ного транспорта. Известия вузов, Северо-Кавказский регион. – Ростов-на-Дону: РГУ, 2004.
4. *Владимиров В.В. Звягинцев Н.С.* Граждан Д. В. Вычисление синусно-косинусных сочетаний алгоритмами дискретных кватернионных преобразований //Перспективные информационные технологии и интеллектуальные системы №1 (21). – Таганрог: Изд-во ТРТУ, 2005.
5. *Звягинцев Н.С. Граждан Д.В.* Об ортогональности бесконечно малого преобразования вектора в *n*-мерном евклидовом пространстве // Сб. научных трудов. Вып. 12. – Новороссийск: МГА им. адм. Ф.Ф. Ушакова, 2007.
6. *Владимиров В.В. Звягинцев Н.С.* Вычислительные возможности алгоритма трехмерного дискретного вращения вектора //Проблемы водного транспорта. Известия вузов, Северо-Кавказский регион. – Ростов-на-Дону: РГУ, 2004.

Я.Е. Ромм, А.А. Лабинцева

ПОИСК КОРНЕЙ ПОЛИНОМА С ПЕРЕМЕННЫМИ КОМПЛЕКСНЫМИ КОЭФФИЦИЕНТАМИ

В [1, 2] изложен метод применения распараллеливаемой внутренней сортировки по ключу (ниже – сортировки) для локализации и приближенного вычисления нулей функций. В частности, в произвольно фиксированной области метод позволяет программно локализовать и с высокой точностью вычислить все нули многочлена, включая случай их плохой отделенности. Степень многочлена и его коэффициенты могут быть произвольными в границах числового диапазона языка программирования. Метод обладает параллелизмом, имеет практические приложения.

Ниже ставится задача быстрой программной локализации области всех нулей многочлена с одновременным их вычислением без использования априорной информации о границах области. При этом задача относится к случаю, когда коэффициенты многочлена являются переменными величинами и могут иметь комплексные значения.

Решение задачи конструируется на основе сортировки слиянием. Известные схемы последней [3, 4] для этой цели модифицируются. На основе матриц сравнения (МС) [5] при выполнении модифицированного слияния достигается максимальный параллелизм, устойчивость, прямая и обратная адресация к входным и выходным индексам. Представленная ниже сортировка не перемещает ключи, используя перемещение их индексов, не ограничена целой степенью двойки для числа элементов.

Конструируемая сортировка и реализующая ее программная процедура именуются *sort*. Сортировка основана на адресном слиянии двух упорядоченных массивов по МС, пример которой представляет таблица:

	$-\infty$	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 МС. Адрес вставки элемента в выходной массив задается соотношениями

$$c_{i+j} = \begin{cases} b_i, \alpha_{ij} = -1 \wedge \alpha_{i(j+1)} \geq 0, \\ a_j, \alpha_{ij} \geq 0 \wedge \alpha_{(i+1)j} = -1, \end{cases} \quad \alpha_{ij} = \begin{cases} 1, a_j > a_i, \\ 0, a_j = a_i, \\ -1, a_j < a_i, \end{cases} \qquad (1)$$

