ВЫЧИСЛЕНИЕ НУЛЕЙ И ПОЛЮСОВ ФУНКЦИЙ НА ОСНОВЕ УСТОЙЧИВОЙ АДРЕСНОЙ СОРТИРОВКИ С ПРИЛОЖЕНИЕМ К ПОИСКУ И РАСПОЗНАВАНИЮ

Я.Е. Ромм, М.Ю. Гуревич, С.С. Белоконова, И.А. Соловьева

Таганрогский Государственный Педагогический Институт, 347936, г. Таганрог, ул. Инициативная, 48,
тел. (863)(44) 2-18-07, факс (863)(44) 7-40-33, rector@tgpi.ttn.ru

The article is devoted to program method of application parallelizing internal sorting on a key for localization and the approximate computation of polynomials` zeroes with the appendix to search and recognition. The method defines multiplicity of zeroes, it is spreaded to calculation of functions` poles taking into account its order and it is applicable to determination of functions` extremes and extremal elements of numerical sequences. Search patterns and recognition of images to extreme attributes are under construction on this basis. Time complexity of consecutive and parallel realization of method is estimated. Stability of the offered patterns is proved and verified experimentally.

Базовая схема сортировки, локализации и вычисления действительных нулей многчленов

Используется быстрая сортировка под названием *sort*, с тем же именем для реализующей ее процедуры. Сортировка *sort* основана на адресном слиянии двух упорядоченных массивов по матрице сравнений:

	$-\infty$	2	3	3	4	∞
$-\infty$	0	+	+	+	+	+
3	-	-	0	0	+	+
4	-	-	-	-	0	+
4	-	-	-	-	0	+
5	-	-	-	-	-	+
∞	-	-	-	-	-	0

Последовательный алгоритм слияния выполняется поэлементным обходом в порядке слева направо, сверху вниз неотрицательной области матрицы сравнений. Адрес вставки элемента в выходной массив задается соотношениями

$$c_{i+j}=\begin{cases} b_i, & \text{если } \alpha_{ij}=-1,\ \alpha_{i(j+1)}\ge 0,\\ a_j, & \text{если } \alpha_{ij}\ge 0,\ \alpha_{(i+1)j}=-1,\end{cases}\qquad \alpha_{ij}=\begin{cases} 1, & a_j>a_i,\\ 0, & a_j=a_i,\\ -1, & a_j<a_i,\end{cases} \tag{1}$$

где i, j – произвольно взятые индексы при ограничениях $n+1\ge i\ge 0\le j\le m+1$; n, m, соответственно, – число элементов упорядоченных массивов b и a; c – массив, получаемый в результате их слияния; α_{ij} – элемент i-й строки и j-го столбца матрицы сравнения; при этом a располагается на входе таблицы горизонтально сверху, b – вертикально слева. По входным индексам i, j на основе (1) однозначно указывается выходной индекс $i+j$ и обратно, – с учетом принадлежности элемента конкретно одному из двух входных массивов. Эта обратная адресация сохраняется для сортировки слиянием, организуемой вначале «слиянием» каждой пары элементов с запоминанием обратных адресов, затем, аналогично, последовательным слиянием упорядоченных пар, упорядоченных четверок и т. д. Программа сортировки представлена процедурой (Object Pascal):

```
CONST n00 = 1024*127; nn = TRUNC(n00/2);
TYPE
vect1 = ARRAY [0..n00] OF EXTENDED;
vect2 = ARRAY [0..2*nn+nn] OF LONGINT;
VAR nn0: LONGINT;
c: vect1; e: vect2;
PROCEDURE sort (VAR nn0: LONGINT; VAR c:vect1; VAR e:vect2);
TYPE
vecc = ARRAY[0..nn+nn div 2] OF LONGINT;
VAR ab:INTEGER; i,j,k,l,m,r,nm,p,n: LONGINT;
   e1, e2: vecc;
BEGIN
  p:= ROUND(LN(nn0)/LN(2)); n:=TRUNC(EXP(p*LN(2)));
  FOR l := 1 TO n DO e[l] := l;
  FOR r := 1 TO p DO BEGIN  m :=TRUNC(exp(r*ln(2))); nm:=n DIV m;
  FOR k := 0 TO nm-1 DO BEGIN
  FOR l := 1 TO m DIV 2 DO BEGIN
```

```
IF k * m + l > nn0 THEN e1[l]:=k * m + l ELSE  e1[l] := e[k * m + l];
IF k * m + m DIV 2+ l > nn0 THEN e2[l]:=k * m + m DIV 2+l ELSE
e2[l] := e[k * m + m DIV 2 + l];  END;
i := 1; j := 0; WHILE i+j <= m DO BEGIN
IF i = m DIV 2 + 1 THEN ab := -1;
IF j = m DIV 2 THEN ab := 1 ;
IF (k * m + i > nn0) OR (k * m + m DIV 2 + j > nn0-1) THEN ab:=1;
IF (i <= m DIV 2) AND (j <= m DIV 2 -1) AND (k * m + i<= nn0)
   AND (k * m + m DIV 2 + j <= nn0-1) THEN  BEGIN
       IF c[e2[j + 1]] - c[e1[i]]= 0 THEN ab:=0;
       IF c[e2[j + 1]] - c[e1[i]]> 0 THEN ab:=1;
       IF c[e2[j + 1]] - c[e1[i]]< 0 THEN ab:=-1  END;
  IF ab >= 0 THEN  BEGIN e[k*m+i+j]:= e1[i]; i:=i+1 END ELSE
   BEGIN e[k * m + i + j] := e2[j + 1]; j := j + 1  END   END  END  END
END;
```

Число сортируемых элементов произвольно при условии nn0<=n00. Освобождение от степени двойки для значения nn0 достигается путем имитации дописывания до ближайшей к nn0 степени двойки возрастающей последовательности элементов входного массива, заведомо больших, чем сортируемые, начиная с номера nn0+1. Имитация реализуется положительными значениями всех элементов всех матриц сравнения, которые сооветствуют входным элементам с номерами > nn0.

Сортировка *sort* не перемещает сортируемые элементы и их ключи, а лишь вызывает ключи на момент сравнения по обратным адресам (входным индексам). Обратные адреса элементов перемещаются в порядке отсортированных элементов. В таком порядке окончательные значения обратных адресов на выходе сортировки располагаются в массиве *e*.

