УДК 519.6

И. В. Мельник, д-р техн.наук
Национальный технический университет Украины
«Киевский политехнический институт»
(Украина, 03056, Киев, пр. Победы, 37, корп. 12, 2203
тел. (044) 2419672, (044) 4549505, E-mail: imelnik@edd.ntu-kpi.kiev.ua)

Анализ возможностей использования матричных макроопераций системы MatLab при решении прикладных задач

(Статью представил чл.-кор. НАН Украины В. В. Васильев)

На примерах решения практических задач программирования проанализированы возможности использования векторных и матричных макроопераций для реализации классических и оригинальных алгоритмов. Для объяснения основных особенностей этих операций введены новые понятия арифметико-логического выражения, рекуррентного арифметико-логического выражения и вектора-функции. Показано, что при решении задач среднего уровня сложности предложенный подход может стать альтернативой современным методам и средствам структурного программирования.

На прикладах розв'язку практичних задач програмування проаналізовано можливості використання векторних та матричних макрооперацій для реалізації класичних та оригінальних алгоритмів. Для пояснення головних особливостей цих операцій введено нові поняття арифметико-логічного виразу, рекурентного арифметико-логічного виразу та вектора-функції. Показано, що при розв'язуванні задач середнього рівня складності запропонований підхід може стати альтернативою сучасним методам та засобам структурного програмування.

К л ю ч е в ы е с л о в а: матричные макрооперации, арифметико-логическое выражение, рекуррентные вычисления, рекуррентное арифметико-логическое выражение, вектор-функция.

Математические САПР находят все более широкое применение при реализации инженерных и научно-технических расчетов, и число сторонников их использования увеличивается с каждым годом. В последнее время среди существующих математических САПР особое место занимает система MatLab, ориентированная на использование матричных и векторных макроопераций при обработке числовых структур данных [1,2]. Основными преимуществами этой САПР является открытый программный код для большинства функций и принцип модульной архитектуры. В языке

программирования системы MatLab реализованы различные современные подходы и парадигмы программирования, включая структурное, объектно-ориентированное, логическое, функциональное и модульное программирование [1,2].

Однако сторонники использования системы MatLab в инженерной и научной деятельности считают, что в системах с развитым набором матричных макроопераций можно вообще отказаться от использования традиционных методов и средств структурного программирования. При этом речь идет в основном об операторах цикла и не упоминаются ветвящиеся вычислительные процессы и условный оператор [2].

В результате анализа литературных источников можно сделать вывод о том, что для определения потенциальных возможностей использования матричных и векторных макроопераций при реализации классических и оригинальных алгоритмов необходимо проведение дополнительных системных исследований по изучению обобщенных свойств этих операций и возможностей их практического применения, включая принципиальные ограничения.

Рассмотрим возможность реализации стандартных вычислительных алгоритмов через матричные макрооперации. В качестве тестовых классических задач будем осуществлять поиск минимальных и максимальных значений и сортировку элементов вектора. Для простоты описания особенностей реализации вычислительных алгоритмов через матричные макрооперации введем такие базовые понятия, как арифметико-логическое выражение, рекуррентное арифметико-логическое выражение и вектор-функция.

Существующие векторные и матричные макрооперации. Наиболее интересным достижением разработчиков системы MatLab в области использования матричных макроопераций является поэлементная обработка упорядоченных числовых структур, включая соответствующие операции над матрицами и векторами [1,2]. В качестве примера рассмотрим две программы для вычисления суммы квадратов элементов двух векторов, написанные с использованием поэлементных операций и средствами структурного программирования.

Пример 1. ***Поэлементные операции***

```
»V1=[1,2,7,3,4];
»V2=[4,2,7,3,9];
»R= V1.^2+V2.^2
»R =
     17     8    98    18    97
»
```

Структурное программирование

```
V1=[1,2,7,3,4];
V2=[4,2,7,3,9];
for ii=1:length(V1)
  R(ii)=V1(ii)^2+V2(ii)^2;
end;
```

Более сложным примером эффективного использования матричных макроопераций в MatLab является множественная индексация [2]. Рассмотрим возможность ее применения на примере обращения к совокупности элементов вектора. Допустим, необходимо выделить элементы вектора с пятого по восьмой.

Пример 2.

```
»V1=[9,8,7,6,5,4,3,2,1];
»ii=[5:8];
»V2=V1(ii)
»V2 =
     5   4   3   2
»
```

Таким образом, поэлементные операции и множественная индексация могут использоваться как декларативные операторы языка высокого уровня вместо операторов цикла. Проанализируем возможности использования этих средств программирования при реализации более сложных вычислений.

