Министерство образования и науки Российской Федерации
Владимирский государственный университет
Кафедра вычислительной техники

Курсовая работа

«Методы оптимизации»

Выполнил:
студент группы ИВТ-201
Морев Н. В.

Проверил:
Козлов Д. В.

Владимир 2004

Содержание

1. Задание

Написать программы, реализующие:

- решение СЛАУ методом Жордана-Гаусса;
- решение задачи динамического программирования методом ветвей и границ, а также решение задачи про аэроплан и задачи о брахистохроне;
- решение задачи минимизации функции методами оптимального поиска и градиентными методами 1-го и 2-го порядка;
- решение задачи линейного программирования симплексным методом.

2. Теоретическая часть

2.1. Метод Жордана-Гаусса

Метод Жордана-Гаусса является одним из методов линейного программирования — численных методов решения оптимизационных задач, сводящихся к формальным моделям линейного программирования.

Метод Жордана-Гаусса почти ничем не отличается от классического метода Гаусса, только матрица системы приводится не к треугольному, а к диагональному единичному виду. После этого последний столбец матрицы A даст решение задачи. По сравнению с классическим методом Гаусса здесь отсутствует обратный ход, но общее число операций больше (хотя порядок по n одинаковый — n^3).

2.2. Метод ветвей и границ

Для рассмотрения теории метода ветвей и границ мы в качестве примера используем так называемую задачу коммивояжёра:

> Имеется N пунктов, расстояния между которыми известны. Необходимо, начав движение из п. 1 последовательно обойти все остальные по самому короткому пути и вернуться в п. 1. Условие α заключается в следующем: в каждый пункт надо войти только один раз и один раз из него выйти. Задачу можно представить в виде графа из N вершин, на ребрах которого указаны расстояния.

Условие α делает невозможным выбор на некотором шаге вычислений оптимального из путей, приводящих в некоторую точку. Решение задачи методом прямого перебора всех возможных вариантов возможно лишь для малых N, Необходим метод, который позволяет уменьшать число рассматриваемых вариантов. Таким методом является метод ветвей и границ.

Основное содержание метода заключается в том, что строятся нижние оценки функции цели (ФЦ), позволяющие отбраковать множество вариантов, для которых значение ФЦ заведомо хуже. По мере рассмотрения новых вариантов граница перемещается вниз (в задаче минимизации) до тех пор, пока число различных вариантов не уменьшается настолько, что выбор лучшего из них будет сделан непосредственным сравнением.

Решение задачи Запишем исходные данные в виде таблицы $C_0 = (C_{ij})$. В ней i и j — номера пунктов, движение на каждом шаге совершается из i в j, длина этого шага пути C_{ij}, крестиками отмечены запрещенные переходы. Пусть $C_i = \min C_{ij}$. Найдем C_i для каждого i, после чего совершим приведение матрицы C_0 к C_0^1 по строкам, где C_{ij}^1 вычисляются по формулам $C_{ij}^1 = C_{ij} - C_i$. Обозначим $d_0^1 = \sum C_i$ и $C_j^1 = \min C_{ij}$. Найдем C_j для всех j и выполним приведение C_0^1 по столбцам. В приведенном примере все $C_j = 0$, поэтому $C_0^2 = C_0^1$, константа приведения по столбцам $d_0^2 = 0$, $d_0 = d_0^1 + d_0^2 = 23$.

C_0

	1	2	3	4	5
1	x	5	7	6	12
2	5	x	5	8	11
3	7	5	x	6	7
4	6	8	6	x	4
5	12	11	7	4	x

$C_0^1 = C_0^2, d_0 = 5 + 5 + 5 + 4 + 4 = 23$

	1	2	3	4	5
1	x	0	2	1	7
2	0	x	0	3	6
3	2	0	x	1	2
4	2	4	2	x	0
5	8	7	3	0	x

Можно утверждать, что задачам с матрицами C_0 и C_0^1 соответствует один и тот же оптимальный маршрут. Длина любого пути $L_S(C_0)$ и $L_S(C_0^2)$ связаны формулой $L_S(C_0) = L_S(C_0^2) + d_0$, что следует из условия α. Тогда любой путь по матрице представляет движение по клеткам таблицы, причем в каждой строке и каждом столбце можно побывать только один раз.

Величину d_0 можно использовать в качестве нижней границы при оценке всех возможных путей.

Рассмотрим путь, который включает переход из i в j. Для него $L_S \geq \gamma_{ij} = d_0 + C_{ij}$. Такой переход всегда есть, так как в приведенной матрице хотя бы один элемент в строке — нулевой. Выбирая один из

них, мы выбираем оптимизацию одного шага пути (самый короткий первый шаг). При этом, конечно, мы не знаем, как это отразится на длине последующего пути. Обозначим (ij) множество путей, не содержащих непосредственный переход из i в j. Поскольку из i надо куда-то выйти, а в j откуда-то прийти, то этому множеству соответствует оценка

$$L_S \geq \gamma_{(ij)} = d_0 + \theta_{ij}, \text{где } \theta_{ij} = \min_k C_{ik} + \min_m C_{mj}$$

Тогда нижней оценкой для множества путей будет $\gamma_2 = d_0 + \theta_{ij}$. Мы заинтересованы в выборе такого перехода ij, для которого эта оценка является самой высокой. Этот выбор можно сделать, отыскав среди нулевых элементов i строки такой, которому соответствует $\theta^2 = \max \theta_{ij}$. Выбором пары (ij) все множество возможных путей разбивается на два подмножества. Одно из них содержит переход (ij), другое — $(\overline{ij})$

После обхода всех вершин и возвращения в первоначальную, принимаем получившуюся длину пути за рекордную и продолжаем движение по дереву вариантов, отсекая все пути, длина которых заведомо больше рекордной. Если был найден вариант обхода всех вершин графа с длиной пути меньше рекордной, то эту длину принимаем за рекордную и прожолжаем перебор вариантов.

Из изложенного следует сделать вывод, что метод ветвей и границ — весьма эффективен для решения задач с аддитивной целевой функцией.

2.3. Численные методы отыскания безусловного экстремума

Рассмотрим методы поиска

$$\min f(X) = f(X^o)$$

путем построения последовательности точек $X_k (k = 0, 1, 2, \ldots)$ такой, что $f(X_0) > f(X_1) > ... > f(X_n)$. Последовательности $\{X_k\}$ называются релаксационными, а методы их построения — методами спуска. Точки X_k связаны формулой

$$X_{k+1} = X_k - h_k P_k$$

где P_k — вектор направления, $h_k > 0$ — величина шага. Последовательность $\{X_k\}$ должна сходиться к X^o, а $f(X_k)$ к $f(X^o)$.

Алгоритмы минимизации, в которых используются только значения самой функции, относят к методам нулевого порядка; если, кроме того, используются первые производные — это методы первого порядка и т. д.

Рассмотрим класс унимодальных функций одного аргумента $y = f(x)$:

1) $f(x)$ задана на отрезке $[a, b]$.

2) Пусть $x_1, x_2 \in [a, b]$, причем $x_1 < x_2$; пусть x^o — точка минимума, тогда из $x_1 > x^o$ следует $f(x_2) > f(x_1)$, а из $x_2 < x^o$ следует $f(x_1) > f(x_2)$.

Возьмем $x_1 = a < x_2 < x_3 < x_4 = b$, тогда в зависимости от того, в каком x_i значение $f(x_i)$ минимально, минимум x^o может находиться в следующих интервалах:

$$\begin{cases} i = 1, & [x_1, x_2] \\ i = 2, & [x_1, x_3] \\ i = 3, & [x_2, x_4] \\ i = 4, & [x_3, x_4] \end{cases}$$

2.3.1. Пассивный поиск

Отрезок $[a, b]$ делится на N точек, в которых находят значения функции и выбирают из них минимальное. Точка минимума функции будет находиться внутри интервала, соответствующего найденной точке.

