NADSTAVBA JAZYKA C++ SYNTAKTICKY PODOBNÁ MATLABu

Vítězslav Veselý, Jiří Zelinka
KAM PřF MU Brno

Jiří Vala
ÚMDG FAST VUT Brno

Abstrakt. Cílem příspěvku je představit projekt objektově a maticově orientované nadstavby jazyka C++. Po stránce syntaktické i funkční je natolik kompatibilní s jádrem MATLABu, aby umožnila zcela nezávisle na něm: (i) snadný (mechanický) převod funkcí zapsaných v jazyce MATLAB do prostředí C++, (ii) přímou implementaci numericky orientovaných algoritmů v C++ stejně pohodlně jako v jazyku MATLABu (zejména bez potíží a rizik spojených s dynamickou alokací paměti), (iii) modulární vytváření samostatných aplikací ve formě spustitelných souborů EXE i DLL–knihoven. Všechny funkce takto vytvořené v C++ lze přímo volat nejen z prostředí MATLABu (což například usnadní výrazně jejich ladění v C++), ale i z jakéhokoliv programového systému podporujícího spolupráci s externími DLL-knihovnami.

1. Úvod

Programový systém MATLAB, primárně navržený a postupně vyvíjený jako prostředí pro realizaci širokého spektra vědeckotechnických výpočtů (na rozdíl od specializovaného a zpravidla méně flexibilního softwaru pro jednotlivé relativně úzké obory) se zabudovanými efektivními numerickými algoritmy a prostředky pro snadnou vizualizaci výsledků výpočtů, bývá často využíván jako účinný prostředek pro vývoj a testování nově navržených algoritmů; žádný z tradičních programovacích jazyků pro vědeckotechnické výpočty (jako Fortran, Pascal či C++) totiž nenabízí tak široké spektrum snadno a přirozeným způsobem (srovnatelným s běžným zápisem v matematickém textu) uživatelsky ovladatelných operací a funkcí pro práci s lineárními operátory (typicky dvourozměrnými maticemi). Po první etapě ověření algoritmu v prostředí MATLABu ovšem často vzniká v další etapě vývoje softwaru potřeba jeho implementace bez vazby na MATLAB — obvykle ve formě EXE-souboru nebo DLL-knihovny, jež lze vytvářet např. prostředky jazyka C++ (srov. [3, str. 443]). K dispozici jsou přitom následující možnosti:

- Přepsat kompletně celý program z jazyka MATLABu do standardního jazyka C++. Tento postup bývá mimořádně pracný; vedle různých technických problémů (např. s alokací a dealokací paměti pro rozsáhlé matice, kterou MATLAB zajišťuje implicitně) naráží totiž na velmi nepříjemnou potřebu nějak obcházet efektivní numerické algoritmy zabudované v MATLABu a jím využívaných knihovnách (i stejně či obdobně pojmenované jednoduché funkce v MATLABu a C++ přitom nezřídka poskytují kvalitativně odlišné výsledky). Nezávislost nového softwaru na MATLABu se tak u složitějších algoritmů stává až poněkud luxusním výsledkem dlouhodobé a krušné programátorské práce, jejíž těžiště leží mimo obor zpracovatelů.
- Použít kompilátor MATLAB Compiler & C/C++ Math Library nabízený společností MathWorks. V českém prostředí je toto řešení pro většinu uživatelů příliš nákladné.
- Nastudovat (např. z [5, str. 3-4]) techniku používání MATLABu jako výpočetního prostředku (computational engine), jehož funkce jsou volány z programu napsaného

např. v jazyku C++, resp. naopak vytváření tzv. MEX-souborů, které mohou být volány z MATLABu. V posledním případě se jedná o zdrojový kód C++ speciální DLL-knihovny s jedinou do MATLABu exportovanou funkcí typu `void` s povinným názvem `mexFunction`, jejímiž parametry jsou skutečné počty levostranných a pravostranných parametrů (typu `int`) a ukazatele na pole ukazatelů na tyto parametry uložené ve zvláštních strukturách `mxArray`. V tomto případě však pro použití takového programu, resp. DLL-knihovny, je nezbytný MATLAB.

