

МИНОБРНАУКИ РОССИИ

---

Санкт-Петербургский государственный электротехнический  
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

---

А. С. ВЕТЧИНКИН, О. Ю. ЛУКОМСКАЯ, А. Г. ШПЕКТОРОВ

## **СОСТАВЛЕНИЕ АЛГОРИТМА И НАПИСАНИЕ ПРОГРАММ ОБРАБОТКИ МАССИВА ДАННЫХ**

Учебно-методическое пособие  
к курсовому проекту

Санкт-Петербург  
Издательство СПбГЭТУ «ЛЭТИ»  
2017

УДК 004.4(075)

ББК 3973-018

С 48

**Ветчинкин А. С., Лукомская О. Ю., Шпекторов А. Г.**

С 48 Составление алгоритма и написание программ обработки массива данных: учеб.-метод. пособие к курсовому проекту. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017. 32 с.

ISBN

Содержит описания основных разделов курсового расчета по дисциплине «Программирование и основы алгоритмизации». Рассмотрены принципы формирования математической модели по постановке задачи. Приведены основные алгоритмы решения задачи, встроенные конструкции и функции среды программирования MATLAB, типовые варианты заданий на курсовой расчет.

Предназначены для студентов бакалавриата, обучающихся по направлению 27.03.04 «Управление в технических системах».

УДК 004.4(075)

ББК 3973-018

Рецензенты:

Утверждено

редакционно-издательским советом университета  
в качестве учебно-методического пособия

ISBN

© СПбГЭТУ «ЛЭТИ», 2017

*Цель курсовой работы* – выработка у студентов умения и практических навыков разработки блок-схем алгоритмов решения задач, написания кодов в программной среде MATLAB, а также навыков по описанию, оформлению и представлению результатов проделанной работы.

Курсовая работа включает в себя следующие основные этапы:

1. Постановка задачи и формирование математической модели.
2. Выбор алгоритма поиска кратчайшего пути и его тестирование.
3. Разработка блок-схемы алгоритма решения задачи.
4. Написание и отладка программы на языке MATLAB.
5. Представление и анализ результатов работы программы.

## **1. ПОСТАНОВКА ЗАДАЧИ И ФОРМИРОВАНИЕ МАТЕМАТИЧЕСКОЙ МОДЕЛИ**

Одной из стадий решения вычислительной задачи является построение математической модели. Под математической моделью понимается совокупность математических объектов и отношений, отображающих объекты и отношения, существующие в некоторой области реального мира [1]. Примерами математических моделей могут быть системы дифференциальных и алгебраических уравнений, ориентированные и неориентированные графы и пр.

Вид математической модели в значительной степени определяется постановкой задачи, но не только ею. Объекты и отношения реального мира, как правило, настолько сложны, что найти для них полное математическое описание не представляется возможным. Поэтому для одного и того же объекта (или системы объектов) можно построить множество различных математических моделей, описывающих поведение реального объекта с разной степенью точности. Поскольку использование математической модели связано с применением компьютерной техники и вычислительными затратами, необходимо искать баланс между точностью описания и простотой модели (чем проще модель, тем меньше времени занимает ее вычисление). Важную роль при построении математической модели играют допущения, которые можно принять для упрощения описываемых объектов и отношений, а также ограничения, которые необходимо учесть.

Рассмотрим процесс формирования математической модели на основе постановки задачи и ряда допущений. Задача состоит в том, чтобы перевести подвижный объект из одной точки пространства в другую за минимальное время. Это может быть автомобиль, движущийся по пересеченной местности,

надводный корабль или подводная лодка, двигающиеся на поверхности воды или на глубине, подводный робот, ползающий по морскому дну, космический аппарат в космосе. Полагаем, что подвижный объект оборудован двигателем, обеспечивающим в любой момент движение в любом направлении. Движение объектов обычно можно описать при помощи второго закона Ньютона, оперируя понятиями массы и силы тяги, которая уравнивается силой сопротивления среды. Однако в данной работе основная цель – выбор траектории движения, поэтому динамикой объекта также можно пренебречь. Считаем, что объект движется в однородной среде с постоянной скоростью, при этом его масса не меняется, что, строго говоря, для ряда реальных объектов также несправедливо, поскольку объекты (автомобили, корабли, ракеты) расходуют запас топлива. Поэтому введем допущение, что энергию для движения объект получает от аккумуляторной батареи.

Наконец, задача выбора траектории не будет иметь смысла без ограничений. С учетом уже принятых допущений ограничение очевидно: время работы аккумуляторной батареи конечно. Для реализуемости поставленной задачи в пространстве движения следует задать ряд пунктов подзарядки аккумулятора или дозаправки. Координаты расположения этих точек в пространстве можно считать постоянными. Таким образом, задача становится более конкретной: найти траекторию движения от одной точки пространства к другой за минимальное время, притом траектория должна проходить через заданные точки, а отрезки траектории ограничены с учетом времени работы аккумуляторной батареи.

После уточнения и конкретизации задачи можно выбрать вид математической модели для ее решения. В данном случае задачу удобнее всего решать с применением теории графов. Под графом понимается совокупность множества вершин и множества ребер, соединяющих вершины (рис. 1.1, а, б).

Граф называется *ориентированным*, если его ребра имеют направление (рис. 1.1, а) и *неориентированным*, если направление не задано. Граф называется *связным*, если последовательность ребер соединяет его две любые вершины, такая последовательность представляет собой путь из одной вершины в другую [2]. Если хотя бы для одной пары вершин не существует пути, граф называется *несвязным*.

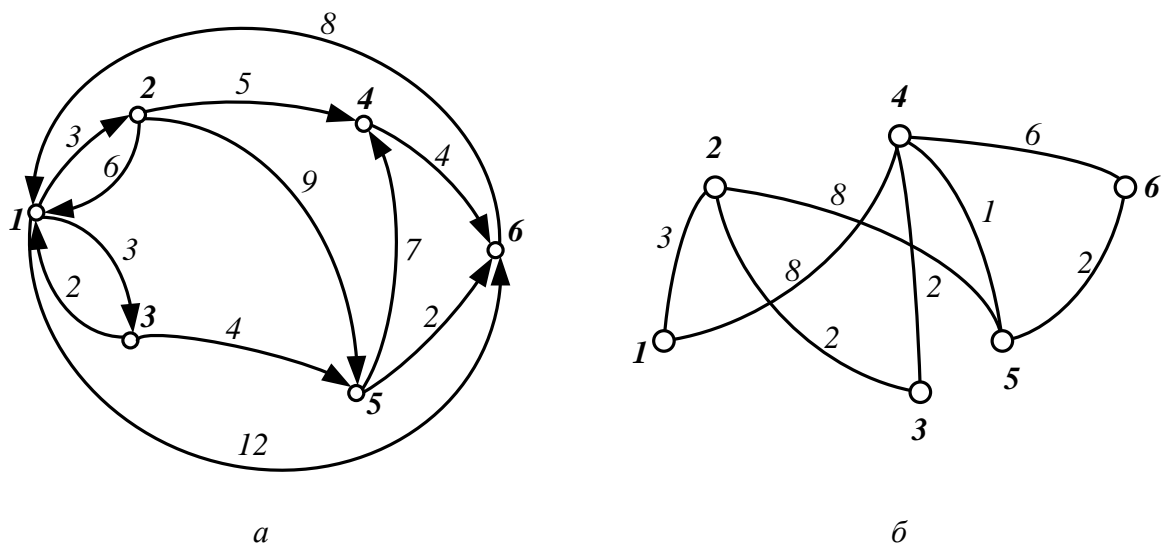


Рис. 1.1

Существует несколько способов математического описания графов. К примеру, графы можно описывать при помощи матрицы смежности – квадратной матрицы, размерность которой равна количеству вершин, а внедиагональные элементы равны единице, если ребро между соответствующими вершинами существует, или нулю, если не существует. Для графов, приведенных на рис. 1.1, матрицы смежности имеют вид:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{– для рис. 1.1, а;}$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \text{– для рис. 1.1, б.}$$

Матрица будет симметричной для неориентированного графа и несимметричной для орграфа. Каждому ребру графа можно поставить в соответствие некоторое число, называемое весом ребра (рис. 1.1, а, б). Тогда исходная задача сводится к поиску такого пути в графе от одной вершины к другой, чтобы сумма длин ребер была минимальной.

Вес графа в общем случае зависит от характера оптимизационной задачи, т. е. от вида критерия. В нашем случае критерием служит время прохождения графа, так что веса должны быть пропорциональны времени прохождения ребер. При допущении, что скорость объекта постоянная, можно перейти от временных

характеристик пути к метрическим. Таким образом, в качестве веса ребер графа в данной задаче можно принять их длину.

Порядок расчета длины зависит от математического описания среды, которая до сих пор принималась несущественной. Возможны следующие варианты описания среды:

- плоскость;
- трехмерная поверхность;
- трехмерное пространство.

Плоскость можно использовать при моделировании движения автомобиля по дорогам, корабля по водной поверхности или подводного робота по ровному участку морского дна. Трехмерная поверхность – более универсальная модель среды (плоскость – частный случай поверхности), позволяющая учесть неровности дороги или дна, гористую местность и т. п. Трехмерное пространство позволяет описать движение в космосе, в воздухе или на глубине. Примем в качестве математического описания среды поверхность, описываемую функцией

$$z = f(x, y),$$

где  $x, y$  – координаты точки пространства в декартовой плоскости;  $z$  – высота (глубина), определяемая координатами  $x, y$ .