2.3.2. Метод дихотомии

Значения функции в точках находятся последовательно по шагам. На каждом шаге учитываются результаты предыдущего шага. Исходный отрезок делится 2-мя точками на 3 отрезка. Эта процедура повторяется до тех пор, пока не будет достигнута требуемая точность.

В методе дихотомии отрезок делится пополам и берутся точки с двух сторон от половины отрезка на расстоянии $\pm\Delta/2$, так что длина серединного отрезка равна Δ.

2.3.3. Метод Фибоначчи

Применяется, когда нужно определить минимум как можно точнее, но при этом можно выполнить только n вычислений функции.

Пусть L_i — длина отрезка на i-м шаге, n — число шагов, тогда

$$L_{n-k} = F_{k+1} L_n - F_{k-1} \Delta$$

где $\{F_i\} = 1, 1, 2, 3, 5, 8, \ldots$ — последовательность чисел Фибоначчи.

2.3.4. Метод «золотого сечения»

Метод «золотого сечения» позволяет исключать интервалы, вычисляя только одно значение функции на каждой итерации. В результате двух рассмотренных значений функции определяется интервал, который должен использоваться в дальнейшем. Этот интервал будет содержать одну из предыдущих точек и следующую точку, помещаемую симметрично ей. Точка делит интервал на части так, что отношение целого к большей части равно отношению большей части к меньшей, т. е. равно так называемому «золотому сечению».

Длина отрезка вычисляется по формуле

$$L_{i+1} = \frac{L_i}{\tau}$$

где $\tau = (1 + \sqrt{5})/2 \approx 1,618033989$.

Поиск с помощью метода «золотого сечения» является асимптотически наиболее эффективным способом реализации минимаксной стратегии поиска, так как требует наименьшего числа оценки значения функции для достижения заданной точности по сравнению с другими методами исключения интервалов.

2.3.5. Методы 1-го порядка

Очередная точка выбирается в направлении скорейшего убывания функции (антиградиента).

$$X_{k+1} = X_k - h_k f'(X_k)$$

Шаг h_k должен выбираться достаточно малым, чтобы выполнялось $f(x_{k+1}) < f(x_k)$. Спуск с постоянным шагом может привести к большому количеству итераций. Величина шага должна уменьшаться по мере приближения к x^o. Метод градиентного спуска с уменьшающимся шагом называют методом с дроблением шага. Шаг h_k выбирают так, чтобы выполнялось неравенство

$$f(x_{k+1}) - f(x_k) \leq -\varepsilon h_k |f'(x_k)|^2$$

2.3.6. Методы 2-го порядка

Градиентные методы очень просты в реализации и поэтому часто используются на практике. Но при сложной структуре $f(x)$ они могут плохо

сходиться. Поэтому используют вторые производные $f(x)$, как в численных методах второго порядка.

В выборе следующей точки участвует 2-я производная функции:

$$x_{k+1} = x_k - h_k \frac{f'(x_k)}{f''(x_k)}$$

Недостатком данного метода можно считать то, что на каждом шаге необходимо вычислять вторую производной $f''(x)$, и она должна быть положительно определена, иначе процесс может расходиться. Метод весьма эффективен, но лучше всего его применять, когда к x^o подошли достаточно близко, причем в районе x^o функция сильно выпукла.

2.4. Симплексный метод

Симплексный метод применяется для решения задач линейного программирования. Рассмотрим случай трех уравнений и пяти переменных, но выкладки справедливы для любого количества переменных и уравнений.

Задача Найти такие неотрицательные значения переменных $x_1, x_2, \ldots, x_5$, которые удовлетворяют системе уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}x_5 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25}x_5 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 + a_{35}x_5 = b_3 \end{cases}$$

и дают наименьшее значение функции

$$z = c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5$$

Решение Если ограничения на выбор переменных даны не в виде системы линейных уравнений, а в виде системы линейных неравенств, то их можно привести к системе линейных уравнений.

Систему линейных уравнений запишем в виде векторного уравнения

$$x_1\vec{P_1} + x_2\vec{P_2} + x_3\vec{P_3} + x_4\vec{P_4} + x_5\vec{P_5} = \vec{b}$$

Если решать такое уравнение методом замены векторов, выбор вектора-столбца, которым заменяли (заменяемый вектор) и вектор, который заменяли (заменивающий вектор) был произвольный: необходимо было, чтобы элемент таблицы на пересечении соответствующего ряда и столбца был не равен нулю и по возможности слишком мал по модулю.

При решении данной задачи на шаги замещения наложим два ограничения: с помощью одного правила выбираем заменивающий вектор, а с помощью другого — заменяемый. Эти два правила образуют симплексный метод.

	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$	$\vec{b}$	$\vec{C}$
$\vec{P}_1$	1	0	0	t_{14}	t_{15}	x'_1	C_1
$\vec{P}_2$	0	1	0	t_{24}	t_{25}	x'_2	C_2
$\vec{P}_3$	0	0	1	t_{34}	t_{35}	x'_3	C_3

Предположим, что в процессе решения задачи получили таблицу T, где $\vec{P}_1, \vec{P}_2, \vec{P}_3$ — линейно независимы, а числа x_1, x_2, x_3 — неотрицательные. Множество векторов $D = \{\vec{P}_1, \vec{P}_2, \vec{P}_3\}$ образуют базис для множества векторов $M = \{\vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{P}_4, \vec{P}_5, \vec{b}\}$.

Предположим, что заменивающим мы выбрали вектор $\vec{P}_5$, т. е. $\vec{P}_5$ вводим в состав нового базиса для множества M (как выбрать заменивающий вектор рассмотрим ниже), и что среди компонентов разложения вектора $\vec{P}_5$

$$\vec{P}_5 = t_{15}\vec{P}_1 + t_{25}\vec{P}_2 + t_{35}\vec{P}_3$$

есть хотя бы одно положительное число. Множество векторов $S = \{\vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{P}_5\}$ линейно зависимо. Пусть $\vec{x}'' = (x''_1, x''_2, x''_3, 0, x''_5$ — неотрицательное решение системы

$$x_1\vec{P}_1 + x_2\vec{P}_2 + x_3\vec{P}_3 + x_5\vec{P}_5 = \vec{b}$$

Кроме того, из таблицы T имеем:

$$x'_1\vec{P}_1 + x'_2\vec{P}_2 + x_3\vec{P}_2 = \vec{b}$$

Рассмотрим, как выбрать заменяемый вектор из множества $D = \{\vec{P}_1, \vec{P}_2, \vec{P}_3\}$, так чтобы новое решение было неотрицательным. Тогда:

$$t_{15}\vec{P}_1x''_5 + t_{25}\vec{P}_2x''_5 + t_{35}\vec{P}_3x''_5 = \vec{P}_5x''_5$$

$$(x'_1 - t_{15}x''_5)\vec{P}_1 + (x'_2 - t_{25}x''_5)\vec{P}_2 + (x'_3 - t_{35}x''_5)\vec{P}_3 + x''_5\vec{P}_5 = \vec{b}$$

Получим новое решение уравнения, из которого образуется базисное неотрицательное решение. Чтобы получить базисное решение, необходимо извлечь из множества D один из векторов $\vec{P}_1, \vec{P}_2, \vec{P}_3$ (причем так, чтобы найденное решение было неотрицательным).

Если какое-нибудь из чисел t_{15}, t_{25}, t_{35} — отрицательное, то соответствующий коэффициент в разложении будет неотрицательным. Следовательно, отрицательные коэффициенты в уравнении, т. е. отрицательные корни из нового решения, могут появиться там, где соответствующие t_{ij} будут положительные. Пусть, например, $t_{15} > 0$ и $t_{35} > 0$. Для того, чтобы числа $x'_1 - t_{15}x''_5$ и $x'_3 - t_{35}x''_5$ были неотрицательными, необходимо значение x_5 взять таким, чтобы оно было равно меньшему из чисел x'_1/t_{15} и x'_3/t_{35}.

Если, например, окажется, что наименьшее из этих чисел будет x'_1/t_{15}, то положив в уравнение $x''_5 = x'_1/t_{15}$, найдем, что коэффициент при $\vec{P_1}$ равняется нулю, следовательно вектор $\vec{P_1}$ есть заменяемый, он заменяется на вектор $\vec{P_5}$.

