**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»**

**Д.В. Капитанов,**
**О.В. Капитанова**

# Введение в SciLab

**Практикум**

Рекомендовано методической комиссией
Института экономики и предпринимательства
для студентов ННГУ, обучающихся по направлению подготовки
38.03.05 «Бизнес – информатика», квалификация (степень) - бакалавр

Нижний Новгород
2019

УДК 004.42
ББК Ж/О 973.26-018.2
К 20

К 20 Капитанов Д.В., Капитанова О.В. Введение в SciLab: практикум. - Нижний Новгород: Нижегородский госуниверситет, 2019. – 56 с.

Рецензент: кандидат физ.-мат. наук **Чубаров Г.В.**

В данном практикуме изложены основные принципы работы с системой компьютерной математики SciLab; перечислены наиболее важные команды и функции, а также приведены основные подходы к решению разнообразных вычислительных задач и построению математических моделей с помощью данного программного средства. Также в пособии представлен ряд заданий для самостоятельного выполнения.

Содержание пособия соответствует программе дисциплины «Применение систем компьютерной математики в экономико-математических исследованиях», преподаваемой студентам Нижегородского госуниверситета, обучающимся по направлению бакалавриата «Бизнес – информатика» (профиль подготовки «Аналитические методы и информационные технологии поддержки принятия решений в экономике и бизнесе»). Практикум также адресован всем желающим освоить работу и программирование в системе SciLab.

*Ответственный за выпуск:*
председатель методической комиссии Института экономики и предпринимательства ННГУ к.э.н., доцент **С.В. Едемская**

УДК 004.42
ББК Ж/О 973.26-018.2

# Содержание

# Введение

Целью освоения дисциплины «Применение систем компьютерной математики в экономико-математических исследованиях», преподаваемой студентам бакалавриата по направлению «Бизнес – информатика» (профиль подготовки «Аналитические методы и информационные технологии поддержки принятия решений в экономике и бизнесе»), является получение студентами основных навыков работы с системами компьютерной математики, например, SciLab, для выполнения расчетов и проведения исследований в различных областях математики и экономики. Процесс изучения дисциплины направлен на формирование у выпускника следующих компетенций:

- способность осуществлять разработку и исследование математических моделей поддержки принятия решений в экономике и бизнесе (ДПК-1);
- способность к конструированию программ и применению компьютерных моделей поддержки принятия решений в экономике и бизнесе (ДПК-2).

В настоящее время система компьютерной математики SciLab обладает достаточно большой популярность. Основным ее достоинством является, то, что это бесплатное программное обеспечение с открытым исходным кодом для инженеров и ученых. Пакет SciLab используется для моделирования, моделирования и анализа данных, в промышленных (Airbus, ArcelorMittal, Air liquide, Sanofi, Microchip и др.) и научно-исследовательских (Fraunhofer Institute, аэрокосмические агентства, …) компаниях.

Первый релиз программы был выпущен в 1994 году. В настоящее время количество ежемесячных загрузок по всему миру достигает 100 000. С начала 2017 года группа разработчиков программного обеспечения SciLab является частью ESI Group, которая является пионером и мировым лидером в области виртуального прототипирования, используя физику материалов.

В настоящем практикуме приводятся основы работы с системой компьютерной математики SciLab, базовые принципы и функции работы с матрицами, описаны способы построения различных двумерных и трехмерных графиков и диаграмм, перечислены типы данных, основы программирования на языке SciLab, функции для работы с символьными данными, а также правила работы с файлами. Каждая тема содержит набор заданий для самостоятельного решения в четырех вариантах.

# Тема 1. Введение в SciLab

По умолчанию после запуска пакета SciLab 5.5.2 на экране появляется комбинированное окно (рис.1), включающее 4 наиболее важные панели – **Командное окно**, **Обозреватель файлов**, **Обозреватель переменных** и **Журнал команд**. Границы окон системы изменяют свои размеры и перемещаются вместе с главным окном. Любое из окон можно свернуть или закрыть с помощью соответствующих кнопок.

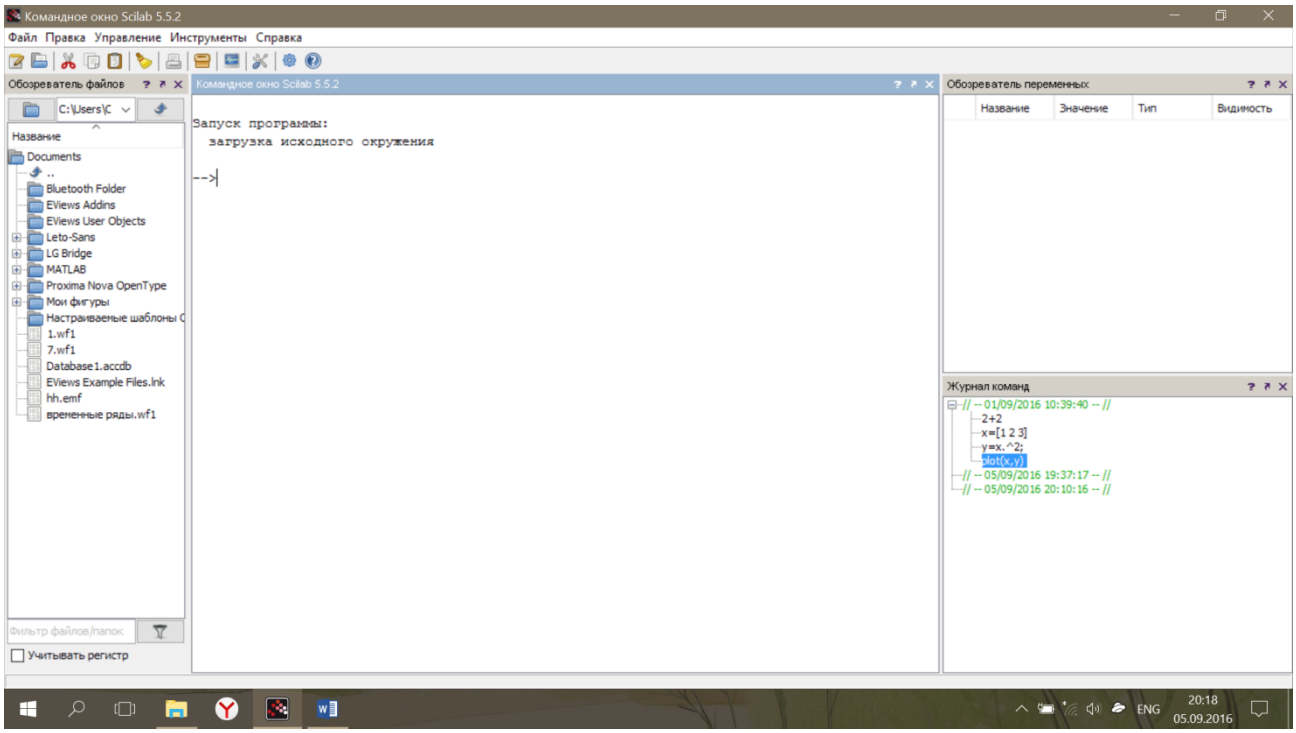


Рис.1. Рабочее окно программы SciLab

Основной панелью является **Командное окно**. В нем набираются команды пользователя, подлежащие немедленному выполнению. Здесь же выдаются результаты исполняемых команд.

Окно **Обозреватель переменных** отражает текущий набор переменных, заведенных пользователем в командном окне. Здесь можно увидеть их имена, размерность (значение), тип представленных данных и видимость переменной (локальная или глобальная). Такую же информацию можно получить в командном окне после выполнения команды *whos()*(но выведены также будут все библиотечные переменные и константы).

Окно **Журнал команд** хранит все команды, набираемые пользователем, однако в отличие от **Командного окна** здесь не выводятся сообщения системы и результаты вычислений.

Сеанс работы со SciLab обычно называют сессией (session).

В левом верхнем углу **Командного окна** находится комбинация знаков -->, которые символизируют начало строки. В этой строке можно набирать формулы или команды, удовлетворяющие синтаксису языка SciLab и завершающиеся нажатием клавиши**Enter**. Если строка не убирается, то для

перехода на новую строку надо набрать **…**, а затем нажать **Enter**, при этом команда выполняться не будет.

Значения всех промежуточных переменных SciLab сохраняет в рабочем пространстве. На выбор имен накладываются следующие ограничения:

- первый символ имени – это либо латинская буква, либо один из символов ‘%’, ‘_’, ‘#’, ‘!’, ‘$’, ‘?’;
- последующие символами могут быть латинские буквы, цифры или символы ‘_’, ‘#’, ‘!’, ‘$’, ‘?’;
- большие и малые буквы в именах различаются;
- длина может быть любой, но учитываются только первые24 символа, которые должны обеспечивать уникальность имени.

Для установки формата представления чисел для печати и вывода на экран используется команда:

*-->format([type],[long])*

гдеtype — символьная строка (*“v”* – для переменного формата по умолчанию; *“e”* – для e-формата); значение long определяет максимальное число знаков (по умолчанию 10). Команда:

*-->format()*

возвращает вектор текущего формата: первый элемент -- это тип формата (1, если *“v”*; 0, если *“e”*; второй элемент -- это число знаков.

В SciLab существует ряд предопределенных переменных (констант), которые задаются системой при загрузке, но могут быть переопределены:

- *%i*– мнимая единица (корень квадратный из -1);
- *%pi*–число π - 3.1415926...;
- *%eps*–относительная точность представления чисел с плавающей точкой ($2^{-52}$);
- *%inf*–машинная положительная бесконечность;
- *ans*– переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея;
- *%nan*– указание на нечисловой характер данных (Not-a-Number);
- *%e* – экспонента (2.7182818…)$
- *%F*или *%f*– логическая переменная ЛОЖЬ;
- *%t*или *%T* – логическая переменная ИСТИНА;
- *%s* и *%z* – переменные для задания полиномов: действительная и мнимая.

Символьная константа — это цепочка символов, заключенных в апострофы, например: ‘Hello my friend!’ или ‘2+3’. Если в апострофы помещено математическое выражение, то оно не вычисляется и рассматривается просто как цепочка символов. Так что ‘2+3’не будет возвращать число 5. Однако с помощью специальных функций преобразования символьные выражения могут быть преобразованы в вычисляемые.

Запись комплексных величин, используемых в формулах, напоминает общепринятый математический стандарт. Мнимая часть записывается как произведение действительного числа и мнимой единицы (*%i*). Когда полные комплексные числа используются в операциях умножения, деления или

возведения в степень, то для устранения неоднозначности их заключают в круглые скобки. В таблице 1 приведены функции для работы с комплексными числами.

Табл. 1. Функции для работы с комплексными числами

| Наименование функции | Выполняемое действие |
|---|---|
| *a=real(x)* | Выделение вещественной части комплексного числа*x* |
| *b=imag(x)* | Выделение мнимой части комплексного числа*x* |
| *x=complex(a,b)* | Построение комплексного числа *x*по паре вещественных чисел, a – действительная часть, b – мнимая часть |
| *y=conj(x)* | Вычисление *y* – сопряженного комплексного числа для числа *x* |
| *y=imult(x)* | Умножение числа на мнимую единицу, которое позволяет избежать проблем, возникающих при записи*y=%i*x*, например, когда x содержит «особые» числа с плавающей запятой, такие как *%inf* и *%nan*. |
| *t=isreal(x)* | Проверка, имеет ли переменная вещественные или комплексные элементы. Если x хранится как вещественная переменная, то возвращает значение «истина»; если x имеет мнимую часть (даже нулевую), то возвращает переменную «ложь». |

В таблицах 2 и 3 приведены базовые арифметические и логические операции, а также элементарные математические функции.

Табл. 2. Арифметические и логические операции

| Символ | Выполняемое действие |
|---|---|
| Операции над числовыми величинами | |
| + | покомпонентное сложение числовых массивов одинаковой размерности;<br>добавление скалярной величины к каждому элементу массива; |
| — | покомпонентное вычитание числовых массивов одинаковой размерности;<br>вычитание скалярной величины из каждого элемента массива; |
| * | умножение матриц в соответствии с правилами линейной алгебры;<br>умножение всех компонент массива на скаляр; |
| .* | покомпонентное умножение элементов массива одинаковой размерности; |
| / | деление скаляра на скаляр; покомпонентное деление всех элементов массива на скаляр;<br>$A/B=A*B^{-1}=A*inv(B)$ (A и B – квадратные матрицы одного порядка) |

| Символ | Выполняемое действие |
|---|---|
| ./ | покомпонентное деление элементов массива одинаковой размерности; |
| \ | левое матричное деление A\B=$A^{-1}$*B (A – квадратная матрица) |
| .\ | A.\B – покомпонентное деление элементов B на A (левое поэлементное деление) |
| ^ | возведение скаляра в любую степень;<br>вычисление целой степени квадратной матрицы; |
| ′ | вычисление сопряженной матрицы; |
| .′ | транспонирование матрицы; |
| Логические операции | |
| & | логическое умножение скаляров; логическое покомпонентное умножение массивов одинаковой размерности; логическое умножение массива на скаляр; |
| \| | логическое сложение скаляров; логическое покомпонентное сложение массивов одинаковой размерности; логическое сложение массива со скаляром; |
| ~ | логическое отрицание скаляра или всех элементов массива4 |
| == | проверка на равенство; |
| ~= или <> | проверка на неравенство; |
| > | проверка на «больше» |
| >= | проверка на «больше или равно» |
| < | проверка на «меньше» |
| <= | проверка на «меньше или равно» |

Табл. 3. Элементарные математические функции

| Категория функций | Наименования функций и их действия |
|---|---|
| Тригонометрические | Для аргумента в радианах:<br>*cos()*–косинус, *cotg()* – котангенс, *sin()* – синус, *tan()*– тангенс<br>Для аргумента в градусах:<br>*cosd()*–косинус, *cotd()* – котангенс, *sind()* – синус, *tand()* – тангенс |
| Обратные тригонометрические | Для результата в радианах:<br>*acos()* – арккосинус, *acot()* – арккотангенс, *asin()* – арксинус, *atan()* – арктангенс<br>Для результата в градусах:<br>*acosd()* – арккосинус, *acotd()* – арккотангенс, *asind()* – арксинус, *atand()* – арктангенс |
| Экспонента, логарифмы, корень | *exp()* – показательная функция (экспонента), *log()* – натуральный логарифм, *log2()* – логарифм по основанию 2, *log10()* – десятичный логарифм, *sqrt()* – квадратный корень |

| Категория функций | Наименования функций и их действия |
|---|---|
| Округления | *ceil()* – округляет до ближайшего большего числа, *fix()*или*int()* – просто отбрасывает дробную часть, *floor()* – до ближайшего меньшего числа, *round()* – округление до ближайшего целого |
| Дискретная математика | *gcd([p1 p2 ...])*– наибольший общий делитель для элементов вектора-строки, *lcm([p1 p2 ...])* – наименьшее общее кратное, *factor(x)* – разложение на множители числа *x*, *factorial(n)* – факториал *n*, *primes(x)* возвращает все простые числа в интервале между 1 и *x* включительно |
| Модуль числа | *abs()* |
| Знак числа | *sign()* |
| Остаток от деления | *k=modulo(n,m)* – симметричный арифметический остаток от деления n на m (для целых чисел): *k=n-m.*int(n./m)*, *k=pmodulo(n,m)* – положительный арифметический остаток от деления n на m (для целых чисел): *k=n-\|m\|.*floor(n./\|m\|)* |

Имена и значения переменных рабочего пространства можно сохранить в бинарный файл либо с помощь команды главного меню **Файл→ Сохранить окружение**, либо набрать команду в текущей строке:

*—>save(filename [,x1,x2,...,xn])*

*filename*–это строка, которая указывает имя файла и путь к нему. Файлы в SciLab имеют расширение .sod. В более старых версиях расширение .dat. Если не указывать переменные *x1,x2,...,xn*, то будут сохранены все переменные файлового пространства.

