = net.b{2} * 0.01;

```

```

goal = 0.05;
net.performFcn = 'sse';
% критерий остановки по средне-квадратичной ошибке
net.trainParam.goal = goal; % значение остановки
net.trainParam.show = 20;
% частота отображения промежуточных результатов
net.trainParam.epochs = 200;
% максимальной число итераций обучения
net.trainParam.mc = 0.95; % параметр обучения
[net, tr]=train(net,YP,YT); % обучаем НС
if(tr.perf(end)>goal)
    [net, tr]=train(net,YP+sigma*rand(size(YP)),YT);
end
if(tr.perf(end)>goal)
    [net, tr]=train(net,YP+sigma*rand(size(YP)),YT);
end
a = sim(net,YP) % проверка НС
a = sim(net,YP+sigma*rand(size(YP)))
corrclassif = sum(sum(round(a)==YT))/prod(size(a))

```

Как видно из примеров, проблема синтеза НС сильно зависит от решаемой задачи, дать общие рекомендации затруднительно. В большинстве случаев приемлемый вариант получается на основе интуитивного подбора. Для решения некоторых типов задач уже существуют оптимальные архитектуры НС. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную задачу синтеза новой архитектуры.

Более подробную информацию по структурам, алгоритмам обучения и использованию НС можно найти в [4, 5].

Литература

1. *Ермаков С.М., Жиглявский А.А.* Математическая теория оптимального эксперимента. М., 1987.
2. *Главные компоненты временных рядов: метод «Гусеница»* / Под ред. Д.Л. Данилова, А.А. Жиглявского. СПб., 1997.
3. *Голяндина Н.Э.* Метод «Гусеница»-SSA: анализ временных рядов: Учеб. пособие. СПб., 2004.
4. *Круглов В.В., Дли М.И., Голунов Р.Ю.* Нечеткая логика и искусственные нейронные сети. М., 2001.
5. *Круглов В.В., Борисов В.В.* Искусственные нейронные сети. Теория и практика. М., 2001.
6. *Мелас В.Б.* Общая теория функционального подхода к оптимальному планированию эксперимента. СПб., 1999.
7. *Пепелышев А.Н.* Некоторые современные методы исследования оптимальных планов эксперимента: Учеб. пособие. СПб., 2006.
8. *Потемкин В.Г.* Система инженерных и научных расчетов МАТЛАБ 5.X. Т. 1, 2. М., 1999.
9. *Рао С.Р.* Линейные статистические методы и их применение. М., 1968.
10. *Getting started with MATLAB.* pdf-документ.

Содержание

Содержание

Предисловие	3
1 Основы программирования в MATLAB	4
1.1 Выражения	5
1.2 Работа с матрицами	6
1.3 Циклы и условные выражения	10
1.4 Строки, массивы и структуры	12
1.5 Графика	14
1.6 Сценарии и функции	16
1.7 Функции от функций	19
1.8 Работа с полиномами	19
1.9 Решение дифференциальных уравнений	20
1.10 Работа с файлами	21
2 Примеры решения задач	22
2.1 Марковские цепи	22
2.2 Вычисление интегралов методом Монте-Карло	23
2.3 Фракталы	24
2.4 Имитационное моделирование	25
2.5 Оценки максимального правдоподобия	25
2.6 Спектральный анализ и фильтрация	30
2.7 Метод «Гусеница»-SSA	34
2.8 Оптимальные планы эксперимента	38
2.9 Нейронные сети	40
Литература	46

Учебное издание

Андрей Николаевич Пепельшев

Статистические вычисления в пакете MATLAB

Учебное пособие

Зав. редакцией	<i>Г.И. Чередниченко</i>	Редактор	<i>Ф.С. Бастиан</i>
Обложка	<i>А.В. Калинин</i>	Техн. редактор	<i>Л.Н. Иванова</i>

Подписано в печать 21.08.2006.

Ф-т 60×84/16. Усл. печ. л. 3,72.

Уч.-изд. л. 3,52. Тираж 110 экз. Заказ N

РОПИ СПбГУ. 199034, С.-Петербург, Университетская наб., 7/9.

Типография Издательства СПбГУ. 199061, С.-Петербург, Средний пр., 41.