Новое базисное неотрицательное решение будет таким:

$$x'_2 - t_{25}x'_1/t_{15}, x'_3 - t_{35}x'_1/t_{15}, x'_1/t_{15}$$

2.5. Динамическое программирование

Метод динамического программирования иллюстрируется задачей об аэроплане. Она представляет собой задачу нахождения минимальной по затратам траектории самолета, где затраты определяются, как функция от скорости и высоты.

2.6. Задача о брахистохроне

Эта задача схожа с задачей об аэроплане: требуется найти оптимальную форму поверхности, по которой скатывается шарик, такую, что шарик скатывается по ней за минимальное время.

Данная задача также решается методами динамического программирования и главной трудностью в ней является преобразование чисто физической задачи в формулировки задачи динамического программирования.

3. Практическая часть

В данной части работы представлены исходные тексты процедур, реализующих соответствующие методы, а также результаты выполнения программы.

3.1. Метод Жордана-Гаусса

Листинг 1. Исходный текст

```
(*Метод ЖорданаГаусса -
 *
 *
 *)

procedure JG_Exclude(var matrix: TMatrix; iter: integer);
var i,j: integer;
    t: double;
    BadMatrix: boolean;
begin
  BadMatrix := True;
  for j := iter to leny(matrix)-1 do
    if not (matrix[iter,j] = 0) then begin
      BadMatrix := False;
      break;
    end;
  if BadMatrix then raise Exception.Create('Матрицавырождена ');
  if not (iter = j) then SwapRows(matrix,j,iter);

  t := matrix[iter,iter];
  if not (matrix[iter,iter] = 1) then
    for i := 0 to lenx(matrix)-1 do
      matrix[i,iter] := matrix[i,iter]/t;

  for j := 0 to leny(matrix)-1 do
    if not (j = iter) then begin
      t := matrix[iter,j];
      for i := 0 to lenx(matrix)-1 do
        matrix[i,j] := matrix[i,j] - matrix[i,iter]*t;
    end;
end;

procedure JG_Solve(matrix: TMatrix; grid: TStringGrid; memo: TMemo);
var i: integer;
begin
  memo.Lines.Append('РешениеСЛАУметодомЖорданаГаусса--    -');
  WriteMatrix(matrix,memo);
  for i:= 0 to leny(matrix)-1 do begin
    memo.Lines.Append('Исключение '+IntToStr(i+1)+'йпеременной- ');
    JG_Exclude(matrix,i);
    WriteMatrix(matrix,memo);
    WriteMatrix(matrix,grid);
  end;
end;
```

Результаты решения уравнения (см. рис. 1):

```
8,221 8,487 8,704 9,616
```

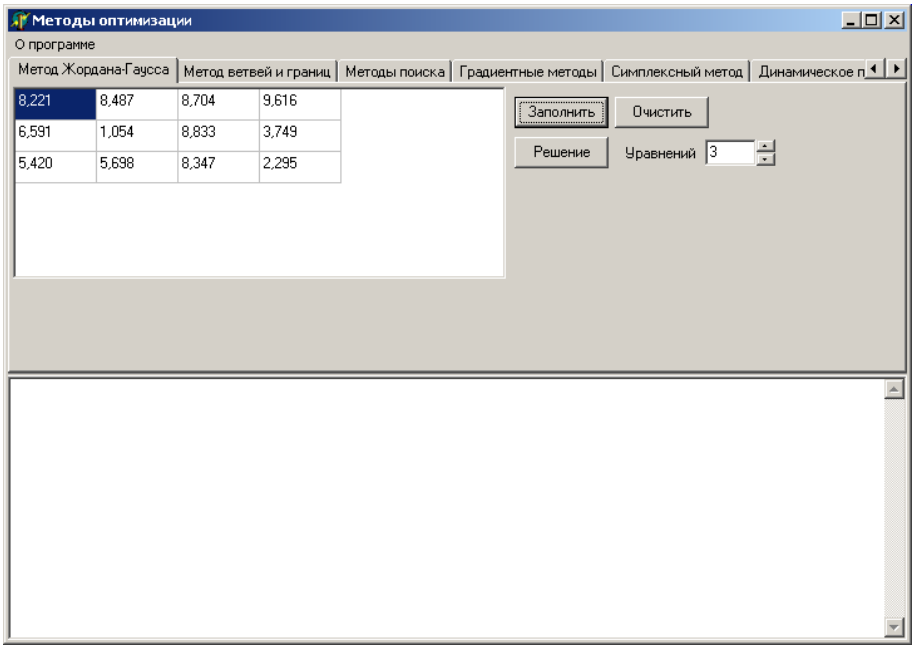

Рис. 1. Метод Жордана-Гаусса

```
6,591 1,054 8,833 3,749
5,420 5,698 8,347 2,295

1,000 1,032 1,059 1,170
0,000 -5,750 1,855 -3,960
0,000 0,103 2,609 -4,045

1,000 0,000 1,392 0,459
0,000 1,000 -0,323 0,689
0,000 0,000 2,642 -4,115

1,000 0,000 0,000 2,627
0,000 1,000 0,000 0,186
0,000 0,000 1,000 -1,558
```

3.2. Метод ветвей и границ

Листинг 2. Исходный текст

```
(*Метод ветвейиграниц
 *
 *
 *)
```

```
function BB_Reduce(const m: TMatrix; x, y: integer): TMatrix;
var i,j: integer;
    n: TMatrix;
begin
  SetLength(n,lenx(m),leny(m));
  for i:=0 to lenx(m)-1 do
  for j:=0 to leny(m)-1 do
    if (i = x) or (j = y) then n[i,j] := Infinity
    else n[i,j] := m[i,j];
  BB_Reduce := n;
end;

function BB_LowerBound(var m: TMatrix): double;
var i,j: integer;
    min,len: double;
begin
  len := 0;
  for j:= 0 to leny(m)-1 do begin
    min := m[0,j];
    for i:= 1 to lenx(m)-1 do
      if m[i,j] < min then min := m[i,j];

    if not IsInfinite(min) then begin
      len := len + min;
      for i:= 0 to lenx(m)-1 do
        m[i,j] := m[i,j] - min;
    end;
  end;

  for i:= 0 to lenx(m)-1 do begin
    min := m[i,0];
    for j:= 1 to leny(m)-1 do
      if m[i,j] < min then min := m[i,j];

    if not IsInfinite(min) then begin
      len := len + min;
      for j:= 0 to leny(m)-1 do
        m[i,j] := m[i,j] - min;
    end;
  end;

  BB_LowerBound := len;
end;

procedure BB_Solve(const m: TMatrix; start: integer;
                   grid: TStringGrid; memo: TMemo);
var thefirst: integer;
    thebest: double;
    temps, bests: TIntStack;
    s: string;
procedure BB_Walk(m: TMatrix; start: integer; len: double); (*Вложенная  *)
var i: integer;
    atleast: double;
    isLeaf: boolean;
begin
  WriteMatrix(m,memo);
  WriteMatrix(m,grid);
  memo.Lines.Append('Вершина: ' + IntToStr(start)
               + 'путь; : ' + FloatToStr(len));
  if len >= thebest then begin
    memo.Lines.Append('отсечениеветви-  ');
    exit;
```

```
  end;

  Push(temps,start);

  isLeaf := true;
  atleast := BB_LowerBound(m);
  for i:= 0 to Length(m)-1 do
    if i = thefirst then continue
    else if not IsInfinite(m[i,start]) then begin
      isLeaf := false;
      BB_Walk(BB_Reduce(m,i,start),i,len+atleast+m[i,start]);
    end;

  i := thefirst;
  if isLeaf and not IsInfinite(m[i,start]) then begin
    isLeaf := false;
    BB_Walk(BB_Reduce(m,i,start),i,len+atleast+m[i,start]);
  end;

  if isLeaf then begin
    if len < thebest then begin
      thebest := len;
      IntStackCopy(temps,bests);
    end;
    memo.Lines.Append('конецветви-  ');
  end;

  Pop(temps);
end;
begin
  IntStackInit(temps);
  IntStackInit(bests);
  thefirst := 0;
  thebest := Infinity;
  memo.Lines.Append('Решениезадачикоммивояжерametодомветвейиграниц--       :');
  BB_Walk(m,thefirst,0);
  memo.Lines.Append('Длинанаименьшегопути  : '+FloatToStr(thebest));
  while Length(bests) > 0 do s := s+IntToStr(Pop(bests))+' <- ';
  memo.Lines.Append('Наименьшийпуть : '+s);
end;
```