Арифметико-логическое выражение. Рассмотрим ветвящийся процесс на простом примере вычисления кусочно-заданных функций. При использовании средств структурного программирования вычислять значения таких функций для различных значений аргумента можно только в случае использования условного оператора внутри оператора цикла. Например, для вычисления значений функций

$$f(x)=\begin{cases} f_1(x), & x>0, \\ f_2(x), & x\le 0, \end{cases} \quad f_1(x)=\frac{e^x}{x^2+1}, \quad f_2(x)=\frac{e^x-x^2}{x^2+1} \qquad (1)$$

на отрезке [– 5; 5] с шагом 0,01 необходимо написать такой фрагмент программы.

Пример 3.

```
x=-5:0.01:5;
for ii=1:length(x)
  if (x(ii)<=0) z(ii)=(exp(x(ii))-x(ii)^2)/(1+x(ii)^2); …
       else z(i)=(exp(x(i)))/(1+x(i)^2); end;
```

Поскольку в языке программирования системы MatLab при использовании условного оператора не предполагается поэлементная обработка упорядоченных структур числовых данных, все ветвящиеся процессы реализуются средствами структурного программирования.

Возможной альтернативой использованию условного оператора являются арифметико-логические выражения, представляющие собой комбинацию арифметических и логических функций вида [3]

$$F(x) = F_1(x)\,L_1(x) + F_2(x)\,L_2(x) + F_3(x)\,L_3(x) + ... + F_n(x)\,L_n(x), \qquad (2)$$

где $F_1(x), F_2(x), F_3(x), ..., F_n(x)$ — арифметические функции; $L_1(x), L_2(x), ...$ $..., L_n(x)$ — логические функции. Если для вычисления (1) использовать выражение (2), переписав кусочно-заданную функцию в виде

$$F(x) = (x > 0)\frac{e^x}{x^2+1} + (x \le 0)\frac{e^x - x^2}{x^2+1},$$

то код программы, по сравнению с примером 3, существенно упрощается.

Пример 4.

```
x=-5:0.01:5;
f=(x>0).*(exp(x)./(x.^2+1))+(x<=0).*((exp(x)-x.^2)./(x.^2+1));
```

Если функция задана не на двух, а на большем числе отрезков, то соответствующие логические выражения $L_n(x)$ записываются через операцию «and» (&).

Результаты проверки эффективности программного кода для примеров 3 и 4 свидетельствуют о том, что вариант с оператором цикла более эффективен только при малом числе элементов вектора **x**, в основном при length $(x) < 10^4$, в то время как для больших числовых структур более эффективным является второй вариант реализации.

На основе проведенного анализа можно сделать следующий вывод.

Ветвящиеся вычислительные процессы в рамках матричного программирования реализуются с помощью арифметико-логических выражений вида (2).

Рекуррентные вычисления. В прикладных задачах вычислительной математики часто используются вычислительные процедуры, основанные на изменяемых переменных цикла. Из них наиболее простыми являются рекуррентные алгоритмы, описываемые соотношениями вида [4]

$$x_1 = c_1;\ x_2 = c_2;\ x_3 = c_3;\ \ldots x_n = c_n;\ x_i = f\,(i, x_{i-1}, x_{i-2}, ..., x_{i-n}), \qquad (3)$$

где $x_1, x_2, \ldots, x_n \ldots x_i$ — значения элементов вектора **x**; $c_1, c_2, \ldots, c_n$ — константы. Вычисление рекуррентных последовательностей с использованием

стандартных векторных макроопераций системы MatLab невозможно. Это видно из следующего примера, в котором сделана попытка вычислить элементы последовательности Фибоначчи [4].

Пример 5.

```
» i=1:10; j=3:10;
» x=i; x(2)=1;
» x(j)=x(j-1)+x(j-2);
» x
x =
     1   1   2   4   7   9  11  13  15  17
»
```

Причиной ошибки является то, что вместо уже вычисленных значений на каждом шаге вычислений используются предыдущие, первоначально заданные значения элементов вектора **x**.

На основе проведенного анализа можно сделать следующий вывод.

Средства множественной индексации системы MatLab не позволяют реализовать рекуррентные вычислительные алгоритмы, а следовательно, набор матричных и векторных макроопераций языка программирования системы является неполным.

В связи с этим в библиотеку стандартных функций системы была добавлена функция recvect, позволяющая реализовывать рекуррентные вычисления через передачу функции $f\ (i, x_{i-1}, x_{i-2}, ..., x_{i-n})$ как параметра в виде строки символов. Эта процедура написана с использованием операторов цикла, однако при ее применении рекуррентные вычисления реализуются без использования циклов.