где *i, j* – произвольно взятые индексы при ограничениях $n \geq i \geq 0 \leq j \leq m+1$; *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 сравнений (МС); при этом *a* располагается на входе таблицы, задающей МС, горизонтально сверху, *b* – вертикально слева. По входным индексам *i*, *j* на основе данных соотношений однозначно указывается выходной индекс *l* и обратно, – с учетом принадлежности элемента конкретно одному из двух входных массивов. Эта обратная адресация сохраняется в целом для сортировки слиянием. Сортировка организуется на основе классической схемы [3] вначале «слиянием» каждой пары элементов с запоминанием обратных адресов, затем, аналогично, последовательным слиянием упорядоченных пар, упорядоченных четверок и т. д. При этом сохраняется взаимный порядок сливаемых пар и массивов внутри пары. Программа сортировки представлена процедурой *sort*:

```
CONST n00=1024*16-1; nn=n00+round(n00/2)+1;
TYPE
vect1=ARRAY[0..3*nn] OF extended; vect2=ARRAY[0..3*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] OF longint;
VAR ab: integer;  i,j,k,l,m,r,nm,p,n: longint; e1, e2: vecc;
BEGIN
```

```
p:= trunc(ln(nn0)/ln(2)); IF p<> ln(nn0)/ln(2) THEN p:=p+1;
n:=round(exp(p*ln(2)));
FOR l := 1 TO n DO IF l<=nn0 THEN e[l] := l ELSE ab:=1;
FOR r := 1 TO p DO BEGIN  m :=round(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) OR (e[k * m + l]>nn0) THEN ab:= 1
ELSE e1[l] := e[k * m + l];
IF (k * m + m DIV 2+ l > nn0) OR (e[k * m + m DIV 2+ l]>nn0) THEN
ab:=1
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 (e[k * m + i]>nn0)
OR (k * m + m DIV 2 + j > nn0-1) OR (e[k * m + m DIV 2+ j]>nn0)
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
IF (e2[j + 1] > nn0) OR (e1[i]> nn0) THEN ab:=1 ELSE
BEGIN IF c[e2[j + 1]] = c[e1[i]] THEN ab:=0;
IF c[e2[j + 1]] > c[e1[i]] THEN ab:=1;
IF c[e2[j + 1]] < c[e1[i]] 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 в данной модификации произвольно при условии $nn0 \Leftarrow n0$. Освобождение от степени двойки для значения *nn0* достигается путем имитации дописывания, начиная с $nn0 + 1$, до ближайшей к *nn0* степени двойки возрастающей последовательности элементов входного массива, заведомо больших, чем сортируемые. Имитация реализуется положительными значениями (ab:=1) всех тех элементов каждой из МС, которые соответствуют номерам $>nn0$. Сортировка не перемещает ключи, вызывая их на момент сравнения по обратным адресам входных индексов. При этом сами обратные адреса перемещаются, имитируя перемещение сортируемых элементов. Окончательные значения обратных адресов на выходе сортировки располагаются в массиве *e* в соответствии порядку отсортированных элементов.

К выходу сортировки подсоединяется условный оператор, локализующий после ее выполнения все минимумы среди значений входной последовательности. Это условие состоит в следующем. Фиксируется $\varepsilon > 0$, значение которого выбирается произвольно при ограничении, что оно меньше половины расстояния между произвольно взятыми соседними минимумами. Условие локализации всех минимальных элементов примет вид неравенства

$$\left| e[k-l] - e[k] \right| > \varepsilon / h, \quad l = 1, 2, \ldots, k-1, \qquad (2)$$

где *e[k]* – элемент массива входных индексов на выходе сортировки. Смысл условия (2) в том, что в ε-окрестности текущего индекса *e[k]* нет входного индекса элемента отсортированного массива, такого, который бы превосходил по значению элемент с входным индексом *e[k]*. Программная реализация (2) имеет вид

```
k:=1;   while k<= n 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;
22:   k:=k+1; end;
```

В этом программном фрагменте *n* – число узлов, совпадающее с числом сортируемых элементов, *h* – шаг равномерной сетки, *eps0* – константа, равная радиусу *eps0*-окрестности текущего узла. Эта константа выбирается априори таким образом, чтобы не превысить половины расстояния между ближайшими друг к другу нулями многочлена.

Для локализации нулей функции достаточно на вход сортировки подать абсолютные величины значений функции

$$a(i) = \left| f(x_{i-1}) \right|, \quad i = 1, 2, \ldots, N \qquad (3)$$

и искать минимумы описанным выше способом.

На этой основе можно программно локализовать и с высокой точностью вычислить все нули многочлена [1, 2]. В качестве средства формального описания алгоритмов используется язык Object Pascal, их реализация дана в среде Delphi 7.

Пусть дан произвольный многочлен *n*-й степени $P_n(z)$ с комплексными коэффициентами от комплексной переменной *z*, где $z = x + I\,y$, $I = \sqrt{-1}$.

В плоских декартовых координатах вводится равномерная прямоугольная сетка, включающая область, которой принадлежат все нули. Сетка строится с постоянным шагом *h* по направлениям осей *OX* и *OY*:

$$x_\ell = x_0 + \ell h, \quad \ell = 0, 1, \ldots, N_x; \quad y_\ell = y_0 + \ell h, \quad \ell = 0, 1, \ldots, N_y. \qquad (4)$$

Следующим этапом реализации вычисления комплексных нулей многочлена является построение функции двух действительных переменных *f(x,y)*. Данная функция представлена не разложением многочлена на множители, как это было в [1, 2], а с помощью схемы Горнера, программная реализация которой имеет вид