Motivováni nespokojeností se všemi zmiňovanými přístupy, rozhodli se autoři tohoto příspěvku vytvořit nadstavbu jazyka C++ pro podporu numerických výpočtů, jejíž jednotlivé funkce by byly jednak přímo volatelné prostřednictvím vhodných MEX-souborů ze standardního MATLABu, ale současně by byly použitelné i zcela nezávisle na něm. Tedy umožňovaly by pohodlnou tvorbu nezávislých EXE-programů, případně tématicky specializovaných DLL-knihoven použitelných z libovolného programovém systému podporujícího spolupráci s takovými knihovnami (např. z hojně používaného tabulkového procesoru EXCEL, aj.).

2. Popis nadstavby

Hlavní snahou při vývoji diskutované nadstavby bylo, aby byla syntakticky co nejpodobnější MATLABu (např., aby byly zachovány symboly všech operací a jména funkcí s výjimkou nevelkého počtu potenciálních syntaktických i sémantických kolizí, z nichž nejmarkantnější je odlišné používání teček, kulatých a hranatých závorek); do jaké míry se to podařilo, snad ještě naznačí demonstrační ukázky. Speciální C++ struktura `mxArray` pro proměnné typu matice (obecně N-D array) z MATLABu (potřebná ve funkci `mexFunction`) byla nahrazena obdobnou třídou `MxArray`, resp. `ARR`. Ta však byla navržena nezávisle na struktuře `mxArray` jednak proto, že tato není dostatečně dokumentována, jednak z důvodů odstranění závislosti na jejích případných změnách v budoucích verzích MATLABu.

Navíc byla zavedena třída `cmplx`, resp. `COMPLEX`, která implementuje komplexní skaláry kompatibilně s `ARR`, tj. umožňuje automatickou konverzi skalárních matic `ARR` na typ `COMPLEX` a naopak (v přiřazovacích příkazech, při přenosu parametrů, ve složených výrazech obou typů apod.).

Při přepisu m-funkce z MATLABu pomocí nadstavby C++ je nutno počítat zejména s následujícími syntaktickými odchylkami:

- Maticové binární operátory pracující po složkách `.*`, `./`, `.^`, `~=` nemohou být v C++ předefinovány, a jsou proto po řadě nahrazeny operátory `_M`, `_D`, `_P`, `!=`.
- Obdobně maticové unární operátory `.'` (transpozice) a `'` (adjunkce) jsou po řadě nahrazeny operátory `_T` a `_A`.
- Blokové skládání matic ve tvaru `[A;B;...]`, resp. `[A,B,..]` je možno realizovat pouze ekvivalentními příkazy `cat(1,A,B)` a `cat(2,A,B)` (prozatím pouze pro dvě matice).