Приведенная в следующем примере процедура позволяет достичь достаточно высокой эффективности и простоты программного кода.

Пример 6.

```
  for ii=1:n if (ii<=nr) w(ii)=v(ii);
     else
       if (ch==0) w(ii)=eval(ff); end;
       if ((ch==1)&(nr==1)) res=eval(ff); w(ii)=res(2);
         w(ii-1)=res(1); end;
       if ((ch==1)&(nr==2)) res=eval(ff); w(ii)=res(3);
         w(ii-1)=res(2); w(ii-2)=res(1); end;
       if ((ch==1)&(nr==3)) res=eval(ff); w(ii)=res(4);
         w(ii-1)=res(3); w(ii-2)=res(2); w(ii-3)=res(1); end;
       if ((ch==1)&(nr==4)) res=eval(ff); w(ii)=res(5);
         w(ii-1)=res(4); w(ii-2)=res(3); w(ii-3)=res(2);
```

```
        w(ii-4)=res(1); end;
      if ((ch==1)&(nr==5)) res=eval(ff); w(ii)=res(6);
        w(ii-1)=res(5); w(ii-2)=res(4); w(ii-3)=res(3);
        w(ii-4)=res(2); w(ii-5)=res(1); end;
      if (abs(w(ii)-w(ii-1))<el)return; end;
      if (abs(w(ii)-w(ii-1))>eh)return; end;
end;  end;
```

Приведем пример вычисления элементов последовательности Фибоначчи с использованием процедуры recvect.

Пример 7.

```
» ffh='w(ii-1)+w(ii-2)';
» c=recvect(2,10,[1,1],ffh,1e-10,1e10)
c =
    1   1   2   3   5   8   13   21   34   55
»
```

Основные входные параметры, передаваемые процедуре recvect при ее вызове, следующие:

n — степень зависимости элементов в рекуррентной формуле, т. е. положение наиболее отдаленного элемента, используемого при вычислении текущего; ссылка на этот элемент осуществляется через параметр (i – – n), например для ряда Фибоначчи $n = 2$.

N — максимальное число вычисляемых членов ряда, $N > n$, по умолчанию $N = 100$;

V — входной вектор начальных значений, число элементов которого должно соответствовать степени зависимости n, т. е. length (V) = n, например для ряда Фибоначчи это вектор [1,1];

ffh — рекуррентная функция, задаваемая в виде строки символов;

ε_l — минимальная разность между соседними членами ряда при его сходимости;

ε_h — максимальная разность между соседними членами ряда при его расходимости; по умолчанию предполагается $\varepsilon_l = 10^{-10}$, $\varepsilon_h = 10^{10}$.

Далее рассмотрим примеры реализации стандартных и оригинальных алгоритмов с использованием векторных и матричных макроопераций.

Алгоритмы поиска и сортировки. Сначала рассмотрим особенности использования векторных и матричных макроопераций при решении стандартных задач обработки данных. Одной из таких задач является поиск максимального или минимального элементов вектора [4]. В общем случае задачу поиска максимального элемента можно записать в виде следующего

рекуррентного соотношения:

$$\mathbf{x}_{\max}(1)=\mathbf{x}(1);\ \ \mathbf{x}_{\max}(i)=\begin{cases}\mathbf{x}(i),\ \text{если}\ \mathbf{x}(i)\geq\mathbf{x}_{\max}(i-1),\\ \mathbf{x}_{\max}(i-1),\ \text{если}\ \mathbf{x}(i)<\mathbf{x}_{\max}(i-1),\end{cases} \tag{4}$$

где $\mathbf{x}$ — исходный вектор, а $\mathbf{x}_{\max}(i)=\max\limits_{k=1...i}(\mathbf{x}(k))$ — вектор результата. Рекуррентную формулу (4) перепишем в виде арифметико-логического выражения (2):

$$\mathbf{x}_{\max}(1)=\mathbf{x}(1);$$

$$\mathbf{x}_{\max}(i)=(\mathbf{x}(i)\geq\mathbf{x}_{\max}(i-1))\cdot\mathbf{x}(i)+(\mathbf{x}(i)<\mathbf{x}_{\max}(i-1))\cdot\mathbf{x}_{\max}(i-1). \tag{5}$$

Соотношения (5), которые фактически являются рекуррентными, но в то же время соответствуют определению арифметико-логического выражения (2), будем называть рекуррентными арифметико-логическими выражениями. Приведем пример, в котором поиск максимального элемента вектора выполнен с использованием соотношения (5) и реализован через процедуру recvect.