Результаты решения задачи Коммивояжёра (см. рис. 2):

```
-- Решение задачи коммивояжёра методом ветвей и границ:
INF 8,756 7,738 9,042
0,545 INF 1,071 9,242
0,623 1,874 INF 6,953
2,066 1,575 1,481 INF

Вершина: 0; путь: 0
INF INF INF INF
0,000 INF 0,526 7,393
0,000 INF INF 5,026
0,585 INF 0,000 INF

Вершина: 1; путь: 12,709
INF INF INF INF
INF INF INF INF
```

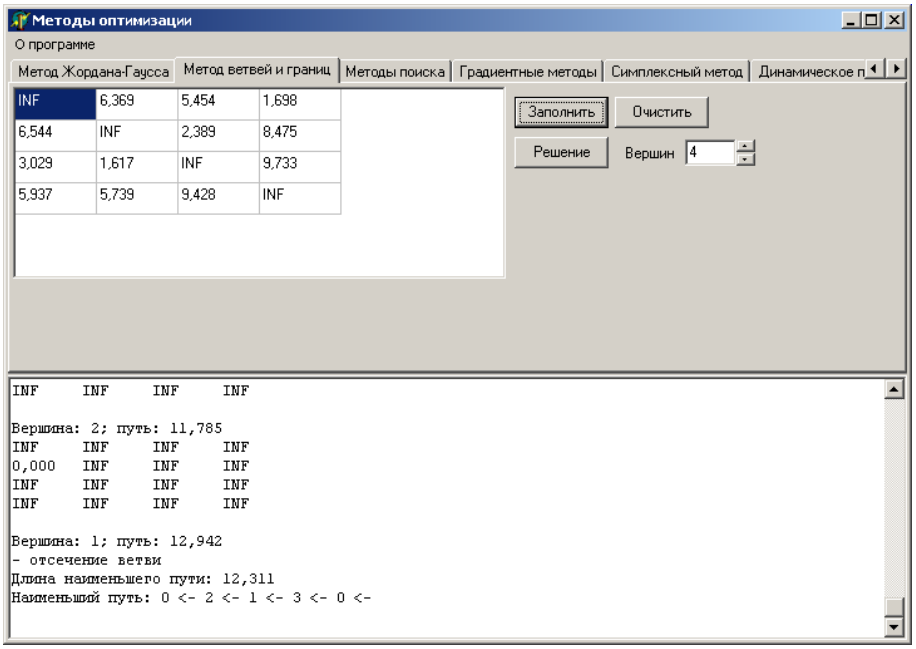

Рис. 2. Решение задачи коммивояжёра

```
0,000 INF INF 0,000
0,585 INF INF INF

Вершина: 2; путь: 18,261
INF INF INF INF
INF INF INF INF
INF INF INF INF
0,000 INF INF INF

Вершина: 3; путь: 18,846
INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Вершина: 0; путь: 18,846
- конец ветви
INF INF INF INF
INF INF INF INF
0,000 INF INF INF
0,585 INF 0,000 INF

Вершина: 3; путь: 20,102
- отсечение ветви
```

```
INF INF INF INF
0,000 INF INF 7,393
0,000 1,157 INF 5,026
0,585 0,000 INF INF

Вершина: 2; путь: 11,785
INF INF INF INF
0,000 INF INF 2,367
INF INF INF INF
0,585 INF INF INF

Вершина: 1; путь: 17,968
INF INF INF INF
INF INF INF INF
INF INF INF INF
0,000 INF INF INF

Вершина: 3; путь: 20,92
- отсечение ветви
INF INF INF INF
0,000 INF INF INF
INF INF INF INF
0,585 0,000 INF INF

Вершина: 3; путь: 16,811
INF INF INF INF
0,000 INF INF INF
INF INF INF INF
INF INF INF INF

Вершина: 1; путь: 16,811
INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Вершина: 0; путь: 16,811
- конец ветви
INF INF INF INF
0,000 INF 0,526 INF
0,000 1,157 INF INF
0,585 0,000 0,000 INF

Вершина: 3; путь: 11,785
INF INF INF INF
0,000 INF 0,526 INF
0,000 INF INF INF
INF INF INF INF
```

```
Вершина: 1; путь: 11,785
INF INF INF INF
INF INF INF INF
0,000 INF INF INF
INF INF INF INF

Вершина: 2; путь: 12,311
INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Вершина: 0; путь: 12,311
- конец ветви
INF INF INF INF
0,000 INF INF INF
0,000 1,157 INF INF
INF INF INF INF

Вершина: 2; путь: 11,785
INF INF INF INF
0,000 INF INF INF
INF INF INF INF
INF INF INF INF

Вершина: 1; путь: 12,942
- отсечение ветви
Длина наименьшего пути: 12,311
Наименьший путь: 0 <- 2 <- 1 <- 3 <- 0 <-
```