Таким образом, математическая модель для поставленной задачи включает:

- описание неориентированного графа;
- описание среды движения;
- порядок определения весов (длин) ребер графа;
- описание графа с учетом длин.

Данная задача даже с учетом введенных упрощений может найти применение для прикладных задач разного характера. Вычисление длин ребер может быть сведено к расчету метрических расстояний только в том случае, когда среда задана однородной плоскостью без ограничений. Если, например, представить, что вершины графа соответствуют населенным пунктам на географической карте, то расчет длин сведется к вычислению криволинейных интегралов, соответствующих соединяющим пункты дорогам. Однако общая постановка задачи и методы решения от этого не изменятся.

Средством математического описания графов с взвешенными ребрами являются матрицы весов (длин), похожие по структуре на матрицы смежности, но имеющие веса в качестве элементов. Элементы матрицы длин вычис-

ляются следующим образом:

$$L_{ij} = \begin{cases} 0, i = j, \\ l_{ij}, a_{ij} = 1, \\ \infty, a_{ij} = 0, \end{cases} \quad (1.1)$$

где  $a_{ij}$  – элементы матрицы смежности. Для графов на рис. 1.1 матрицы имеют вид

$$L_1 = \begin{bmatrix} 0 & 3 & 3 & \infty & \infty & 12 \\ 6 & 0 & \infty & 5 & 9 & \infty \\ 2 & \infty & 0 & \infty & 4 & \infty \\ \infty & \infty & \infty & 0 & \infty & 4 \\ \infty & \infty & \infty & 7 & 0 & 2 \\ 8 & \infty & \infty & \infty & \infty & 0 \end{bmatrix}; \quad L_2 = \begin{bmatrix} 0 & 3 & \infty & 8 & \infty & \infty \\ 3 & 0 & 2 & \infty & 8 & \infty \\ \infty & 2 & 0 & 2 & \infty & \infty \\ 8 & \infty & 2 & 0 & 1 & 6 \\ \infty & 8 & \infty & 1 & 0 & 2 \\ \infty & \infty & \infty & 6 & 2 & 0 \end{bmatrix},$$

где  $L_1$  и  $L_2$  – матрицы длин для графов соответственно на рис. 1.1, а и б.

**Уточнение математической модели.** Простые математические модели описывают идеализированное поведение процессов и систем. Для учета реальных факторов, влияющих на поведение объекта, модель уточняют, добавляя различные физические эффекты.

Допустим, предложенную ранее модель для решения траекторной задачи нужно использовать для оценки общего времени пути из начальной точки в конечную. Расчет не представляет сложности, однако модель не учитывает времени, затраченного на подзарядку аккумуляторов (или дозаправку) подвижного объекта. Если считать пункты подзарядки идентичными по времени подзарядки, то учет этого времени сделает модель более реалистичной без существенных усложнений.

В данной задаче полагаем, что время подзарядки аккумуляторов одинаковое для всех пунктов и не зависит от текущего состояния аккумулятора. Можно далее уточнять модель: ввести подзарядку с учетом разрядки или задавать разное время подзарядки на разных пунктах. Правда, в последнем случае придется отказаться от метрических весов графа и рассчитывать веса ребер пропорционально затраченному времени на путь и подзарядку.

Итак, для формализации задачи получения математической модели и численного решения требуются следующие исходные данные:

- скорость подвижного объекта;
- время работы аккумулятора;

- время подзарядки аккумулятора;
- координаты начальной и конечной точек объекта;
- координаты промежуточных точек – пунктов подзарядки.

На основе этих данных можно построить неориентированный граф и сформировать матрицу длин для его описания в вычислительной машине. Теперь задача поиска минимального пути на графе справедлива.

## 2. АЛГОРИТМЫ ПОИСКА ОПТИМАЛЬНОЙ ТРАЕКТОРИИ

Задача поиска оптимальной траектории, соответствующей минимальной длине на взвешенном графе широко известна. Наиболее популярны следующие алгоритмы поиска:

- алгоритм Форда–Беллмана.
- алгоритм Дейкстры;
- алгоритм Флойда;

Существуют и другие алгоритмы, но они, как правило, представляют собой модифицированные варианты алгоритмов, приведенных ранее.

### 2.1. Алгоритм Форда–Беллмана

Алгоритм Форда–Беллмана основан на теореме, согласно которой любой участок минимального пути, соединяющий некоторые вершины графа, представляет минимальный путь между этими вершинами. Алгоритм предполагает вычисление всех минимальных путей из начальной вершины.

Исходные данные: матрица длин  $L$ , начальная вершина имеет индекс 1.

Требуется выполнить следующие действия:

1. Определить для каждой вершины набор величин  $\lambda_i^{(k)}$ , означающих длину пути от 1-й до  $i$ -й вершины, состоящего не более чем из  $k$  ребер. Набор формируется по правилу

$$\lambda_i^{(k+1)} = \min_j \{ \lambda_i^{(k)} + l_{ij} \}, \quad k = 0 \dots n-1, \quad \lambda_i^{(0)} = \begin{cases} 0, & i = 1, \\ \infty, & i \neq 1, \end{cases} \quad (1.2)$$

где  $n$  – число вершин;  $k$  – число ребер.

Величины  $\lambda_i^{(k)}$  можно представить в виде квадратной матрицы  $\Lambda$ . Если значение  $\lambda_i^{(n-1)}$  отлично от  $\infty$ , значит, оно равно длине минимального пути в вершину  $i$ , в противном случае данная вершина недостижима (граф несвязный).



2. Определить последовательность ребер, приводящих из 1-й в  $i$ -ю вершину за путь  $\lambda_i^{(k)}$ . Для этого нужно определить номера  $i_1, i_2, \dots, i_k$ , для которых выполняются соотношения:

$$\begin{aligned}\lambda_i^{(k)} &= \lambda_{i_1}^{(k-1)} + l_{i_1, i}; \\ \lambda_{i_1}^{(k-1)} &= \lambda_{i_2}^{(k-2)} + l_{i_2, i_1}; \\ &\dots \\ \lambda_{i_{k-1}}^{(1)} &= \lambda_{i_k}^{(0)} + l_{i_k, i_{k-1}}.\end{aligned}\tag{1.3}$$

Для графов, приведенных на рис. 1.1, по алгоритму Форда–Беллмана матрицы минимальных путей  $\Lambda_1$  и  $\Lambda_2$  имеют вид

$$\Lambda_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \infty & 3 & 3 & 3 & 3 & 3 \\ \infty & 3 & 3 & 3 & 3 & 3 \\ \infty & \infty & 8 & 8 & 8 & 8 \\ \infty & \infty & 7 & 7 & 7 & 7 \\ \infty & 12 & 12 & 9 & 9 & 9 \end{bmatrix}; \quad \Lambda_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \infty & 3 & 3 & 3 & 3 & 3 \\ \infty & \infty & 5 & 5 & 5 & 5 \\ \infty & 8 & 8 & 7 & 7 & 7 \\ \infty & \infty & 9 & 9 & 8 & 8 \\ \infty & \infty & 14 & 11 & 11 & 10 \end{bmatrix}.$$

Для графа, приведенного на рис. 1.1, *а*, минимальный путь до любой вершины достигается максимум за три ребра. По формулам (1.3) легко восстановить, что путь до 6-й вершины с конца проходит через 5-ю ( $\lambda_6^{(3)} = \lambda_5^{(2)} + l_{56}$ ), а далее через 3-ю ( $\lambda_5^{(2)} = \lambda_3^{(1)} + l_{35}$ ) вершины. Таким образом, искомый минимальный путь имеет вид 1–3–5–6. Длина минимального пути равна 9. Минимальный путь для графа на рис. 1.1, *б* содержит 5 ребер. Аналогично (1.3) можно восстановить в обратном порядке искомую последовательность вершин 1–2–3–4–5–6, длина минимального пути равна 10.

## 2.2. Алгоритм Дейкстры

Этот алгоритм, предложенный в 1959 г. Дейкстрой, считается одним из наиболее эффективных алгоритмов решения задачи нахождения в графе минимальных путей от начальной вершины до всех остальных при положительных длинах дуг.

Алгоритм также использует принцип формирования минимального пути из минимальных участков. Однако итерационный процесс алгоритма основан на формировании последовательности вершин, для которых гарантированно известны минимальные пути. Алгоритм фактически строит минимальное дерево (под деревом понимается связный граф без циклов), добавляя на каждой итерации к дереву одну вершину и одно ребро.

Для реализации алгоритма Дейкстры требуется итерационно сформировать два вектора: вектор минимальных путей  $\mathbf{D}$ , и вектор индексов учтенных вершин  $\mathbf{Ind}$ , изначально не содержащий элементов.