Пример 8.

```
» x=[1,100,-25,20,150,40,-50];
» ffh='v(ii)*(v(ii)>=w(ii-1))+(v(ii)<w(ii-1))*w(ii-1)';
» c=recvect(1,length(x),x,ffh,1e-10,1e10)
c =
   1   100   100   100   150   150   150
»
```

Здесь v — исходный вектор, w — вектор результата, а функция, описываемая соотношением (5), записана в виде строковой переменной ffh, которая является входным параметром для процедуры recvect.

Из примера 8 видно, что векторные и матричные макрооперации являются избыточными по сравнению со средствами структурного программирования с точки зрения экономии вычислительных ресурсов, поскольку для их реализации требуется сохранение всех результатов предыдущих вычислений. Однако такие результаты легко анализировать при отладке программ, целесообразна такая форма представления и при изучении теории алгоритмов.

Более сложным способом реализации рекуррентных вычислений через матричные макрооперации является вычисление каждой следующей строки матрицы через элементы ее предыдущих строк. В общем случае соответствующий рекуррентный алгоритм можно записать в виде

$$\begin{gathered}\mathbf{M}_{<1>}=\mathbf{v}_1,\ \mathbf{M}_{<2>}=\mathbf{v}_2,\ ...,\ \mathbf{M}_{<n>}=\mathbf{v}_n,\\ \mathbf{M}_{<i>}=\mathbf{F}(i,\mathbf{M}_{<i-1>},\mathbf{M}_{<i-2>},\ ...,\ \mathbf{M}_{<i-n>}),\end{gathered} \tag{6}$$

где **v** — вектор; $\mathbf{M}_{<i>}$ — строка матрицы с номером *i*; **F** — вектор-функция, определяющая функции для вычисления каждого элемента строки результирующей матрицы. Такие рекуррентные алгоритмы также составляют основу вычислительной математики и используются не реже, чем стандартные, определяемые соотношением (3). Для реализации рекуррентных алгоритмов, описываемых соотношением (6), получена функция recmat, программный код которой здесь не приведен из-за ограниченного объема статьи.

Рассмотрим более сложный пример, в котором через рекуррентное арифметико-логическое выражение реализован классический алгоритм пузырьковой сортировки элементов вектора по возрастанию и убыванию [4, 5]. Арифметико-логическое выражение для алгоритма сортировки по возрастанию имеет вид

$$\left.\begin{pmatrix} \mathbf{xs}_{i-1} \\ \mathbf{xs}_{i} \end{pmatrix}\right|_{i=2...n} = (\mathbf{x}_i > \mathbf{xs}_{i-1}) \begin{pmatrix} \mathbf{xs}_{i-1} \\ \mathbf{x}_{i} \end{pmatrix} + (\mathbf{x}_i \le \mathbf{xs}_{i-1}) \begin{pmatrix} \mathbf{x}_{i} \\ \mathbf{xs}_{i-1} \end{pmatrix}, \qquad (7)$$

где **x** — исходный вектор; **xs** — вектор результата; *n* — число сортируемых элементов.

Для сортировки по убыванию выражение (7) запишем в виде

$$\left.\begin{pmatrix} \mathbf{xs}_{i-1} \\ \mathbf{xs}_{i} \end{pmatrix}\right|_{i=2...n} = (\mathbf{x}_i > \mathbf{xs}_{i-1}) \begin{pmatrix} \mathbf{x}_{i} \\ \mathbf{xs}_{i-1} \end{pmatrix} + (\mathbf{x}_i \le \mathbf{xs}_{i-1}) \begin{pmatrix} \mathbf{xs}_{i-1} \\ \mathbf{x}_{i} \end{pmatrix}. \qquad (8)$$

Необходимо заметить, что с помощью соотношений (7), (8), в соответствии с алгоритмом пузырьковой сортировки, можно проанализировать только два соседних элемента, поэтому следует использовать итерационную процедуру. Вычисления прекращаются, когда получены две одинаковые строки результирующей матрицы.

Приведем пример программы sortbuble, сортирующей элементы вектора по возрастанию при m=1 или по убыванию при m=-1. Для расчетов по формулам (7), (8) использована процедура recmat. Матрица результата обозначена через M.

Пример 9.

```
function ws=sortbuble (vv,m)
fh1='recvect(1,length(M(ii-1,:)),M(ii-1,:),''';
fh2='([v(ii)>w(ii-1),v(ii)>w(ii-1)].*[w(ii-1),v(ii)])+...
    ([w(ii-1)>=v(ii),w(ii-1)>=v(ii)].*[v(ii),w(ii-1)])';
fh3='',1e-10,1e10)';
fh4='([v(ii)<w(ii-1),v(ii)<w(ii-1)].*[w(ii-1),v(ii)])+...
    ([w(ii-1)<=v(ii),w(ii-1)<=v(ii)].*[v(ii),w(ii-1)])';
if (m==1) fh=strcat(fh1,fh2,fh3); end;
```

```
if (m==-1) fh=strcat(fh1,fh4,fh3);end;
ws=recmat(1,length(vv),vv,fh,1e-10,1e10);
```

В следующем примере приведен результат работы этой программы.