- Generování ekvidistantních vektorů konstrukcí `a:b`, resp. `a:d:b`, je nahrazeno voláním funkce `spc(a,b)`, resp. `spc(a,d,b)`.
- Konstantním funkcím `i`, `pi`, `eps`, `Inf`, `NaN` odpovídají po řadě funkce `im()`, `pi()` atd. Navíc je k dispozici funkce `empty()` jako náhrada prázdné matice `[]`.
- Jména několika málo dalších funkcí, jež vyvolávají kolizi v C++, jsou převedena na velká písmena (`LOGICAL`, `DOUBLE`, `CHAR`, `REAL`, `IMAG`, `MAX`, `MIN`).
- Maticový podmiňovací operátor `if` je nahrazen operátorem `IF`.
- Volání funkcí s více výstupními parametry ve tvaru `[X1,X2,...,Xn]=fun(A1,A2,...)` je třeba nahradit konstrukcí `LHS[X1][X2]...[Xn] OF fun(A1,A2,...)`, přičemž za skutečné parametry `A1,A2,...` není dovoleno dosazovat výrazy, v nichž vystupují volání jiných funkcí.
- V souladu s jazykem C++ je nutno všechny lokální proměnné v těle funkce před použitím deklarovat. Výstupní parametry `X1,X2,...,Xn` musejí být deklarovány na začátku funkce speciálním způsobem ve tvaru
  `ARR OUTP(X1,1),OUTP(X2,2),...,LASTOUTP(Xn,n)`. Tato konstrukce rovněž automaticky zajistí uložení prvního výstupního parametru do předdefinované globální proměnné `ans` stejně jako v MATLABu v případě, že funkce je volána bez použití konstrukce `LHS[X1][X2]...[Xn] OF`.
- Indexování na levé straně v přiřazení `X(I)=...`, resp. `X(I,J)=...`, je nahrazeno po řadě konstrukcemi `X[I]=...`, resp. `X[I][J]=...`.
- Navíc jsou k dispozici funkce `isdefault` (detekce nezadaných implicitních vstupních parametrů), `isscalar`, `isvector` (detekce skalární, resp. vektorové matice) a některé další funkce nižší úrovně, které zprostředkovávají přímou manipulaci s položkami třídy ARR jako obdoba některých podobných funkcí, které nabízí systém MATLAB tvůrcům MEX-souborů v C++.

Některé náročnější operace byly realizovány prostřednictvím univerzálních volně šířených a dobře prověřených knihoven: LAPACK [1] pro lineární maticovou algebru (rozklady, pseudoinverze aj.) a DCDFLIB pro elementární statistické výpočty (distribuční funkce a inverzní kvantilové funkce pro běžná rozložení pravděpodobnosti). Při instalaci softwaru vyvinutého s využitím prezentované nadstavby by tedy neměly vzniknout žádné obtíže související s autorskými právy a mnohdy spletitými licenčními podmínkami dodavatelů a prodejců rozličného komerčního softwaru.

Výše popsané jádro nadstavby může být zkompilováno jako statická nebo dynamická knihovna `mxalg.lib` nebo `mxalg.dll`. Autorům bude v budoucnu sloužit pro efektivní vývoj dalších specializovaných knihoven v oblastech jejich odborného zájmu (viz následující odstavec 3).

3. Možnosti použití

Nadstavba byla vytvořena a odladěna ve vývojovém prostředí Microsoft Visual C++ 5.0 a 6.0 pod operačním systémem Windows 98, funkce však nejsou závislé na konkrétním

operačním systému ani na speciálních knihovnách C++ pro Windows jako MFC (srov. [3, str. 448]). Nepředpokládá se vytváření komunikačních rozhraní COM pro Windows (ve smyslu [3, str. 487]), ale ani vývoj na MATLABu nezávislého prostředí pro dvoj- či trojrozměrné grafické znázorňování výsledků výpočtů. Autoři nadstavby ji v současné fázi používají pro implementaci vlastních specializovaných knihoven, ale mohou také případným zájemcům (oproti srovnatelným nástrojům za uživatelsky výhodných podmínek):

- poskytnout DLL-knihovnu `mxalg.dll` jako MEX-soubor k MATLABu, což uživateli umožní zaměnit v jeho m-souboru volání některých pomalých (interpretovaných) m-funkcí jádra MATLABu obdobnými (zkompilovanými) funkcemi z `mxalg.dll`, a urychlit tak výpočet (např. místo funkce `pinv(A)` voláme `mxalg('pinv',A)`);
- převést knihovnu m-souborů vytvořených uživatelem do jazyka C++ a zkompilovat ji do požadované podoby, např. ve formě plně funkčního rozhraní (DLL-knihovna, EXE-soubor apod.) pro vhodné programové prostředí (EXCEL aj.);
- nabídnout (dle svých sil a možností) komplexní řešení od analýzy problému přes návrh algoritmu až po jeho implementaci ve výše uvedené podobě v oblastech svého odborného zaměření — analýza časových řad (filtrace, predikce), vybrané statistické metody, jádrové vyhlazování, jádrové odhady hustoty pravděpodobnosti, speciální algoritmy rychlé Fourierovy a Hartleyho transformace optimalizované dle délky transformace, reprezentace (aproximace) funkcí pomocí tzv. "frejmových rozvojů" (angl. "frame expansions") ve funkcionálních prostorech (zobecnění nyní moderních waveletových rozvojů), případně i v jiných souvisejících oblastech ve spolupráci s dalšími odborníky doma i v zahraničí.