Максимально параллельный вариант сортировки *sort* оценивается временем $T(N^2/4)=O(\log_2 N)$, в скобках левой части – количество процессоров / 1 /, последовательный вариант – выполняется за время $T=O(N\log_2 N)$. Такая сортировка является частным случаем класса сортировок *m*-путевым слиянием / 2 /, отличающихся от аналогов / 3 / параллелизмом, – в пределе достигается оценка времени $T(N^2/2)=O(1)$, – а также прямой и обратной адресностью. По аналогии с приведеннной процедурой все такие сортировки могут выполняться без перемещения записей и ключей.

Нахождение нулей многочлена. На вход процедуры *sort* подается последовательность значений модуля этого многочлена на равномерной сетке, взятой на начальном промежутке конечной длины. К выходу процедуры подсоединяется условный оператор, локализующий после выполнения сортировки все минимумы среди этих значений, – минимумы совпадают с искомыми нулями многочлена. Данный оператор (ниже оператор локализации минимума) имеет вид

```
{sort(nn0,c,e);} k:=1;WHILE k<= nn0 DO BEGIN FOR r := 1 TO k-1 DO
IF ABS(e[k]-e[k-r]) <= eps0 / h THEN GOTO 22; xk:= x0+e[k]*h;
22: k:=k+1 END;
```

В этом программном фрагменте *nn*0 – число узлов, совпадающее с числом сортируемых элементов, *h*– шаг равномерной сетки, *eps*0 – константа, равная радиусу *eps*0-окрестности текущего узла. Эта константа выбирается априори таким образом, чтобы не превысить половины расстояния между ближайшими друг к другу действительными нулями многочлена.

Оператор локализации находит минимальные элементы дискретной последовательности по смыслу своего построения. Работая после сортировки, он для текущего узла, определяемого индексом, записанным в *e*[*k*], находит каждый узел $x0+e[k]*h$, в *eps*0-окрестности которого нет узлов, доставляющих значения элементов входной последовательности (модуля многочлена на сетке), предшествующие *c*[*e*[*k*]] в отсортированном массиве. Иными словами, выявляется *e*[*k*], для которого

$$\mathrm{ABS}(x0+e[k]*h-(x0+e[k-r]*h))=\mathrm{ABS}(e[k]-e[k-r])*h\ >\ eps0\quad\forall\quad r=1,2,\ldots,k-1. \qquad (2)$$

Это означает, в частности, что модуль многочлена в узле $x0+e[k]*h$ не больше значений в остальных узлах его $eps0$-окрестности, точнее, – строго меньше, или это значение является первым слева направо среди равных элементов на выходе отсортированного массива. При выполнении (2) *c*[*e*[*k*]] является в данной окрестности наименьшим в смысле отношения порядка $\le$, что не полностью совпадает со смыслом арифметического неравенства. Среди узлов с равными значениями модуля многочлена в рассматриваемой окрестности – при истинности условия (2) – оператор локализации минимума фиксирует узел, соответствующий первому в порядке нумерации элементу отсортированного массива, независимо от его расположения на сетке. Этот факт опирается на устойчивость сортировки *sort*, – под устойчивостью понимается / 4 / сохранение порядка равных элементов. Локализовать минимальный элемент в рассматриваемом смысле предлагаемым способом можно только при сохранении обратных адресов на выходе сортировки, причем строго в том порядке, в каком располагаются отсортированные элементы.

Аналогично обосновывается работа оператора локализации максимума, при этом максимум также понимается в смысле отношения порядка. Такой оператор имеет вид

```
{sort(nn0,c,e);} k:=1; WHILE k<= nn0 DO BEGIN FOR r := 1 TO nn0-k DO
IF ABS(e[k]-e[k+r]) <= eps0 / h THEN GOTO 222; xk:= x0+e[k]*h;
222: k:=k+1 END;
```

То, что значение параметра *eps*0 не должно достигать половины расстояния между ближайшими друг к другу нулями многочлена следует из построения оператора локализации. В остальных отношениях значение *eps*0 произвольно, но должно быть выбрано и зафиксировано априори.

Замечание 1. Если локализуется экстремальный элемент числовой последовательности, то *h* в операторах локализации минимума и максимума следует положить равным единице. Тогда значение *eps*0 равно целому числу номеров влево и вправо от входного номера *e*[*k*]. Экстремум, локализуемый данными операторами, является локальным экстремумом среди ближайших значений последовательности, отсчитанных на *eps*0 влево и вправо от *e*[*k*]. При этом локализация означает конкретное значение как точки (индекса) экстремума, так и экстремумального значения. Дальнейшей вычислительной обработки, в частности, спуска к точке экстремума в случае собственно числовой последовательности не требуется.

Ниже конструируется алгоритм локализации и вычисления нулей многочлена, который практически универсален для поиска действительных нулей многочлена от одной действительной переменной с действительными коэффициентами на произвольном промежутке. В этом алгоритме к процедуре сортировки и локализации минимума добавляется процедура спуска к наименьшему значению модуля многочлена на равномерной сетке $\min 1$. Процедура циклически повторяется в сужаемой окрестности локализованного минимума до достижения заданной точности приближения к точке минимума.

Алгоритм реализован на Object Pascal в среде Delphi. Программа (с точностью до раздела описаний) не меняется для различных многочленов и различных промежутков поиска нулей. Границы промежутка [*x*00, *x*11] задаются в разделе описания констант. В разделе констант задается граница абсолютной погрешности *eps*, а также параметр *eps*0 для оператора локализации. Как отмечалось, значение *eps*0 не должно достигать половины расстояния между ближайшими друг к другу нулями многочлена. При отсутствии информации о расстоянии между нулями это значение следует выбирать с запасом малости. Выбор *eps*0 определяет ограничение на шаг *h* равномерной сетки. Согласно численным экспериментам $/ 5, 6 /$, допустимо произвольное значение $h \le eps0 / 20$. В разделе констант задается число сортируемых элементов $nn0$ (равное числу узлов сетки). Эти узлы первоначально определяются на текущем промежутке постоянной длины *hh*=*nn*0**h*. В конце программного блока на такой промежуток производится смещение до тех пор, пока не будет пройден весь априори заданный промежуток поиска нулей [*x*00, *x*11]. Константа *mm* равна числу узлов сетки в сужаемой окрестности локализованного минимума. Константа *nn* используется для задания границ массива в соответствии с алгоритмом сортировки *sort*.