Требуется выполнить следующие действия:

1. Проинициализировать начальные значения векторов (для  $j = 1$ ):

$$\mathbf{d}_i^{(1)} = \begin{cases} 0, & i = 1, \\ \infty, & i \neq 1, \end{cases} \quad \mathbf{Ind}_1 = 1. \quad (1.4)$$

2. Пересчитать вектор  $\mathbf{D}^{(j)}$  и  $\mathbf{Ind}$  для следующих итераций по выражениям:

$$\mathbf{d}_i^{(j+1)} = \begin{cases} \mathbf{d}_i^{(j)}, & i \in \mathbf{Ind}; \\ \min_{k=1, \dots, n} \left( \mathbf{d}_k^{(j)} + l_{ki} \right), & i \notin \mathbf{Ind}; \end{cases} \quad (1.5)$$

$$\mathbf{Ind}_j = \text{index} \left( \min_k \left( \mathbf{d}_k^{(j)} \right) \right), k \notin \mathbf{Ind}; j = 2, \dots, n,$$

где  $\text{index}(\ )$  – индекс минимального элемента вектора  $\mathbf{D}^{(j)}$ , не считая тех элементов, индексы которых уже присутствуют в векторе  $\mathbf{Ind}$ ;  $n$  – число вершин графа.

Если в векторе  $\mathbf{D}^{(j)}$  есть несколько элементов с минимальными значениями, нужно выбрать наименьший индекс.

Элементы вектора  $\mathbf{d}_i^{(n)}$  содержат длину минимального пути от первой вершины до  $i$ -й. Если в векторе остались элементы со значением бесконечности, значит, граф несвязный и вершины с соответствующим индексом недостижимы.

3. Восстановить последовательность ребер, приводящих из первой вершины в искомую, аналогично выражениям (1.3), так как последний столбец матрицы  $\mathbf{L}$  алгоритма Форда–Беллмана идентичен вектору  $\mathbf{D}^{(n)}$  алгоритма Дейкстры.

Для графа, представленного на рис. 1.1, *а*, будут получены следующие векторы:

$$\mathbf{D}^{(1)} = [0 \ \infty \ \infty \ \infty \ \infty \ \infty]^T; \quad \mathbf{Ind} = [1];$$

$$\mathbf{D}^{(2)} = [0 \ 3 \ 3 \ \infty \ \infty \ 12]^T; \quad \mathbf{Ind} = [1 \ 2];$$

$$\mathbf{D}^{(3)} = [0 \ 3 \ 3 \ 8 \ 12 \ 12]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3];$$

$$\mathbf{D}^{(4)} = [0 \ 3 \ 3 \ 8 \ 7 \ 12]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 5];$$

$$\mathbf{D}^{(5)} = [0 \ 3 \ 3 \ 8 \ 7 \ 9]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 5 \ 4];$$

$$\mathbf{D}^{(6)} = [0 \ 3 \ 3 \ 8 \ 7 \ 9]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 5 \ 4 \ 6];$$

Для графа, представленного на рис. 1.1, *б*, последовательность векторов имеет вид:

$$\mathbf{D}^{(1)} = [0 \ \infty \ \infty \ \infty \ \infty \ \infty]^T; \quad \mathbf{Ind} = [1];$$

$$\mathbf{D}^{(2)} = [0 \ 3 \ \infty \ 8 \ \infty \ \infty]^T; \quad \mathbf{Ind} = [1 \ 2];$$

$$\mathbf{D}^{(3)} = [0 \ 3 \ 5 \ 8 \ 11 \ \infty]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3];$$

$$\mathbf{D}^{(4)} = [0 \ 3 \ 5 \ 7 \ 11 \ \infty]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 4];$$

$$\mathbf{D}^{(5)} = [0 \ 3 \ 5 \ 7 \ 8 \ 13]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 4 \ 5];$$

$$\mathbf{D}^{(6)} = [0 \ 3 \ 5 \ 7 \ 8 \ 10]^T; \quad \mathbf{Ind} = [1 \ 2 \ 3 \ 4 \ 5 \ 6];$$

Деревья с минимальными путями, полученные в ходе выполнения алгоритма Дейкстры, представлены на рис. 2.1.

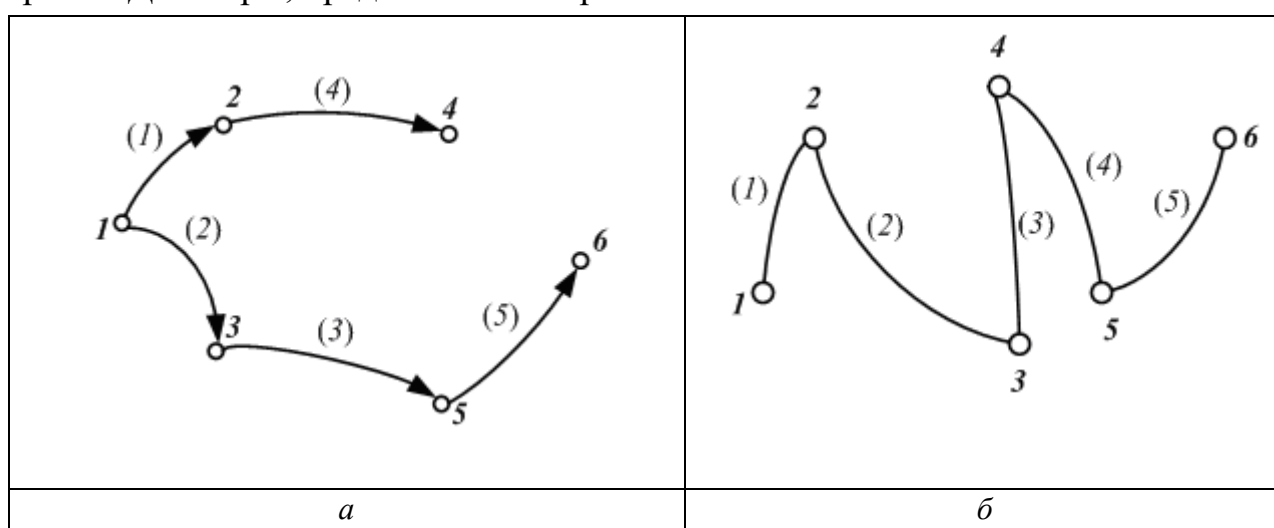


Рис. 2.1

По результатам выполнения алгоритма Дейкстры можно убедиться, что итоговые векторы **D** совпадают с последними столбцами матриц минимальных путей алгоритма Форда–Беллмана, поэтому можно восстановить последовательность вершин минимального пути аналогично (1.3). Следует отметить, что в векторе **D** не определено количество ребер, образующих минимальный путь, поэтому для восстановления вершин нужно использовать цикл с неопределенным числом итераций.

### 2.3. Алгоритм Флойда

Алгоритм Флойда, в отличие от рассмотренных ранее, определяет минимальные пути между всеми парами вершин в графе. Фактически алгоритм Флойда осуществляет последовательный перебор всех вершин, проверяя, проходит ли через вершину путь, более короткий, чем прочие пути.

Аналогичная задача может быть решена многократным применением алгоритма Форда–Беллмана или Дейкстры (последовательно задавая вершины графа как стартовые для поиска), однако реализация подобной процедуры потребовала бы значительных вычислительных затрат.

Реализация алгоритма Флойда сводится к формированию матрицы всех минимальных путей. Для неориентированного графа матрица будет симметричной, что позволяет сократить вычислительные затраты, заполняя только значения из верхнего (нижнего) треугольника матрицы.

Имея матрицу всех минимальных путей  $M$ , несложно восстановить порядок следования вершин любого пути. Исходной для алгоритма Флойда, как и раньше, является матрица весов (длин)  $L$ .

Рассмотрим алгоритм для общего случая (ориентированного графа с отрицательными весами). Требуется выполнить следующие действия:

1. Проинициализировать матрицу  $M$ :

$$M^{(1)} = L.$$

2. Пересчитать элементы матрицы для следующих итераций по выражениям

$$m_{ij}^{(k)} = \min \left( m_{ij}^{(k-1)}, m_{ik}^{(k-1)} + m_{kj}^{(k-1)} \right);$$

$$i = 1, \dots, n, \quad j = 1, \dots, n, \quad k = 2, \dots, n,$$

где  $n$  – количество вершин графа (порядок матриц  $L, M$ ).

3. Для двух заданных вершин восстановить последовательность прохождения вершин, соответствующую минимальному пути, по соотношениям, аналогичным (1.3).