Для загрузки переменных начале следующей сессии достаточно выполнить команду load:

*—>load(filename [,x1,...,xn])*

Если команда load будет использована в середине сессии, то текущие значения переменных изменятся на загружаемые.

Для того чтобы очистить все рабочее пространство используется команда *clear* без параметров, но если указать список переменных, то она удалит только заданные переменные.

*—>clear*

*—>clear x y*

*—>clear('a','b','c')*

Команда *whos()* позволяет получить подробную информацию о переменных рабочего стола. Также может быть использована команда *who*, которая выводит только список имен переменных. Для просмотра значения любой переменной достаточно просто набрать ее имя и нажать клавишу Enter.

Стандартная команда *save(filename [,x1,x2,...,xn])* не позволяет сохранить на диск содержание сессии, поэтому если существует такая необходимость, то

следует воспользоваться специальной командой для ведения так называемого дневника сессии:

*—>diary(filename)*

или

*—> [id,filename] = diary(filename, ['new'|'append'])*

Эта команда ведет запись на диск всех команд в строках ввода и полученных результатов в виде текстового файла с указанным именем *filename*; *id* – это идентификатор (число) дневника.

Для окончания записи в файл используется команда:

*—>diary(filename, 'close')*

или

*—>diary(id, 'close')*

Приостановить запись можно с помощью команды:

*—>diary(filename, 'pause'|'off')*

или

*—>diary(id, 'pause'|'off')*

Возобновляется запись в дневник командами:

*—>diary(filename, 'resume'|'on')*

или

*—>diary(id, 'resume'|'on')*

Дневник сессии сохраняется в формате .txt и просмотреть его можно с помощью Блокнота.

Закончить работу с программой можно с помощью команды

*—>exit*

## *Задание 1*

Вариант 1

1. Вычислить:

*a*. $(-0.6)\cdot\left(0.0081^{\frac{1}{2}}+\left(1\frac{1}{9}\right)^{-2}\right):1\frac{5}{13}$; *b*. $\sqrt{0.04}-\left(\sqrt{7}-2\sqrt{2}\right)\left(\sqrt{8}+\sqrt{7}\right)$

2. С помощью команды *format*() вывести значение выражения *f* на экран в возможных форматах с разным количеством знаков.

$$f = 2\cdot\pi$$

3. Чему равно значение *e*?

4. Задать значения комплексных чисел x (с помощью *i*) и y (с помощью функции *complex()*). Найти сумму, разность, произведение, сопряженное. Выделить действительную и мнимую части. Проверить на комплексность.

$$x = 1.5 + 0.5i;\ y = 2.5 - 7i$$

5. Подсчитать значения косинуса, синуса, тангенса и котангенса для 8π/5и 288°.

6. Округлить числа всеми возможными способами: 1.1, 1.5, 1.8.

7. Вычислить остаток от деления двумя способами для чисел: 8 и -5.

8. Вычислить НОД и НОК для чисел: 12 и 27. Разложите указанные числа на множители. Найдите все простые числа, их не превышающие.

9. Сохранить переменные рабочего пространства в файле*tt.sod.*

10. Очистить сначала значения переменных *x* и *y*, а затем все рабочее пространство.

11. Загрузить переменные из файла *tt.sod*

12. Просмотреть информацию о переменных с помощью команд *whos()* и *who().*

13. Записать в дневник (файл *mydiary.txt*) следующие действия:

$$a = 2;$$
$$b = 3;$$
$$c = (a - b)/b$$

14. Посмотреть текст файла *mydiary.txt*

Вариант 2

1. Вычислить:

*a*. $3.5 \cdot \left(0.027^{\frac{1}{3}} - \left(1\frac{3}{7}\right)^{-1}\right) : \left(-2\frac{6}{11}\right);$ *b*. $\sqrt{0.09} - \left(\sqrt{11} - 2\sqrt{3}\right)\left(\sqrt{11} + \sqrt{12}\right)$

2. С помощью команды *format*() вывести значение выражения *f* на экран в возможных форматах с разным количеством знаков.

$$f = e^2$$

3. Чему равно значение $\pi$?

4. Задать значения комплексных чисел x (с помощью *i*) и y (с помощью функции *complex()*). Найти сумму, разность, произведение, сопряженное. Выделить действительную и мнимую части. Проверить на комплексность.

$$x = 8 + 5i;\ y = 7 + 0.8i$$

5. Подсчитать значения косинуса, синуса, тангенса и котангенса для 7π/11 и 115°.

6. Округлить числа всеми возможными способами: -1.1, -1.5, -1.8.

7. Вычислить остаток от деления двумя способами для чисел: -23 и 4.

8. Вычислить НОД и НОК для чисел: 14 и 35. Разложите указанные числа на множители. Найдите все простые числа, их не превышающие.

9. Сохранить переменные рабочего пространства в файле*tt.sod.*

10. Очистить сначала значения переменных *x* и *y*, а затем все рабочее пространство.

11. Загрузить переменные из файла *tt.sod*

12. Просмотреть информацию о переменных с помощью команд *whos()* и *who().*

13. Записать в дневник (файл *mydiary.txt*) следующие действия:

$$a = 2\pi;$$
$$b = \cos a$$
$$c = \sin a$$

14. Посмотреть текст файла *mydiary.txt*

Вариант 3

1. Вычислить:

*a.* $\left(0.48^0 + \left(1\frac{9}{16}\right)^{\frac{3}{2}} : 0.8^{-4}\right) \cdot 0.81^{-\frac{1}{2}}$; *b.* $\left(2-\sqrt{3}\right)^2\left(7+4\sqrt{3}\right) + 3\sqrt{12\frac{1}{4}}$

2. С помощью команды *format*() вывести значение выражения *f* на экран в возможных форматах с разным количеством знаков.

$$f = e^{\sqrt{3}}$$

3. Чему равно значение *eps*?

4. Задать значения комплексных чисел x (с помощью *i*) и y (с помощью функции *complex()*). Найти сумму, разность, произведение, сопряженное. Выделить действительную и мнимую части. Проверить на комплексность.

$$x = 2 - 3i;\ y = -8 + 4i$$

5. Подсчитать значения косинуса, синуса, тангенса и котангенса для π/13 и 14°.

6. Округлить числа всеми возможными способами:2.2, -2.5, 2.7.

7. Вычислить остаток от деления двумя способами для чисел: 12 и -7.

8. Вычислить НОД и НОК для чисел: 16 и 2. Разложите указанные числа на множители. Найдите все простые числа, их не превышающие.

9. Сохранить переменные рабочего пространства в файле*tt.sod.*

10. Очистить сначала значения переменных *x* и *y*, а затем все рабочее пространство.

11. Загрузить переменные из файла *tt.sod*

12. Просмотреть информацию о переменных с помощью команд *whos()* и *who().*

13. Записать в дневник (файл *mydiary.txt*) следующие действия:

$$a = 2;$$
$$b = \sqrt{a}$$
$$c = a + b$$

14. Посмотреть текст файла *mydiary.txt*

Вариант 4

1. Вычислить:

*a.* $\left(0.49^{-1.5} : \left(1\frac{3}{7}\right)^4 + 0.64^{-\frac{1}{2}}\right) \cdot 3\frac{1}{13}$; *b.* $\left(\sqrt{5}-2\right)^2\left(9+4\sqrt{5}\right) - 2\sqrt{5\frac{4}{9}}$

2. С помощью команды *format*() вывести значение выражения *f* на экран в возможных форматах с разным количеством знаков.

$$f = \pi \cdot e$$

3. Чему равно значение выражения: 1/∞?

4. Задать значения комплексных чисел x (с помощью *i*) и y (с помощью функции *complex()*). Найти сумму, разность, произведение, сопряженное. Выделить действительную и мнимую части. Проверить на комплексность.

$$x = 4 - 8i;\ y = 5 - 2i$$

5. Подсчитать значения косинуса, синуса, тангенса и котангенса для 14π/6 и 60°.

6. Округлить числа всеми возможными способами:-2.2, 2.5, -2.7.

7. Вычислить остаток от деления двумя способами для чисел: -18 и 8.

8. Вычислить НОД и НОК для чисел: 48 и 8. Разложите указанные числа на множители. Найдите все простые числа, их не превышающие.

9. Сохранить переменные рабочего пространства в файле*tt.sod.*

10. Очистить сначала значения переменных *x* и *y*, а затем все рабочее пространство.

11. Загрузить переменные из файла *tt.sod*

12. Просмотреть информацию о переменных с помощью команд *whos()* и *who().*

13. Записать в дневник (файл *mydiary.txt*) следующие действия:

$$a = 7;$$
$$b = \log a$$
$$c = a * b$$

14. Посмотреть текст файла *mydiary.txt*

# Тема 2. Матрицы. Операции с матрицами в SciLab

Оператор двоеточие используется для задания последовательности чисел:

—>*a:b:c* – задает последовательность чисел от a до c с шагом b. Если шаг равен 1, то значение b можно пропустить и записать a:c.

Для задания значений элементов матрицы используются квадратные скобки, в которых числовые данные отделяются друг от друга пробелами или запятыми (для задания чисел в строке) и точкой с запятой (для задания столбцов). Матрица задается по строкам.

Также матрицу можно задавать или изменять, используя индексное обозначение *A(i,j)*, где *i* указывает номер строки, а *j* – номер столбца. Нумерация строк и столбцов начинается с 1.

Для создания типовых матриц существует ряд специальных функций:

- Функции *zeros(n,m)* и *ones(n,m)* используются для заполнения квадратных или прямоугольных матриц нулями и единицами.
- С помощью функции *eye(n,m)* формируется единичная квадратная или прямоугольная матрица.
- Функции *rand(n,m,"uniform")* и *rand(n,m,"normal")* заполняют элементы матриц случайными числами с равномерным или нормальным распределением.
- Функция *testmatrix('magi',n)* задает магическую матрицу размера $n \times n$, у которой сумма всех столбцов, всех строк и даже диагоналей равна одному и тому же числу.

В SciLab существует функция для включения и исключения диагоналей в матрицу:

—> *X=diag(v,[k])*

где *v* – вектор; *k* – номер диагонали: *k=0* – главная диагональ; *k>0* – диагональ, расположенная выше главной; *k<0* – диагональ расположенная ниже главной; *X* – матрица, на *k*-ой диагонали которой расположен вектор *v*, остальная часть заполнена нулями.

Если эту функцию записать в виде *v=diag(X,k)*, то результатом будет вектор-столбец, состоящий из элементов k-ой диагонали матрицы X.

Ранее сформированные массивы могут участвовать в формировании новых в качестве их клеток: *D=[A; B+5 C]* – однако следует смотреть за соответствием размерностей матриц (блочный способ).

Функция *sysdiag(A,B,C,…)* объединяет матрицы по диагонали, остальные элементы заполняются нулями.

Конкатенацией называют объединение массивов, которое реализует функция cat:

—>*C=cat(dim,Al,A2,A3,A4,...)*– объединяет все входные массивы в соответствии со спецификацией размерности *dim* и возвращает объединенный массив; *dim*=1 –конкатенация по строкам входных аргументов, *dim=2* – конкатенация по столбцам входных аргументов.

Для формирования матриц и выполнения ряда матричных операций возникает необходимость удаления отдельных столбцов и строк матрицы. Для этого используются пустые квадратные скобки []. Удаление *i*-го столбца – *M(:,i)=[]*; удаление *i*-ой строки –*M(i,:)=[]*. Полностью очистить матрицу таким способом нельзя.

Для поиска обратной матрицы для квадратной матрицы *A* используется команда:

—>*inv(A)*

Для вычисления определителя матрицы *A* используется функция:

—>*det(A)*

Для вычисления ранга матрицы *A* используется команда:

—> *rank(A)*

Для вычисления суммы и произведения элементов массива *A* используются функции *sum(A)*и*prod(A)*, соответственно. Максимальный и минимальный элемент в матрице можно найти с помощью функций *max(A)* и *min(A).*

Решение систем линейных уравнений

Система линейных уравнений в матричном виде записывается:

$$A*X=B$$

где *A*– матрица коэффициентов; *X*– вектор-столбец неизвестных; *B*– вектор-столбец правых частей. Тогда решение системы линейных уравнений может быть записано в виде:

$$X=A^{-1}*B$$

В SciLab систему линейных уравнений можно решить тремя способами:

—>*X=inv(A)*B*

или

—>*X=A\B*

или с помощью функции:

—> *X=linsolve(A,-B)*

## *Задание 2*

Вариант 1

1. Задать матрицу $A = \begin{pmatrix} 2 & 1 & 3 \\ 5 & 10 & 2 \\ 1 & 4 & 3 \end{pmatrix}$.