V budoucnu po dokonalém otestování jádra nadstavby `mxalg` a jeho rozšíření o další funkce počítají autoři v případě většího zájmu uživatelské veřejnosti i s možností jeho nabídky ve formě statické knihovny `mxalg.lib`. To umožní uživatelům přímo efektivně vyvíjet vlastní numericky orientované aplikace v jazyku C++. Totéž se týká i dalších do té doby vyvinutých specializovaných knihoven. Autoři se neuzavírají ani před možností rozšířit stávající autorský kolektiv o další odborníky, kteří předloží k realizaci kvalitní projekt vlastní vhodně tématicky zaměřené knihovny.

Pro demonstraci byly vybrány dvě typické úlohy, které lze efektivně řešit pomocí výše popsané nadstavby jazyka C++:

(1) Převod m-funkce do C++: Pro tento účel byla vybrána m-funkce `gelim.m`, která řeší systém lineárních rovnic gaussovou eliminaci s pivotováním ve sloupcích. Tento algoritmus je všeobecně známý a je přednášen ve standardních kurzech numerické matematiky na vysokých školách. Prvý z autorů ji vybral z knihovny m-funkcí používaných při praktikách ke své přednášce. Přes svou jednoduchost tento algoritmus dobře ilustruje typické rysy jazyka MATLAB a po převedení do C++ (`gelim.cpp`) tak umožňuje čtenáři snadno posoudit možnosti nadstavby porovnáním výpisů obou kódů.

```
/*====================================================================*/
function  x = gelim(A,b)
/*====================================================================*/
% GELIM   Gaussova eliminace pro reseni systemu linearnich rovnic.
%         Pivotovani ve sloupcich.
% Vzorove volani
%   x = gelim(A,b)
% Vstupy
%   A   ctvercova matice soustavy
%   b   sloupcovy vektor prave strany
% Vystup
%   x   sloupcovy vektor reseni
%--------------------------------------------------------------------
[n,n] = size(A);
A = [A,b];
x = zeros(n,1);
row = 1:n;       % Inicializace vektoru radkovych permutaci
for p = 1:n-1,
  [Max J] = max(abs(A(row(p:n),p)));   % Hledani pivotniho radku
  row([J+p-1,p]) = row([p,J+p-1]); % Zaznam simulovaneho prehozeni radku
  if  A(row(p),p)==0, x=[]; break, end
% Eliminace poddiagonalove casti p-ho sloupce:
  m = A(row(p+1:n),p)./A(row(p),p);
  A(row(p+1:n),p+1:n+1) = A(row(p+1:n),p+1:n+1) - m*A(row(p),p+1:n+1);
end
if isempty(x) | A(row(n),n) == 0, error('Matice soustavy je singularni'); end
% Zpetne dosazovani:
x(n) = A(row(n),n+1)./A(row(n),n);
for k = n-1:-1:1,
  x(k) = (A(row(k),n+1) - A(row(k),k+1:n)*x(k+1:n))./A(row(k),k);
end
%--------------------------------------------------------------------
```

```
#include "mxalg.h"
/*=========================================================================*/
ARR gelim(ARR A,ARR &b)                                  /* Gaussova eliminace */
/*=========================================================================*/
/*--->DESCRIPTION:
GELIM   Gaussova eliminace pro reseni systemu linearnich rovnic.
        Pivotovani ve sloupcich.
 Vzorove volani
  x = gelim(A,b)
 Vstupy
   A   ctvercova matice soustavy
   b   sloupcovy vektor prave strany
 Vystup
   x   sloupcovy vektor reseni
Returns: x
*/
/*-------------------------------------------------------------------------*/
/*--->CODE:                                                                */
/*                                                                         */
{
ARR LASTOUTP(x,1);
ARR m,n,row,Max,J;
int k,p;
LHS[n][n] OF size(A);
A = cat(2,A,b);
x = zeros(n,1);
row = spc(1,n);              // Inicializace vektoru radkovych permutaci
for(p=1;p<=Int(n)-1;p++)
 {abs(A(row(spc(p,n)),p));
  LHS[Max][J] OF MAX(ans);   // Hledani pivotniho radku
// Zaznam simulovaneho prehozeni radku:
  row[cat(2,J+p-1,p)] = row(cat(2,p,J+p-1));
  IF(A(row(p),p)==0){x=empty(); break;}
// Eliminace poddiagonalove casti p-ho sloupce:
  m = A(row(spc(p+1,n)),p) _D A(row(p),p);
  A[row(spc(p+1,n))][spc(p+1,n+1)] =
                    A(row(spc(p+1,n)),spc(p+1,n+1)) - m*A(row(p),spc(p+1,n+1));
 }
IF(isempty(x) | A(row(n),n) == 0) ERR("Matice soustavy je singularni");
// Zpetne dosazovani:
x[n] = A(row(n),n+1) _D A(row(n),n);
for(k=Int(n)-1;k>=1;k--)
  x[k] = (A(row(k),n+1) - A(row(k),spc(k+1,n))*x(spc(k+1,n))) _D A(row(k),k);
return x;
}
/*-------------------------------------------------------------------------*/
```