Матрицы минимальных путей, полученные с помощью алгоритма Флойда для графов, приведенных на рис. 1.1, имеют вид

$$M_1 = \begin{bmatrix} 0 & 3 & 3 & 8 & 7 & 9 \\ 6 & 0 & 9 & 5 & 9 & 9 \\ 2 & 5 & 0 & 10 & 4 & 6 \\ 12 & 15 & 15 & 0 & 19 & 4 \\ 10 & 13 & 13 & 7 & 0 & 2 \\ 8 & 11 & 11 & 16 & 15 & 0 \end{bmatrix}; M_2 = \begin{bmatrix} 0 & 3 & 5 & 7 & 8 & 10 \\ 3 & 0 & 2 & 4 & 5 & 7 \\ 5 & 2 & 0 & 2 & 3 & 5 \\ 7 & 4 & 2 & 0 & 1 & 3 \\ 8 & 5 & 3 & 1 & 0 & 2 \\ 10 & 7 & 5 & 3 & 2 & 0 \end{bmatrix}.$$

У неориентированного графа матрица  $M_2$  симметричная. Легко убедиться, что первая строка матриц по значениям идентична вектору  $D$  алгоритма Дейкстры или последним столбцом матрицы  $\Lambda$  алгоритма Форда–Беллмана. Значит, по выражениям, аналогичным (1.3), можно восстановить порядок следования вершин, соответствующих минимальной длине пути. Более того, можно восстановить подобную последовательность для любых двух пар вершин, объединив их в матрицу или таблицу путей.

$$W_1 = \begin{bmatrix} 0 & (2) & (3) & (2-4) & (3-5) & (3-5-6) \\ (1) & - & (1-3) & (4) & (5) & (4-6) \\ (1) & (1-2) & 0 & (1-2-4) & (5) & (5-6) \\ (6-1) & (6-1-2) & (6-1-3) & 0 & (6-1-3-5) & (6) \\ (6-1) & (6-1-2) & (6-1-3) & (4) & 0 & (6) \\ (1) & (1-2) & (1-3) & (1-2-4) & (1-3-5) & 0 \end{bmatrix};$$

$$W_2 = \begin{bmatrix} 0 & (2) & (2-3) & (2-3-4) & (2-3-4-5) & (2-3-4-5-6) \\ - & 0 & (3) & (3-4) & (3-4-5) & (3-4-5-6) \\ - & - & 0 & (4) & (4-5) & (4-5-6) \\ - & - & - & 0 & (5) & (5-6) \\ - & - & - & - & 0 & (6) \\ - & - & - & - & - & 0 \end{bmatrix}.$$

Элементами матриц  $w_{ij}$  являются индексы вершин, через которые нужно пройти от вершины  $i$  до вершины  $j$  (индекс  $i$ -й вершины можно не указывать, так как он задан номером строки матрицы  $W$ ). Для неориентированного графа достаточно заполнить только верхнюю треугольную область, так как матрица минимальных путей симметрична, пути взаимно обратны. Для ориентированных графов матрицы маршрутов нужно определять полностью.

Достоинством алгоритма Флойда является его простота, недостатком – время выполнения.

Алгоритм поиска минимального пути для поставленной задачи выбирает разработчик. Допускается использовать другие алгоритмы или модификации рассмотренных ранее, однако необходимо соблюдать свойство массовости – алгоритм должен решать задачу для разных наборов исходных данных.

### **3. РАЗРАБОТКА БЛОК-СХЕМЫ АЛГОРИТМА**

При проектировании алгоритмов используют специальные графические элементы, называемые графическими блоками. Результатом алгоритмизации решения задачи становится блок-схема алгоритма, состоящая из некоторой последовательности таких графических блоков.

Блок-схема – это последовательность блоков, предписывающих выполнение определенных операций, а также связей между этими блоками. Внутри блоков указывается информация об операциях, подлежащих выполнению. Конфигурация и размеры блоков, а также порядок графического оформления блок-схем регламентированы нормативными документами ГОСТ 19002–80 и ГОСТ 19003–80 «Схемы алгоритмов и программ». Наиболее часто используемые блоки, элементы связей между ними и краткое пояснение к реализации блоков представлены в [3].

При соединении блоков следует использовать только горизонтальные и вертикальные линии потоков. Горизонтальные линии, имеющие направление справа налево, и вертикальные потоки, направленные снизу вверх, обязательно должны быть помечены стрелками. Прочие потоки допускается оставлять непомеченными. Линии потоков должны быть параллельны линиям внешней рамки или границам листа.

При проектировании алгоритмов применяются следующие общие правила:

- в начале алгоритма должны находиться блоки ввода значений исходных данных;
- после ввода значений исходных данных могут следовать блоки обработки данных либо вызова подпрограмм, если алгоритм предусматривает обработку данных в модулях;
- в конце алгоритма должны располагаться блоки вывода значений выходных данных;
- в алгоритме должны присутствовать только один блок начала и один блок окончания [4].

*Примечание:* при оформлении информации о выполняемых операциях следует избегать конструкций, операторов и функций языка программирования.

## 4. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА MATLAB

### 4.1. Условия и циклы

Условные операторы необходимы для организации принятия решения в ходе выполнения программы.

Условный оператор `if` в общем случае осуществляет проверку нескольких условий и записывается следующим образом:

```
if условие1
    операторы1
elseif условие2
    операторы2
elseif условие3
    операторы3
.....
else
    операторы4
end
```

Альтернативные ветви не являются обязательными частями оператора.

Оператор множественного выбора `switch` выполняет выбор одной альтернативы из нескольких по равенству переменной выбора одному из выборки значений:

```
switch (переменная_выбора)
    case значение1,
        операторы1
    case { значение2, значение3, значение4,...}
        операторы2
    ...
    otherwise,
        операторы
end
```

Операторы выполняются до тех пор, пока в коде не встретится следующее ключевое слово: `case`, либо `otherwise`, либо `end`. Необязательный оператор `otherwise` предлагает альтернативу для случая по умолчанию – когда переменная выбора не принимает предложенных значений.

Оператор цикла типа `for ... end` используется для организации вычислений с заданным числом повторяющихся итераций. Конструкция такого оператора имеет вид

```
for v=M
    операторы
end
```

Параметр  $M$  – это чаще всего вектор. В большинстве случаев он представляется с использованием оператора «:» в виде  $s:d:e$ , где  $s$  – начальное значение управляющей переменной цикла ( $v$ );  $d$  – приращение этой переменной;  $e$  – конечное значение управляющей переменной. На каждом шаге цикла переменная  $v$  последовательно принимает значения, соответствующие компонентам вектора  $M$ . Цикл выполняется до тех пор, пока  $v \leq d$ .

Возможен также вариант, когда  $M$  – не вектор, а матрица. Тогда цикл будет работать иначе: на каждом шаге управляющая переменная будет вектором, последовательно совпадающим со столбцами матрицы  $M$ . В этом случае в цикле будет столько шагов, сколько имеется столбцов в указанной матрице.

Оператор цикла типа `while` выполняется до тех пор, пока выполняется условие, указанное в заголовке:

```
while условие
    операторы
end
```

Если есть необходимость в досрочном прерывании выполнения цикла, используется оператор `break`. Оператор `continue` досрочно возобновляет цикл. Вне циклов слова `continue` и `break` не применяются.

Подробную справку по любой функции или оператору можно получить, введя команду `help имя_функции/оператора`.

## 4.2. Функции пользователя

Создание функций позволяет организовать алгоритм модульной структуры, локализовать часто выполняющиеся операции с разными наборами входных данных, распределить области видимости переменных.

Файлы-функции в языке MATLAB обязательно начинаются с объявления `function`, после которого указывается имя переменной (или имена нескольких переменных) – выходного параметра, имя самой функции и список ее входных параметров.



Все переменные, используемые в теле файла-функции, являются локальными, т. е. действуют только в пределах тела функции. При этом переменные общего рабочего пространства внутри функции не видны.

Структура файла-функции с одним выходным параметром выглядит следующим образом:

```
function var=f_name(список_параметров) ← заголовок
% Основной комментарий
% Дополнительный комментарий
Тело функции, состоящее из любой совокупности операторов
var=выражение
```

*Замечание 1:* оператор `var=выражение` (и сама переменная `var`) используются в тех случаях, когда требуется, чтобы функция возвращала некоторый результат.

*Замечание 2:* если выходных параметров больше одного, то необходимо использовать конструкцию типа

```
function [var1, var2, ...] = f_name(список_параметров)
```

В теле функции должны быть проинициализированы все выходные параметры.

*Замечание 3:* имя функции `f_name`, указанное в определении функции, может не совпадать с именем файла, но вызов функции будет осуществляться по имени файла. Для лучшей читаемости программы рекомендуется назначать одинаковыми имена файлов и функций в описании (функция `f_name` должна соответствовать файлу `f_name.m`).

Как уже было сказано, в файлах-функциях используются локальные переменные. Но наряду с ними нередко возникает необходимость в использовании данных, находящихся в рабочем пространстве MATLAB, или передача данных из одной функции в другую не через выходные параметры. В этих случаях используется понятие глобальных переменных, объявляемых командой

```
global var1 var2 ...;
```

Чтобы несколько программных модулей могли совместно использовать глобальную переменную, ее идентификатор должен быть объявлен как `global` во всех этих модулях. Если в функции будут использоваться общие переменные, их необходимо также объявить глобальными в функции.

Внутри файла-функции также допускается описывать и вызывать другие функции, называемые локальными. Программный модуль будет иметь вид

```
function val = func(var1, var2)
    % осн. комментарий
    % доп. комментарий
    x1 = sub_func(var1)
    операторы
    val = выражение
end
```

```
val = sub_func(x1)
    val = выражение
end
```

Функция `sub_func` локальная для функции `func`. Другим функциям и программным модулям локальная функция будет недоступна. При этом у локальной функции область видимости своя, отличающаяся от области видимости «старшей» функции. При наличии локальных функций для обозначения тел функций необходимо писать ключевое слово `end`.

Кроме файлов-функций в MATLAB также различают файлы-сценарии. Они не имеют ключевого слова `function`, используют общую область переменных. В прежних версиях пакета MATLAB не допускалось размещение локальных функций внутри сценариев, но в последних версиях такая возможность реализована.

### 4.3. Индексация матриц

Пакет MATLAB специально предназначен для обработки матриц, или многомерных массивов. Операции над матрицами, реализуемые в других языках с помощью дополнительных библиотек, в MATLAB являются встроенными.

