MATLAB — plusy kontra minusy

Norbert Jankowski
Katedra Metod Komputerowych
Uniwersytet Mikołaja Kopernika
ul. Grudziądzka 5, 87–100 Toruń
e-mail: norbert@phys.uni.torun.pl, http:/www.phys.uni.torun.pl/˜norbert

Streszczenie. MATLAB przyciąga nas niezwykle bogatymi bibliotekami funkcji z bardzo wielu gałęzi nauk ścisłych. Drugą wielką, pozytywną cecha, MATLABa jest łatwość generowania różnorodnych wykresów, animacji. Cechy te sprawiają, że programy w języku MATLAB pisze się szybko, zaś same programy umożliwiają bardzo sprawną analizę prowadzonych doświadczeń. Jednak należy pamietać, że MATLAB jest (niestety) interpreterem[1]. Inną wadą MATLABa jest zbyt ubogi język programowania.

W niniejszym artykule chciałbym wskazać te elementy języka MATLAB, które wydają się być złe, lub którymi MATLAB nie dysponuje, a o które można by go rozszerzyć. Chciałbym również zwrócić uwagę na możliwość rozbudowania systemu MATLAB o bibliotekę klas/funkcji w języku C++/C, która umożliwiłaby pisanie aplikacji bezpośrednio w języku C++ lub C.

1 WSTĘP

Już od wielu lat język FORTRAN 77 jest jednym z języków, w których pisze się bardzo wiele programów, ale głównie (jeśli nie wyłącznie) są to programy numeryczne. Sam FORTRAN 77 nigdy nie dorobił się wspólnej dla różnych systemów operacyjnych biblioteki funkcji, umożliwiającej dołączanie prezentacji graficznych do pisanych programów. Także sam język programowania, jak na koniec XX wieku, jest bardzo prymitywny, choć wciąż bardzo efektywny obliczeniowo [1]. Brak takich elementów jak dynamiczny przydział pamięci, czy fakt, iż FORTRAN 77 nie jest językiem strukturalnym, świadczą również na jego niekorzyść. Istnieje obecnie również FORTRAN 90, który jest już językiem strukturalnym i posiada dynamiczny przydział pamięci, jednakże nie jest to jeszcze język powszechnie używany (brak zgodności z FORTRANem 77 sprawił iż do tej pory nie wyparł on FORTRANU 77).

Odmiennie prezentuje się najmodniejszy chyba obecnie język programowania C++. Nie ma on już takich wad jak język FORTRAN i dysponuje bardzo bogatymi konstrukcjami językowymi [2]. Jest to poza tym język programowania obiektowego. Ale podobnie jak FORTRAN nie dorobił się takiej bilioteki funkcji graficznych (spójnej dla różnych systemów operacyjnych), jaką dysponuje system MATLAB.

MATLAB bardzo szybko znalazł sobie wielu zwolenników. Stało się tak dzięki połączeniu 3 elementów: prosty[2] język programowania; wiele rozbudowanych, złożonych strukturalnie bibliotek funkcji dla wielu różnych dyscyplin nauki, których nie potrzeba implementować po raz kolejny, oraz bardzo bogate możliwości prezentacji danych.

Niestety MATLAB posiada również szereg wad, ale wydaje się, że można je zwalczyć lub przynajmniej częściowo poprawić i o tym głównie chciałbym napisać w dalszej części tego artykułu.

2 PROPOZYCJE WPROWADZENIA ZMIAN DO SYSTEMU I JĘZYKA MATLAB

2.1 Przekazywanie argumentów do funcji

Jak wiadomo język MATLAB [3] umożliwia nam przekazywanie argumentów do funkcji. Problem polega jednak na tym, że argumenty mogą być przekazywane jedynie przez stałą. Oznacza to, że zmiany takiego argumentu dokonane wewnatrz funkcji nie są zachowane po zakończeniu działania funkcji, w przeciwieństwie do argumentów przekazywanych przez zmienną.

