Лабораторная работа №1

Введение в базовые операции

Цель этой лабораторной работы состоит в ознакомлении с основными средствами, которые Matlab предоставляет для эффективной и быстрой работы с базовыми типами данных — в первую очередь, с матрицами.

M-язык. Основным языком программирования, используемым в среде Matlab, является M-язык. Это проприетарный мультипарадигмный интерпретируемый язык со слабой динамической типизацией. Он похож на языки C/C++ и Java. Вот пример кода на нём:

```
N = 10;  % Процент обозначает комментарий, как // в C++
% Создание массива размером в N строк и 1 столбцов, каждый элемент которого равен нулю:
a = zeros(N,1);
for i = 1:N % Аналог в C: for(double i = 1; i <= N; i+=1)
    % обращение к i-му элементу массива; индексация начинается с 1 (а не с 0, как в C)
    a(i) = 10*i + 1;
end
```

Обратите внимание на синтаксис оператора цикла `for`. Кроме того, заметьте, что в приведённом фрагменте каждая строчка заканчивается точкой с запятой. В отличие от языков C или C++, её наличие не является обязательным. Если её не поставить, то результат выполнения команды будет выведен в командное окно. Так, например, команда `b = 20; c= 22; a = b+c` выведет в командное окно «`a = 42`». Это может оказаться полезным на этапе отладки. Важным элементом идеологии M-языка является минимизация использования циклов за счёт использования средств среды. Так, цикл в нашем примере можно было бы выполнить одной командой `a = 10*(1:N) + 1`. Здесь фундаментальную роль играет двоеточие. Почитайте в справочной системе Matlab статью «colon (:)» (просто вбейте слово colon в окно поиска этой системы).

M-язык является интерпретируемым (как shell, язык командных скриптов *NIX-систем, или Python), а не компилируемым (как C/C++, ассемблер или Fortran). Иными словами, между текстом программы и её запуском отсутствует промежуточный объект (бинарный файл). За счёт этого запуск осуществляется быстрее (не тратится время на компиляцию и компоновку), но теряется производительность[1].

Работа системы Matlab состоит в последовательном исполнении команд M-языка. Команды могут выполняться по одной (через командное окно, см. ниже) или же блоками (скриптами). Скрипт — это набор команд M-языка, записанных в файл, традиционно имеющий расширение `.m`.

Рабочее окружение. Набор всех существующих на данный момент переменных называется *рабочим окружением* (workspace). Важной особенностью является сохранение рабочего окружения после завершения выполнения команд или скрипта. Иными словами, если в командой строке выполнить команду `a=5`, то, пока не будет вызвана команда, удаляющая переменную `a`, она будет видна для всех других команд, функций и скриптов, запущенных потом. Это правило распространяется и на циклы: так, код `for i = 1:10; a = 8; end;` добавит в рабочее окружение переменные `i` и `a`. Заметьте, что это будут две матрицы размером 1×1. Локальные переменные функций после окончания их работы из рабочего окружения удаляются. Все числовые переменные, если явно не указать иного, имеют тип матрицы с элементами типа `double`. Этот тип позволяет хранить без потери точности целые числа в диапазоне, как минимум, 32-х битного целого (`int` в большинстве реализаций C/C++). Обращение к элементам осуществляется при помощи круглых скобок, нумерация начинается с 1 (а не с 0, как в C). Для адресации матриц индексы идут через запятую, $A(3,2)$— элемент матрицы A на пересечении третьей строки и второй колонки. Кроме «двойной», можно использовать «линейную» нумерацию, в которой элементы матриц занумерованы по столбцам: так, у матрицы A размером 3×3 $A(2,3)$ есть то же самое, что и $A(6)$.

Обратите внимание, что забывание о «выживании» переменных после окончания выполнения скрипта, вместе с отсутствием требования явного объявления переменных, может привести к ошибкам. В M-языке не обязательно объявлять переменные перед их использованием, поэтому скрипт с кодом (предполагается, что переменная `res` ранее в программе не встречалась)

```
for i = 1:10
  res = [res, f(i)];
end
```

после первого выполнения создаст вектор `res` с 10 элементами. После второго выполнения `res` будет иметь 20 элементов, после третьего — 30, и так далее. Дело в том, что после окончания работы скрипта переменная `res` не удаляется и не обнуляется. Для удаления переменной[2] из рабочей области можно выполнить команду `clear имя_переменной`. Очистка всей рабочей области осуществляется с помощью команды `clear`

[1] Это замечание касается только программ, написанных на M-языке. В отличие от C / C++, где стандартные библиотеки реализованы на них же (или на ассемблере), «стандартная библиотека» Matlab — например, работа с матрицами — реализована не только на M-языке, но и на других языках, в том числе C, Fortran и Java, за счёт чего обладает высокой производительностью. M-язык лишь предоставляет высокоуровневый интерфейс для доступа к таким функциям.