Как видно из постановки задачи курсового проекта, информацию о графах удобно хранить в виде матриц, например в виде матрицы смежности и матрицы весов. В то же время, рассмотренные алгоритмы обработки графов не содержат операций над матрицами. Зато многие действия в указанных алгоритмах связаны с получением и обработкой индексов матриц. Рассмотрим возможности MATLAB, связанные с индексацией матриц.

При обращении к элементам матриц можно использовать один индекс вместо двух. В памяти матрицы хранятся по столбцам, поэтому увеличение индекса соответствует направлению сверху вниз по первому столбцу, затем по

второму, и т. д. Взаимный переход между линейной и матричной (по строкам и столбцам) индексацией осуществляется функциями `sub2ind()` и `ind2sub()`.

В качестве индексов матрицы можно указывать не только целые числа, но также векторы или матрицы, состоящие из целых чисел. Такая индексация называется блочной, поскольку ее результат – матрица, блоком входящая в индексируемую:

```
A1 = 2:2:20;      % A1 = [2 4 6 8 10 12 14 16 18 20]
B1 = A1(1:2:9);   % B1 = [2 4 10 14 18]
C1 = B1([2 4]);   % C1 = [4 14]
```

При обращении к матрицам можно применять индексы типа `logical`, т. е. имеющие два значения: «истина» (`true` или `1`) и «ложь» (`false` или `0`). При этом элементы с индексами, соответствующими значению «ложь», будут удалены из матрицы:

```
A2 = 1:9;          % A2 = [1 2 3 4 5 6 7 8 9]
B2 = A2 > 6 | A2 < 3; % B2 = [1 1 0 0 0 0 1 1 1]
C2 = A2(B2);       % C2 = [1 2 7 8 9]
```

Функция `find()` позволяет определить индексы ненулевых элементов для аргумента числового типа, либо индексы истинных элементов, если аргумент относится к логическому типу. Данную функцию можно использовать, чтобы найти индекс элемента матрицы по его значению:

```
A3 = [1 2 3; 4 5 6; 7 8 9];
ind = find(A3 == 6);          % ind = 8 – линейный индекс элемента со
                              % значением 6 в матрице A3
[i, j] = ind2sub(size(A3), ind); % i = 2, j = 3 – матричный индекс элемента
```

#### 4.4. Функции ввода-вывода в MATLAB

При разработке программы необходимо предусмотреть способы получения исходных данных и вид представления результатов обработки этих данных. Типовыми источниками информации для программы служат средства ввода (клавиатура, мышь) или файлы. Результаты обработки данных могут выводиться на экран в виде текста и графики, а также в файлы различных форматов. Рассмотрим функции MATLAB для ввода и вывода данных.

Функция `input('prompt')` позволяет ввести с клавиатуры любые данные, в соответствии с синтаксическими правилами языка программирования MATLAB. Таким способом можно задавать числа, матрицы, выражения, даже вызывать функции, которые в текущем состоянии программы вычисляемы. Если при вызове функции `input` указать второй параметр `'s'`, данные, задава-

емые с клавиатуры, воспринимаются как последовательность символов – вектор-строка.

Вывод на экран осуществляет функция `disp(x)`, в качестве аргумента могут быть указаны любые однотипные данные – числа, матрицы, строки символов, а также структуры. Если необходимо вывести на экран комбинированные данные, например сочетание текста и числовых значений, нужно использовать функцию форматного вывода в строку `sprintf()`. Данная функция формирует строку, которую затем можно вывести стандартной функцией `disp()`.

Обратная функция `sscanf()` позволяет считать произвольные данные из строки символов.

Для обмена информацией с файлами (чтение и запись) существует много различных функций, их список можно получить, введя в командном окне команду `help iofun`. Рекомендуется использовать функции форматного чтения и записи `fscanf()` и `fprintf()`, аналогичные рассмотренным выше `sscanf()` и `sprintf()`. Для работы с функциями форматного чтения и записи требуется создать идентификатор файла в MATLAB с помощью функции `fopen()`. После окончания операций чтения и записи его необходимо освободить с помощью функции `fclose()`:

```
fileid1 = fopen('my_file1.txt', 'r'); % открытие файла для чтения
fileid2 = fopen('my_file2.txt', 'w'); % открытие файла для записи
A = fscanf(fileid1, '%d', inf); % считывание целых чисел из файла
fprintf(fileid2, '%d', A); % запись целых чисел из матрицы A в файл
fclose(fileid1); % закрытие файла my_file1.txt
fclose(fileid2); % закрытие файла my_file2.txt
```

Примечание – функция `fprintf` без указания идентификатора файла осуществляет вывод форматированной строки в командное окно, т. е. инструкция `fprintf('%d', A)` идентична вызову `disp(sprintf('%d', A))`.

Средства графического вывода в MATLAB объединены в целую графическую подсистему. Простейшей функцией для вывода двумерного графика является функция `plot()`. Оформление графического окна (вывод сетки, подписи осей и т. д.) осуществляется функциями `grid`, `title`, `xlabel`, `ylabel`, `subplot`, `hold`, `legend` и др. Пример программы вывода на экран нескольких график представлен далее, соответствующий график показан на рис. 3.1.

```
x1 = 0:0.01:10;
x2 = 0:10;
y1 = sin(x1); y2 = sin(x2);
plot (x1, y1, 'r--', x2, y2, 'k*'), grid
```

```
xlabel('x'), ylabel('sin(x)')
title('Sine plotting')
```

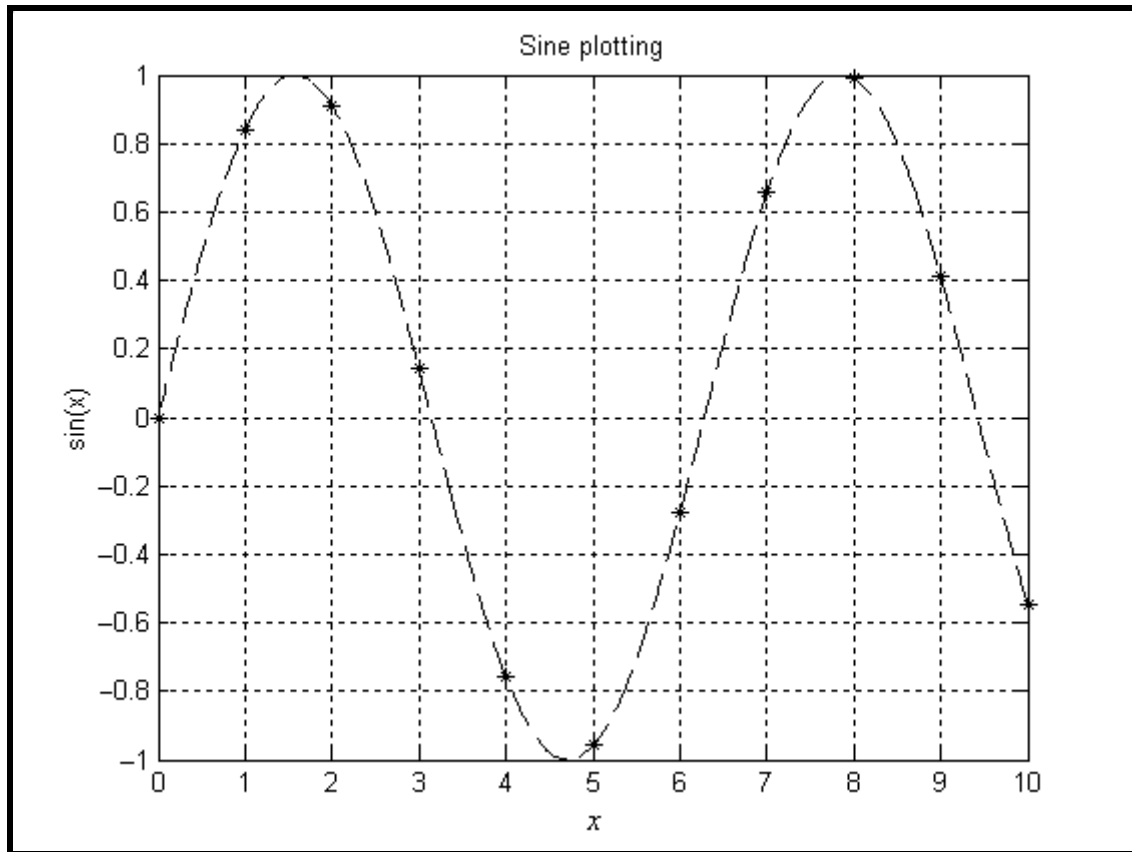


Рис. 3.1

Развернутое управление выводом графиков (толщина линий, размер и вид шрифта и др.) можно получить с помощью дескрипторов – графических указателей. Дескрипторы можно создать функциями `plot`, `figure`, `axes`, `text` либо получить по имеющимся графическим объектам – `gcf`, `gca`. Просмотр и изменение параметров графиков осуществляется функциями `get` и `set`.