2. Изменить значение элемента матрицы *A*, используя индексы: *A(2,2)=3.*

3. Создать квадратные матрицы размерности 4 из нулей, единиц, случайных чисел с нормальным и равномерным распределением, единичную матрицу и магический квадрат.

4. Задать матрицу *X*, где вектор *v* будет *k*-ой диагональю матрицы.

*v=(1 2 3 4), k=2*

5. Объединить в матрицах *M, N, L* матрицы *A* и *B* блочным способом, а также с помощью функций *sysdiag()* и *cat()*.

$$B = \begin{pmatrix} 5 & 2 & 0 \\ 7 & 3 & 0 \end{pmatrix}$$

6. Удалить из матрицы *B* третий столбец, из матрицы *L*– вторую строку.

7. Посчитать определители и ранги матриц *A* и *B*. Найти их обратные матрицы, а также сумму и произведение элементов, максимальный и минимальный элемент.

8. Решить систему линейных уравнений тремя способами:

$$2x_1 + 3x_2 + 5x_3 = 1$$
$$3x_1 + 7x_2 + 4x_3 = 3$$
$$x_1 + 2x_2 + 2x_3 = 3$$

Вариант 2

1. Задать матрицу $A = \begin{pmatrix} 1 & 10 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix}$.

2. Изменить значение элемента матрицы *A*, используя индексы: *A(1,2)=1*.

3. Создать квадратные матрицы размерности 2 из нулей, единиц, случайных чисел с нормальным и равномерным распределением, единичную матрицу и магический квадрат.

4. Задать матрицу *X*, где вектор *v* будет k-ой диагональю матрицы.

*v=(65 8 7), k=3*

5. Объединить в матрицах *M, N, L* матрицы *A* и *B* блочным способом, а также с помощью функций *sysdiag()* и *cat()*.

$$B = \begin{pmatrix} 3 & 2 & 0 \\ 8 & 5 & 0 \end{pmatrix}$$

6. Удалить из матрицы *B* третий столбец, из матрицы *L*– вторую строку.

7. Посчитать определители и ранги матриц *A* и *B*. Найти их обратные матрицы, а также сумму и произведение элементов, максимальный и минимальный элемент.

8. Решить систему линейных уравнений тремя способами:

$$5x_1 - 6x_2 + 4x_3 = 3$$
$$3x_1 - 3x_2 + 2x_3 = 2$$
$$4x_1 - 5x_2 + 2x_3 = 1$$

Вариант 3

1. Задать матрицу$A = \begin{pmatrix} 1 & 1 & 1 \\ 4 & 5 & 10 \\ 16 & 25 & 81 \end{pmatrix}$.

2. Изменить значение элемента матрицы *A*, используя индексы: *A(2,3)=9*.

3. Создать квадратные матрицы размерности 5 из нулей, единиц, случайных чисел с нормальным и равномерным распределением, единичную матрицу и магический квадрат.

4. Задать матрицу *X*, где вектор *v* будет *k*-ой диагональю матрицы.

*v=(5 3), k=4*

5. Объединить в матрицах *M, N, L* матрицы *A* и *B* блочным способом, а также с помощью функций *sysdiag()* и *cat()*.

$$B = \begin{pmatrix} 6 & 9 & 0 \\ 8 & 12 & 0 \end{pmatrix}$$

6. Удалить из матрицы *B* третий столбец, из матрицы *L*– вторую строку.

7. Посчитать определители и ранги матриц *A* и *B*. Найти их обратные матрицы, а также сумму и произведение элементов, максимальный и минимальный элемент.

8. Решить систему линейных уравнений тремя способами:

$$4x_1 - 3x_2 + 2x_3 = -4$$
$$6x_1 - 2x_2 + 3x_3 = -1$$
$$5x_1 - 3x_2 + 2x_3 = -3$$

Вариант 4

1. Задать матрицу $A = \begin{pmatrix} 1 & 5 & 25 \\ 1 & 7 & 49 \\ 1 & 8 & 10 \end{pmatrix}$.

2. Изменить значение элемента матрицы *A*, используя индексы: *A(3,3)=64*.

3. Создать квадратные матрицы размерности 3 из нулей, единиц, случайных чисел с нормальным и равномерным распределением, единичную матрицу и магический квадрат.

4. Задать матрицу *X*, где вектор *v* будет *k*-ой диагональю матрицы.

*v=(15 7 6 12), k=1*

5. Объединить в матрицах *M, N, L* матрицы *A* и *B* блочным способом, а также с помощью функций *sysdiag()* и *cat()*.

$$B = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \end{pmatrix}$$

6. Удалить из матрицы *B* третий столбец, из матрицы *L*– вторую строку.

7. Посчитать определители и ранги матриц *A* и *B*. Найти их обратные матрицы, а также сумму и произведение элементов, максимальный и минимальный элемент.

8. Решить систему линейных уравнений тремя способами:

$$5x_1 + 2x_2 + 3x_3 = -2$$
$$2x_1 - 2x_2 + 5x_3 = 0$$
$$3x_1 + 4x_2 + 2x_3 = -10$$

# Тема 3. Построение графиков на плоскости и в пространстве

При построении графиков функций в SciLab для задания значений по оси абсцисс – вектора *x* – удобно пользоваться командой:

—> *x=a:b:c;*

которая задает вектор значений от *a* до *c* с шагом *b*.

Для построения графиков функций на плоскости используется команда *plot()*, которая может иметь следующие виды синтаксиса:

—>*plot(y,<LineSpec>,<GlobalProperty>)*

—>*plot(x,y,<LineSpec>,<GlobalProperty>)*

—> *plot(x1,y1,<LineSpec1>,x2,y2,<LineSpec2>,...xN,yN,<LineSpecN>, <GlobalProperty1>,<GlobalProperty2>,..<GlobalPropertyM>)*

Если *y* – вектор, то *plot(y)* построит вектор *y*, где по оси абсцисс значения от *1* до *length(y)* (длина вектора *y*). Если *y* – матрица, то *plot(y)* построит каждый столбец матрицы в осях, где по оси абсцисс значения от 1 до *length(y)*.

Спецификация *plot(x,y)* строит график, где *x* – вектор значений аргументов, *y* – вектор соответствующих значений функции. Вектора должны иметь одинаковый размер. В случае, когда *y* – матрица, указанная команда построит каждый столбец матрицы. Количество строк матрицы *y* должно соответствовать размеру вектора *x*. В случае, когда *x* – тоже матрица, команда *plot(x,y)* построит графики соответствующих пар столбцов двух матриц (которые должны иметь одинаковую размерность).

Третий вариант синтаксиса (*plot(x1,y1,x1,y2,…)*) также позволяет строить график нескольких функций в одном окне.

По умолчанию все последующие команды *plot()* будут строить графики в том же, ранее открытом, окне. Для очистки графического окна используется команда *clf()*.

Вместо параметра *<LineSpec>* в одинарных кавычках указываются дополнительные параметры для оформления линий:

- стиль линии (табл. 4);
- цвет линии (при построении графика нескольких функций без указания их цветов, цвета перебираются по умолчанию, табл. 5);
- тип маркеров (если указать тип маркера без типа линии, то будут нарисованы только маркеры, табл. 6);

Табл. 4. Стили линий

| Символ | Стиль линии | Символ | Стиль линии |
|---|---|---|---|
| - | Сплошная линия (по умолчанию) | -. | Штрих пунктирная линия |
| : | Пунктирная линия | -- | Штриховая линия |

Табл. 4. Цвета линий

| Символ цвета | Цвет графика | Символ цвета | Цвет графика |
|---|---|---|---|
| y | желтый (Yellow) | g | зеленый (Green) |
| m | малиновый (Magenta) | b | синий (Blue) |
| c | бирюзовый (Cyan) | w | белый (White) |
| r | красный (Red) | k | черный (blacK) |

Табл. 5. Типы маркеров

| Символ | Маркер | Символ | Маркер |
|---|---|---|---|
| . | жирная точка | d | ромбик |
| o | кружок | v | треугольник вершиной вниз |
| x | крестик | ^ | треугольник вершиной вверх |
| + | плюс | < | треугольник вершиной влево |
| * | восьмиконечная снежинка | > | треугольник вершиной вправо |
| s | квадратик | p | пятиконечная звезда |
| | отсутствие маркера | | |

Значения указываются в произвольном порядке, но они должны восприниматься однозначно.

Необязательный параметр <*GlobalProperty*> позволяет выполнить глобальную настройку построения всех новых линий. Имеет вид пары ‘имя свойства’, ‘значение свойства’. Например, пара *‘markersize’,5* позволяет задать размер маркера в пунктах (здесь 5).

Для размещения легенды на графике можно воспользоваться командой:

*—>hl=legend([h,] string1,string2, ... [,pos] [,boxed])*

Здесь необязательный параметр h – это указатель на графическое окно. Строки *string* содержат подписи графиков в порядке построения. Параметр *pos* управляет размещением легенды в графическом окне:

- 1 – легенда размещается в правом верхнем углу поля графика (по умолчанию);
- 2 – легенда размещается в левом верхнем углу поля графика;
- 3 – легенда размещается в левом нижнем углу поля графика;
- 4 – легенда размещается в правом нижнем углу поля графика;
- 5 – интерактивное размещение мышкой;
- -1 – легенда размещается вне поля графика справа вверху;
- -2 – легенда размещается вне поля графика слева вверху;
- -3 – легенда размещается вне поля графика слева внизу;
- -4 – легенда размещается вне поля графика справа внизу;
- -5 – легенда размещается вне поля графика сверху слева;
- -6 – легенда размещается вне поля графика снизу слева.

Необязательный параметр *boxed* –это значение истина (*%t*) или ложь (*%f*), которое указывает нужно ли рисовать рамку вокруг легенды.

Для создания заголовка применяется команда *title(‘my title’)*.

Для добавления обозначений осей применяются команды *xlabel('X')*, *ylabel('Y')* и *zlabel('Z')*, где третья команда используется для пространственных графиков (3d).

Если в одном графическом окне необходимо отобразить несколько графиков в разных осях, то следует использовать функцию *subplot()*, которая позволяет разделить область рисования на несколько прямоугольных областей равного размера:

*—>subplot(row,col,cur);*

Первые два аргумента задают количество строк (*row*) и столбцов (*col*), третий параметр (*cur*) объявляет порядковый номер подобласти, в котором будет построен очередной график.

Для построения графиков функций, заданных в полярной системе координат, используется процедура:

*—>polarplot(phi,rho)*

где *phi* – угол, а *rho* – расстояние.

Процедура *fplot2d(x,f)* строит график функции *y=f(x)* без предварительного вычисления вектора *y*, где *x* – это вектор, а *f* – это либо функция SciLab, либо строка, описывающая функцию. Для задания такой функции используется команда:

*—>deff('[s1, s2, ...] = newfunction(e1, e2, ...)',text)*

где *e1, e2, ...* – входные переменные, *s1, s2, ...* – выходные переменные, *text* – матрица символьных строк.

Например, для построения графика функции *sin(x)+cos(x)* нужно задать следующие команды:

*—>x=0:0.1:2*%pi;*

*—>deff("[y]=f(x)","y=sin(x)+cos(x)")*

*—>fplot2d(x,f)*

Для построения трехмерных графиков используются функции *mesh(X,Y,Z)* и *surf(X,Y,Z)*. При этом *mesh()* создает каркасную поверхность, где цветные линии соединяют только заданные точки, а функция *surf()* вместе с линиями отображает в цвете и саму поверхность.

Для отображения функции двух переменных *Z=f(X,Y)*, создаются матрицы *X* и *Y*, состоящие из повторяющихся строк и столбцов. Затем эти матрицы используются для вычисления и отображения функции. Таким образом, перед построением трехмерных графиков следует задать сетку с помощью функции *[X,Y]=meshgrid(x,y)*, которая преобразует векторы *x* и *y* в двумерные массивы. После этого рассчитывается значение функции *Z*, используя значения *X* и *Y*.

Функция *colorbar(zmin,zmax)* строит рядом с полем графика столбик с цветовой гаммой, которая демонстрирует привязку цвета к значению аппликат z (от значения *zmin* до значения *zmax*, которые нужно определить).

Для отображения движения точки по траектории используется команда *comet(x,y)*. При этом движущаяся точка напоминает ядро кометы с хвостом. Также можно построить анимированный график в пространстве с помощью функции *comet3d(X,Y,Z)*.

Гистограмма, отражающая значения компонент вектора *y*, строится с помощью функции *bar(y)* (вертикальная) и *barh(y)* (горизонтальная). Если задать *y* с помощью матрицы, то можно представить несколько диаграмм одновременно. Также можно использовать спецификацию bar(x,y), которая поставит в соответствие столбцы в векторе *y* столбцам в *x*. Также можно определить стиль гистограммы:

—>*bar(x,y,'grouped')* – обычная гистограмма;

—>*bar(x,y,'stacked')* – гистограмма с накоплением/

Объемную гистограмму можно построить с помощью команды *hist3d(A)*, где *A* – матрица значений гистограммы.

Круговые диаграммы строятся с помощью функций *pie(q[,sp[,text]])*. Чтобы выдвинуть один или несколько секторов, надо указать параметр *sp* как вектор, где выдвигаемые сектора соответствуют единицам, а остальные нулям. Третий параметр *text* – это вектор из строк для подписи секторов.

## *Задание 3*

Вариант 1

1. Построить график функции: $y = -2x^2 + 4x - 1$.

2. Построить графики функций в одних осях, задав *y*, как матрицу с 2 столбцами из значений функций:

$$y_1 = \sin x,\ y_2 = \cos x + \sin x$$

3. Построить в одних осях графики функций, изменяя цвет и тип линий, а также тип маркера:

$$y_1 = |x|,\ y_2 = |4x - 2|,\ y_3 = |x| - 5,\ y_4 = \big||x| - 8\big|$$

4. На предыдущий график добавить заголовок, подписать оси и добавить легенду.