[2] В Matlab существует набор переменных, существующих всегда, но не обязательно входящих в рабочее окружение. Попробуйте присвоить переменной `pi` значение 1, затем напишите `clear pi`, и помотрите, чему будет равно значение `pi`.

без параметров. Для просмотра имен переменных, находящихся в рабочей области, служит команда `whos`. Для просмотра информации об одной переменной можно воспользоваться `whos имя_переменной`.

Рабочее окружение можно экспортировать в файл. Для сохранения на диске необходимо ввести команду `save имя_файла`. Данные будут сохранены в файле с расширением `.mat` — это бинарный формат хранения данных[3]. Команды `save` и `load` могут работать с переменными и в других форматах, например, текстовых[4]. Выборочное сохранение переменных из рабочей области обеспечивается командой `save имя_файла имя_переменной`: save b.mat a (`save b.mat a-append` — добавление массив a в существующий файл `b.mat`). Загрузка всех данных из файла реализуется командой `load имя_файла`, а выборочная загрузка — командой `load имя_файла имя_переменной`.

Сохранение и загрузка рабочего окружения может быть полезна в ситуациях, когда необходимо прервать вычисления по каким-либо причинам, или когда необходимо хранить большие объемы входных или выходных данных. Кроме того, если для запуска программы требуется большое (более пяти) количество параметров, то бывает удобно записать эти параметры в `mat`-файл.

Типизация переменных. Другой важной особенностью M-языка является *слабая динамическая типизация* (в противовес строгой статической типизации языков C/C++). Тип каждой переменной не фиксирован и определяется в момент присвоения переменной значения. Таким образом, команда `a=5` создаст переменную типа «матрица размером 1×1 с элементами типа `double`», а `a = 'Hello, world!'` — строковую переменную. Последовательное выполнение этих команд не приведет к ошибке; переменная просто сменит свой тип. Хотя это упрощает запись кода, злоупотреблять этой возможностью не стоит. M-язык также берет на себя управление памятью под переменные — иными словами, нет необходимости явно выделять память под переменные (с помощью функций наподобие `malloc` или `new` в C и C++).

Из интерпретируемости и динамической типизации вытекает важная особенность M-языка: при запуске скрипта на исполнение он проверяется только на синтаксические, но не семантические ошибки. Так, скажем, код не будет запущен, если в нём нарушен баланс скобок или нарушен порядок записи операндов. Однако ошибки вроде неправильного типа переменной (скажем, использование матрицы в качестве индекса массива или строки в качестве аргумента функции `sqrt`) будут обнаружены только тогда, когда выполнение до них дойдёт. Другой пример: если, например, функция `f` не возвращает никакого значения (т.е. в C имела бы заголовок типа `void f();`), то команда `A = f();` успешно пройдёт на запуск, но вызовет ошибку в момент своего непосредственного исполнения. Таким образом в M-языке гораздо более распространенны ошибки времени выполнения (runtime error).

Стоит отметить, что M-язык поддерживает методы нескольких парадигм программирования: процедурной, объектно-ориентированнной, функциональной. Для наших лабораторных работ будут использоваться процедурные и функциональные методы.

Задание функций. Функции в M-языке могут быть именованные и анонимные. Для задания собственной именованной функции необходимо создать отдельный файл, имя которого совпадает с именем функции. С точки зрения «внешнего мира» файл будет предоставлять только эту функцию, хотя в нём могут быть объявлены еще какие-нибудь вспомогательные функции. Пример файла `testFunction.m`:

```
function val = testFunction(x)
   val = f(x); % Функции возвращает значение переменных, указанных в заголовке - здесь, val
end

function y = f(x) % функцию f снаружи не видно
   y = sin(x);
end
```

После создания такого файла в текущем рабочем каталоге функцию можно будет вызвать командой `testFunction(x)`. Более общий синтаксис задания функций выглядит так: `function [список возвращаемых значений] = имяФункции(аргументы)`. Обратите внимание на то, что типы аргументов не указываются, и вызов `testFunction('peace')` приведёт к ошибке. Контроль за соблюдением типов (и их свойств) возлагается на программиста. Например:

```
function [val1, val2] = testFunction(x)
   if ~isnumeric(x) || x < 0 % ~ - отрицание, как ! в C
     error('Value passed to function must be positive!')
   end
   val1 = sqrt(x);
   val2 = log(x);
end
```

[3]Более–менее открытый в том смысле, что существуют библиотеки для работы с ним для других языков.