В приводимом ниже примере все действительные нули многочлена степени *n*1 априори заданы как элементы одномерного массива *b* в разделе констант. Многочлен на основе его разложения по нулям строит функция *func*, принимающая значение модуля многочлена. Даны два варианта нулей и соответственных констант, один из которых закомментирован и поясняется ниже. Процедура сортировки *sort* полностью совпадает с приведенной выше.

```
PROGRAM Zero;
{$APPTYPE CONSOLE}
uses
  SysUtils;
LABEL 21, 22;
CONST n1 = 24;
b: ARRAY [1..n1] OF EXTENDED =
(1 ,2 ,3 ,4 ,5 ,6.01,6.02,7,8,9 ,10 ,11 ,12 ,13,
14 ,15 ,16 ,17 ,18 ,19 ,20 ,21 ,22 ,23);
CONST eps = 0.0000000000000000001 *1E-14;      eps0 = 0.0049; h = eps0/30;
n00 = 1024*128; nn = TRUNC(n00/2);mm = 128;
x00 = -500; x11 = 500;
{CONST n1=9;
b: ARRAY [1..n1] OF EXTENDED = (1.000000000000104*1E-480,
1.000000000000204*1E-480,1.000000000000304*1E-480,1.000000000000404*1E-480,
1.000000000000504*1E-480,1.000000000000604*1E-480,1.000000000000704*1E-480,
1.000000000000804*1E-480,1.000000000000904*1E-480);
CONST eps = 0.00000000000000000000000001*1E-480;   eps0 = 0.000000000000049*1E-480;
h = eps0/30;
n00 = 1020*128; nn = TRUNC(n00/2); mm = 128;
x00 = 0.99999999995*1E-480; x11 = 1.0000000000115*1E-480;}
TYPE
vect1 = ARRAY [0..n00] OF EXTENDED;
```

```
vect2 = ARRAY [0..2*nn+nn] OF LONGINT;
vect3 = ARRAY [0..n1] OF EXTENDED;
VAR i,j,k,l,ee,nn0: LONGINT;
c: vect1; e: vect2; b1: vect3;
hh,aaa,x,x0,x1,xk,xk0,xk1,h0,min,eps1: EXTENDED;
FUNCTION func (VAR x: EXTENDED; VAR b1: vect3): EXTENDED;
VAR i1: 1..n1;
p: EXTENDED;
BEGIN p:=1; FOR i1:=1 TO n1 DO p:=p*(x-b1[i1]); func := ABS(p) END;
PROCEDURE min1 (VAR x: EXTENDED; VAR ee: INTEGER);
BEGIN  min:=func(x,b1); ee:=0;
FOR i:=1 TO mm DO BEGIN
x:=xk0+i*h0;IF min > func(x,b1) THEN BEGIN min:=func(x,b1);ee:=I END;   END;
END;
PROCEDURE sort (VAR nn0: INTEGER; VAR c: vect1; VAR e: vect2);
```

Описание процедуры sort

```
BEGIN
aaa:=1e62;    nn0:=n00-17;hh:=nn0*h;
x0:=x00;x1:=x11; FOR i:=1 TO n1 DO
b1[i]:=b[i]; WHILE x0 <= x1-h DO BEGIN
FOR i:=1 TO nn0 DO BEGIN x:=x0+i*h; c[i]:=func(x,b1) END;
          Sort (nn0,c,e);
k:=1; WHILE k<= nn0 DO BEGIN
     FOR l := 1 TO k-1 DO
IF ABS(e[k]-e[k-l]) <= eps0/h THEN GOTO 22;       xk:= x0+e[k]*h;
eps1:=eps0;xk0:=xk-eps1;xk1:=xk+eps1;h0:=abs(2*eps1)/mm;
WHILE abs(eps1) > eps DO BEGIN
x:=xk0;  min1(x,ee); eps1:=eps1/1.2;
xk0:=xk0+ee*h0-eps1;xk1:=xk0+ee*h0+eps1;h0:=abs(2*eps1)/mm END;
IF func (xk,b1) = 0 THEN GOTO 21; x:=xk0+ee*h0+eps1;
IF ABS(func(x,b1)/func(xk,b1)) > 0.0000001  THEN GOTO 22;
IF aaa=x THEN GOTO 22;
21:    WRITE ('       ',x:30); aaa:=x;
22:    k:=k+1    END;
 x0:=x0+hh END;
READLN;
END.
```

Результат работы программы

```
6.01000000000000000E+0000  6.02000000000000000E+0000  8.00000000000000000E+0000
1.00000000000000000E+0001  1.20000000000000000E+0001  1.10000000000000000E+0001
1.30000000000000000E+0001  9.00000000000000000E+0000  7.00000000000000000E+0000
5.00000000000000000E+0000  4.00000000000000000E+0000  3.00000000000000000E+0000
2.00000000000000000E+0000  1.00000000000000000E+0000  1.40000000000000000E+0001
1.50000000000000000E+0001  1.70000000000000000E+0001  1.60000000000000000E+0001
1.90000000000000000E+0001  1.80000000000000000E+0001  2.00000000000000000E+0001
2.10000000000000000E+0001  2.20000000000000000E+0001  2.30000000000000000E+0001
```

Переход к метке 22 по условию *func*(*x*) / *func*(*xk*)>0.000001 исключает возможность ошибочного принятия минимумов на концах текущего шага длины *hh* за нули многочлена / 5, 6 /.

Таким образом, все нули заданного многочлена локализованы на промежутке [–500, 500] фактически без погрешности в принятом числовом формате. Промежуток можно существенно увеличить без потери точности, но с замедлением работы программы. Если теперь закомментировать часть раздела констант, относящуюся к данному многочлену, и снять комментарий с другого многочлена, то результатом работы программы окажутся все 9 заданных нулей без погрешности в данном числовом формате. При этом они априори имели 492 совпадающих цифры десятичной мантиссы. Точность в этом случае можно дополнительно увеличить на два десятичных порядка. Этот пример характеризует возможности взаимного отделения и локализации нулей многочлена по предложенной схеме.

Замечание 2. Программа работает устойчиво при любых значениях *nn*0, *n*00, *mm*, по крайней мере, если $64 \le nn0 \le n00 \le 1024*128$, $32 \le mm \le 256$. Значение многочлена можно задать не разложением на множители, что здесь сделано с целью иллюстрации, а с помощью схемы Горнера. При этом в разделе констант будут заданы не нули, а коэффициенты, изменится подпрограмма *func*, других изменений в программе не будет.