Пример 10.

```
» x=[1,-3,10,-5,50,5,15,10,20];
» sortbuble(x,1)
ans =
    1   -3   10   -5   50    5   15   10   20
   -3    1   -5   10    5   15   10   20   50
   -3   -5    1    5   10   10   15   20   50
   -5   -3    1    5   10   10   15   20   50
   -5   -3    1    5   10   10   15   20   50
»
```

Как показали результаты вычислительных экспериментов, эффективность функции sortbuble ниже, чем встроенной функции MatLab sort. Тем не менее, возможность реализации алгоритмов вложенных циклов позволяет расценивать процедуры recvect и recmat как эффективные средства матричного программирования.

Далее рассмотрим особенности реализации средствами матричного программирования итерационных вычислительных алгоритмов, которые составляют основу численных методов.

Итерационные вычислительные алгоритмы. Рекуррентные вычисления по формулам (3), (6) могут найти широкое применение в вычислительной математике. Например вычисление степенных рядов для заданного вектора аргументов **X** можно реализовать, используя рекуррентную формулу (6):

$$\mathbf{A}_{<1>}=\mathbf{X},\ \ \mathbf{A}_{<n>}=\mathbf{F}\,(\mathbf{X},\mathbf{A}_{<n-1>},n);\ \ R\,(X)=\sum_{i=1}^{n}A_{<n>}. \qquad (9)$$

В следующем примере приведена программа, в которой реализованы рекуррентные вычисления по формуле (9).

Пример 11.

```
function R=powseries(x,n,ff,el)
c1=0; c2=0;d=size(x);
if (d(1)==1) nr=1; fs=strcat('[M(ii-1,:);',ff,']'); end;
if (d(1)==2) nr=2; fs=strcat('[M(ii-2,:);M(ii-1,:);',ff,']');end;
Rm=recmat(nr,n,x,fs,el,1e10);R=sum(Rm);
return;
```

Например для функции $\frac{\sin(x)}{x}$ степенной ряд имеет вид [6]

$$\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + ... + O(x^n) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n+1)!},$$

или в рекуррентной форме —

$$R_1(x) = 1,\ R_2(x) = -\frac{x^2}{6},\ ...,\ R_n(x) = -R_{n-1}(x)\frac{x^2}{2(n-1)(2n-1)};$$

$$S(x) = \sum_{n=1}^{\infty} R_n(x). \qquad (10)$$

В следующем примере приведена последовательность командных строк для вычисления ряда (10).

Пример 12.

```
>> x=-10:0.001:10;
>> xx=ones(1,length(x));
>> xr=[xx;-x.^2/6];
>> S=powseries(xr,1000,'6*M(ii-1,:).*M(2,:)./(2*ii-2)./(2*ii-1)',1e-30);
```

Меняя функцию $\mathbf{F}(\mathbf{X}, \mathbf{A}_{<n-1>}, n)$, можно формировать различные степенные и функциональные ряды, что дает широкие возможности для создания математических библиотек.

С использованием процедур recvect и recmat легко реализовать и итерационные алгоритмы численных методов. Например численное интегрирование функций методом трапеций реализуется с помощью следующего простого рекуррентного соотношения [6, 7]:

$$\mathbf{S}(0) = 0;\ \mathbf{S}(i) = \mathbf{S}(i-1) + \frac{(f(\mathbf{x}_i) + f(\mathbf{x}_{i-1}))(\mathbf{x}_i - \mathbf{x}_{i-1})}{2},$$

где $\mathbf{x}$ — переменная интегрирования; $f(\mathbf{x}_i)$ — значения интегрируемой функции в точках отсчета; $\mathbf{S}$ — вычисляемое значение интеграла.

В следующем примере вычислим значение интеграла $f(x) = \int_0^x e^{-x^2} dx$ на интервале $0 \le x \le 3$ с шагом 10^{-4}.

Пример 13.

```
>> x=0:0.0001:3;
>> f=[0,exp(-x.^2)];
>>S=recvect(1, length(f), f,'(w(ii-1)+((v(ii-1)+v(ii))*0.0001/2)).*(ii~=2)+0.*(ii==2)', 0,1e50);
```

Аналогично, с использованием функции recvect, можно численно решать нелинейные и дифференциальные уравнения [3].