5. Построить графики функций, разделив области рисования (*subplot*):

$$y_1 = \tan x,\ y_2 = \ln x - 15x,\ y_3 = \cos|x|,\ y_4 = (1 + \tan x)^2$$

6. Построить график функции в полярных координатах: $\rho = \sqrt{3\cos 8\varphi}$.

7. Построить график функции с помощью функции *fplot*: $y = \frac{\sin x}{x}$.

8. Построить анимированный график функции: $y = \tan x$

9. Построить трехмерный график функции, используя функции *mesh* и *surf*: $Z = \sin(X + Y)$.

10. Построить анимированный трехмерный график для функции $Z$ из пункта 9.

11. Построить три вида гистограмм по следующим данным:

$$\begin{bmatrix} 3 & 5 & 9 \\ 7 & 8 & 2 \\ 1 & 4 & 10 \end{bmatrix}$$

12. Построить круговую диаграмму с выдвинутым сектором по данным: *[1 5 7]*.

Вариант 2

1. Построить график функции: $y = x^2 - 6x$.

2. Построить графики функций в одних осях, задав $y$, как матрицу с 2 столбцами из значений функций:

$$y_1 = \cos x,\ y_2 = \cos x + \sin x$$

3. Построить в одних осях графики функций, изменяя цвет и тип линий, а также тип маркера:

$$y_1 = \cos x,\ y_2 = |\cos 2x|,\ y_3 = \cos x + 2,\ y_4 = \cos|3x|$$

4. На предыдущий график добавить заголовок, подписать оси и добавить легенду.

5. Построить графики функций, разделив области рисования (*subplot*):

$$y_1 = \cos\left(\frac{\pi}{6} - x\right),\ y_2 = \tan x + \cot x,\ y_3 = \sqrt{\big||x| - 5\big|},\ y_4 = \sin^2 x - \cos x$$

6. Построить график функции в полярных координатах: $\rho = 2 + 5\cos\varphi$.

7. Построить график функции с помощью функции *fplot*: $y = \frac{\tan x}{x}$.

8. Построить анимированный график функции: $y = \cot x$

9. Построить трехмерный график функции, используя функции *mesh* и *surf*: $Z = \sin X + \tan Y$.

10. Построить анимированный трехмерный график для функции $Z$ из пункта 9.

11. Построить три вида гистограмм по следующим данным:

$$\begin{bmatrix} 1 & 2 & 7 \\ 4 & 8 & 12 \\ 5 & 9 & 4 \end{bmatrix}$$

12. Построить круговую диаграмму с выдвинутым сектором по данным: *[2 6 1]*.

Вариант 3

1. Построить график функции: $y = -4x - 1$.

2. Построить графики функций в одних осях, задав $y$, как матрицу с 2 столбцами из значений функций:

$$y_1 = \sin x,\ y_2 = \cos x - \sin x$$

3. Построить в одних осях графики функций, изменяя цвет и тип линий, а также тип маркера:

$$y_1 = \log x,\ y_2 = \log(-x),\ y_3 = |\log 2x|,\ y_4 = -2\log 3x$$

4. На предыдущий график добавить заголовок, подписать оси и добавить легенду.

5. Построить графики функций, разделив области рисования (*subplot*):

$$y_1 = \frac{1}{x} + \frac{2}{x^2},\ y_2 = \tan x + \cos x,\ y_3 = x^2 - 4,\ y_4 = \sqrt{\sin x}$$

6. Построить график функции в полярных координатах: $\rho = \sqrt{4\cos 2\varphi}$.

7. Построить график функции с помощью функции *fplot*: $y = \frac{\cot x}{x}$.

8. Построить анимированный график функции: $y = \log x$

9. Построить трехмерный график функции, используя функции *mesh* и *surf*: $Z = \sqrt{X^2 + Y^2}$.

10. Построить анимированный трехмерный график для функции $Z$ из пункта 9.

11. Построить три вида гистограмм по следующим данным:

$$\begin{bmatrix} 8 & 6 & 4 \\ 5 & 7 & 13 \\ 1 & 10 & 2 \end{bmatrix}$$

12. Построить круговую диаграмму с выдвинутым сектором по данным: *[15 7 8]*.

Вариант 4

1. Построить график функции: $y = -x^3$.

2. Построить графики функций в одних осях, задав $y$, как матрицу с 2 столбцами из значений функций:

$$y_1 = \cos x, y_2 = \cos x - \sin x$$

3. Построить в одних осях графики функций, изменяя цвет и тип линий, а также тип маркера:

$$y_1 = 2^x, y_2 = |10 - 2^x|, y_3 = 2^{1-2x}, y_4 = x^2$$

4. На предыдущий график добавить заголовок, подписать оси и добавить легенду.

5. Построить графики функций, разделив области рисования (*subplot*):

$$y_1 = e^{x-5}, y_2 = \frac{x}{2} + \frac{2}{x}, y_3 = |\cos x - \cos^2 x|, y_4 = \log(\tan x)$$

6. Построить график функции в полярных координатах: $\rho = -1 + 7\sin\varphi$.

7. Построить график функции с помощью функции *fplot*: $y = \frac{\cos x}{x}$.

8. Построить анимированный график функции: $y = e^x$.

9. Построить трехмерный график функции, используя функции *mesh* и *surf*: $Z = \log(X + Y)$.

10. Построить анимированный трехмерный график для функции $Z$ из пункта 9.

11. Построить три вида гистограмм по следующим данным:

$$\begin{bmatrix} 12 & 4 & 2 \\ 3 & 18 & 15 \\ 4 & 6 & 7 \end{bmatrix}$$

12. Построить круговую диаграмму с выдвинутым сектором по данным: *[2 6 9]*.

# Тема 4. Типы данных

Данные в SciLab могут быть символьными или числовыми (целыми или вещественными). Также в программе представлены такие типы данных, как массивы, структуры, списки, массивы ячеек и указатели.

SciLab допускает создание и хранение числовой информации в виде одно-, двух- и четырехбайтовых форматов со знаком или без знака. В пакете предусмотрены функции конвертирования вещественных и комплексных данных типа *double* в любой из допустимых форматов целых чисел, а также обратные преобразования.

Табл. 6. Функции преобразования

| Функция | Назначение | Диапазон допустимых значений |
|---|---|---|
| *y=int8(x)* | Преобразование в формат однобайтовых целых чисел со знаком | от -128 до 127 |
| *y=uint8(x)* | Преобразование в формат однобайтовых целых чисел без знака | от 0 до 255 |
| *y=int16(x)* | Преобразование в формат двухбайтовых целых чисел со знаком | от -32768 до 32767 |
| *y=uint16(x)* | Преобразование в формат двухбайтовых целых чисел без знака | от 0 до 65535 |
| *y=int32(x)* | Преобразование в формат четырехбайтовых целых чисел со знаком | от -2147483648 до 2147483647 |
| *y=uint32(x)* | Преобразование в формат четырехбайтовых целых чисел без знака | от 0 до 4294967295 |
| *y=double(x)* | Преобразование числового аргумента из целочисленного представления в формат вещественного числа с удвоенной точностью | |
| *y=iconvert(x,itype)* | Преобразование числового аргумента x в целочисленное представление согласно типу преобразования itype: 0 – число с плавающей запятой, 1 – int8, 11 – uint8, 2 – int16, 12 – uint16, 4 – int32, 14 – uint32. | |

При конвертировании вещественных данных в формат целых чисел аргументы, выходящие за пределы допустимого диапазона, заменяются соответственно самым большим и самым маленьким числом. В случае преобразования числа с дробной частью к целочисленному виду SciLab округляет их до ближайшего целого числа.

Для определения типа целых чисел используется функция:

*—>[i] = inttype(x)*

где x – матрица целых чисел, i – целое число, которое определяет тип элементов x:

- 1 – однобайтное представление целого числа;
- 2 – двухбайтное представление целого числа;
- 4 – четырёхбайтное представление целого числа;
- 11 – однобайтное представление беззнакового целого числа;
- 12 – двухбайтное представление беззнакового целого числа;
- 14 – четырёхбайтное представление беззнакового целого числа;
- 0 – число удвоенной точности (double).

Для хранения матриц, большая часть элементов которых равна нулю, используют разреженные матрицы (массивы), которые описываются командой *sparse(A)*.

Символьные векторы (массивы размерности $1 \times n$) представляют собой обычные строки. Для задания их значений в правой части оператора присваивания записывают цепочку символов, заключенную в одинарные или двойные кавычки.

*—>q='Symbol'*

Для определения текущих длин, размеров и количества измерений символьных массивов можно использовать функции *length(q), size(q)* и *ndims(q).*

Символьные матрицы можно складывать друг с другом. При этом происходит слияние соответствующих элементов.

Структуры представляют собой особый тип данных и состоят из полей. Структуры могут содержать разнородные данные, относящиеся к некоторому именованному объекту. Поле структуры может содержать другую вложенную структуру или массив структур. Это позволяет создавать вложенные структуры и даже многомерные массивы структур.

Одним из способов формирования структуры является использование функции *struct()*, которая заполняет указанный элемент массива структур. Аргументами этой функции являются пары, содержащие имя поля и соответствующее значение. Удобнее всего начинать заполнение с последнего элемента, чтобы задать массив нужного размера.

*—>str(n)=struct('Name1','Value1','Name2','Value2','Name3','Value3',...);*

где *n* – номер последнего элемента структуры; *str* – название структуры; *Name1, Name2, Name3* – имена полей структуры; *Value1, Value2, Value3* – значения полей структуры для *n*-го элемента.

Теперь можно заполнять пустые поля в двух предшествующих структурах, используя индексирование элементов массива и составные имена соответствующих полей:

*—>str(i).Name=' Value';*

*i* – номер заполняемого элемента структуры; *Name* – имя поля структуры; *Value* – значение поля.

С помощью функции *fieldnames()* можно получить информацию об именах полей указанной структуры:

*—>fieldnames(str)*

Однако для вывода содержимого полей приходится набирать имя каждой структуры – элемент соответствующего массива.

*—>str(i)*

Второй способ, который требует существенно меньших затрат по набору исходных данных, использует групповое задание значений каждого поля.

*—>str=struct('Name',{'Value1','Value2'},...*

В существующем массиве структур легко добавить новое поле путем присвоения значения в любом элементе массива:

*—>str(i).Newname= 'Newvalue';*

Для удаления существующего поля следует воспользоваться командой *null*():

*—>str.Name=null();*

Также в SciLab существуют массивы ячеек (Cell Array), где каждый элемент (ячейка) может быть представлена значением любого типа, независимо от типа соседних элементов. Элементы массива ячеек задаются с помощью команды:

*—>s=makecell(dims,a1,a2,...an)*

где *dims* – вектор положительных целых чисел, указывающих размерность массива ячеек; *a1,a2,…,an* – значения ячеек массива.

Для структурирования данных различного типа применяется объект список. Для создания списка существует команда:

*—>L=list(a1, ..., an)*

которая создает список *list* с элементами *ai*. Пустой список можно создать с помощью команды *list()*. Со списками можно выполнять следующие операции:

*L(i)=a* – вставка на место индекса *i*;

*L($+1)=e* – добавление элемента в хвост;

*L(0)=e* – добавление элемента в начало;

*L(i)=null()* – удаление i-ого элемента списка L;

*L3 = lstcat(L1,L2)* – объединение двух списков;

*size(L)* или *length(L)* – определение числа элементов списка.

Логические массивы появляются в результате различных проверок, выполняемых с помощью стандартных функций. Элементы логического массива занимают в памяти по одному байту, в которых может находиться либо 1 (логическая истина), либо 0 (логическая ложь).

Табл. 7. Функции проверки

| Функция | Действие |
|---|---|
| *isletter(q)* | Используется для выделения букв в символьной строке или массиве. |
| *isdigit(q)* | Проверяет являются ли символы в строке цифрами между 0 и 9. |
| *isreal(q)* | Проверяет, являются ли элементы массива *q* вещественными или комплексными числами. Если число задано так, что его мнимая часть равна 0, то оно все равно будет считаться комплексным. |

| Функция | Действие |
|---|---|
| *isnum(q)* | Проверяет, являются ли строка представлением числа |
| *iscell(q)* | Позволяет узнать, является ли аргумент массивом ячеек. |
| *iscellstr(q)* | Позволяет узнать является ли переменная cell-массивом строковых значений |
| *isstruct(q)* | Проверяет, является ли ее аргумент структурой. |
| *issparse(A)* | Позволяет проверить, является ли матрица *A* разреженной. |
| *isempty(A)* | Проверяет, является ли ее аргумент пустым массивом. |
| *isinf(A)* | Определяют, какие из элементов вещественного массива *A* принадлежат к бесконечным значениям. |
| *isnan(A)* | Определяют, какие из элементов вещественного массива *A* принадлежат к неопределенным значениям. |
| *isfield(s,fieldname)* | Проверяет существует ли поле с именем *fieldname* в структуре *s* |
| *isascii(q)* | Проверяет - является ли символ *q* 7-битным US-ASCII-символом |
| *isalphanum(q)* | Проверяет, что символы строки *q* являются буквенно-цифровыми. |

Проверка типа переменной объекта *x* осуществляется с помощью функции:

*—>[i]=type(x)*

Функция возвращает целое число, обозначающее тип объекта:

1 – матрица вещественных или комплексных значений удвоенной точности (double);

2 – матрица полиномов;

4 – матрица логических значений;

5 – разрежённая матрица;

6 – разрежённая матрица логических значений;

7 – разрежённая матрица SciLab;

8 – матрица целочисленных значений, хранимых в 1 (int8), 2 (int16) или 4 (int32) байтах.

9 – матрица графических дескрипторов;

10 – матрица символьных строк;

11 – некомпилированная функция (Scilab-код). Функция, созданная с помощью deff с аргументом 'n';

13 – компилированная функция (Scilab code);

14 – библиотека функций;

15 – список (list);

16 – типизированный список (tlist);

17 – матричноориентированный типизированный список (mlist);

0 –нуль-переменная.