Замечание 3. Программа устойчиво работает на произвольном промежутке, – однако медленно, если промежуток большой длины и включает плохо отделенные нули.

Обоснование изложенного метода опирается на принцип максимума модуля / 7 /, согласно которому функция $1/\left| P_n(x) \right|$ не может иметь максимума внутри области своей аналитичности, – вне окрестности нулей многочлена $P_n(x)$, – все минимумы модуля этого многочлена совпадают с его нулями.

На основе нестрогой монотонности модуля многочлена вне достаточно больших границ области его нулей строится схема, локализующая область всех действительных нулей многочлена и одновременно окрестность каждого нуля в отдельности. После локализации нуля осуществляется спуск к его значению до достижения заданной точности приближения. Этот вариант метода с программной реализацией описан в / 8, 9 /. Программа универсально локализует и вычисляет нули многочлена в границах числового диапазона Object Pascal, однако она сравнительно громоздка, поэтому здесь не характеризуется.

С точностью до замены многочлена на другую функцию, программа *Zero* с сохранением устойчивости отыскивает нули функций действительных переменных. Один из примеров для функции $f(x)=x^{41.305+1/3}(\sin(1/x))^{(41.305+1/3)/4}$ на промежутке [- 0.00001, - 0.00001 + 0.0000001] дан в / 5 /.

Пусть рассматривается многочлен *n*-й степени $P_n(z)$ от комплексной переменной *z*. Выполняется предварительное умножение $P_n(z)$ на комплексно-сопряженное значение $\overline{P_n(z)}$ с целью свести задачу поиска нулей многочлена к поиску нулей действительной функции $f(x, y)$ двух действительных переменных

$$f(x,y)=P_n(z)\times\overline{P_n(z)}=\left|P_n(z)\right|^2, \quad z=x+I\,y, \qquad I=\sqrt{-1}. \tag{3}$$

В плоских декартовых координатах вводится равномерная прямоугольная сетка, включающая область, которой принадлежат все нули:

$$x_\ell = x_0 + \ell h, \ \ \ell = 0, 1, ..., \ N_x; \quad y_\ell = y_0 + \ell h, \ \ \ell = 0, 1, ..., \ N_y. \tag{4}$$

Значения $f(x, y)$ из (3) берутся во всех узлах сетки и среди них ищутся все возможные минимумы.

Опираясь на принцип максимума, излагаемая схема учитывает дискретность значений функции на сетке.

Во избежание квадратичного по сравнению с исходным случаем роста памяти, локализация и вычисление нулей $f(x, y)$ выполняется с вспомогательной минимизацией, сводящей пространственный случай к плоскому. С этой целью посредством внешнего цикла выполняется проход по направлению оси *OY* вдоль текущего столбца сетки. В организованном внутри внешнего внутреннем цикле определяется минимальное по столбцам текущей строки (направление вдоль оси *OX*) значение $f(x_i, y_j)$. Для текущего i этот минимум,

$$c_i = \min_{\substack{i=\text{const} \\ 1\le j\le M_y}} f(x_i, y_j), \tag{5}$$

заносится как *i*-й элемент одномерного массива на вход сортировки, *i*=1,2,…,*N*. После формирования входного массива, без изменения ядра программы *Zero*, локализуется абсцисса элемента, локально минимального среди c_i из (5). Текущее значение локализованной абсциссы – $x_k = x_0 + e(k)h$. Значение x_k дает привязку к локализуемому приближению точки минимума на плоскости, x_k фиксируется и по аналогичной схеме локализуется приближение к ординате, обозначаемой $y_{k'}$. В результате

$$f(x_k, y_{k'}) = \min_{\text{j}} f(x_k, y_j), \tag{6}$$

где минимум $\min\limits_{j}$ также найден с помощью оператора локализации. Затем выполняется пространственный аналог плоского спуска к наименьшему значению в окрестности точки $(x_k, y_{k'})$ из (6).

Осуществима локализация области всех комплексных нулей многочлена с одновременным их вычислением в расширенной области на комплексной плоскости. Соответственные программы даны в / 8, 9 /.

Если на вход представленного метода подавать попрограмму *func* (*x, b*1) без взятия модуля значения *func*, то метод будет вычислять все минимумы функции. С использованием оператора локализации максимума аналогично локализуются максимумы. Утверждение относится к функциям одной и двух переменных. Для вычисления локализованных экстремумов достаточно незначительно изменить схему спуска, использованную в программе *Zero*.

Изложенный метод обладает существенным параллелизмом. Параллельно могут выполняться сортировки, на параллельные фрагменты могут разделяться области локализации экстремумов. На случай, когда совокупность всех шагов длины *hh* выполняется параллельно, все экстремумы функции $f(x, y)$ в квадрате со стороной *D* могут быть вычислены по максимально параллельной схеме с точностью до ε_0 за время / 6 / $T\left(\frac{D^2N}{2h^2}\right)\approx 2\left(2+\log_2\frac{\varepsilon}{\varepsilon_0}\right)\tau=O(1)$, где τ – время одного сравнения, ε – произвольная положительная констан-

та, не большая половины минимальной разности одноименных координат соседних экстремумов, при условии, что значения этих координат различны.

Приближенное вычисление нулей многочленов с учетом их кратности. Будем рассматривать многочлен $P_n(z)=\sum_{\ell=0}^{n} a_\ell z^\ell$ и сопоставленную ему функцию $f(x,y)$ из (3). Для вычисления кратности нулей описанная выше схема их вычисления циклически возобновляется с числом повторных шагов, пропорциональным кратности каждого найденного нуля. Определение кратности нулей использует деление функции, нули которой уже найдены, на циклически подсчитываемое выражение