Тестирование функций recvect и recmat проводилось и для более сложных вычислительных алгоритмов, один из которых рассмотрим далее.

Использование методов матричного программирования для решения физических задач. Рассмотрим возможности использования матричных макроопераций для решения практических инженерных задач на примере расчета траекторий заряженных частиц в поле симметричной короткой магнитной линзы. Математическая постановка данной задачи, в общем случае, сводится к системе алгебро-дифференциальных уравнений [8, 9]

$$v=\sqrt{2U_0\eta},\ v_r=v\sin(\alpha),\ v_2=v\cos(\alpha),\ \frac{dr}{dz}=\frac{v_r}{v_z},\ \eta=\frac{e}{m},$$

$$\frac{d^2r}{dz^2}=\frac{erB_{z0}^2}{8mU_0},\quad B_{z0}(z)=\frac{\mu_0 INR^2}{2\sqrt{(z^2+R^2)^3}},\quad \frac{d\theta}{dt}=-\frac{\eta B_{z0}}{2}, \tag{11}$$

$$\frac{dr}{dz}=\frac{dr}{dz}-\frac{d^2r}{dz^2}dz,\ \theta=\theta-\frac{\eta B_{z0}dz}{2v_z},\ r=r+\frac{dr}{dz}dz-\frac{d^2r(dz)^2}{2dz^2},$$

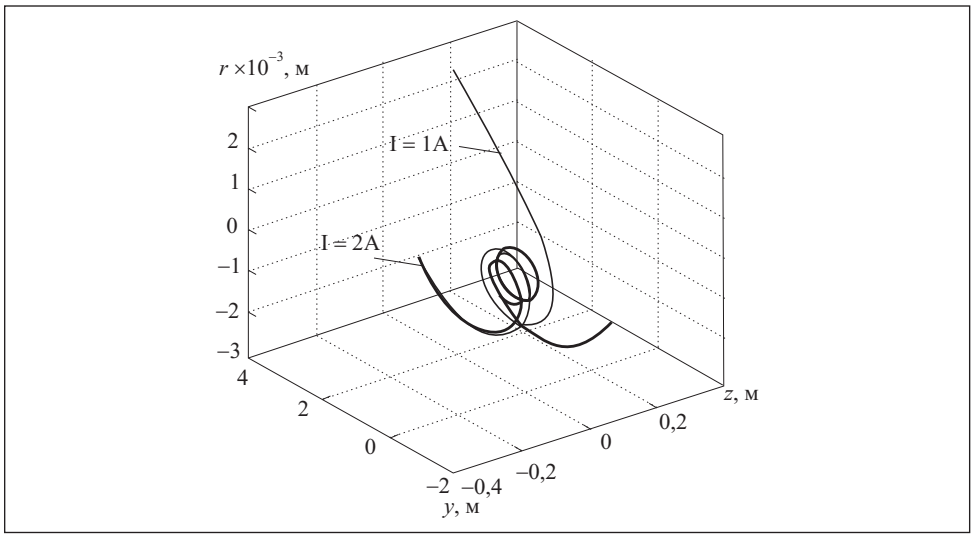
где z и r — продольная и поперечная координаты; B_{z0} — функция распределения магнитного поля вдоль оси z; U_0 — ускоряющее напряжение; I — ток линзы; N — число витков; R — радиус линзы; μ_0 — магнитная постоянная; v — скорость движения частицы; α — угол влета частицы; e — заряд электрона; m — его масса.

Запись алгоритма в виде системы уравнений (11) неудобна, так как не указан порядок использования уравнений, который при решении реальных физических задач может оказывать существенное влияние на сходимость итерационных процедур [10]. Поэтому перепишем систему (11) в итерационной форме:

$z_0,\ B_{z0},\ \left(\frac{dr}{dz}\right)_0,\ \theta_0,\ r_0$ — начальные значения; $z_n=z_{n-1}+dz,\ B_{zn}=$ $=B_z(z_{n-1})$;

$$\left(\frac{dr}{dz}\right)_n=\left(\frac{dr}{dz}\right)_{n-1}-\left(\frac{d^2r}{dz^2}\right)_n;\ \theta_n=\theta_{n-1}-\sqrt{\frac{\eta}{2U_0}}\frac{B_{z0}(z_{n-1})\,dz}{\cos\left(\frac{dr}{dz}\right)_{n-1}};$$

$$r_n=r_{n-1}+\frac{dr}{dz}dz+\frac{d^2r}{2dz^2}(dz)^2, \tag{12}$$



Моделирование траекторий электронов в короткой магнитной линзе с использованием программы, приведенной в примере 14