Также можно воспользоваться функцией:

*—>[t]=typeof(x)*

которая выведет тип объекта в виде строки. Перечень выводимых строк можно найти в Справке.

### *Задание 4*

Вариант 1

1. Задать числа класса: *int8*, *int16* и *int32*. Можно ли выполнить над ними арифметические операции и функции (например, корень)?

2. Создать единичную матрицу размером *10x10*, преобразовать ее в разреженную и сравнить объем памяти, который они занимают.

3. Задать символьные массивы вида:

$$A = \begin{matrix} Robert \\ Bill \\ Poul \end{matrix}, B = \begin{matrix} 111 \\ 2 \\ 33 \end{matrix}$$

4. Определить длину, размер и количество измерений символьных массивов. Вычислить их сумму.

5. Создать структуру из 3 элементов со следующими полями:

| *Surname* | *Name* | *Age* |
|---|---|---|

6. Вывести имена полей указанной структуры; вывести второй элемент структуры.

7. Добавить к структуре новое поле *Patronymic* (отчество).

8. Удалить из структуры поле: *Age*.

9. Показать значение поля *Name* для 2-го элемента структуры.

10. Проверить есть ли в структуре поле с именем *Age*.

11. Создать массив ячеек вида:

$$3 \quad Hello \quad \begin{matrix} 1 & 3 \\ 5 & 0 \end{matrix} \quad 2 + i$$

12. Создать аналогичный список.

13. Добавьте в начало списка элемент 145.

14. Добавьте в конец списка элемент *f*.

15. Удалите из списка 4-ый элемент.

16. Создайте список вида: $128 \quad true$. Добавьте его к ранее созданному. Определите размер получившегося списка.

17. Какие элементы этого списка являются буквами, цифрами, комплексными числами, неопределенными или пустыми значениями.

18. Задайте матрицу A и проверьте является ли она пустой или разреженной. Определите ее тип с помощью функций *type*() и *typeof*()

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 5 & 10 & 2 \\ 1 & 4 & 3 \end{bmatrix}$$

Вариант 2

1. Задать числа класса: *int8*, *int16* и *int32*. Можно ли выполнить над ними арифметические операции и функции (например, корень)?

2. Создать единичную матрицу размером *15x15*, преобразовать ее в разреженную и сравнить объем памяти, который они занимают.

3. Задать символьные массивы вида:

$$A = \begin{matrix} Elena \\ Maria \\ Gloria \end{matrix}, B = \begin{matrix} 4 \\ 55 \\ 666 \end{matrix}$$

4. Определить длину, размер и количество измерений символьных массивов. Вычислить их сумму.

5. Создать структуру из 3 элементов со следующими полями:

| *Firm* | *Activity_category* | *Interest_rate* |
|---|---|---|

6. Вывести имена полей указанной структуры; вывести второй элемент структуры.

7. Добавить к структуре новое поле *Adress*.

8. Удалить из структуры поле *Interest rate*.

9. Показать значение поля *Firm* для 3-го элемента структуры.

10. Проверить есть ли в структуре поле с именем *Interest rate*.

11. Создать массив ячеек вида:

$$Street \quad 17 \quad 17 + 6i \quad \begin{matrix} 1 & 3 \\ 2 & 5 \end{matrix}$$

12. Создать аналогичный список.

13. Добавьте в начало списка элемент 7.

14. Добавьте в конец списка элемент *t*.

15. Удалите из списка 5-ый элемент.

16. Создайте список вида: $45 \quad false$. Добавьте его к ранее созданному. Определите размер получившегося списка.

17. Какие элементы этого списка являются буквами, цифрами, комплексными числами, неопределенными или пустыми значениями.

18. Задайте матрицу A и проверьте является ли она пустой или разреженной. Определите ее тип с помощью функций *type*() и *typeof*()

$$A = \begin{bmatrix} 1 & 10 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix}$$

Вариант 3

1. Задать числа класса: *int8*, *int16* и *int32*. Можно ли выполнить над ними арифметические операции и функции (например, корень)?

2. Создать единичную матрицу размером *12x12,* преобразовать ее в разреженную и сравнить объем памяти, который они занимают.

3. Задать символьные массивы вида:

$$A = \begin{matrix} Europe \\ Asia \\ Africa \end{matrix}, B = \begin{matrix} 77 \\ 8888 \\ 9 \end{matrix}$$

4. Определить длину, размер и количество измерений символьных массивов. Вычислить их сумму.

5. Создать структуру из 3 элементов со следующими полями:

| *Surname* | *Brand* | *Car_number* |
|---|---|---|

6. Вывести имена полей указанной структуры; вывести второй элемент структуры.

7. Добавить к структуре новое поле *Color*.

8. Удалить из структуры поле *Car number*.

9. Показать значение поля *Color* для 1-го элемента структуры.

10. Проверить есть ли в структуре поле с именем *Car number*.

11. Создать массив ячеек вида:

$$10i \quad \begin{matrix} 6 & 5 \\ 3 & 41 \end{matrix} \quad 100 \quad Apple$$

12. Создать аналогичный список.

13. Добавьте в начало списка элемент 17.

14. Добавьте в конец списка элемент *m*.

15. Удалите из списка 3-ий элемент.

16. Создайте список вида: $true \quad 24$. Добавьте его к ранее созданному. Определите размер получившегося списка.

17. Какие элементы этого списка являются буквами, цифрами, комплексными числами, неопределенными или пустыми значениями.

18. Задайте матрицу A и проверьте является ли она пустой или разреженной. Определите ее тип с помощью функций *type*() и *typeof*()

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 5 & 10 \\ 16 & 25 & 81 \end{bmatrix}$$

Вариант 4

1. Задать числа класса: *int8*, *int16* и *int32*. Можно ли выполнить над ними арифметические операции и функции (например, корень)?

2. Создать единичную матрицу размером *17x17*, преобразовать ее в разреженную и сравнить объем памяти, который они занимают.

3. Задать символьные массивы вида:

$$A = \begin{matrix} Germany \\ England \\ France \end{matrix}, B = \begin{matrix} 11 \\ 444 \\ 7 \end{matrix}$$

4. Определить длину, размер и количество измерений символьных массивов. Вычислить их сумму.

5. Создать структуру из 3 элементов со следующими полями:

| *Surname* | *Course* | *Mark* |
|---|---|---|

6. Вывести имена полей указанной структуры; вывести второй элемент структуры.

7. Добавить к структуре новое поле *Exam*.

8. Удалить из структуры поле *Mark*.

9. Показать значение поля *Course* для 2-го элемента структуры.

10. Проверить есть ли в структуре поле с именем *Mark*.

11. Создать массив ячеек вида:

$$\begin{matrix} 3 & 15 \\ 2 & 7 \end{matrix} \quad 1-2i \quad University \quad 16.7$$

12. Создать аналогичный список.

13. Добавьте в начало списка элемент 5.

14. Добавьте в конец списка элемент $k$

15. Удалите из списка 2-ой элемент.

16. Создайте список вида: $245 \quad false$. Добавьте его к ранее созданному. Определите размер получившегося списка.

17. Какие элементы этого списка являются буквами, цифрами, комплексными числами, неопределенными или пустыми значениями.

18. Задайте матрицу A и проверьте является ли она пустой или разреженной. Определите ее тип с помощью функций *type*() и *typeof*()

$$A = \begin{bmatrix} 1 & 5 & 25 \\ 1 & 7 & 49 \\ 1 & 8 & 10 \end{bmatrix}$$

# Тема 5. Программирование на языке SciLab

Файл-сценарий – это запись серии команд без входных и выходных параметров. Главной особенностью такого файла является то, что он работает с переменными Командного окна. Файл создается с помощью редактора SciNotes (крайняя левая кнопка на панели инструментов) и сохраняется с расширением .sce.

Для выполнения файла-сценария применяется команда

—>*exec(path [, mode])*

где *path* – строка, путь к файлу-сценарию; *mode* – необязательный параметр, режим исполнения: 0 – значение по умолчанию; -1 – вывод на экран не производится; 1 – отображает каждую командную строку; 2 – на экран выводится строка приглашения -->; 3 – отображения командных строк + строки приглашения; 4 – останавливается перед каждой строкой приглашения. Исполнение возобновляется после возврата каретки; 7 – остановки + строки приглашения + вывод на экран командных строк: режим, полезный для демонстраций.

Длина каждой строки в файле не должна превышать 4096 символов. Функция *exec*() также может использоваться для определения функций, используя синтаксис определения функции в строке.

Функции SciLab (макросы или процедуры) – это определенный класс объектов. Они могут определяться в отдельных файлах с помощью редактора и загружаться в SciLab через команду *exec*() или через библиотеки (*lib*() или *genlib*()). Также функции могут определяться внутри программы с помощью команд *deff*() или *function*.

Имена функций должны удовлетворять тем же правилам, что и имена переменных (см. Тему 1).

Определение функции состоит из двух частей: синтаксис определения и ряд инструкций.

Рассмотрим команду *function*, которая открывает определение функции, и команду *endfunction*, которая ее закрывает. Они имеют следующий синтаксис:

*function <выходные_аргументы>=<имя_функции><входные_аргументы>*

*<операторы>*

*endfunction*

Здесь *<имя_функции>* указывает имя функции, *<входные_аргументы>* указывают список входных аргументов. Он может быть рядом имён переменных, разделённых запятой и заключённым в круглые скобки, например *(x1,...,xm)*: последовательностью () или ничего, если у функции нет входных аргументов. Последнее имя переменной может быть ключевым словом *varargin*. *<выходные_аргументы>* указывают список выходных аргументов. Он может быть рядом имён переменных, заключённым в квадратные скобки, например *[y1,...,yn]*, или последовательностью [], если у функции нет выходных аргументов. Последнее имя переменной может быть ключевым словом *varargout*. *<операторы>* указывают набор Scilab-инструкций (операторов).

Этот синтаксис можно использовать для определения функции как встраиваемой, так и в файле-сценарии.

*—>function [x, y]=mynewfunc(a, b)*

*—>x=a*b*

*—>y=a/b*

*—>endfunction*

*—>[x,y]=mynewfunc(10,2)*

Ключевое слово *varargin* указывает на переменное число аргументов в списке входных параметров. То есть функцию, в синтаксисе определения которой последним (!!!) указан такой входной аргумент, можно вызвать с большим числом входных аргументов, чем перечислено. *function y=ex(varargin)* может быть вызвана с любым числом аргументов. Внутри функции *ex* входные аргументы могут быть получены в *varargin(i), i=1:length(varargin).*

Аналогичный смысл имеет ключевое слово *varargout* – переменное число аргументов в списке выходных параметров. *varargout = function ex()* может вызываться с любым количеством выходных аргументов. Внутри функции *ex* выходные аргументы могут храниться в *varargout(i).*

При вызове функции можно использовать более короткий список входных и выходных переменных, чем определенный. В так случае переменные используются или устанавливаются слева направо.

Функция *[lhs [,rhs] ]=argn()* возвращает фактическое количество входных (*rhs*) и выходных (*lhs*) аргументов, переданных в функцию при ее вызове.

Также для определения функции во время выполнения программы можно использовать оператор *deff*():

*—>deff('[s1, s2, ...] = newfunction(e1, e2, ...)',text)*

где *e1, e2, ...* – входные переменные, *s1, s2, ...* – выходные переменные, *text* – матрица символьных строк, описывающих функцию. Например:

*—>deff('[x,y]=mynewfunc(a,b)',['x=a*b'; 'y=a/b'])*

Функции в SciLab имеют свои собственные локальные переменные. Ключевое слово *global* позволяет использовать значения указанных переменных во всех функциях. Любое присвоение значения этой переменной в любой функции возможно во всех остальных функциях при объявлении её глобальной:

*—>global('nam1',...,'namn')*

или

*—>global nam1 ... namn*

Если глобальная переменная не существует при первом вызове инструкции *global*, то она будет проинициализирована пустой матрицей.

Для проверки является ли переменная *x* глобальной может быть использована функция:

*—>t=isglobal(x)*

Для удаления глобальных переменных используется команда:

*—>clearglobal()*

которая удаляет все глобальные переменные, или

*—>clearglobal nam1 ... namn*

которая удаляет перечисленные переменные.

Комментарии в SciLab начинаются с символа //. Они могут располагаться с начала строки или находиться правее любого оператора. Внутри функции первые строки, вплоть до первой инструкции или пустой строки могут использоваться для указания содержимого для функции *help*.

Рассмотрим управляющую логику SciLab.

Табл. 8. Управляющая логика

| № | Формат оператора | Пояснение |
|---|---|---|
| 1 | *var=expr* | Оператор присваивания, вычисляет значения выражения *expr* и заносит результаты вычисления в переменную *var*. |
| 2 | *if* условие_1 *then*<br>операторы_1<br>[*elseif* условие_2 *then*<br>операторы_2<br>*elseif* условие_3 then<br>операторы_3<br>……<br>*else* операторы]<br>*end* | Условный оператор. Если справедливо условие_1, то выполняется группа операторы_1; если справедливо условие_2, то выполняется группа операторы_2;…<br>Если все указанные условия оказываются ложными, то выполняются операторы, расположенные между *else* и *end*.<br>Ключевое слово *then* должно быть на той же самой строке, что и соответствующее ключевое слово *if* или *elseif*.<br>Ключевое слово *then* можно заменить на возврат каретки или запятую. |
| 3 | *select expr*<br>*case val1 then*<br>операторы_1<br>*case val2 then*<br>операторы_2<br>……<br>[*else* операторы]<br>*end* | Переключатель по значению выражения *expr*. Если оно совпадает с величиной *val1*, то выполняется группа операторы_1, если оно совпадает с величиной *val2*, то выполняется группа операторы_2.<br>Если значение *expr* не совпадает ни с одной из перечисленных величин, то выполняются операторы, расположенные между *else* и *end*.<br>Ключевое слово *then* должно быть на той же самой строке, что и соответствующее ключевое слово *case*. |
| 4 | *for var=e1:[e2:]e3*<br>операторы<br>*end* | Цикл типа арифметической прогрессии, в котором переменная *var* при каждом повторении тела цикла изменяется от начального значения *e1* с шагом *e2* до конечного значения *e3*. |
| 5 | *while* условие<br>операторы<br>[*else* операторы_1]<br>*end* | Цикл с предусловием, повторяющийся до тех пор, пока истинно указанное условие.<br>Необязательная конструкция [else операторы_1] позволяет указать операторы, которые будут выполняться, когда указанное условие не выполняется. |

| № | Формат оператора | Пояснение |
|---|---|---|
| 6 | *break* | Досрочный выход из управляющих конструкций типа *for* или *while*. |
| 7 | *function f1*<br>*function f2(x1,x2,…)*<br>*function y=f3(x1,x2,…)*<br>*function [y1,y2,…]=f4(x1,x2,…)* | Заголовок функции.<br>(x1,x2,…) – входные параметры; (y1,y2,…) – выходные параметры. |
| 8 | *return* | Досрочный выход из тела функции. |
| 9 | *abort* | Инструкция *abort* прерывает текущее исполнение и даёт приглашение к вводу. С того уровня, на котором вызвана *pause*, инструкция abort возвращает на уровень 0. |
| 10 | *continue* | Внутри цикла *for* или *while* команда *continue* передаёт управление следующей итерации цикла, в котором она стоит, пропуская все оставшиеся инструкции между этой командой и инструкцией *end* конца цикла. |

Для ввода числовой и символьной информации используется функция *input*():

*—>x=input(“приглашение”)*

В ответ на приглашение, выданное программой, пользователь может набрать требуемое значение или выражение, величина которого будет подсчитана с учетом текущего состояния переменных рабочего пространства и возвращена в качестве значения функции.