$$pp := pp * ((x - kor\ x)^2 + (y - kor\ y)^2), \qquad (7)$$

где число шагов цикла равно текущему счетчику кратности, $kor\ x + I \times kor\ y$ – текущий корень. После деления исходной функции на выражение (7) программа заново выполняет нахождение всех нулей полученной в результате деления функции, при этом кратность текущего нуля на единицу меньше предыдущей. Деление на машинный ноль обходится делением исходной функции на циклически подсчитываемое выражение, отличное от нуля *ppp*:=*ppp***eps*, где число шагов цикла совпадает с текущим значением порядка найденных нулей, *eps* – наперед заданная граница погрешности вычислений. Таким образом, исходная функция конструируется в виде:

```
FUNCTION func(VAR x,y,korx,kory: EXTENDED;
VAR r:BOOLEAN;VAR param: INTEGER):EXTENDED;
CONST b: vek = (……);  b1: vek = (……);
VAR p, pp,ppp: EXTENDED; i1: INTEGER;
BEGIN p:=1;FOR i1:=1 TO np DO
p:=p*(SQR(x-b[i1])+SQR(y-b1[i1]));
pp := 1; FOR i1 := 1 TO param DO pp := pp * (SQR(x-korx)+SQR(y-kory));
IF r THEN func := ABS(p) ELSE
IF ABS(SQR(x-korx)+SQR(y-kory))> eps THEN func :=  ABS(p/(pp)) ELSE
BEGIN ppp := 1; FOR i1 := 1 TO param DO ppp := ppp * eps;
func :=ABS(p/ppp) END END;
```

Зависимость функции $func\ (xk, yk, kor\ x, kor\ y, r, param)$ полагается от переменных:

xk – действительная часть нуля, вычисленная при первом прогоне программы (далее «фиксированный *xk*»); *yk* – мнимая часть нуля, вычисленная при первом прогоне программы (далее «фиксированный *yk*»); *r* – переменная типа *boolean*, которая изначально принимает значение *true*; *param* – переменная, в которой накапливается порядок найденных нулей; *korx* – текущее значение действительной части нуля; *kory* – текущее значение мнимой части нуля.

В блоке констант под многоточием понимаются конкретные числовые массивы с числом элементов *np*.

Для вычисления кратности нулей организуется разветвление программы в зависимости от найденного количества различных нулей. Если у функции нашелся единственный нуль, то после деления функции $func\ (xk, yk, kor\ x, kor\ y, r, param)$ на выражение (7) и повторного нахождения нулей производится проверка:

```
IF (param<np) AND (ABS(s-xk1)<1e-13) AND (ABS(ss-yk1)<1e-13) THEN ……
```

Если условие истинно, кратность нуля увеличивается на единицу и осуществляется деление на (7), иначе кратность нуля остается равной текущему значению переменной *param*. В случае двух и более различных нулей после аналогичного деления функции $func\ (xk, yk, kor\ x, kor\ y, r, param)$ на выражение (7) программа повторно возвращается к нахождению нулей многочлена, степень которого на единицу меньше предыдущей.

Сравнение найденных ранее нулей с вновь полученным выполняется по метрике Евклида. Если минимум метрики по всем таким сравнениям достаточно мал, то нуль повторился, его кратность увеличивается на единицу, переменной *r* присваивается значение *r* = *false* и производится деление функции на (7) с рекуррентным возвращением программы к началу, иначе запоминается кратность и выполняется переход к следующему фиксированному нулю до тех пор, пока не будет установлена кратность каждого. Сказанное выше представлено следующим программным фрагментом:

```
IF r=true THEN BEGIN  min:=ABS(SQR(korx-aax[1])+SQR(kory-aay[1]));
FOR l:=1 TO ll DO
IF min > ABS(SQR(korx-aax[l])+SQR(kory-aay[l])) THEN
min:=ABS(SQR(korx-aax[l])+SQR(kory-aay[l]))END;
IF r=false THEN BEGIN min:= ABS(SQR(korx-bbx[1])+SQR(kory-bby[1]));
FOR l:=1 TO kk DO
IF min > ABS(SQR(korx-bbx[l])+SQR(kory-bby[l])) THEN
min:=ABS(SQR(korx-bbx[l])+SQR(kory-bby[l]))END;
IF min <= 1e-14 AND (param < np) THEN
BEGIN  param:=param+1; r:=false;……. END;
```

Значение меры «малости» 1e–14 минимума выбрано эвристически в ходе численного эксперимента и может быть варьировано в сторону уменьшения погрешности, что, однако, замедлит выполнение программы.

Схему можно модифицировать для нахождения полюсов комплексной функции с учетом их порядка. Будем рассматривать такую функцию $f(x,y) = g(x,y) + I \cdot p(x,y)$, квадрат модуля обратной к которой возможно представить в виде:

$$\left| \frac{1}{f(x,y)} \right|^2 = u^2(x,y) + v^2(x,y). \qquad (8)$$

Полюса функции $f(x, y)$ с учетом их порядка можно искать как нули функции (8) с учетом кратности. При этом функции, имеющие устранимые особенности, из рассмотрения исключались. Численный эксперимент показал устойчивость предложенной схемы. Примеры и полный текст программ представлены в / 10 /.

Применение локализации экстремумов на основе сортировки к поиску

В целом схема поиска конструируется из двух следующих компонент: из сортировки *sort* в качестве главной конструктивной части схемы поиска и дополнительной составляющей – оператора локализации минимума среди элементов входной последовательности. С помощью схемы, ввиду её математического характера, конструируются более сложные, чем обычно комбинированные признаки («маски») с дополнительными математическими и алгоритмическими условиями поиска текстовых фрагментов. Схема достаточно проста и сравнительно универсальна относительно построения масок на основе выражений отношения, – отчасти потому, что сводит операции поиска к арифметической обработке массивов целого типа.

Поиск в числовом массиве. В случае числового массива оператор локализации позволяет сконструировать условие поиска как нахождение нулевого элемента абсолютной разности между текущим элементом массива и заданным искомым элементом.

Пусть требуется найти в числовой последовательности $a = (a_1, a_2, ..., a_n)$ заданное число b. Для этой цели на вход сортировки подается следующее преобразование входной последовательности: $\tilde{a} = (|a_1 - b|, |a_2 - b|, ..., |a_n - b|)$. К элементам отсортированного массива применяется оператор локализации минимумов, который идентифицирует все нули. Способ распространяется на поиск чисел любого типа. Поиск происходит по той же схеме, однако, не как поиск нуля в массиве, а как поиск минимума модуля, идентифицируемого с точностью до заданной границы погрешности.

Пример 1. Найти в числовой последовательности (2.00600077709, 2.00600077709, 31.123, 2.00600077708, 7.25) заданное число 2.006 с точностью eps=0.00000000001.