[4]И даже с форматами MS Office, в первую очередь — xls и xlsx, правда, возможно, для этого требуется наличие на машине MS Office.

Анонимные функции определяются непосредственно в тексте программы, например, `f =(x) sin(x)`. Это добавит в рабочее окружение переменную `f`, с которой можно работать, как с обычной функцией: `val= f(2*pi)`. Подробнее о работе с такими функциями будет в дальнейших лабораторных работах.

Работа со скриптами. Настоящая лабораторная работа должна быть выполнена в виде скрипта. Для создания файла скрипта можно щелкнуть правой кнопкой по Current Directory, и выбрать New File → Script. После этого будет создан файл. Щелкните по нему, и он откроется во встроенном текстовом редакторе[5] По нажатию CTRL + ENTER скрипт будет запущен на исполнение. Обратите внимание, что в текстовом редакторе подсвечиваются предупреждения о фрагментах кода, которые Matlab считает не очень хорошими (оптимальными, надёжными, быстрыми...). Стоит стараться не допускать появления таких предупреждений.

M-язык допускает разбиение скриптов на блоки с помощью двойного процента[6]:

```
%% Блок 1

% Исходный код блока 1
...
%% Задание 2

% Исходный код блока 2
...
```

При нажатии CTRL + ENTER будет осуществлён запуск только того блока, где находится курсор. Для быстрого перемещения между блоками можно использовать комбинации клавиш CTRL + стрелки. Аварийно прервать выполнение скрипта можно, нажав CTRL+C в командном окне или редакторе.

Замечание. По умолчанию в дистрибутивах Matlab для *NIX-систем действует клавиатурная раскладка из редактора Emacs. Вернуть более привычную раскладку (сохранить — Ctrl+S, копировать — Ctrl+C, и так далее) можно через Home (File) → Preferences → Keyboard → Shortcuts.

Некоторые команды. В Matlab действуют все команды DOS и частично — команды POSIX (например, `ls`). То есть, если в командном окне набрать, например, команду `dir`, то в результате на экран будет выведено содержимое текущей директории. Кроме того, есть ряд команд, призванных помочь при разработке.

Работа с матрицами. Важно понимать, что среда Matlab представляет собой, в первую очередь, довольно гибкий интерфейс доступа к многочисленным вычислительным библиотекам. M-язык представляет собой язык команд этого интерфейса, и реализация непосредственно на нём алгоритмов почти всегда будет проигрывать в производительности вызову из него специализированных библиотек.

Как уже было сказано, основным типом данных в Matlab являются матрицы, и все вычисления, с ними связанные, производятся весьма быстро; поэтому сведение к ним прочих операций может оказаться весьма полезным. Так, скажем, гораздо быстрее вычислять скалярное произведение n-мерных векторов как `x'*y`, нежели «классическим» методом, через цикл `for` и аккумулирующую переменную. Большинство встроенных команд принимают на вход матрицы и действуют на них поэлементно; так, `sqrt(A)` вернёт матрицу $B = \{b_{ij}\}$, такую, что $b_{ij} = \sqrt{a_{ij}}$, но не матричный корень из A (т.е. такую матрицу S, что $SS = A$).

Поскольку, как уже упоминалось, M-язык представляет собой «высокоуровневую» надстройку над библиотеками «низкоуровневых» средств и методов, использование библиотечных функций практически всегда оказывается эффективнее написания конструкций на M-языке. Поэтому Matlab предлагает большой спектр встроенных команд для векторов и матриц, позволяющих не прибегать к методам M-языка. К таким командам относятся команды `mean, median, sum, prod, flipud` и многие другие (см. список в конце инструкции).