Второй формат обращения к функции *input*() имеет вид:

*—>x=input(“приглашение”, “s”)*

в этом случае текст, набираемый пользователем, рассматривается как строка символов, которая и возвращается в качестве значения функции. Если в качестве строки *x* было введено выражение, то вычислить его значение можно с помощью функции:

*—>y=evstr(x)*

## *Задание 5*

Вариант 1

1. С помощью функции *deff*() задайте функцию, которая переводит градусы Цельсия в градусы Фаренгейта.

2. Создайте файл-функцию, которая определяет знак многочлена $ax^3 - bx^2 - cx + d$ в точке, значение которой задается с клавиатуры. Параметры многочлена передаются в функцию в качестве входных параметров.

3. Напишите функцию, которая вычисляет для данного *n*:

$$\sum_{k=0}^{n} \frac{(-1)^k (k+1)}{k!}$$

Вариант 2.

1. С помощью функции *deff*() задайте функцию, которая переводит градусы Фаренгейта в градусы Цельсия.

2. Создайте файл-функцию, которая определяет, принадлежит ли число, заданное с клавиатуры, массиву чисел, который передается в функцию как параметр.

3. Пусть $y_0=0$; $y_k = \frac{y_{k-1}+1}{y_{k-1}+2}$, $k=1,2,\ldots$ Дано действительное $\varepsilon>0$, найти первый член $y_n$ , для которого выполняется $y_n - y_{n-1} < \varepsilon$.

Вариант 3.

1. С помощью функции *deff*() задайте функцию, которая переводит мили в километры.

2. Создайте файл-функцию, которая определяет, содержит ли строка, введенная с клавиатуры, пробелы. Вывести результат в виде строки.

3. Пусть $y_0=0$; $y_k = \frac{y_{k-1}+3}{y_{k-1}+5}$, $k=1,2,\ldots$ Дано действительное $\varepsilon>0$, найти первый член $y_n$ , для которого выполняется $y_n - y_{n-1} < \varepsilon$.

Вариант 4

1. С помощью функции *deff*() задайте функцию, которая переводит километры в мили.

2. Создайте файл-функцию, которая определяет, какими числами являются корни квадратного уравнения $ax^2 + bx + c$: действительными или комплексными. Параметры многочлена передаются в функцию в качестве входных параметров.

3. Напишите функцию, которая вычисляет для данного *n*:

$$\sum_{k=0}^{n} \frac{(-1)^{k+1} k^2}{(k+1)}$$

# Тема 6. Обработка символьных данных

Символьные данные могут быть представлены как в виде отдельных строк (вектор-строка), так и виде массивов строк (матрица) и массивов ячеек. Создание переменных происходит при формировании символьных значений с помощью оператора присваивания.

На каждый символ во внутреннем представлении отводится по 2 байта в кодировке ASCII Unicode. Функция *char(X)* — преобразует массив X положительных целых чисел (числовых кодов от 0 до 65 535) в массив символов системы SciLab.

Если в качестве аргумента функции *char*() выступает cell-массив строк разной длины, то функция возвращает символьный массив, котором строки являются конкатенированными элементами соответствующих строк cell-массива строковых значений, дополняя более короткие аргументы пробелами.

Пробелы в символьных данных играют роль разделителя слов. Иногда их приписывают в начале или в конце строки для того, чтобы произвести соответствующее выравнивание по левой границе или по длине. Для формирования строки, содержащей заданное количество пробелов (n), можно использовать функцию:

—>*blanks(n)*

Начальные и концевые пробелы можно удалить с помощью функции:

—>*txt=stripblanks(txt[,tabs])*

Если на месте необязательного логического параметра *tabs* указано значение *%t*, то также будут удаляться символы табуляции (значение по умолчанию – *%f*).

Значение длины строки определяется с помощью функции *length*():

—>*length(s)*

Для выравнивания массива символов применяется функция *justify()*, которая выравнивает столбцы матрицы строк по центру (*opt=‘center’*), по левому (*opt=‘left’*) или правому краю (*opt=‘right’*):

—>*justify(s,opt)*

Для выделения *k*-го символа из строки *str* применяется функция:

—>*str_out=part(str,k)*

*k* может быть вектором-строкой целочисленных значений.

Массивы строк размером $m \times n$, заполненные одним и тем же символом (например, $), создаются с помощью функции *repmat*():

—>*repmat('$',m,n)*

Для связывания символьных строк используется функция:

—>*txt=strcat(strings[,string_added, [“flag”]])*

Здесь *strings* – это вектор или матрица строк, которые связываются; строка *string_added* добавляется к каждому элементу матрицы *strings*, кроме последнего; параметр “flag” указывает на объединение в строку или в столбец. По умолчанию, строки объединяются в строку, что эквивалентно записи:

—>*txt=strcat(strings,“”,“r”)*

Для объединения по вертикали, нужно записать:

*—>txt=strcat(strings,“”,“c”)*

При горизонтальном объединении результатом будет одна строка, а при вертикальном объединении – массив строк.

Так как символьные данные представлены в памяти компьютера числовыми кодами своих символов, их можно сравнивать. Для сравнения символьных данных используются следующие функции:

*—>strcmp(string1,string2)* – возвращает ноль, если две сравниваемые строки *string1* и *string2* идентичны. Значение больше нуля указывает, что первый символ, который не соответствует, имеет большее значение в *string1*, чем в *string2*, а значение меньше нуля указывает обратное.

*—>strcmpi(string1,string2)* – выполняет аналогичное сравнение, игнорируя разницу между кодами больших и малых букв (без учета регистра).

Для поиска первого появления символа в строке существует функция:

*—>res=strchr(strings,char)*

Функция возвращает остаток строки после первого появления символа *char* в строке *strings*. Если *strings* – массив строк, то символ *char* ищется в каждой строке. *char* должна содержать либо одиночный символ, либо массив одиночных символов, количество строк которого совпадает с количеством строк в массиве *strings*.

Для поиска последнего символа в строке используется функция *strrchr(),* ее синтаксис и правила использования аналогичны *strchr().*

*—>res=strrchr(strings,char)*

Для обнаружения подстроки в строке применяется функция:

*—>res=strstr(strings,str)*

Она возвращает матрицу строк начиная от того места, где в *strings* впервые встретилась *str* и до конца *strings*, либо ‘’, если *str* не является частью *strings*.

Функция *strindex()* ищет положение (индексы) символьной строки *str* в другой строке *strings*:

*—>ind=strindex(strings,str)*

Для поиска в строке *str1* всех вхождений заданной цепочки символов *str2* и замены каждой из них новым значением *str3* предназначена функция:

*—>strsubst(str1, str2, str3)*

Если *str2* не найдена, то возвращается строка *str1*.

Лексема – это цепочка символов, завершающаяся тем или иным разделителем. Функция *strtok*() выделяет лексему:

*—>res=strtok(str,delimiters)*

В строку *res* заносится первая найденная лексема. Второй аргумент *delimeters* позволяет задать нестандартный набор символов разделителей.

Также лексемы можно выделить с помощью функции:

*—>res=tokens(str[,delimiter])*

Эта функция выводит массив строк – лексем. Здесь *delimiter* является необязательным параметром (по умолчанию, *[" ",<Tab>]*), это или строка с символом-разделителем или вектор таких строк.

Функция *tokenpos*() имеет синтаксис аналогичный *tokens*() и возвращает индексы первого и последнего символов каждой найденной лексемы.

*—>res=tokenpos(str[,delimiter])*

Функция *convstr(str,[flag])* преобразует матрицу строк *str* в нижний регистр (для *flag="l"*; значение по умолчанию) или в верхний регистр (для *flag="u"*).

Функция *res=strrev(str)* возвращает перевернутую строку *str*.

Существует ряд функций для преобразования символов и чисел:

Табл. 9. Функции для преобразования

| Функция | Действие |
|---|---|
| *a=ascii(txt)*<br>*txt=ascii(a)* | Эта функция преобразует Scilab-строку *txt* в вектор ASCII-кодов *a* (первые 127 кодов являются ASCII) или вектор ASCII-кодов *a* в Scilab-строки *txt*. |
| *y=char(x)*<br>*y=char(st1,st2,st3,...)* | Преобразует cell-массив строковых значений или массив ASCII-кодов *x* или массивы строковых значений *st1,st2,st3,…* в массив строковых значений *y* |
| *t=sci2exp(a[,nam][,lmax])* | Преобразует выражение *a* (константа, полином, матрица строк, список или матрица логических значений) в строку инструкции, если указано *nam* (символьная строка) или в строку выражения. *lmax* – целое число, содержащее длину строки. |
| *string(x)* | Преобразует булеву, комплексную, вещественную, целочисленную, полиномиальную матрицу, библиотеку или функцию в матрицу строк. |
| *d=strtod(str)*<br>*[d,endstr]=strtod(str)* | Преобразует строку *str* в число удвоенной точности *d*. *endstr* – символьная строка или матрица символьных строк (следующий символ в *str* после числового значения). |

Существует ряд строковых функций для преобразования систем счисления:

Табл. 10. Представления целых чисел

| Функция | Преобразует | |
|---|---|---|
| | из | в |
| *base2dec(s,b)* | *s* – строка с числом по основанию *b* | десятичное число |
| *bin2dec(str)* | *str* – строка с двоичным числом | целое десятичное число |
| *dec2base(d,base,n)* | *d* – десятичное положительное целое число | строка с числом, соответствующим представлению в системе счисления по основанию *base* с *n* знаками |
| *dec2bin(x,n)* | *x* – десятичное положительное целое число | строка с двоичным представлением числа с *n* знаками (при необходимости слева дополняется 0) |

| Функция | Преобразует | |
|---|---|---|
| | из | в |
| *dec2hex(d)* | *d* – десятичное положительное целое число | строка с шестнадцатеричным представлением числа |
| *dec2oct(d)* | *d* – десятичное положительное целое число | строка с восьмеричным представлением числа |
| *hex2dec(h)* | *h* – строка с шестнадцатеричным представлением числа | десятичное положительное целое число |
| *oct2dec(o)* | *o* – строка с восьмеричным представлением числа | десятичное положительное целое число |

Строковые переменные в SciLab могут также использоваться для символьных вычислений. В этом случае необходимые переменные и выражения задаются как строки.

Символьное суммирование осуществляется с помощью функции:

*—>c=addf(a,b)*

Здесь *a, b, c* – это строки. Функция выполняет простейшие упрощения.

Для символьного вычитания применяется функция:

*—>c=subf(a,b)*

Для символьного умножения предназначена функция:

*—>c=mulf(a,b)*

Символьное деление бывает левым и правым:

*—>c=ldivf(a,b)*

*—>c=rdivf(a,b)*

*ldivf('x', 'y')* выполняет левое символьное деление (деление справа налево) и возвращает строку *'x\y'*. *rdivf('x','y')* выполняет правое символьное деление (деление слева направо) и возвращает строку *'x/y'*. Обе функции, также как все перечисленные выше, выполняют простейшие упрощения.

Функция *cmb_lin(alfa,x,beta,y)*, где все аргументы *alfa, beta, x, y* – символьные строки, выполняет символьное линейное сочетание *alfa*x-beta*y*. (низкоуровневая программа).

Для решения системы линейных уравнений в символьном виде применяется функция:

*—>[x]=solve(A,b)*

Она решает уравнение *A*x=b*, где *A* является верхней треугольной матрицей, составленной из символьных строк.

Для вычисления выражений, заданных матрицей символьных строк Z можно воспользоваться одной из двух функций: *evstr(Z)* или *eval(Z)*.

## *Задание 6*

Вариант 1

1. Создать массив ячеек *str_cell*:

$$'123456789' \quad 'ab'$$
$$'0' \quad 'cdefgh'$$

2. Содержимое массива *str_cell* выровнять по центру, по левой и правой стороне.

3. Преобразовать массив кодов в массив символов: *A=32:52*.

4. Создать строку *str1=‘ East ‘*, вычислить ее длину.

5. Удалить из строки *str1* концевые пробелы. Вычислить длину строки.

6. Показать пятый символ из строки *str= ‘It is life’*.

7. Создать массив 3×4 заполненный символами +.

8. Объединить строки вертикально и горизонтально:

*s1=‘Happy’ s2=‘New’ s3=‘Year’*

9. Сравнить строки двумя способами:

*st1=‘example’ u st2=‘EXAMple’*

10. Преобразовать строку *st2* к верхнему и нижнему регистрам.

11. Входит ли строка *s= ‘oo’* в строку *str=‘boom’*. Найти первое и последнее вхождение символа *‘o’* в строку *str*, а также определить индексы ее вхождения.

12. Заменить в строке *str=‘London is the capital of Great Britain’* символ *‘o’* на *‘*’*.