На вход сортировки подается числовая последовательность, составленная из модулей разностей между элементами данного массива и числом 2.006. После сортировки массива по программе *sort* применяется оператор локализации и выполняется просмотр идентифицированных минимумов. Дополнительное условие *if c[e[k]]<=eps* выделит все искомые приближения к числу 2.006.

Поиск в массиве строковых элементов. Изложенная схема переносится на поиск в массиве слов следующим образом. Входному массиву слов сопоставляется числовой массив из абсолютных величин разностей asc-кода символа, стоящего на заданной позиции слова входного массива и asc-кода символа, указанного в маске поиска:

$$R[i] := abs(ord(c[i[k]]) - ord(w)); \qquad (9)$$

В (9) c – входной массив строковых элементов, k – номер позиции, заданной в маске поиска, w – символ, заданный в маске поиска, i – номер элемента массива c, $r[i]$ – элемент сопоставляемого числового массива r.

При переходе от массива слов c к числовому массиву r нумерация элементов обоих массивов сохраняется. Поэтому, если элемент входного массива удовлетворяет маске поиска, то $r[i]=0$. Отсюда искомые слова идентифицируются по индексам нулевых элементов числового массива. Адресация к словам происходит как обращение к элементам строкового массива через сохраненные индексы локализованных нулей.

Описанная схема переносится на поиск слов по некоторой комбинации символов. Для этого массив слов переводится в числовой массив путем суммирования абсолютных величин разностей asc-кода символа входного массива и символа «маски» поиска с учетом соответствия позиций. Идентификация искомой комбинации слов основана на том, что сумма нулевых разностей сохранит нулевое значение.

Пример 2. В массиве слов *c*=(*'Katy', 'Ivan', 'Dasha', 'Aleksey', 'Natasha'*) найти слова, содержащие на второй позиции букву *«a»* и на третьей позиции букву *«t»*.

Массив *c* программно переводится в числовой массив *r*, который примет вид *r*=(*0, 40, 1, 26, 0*). В таком виде массив *r* поступает на вход сортировки *sort*, после которой применяется оператор локализации. В результате искомые элементы идентифицируются как два минимальных элемента этого массива, равные нулю. По сохраненным индексам найденных нулей массива *r* определяются искомые элементы строкового массива:

0 1 Katy 0 5 Natasha.

Рассмотренная схема переносится на поиск текстовых фрагментов при более сложных условиях на признаки искомых фрагментов. В качестве признака поиска может быть рассмотрен символ, сочетание символов, сочетание слов, предложений и т.д. Маску поиска можно дополнительно усложнить, используя величину расстояния между символами, словами, предложениями, – на основе элементов массива хранимых входных индек-

сов. Для выполнения поиска заданный текстовый фрагмент будет переводиться в массив строковых элементов (массив слов, словосочетаний или фраз). Затем к массиву строковых элементов будет применяться описанная выше схема.

Поиск в текстовом файле. Изложенный подход применим для решения более общей задачи поиска по заданной маске при дополнительных условиях в текстовом файле / 11 /. Пусть ставится задача найти все предложения текстового файла, удовлетворяющие условию поиска, заключающемуся в наличии в предложениях комбинаций символов и слов, удовлетворяющих заданной маске.

В основе решения лежат следующие составляющие операции. Открывается текстовый файл, и текст из файла переводится в массив фраз (предложений), с сохранением порядка фраз, слов и символов в тексте. Для каждой текущей фразы преобразованного текста запоминается входной индекс (адрес), по которому потом можно выполнить адресацию к этой фразе. Таким образом, рассматриваемая фраза интерпретируется как элемент одномерного строкового массива, которому сопоставлен массив входных индексов. По задаваемым условиям поиска массиву таких фраз ставится в однозначное соответствие числовой массив, элементы которого задаются аналогично $r[i]$ из (9), с тем усложнением, что маска поиска содержит несколько символов на различных позициях различных составляющих слов одной фразы и, при необходимости, нескольких фраз. При рассматриваемом усложнении условий поиска входному массиву фраз сопоставляется числовой массив путем суммирования абсолютных величин разностей asc-кода символов входного массива и символа «маски» поиска с учетом соответствия позиций по индексам строковых элементов. Поиск осуществляется с помощью сортировки и оператора локализации как поиск нулевых значений элементов сопоставленного одномерного числового массива. Следует заметить, что имеет место взаимно однозначное индексное соответствие между искомыми фразами и нулевыми элементами сопоставленного числового массива.

В дополнительное условие поиска можно включить априори задаваемые расстояния между фразами, словами и символами, которые определяются по соответственным входным индексам, хранимым в массиве *e*. При этом можно организовать иерархию таких условий, вложенных друг в друга, применяя их заново к уже найденным фразам. Очевидный результат применения таких иерархических условий – все более усложняющийся набор условий поиска и все более точное местоположение искомых фрагментов текста. При этом число вложений, по построению схемы, априори не ограничено. В частности, с помощью такой схемы можно указывать границы положения искомой комбинации фраз на странице текста, а также ограничивать положение страниц с искомыми фразами.

Отметим дополнительно, что по условию локализации минимумов выделяются цепочки повторяющихся текстовых фрагментов на основе следующей модификации оператора локализации:

```
{sort(nn0,c,e);}
k:=1; WHILE k<= nn0 DO BEGIN FOR r := 2 TO k-1 DO
IF ABS(e[k]-e[k-r]) <= eps0/h THEN GOTO 22; xk:= x0+e[k]*h;
22: k:=k+1 END;
```

Поиск текстовых файлов. Изложенная схема поиска текстовых фрагментов в заданном файле переносится на поиск файлов по заданной маске поиска среди группы текстовых файлов. Это достигается следующим образом. Первоначально задается строковый массив, каждая строка которого – имя исследуемого файла. Задается условие рассмотренного вида, по которому должен идентифицироваться искомый файл. Файл идентифицируется, если комбинация содержащихся в нем фраз удовлетворяет заданному условию. При этом на список имен файлов могут накладываться ограничения, идентичные ограничениям на строковые элементы.

Проводился программный эксперимент, в ходе которого были найдены все файлы, удовлетворяющие, например, следующим условиям поиска. Искомые фразы должны были располагаться внутри файлового текста в промежутке между 4-й и 13-й фразами; при этом маска определялась тем, что на 2 позиции искомой фразы должно находиться «о», на 11-й позиции – пробел и на 20-й – «м». При реализации на языке Object Pascal в среде Delphi на вход программы подавались 12000 текстовых файлов. В результате выполнения программы поиска формировался список всех файлов, достоверно удовлетворяющих заданным условиям.