Вывод линии в трехмерной плоскости осуществляется функцией `plot3()`. Для вывода поверхностей можно применять функции `mesh()`, `surf()`. Пример программы для вывода сетчатой поверхности приведен далее, соответствующий график показан на рис. 3.2.

```
x = -2:0.2:2;
y = -4:0.4:4;
[X,Y] = meshgrid(x, y); % создание сетки для построения поверхности
Z = sin(X-1).^2 + sin(Y+1).^2 + 4;
mesh(X,Y,Z), grid
colormap([0.5 0.5 0.5]) % установка цвета поверхности
title('meshed surface')
xlabel('x'), ylabel('y'), zlabel('height')
```

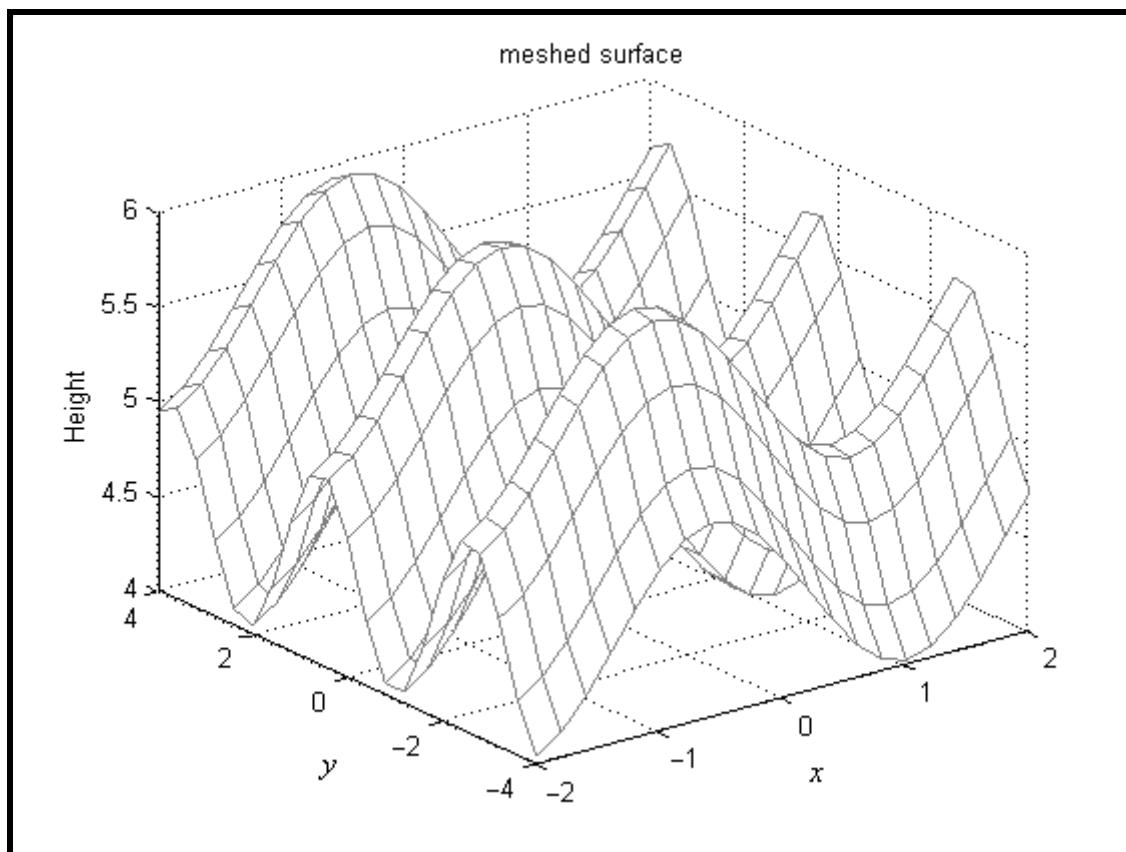


Рис. 3.2

Для ввода графической информации с помощью мыши предусмотрена функция `ginput()`. Другие функции ввода с помощью мыши позволяют создать более удобный для пользователя интерфейс:

- `menu()` – создание меню и передача управления пользователю для выбора;
- `msgbox()` – создание окна для вывода сообщения;
- `inputdlg()` – создание окна диалога – формы для ввода данных.

## 5. ТРЕБОВАНИЯ К ВХОДНЫМ И ВЫХОДНЫМ ДАННЫМ ПРОГРАММЫ

Программные модули при выполнении взаимодействуют с оператором (пользователем) и (или) другими программными модулями. При формировании данных для пользователя обычно учитываются требования понятности и удобства представления информации.

Для взаимодействия с другими программными модулями необходимо детальное и точное согласование формата входных и выходных данных. Обычно процедура согласования программных модулей оформляется протоколами информационного взаимодействия, в которых разработчики указы-

вают полный состав, порядок следования, форматы, вид и расширение файлов и прочую информацию. Существуют и стандартные протоколы, одним из примеров которых (для передачи навигационной информации) является протокол NMEA.

National Marine Electronics Association (NMEA) - Национальная ассоциация морской электроники, разработала специальный протокол для поддержания совместимости морского навигационного оборудования различных производителей. Этот NMEA-протокол описывает данные, полученные с GPS приемников, но и измерения сонаров, радаров, электронных компасов, барометров и других навигационных устройств, используемых в морской технике. Все NMEA-сообщения состоят из последовательного набора данных, разделенных запятыми. Каждое отдельное сообщение не зависит от других и является полностью «завершенным». NMEA-сообщение включает заголовок, набор данных, представленных ASCII-символами, и поле для проверки достоверности переданной информации.

Заголовок стандартных NMEA-сообщений состоит из 5 символов, из которых два первых определяют тип сообщения, а оставшиеся три – его название. Каждое NMEA-сообщение начинается с символа «\$» и обычно ограничено 80 символами.

Уточним для поставленной задачи формат и вид входных и выходных данных:

- координаты промежуточных точек (пунктов подзарядки) должны вводиться из файла, формат определяет разработчик (рекомендуется задавать данные в текстовом файле в виде таблицы координат; можно задавать также столбец с номером точки);
- координаты начальной и конечной точки должны задаваться с клавиатуры;
- прочие исходные данные (скорость объекта, время работы батареи и пр.) можно считать условно постоянными и задавать внутри программы;
- выходные данные должны выводиться на экран и в файл в составе:
  - матрицы координат, включающая начальную, промежуточные и конечную точки,
  - матрицы смежности неориентированного графа,
  - вспомогательных массивов для вычисления минимального пути,
  - последовательности вершин минимального пути,
  - последовательности NMEA-сообщений о маршруте объекта;

– график с отображением графа и минимального пути.

Сообщение NMEA имеет следующий формат:

«\$UTHDG,XX,Y.Y,DD.DD,S1,Z.Z,S2», без пробелов, где XX – часы (целое число), Y.Y – минуты (вещественное число), DD – расстояние до следующей точки; Z.Z – угол направления в градусах.

S1 – сигнал наличия поворота (символ), возможные значения: ‘Т’ – есть поворот, ‘N’ – нет поворота.

S2 – сигнал окончания движения (символ), возможные значения: ‘Е’ – конец движения, ‘N’ – движение не окончено.

Угол направления (азимута) отсчитывается от оси *OX* в декартовой системе координат, диапазон – от 0 до 360°.

## 6. ТИПОВОЙ ВАРИАНТ ЗАДАНИЯ НА КУРСОВОЙ ПРОЕКТ

Разработать программу на языке MATLAB, выполняющую следующие действия:

- формирование массива данных в виде неориентированного графа;
- поиск оптимальной траектории;
- расчет маршрута и отображение результатов в графике и специальном формате NMEA-сообщений.

**Исходные данные (по вариантам в соответствии с таблицей).**

### Характеристики подвижного объекта

Скорость, км/ч	Время работы аккумулятора, ч	Время зарядки на з/п, мин

### Описание территории

Координаты начальной точки:

Координаты конечной точки:

Набор координат точек (заправочных пунктов):

8 – 10 узлов сетки с шагом 1 ( $-10 \leq X \leq 10$  и  $-10 \leq Y \leq 10$ )

Масштаб сетки: \_\_\_\_ км.

### Задание

1. Занести информацию о начальной, конечной точках и наборе заправочных пунктов в массив.



### Варианты заданий

№ варианта	Скорость, км/ч	Время работы аккумулятора, ч	Время зарядки, мин	Начальная точка	Конечная точка	Масштаб сетки, км
1	16	6.5	15	(-9; 7)	(7; -6)	25
2	16	6	27	(5; 5)	(-8; -9)	30
3	18	5	25	(6; -7)	(-7; 6)	20
4	18	6	30	(-7; -7)	(5; 6)	25
5	20	5	17	(10; -8)	(-8; 8)	30
6	20	4.5	15	(-5; -5)	(4; 6)	20
7	22	4.5	20	(10; 10)	(-6; -7)	30
8	22	4	17	(-9; 7)	(8; -5)	20
9	16	5.5	30	(4; 10)	(-8; -9)	20
10	16	4.5	20	(-5; 6)	(4; -10)	20
11	18	3.5	27	(-3; 9)	(10; 8)	15
12	18	4	15	(7; 8)	(-9; -6)	20
13	18	4.5	30	(-9; -8)	(10; 9)	20
14	19	5	25	(8; -5)	(-8; 5)	30
15	19	5.5	15	(-6; 6)	(5; -5)	15
16	19	6.5	30	(-7; 9)	(9; -6)	25
17	20	3.5	30	(-10; -10)	(10; 8)	40
18	20	4	17	(6; 10)	(-8; -5)	25
19	17	5	22	(-9; 9)	(7; -7)	20
20	17	5.5	15	(8; -8)	(-7; 6)	20
21	17	4.5	25	(7; 5)	(-8; -6)	30
22	17	6	20	(-7; 7)	(7; -7)	25
23	19	4	23	(4; 10)	(-4; -9)	20
24	21	3.5	15	(-4; -8)	(8; 4)	15
25	21	5	25	(-5; -8)	(9; 9)	25

2. Построить массив возможных перемещений между точками с учетом ограничения работы аккумулятора (квадратный массив, содержащий 1 и 0 – соответственно, перемещение возможно и невозможно) (матрица смежности)

3. Построить матрицу длин перемещений.