где n — номер текущей итерации; $n-1$ — номер предыдущей итерации. Тогда совокупность переменных, вычисляемых с помощью итерационных соотношений (12), легко упорядочить в виде строк матрицы. При этом итерационная процедура вычислений сводится к соотношению (6), а результирующая матрица будет иметь вид

$$\begin{bmatrix} z_0 & B_{z0}(z_0) & \left(\frac{dr}{dz}\right)_0 & \theta_0 & r_0 \\ z_1 & B_{z0}(z_1) & \left(\frac{dr}{dz}\right)_1 & \theta_1 & r_1 \\ \dots & \dots & \dots & \dots & \dots \\ z_n & B_{z0}(z_n) & \left(\frac{dr}{dz}\right)_n & \theta_n & r_n \end{bmatrix}. \qquad (13)$$

Представим рекуррентные соотношения (12) в виде вектора-функции, последовательно формирующего строки матрицы (13):

$$\left[z_n, (B_{z0})_n, \left(\frac{dr}{dz}\right)_n, \theta_n, r_n\right] = \left[z_{n-1} + dz,\ \frac{\mu_0 INR^2}{2\sqrt{(z_{n-1}^2 + R^2)^3}},\ \left(\frac{dr}{dz}\right)_{n-1} - \right.$$

$$\left. - \frac{erB_{z0}^2(z_{n-1})\,dz}{8mU_0},\ \theta_{n-1} - \sqrt{\frac{\eta}{2U_0}}\,\frac{B_{z0}(z_{n-1})\,dz}{\cos\left(\frac{dr}{dz}\right)_{n-1}},\ r_{n-1} + \right.$$

$$+dz\left(\frac{dr}{dz}\right)_{n-1}+\frac{erB_{z0}^2(z_{n-1})(dz)^2}{16mU_0}\Bigg]. \qquad (14)$$

Таким образом, решение системы уравнений (12) сводится к вычислению строк рекуррентной матрицы (13) с помощью соотношений (14). Предложенная методика расчета реализована в следующем примере, а результаты в графическом виде представлены на рисунке.

Пример 14.

```
I=1; N=1000; mu0=1.26e-6;R=0.05; teta=1.6e-19/9.1e-31;U0=1000;
IS=num2str(I); mu0S=num2str(mu0); NS=num2str(N);RS=num2str(R);
TetaS=num2str(teta);U0S=num2str(U0);r0=0.001; v=sqrt(2*U0*teta);
dzS='1e-4';a=0.01; vr=v*sin(a); vz=v*cos(a);
vzS=num2str(vz); zS=strcat('M(ii-1,1)+',dzS);
BZS=strcat('((0.5*',RS,'^2*',IS,'*',NS,'*',mu0S,')./',...
      '(sqrt((M(ii-1,1))^2+',RS,'^2))^3)');
d2rdzS=strcat('0.125*',TetaS,'*M(ii-1,5)*(M(ii-1,2))^2/',U0S);
drdzS=strcat('M(ii-1,3)-(',d2rdzS,')*',dzS);
thetaS=strcat('M(ii-1,4)-(0.5*',dzS,'*',TetaS,'*M(ii-1,2))/',vzS);
drS=strcat('M(ii-1,5)+M(ii-1,3)*',dzS,'-',d2rdzS,')*0.5*',dzS,'^2');
Vin=[-0.2,0.5*R^2*I*N*mu0*((-0.2)^2+R^2)^(-1.5),vr/vz,0,r0];
funstr=strcat('[',zS,',',BZS,',',drdzS,',',thetaS,',',drS,']');
Mout=recmat(1,4002,Vin,funstr,1e-10,1e10);
```

По результатам тестовых экспериментов время работы программы, приведенной в примере 14, приблизительно в три раза превышает время работы аналогичной программы, написанной с использованием оператора цикла. Однако простота исходного кода и возможность эффективной отладки такой программы являются ее неоспоримыми преимуществами, а время счета на компьютере класса AMD Duron 800 МГц 256 Мб ОЗУ не превышает одной секунды.

Анализ результатов исследований и рекомендации. Средства матричного программирования при решении реальных задач среднего уровня сложности являются вполне приемлемыми. Даже с учетом сложных операций со строками отсутствие структур делает программу простой и понятной, а использование арифметико-логических выражений вида (2) позволяет легко реализовывать и ветвящиеся вычислительные процессы. Разумеется, реализация разработчиком рекуррентных вычислений в циклах на уровне ядра системы MatLab значительно повысила бы их эффективность, а при простоте соответствующих лингвистических средств упростился бы и программный код. Однако использование функций recvect и

recmat позволяет вводить необходимое число параметров для ограничения проводимых вычислений не только по числу итераций, но и по результатам сходимости или расходимости итерационного процесса.