В рассмотренных выше схемах, операторы локализации экстремумов не являются существенно необходимыми, поскольку в результате сортировки нули идентифицируются в начале отсортированного массива по их входным индексам из массива *e*. Однако, именно на основе этих операторов можно сформировать меры сходства и отличия искомого текстового фрагмента от эталонного значения. Например, если искомый фрагмент отличается от эталонного на один заключительный символ, первый в алфавитном порядке, этому фрагменту можно поставить в соответствие число 1. При различии в следующем по алфавиту символе фрагменту можно сопоставить число 2 и т.д. Совпадение с эталоном оценивается как число 0. Полное отсутствие допустимых отличий можно оценить числом, заведомо большим выбранных весов. Если к массиву сопоставленных чисел, появившихся после предварительного просмотра, применить оператор локализации экстремумов, то выявятся минимальные, максимальные и промежуточные отличия от эталонов. В более общем случае, с помощью функций строковых преобразований весовые отличия можно ввести для любого фрагмента слова в его начале, середине и т.д. При задании весов, можно учесть позицию, на которой обнаружено отличие (от начала до конца слова). То же можно сделать для отдельных текстовых фрагментов. Иными словами все возможные отличия от эталонного значения можно перенумеровать или им сопоставить пронумерованную последовательность весов. Последующая обработка с помощью сортировки и локализации экстремумов автоматически выявит все сходные и различные фрагменты с указанием меры сходства и различия. Здесь требуется оговорка, что мера сходства и

отличия в содержательном смысле формируется программистом. Ясно, что в такой схеме не является обязательным алфавитный порядок отличий, он может быть лексикографическим, либо определяемым в специальном смысле с помощью функций строковых преобразований как наличие (отсутствие) отклонения от эталона заданного вида с заданным местоположением. При данном подходе к поиску оператор локализации является инструментом оценки меры сходства и отклонения. Они дают возможность совместить схемы поиска с алгоритмами распознавания текстовых особенностей.

Изложенный подход дает возможность сформировать и распознавать систему признаков фрагментов текста по экстремальным особенностям числовых последовательностей, элементами которых являются идентификаторы символов и сочетаний символов, а также слов и сочетаний слов текстовых фрагментов. Позволяя находить в текстовом файле слова, последовательности слов или фраз, находящиеся на заданном расстоянии друг от друга и удовлетворяющие при этом дополнительным ограничениям, конструкция схемы поиска дает дополнительную возможность формировать условия поиска в виде математических зависимостей. Например, можно потребовать, чтобы числовые идентификаторы искомых фрагментов удовлетворяли некоторым уравнениям или неравенствам заданного вида, соответствовали рекуррентным соотношениям кодов Фибоначчи, Хемминга и др. Эта возможность опирается на тот факт, что схема поиска в тексте сводится к алгоритму поиска нулей и экстремумов числовой последовательности.

Необходимо отметить, что охарактеризованная схема поиска обладает устойчивостью и параллелизмом в той мере, в какой устойчива и параллельна сортировка. Эти качества и сведение поиска к числовой обработке составляют основные отличия данной схемы от известных схем поиска / 3, 4 /. Кроме того, поиск естественно разделяется на параллельный поиск в составляющих текст (набор файлов) отдельных фрагментах.

С точностью до способа определения кодировки, предложенная схема поиска применима к текстовым файлам различных типов и в различных операционных системах. В частности, данная схема положительно апробировалась для поиска файлов нужного содержания в Internet.

Использование схемы локализации экстремумов на основе сортировки для распознавания изображений

Ниже будет обсуждаться распознавание плоских изображений. С помощью ранее описанной схемы можно выделить экстремальные особенности изображения. Непосредственно экстремальные значения могут рассматриваться как признаки распознаваемого изображения. Однако целесообразно в качестве признаков рассматривать отношения величин экстремумов, например, контура изображения, расположенных в некотором заданном порядке. К числу признаков при этом необходимо отнести само количество максимальных и минимальных значений, определяемых при априори заданном значении диаметра области локализации экстремума. Далее к числу признаков целесообразно относить расстояния между, например, соседними экстремумами, при этом такое расстояние легко определяется по входным индексам локализованных экстремумов. Наконец можно перейти к вторичным признакам, которые формируются из экстремальных элементов числовой последовательности, предварительно образованной пропорциями и другими соотношениями упорядоченных экстремальных значений. При реализации этих общих соображений необходимо априори выделить некоторые классы распознаваемых изображений. Условно предметные области для распознавания можно разделить на три категории: 1) двуцветные изображения с одним тонким контуром; 2) цветные последовательные изображения; 3) двуцветные изображения сложной конфигурации.

В первом случае в качестве предметной области могут выступать графики функций, простейшие геометрические фигуры на плоскости и т.д. В качестве входного массива для сортировки и последующей локализации экстремумов в данном случае рассматривается последовательность ординат точек контура, абсциссы которых соответствуют узлам равномерной сетки, с некоторыми параметрами, изменяющимися в зависимости от исследуемой области. На выходе схемы выделяются все экстремальные элементы входной последовательности. Затем производится их классификация в соответствии с системой признаков, построенной для данной предметной области и для данного класса изображений внутри этой области. Анализ полученных данных позволяет распознавать математические особенности контура, например, расстояния между экстремумами соответствующих типов могут говорить о периодичности или непериодичности изображения, наличие повторяющихся значений в последовательности конечных разностей первого порядка позволяет выделять прямолинейные участки, аналогичные закономерности для конечных разностей второго и выше порядков позволяют выделять параболические, кубические и другие характерные участки контура изображения. По таким особенностям можно построить координатную привязку для перевода фигуры в каноническое положение. На основе охарактеризованного подхода удается устойчиво распознавать геометрические фигуры, отдельные рукопечатные, рукописные символы, идентифицировать кисть руки.