```
FUNCTION func (VAR x,y: extended; var bdv, bmv: vect3): extended;
VAR i1: 0..n1; p1,p2,pp1,pp2,d1,d2,d3,d4:extended;
BEGIN p1:=bdv[n1];p2:=bmv[n1];
FOR i1:=1 TO n1 DO
begin
um(p1,p2,x,y,d1,d2);                    pp1:=bdv[n1-i1];pp2:=bmv[n1-
i1];sum(pp1,pp2,d1,d2,d3,d4);
p1:=d3; p2:=d4;
end;
func:=sqr(p1)+sqr(p2);
END;
```

Процедуры um и sum, используемые выше, реализуют соответственно умножение и сложение двух комплексных чисел, а значения функции *f(x,y)* берутся во всех узлах сетки и среди них ищутся все возможные минимумы. На основе принципа максимума модуля [6] все минимумы *f(x,y)* достигаются в каждом из нулей многочлена $P_n(z)$ и других минимумов эта функция не имеет. По непрерывности *f(x,y)*, в достаточно малой окрестности каждого из нулей, при достаточной малости шага сетки, значения *f(x,y)*в узлах из этой окрестности будут меньшими, чем значения *f(x,y)*вне этой окрестности. Эти значения можно принять за приближения к искомым нулям. Излагаемая схема сужает шаг сетки в окрестности локализованного приближения, при этом использует исключение операторами программы неточных приближений и ненулевых минимумов, возникающих на границах смещаемых прямоугольников.

Данная схема легко программируется. В приводимом ниже примере программа находит все нули многочлена $x^6 - 4x^5 + 5x^4 - x^2 + 4x - 5$ степени *n1*, действительные и мнимые части (в данном случае мнимые части равны нулю) заданы в разделе констант как одномерные массивы *bdv* и *bmv* соответственно.

```
PROGRAM korkomp444Nastya111;
{$APPTYPE CONSOLE}
USES
SysUtils;
LABEL 21,22,23; const n1=6; bd: ARRAY [0..n1] OF extended =(-5,4,-
1,0,5,-4,1);
                                bm: ARRAY [0..n1] OF extended
=(0,0,0,0,0,0,0);
eps=1e-22*1E-33; eps0=0.049 ;h=eps0/30;
n00=4000; mm=64; nn=n00+round(n00/2)+1;
x00=-5; x11=5; y00=-3; y11=3;
TYPE
vect1=ARRAY[0..2*nn+nn] OF extended;
vect2=ARRAY[0..2*nn+nn] OF longint;
vect3=ARRAY[0..n1] OF extended;
VAR     i,j,k,k1,r,ee,ee1,tty,nn0 : longint;
      c,a1: vect1;
      e,e3, e33: vect2;
      x,x0,x1,xk,xk0,xk1,hx,hy,min,eps1,eps11,eps12,eps13: extended;
      y,y0,y1,yk,yk0,yk1, ykk0,aak,bbk,hh: extended;
      bdv, bmv: vect3;
VAR a,b,a11,b11,ca,cb: extended;
PROCEDURE um(VAR a,b,a11,b11: extended; VAR ca,cb: extended);
BEGIN ca:=a*a11-b*b11; cb:=a*b11+b*a11 END;
PROCEDURE sum(VAR a,b,a11,b11: extended; VAR ca,cb: extended);
BEGIN ca:=a+a11; cb:=b+b11 END;
FUNCTION func (VAR x,y: extended; var bdv, bmv: vect3): extended;
VAR i1: 0..n1; p1,p2,pp1,pp2,d1,d2,d3,d4:extended;
```