Было проведено тестирование матричных методов вычислений на более сложных реальных задачах, в частности на задаче транспортировки электронного пучка из низкого в высокий вакуум [8]. В этом случае реализовать матричный подход для всей задачи было сложно вследствие большого числа входных параметров и соответствующей размерности результирующей матрицы. Однако совмещение методов матричного и структурного программирования позволило создать достаточно простой и эффективный программный код.

Методика написания программ средствами матричного программирования может быть формализована следующим образом:

1. Формируется система алгебро-дифференциальных уравнений, аналогичная (11).

2. На ее основе формируется система итерационных уравнений, аналогичная (12).

3. На основе системы итерационных уравнений формируется рекуррентная матрица, аналогичная (13).

4. Для полученной системы итерационных уравнений и рекуррентной матрицы формируется вектор-функция, аналогичная (14).

5. Полученное рекуррентное матричное уравнение, аналогичное (6), решается с использованием процедуры recmat.

Вопрос, связанный с избыточностью используемой оперативной памяти при построении матрицы результатов, может быть решен путем оптимизации алгоритма и удаления строк, ненужных для последующих вычислений.

Выводы. Матричные методы программирования успешно протестированы на ряде стандартных и оригинальных задач, что позволяет с большой достоверностью предполагать возможность их эффективного использования в качестве реальной альтернативы структурному программированию. Тем не менее, вопрос об эквивалентности этих двух подходов при реализации различных вычислительных алгоритмов пока остается открытым. Он может стать предметом отдельных исследований в области теории алгоритмов и прикладных вопросов программирования.

При введении с помощью соответствующих лингвистических средств механизмов наследования и полиморфизма методы матричного программирования могут быть совмещены с современными методами объектно-ориентированного программирования, а использование арифметико-логических выражений может составить серьезную альтернативу современным методам и средствам логического программирования. Предложенная

методика программирования отличается относительной простотой и может найти применение при изучении теории алгоритмов, а также при разработке новых эффективных вычислительных методов.

The possibilities of using the vector and matrix macrooperations to realize classical and original algorithms have been analyzed in the paper on the examples of solving the pragmatic programming tasks. Some new definitions of the arithmetic-logical expression, recurrent arithmetic-logical expression and vector-function were introduced to explained the basic singularities of the analyzed macrooperations. It is demonstrated on a lot of different examples, that the proposed method can be really considered as an alternative to currently used structured coding for the tasks of middle-level complicity.

1. *Дьяконов В. П.* Matlab 6/6.1/6.5+Simulink 4/5. Полное руководство пользователя. — М. : Солон-Пресс, 2002. — 768 с.
2. *Мартынов Н. Н.* Введение в Matlab 6. — М. : Кудиц-Образ, 2002. — 352 с.
3. *Мельник И. В.* Использование матричных макроопераций при работе в математических САПР на примере системы MATLAB//Вест. Херсонского государственного технического университета. Вып. 2 (28). — Херсон, 2007. — С. 199—205.
4. *Кнудт Д.* Искусство программирования/Пер. с англ. под общ. ред. Ю.В. Козаченко. — М. : Вильямс, 2003. — 560 с.
5. *Сэджвик Р.* Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск/ Пер с англ. — С-Пб. : ООО «ДиаСофт ЮП», 2002. — 688 с.
6. *Мэтьюз Д., Куртис Д.* Численные методы. Использование Matlab. — М. : Изд. дом «Вильямс», 2001. — 720 с.
7. *Самарский А. А., Гулин А. В.* Численные методы. — М. : Наука, 1989. — 432 с.
8. *Мельник И. В.* Моделирование транспортировки электронных пучков из области низкого в область высокого вакуума в эквипотенциальном канале// Электрон. моделирование. — 2001. — **23,** № 4. — С. 82—92.
9. *Молоковский С. И., Сушков А. Д.* Интенсивные электронные и ионные пучки. — М. : Энергоатомиздат, 1991. — 304 с.
10. *Ильина В. А., Силаев П. К.* Численные методы для физиков-теоретиков. — Москва-Ижевск : Институт компьютерных исследований, 2003. — 132 с.

Поступила 18.11.08;
после доработки 25.02.09

МЕЛЬНИК Игорь Витальевич, д-р техн. наук, доцент кафедры электронных приборов и устройств Национального технического университета Украины «Киевский политехнический ин-т», который окончил в 1989 г. Область научных исследований — изучение самосогласованной электронно-ионной оптики в высоковольтном тлеющем разряде, энергетика газового разряда и разрядной плазмы, взаимодействия электронных и ионных пучков с веществом.