13. Выделить в строке *str* лексемы разными способами.

14. Преобразовать матрицу случайных чисел с нормальным распределением размера 3x3 в матрицу строк.

15. Преобразовать число 123 в строки с двоичным, восьмеричным и шестнадцатеричным представлениями числа.

16. Определите перевернутую строку для строки: *“Madam I'm Adam”*

17. Задайте символьные переменные *a=‘x’, b=‘y’, c=‘2’, d=‘1’, e=‘0’*. Найдите их суммы, разности, произведения и частное в различных комбинациях.

18. Определите символьное линейное сочетание для *alfa=‘2’, beta=‘3’, x=‘x’, y=‘y’*.

19. Задайте символьные матрицы *A* и *b*, решите уравнение *A*x=b*, вычислите корни с помощью функции *eval*() и выполните проверку.

$$A = \begin{bmatrix} 1 & 2 & -5 \\ 0 & 4 & -3 \\ 0 & 0 & 2 \end{bmatrix}, b = \begin{bmatrix} 10 \\ 16 \\ 4 \end{bmatrix}$$

Вариант 2

1. Создать массив ячеек *str_cell*:

$$'123456' \quad 'a'$$
$$'78' \quad 'bcdef'$$

2. Содержимое массива *str_cell* выровнять по центру, по левой и правой стороне.

3. Преобразовать массив кодов в массив символов: *A=53:64.*

4. Создать строку *str1=‘ West ‘*, вычислить ее длину.

5. Удалить из строки *str1* концевые пробелы. Вычислить длину строки.

6. Показать пятый символ из строки *str=‘be happy’.*

7. Создать массив 3×4 заполненный символами *.

8. Объединить строки вертикально и горизонтально:

*s1=‘Merry’ s2=‘Christ’ s3=‘mas’*

9. Сравнить строки двумя способами:

*st1=‘examination’* и *st2=‘exaMINation’*

10. Преобразовать строку *st2* к верхнему и нижнему регистрам.

11. Входит ли строка *s= ‘oo’* в строку *str=‘cool’*. Найти первое и последнее вхождение символа *‘o’* в строку *str*, а также определить индексы ее вхождения.

12. Заменить в строке *str=‘London is the capital of Great Britain’* символ *‘a’* на *‘@’*.

13. Выделить в строке *str* лексемы разными способами.

14. Преобразовать матрицу случайных чисел с нормальным распределением размера 2x2 в матрицу строк.

15. Преобразовать число 78 в строки с двоичным, восьмеричным и шестнадцатеричным представлениями числа.

16. Определите перевернутую строку для строки: “*Was it a car or a cat I saw?*”

17. Задайте символьные переменные *a=‘k’, b=‘n’, c=‘5’, d=‘0’, e=‘2’*. Найдите их суммы, разности, произведения и частное в различных комбинациях.

18. Определите символьное линейное сочетание для *alfa=‘4’, beta=‘1’, x=‘x’, y=‘y’.*

19. Задайте символьные матрицы *A* и *b*, решите уравнение *A*x=b*, вычислите корни с помощью функции *eval*() и выполните проверку.

$$A = \begin{bmatrix} 8 & 4 & -6 \\ 0 & 10 & -2 \\ 0 & 0 & -3 \end{bmatrix}, b = \begin{bmatrix} 22 \\ -3 \\ 6 \end{bmatrix}$$

Вариант 3

1. Создать массив ячеек *str_cell*:

$$'zyxwv' \quad '12345'$$
$$'klm' \quad '76'$$

2. Содержимое массива *str_cell* выровнять по центру, по левой и правой стороне.

3. Преобразовать массив кодов в массив символов: *A=65:86.*

4. Создать строку *str1=‘ North ‘*, вычислить ее длину.

5. Удалить из строки *str1* концевые пробелы. Вычислить длину строки.

6. Показать пятый символ из строки *str=‘Are you ready’.*

7. Создать массив 3×4 заполненный символами #.

8. Объединить строки вертикально и горизонтально:

*s1='Happy' s2='Birth' s3='day'*

9. Сравнить строки двумя способами:

*st1='corporation'* и *st2='CORPOratioN'*

10. Преобразовать строку *st2* к верхнему и нижнему регистрам.

11. Входит ли строка *s= 'oo'* в строку *str='loop'*. Найти первое и последнее вхождение символа *'o'* в строку *str*, а также определить индексы ее вхождения.

12. Заменить в строке *str='London is the capital of Great Britain'* символ *'n'* на *'$'*.

13. Выделить в строке *str* лексемы разными способами.

14. Преобразовать матрицу случайных чисел с равномерным распределением размера 2x2 в матрицу строк.

15. Преобразовать число 112 в строки с двоичным, восьмеричным и шестнадцатеричным представлениями числа.

16. Определите перевернутую строку для строки: "*Some men interpret nine memos*"

17. Задайте символьные переменные *a='z', b='t', c='0', d='3', e='6'*. Найдите их суммы, разности, произведения и частное в различных комбинациях.

18. Определите символьное линейное сочетание для *alfa='5', beta='7', x='x', y='y'*.

19. Задайте символьные матрицы *A* и *b*, решите уравнение *A*x=b*, вычислите корни с помощью функции *eval*() и выполните проверку.

$$A = \begin{bmatrix} 5 & -3 & 8 \\ 0 & 2 & 4 \\ 0 & 0 & 6 \end{bmatrix}, b = \begin{bmatrix} 10 \\ 16 \\ 12 \end{bmatrix}$$

Вариант 4

1. Создать массив ячеек *str_cell*:

$$\begin{matrix} 'kkllmmnn' & 'a' \\ '678' & '543210' \end{matrix}$$

2. Содержимое массива *str_cell* выровнять по центру, по левой и правой стороне.

3. Преобразовать массив кодов в массив символов: *A=87:108*.

4. Создать строку *str1= '  South  '*, вычислить ее длину.

5. Удалить из строки *str1* концевые пробелы. Вычислить длину строки.

6. Показать пятый символ из строки *str='Do not worry'*.

7. Создать массив 3×4 заполненный символами |.

8. Объединить строки вертикально и горизонтально:

*s1='Good' s2='after' s3='noon'*

9. Сравнить строки двумя способами:

*st1='MARKET'* и *st2= 'MARket'*

10. Преобразовать строку *st2* к верхнему и нижнему регистрам.

11. Входит ли строка *s= 'oo'* в строку *str='ooooops'*. Найти первое и последнее вхождение символа *'o'* в строку *str*, а также определить индексы ее вхождения.

12. Заменить в строке *str='London is the capital of Great Britain'* символ *'i'* на *'&'*.

13. Выделить в строке *str* лексемы разными способами.

14. Преобразовать матрицу случайных чисел с равномерным распределением размера 3x3 в матрицу строк.

15. Преобразовать число 92 в строки с двоичным, восьмеричным и шестнадцатеричным представлениями числа.

16. Определите перевернутую строку для строки: "*Live not on evil*"

17. Задайте символьные переменные *a='m', b='l', c='3', d='7', e='0'*. Найдите их суммы, разности, произведения и частное в различных комбинациях.

18. Определите символьное линейное сочетание для *alfa='6', beta='2', x='x', y='y'*.

19. Задайте символьные матрицы *A* и *b*, решите уравнение *A*x=b*, вычислите корни с помощью функции *eval*() и выполните проверку.

$$A = \begin{bmatrix} 7 & 3 & 1 \\ 0 & 8 & 2 \\ 0 & 0 & 11 \end{bmatrix}, b = \begin{bmatrix} 28 \\ 30 \\ 121 \end{bmatrix}$$

# Тема 7. Работа с файлами

Файл обычно является некоторой совокупностью данных, объединенных одним именем.

Основными операциями с файлами в SciLab являются: открытие файла, чтение из файла или запись в файл, закрытие файла. В программе можно работать с текстовыми или двоичными файлами.

Файла открываются с помощью функции *mopen*(), синтаксис которой имеет вид:

*—>[fd,err]=mopen(file[,mode])*

Первый аргумент *file* – это символьная строка с именем файла, который требуется открыть. Второй необязательный аргумент функции *mode* определяет режим доступа к открываемому файлу. В качестве этого параметра могут выступать один, два или три символа, заключенные в одинарные кавычки.

Табл. 11. Режимы доступа к файлу

| Параметр *mode* | Режим доступа |
|---|---|
| r | Открывает существующий файл для чтения (по умолчанию). |
| w | Открывает файл на запись. Если этот файл существует, то его содержимое будет уничтожено. |
| a | Открывает файл для добавления записи. Создаёт файл если он не существует |
| r+ | Открывает существующий файл для чтения и записи. |
| w+ | Открывает файл для чтения и записи. Если файл существует, то его содержимое будет уничтожено. |
| a+ | Открывает файл для чтения и записи. Создаёт файл если он не существует. |

Дополнительно для определения типа файла могут указываться символы:

Табл. 12. Режимы доступа к файлу

| Параметр *mode* | Режим доступа |
|---|---|
| t | Текстовый файл |
| b | Двоичный файл (по умолчанию) |

По умолчанию режим доступа ‘*rb'* (чтение двоичного файла).

При открытии файла для обновления можно выполнять операции как ввода, так и вывода в результирующем потоке. Однако за операцией вывода не может напрямую идти операция ввода без операции позиционирования файла (функция *mseek*()). Также, за операцией ввода не может идти операция вывода без промежуточной операции позиционирования файла пока операция ввода не встретит конец файла.

При открытии файла для добавления записи (то есть, когда параметр *mode* равен *a* или *a+*) невозможно перезаписать информацию, которая уже есть в файле. Для смены положения указателя позиции в двоичном файле в любое

место файла можно использовать функцию *mseek*() (описана ниже), но, когда вывод записан в файл, текущий указатель позиции в файле игнорируется. Весь вывод записывается в конец файла и указатель позиции в файле перемещается в конец вывода.

Выходной параметр *fd* – это дескриптор файла (положительное целое число), второй параметр *err* – это индикатор ошибки, который может принимать следующие значения: 0 – нет ошибки; -1 – больше нет логических модулей; -2 – не могу открыть файл; -3 – больше нет памяти; -4 – некорректное имя; -5 – некорректный статус.

Файл, работа с которым закончена, необходимо закрыть, с помощью функции *mclose*(). Существует два формата вызова этой функции, в первом случае закроется файл с номером *fd*, во втором – все открытые файлы. При нормальном закрытии функция возвращает нулевое значение.

*—>mclose(fd)*

*—>mclose('all')*

При использовании *mclose('all')* внутри файла-сценария SciLab она закроет не только открытые файлы, но и сам сценарий, и последующие команды не будут выполняться.

При выборке информации из текстового или двоичного файла, открытого для чтения, может наступить момент, когда прочитана последняя порция данных и в файле больше ничего нет. Контроль за тем, достиг ли указатель конца данных (end-of-file), обеспечивает функция:

*—>k=meof(fd);*

Функция *meof*() вернёт ненулевое значение, если конец файла был достигнут в предыдущем вызове *mget*() или *mgetstr*().

*Работа с двоичными файлами*

Указатель файла смотрит на текущую порцию данных. При чтении эта информация извлекается из файла, а указатель автоматически перемещается на следующую доступную порцию, если таковая еще есть. При выводе в файл новая информация записывается, начиная с адреса, на который смотрит указатель. В двоичных файлах возможно дополнительное перемещение указателя.

Текущая позиция определяется как смещение в байтах относительно начала файла. Для открытого файла с номером *fd* текущее положение указателя можно узнать, обратившись к функции *mtell()*:

*—>pos=mtell(fd);*

Функция *mseek*() позволяет задать текущую позицию в двоичном файле:

*—>mseek(n[,fd,flag])*

Функция задает новое положение для следующей операции ввода или вывода потока *fd* (дескриптор файла; по умолчанию -1 – последний открытый файл) на расстоянии (со знаком) *n* байт от начала (*flag='set'*, по умолчанию), от текущего положения (*flag='cur'*) или от конца файла (*flag= 'end'*).

Функция *mseek*() позволяет индикатору положения в файле быть установленным за пределами конца существующих данных в файле. Если

данные будут позднее записаны в это место, то последующее чтение этих данных в пропущенном месте будет возвращать ноль до тех пор, пока данные не будут действительно записаны в пропущенное место.

Для чтения байтов или слов в заданном двоичном формате используются функции *mget*() и *mgeti*(). Разница их в том, что первая возвращает число удвоенной точности, а вторая – целочисленное значение типа int. Синтаксис функций практически совпадает:

—>*x=mget([n,type,fd])*

—>*x=mgeti([n,type,fd])*

Здесь *n* – положительное число, количество считываемых данных; *fd* – идентификатор файла (для последнего открытого файла можно указать -1); *type* – строка, двоичный формат, используемый для записи всех элементов *x*:

Табл. 13. Типы данных

| Символ | Тип данных |
|---|---|
| *d* | удвоенная точность, double, только для функции *mget*() |
| *f* | плавающая запятая, float, только для функции *mget*() |
| *l* | длинное, long, значение по умолчанию, только для функции *mget*() |
| *i* | целочисленное, int |
| *s* | короткое, short |
| *c* | символ, char |

Табл. 14. Необязательные флаги

| Символ | Необязательный флаг |
|---|---|
| *u..* | беззнаковый (в сочетании с одним из вышеперечисленных типов, кроме *f* и *d*) |
| *..l* | прямой порядок байтов (в сочетании с одним из вышеперечисленных типов) |
| *..b* | обратный порядок байтов (в сочетании с одним из вышеперечисленных типов) |

Данные считываются в положении, на которое в данный момент указывает указатель файла и передвигает индикатор далее соответствующим образом.

Для записи байта или слова в заданном двоичном формате применяется функция:

—>*mput(x[,type,fd])*

Здесь *x* – вектор целых чисел или чисел с плавающей запятой, которые нужно записать в файл с идентификатором *fd*; *type* – двоичный формат, используемый для записи всех элементов *x* (см. табл. 13 и 14). Данные записываются в место, на которое в данный момент указывает указатель положения в файле, и передвигает указатель дальше соответствующим образом.