Пример предметной области для второго случая – спектрография. Оптические спектры веществ представляют собой наборы цветных линий. Здесь в качестве входных последовательностей числовые значения цветов точек на равномерной сетке. Цвета разлагаются по RGB компонентам. В соответствие этим компонентам из исходной последовательности формируются три новые последовательности, каждая из которых поступает по отдельности поступает на вход схемы сортировки и локализации экстремальных элементов. Экстремальные элементы при заданном диаметре области локализации в таких последовательностях соответствуют всплескам или падениям интенсивности соответствующего цвета на изображении. Набор экстремальных особенностей и

их математических соотношений составляет систему признаков. Если распознаются полные спектральные изображения, то достаточно хранить только набор признаков для каждого спектра. Эта же схема позволяет производить поиск фрагмента по всем спектральных изображениям, если они идентифицируются не системой признаков, а исходными числовыми последовательностями. В данном случае, входная последовательность составляется из сумм абсолютных разностей цвета линии в маске и на изображении по каждой из компонент. Нулевое значение в полученной последовательности укажет на вхождение линии спектра из маски поиска в изображение.

В качестве третьей предметной области может выступать распознавание графически представленного текста, а также фигур со сложным контуром / 12 / и сложной цветовой гаммой изображения. В этом случае возникает задача построения способа предварительной обработки изображения с той целью, чтобы на выходе такой обработки получить числовую последовательность, которая затем обрабатывается по описанной выше схеме сортировки и локализации экстремальных элементов. Следует отметить, что для каждого класса изображений внутри данной области требуется, как правило, свой особый прием построения идентифицирующей числовой последовательности. Например, для распознавания рукопечатного символа достаточно выполнить считывание координат контура по четырем взаимно перпендикулярным направлениям и по отдельности обработать экстремальные элементы каждой из таких четырех последовательностей. В процессе практической реализации этого подхода удается получить систему признаков, по которой изображение устойчиво идентифицируется в условиях непропорционального растяжения или сжатия изображения, искажения и разрывов контура при условии достаточной малости этих разрывов (эти и другие мелкие искажения поглощаются размером области локализации экстремума). Следует оговориться, что данное утверждение непосредственно относится только к контурно представимым изображениям. В случае сложной цветовой гаммы изображения, например, изображения гидроакустических сигналов, целесообразно осуществить предварительный переход к графическому представлению амплитудных значений сигналов в декартовых координатах. По полученному графику можно определить все локальные экстремумы и затем идентифицировать объект путем сопоставления экстремальных значений и расстояний между точками экстремумов, выражаемых в индексах входных элементов соответствующей графику числовой последовательности. Существенно отметить, что сходной схемой можно воспользоваться для установления диагноза по графикам биометрических данных. При этом искажения сигналов в графическом представлении в значительной мере поглощаются областью локализации экстремумов, а расстояния между таковыми и их изменения идентифицируется по индексам элементов входных последовательностей. В результате возникает возможность обработки сигналов без предварительного математического сглаживания. Это увеличивает достоверность распознавания в виду отсутствия искусственных искажений входной информации.

Необходимо отметить, что использование схемы локализации экстремумов на основе сортировки при распознавании изображений всегда дает возможность выделить наиболее контрастные отличительные особенности рядом расположенных сходных фрагментов и на этой основе выделить существенный признак фрагмента. Данной особенностью и по построению предложенный метод отличается от известных / 13, 14 / подходов к распознаванию в данной области.

Отметим в заключение, что изложенный метод в целом характеризуется единообразной применимостью в различных актуальных аспектах – от приближенного решения алгебраических уравнений высших степеней до поиска и распознавания. В математических приложениях метод отличается высокой точностью за счет фактического отсутсвия иных вычислительных операций, кроме операций сравнения. При поиске и распознавании он позволяет учесть отличительные признаки экстремального характера в локальной области.

Литература

1. *Ромм Я.Е.* Параллельная сортировка слиянием по матрицам сравнений. I // Кибернетика и системный анализ. – 1994. – № 5. – С.3-23.
2. *Ромм Я.Е.* Параллельная сортировка слиянием по матрицам сравнений. II // Кибернетика и системный анализ. – 1995. – № 4. – С.13-37.
3. *Кнут Д.* Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. – М.: Мир, 1978. – 844 с.
4. *Вирт Н.* Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.
5. *Ромм Я.Е.* Метод вычисления нулей и экстремумов функций на основе сортировки с приложением к поиску и распознаванию. I // Кибернетика и системный анализ. – 2001. – № 4. – С.142-159.
6. *Ромм Я.Е.* Метод вычисления нулей и экстремумов функций на основе сортировки с приложением к поиску и распознаванию. II // Кибернетика и системный анализ. – 2001. – № 5. – С.81-101.
7. *Маркушевич А.И., Маркушевич Л.А.* Введение в теорию аналитических функций. – М.: Просвещение, 1997. – 320 с.
8. *Ромм Я.Е.* Схемы устойчивого вычисления нулей многочленов на основе сортировки с приложением к распознаванию. I / ТГПИ. – Таганрог, 2002. – 49 с. Деп. В ВИНИТИ 31. 12. 2002, № 2320-В2002.
9. *Ромм Я.Е.* Схемы устойчивого вычисления нулей многочленов на основе сортировки с приложением к распознаванию. II / ТГПИ. – Таганрог, 2002. – 24 с. Деп. В ВИНИТИ 31. 12. 2002, № 2321-В2002.
10. *Ромм Я.Е., Соловьева И.А.* Метод вычисления нулей и полюсов функций одной комплексной переменной с учетом их кратности на основе сортировки. / ТГПИ. – Таганрог, 2003. – 27 с. Деп. В ВИНИТИ 04. 08. 2003, № 1525-В2003.
11. *Ромм Я.Е., Белоконова С.С.* Схема поиска на основе сортировки и локализации экстремальных элементов. / ТГПИ. – Таганрог, 2003. – 31 с. Деп. В ВИНИТИ 12.03.2003, № 436-В2003.
12. *Гуревич М.Ю.* Распознавание плоских двуцветных изображений на основе адресной сортировки. // Известия ТРТУ. Тематический выпуск: Интеллектуальные САПР / Материалы Международной научно-технической конференции «Интеллектуальные САПР». Таганрог: ТРТУ, 2001. – №4(22) – С.261-267.
13. Журавлев Ю.И., Гуревич И.Б., Распознавание образов и распознавание изображений. – В кн.: Распознавание. Классификация. Прогноз. Математические методы и их применение. Вып.2. – М.: Наука, 1989. – С. 5-72.
14. Ronald Duncan A., Rerret David I. Manipulating facial appearance trough shape and color // IEEE Comput. Graph. and Appl. – 1995, – 15, №5. – pp. 70-76.