3.3. Методы поиска

Листинг 3. Исходный текст

```
(*Методы поиска
 *
 *
 *)

type TRealFunc = function(x: double): double;
type TInterval = record a: double; b: double; end;

function search_f(x: double): double;
begin
  search_f := Power(x, 4) - 14*Power(x,3) + 60*Power(x,2) - 70*x;
end;

function grad_f(x: double): double;
begin
  grad_f := Power(x-4,2) + 1;
end;
```

```
function grad_df(x: double): double;
begin
  grad_df := 2*(x-4);
end;

function grad_d2f(x: double): double;
begin
  grad_d2f := 2;
end;

function xi(i: integer; a,h: double): double;
begin
  xi := a + h*i;
end;

function prev(i,mini: integer): integer;
begin
  if i = mini then prev := i
  else prev := i-1;
end;

function next(i,maxi: integer): integer;
begin
  if i = maxi then next := i
  else next := i+1;
end;

function passive(f: TRealFunc; a, b, eps: double): TInterval;
var N, i, imin: integer;
    t, min, h: double;
begin
  h := eps/2;
  N := trunc((b-a) / h);

  imin := 0;
  min := f(xi(imin,a,h));
  for i:= 1 to N do begin
    t := f(xi(i,a,h));
    if t < min then begin
      min := t;
      imin := i;
    end;
  end;

  passive.a := xi(prev(imin,0), a, h);
  passive.b := xi(next(imin,N), a, h);
end;

function minof4(a: array of double): integer;
var i,imin: integer;
begin
  imin := 0;
  for i:= 1 to 3 do
    if a[i] < a[imin] then imin := i;
  minof4 := imin+1;
end;

function dichotomy(f: TRealFunc; a, b, eps: double; memo: TMemo): TInterval;
var delta: double;
    newi,i,maxcycles: integer;
    x,y: array[1..4] of double;
    s: string;
```

```
begin
  delta := eps/2;
  x[1] := a; y[1] := f(x[1]);
  x[4] := b; y[4] := f(x[4]);
  x[2] := (x[1]+x[4])/2 - delta/2; y[2] := f(x[2]);
  x[3] := (x[1]+x[4])/2 + delta/2; y[3] := f(x[3]);

  if abs(x[2]-x[3]) = 0 then
    raise Exception.Create('Переменная epsслишкоммала  ');

  maxcycles := 999;
  while not (x[4]-x[1] < eps) do begin
    newi := minof4(y);

    s := '';
    for i:= 1 to 4 do s := s + FloatToStr(x[i]) + #9;
    memo.Lines.Append(s);

    x[1] := x[prev(newi,1)]; y[1] := y[prev(newi,1)];
    x[4] := x[next(newi,4)]; y[4] := y[prev(newi,4)];
    x[2] := (x[1]+x[4])/2 - delta/2; y[2] := f(x[2]);
    x[3] := (x[1]+x[4])/2 + delta/2; y[3] := f(x[3]);

    dec(maxcycles);
    if maxcycles = 0 then raise Exception.Create('Методнесходится  ');
  end;

  dichotomy.a := x[1];
  dichotomy.b := x[4];
end;

function fib(n: integer): integer;
begin
  if n > 2 then fib := fib(n-1) + fib(n-2)
  else fib := 1;
end;

function fibonacci(f: TRealFunc; a, b, eps: double; n: integer;
                   memo: TMemo): TInterval;
var x,y: array[1..4] of double;
    i,j: integer;
    t: double;
    s: string;
begin
  t := ((b-a)*fib(n-1)+eps*sign(n))/fib(n);
  x[1] := a;                      y[1] := f(x[1]);
  x[4] := b;                      y[4] := f(x[4]);
  x[2] := x[4] - t;               y[2] := f(x[2]);
  x[3] := x[1] + t;               y[3] := f(x[3]);

  for i:= 1 to n-3 do begin
    s := '';
    for j:= 1 to 4 do s := s + FloatToStr(x[j]) + #9;
    memo.Lines.Append(s);

    if y[2] < y[3] then begin
      x[4] := x[3];               y[4] := y[3];
      x[3] := x[2];               y[3] := y[2];
      x[2] := x[4] - x[3] + x[1]; y[2] := f(x[2]);
    end
    else begin
      x[1] := x[2];               y[1] := y[2];
```

```
      x[2] := x[3];               y[2] := y[3];
      x[3] := x[1] + x[4] - x[2]; y[3] := f(x[3]);
    end;
  end;

  fibonacci.a := x[1];
  fibonacci.b := x[4];
end;

function goldenratio(f: TRealFunc; a, b, eps: double; memo: TMemo): TInterval;
var x,y: array[1..4] of double;
    j: integer;
    s: string;
    tau: double;
begin
  tau := (1 + sqrt(5))/2;
  x[1] := a;                      y[1] := f(x[1]);
  x[4] := b;                      y[4] := f(x[4]);
  x[2] := x[4] - (x[4]-x[1])/tau; y[2] := f(x[2]);
  x[3] := x[1] + (x[4]-x[1])/tau; y[3] := f(x[3]);

  while not (x[4]-x[1] < eps) do begin
    s := '';
    for j:= 1 to 4 do s := s + FloatToStr(x[j]) + #9;
    memo.Lines.Append(s);

    if y[2] < y[3] then begin
      x[4] := x[3];               y[4] := y[3];
      x[3] := x[2];               y[3] := y[2];
      x[2] := x[4] - x[3] + x[1]; y[2] := f(x[2]);
    end
    else begin
      x[1] := x[2];               y[1] := y[2];
      x[2] := x[3];               y[2] := y[3];
      x[3] := x[1] + x[4] - x[2]; y[3] := f(x[3]);
    end;
  end;

  goldenratio.a := x[1];
  goldenratio.b := x[4];
end;
```

Результаты (см. рис. 3):

```
-- Пассивный поиск: x^o = [0,775, 0,785]; f(x^o) = -24,368530859375
-4 1,9975 2,0025 8
-4 -1,00125 -0,99625 2,0025
-1,00125 0,498125 0,503125 2,0025
0,498125 1,2478125 1,2528125 2,0025
0,498125 0,87046875 0,87546875 1,2478125
0,498125 0,684296875 0,689296875 0,87546875
0,684296875 0,7773828125 0,7823828125 0,87546875
0,7773828125 0,82392578125 0,82892578125 0,87546875
0,7773828125 0,798154296875 0,803154296875 0,82392578125
0,7773828125 0,7852685546875 0,7902685546875 0,798154296875
-- Метод дихотомии: x^o = [0,7773828125, 0,7852685546875]; f(x^o) =
-24,3692227783499
-4 0,583587977457733 3,41641202254227 8
-4 -1,16717595491547 0,583587977457733 3,41641202254227
```

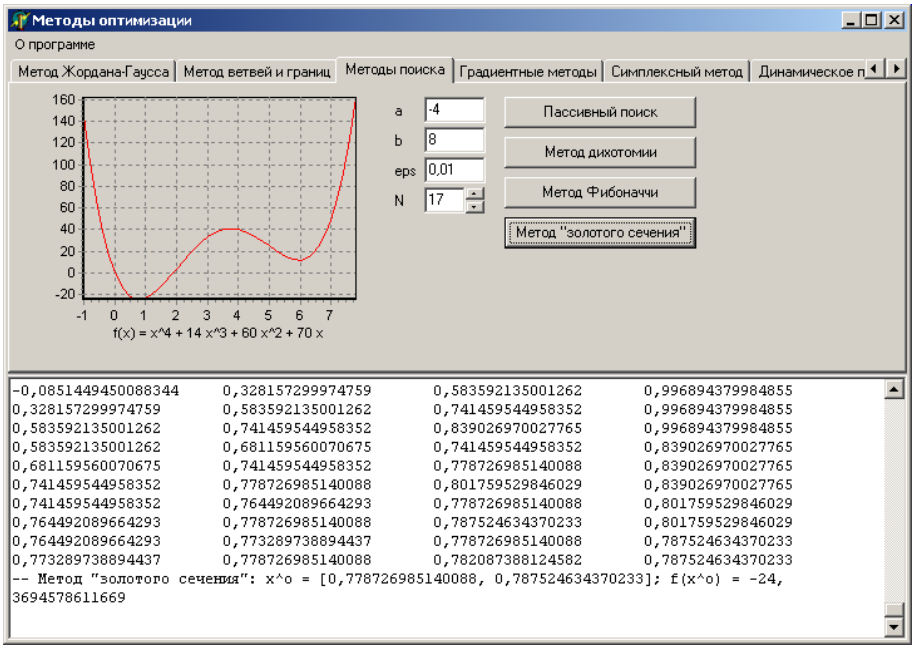

Рис. 3. Методы поиска

```
-1,16717595491547 0,583587977457733 1,66564809016907
3,41641202254227
-1,16717595491547 -0,0851158422041314 0,583587977457733
1,66564809016907
-0,0851158422041314 0,583587977457733 0,996944270507203
1,66564809016907
-0,0851158422041314 0,328240450845339 0,583587977457733
0,996944270507203
0,328240450845339 0,583587977457733 0,741596743894809
0,996944270507203
0,583587977457733 0,741596743894809 0,838935504070127
0,996944270507203
0,583587977457733 0,680926737633051 0,741596743894809
0,838935504070127
0,680926737633051 0,741596743894809 0,778265497808369
0,838935504070127
0,741596743894809 0,778265497808369 0,802266750156567
0,838935504070127
0,741596743894809 0,765597996243008 0,778265497808369
0,802266750156567
0,765597996243008 0,778265497808369 0,789599248591206
0,802266750156567
0,765597996243008 0,776931747025845 0,778265497808369
0,789599248591206
```

```
-- Метод Фибоначчи: x^o = [0,776931747025845, 0,789599248591206]; f(x^o) =
-24,3691188159034
-4 0,583592135001262 3,41640786499874 8
-4 -1,16718427000252 0,583592135001262 3,41640786499874
-1,16718427000252 0,583592135001262 1,66563145999495
3,41640786499874
-1,16718427000252 -0,0851449450088344 0,583592135001262
1,66563145999495
-0,0851449450088344 0,583592135001262 0,996894379984855
1,66563145999495
-0,0851449450088344 0,328157299974759 0,583592135001262
0,996894379984855
0,328157299974759 0,583592135001262 0,741459544958352
0,996894379984855
0,583592135001262 0,741459544958352 0,839026970027765
0,996894379984855
0,583592135001262 0,681159560070675 0,741459544958352
0,839026970027765
0,681159560070675 0,741459544958352 0,778726985140088
0,839026970027765
0,741459544958352 0,778726985140088 0,801759529846029
0,839026970027765
0,741459544958352 0,764492089664293 0,778726985140088
0,801759529846029
0,764492089664293 0,778726985140088 0,787524634370233
0,801759529846029
0,764492089664293 0,773289738894437 0,778726985140088
0,787524634370233
0,773289738894437 0,778726985140088 0,782087388124582
0,787524634370233
-- Метод "золотого сечения": x^o = [0,778726985140088, 0,787524634370233];
f(x^o) = -24,3694578611669
```