(2) Implementace nové funkce s využitím kódu jiných autorů: Tato ukázka již představuje řešení podstatně náročnější úlohy implementace rychlé Fourierovy transformace pomocí algoritmu převzatého ze [4]. Jelikož tento klasický algoritmus zde byl publikován v jazyce FORTRAN, byl nejprve převeden kompilátorem `f2c` [2] do jazyka C (`fft_.c`) a poté po menších úpravách využit při tvorbě funkce `fft.cpp`, která je obdobou vestavěné funkce `fft` v MATLABu. Pro zjednodušení kódu nebyla použita úplná optimalizace pro případ reálných dat sudé délky, kdy lze jistým opatřením zkrátit čas ještě na polovinu.

```
#include "mxalg.h"
#include "f2c.h"

/* EXTERNAL DECLARATIONS: */
extern "C" int fft_(doublereal *a, doublereal *b, integer *ntot, integer *n,
                  integer *nspan, integer *inc, integer *isn, integer *ierr);

/*=========================================================================*/
ARR fft                                     /* Discrete Fourier transform  */
/*=========================================================================*/
(ARR &X,
 ARR N,       // optional
 ARR DIM)     // optional
/*---> DESCRIPTION: Discrete Fourier transform
 FFT Discrete Fourier transform.
    FFT(X) is the discrete Fourier transform (DFT) of vector X.  If the
    length of X is well decomposable (e.g. a power of two), a mixed-radix
    (radix-2) fast-Fourier transform algorithm is used.
    If the length of X is not decomposable (a prime), a slower algorithm
    is employed. For matrices, the FFT operation is applied to each column.

    FFT(X,N) is the N-point FFT, padded with zeros if X has less
    than N points and truncated if it has more.

    FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the
    dimension DIM.

    For length N input vector x, the DFT is a length N vector X,
    with elements
                      N
       X(k) =        sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
                     n=1                                                 */
/*-------------------------------------------------------------------------*/
/*---> CODE:                                                              */
{ ARR LASTOUTP(Y,1); ARR sz,nr,nc; int dim,nd;
// f2c types:
  doublereal *pr,*pi,*px,*py,scale=1.;
  integer ntot,n,nspan,inc=1,isn=-1,ierr;
  Y=X;
  if(isempty(Y))return Y;
  sz=size(X);
  if(isdefault(DIM))
     if(isvector(X)){LHS[ans][DIM] OF MAX(sz);dim=Int(DIM);}
     else dim=1;
  else
     if(!isscalar(DIM)||(dim=Int(round(DIM)))<=0)
       ERR("fft: Dimension argument must be a positive integer scalar.");
     else
       if(dim>X.n_dims)return Y;
  if(isempty(N)||isdefault(N))n=Int(N=sz(dim));
  else if(!isscalar(N)||(n=Int(N=round(N)))<0)
       ERR("fft: Length argument must be a non-negative integer scalar.");
  if(n==0)return Y;
  nd = Int(n-sz(dim));
  if(nd<0){Y=0;Y=X(spc(1,dim==1?n:sz(1)),spc(1,dim==2?n:sz(2)));}
  if(nd>0) Y=cat(dim,X,zeros(dim==1?nd:sz(1),dim==2?nd:sz(2)));
  sz=size(Y);
  ntot=Int(prod(sz));
  nspan=Int(prod(sz(spc(1,dim))));
  //data pointers for fft_:
  pr = GRE(&Y);
  if(isreal(Y)){Y[1]=Y(1)+im(); *GIM(&Y)=0.0;}
  pi=GIM(&Y);
  // executing the transform:
  fft_(pr, pi, &ntot, &n, &nspan, &inc, &isn, &ierr);
  if (ierr!=0) ERR("fft: Working storage allocation error.");
  return Y;
}
/*-------------------------------------------------------------------------*/
```