4. Определить вектор, состоящий из номеров точек, соответствующих минимальному пути от начальной до конечной точки.

5. Определить последовательность углов направлений и моментов времени для оптимальной траектории и сформировать NMEA-сообщения.

6. Построить график с указанием графа перемещений и оптимальной траектории.

7. Вывести результаты расчетов (пп. 1 – 5) на экран и в файл.

## Требования к программе

1. Ввод данных должен осуществляться из файла данных (набор точек) и с клавиатуры (координаты начальной и конечной точки, начальное время перемещения).
2. Уравнение поверхности задается дополнительно преподавателем.
3. Программа должна иметь возможность последовательного выполнения действий 1–6 с просмотром промежуточных результатов.
4. Процедуры вычисления длины, направления движения должны быть реализованы в виде функций пользователя.

## Содержание отчета по курсовой работе

1. Задание на курсовую работу.
2. Постановка задачи и математическая модель задачи, построенная в соответствии с заданием.
3. Блок-схема алгоритма решения задачи, оформленная по правилам ГОСТ.
4. Контрольный расчет и тестирование алгоритма поиска минимального пути (2–3 точки).
5. Текст отлаженной программы.
6. Результаты работы программы (распечатка файла данных и файла результатов, графика).
7. Выводы по проделанной работе.

## ПРИЛОЖЕНИЕ.

### Основы работы со средой программирования MATLAB 6.5

Основам работы в среде MATLAB посвящено много учебной и научной литературы [5 – 11]. Приведем краткое изложение базовых средств MATLAB версии 6.5.

По умолчанию после запуска пакета MATLAB 6.5 на экране появляется комбинированное окно, включающее наиболее важные панели: 1 – **Command Window** (Командное окно), 2 – **Command History** (История команд), 3 – окно для отображения вкладок **Workspace** (Рабочее пространство), 4 – **Current Directory** (Текущий каталог), 5 – Панель инструментов (рис. П.1).

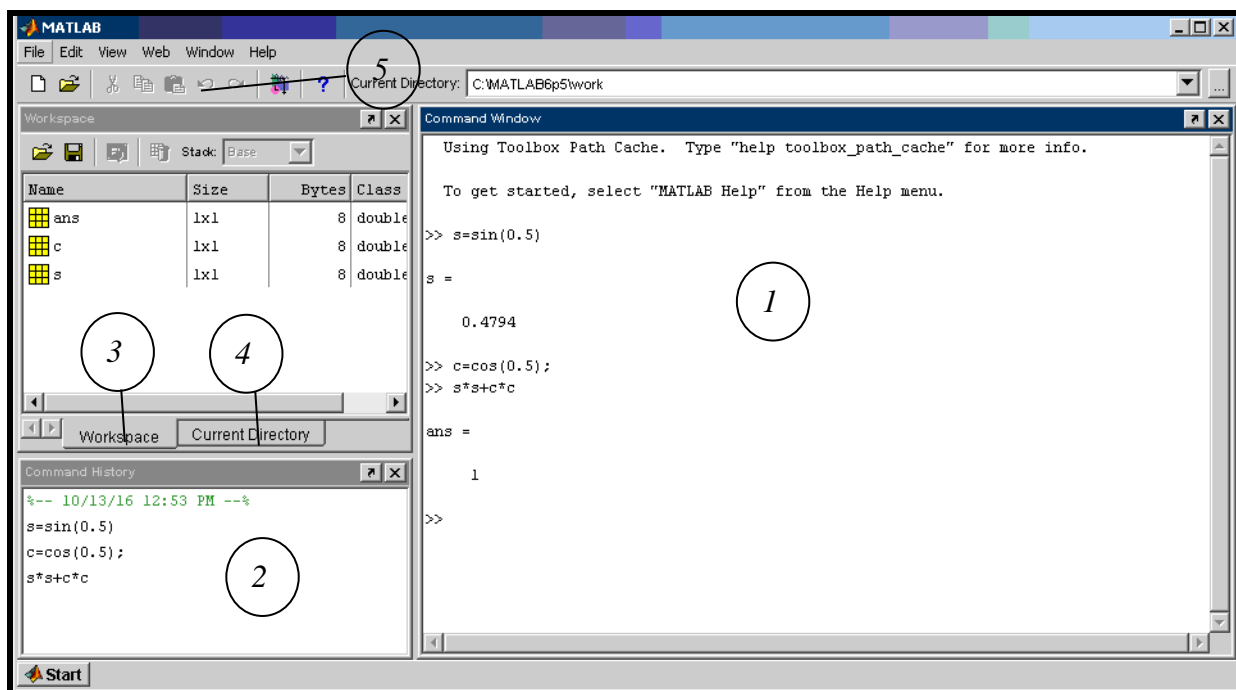



Рис. П.1

Для выбора вкладки нужно щелкнуть мышью по названию. Три окна, вписанные в окно приложения, передвигаются вместе с ним. Каждое окно можно отстыковать (кнопка **Undock**  в правом верхнем углу), и тогда оно может занимать автономную позицию на экране.

Основной рабочей панелью является **Командное окно**. В нем набираются команды пользователя, подлежащие немедленному исполнению, выводятся результаты исполняемых команд и программ. В командном окне можно обратиться за справкой по поводу термина или функции с помощью одной из команд – doc, help или lookfor.

Окно **истории команд** хранит все команды, набираемые пользователем, однако в отличие от содержимого «Командного окна» сюда не попадают сообщения системы и результаты вычислений. История сохраняется после закрытия главного окна приложения.

Вкладка **Workspace** отображает текущий набор переменных, существующих в ходе или результате выполнения команд и программ. Здесь можно увидеть их имена (колонок Name (Имя)), размерность (колонок Size (Размер)), количество отведенных байт (колонок Byte (Количество байтов)) и тип представляемых данных (колонок Class (Тип данных)). Точно такую же информацию можно увидеть в командном окне после исполнения команды whos. Переменные стираются при закрытии главного окна приложения, также их можно стереть вручную из командного окна командой clear.

Вкладка «Текущий каталог» отображает путь на диске к рабочему каталогу (он также продублирован на панели инструментов), а также список находящихся в нем файлов. Любые файлы, созданные пользователем или выполняемой программой, по умолчанию будут помещены в рабочий каталог. Настроить рабочий каталог можно иконками навигации, а также иконкой на панели инструментов.

На панель инструментов вынесены наиболее употребительные команды главного меню (рис. П.2). Кнопка **Simulink** обеспечивает вызов самого популярного расширения пакета MATLAB – системы имитационного моделирования. Кнопка (**Browse for Folder** (Перейти в каталог)) на панели инструментов используется для смены рабочего каталога.

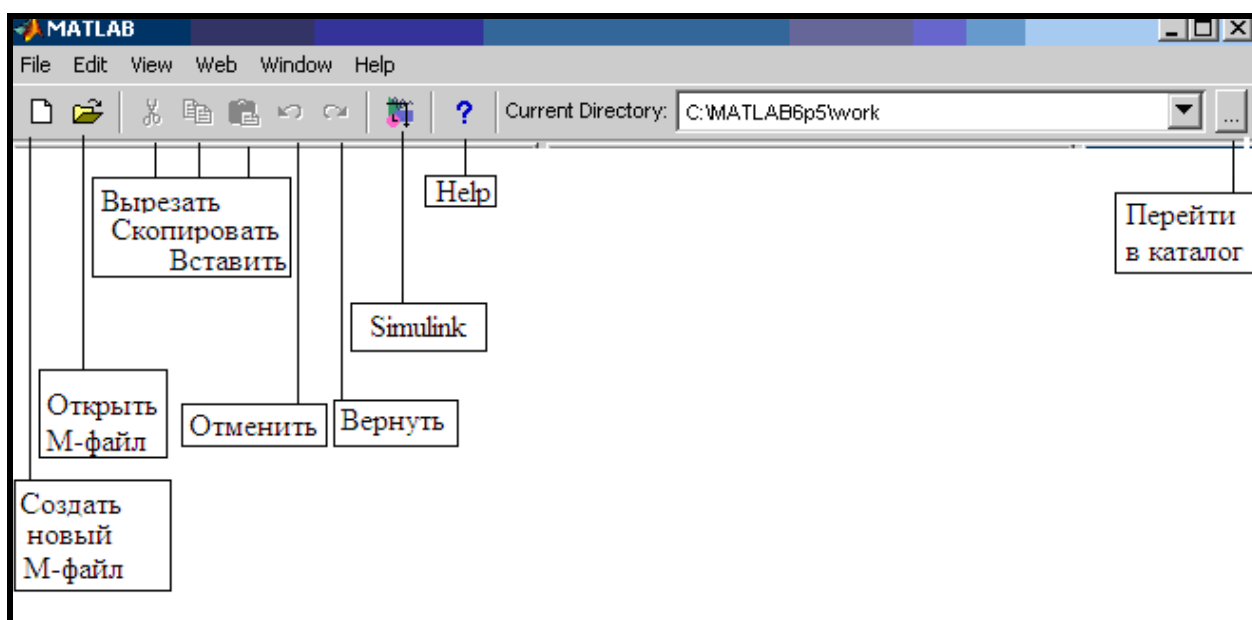


Рис. П.2

Настройка внешнего вида окон приложения MATLAB доступна по команде меню View (Вид) (рис. П.3). Команда Desktop Layout (План рабочего стола) определяет количество и расположение одновременно видимых панелей среды. Можно выбрать конфигурацию среды по умолчанию Default (Установки по умолчанию) или сохранить на экране только Командное окно (Command Window Only).