3.4. Градиентные методы

Листинг 4. Исходный текст

```
(*Градиентные методы
 *
 *
 *)

function firstorder(f,df: TRealFunc; a, b, h, eps: double; memo: TMemo): double;
var prevx,x: double;
    maxcycles: integer;
begin
  memo.Lines.Append('x'+#9
                  +'f'+#9
                  +'df'+#9
                  +'h');
  x := b;
```

```
  maxcycles := 999;
  repeat
    dec(maxcycles);
    if maxcycles = 0 then raise Exception.Create('Методнесходится  ');
    memo.Lines.Append(FloatToStr(x)+#9
                     +FloatToStr(f(x))+#9
                     +FloatToStr(df(x))+#9
                     +FloatToStr(h));
    prevx := x;
    x := x - h * df(x);
  until abs(prevx-x) < eps;

  firstorder := x;
end;

function firstorderext(f,df: TRealFunc; a, b, h, eps: double; memo: TMemo): double;
var prevx,prevdf,x,t: double;
    maxcycles: integer;
begin
  memo.Lines.Append('x'+#9
                   +'f'+#9
                   +'df'+#9
                   +'h');
  x := b; prevx := a; prevdf := df(x);
  memo.Lines.Append(FloatToStr(x)+#9
                   +FloatToStr(f(x))+#9
                   +FloatToStr(df(x))+#9
                   +FloatToStr(h));
  maxcycles := 999;
  repeat
    dec(maxcycles);
    if maxcycles = 0 then raise Exception.Create('Методнесходится  ');

    t := x;
    x := x - h * df(x);

    if not (f(x) - f(prevx) <= -eps*h*Power(abs(df(x)),2)) then begin
      x := t;
      h := h / 2;
      continue;
    end;

    if sign(df(x)) = sign(prevdf) then begin
      x := t;
      h := h * 4/3;
      continue;
    end;

    prevx := t;
    prevdf := df(x);

    memo.Lines.Append(FloatToStr(x)+#9
                     +FloatToStr(f(x))+#9
                     +FloatToStr(df(x))+#9
                     +FloatToStr(h));
  until abs(prevx-x) < eps;

  firstorderext := x;
end;

function secondorder(f,df,d2f: TRealFunc; a, b, h, eps: double; memo: TMemo): double;
var prevx,x: double;
```

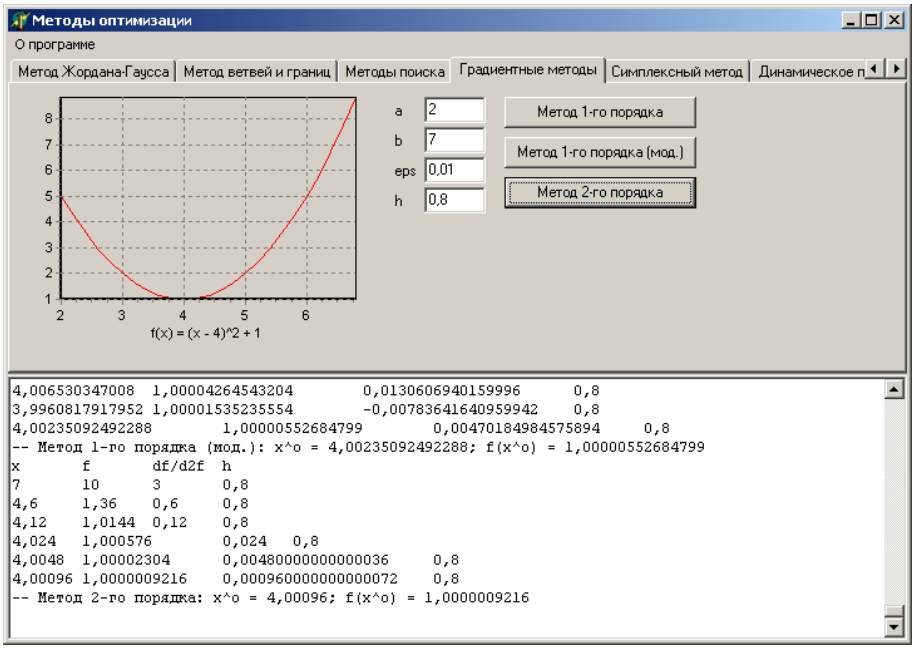

Рис. 4. Градиентные методы

```
    maxcycles: integer;
begin
  memo.Lines.Append('x'+#9
                    +'f'+#9
                    +'df/d2f'+#9
                    +'h');
  x := b;
  memo.Lines.Append(FloatToStr(x)+#9
                    +FloatToStr(f(x))+#9
                    +FloatToStr(df(x)/d2f(x))+#9
                    +FloatToStr(h));
  maxcycles := 999;
  repeat
    dec(maxcycles);
    if maxcycles = 0 then raise Exception.Create('Методнесходится  ');

    prevx := x;
    x := x - h * df(x) / d2f(x);

    memo.Lines.Append(FloatToStr(x)+#9
                      +FloatToStr(f(x))+#9
                      +FloatToStr(df(x)/d2f(x))+#9
                      +FloatToStr(h));
  until abs(prevx-x) < eps;

  secondorder := x;
end;
```

Результаты (см. рис. 4):

```
x f df h
```

```
7 10 6 0,8
2,2 4,24 -3,6 0,8
5,08 2,1664 2,16 0,8
3,352 1,419904 -1,296 0,8
4,3888 1,15116544 0,7776 0,8
3,76672 1,0544195584 -0,466559999999999 0,8
4,139968 1,019591041024 0,279935999999999 0,8
3,9160192 1,00705277476864 -0,1679616 0,8
4,05038848 1,00253899891671 0,100776959999999 0,8
3,969766912 1,00091403961002 -0,0604661759999994 0,8
4,0181398528 1,00032905425961 0,0362797056000002 0,8
3,98911608832 1,00011845953346 -0,0217678233600003 0,8
4,006530347008 1,00004264543204 0,0130606940159996 0,8
3,9960817917952 1,00001535235554 -0,00783641640959942 0,8
4,00235092492288 1,00000552684799 0,00470184984575894 0,8
-- Метод 1-го порядка (мод.): x^o = 4,00235092492288; f(x^o) = 1,00000552684799
x f df/d2f h
7 10 3 0,8
4,6 1,36 0,6 0,8
4,12 1,0144 0,12 0,8
4,024 1,000576 0,024 0,8
4,0048 1,00002304 0,00480000000000036 0,8
4,00096 1,0000009216 0,000960000000000072 0,8
-- Метод 2-го порядка: x^o = 4,00096; f(x^o) = 1,0000009216
```