Brak możliwości przekazywania argumentów przez zmienną do funkcji sprawia, że nie można efektywnie pisać programów. Albo piszemy nieczytelne programy, umieszczając kod takiej funkcji w miejscu jej potencjalnego wywołania, a bardzo często trzeba to zrobić w wielu miejscach, co z kolei powoduje ogromne źródło błędów; albo powstają bardzo nieefektywne programy, ponieważ umieszczamy wywołanie funkcji, do której przekazywane są argumenty przez wartość

[1] Obecnie jest już dostępny translator (public domain) z programów napisanych w języku MATLAB do języka C++. Niestety, jak na razie jedynie dla systemów MS Windows i Sun OS.

[2] czyt. nie bardzo skomplikowany

i wtedy zamiast dokonać operacji na nieznacznej części elementów macierzy, najpierw tworzona jest lokalna kopia takiej macierzy, potem dokonywane są odpowiednie operacje, a na koniec macierz lokalna jest kopiowana we właściwe miejsce pamięci.

Poniższy program ilustruje ten problem. Pierwsza jego część wykonuje pewne operacje bez wywoływania dodatkowych funkcji, a część druga korzysta z funkcji **f**. Rezultaty wykonywania obu części są identyczne, jednakże czasy wykonania różnią się znacznie (patrz rysunek **Wynik**).

Poza tym wydaje się bardzo dziwne, że nawet jeśli w ciele funkcji (np. **function [a] = f(a)**) identyfikatory argumentów funkcji występują również w liście wyników, to argumenty te i tak nie są przekazywane przez zmienną. A przecież w takiej sytuacji argumenty występujące również w liście wyników mogłyby być domyślnie przekazywane przez zmienną.

Program 1:

```
a = rand(1000);

tic;
a(1,1) = -15;
toc

a = rand(1000);
tic;
a = f(a);
toc
```

Funkcja f:

```
function [a] = f(a)

a(1,1) = -15;
```

Wynik :

```
elapsed_time =
    0.0550

elapsed_time =
    1.3122
```

2.2 MATLAB to interpreter

Język MATLAB jest dostarczany tylko wraz z interpreterem. Jest to bardzo często wygodne, bo zaoszczędza czas, który musielibyśmy stracić na proces kompilacji. Mamy dzięki temu możliwość szybkiego uruchamiania fragmentów kodu.

Niestety programy uruchamiane poprzez interpretery zawsze są o wiele wolniejsze niż te same programy, które zostały skompilowane. W połowie tego roku pojawił się translator MATCOM[3] autorstwa Yarona Kerena, który tłumaczy programy napisane w języku MATLAB na język C++. Jednakże i to nie rozwiązuje większości problemów ponieważ w chwili obecnej istnieje możliwość używania translatora wyłącznie dla komputerów PC (MS Windows, Win32 i Extended DOS) i systemu Sun OS. Poza tym MATCOM ma kilka ograniczeń: pod systemem MS Windows rozmiar macierzy jest ograniczony do 8190 elementów (jest to związane z wielkością segmentu – 64KB), MATCOM nie akceptuje w pełni grafiki MATLABa 4. Nie zostały również zaimplementowane rzadkie macierze.

Powyżej wymienione wady translatora MATCOM sprawiają, że używanie go nie rozwiązuje naszych problemów, a wręcz nastręcza kolejne.

Problem powolności interpretera można by rozwiązać na jeden z dwóch sposobów:

- Napisanie kompilatora języka MATLAB
- Napisanie kompletnej biblioteki MATLABa w języku C++ (kompilator również wymagałby takiej biblioteki) dla wszystkich systemów pod którymi pracuje MATLAB

Drugi sposób jest szybszy i łatwiejszy do osiągnięcia. Co więcej samo stworzenie biblioteki funkcji MATLABa umożliwi pisanie programów również w języku C++. W takich programach moglibyśmy wywoływać wszystkie funkcje biblioteczne MATLABa. Wtedy należałoby dołączyć jeszcze program, który przed procesem konsolidacji (ang. linking) produkowałby odpowiednie kody funkcji z bibliotek MATLABa (tzw. toolboxów) w C++, które mogłyby być usuwane zaraz po konsolidacji.