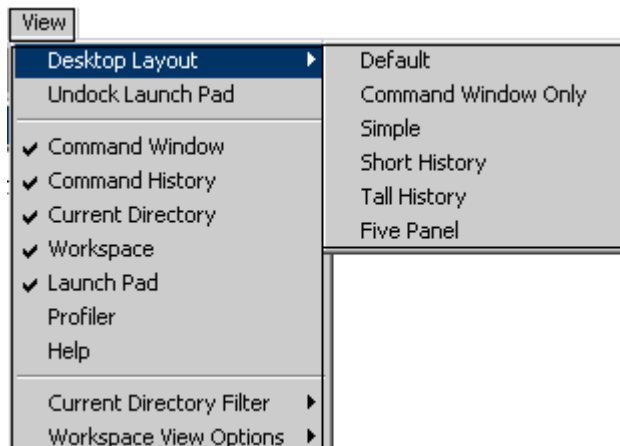


Рис. П.3

Редактор m-файлов вызывается по команде **File (Файл)→New (Новый)→M-file (М-файл)** или при щелчке по имени m-файла. Главное меню редактора несколько отличается от главного меню MATLAB, линейка инструментов здесь гораздо насыщенней (рис. П.4).

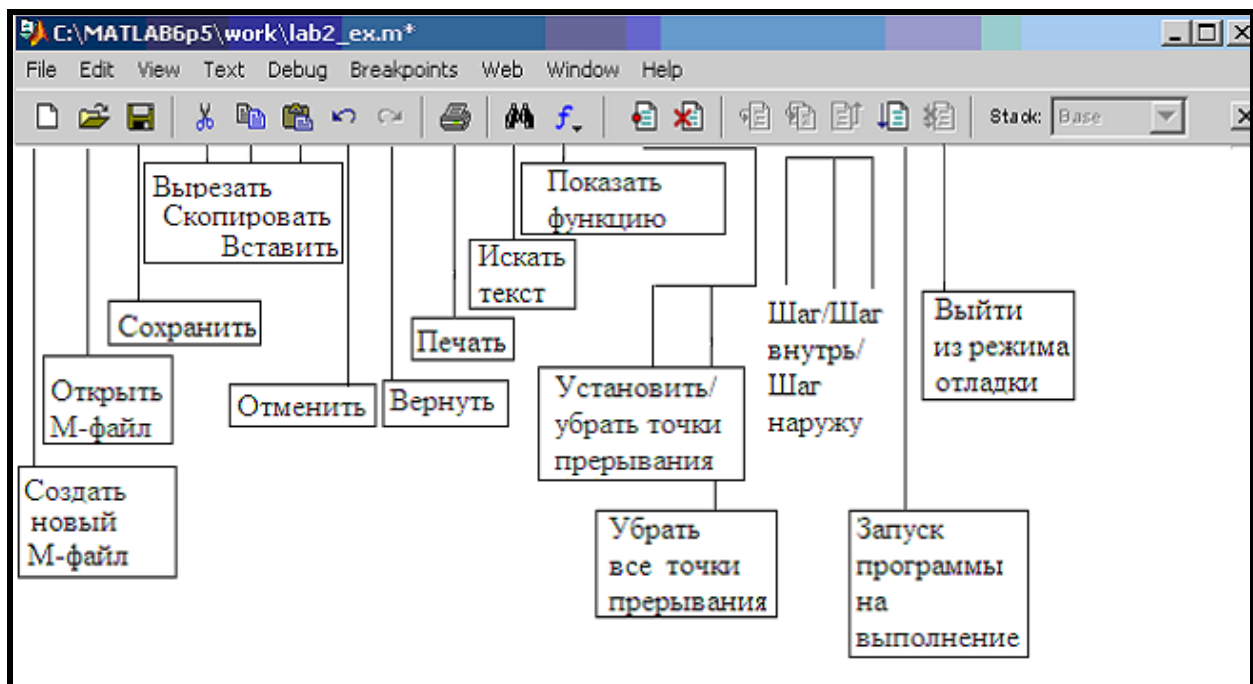


Рис. П.4

Среди разделов главного меню появились новые – **Text (Текст)**, **Debug (Отладка)**, и **Breakpoints (Точки прерывания)**.

Раздел **Text (Текст)** главным образом ориентирован на управление текстом программы на поле редактора, содержит вспомогательные команды. Иконка «Показать функцию» служит для перемещения между локальными функциями в тексте программы.

Раздел **Breakpoints (Точки прерывания)** предоставляет пользователю возможность установить точки прерывания на любой исполняемой строке программы для ее отладки.

Раздел **Debug (Отладка)** включает команды, управляющие режимом отладки m-файлов: пошаговое выполнение – **Step (Шаг)**, **Step In (Шаг внутрь)** – устанавливает точки останова на первых исполняемых строках каждой подфункции. **Step Out (Шаг наружу)** – исключает подфункции из отладочного режима, предоставляя им возможность выполняться автоматически. В ходе отладки программы пользователь может просмотреть доступные переменные (в командном окне при останове в режиме отладки командная строка имеет вид «**K>>**»). При отладке функций переключение в панели **Stack (Стек)** в правой части панели инструментов позволяет просматривать переменные, видимые в функции и в точках вызова функции.

## СПИСОК ЛИТЕРАТУРЫ

1. Андриевский Б. Р., Фрадков А. Л. Элементы математического моделирования в программных средах MATLAB 5 и Scilab. СПб.: Наука, 2001
2. Яблонский С. В. Введение в дискретную математику. 4-е изд. М.: Высш. шк., 2006.
3. Ветчинкин А. С., Стариченков А. Л. Программирование и основы алгоритмизации: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2005.
4. Основы составления алгоритмов и написания программ на языке СИ: метод. указания к курсовой работе по дисциплине «Программирование и основы алгоритмизации» / А. С. Ветчинкин, О. Ю. Лукомская, А. Л. Стариченков, А. Г. Шпекторов. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2008.
5. Дьяконов В. MATLAB 6: учеб. курс. СПб.: Питер, 2001.
6. Кетков Ю. Л., Кетков А. Ю., Шульц М. М. MATLAB 6.x: программирование численных методов. СПб.: БХВ-Петербург, 2004. 672 с.
7. Кондрашев В., Королев С. MATLAB как система программирования научно-технических расчетов. М.: Мир, 2002.
8. Кривилев А. Основы компьютерной математики с использованием системы MATLAB. М.: Лекс-Книга, 2005.
9. Мартынов Н. Введение в MATLAB 6. М.: Кудиц-образ, 2002.
10. Мирошников А. Н., Румянцев С. Н. Моделирование систем управления технических средств транспорта: учеб. издание. СПб.: Элмор, 1999.
11. Потемкин В. Г. Введение в MATLAB. URL: <http://matlab.exponenta.ru/ml/book1/index.php> (дата обращения 17.10.2016).

## СОДЕРЖАНИЕ

1. Постановка задачи и формирование математической модели .....	3
2. Алгоритмы поиска оптимальной траектории.....	8
2.1. Алгоритм Форда–Беллмана .....	8
2.2. Алгоритм Дейкстры.....	9
2.3. Алгоритм Флойда .....	12
3. Разработка блок-схемы алгоритма .....	14
4. Основные конструкции языка MATLAB.....	15
4.1. Условия и циклы.....	15
4.2. Функции пользователя .....	16
4.3. Индексация матриц.....	18
4.4. Функции ввода-вывода в MATLAB.....	19
5. Требования к входным и выходным данным программы.....	22
6. Типовой вариант задания на курсовой проект .....	24
Приложение. Основы работы со средой программирования MATLAB 6.5 ...	25
Список литературы .....	31

Ветчинкин Александр Сергеевич,  
Лукомская Ольга Юрьевна,  
Шпекторов Андрей Григорьевич

### **Составление алгоритма и написание программ обработки массива данных**

Учебно-методическое пособие

Редактор Н. В. Лукина

---

Подписано в печать . Формат 60×84 1/16.  
Бумага офсетная. Печать цифровая. Печ. л. 2,0.  
Гарнитура «Times New Roman». Тираж 50 экз. Заказ

---

Издательство СПбГЭТУ «ЛЭТИ»  
197376, С.-Петербург, ул. Проф. Попова, 5