3.5. Симплексный метод

Листинг 5. Исходный текст

```
(*Симплексный метод
 *
 *
 *)

procedure Simplex_Exclude(var matrix: TMatrix; n,m: integer);
var i,j: integer;
    t: extended;
begin
  for j := 0 to leny(matrix)-1 do begin
    if j = m then continue;
    t := matrix[n,j];
    for i := 0 to lenx(matrix)-1 do
      matrix[i,j] := matrix[i,j] - matrix[i,m]*t;
  end;
end;

function simplex(var c: TArray; var ogr: TMatrix; memo: TMemo): TArray;
var i,j,t,imin,jmin: integer;
    EOC: boolean;
    tt: double;
    a: TMatrix;
    cd,d,m1,res: TArray;
    maxcycles: integer;
```

```
begin
  t := Length(c);
  SetLength(c,t+leny(ogr)+1);
  for i:= t to length(c)-1 do c[i] := 0;
  SetLength(a,lenx(ogr)+leny(ogr),leny(ogr));
  for j:= 0 to leny(a)-1 do
  for i:= 0 to lenx(a)-1 do
    if i < lenx(ogr)-1 then a[i,j] := ogr[i,j]
    else if i = j+lenx(ogr)-1 then a[i,j] := 1
    else if i = lenx(a)-1 then a[i,j] := ogr[lenx(ogr)-1,j]
    else a[i,j] := 0;

  SetLength(m1,lenx(a));

  SetLength(d,leny(a));
  for i:= 0 to Length(d)-1 do d[i] := i + leny(a)-1;
  SetLength(cd,leny(a));

  maxcycles := 99;
  repeat
    dec(maxcycles);
    if maxcycles = 0 then
      raise Exception.Create('Методнесходится  ');

  for i:= 0 to Length(cd)-1 do cd[i] := c[round(d[i])];

  memo.Lines.Append('D:'); WriteArray(d,memo);
  memo.Lines.Append('бС:'); WriteArray(cd,memo);
  memo.Lines.Append('C:'); WriteArray(c,memo);
  memo.Lines.Append('A|B:'); WriteMatrix(a,memo);

  for i:= 0 to Length(m1)-1 do begin
    m1[i] := 0;
    for j:= 0 to Length(m1)-1 do
      m1[i] := m1[i] + cd[j] * a[i,j];
    m1[i] := m1[i] - c[i];
  end;

  memo.Lines.Append('A|B (m+1):'); WriteArray(m1,memo);

  EOC := True;
  for i:= 0 to Length(m1)-1 do
    if m1[i] < 0 then begin
      EOC := False;
      break;
    end;
  if EOC then break;

  imin := 0;
  for i:= 1 to length(m1)-1 do
    if m1[i] < m1[imin] then imin := i;

  jmin := 0;
  for j:= 1 to leny(a)-1 do
    if not (a[imin,j] = 0) and
       not (a[imin,jmin] = 0) then
    if a[lenx(a)-1,j] / a[imin,j] < a[lenx(a)-1,jmin] / a[imin,jmin] then
      jmin := j;

  tt := a[imin,jmin];
  d[jmin] := imin;
  cd[jmin] := c[jmin+t];
```

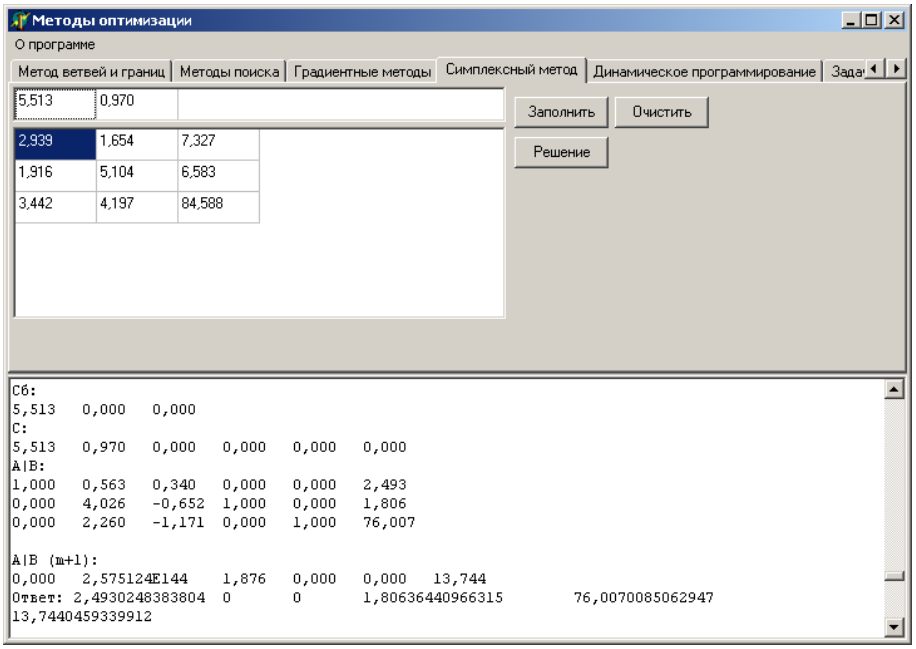

Рис. 5. Симплексный метод

```
  for i:= 0 to lenx(a)-1 do
    a[i,jmin] := a[i,jmin] / tt;
  Simplex_Exclude(a,imin,jmin);

  until false;

  SetLength(res,lenx(a));
  for i:= 0 to Length(res)-1 do res[i] := 0;
  for i:= 0 to Length(d)-1 do res[round(d[i])] := a[lenx(a)-1,i];
  res[lenx(a)-1] := m1[length(m1)-1];

  simplex := res;
end;
```

Результаты (см. рис. 5):

```
-- Симплексный метод:
D:
2,000 3,000 4,000
Сб:
0,000 0,000 0,000
C:
5,513 0,970 0,000 0,000 0,000 0,000
A|B:
2,939 1,654 1,000 0,000 0,000 7,327
1,916 5,104 0,000 1,000 0,000 6,583
3,442 4,197 0,000 0,000 1,000 84,588
```

```
A|B (m+1):
-2,771607E15 -0,970 0,000 0,000 0,000 0,000
D:
0,000 3,000 4,000
Сб:
5,513 0,000 0,000
C:
5,513 0,970 0,000 0,000 0,000 0,000
A|B:
1,000 0,563 0,340 0,000 0,000 2,493
0,000 4,026 -0,652 1,000 0,000 1,806
0,000 2,260 -1,171 0,000 1,000 76,007

A|B (m+1):
0,000 2,575124E144 1,876 0,000 0,000 13,744
Ответ: 2,4930248383804 0 0 1,80636440966315
76,0070085062947 13,7440459339912
```

3.6. Динамическое программирование

Листинг 6. Исходный текст

```
(*Динамическое программирование
 *
 *
 *)

type TCompareFun = function(a,b: double): integer;

function less(a,b: double): integer;
begin
  if a < b then less := 1
  else if a = b then less := 0
  else less := -1;
end;

function greater(a,b: double): integer;
begin
  if a > b then greater := 1
  else if a = b then greater := 0
  else greater := -1;
end;

function dynamic_progr(mx, my: TMatrix; compare: TCompareFun; memo: TMemo): TMatrix;
var i,j: integer;
    t: TMatrix;
begin
  memo.Lines.Append('Числовячейкесоответствуетценеперемещениявсоотвклетку(       . )');
  memo.Lines.Append('Матрицаперемещенийпогоризонтали   :');
  WriteMatrix(mx, memo);
  memo.Lines.Append('Матрицаперемещенийповертикали   :');
  WriteMatrix(my, memo);

  SetLength(t, lenx(mx), leny(my));
```

```
  for i:= 0 to lenx(mx)-1 do
  for j:= 0 to leny(my)-1 do
    if (i = 0) and (j = 0) then
      t[i,j] := 0
    else if j = 0 then
      t[i,j] := t[i-1,j] + mx[i,j]
    else if i = 0 then
      t[i,j] := t[i,j-1] + my[i,j]
    else if compare(mx[i,j]+t[i-1,j], my[i,j]+t[i,j-1]) <> -1 then
      t[i,j] := t[i-1,j] + mx[i,j]
    else if compare(my[i,j]+t[i,j-1], mx[i,j]+t[i-1,j]) <> -1 then
      t[i,j] := t[i,j-1] + my[i,j];

  dynamic_progr := t;
end;

function getpath(m: TMatrix; compare: TCompareFun): TMatrix;
var i,j,n: integer;
    t: TMatrix;
begin
  i := lenx(m)-1; j := leny(m)-1;
  SetLength(t,2,j+i+1);

  t[0,0] := i; t[1,0] := j;

  n := 1;
  while not ((i = 0) and (j = 0)) do begin
    if j = 0 then
      i := i-1
    else if i = 0 then
      j := j-1
    else if compare(m[i-1,j], m[i,j-1]) <> -1 then
      i := i-1
    else if compare(m[i,j-1], m[i-1,j]) <> -1 then
      j := j-1;

    t[0,n] := i; t[1,n] := j;

    n := n + 1;
  end;

  getpath := t;
end;
```