Inną cechą, jaką warto by dodać do samego języka MATLAB jest odpowiednik klasy zmiennych **exterm** z języków C i C++. Mogłoby to umożliwić łączenie kodów w MATLABie z innymi językami programowania (C, C++, FORTRAN, . . .). Rozbudowałoby to znacznie możliwości samego MATLABa.

Poza tym, implementując bibliotekę funkcji MATLABa w C++, można by tak podociążać operatory, że pisanie operacji macierzowych w C++ bardzo przypominałoby pisanie programu w MATLABie. A dobrze dociążone operatory sprawiłyby, że wyrażenia typu:
P = [Pw(1:n,n+1:n*gr_param), Pwv; Pwv', Pv];
(gdzie **Pw**, **Pwv**, **Pv** to macierze), byłyby realizowane bardzo efektywnie – tak naprawdę zostałaby utworzona tylko jedna macierz pomocnicza, która *stałaby* się macierzą wynikową (nawet pomimo iż macierz **Pwv** występuje jako **Pwv** i **Pwv'**).

2.3 Funkcja ≡ plik

W opinii wielu osób ograniczeniem jest również to, że kod każdej nowo implementowanej funkcji musi znajdować się w osobnym pliku. Skutkiem tego jest bar-

[3] MATCOM to produkt typu *free ware* i można go znaleść pod adresem: ftp://ftp.eeng.dcu.ie/pub/matlab/MATCOM

dzo szybko rosnąca liczba plików i w efekcie końcowym czasem bałagan. Oczywiście można by potworzyć mnóstwo podkartotek, ale to nie zmieniejsza liczby plików. Pisząc nasze programy musimy więc często mieć otwartych wiele plików, a to jest bardzo uciążliwe.

Wydaje się, że nic nie stoi na przeszkodzie, aby w jednym pliku umieszczać więcej niż jedną funkcję – patrz rysunek **Biblioteka NN** – tym bardziej, że w nagłówku funkcji widnieje jej nazwa. Trzeba by jedynie zmienić system poszukiwania funkcji w bibliotekach, ale chyba warto . . .

Biblioteka NN:

```
function phi=rbf_act_sigm(
        x_n, ro, u, s)

N = size(u, 2);
phi = zeros(N,1);
s2 = s .* s;
...

function d=rbf_deriv_sigm(
        x_n, w, ro, u, s)
N = size(u, 2);
M = size(x_n, 1);

phi = zeros(N,1);
du  = zeros(M, N);
dro = zeros(M, N);
ds  = zeros(M, N);

s2 = s .* s;
ro2= ro .* ro;
for j=1:N
  dx = x_n-u(:,j);

...
```

3 KONKLUZJE

Dokonanie zmian o których mowa powyżej, dałoby wielkie korzyści. Można by wykorzystywać wszystkie zalety MATLABa, doprowadzając do programu wynikowego (po kompilacji) z możliwością programowania mieszanego (w różnych językach), lub wykorzystywać funkcje biblioteczne MATLABa w programowaniu w języku C++.

Jak wynika z powyższych rozważań nie ma większych problemów natury teoretycznej, aby język MATLAB stał się językiem o wiele wydajniejszym, o bogatszych możliwościach. Miejmy nadzieję, że firma MathWorks podzieli moją opinię i wykorzysta wskazane przeze mnie niedoskonałości do stworzenia naprawdę dobrego języka programowania, na miarę potrzeb użytkownika z końca XX wieku.

Bibliografia

[1] Scott W. Haney. Is C++ fast enough for scientific computing? *Computers in Physics*, 8(6):690–694, Nov–Dec 1994.

[2] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, 1991. Second Edition.

[3] The MathWorks, Inc. *MATLAB. Reference Guide*, 1994.