5. Závěr

V příspěvku byl představen projekt, který posunuje tvorbu numericky a maticově orientovaných aplikací v C++ na kvalitativně vyšší úroveň jak z hlediska efektivity programátorské práce, tak z hlediska spolehlivosti výsledného kódu. Nabízí i vysokou míru syntaktické a funkční kompatibility s jádrem obdobně orientovaného systému MATLAB při zachování mnohem větší variability jazyka C++ a s vyšší rychlostí výsledného kódu (je odstraněna nevýhoda pomalé interpretace většiny příkazů MATLABu). Kompatibilita nabízí možnost snadného převodu algoritmů vytvořených v jazyce MATLABu do C++ ve formě nezávislých aplikací a naopak též snadnou tvorbu MEX-souborů (DLL-knihoven, jejichž funkce lze volat přímo z MATLABu).

Literatura

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK users' guide*, third ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
2. S. I. Feldman, D. M. Gay, M. W. Maimone, and N. L. Schryer, *A Fortran-to-C converter*, Computing Science Technical Report 149, AT&T Bell Laboratories, Murray Hill, NJ 07974, USA, March 22 1995.
3. D. J. Kruglinski, *Mistrovství ve Visual C++*, Computer Press, Brno, 1999.
4. R. C. Singleton, *An algorithm for computing the mixed radix fast Fourier transform*, IEEE Trans. on Audio and Electroacoustics **AU-17** (1969), no. 2, 93–103.
5. The MathWorks, Inc., 24 Prime Park Way, Natick, Mass. 01760, *MATLAB Application program interface guide*, January 1998.

Doc. RNDr. Vítězslav Veselý, CSc., Katedra aplikované matematiky, Přírodovědecká fakulta Masarykovy univerzity, Janáčkovo nám. 2a, 662 95 Brno.

tel.: +420-5-41321251/35, fax: +420-5-41210337
E-mail address: vesely@math.muni.cz, *URL:* http://www.math.muni.cz/~vesely

Mgr. Jiří Zelinka, Dr., Katedra aplikované matematiky, Přírodovědecká fakulta Masarykovy univerzity, Janáčkovo nám. 2a, 662 95 Brno.

tel.: +420-5-41321251/35, fax: +420-5-41210337
E-mail address: zelinka@math.muni.cz

Doc. Ing. Jiří Vala, CSc., Ústav matematiky a deskriptivní geometrie, Fakulta stavební Vysokého učení technického, Žižkova 17, 602 00 Brno.

tel.: +420-5-41147604
E-mail address: vala.j@fce.vutbr.cz