Для записи символьной строки *str* в двоичный файл с идентификатором *fd* применяется функция:

—>*mputstr(str[,fd]);*

*Работа с текстовыми файлами*

Содержимым текстового файла являются строки – цепочки символов, завершающиеся парой управляющих символов.

Чтение строки из файла осуществляется с помощью функции *mgetl*():

*—>txt=mgetl(fd[,m])*

Функция читает строки из файла с идентификатором *fd* в вектор столбец строковых значений *txt*. Если не указано количество строк, которые нужно прочитать, *m*, или указано -1, то будут считываться все строки до тех пор, пока не встретится конец файла. Если конец файла встретится прежде, чем будет прочитано *m* строк, то будет прочитано, только то, что имеется. Проверить встретился ли конец файла можно с помощью функции *meof*().

Также для чтения символьной строки *str* из файла с идентификатором *fd* можно воспользоваться функцией:

*—>str=mgetstr(n,[fd])*

Ее отличие от функции *mgetl*() в том, что *n* – это количество символов (неотрицательное целое число), которое необходимо прочитать. Если конец файла достигнут прежде, чем будут прочитаны *n* символов, то *mgetstr*() возвращает лишь те значения, которые удалось прочитать.

Для записи вектора строк txt в файл с идентификатором fd применяется функция:

*—>r=mputl(txt,fd)*

Функция возвращает значения *%t* или *%f* для проверки правильно ли записан файл.

Для форматирования и конвертирования и вывода данных в командное окно SciLab используется функция:

*—>mprintf(format,a1,…,an)*

Строка *format* описывает формат, который используется для записи данных *a1,…,an*. Синтаксис формата очень близок к синтаксису функции *printf*() в языке C. Аналогичным образом работает функция *mfprintf*(), которая преобразует, форматирует и записывает данные в файл с идентификатором *fd*.

*—>mfprintf(fd,format,a1,…,an)*

Строка *format* для этих функций имеет следующий синтаксис:

- Знак % («процент»)
- Ноль или более параметров, которые изменяют спецификацию формата. Они могут быть следующие:
  - ✓ - («минус»): результат преобразования выравнивается по левому краю, в поле.
  - ✓ + («плюс»): результат преобразования начинается со знака (+ или -).
  - ✓ «пробел»: префикс пробела к результату, если первый символ указанного формата не является знаком. Если одновременно указать «пробел» и +, то «пробел» игнорируется.
  - ✓ #: Преобразует значение в альтернативную форму. Для форматов c, d, i, s и u параметр # не действует. Для

преобразования o, # увеличивает точность, для добавления в качестве первой цифры результата 0 (нуля). Для преобразований x и X ненулевой результат имеет префикс 0x или 0X. Для преобразований e, E, f, g и G результат всегда содержит десятичную точку, даже если за ней не следует ни одна цифра. Для преобразований g и G конечные нули не удаляются из результата.

- ✓ 0: заполнитель к ширине поля, используя ведущие нули (после любого указания знака или базы) для форматов D, i, o, u, x, X, e, f, g и G; без заполнения пространства не выполняется. Если появляются флаги 0 и \- (тире), флаг 0 игнорируется. Для преобразований d, i, o u, x и X, если указана точность, флаг 0 также игнорируется.

- Необязательное десятичное число, задающее минимальную ширину поля. Если преобразованное значение содержит меньше символов, чем ширина поля, поле дополняется слева длиной, указанной шириной поля. Если задана опция регулировки по левому краю, поле дополняется справа.
- Необязательная точность. Точность задается точкой «.», далее следует десятичное число. Если точность не задана, параметр обрабатывается как 0 (ноль). Точность определяет:
  - ✓ Минимальное количество цифр для преобразования d, u, o, x или X
  - ✓ Число цифр после десятичной запятой для преобразования e, E и f
  - ✓ Максимальное число значащих цифр для преобразований g и G
  - ✓ Максимальное число символов, выводимых из строки в преобразовании s
- Символ, указывающий тип применяемого преобразования:
  - ✓ %: Преобразование не выполняется. Отображается %.
  - ✓ d, i: принимает целочисленное значение и преобразует его в десятичную запись со знаком.
  - ✓ o: принимает целочисленное значение и преобразует его в беззнаковую восьмеричную запись.
  - ✓ x, X: принимает целочисленное значение и преобразует его в шестнадцатеричную запись без знака.
  - ✓ f: принимает значение типа float или double и преобразует его в десятичную запись в формате %[\-]ddd.ddd. Количество цифр после десятичной точки равно спецификации точности.
  - ✓ e, E : принимает вещественное и преобразует его в экспоненциальную форму %[\-]d.ddde+/\-dd. Перед десятичной запятой стоит одна цифра, а число цифр после десятичной запятой равно спецификации точности.

- ✓ g, G: принимает вещественное и преобразует его в стиле символов преобразования e, E или f с точностью, задающей количество значащих цифр. Нули удаляются из результата. Десятичная точка отображается, только если за ней следует цифра. Используемый стиль зависит от преобразованного значения. Стиль e (E, если используется флаг G) приводит только к тому, что показатель степени, полученный в результате преобразования меньше -4, или если он больше или равен точности.
- ✓ c: принимает и отображает целочисленное значение, преобразованное в символ.
- ✓ s: принимает строковое значение и отображает символы от строки до конца или количество символов, указанное точностью. Если точность не указана, отображаются все символы до конца.

Примеры:

*—>mprintf('a string: %s\n', 'Scilab');*
*—>mprintf('an integer: %4d\n', 10);*
*—>mprintf('a left justified integer: %-4d\n', 10);*
*—>mprintf('an integer converted to float: %#fd\n',10);*
*—>mprintf('an integer with a sign: %+4d\n', -10);*
*—>mprintf('an integer padded with zeros: %04d\n', 10);*
*—>mprintf('an unsigned integer: %u\n', 10);*
*—>mprintf('an integer converted to hexadecimal: %x\n', 10);*
*—>mprintf('a float: %3.2d\n', %pi);*
*—>mprintf('a float (exponential form): %3.2e\n', %pi);*
*—>mprintf('a character: %c\n', 'aaa');*

Для считывания данных из файла с идентификатором *fd* согласно заданному формату *format* применяется функция:

*—>[n,v1,...,vm]=mfscanf([k,]fd,format)*

Для считывания символов из окна SciLab с применением формата *format* следует использовать функцию:

*—>[n,v1,...,vm]=mscanf([k,]format)*

Здесь *k* – целое число, количество раз, которое нужно использовать *format*; *n* – количество данных (целое число); *v1,…,vm* – выходные аргументы, которые считаны и интерпретированы в соответствии с форматом *format*, если их указано больше, чем *n*, то последние аргументы с (*n+1*)-го по *m*-ый останутся пустыми.

Каждая спецификация преобразования в параметре *format* содержит следующие элементы:

- символ % («процент»);
- необязательный символ подавления присвоения * («звёздочка»);
- необязательное числовое значение максимальной ширины поля;
- код преобразования.

Спецификация преобразования имеет следующий синтаксис:

*[*][ширина][размер]код_преобразования.*

Результаты преобразования помещаются в аргументы *v1,...,vm* (если не указано обратное с помощью символа «звездочка»). Подавления присвоения предоставляет способ описать входное поле, которое следует пропустить. Для него аргумент *vi* не указывается. Входное поле является строкой символов-непробелов. Оно простирается до ближайшего некорректного символа или до тех пор, пока ширина поля, если она указана, не кончится.

Код преобразования указывает, как интерпретировать входное поле:

- % - принимает отдельный символ % («процент»), вводимый в этом месте; присвоение не делается.
- d, i – принимает десятичное целое число.
- u – принимает беззнаковое десятичное целое число.
- – принимает восьмеричное целое число.
- x – принимает шестнадцатеричное целое число.
- e, f, g – принимает число с плавающей запятой.

  Следующее поле преобразуется соответствующим образом и сохраняется через соответствующий параметр, который должен быть указателем на число с плавающей запятой. Формат ввода для чисел с плавающей запятой является строкой цифр со следующими необязательными характеристиками:
  - ✓ это может быть значение со знаком;
  - ✓ это может быть экспоненциальное значение, содержащее десятичную запятую, за которой следует экспоненциальное поле, которое состоит из E или e, за которым следует целое число (может быть со знаком);
  - ✓ это может быть одно из специальных значений %inf, %nan.
- s – принимает строку символов;
- c – ожидается символьное значение. Нормальный пропуск пробела подавляется.
- %lg – получает значение в виде числа удвоенной точности (double).

Например, следующая команда позволяет прочитать из файла с идентификатором *u* два числа с экспоненциальным форматом:

*—>[n,a,b]=mfscanf(u,'%e %e')*

## *Задание 7*

Вариант 1

1. Записать в двоичный файл *myexample.bin* строковый массив *A*:

   *'A free market economy has no government intervention'*
2. Закрыть файл *myexample.bin*.

3. Открыть двоичный файл *myexample.bin* и прочитать из него все данные (для этого определить длину строки *A*). Представить прочитанные данные в символьной форме в виде строки.

4. Открыть текстовый файл *myexample.txt* для записи и записать в него массив строк:

$$B = \begin{matrix} \text{‘A free market economy ’} \\ \text{′has no′} \\ \text{′government intervention′} \end{matrix}$$

5. Закрыть все файлы.

6. Открыть файл *myexample.txt* и прочитать из него первую строку с помощью функции *mgetl*().

7. Прочитать из файла *myexample.txt* 3 символа с помощью функции *mgetstr*().

8. Прочитать из файла *myexample.txt* оставшиеся символы.

9. Перевести указатель на начало файла.

10. Прочитать из этого файла пять слов в пять переменных.

11. Вывести в командное окно данные *a=15 b=-15* в виде:

```
-->mp
 15
 -15
```

Вариант 2

1. Записать в двоичный файл *myexample.bin* строковый массив *A*:

*‘Macroeconomics is the study of the economy as a whole’*

2. Закрыть файл *myexample.bin*.

3. Открыть двоичный файл *myexample.bin* и прочитать из него все данные (для этого определить длину строки *A*). Представить прочитанные данные в символьной форме в виде строки.

4. Открыть текстовый файл *myexample.txt* для записи и записать в него массив строк:

$$B = \begin{matrix} \text{‘Macroeconomics ’} \\ \text{′is the study of the economy′} \\ \text{′as a whole′} \end{matrix}$$

5. Закрыть все файлы.

6. Открыть файл *myexample.txt* и прочитать из него первую строку с помощью функции *mgetl*().

7. Прочитать из файла *myexample.txt* 3 символа с помощью функции *mgetstr*().

8. Прочитать из файла *myexample.txt* оставшиеся символы.

9. Перевести указатель на начало файла.

10. Прочитать из этого файла пять слов в пять переменных.

11. Вывести в командное окно данные *a=15 b=-15* в виде:

```
-->mp
+15
 -15
```

Вариант 3

1. Записать в двоичный файл *myexample.bin* строковый массив *A*:

*'GNP measures the total income of the economy'*

2. Закрыть файл *myexample.bin*.

3. Открыть двоичный файл *myexample.bin* и прочитать из него все данные (для этого определить длину строки *A*). Представить прочитанные данные в символьной форме в виде строки.

4. Открыть текстовый файл *myexample.txt* для записи и записать в него массив строк:

$$B = \begin{matrix} \text{'GNP measures'} \\ \text{'the total income'} \\ \text{'of the economy'} \end{matrix}$$

5. Закрыть все файлы.

6. Открыть файл *myexample.txt* и прочитать из него первую строку с помощью функции *mgetl*().

7. Прочитать из файла *myexample.txt* 3 символа с помощью функции *mgetstr*().

8. Прочитать из файла *myexample.txt* оставшиеся символы.

9. Перевести указатель на начало файла.

10. Прочитать из этого файла пять слов в пять переменных.

11. Вывести в командное окно данные a=15 b=-15 в виде:

```
-->mp
17
 -15
```

Вариант 4

1. Записать в двоичный файл *myexample.bin* строковый массив *A*:

*'Firms and government finance R&D activities'*

2. Закрыть файл *myexample.bin*.

3. Открыть двоичный файл *myexample.bin* и прочитать из него все данные (для этого определить длину строки *A*). Представить прочитанные данные в символьной форме в виде строки.

4. Открыть текстовый файл *myexample.txt* для записи и записать в него массив строк:

$$B = \begin{matrix} \text{'Firms and government'} \\ \text{'finance'} \\ \text{'R\&D activities '} \end{matrix}$$

5. Закрыть все файлы.

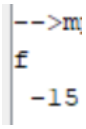
6. Открыть файл *myexample.txt* и прочитать из него первую строку с помощью функции *mgetl*().

7. Прочитать из файла *myexample.txt* 3 символа с помощью функции *mgetstr*().

8. Прочитать из файла *myexample.txt* оставшиеся символы.

9. Перевести указатель на начало файла.

10. Прочитать из этого файла пять слов в пять переменных.
11. Вывести в командное окно данные *a=15 b=-15* в виде:



# Список литературы


1. Говорухин В., Цибулин В. Компьютер в математическом исследовании. Учебный курс. – СПб.: Питер, 2001. – 624 с.
2. Капитанов Д.В., Капитанова О.В. Введение в MatLab: Лабораторный практикум. - Нижний Новгород: Нижегородский госуниверситет, 2016. – 65 с.
3. Квасов, Б.И. Численные методы анализа и линейной алгебры. Использование Matlab и Scilab [Электронный ресурс]: учебное пособие / Б.И. Квасов. — Электрон. дан. — Санкт-Петербург: Лань, 2016. — 328 с. — Режим доступа: https://e.lanbook.com/book/71713.
4. Решение инженерных задач в среде Scilab [Электронный ресурс]: учебное пособие / А.Б. Андриевский [и др.]. — Электрон. дан. — Санкт-Петербург: НИУ ИТМО, 2013. — 97 с. — Режим доступа: https://e.lanbook.com/book/71062.
5. SciLab Documentation // http://www.scilab.org [Режим доступа: свободный, 09.11.2018]

Денис Владимирович **Капитанов,**
Ольга Владимировна **Капитанова**

# Введение в SciLab

***Практикум***

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
603950, Нижний Новгород, пр. Гагарина, 23