Результаты (см. рис. 6):

```
-- Метод динамического программирования:
(число в клетке соответствует цене перемещения в соотв. клетку)
Матрица перемещений по горизонтали:
7,031 3,204 6,672 6,085
2,801 7,712 3,821 3,159
4,304 8,775 1,747 4,806
2,296 2,627 2,539 4,626

Матрица перемещений по вертикали:
5,248 7,384 3,380 6,381
6,397 3,508 4,398 1,228
6,998 5,011 3,209 2,109
0,078 3,955 9,727 6,056
```

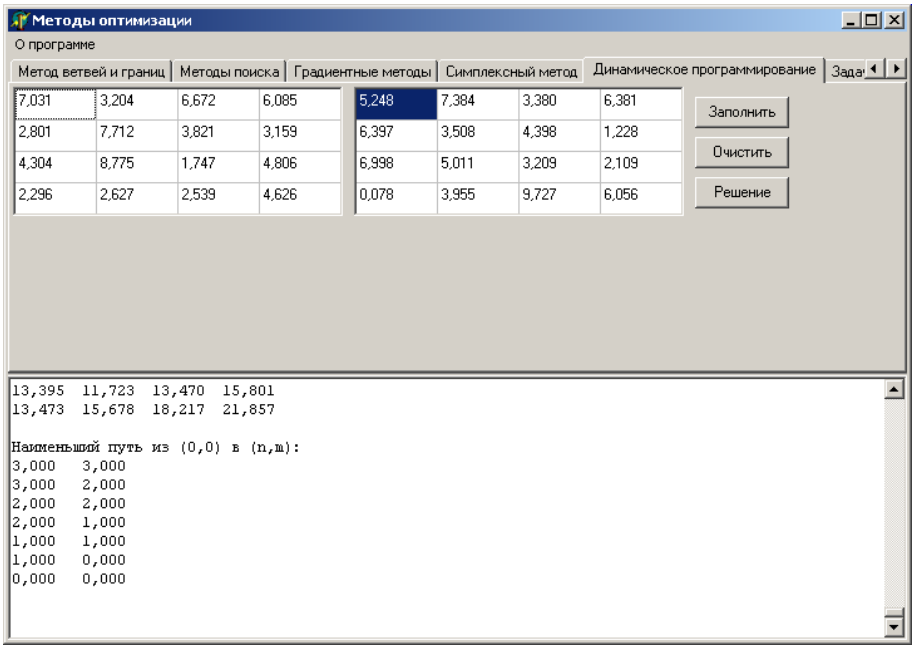

Рис. 6. Динамическое программирование

```
Матрица наименьших путей:
0,000 3,204 9,876 15,961
6,397 6,712 10,533 13,692
13,395 11,723 13,470 15,801
13,473 15,678 18,217 21,857

Наименьший путь из (0,0) в (n,m):
3,000 3,000
3,000 2,000
2,000 2,000
2,000 1,000
1,000 1,000
1,000 0,000
0,000 0,000
```

3.7. Задача о брахистохроне

Листинг 7. Исходный текст

```
(*Задача обрахистохроне
 *
 *
 *)
```

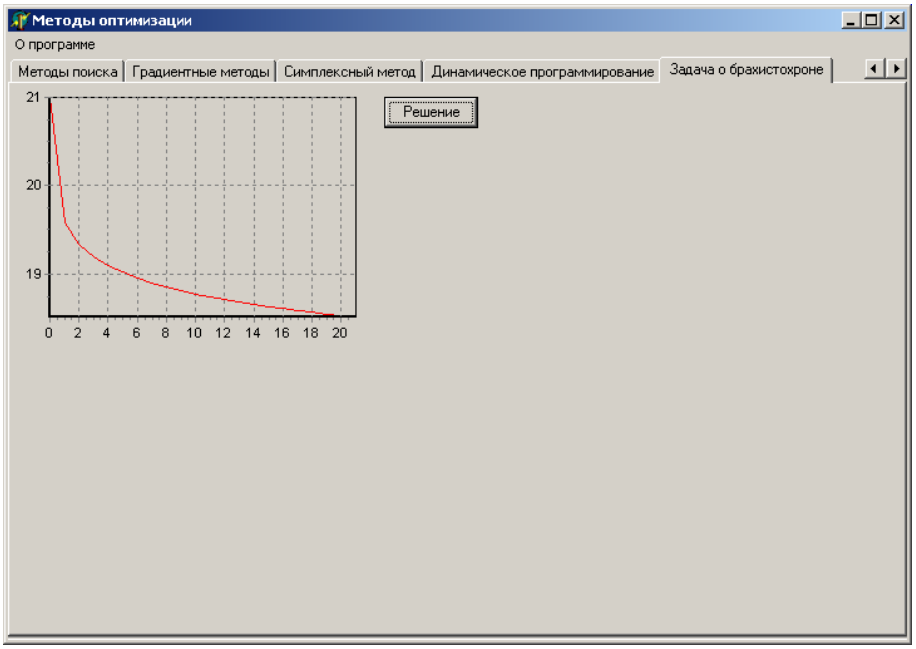

Рис. 7. Задача о брахистохроне

```
function ds(dx,dy: double): double; overload;
begin
  ds := sqrt(power(dx,2) + power(dy,2));
end;

function ds(v0,a,t: double): double; overload;
begin
  ds := v0*t + a*power(t,2)/2;
end;

function dt(dx,dy,a,v0: double): double;
begin
  if (abs(a) < 0.00001) and (abs(v0) < 0.00001) then dt := Infinity
  else if abs(a) < 0.00001 then dt := ds(dx,dy)/v0
  else dt := (-v0 + sqrt(2*a*ds(dx,dy) + power(v0,2)))/a;
end;

procedure brachistochrone(sx,sy: integer;
                          memo: TMemo; series: TLineSeries);
var Level,i,Index: integer;
    Time: TMatrix;
    TimeMin,dX,A,X,Y,Amin: array[0..1000] of real;
    Remove,dY,AllTime: real;
const g = 9.81;

function IndexMin(m: TMatrix;
                  level,index: integer):integer;
var i,mini: integer;
begin
```

```
  mini := Index+1;
  for i:= Index+2 to leny(m)-1 do
    if m[level+1,i] < m[level+1,mini] then
      mini := i;
  IndexMin := mini;
end;

begin
  SetLength(time,sx+1,sx);
  Remove := 1;
  dY := 1;
  Index := 0;
  AllTime := 0;

  for i:= 0 to sx-1 do Time[0,i]:=0;
  TimeMin[0] := 0;

  for Level:= 0 to sx-1 do begin
    dX[Index] := 0;
    for i:= Index+1 to sx-1 do begin
      dX[i] := dX[i-1] + Remove;
      A[i] := g/ds(dx[i],dy);
      Time[Level+1,i] := dt(dx[i],dy,a[i],a[i]*alltime);
    end;

    if Level = 0 then begin
      X[Level] := 0;
      Y[Level] := sx-1;
    end;

    Index := IndexMin(time,level,index);
    Amin[Level+1] := A[Index];
    TimeMin[Level+1] := Time[Level+1,Index];
    AllTime := AllTime + TimeMin[Level+1];
  end;

  for i:=0 to sx-1 do begin
    X[i+1] := X[i]+Remove;
    Y[i+1] := Y[i]-Amin[i+1]*TimeMin[i+1]*TimeMin[i+1]/2;
    series.AddXY(X[i],Y[i]);
  end;
end;
```

Список литературы

[1] Жирков В. Ф. Моделирование производственных процессов с дискретным характером управления: Учеб. пособие. — Владимир: ВПИ, 1984. — 84 с.

[2] Жирков В. Ф. Методы и алгоритмы оптимизации нелинейных задач: Учеб. пособие / ВПИ. Владимир, 1986. — 84 с.