```
BEGIN p1:=bdv[n1];p2:=bmv[n1];
FOR i1:=1 TO n1 DO
begin
um(p1,p2,x,y,d1,d2);                    pp1:=bdv[n1-i1];pp2:=bmv[n1-
i1];sum(pp1,pp2,d1,d2,d3,d4);
p1:=d3; p2:=d4;
end;
func:=sqr(p1)+sqr(p2);
END;
PROCEDURE minx (VAR x,y,min:extended;VAR ee:integer);
BEGIN
min:=func(x,y,bdv,bmv); ee:=0; FOR i:=1 TO mm DO BEGIN
x:=xk0+i*hx;
IF min > func(x,y,bdv,bmv) THEN BEGIN min:=func(x,y,bdv,bmv);ee:=i
END  END
END;
PROCEDURE miny (VAR x,y,min:extended;VAR ee1:integer);
BEGIN
min:=func(x,y,bdv,bmv); ee1:=0; FOR i:=1 TO tty DO  BEGIN
y:=ykk0+i*hy;
IF min > func(x,y,bdv,bmv) THEN BEGIN
min:=func(x,y,bdv,bmv);ee1:=i END  END
END;
```

Описание процедуры сортировки sort

```
PROCEDURE spuskx( VAR eps1, xk0,xk1,hx,y: extended);
BEGIN
WHILE abs(eps1) > eps DO BEGIN x:=xk0; minx (x,y,min,ee);
eps1:=eps1/2;
xk0:=xk0+ee*hx-eps1;xk1:=xk0+eps1;hx:=abs(2*eps1)/mm END
END;
PROCEDURE spusky(VAR eps11,yk0,yk1,hy,x: extended);
BEGIN
WHILE abs(eps11) > eps DO BEGIN ykk0:=yk0; y:=yk0; tty:=mm;
miny (x,y,min,ee1); eps11:=eps11/2; yk0:=yk0+ee1*hy-eps11;
yk1:=yk0+eps11;
hy:=abs(2*eps11)/mm END
END;
BEGIN
aak:=1e474; bbk:=1e474; x0:=x00; x1:=x11; y0:=y00; y1:=y11;
nn0:=n00-3;
hh:=nn0*h;
FOR i:=0 TO n1 DO bdv[i]:=bd[i];
FOR i:=0 TO n1 DO bmv[i]:=bm[i];
WHILE x0 <= x11+hh DO BEGIN
WHILE y0 <= y11+hh DO BEGIN
FOR r:=1 TO nn0 DO BEGIN x:=x0+r*h;
```