О вычислениях. Прежде, чем перейти непосредственно к программированию, необходимо сделать несколько замечаний о самом предмете численных методов безотносительно среды, в которых они используются. Стоит помнить, что Matlab — в первую очередь, среда для решения задач *численными* методами, а не символьными[7]. При этом важно помнить, что все числа с плавающей точкой представлены в машинной арифметике с ошибкой. Отсюда следует, что результаты вычислений, в теории абсолютно совпадающие, на практике будут различаться и будут зависеть от способа, которым были получены. Например, логическое сравнение[8] `sqrt(2) == 4. ^ (1/4)` будет ложным, потому как его операнды, равные на бумаге, в памяти компьютера имеют различное представление. С другой стороны, выражение `abs(sqrt(2) - 4.^(1/4))` оказывается равным $2.22 \cdot 10^{-16}$, т.е. «очень маленьким» по сравнению с исходными числами. Это важное свойство устойчивых алгоритмов: «небольшие» отклонения и погрешности оказывают «небольшое» влияние на результат, и, имея с ними дело, мы можем ожидать, что ошибка будет пропорциональна вносимой погрешности[9].

[5] По умолчанию текстовый редактор открывается в отдельном окне, что может не всегда быть удобным. Если щелкнуть по небольшой стрелочке в верхнем правом углу окна, то он окажется «встроен» в основное окно Matlab, так же как и Command Window, Workspace и другие окна. Эти окна можно передвинуть внутри основного окна, как Вам покажется удобнее.

[6] В Windows 8 этот удобный и полезный механизм, как правило, не работает.

[7] Хотя средства символьной арифметики в Matlab присуствуют, они играют вспомогательную роль и их изучение выходит за рамки нашего курса.

[8] Проверялось в Matlab R2013a.

[9] Именно поэтому так много внимания уделяется вопросам непрерывной зависимости от начальных данных вообще и корректности по Адамару в частности.

Отсюда следует важное правило: сравнение чисел с плавающей точкой на равенство через оператор `==` ни к чему хорошему не приведёт. Если мы хотим сравнить два числа, то стоит рассматривать модуль их разности. Соответственно, для векторов и матриц стоит сравнивать их нормы, см. команду `norm`.

При осуществления расчётов численного метода (как «явных», которые вы пишите сами, так и «неявных», которые вызываются из библиотеки) имеет смысл проверить, насколько точным оказалось полученное решение — вычислить *невязку*. Например, при решении системы уравнений $F(x) = y$ близость величины $\|F(x^*) - y\|$ к нулю даст представление о «пригодности» решения x^*.

Другим важным свойством численных алгоритмов является их эффективность, которая обычно оценивается по времени, которое уходит на его работу на входных данных заданного размера. Для грубой оценки времени работы функции произвольной функции `F` необходимо усреднить время её работы на нескольких запусках:

```
res = zeros(numTries,1); % Заранее выделить память
for i = 1:numTries
  x = getRandomData(i); % получить произвольные входные данные
  tic();
  F(x);
  res(i) = toc(); % записать время i-го запуска
end
disp(median(res))
```

Команда `median` возвращает медиану вектора. Использование медианы здесь предпочтительнее использования среднего арифметического (`mean`) из-за возможных выбросов на отдельных запусках в силу многозадачности операционной системы.

Сводка команд. Для полноразмерного изучения возможностей Matlab по работе с матрицами следует посмотреть соответствующий раздел справочного руководства (Getting Started → Working with Matrices). Обязательно прочитайте справочное руководство по следующим функциям:

Базовые манипуляции с векторами и матрицами: `zeros, ones, linspace, logspace, max, min, size, reshape, repmat, rot90, sortrows, diff, trapz, mean, median`

Визуализация: `plot, subplot`

Ввод-вывод: `str2num, num2str, disp, format, input, error`

Информация о числе элементов: `size, numel, length`

Случайные матрицы: `rand, randn`

Разработка: `edit`, `type`, `lookfor`, `help`, `iso`, `which`, `whos`, `clear`, `clc`.

Замер времени: `tic, toc`

Другие полезные команды: `ismember, cumsum, prod, flipud, norm, fliplr, ceil, round, sub2ind`.

(В последующих лабораторных работах вы узнаете много других интересных, увлекательных и полезных команд.)

После ознакомления можно приступить к выполнению лабораторной работы. Все манипуляции над матрицами и векторами должны делаться исключительно при помощи встроенных команд и матричных операций, использование циклов не допускается. Также не стоит пользоваться средствами функционального программирования вроде команды `arrayfun`. Результаты работы должны выводиться в командное окно. Каждое задание должно быть оформлено в виде блока скрипта.