```
        ykk0:=y0; y:=y0; tty:=n00;hy:=h; miny (x,y,min,ee1); a1[r]:=min END;
        sort( nn0, a1, e3); k:=1;  WHILE k<= nn0 DO  BEGIN
        FOR r := 1 TO k-1 DO IF abs(e3[k]-e3[k-r]) <= eps0/h THEN GOTO 23;
        xk:= x0+E3[K]*h;
        FOR r:=1 TO nn0 DO  BEGIN y:=y0+r*h; a1[r]:=func(xk,y,bdv,bmv)
END;
        sort( nn0, a1, e33); k1:=1; WHILE k1<= nn0 DO  BEGIN
        FOR r := 1 TO k1-1 DO IF abs(e33[k1]-e33[k1-r]) <=eps0/h THEN
GOTO 22;
        yk:= y0+E33[K1]*h; eps1:=eps0; eps11:=eps0;
        xk0:=xk-eps1; xk1:=xk+eps1; hx:=abs(2*eps1)/mm; y:=yk;
        spuskx(eps1,xk0,xk1,hx,y);
        yk0:=yk-eps11;          yk1:=yk+eps11;          hy:=abs(2*eps11)/mm;
x:=xk0+ee*hx+eps1;
        spusky( eps11,yk0,yk1,hy,x); eps12:=eps0/2;
        xk0:=x-eps12;           xk1:=x+eps12;           hx:=abs(2*eps12)/mm;
y:=yk0+ee1*hy+eps11;
        spuskx(  eps12, xk0,xk1,hx,y); eps13:=eps0/2;
        yk0:=yk0+ee1*hy-eps13; yk1:=yk0+2*eps13; hy:=abs(2*eps13)/mm;
        x:=xk0+ee*hx+eps12; spusky( eps13,yk0,yk1,hy,x);
        IF func(xk,yk,bdv,bmv)= 0 THEN begin x:=xk; yk0:=yk; GOTO 21 end;
        IF  abs(func(x,yk0,bdv,bmv)/func(xk,yk,bdv,bmv)) > 0.01 THEN GOTO
22;
        21: IF (abs(aak-x) < eps) AND (abs(bbk-yk0) < eps) THEN GOTO 22;
        IF abs( func(x,yk0,bdv,bmv))>1e-4 THEN GOTO 22; writeln ('    ', x:30,'
');
        writeln ('      ', yk0:30,'      ', func(x,yk0,bdv,bmv)); writeln; aak:=x;
bbk:=yk0;
        22: k1:=k1+1  END;
        23: k:=k+1  END; y0:=y0+hh  END; x0:=x0+hh; y0:=y00  END; readln;
        END.
```

Результат работы программы (приближенные комплексные значения нулей многочлена в левой колонке отделены парами: верхний элемент пары – действительная, нижний – мнимая части приближения нуля; в правой колонке – приближенное значение многочлена):

```
 1.00000000000000000E+0000
-6.21400594831414559E-0056                2.47128767524375E-0109

 3.38813178901720136E-0020
 1.00000000000000000E+0000                 2.16777688680938E-0037

 8.92433913227125115E-0018
-1.00000000000000000E+0000                 4.05294135291744E-0032

-1.00000000000000000E+0000
-6.33193397655751137E-0056                 6.41494206135735E-0108
```

```
 2.00000000000000000E+0000
 1.00000000000000000E+0000                0.00000000000000E+0000

 2.00000000000000000E+0000
-1.00000000000000000E+0000                2.44502824971036E-0034
```

В результате работы программы найдены все нули многочлена $x^6 - 4x^5 + 5x^4 - x^2 + 4x - 5$ в прямоугольнике $[5, -5] \times [3, -3]$, которые вычислены с точностью формата выходных данных. Прямоугольник можно существенно увеличить без потери точности, но с замедлением работы программы. Спуск к каждому локализованному приближению нуля выполняется двукратно с целью повысить точность приближения, которая иначе не достигает представленного уровня. Переход к метке 22 по условию перед меткой 21 исключает возможность ошибочного принятия минимумов на концах текущего шага длины *hh* за нули многочлена.

Изменения для общего случая сводятся к заданию соответственной степени многочлена и его коэффициентов в разделе констант.

По построению схемы и на основе численных экспериментов можно утверждать, что с помощью данной программы (с точностью до значений именованных констант) удается устойчиво локализовать и вычислить все комплексные нули произвольного многочлена с комплексными коэффициентами в произвольно фиксированной прямоугольной области. При этом не требуется учитывать математические ограничения на вид комплексных коэффициентов многочлена, так что они могут вырабатываться динамически. Утверждение предполагает, что не нарушается числовой диапазон языка программирования. В общем случае без фиксации области корни могут быть найдены по соответственным схемам из [1, 2].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Ромм Я.Е.* Локализация и устойчивое вычисление нулей многочлена на основе сортировки. I // Кибернетика и системный анализ. – 2007. – № 1. – С. 165 – 183.
2. *Ромм Я.Е.* Локализация и устойчивое вычисление нулей многочлена на основе сортировки. II // Кибернетика и системный анализ. – 2007. – № 2. – С. 161 – 175.
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.* Параллельная сортировка слиянием по матрицам сравнений. I // Кибернетика и системный анализ. – 1994. – № 5. – С. 3 – 23.
6. *Маркушевич А.И., Маркушевич Л.А.* Введение в теорию аналитических функций. – М.: Просвещение, 1997. – 320 с.