

Aplicacions Híbrides, Programari Lliure i Matlab

Autor: Guillem Borrell i Nogueras
Englobe Technologies
Laboratorio de Mecánica de Fluidos Computacional
ETSI Aeronáuticos
Universidad Politécnica de Madrid

Resum

Les Aplicacions Híbrides, escrites unint llenguatges interpretats i compilats, guanyen espai dins de la simulació en Ciència i Enginyeria dia a dia. La tendència actual és utilitzar Matlab com a base. A continuació s'introduirà breument a Octave i Python com alternatives amb alguns exemples i s'exploraran les seves possibilitats per desenvolupar aplicacions científiques amb eficiència i productivitat.

1. Introducció

Una aplicació híbrida és un element de programa o conjunt de rutines que utilitzen alhora llenguatges compilats i interpretats. La modalitat més comú és partir d'un llenguatge compilat i utilitzar “extending” i “wrapping”.

Extending és escriure un element de programa amb un llenguatge compilat, normalment C i C++ i encara Fortran dins del món científic, de tal manera que l'interpret pugui entendre-la com si fos escrita amb el seu propi llenguatge. Sol utilitzar-se quan el llenguatge interpretat no és capaç de realitzar òptimament una tasca, normalment per motius de velocitat d'execució.

Wrapping és aconseguir que l'interpret reconegui com a pròpia una biblioteca o part d'ella escrita amb un llenguatge compilat. És una tècnica molt comú per reutilitzar codi ja escrit i provat amb millor rendiment.

A la figura 1 s'esquematitza una aplicació híbrida tipus.

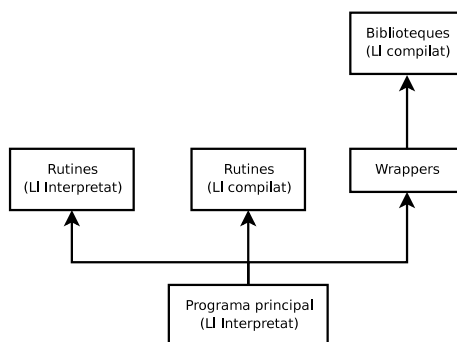


Figura 1: Esquema d'una aplicació híbrida.

És un concepte de desenvolupament de programari modern, la popularització dels llenguatges interpretats i interactius va arribar a finals dels anys 90 y la maduració de les eines d'extensió és encara més moderna. Cal afegir que món científic i de l'Enginyeria és poc propens a utilitzar eines noves.

En molts casos els programes de simulació necessiten un rendiment altíssim. Aquesta limitació fa que el primer criteri de disseny sigui utilitzar un llenguatge que proporcioni la màxima velocitat, normalment compilat, descartant els beneficis que aporten els llenguatges interpretats. Aquí és on entren les aplicacions híbrides. Tot el nucli del programa principal estaria escrit amb un llenguatge interpretat i només les rutines més exigents computacionalment s'escriurien amb un llenguatge compilat. És la manera més senzilla d'obtenir els beneficis d'ambdues pràctiques sense haver d'arrossegar-ne els inconvenients.

Un altre cas d'aplicació es troba en els petits programes on només una petita part de l'algoritme no és òptima. En aquest cas caldria tornar a escriure només unes quantes línies de codi per arreglar el problema.

Si les aplicacions híbrides no són habituals avui en dia és principalment perquè no formen part dels coneixements més extesos de programació. Cal estar al dia i comprobar l'evolució d'aquesta disciplina per adonar-se del seu creixement. Alguns dels factors que han provocat aquesta carència són:

- Cal conèixer i confiar en els llenguatges interpretats moderns.
- Es necessiten coneixements de programació més amplis.
- Cal dominar més d'un llenguatge de programació; com a mínim dos.
- Cal saber utilitzar eines específiques.
- La configuració de l'entorn de desenvolupament no és trivial i alguns cops no és senzill.

De totes les modalitats de desenvolupament d'aplicacions híbrides el més utilitzat dins del ram de la simulació numèrica és partir de Matlab. Matlab disposa d'eines per acoplar rutines escrites amb C i Fortran com a funcions però com tothom sap no es tracta de software lliure.

Els principals usuaris de Matlab avui en dia són les empreses d'enginyeria, els centres d'investigació i les universitats. Són també les organitzacions que normalment ténen més despeses en llicències de programari. Els cal un motiu tècnic i no només filosòfic perquè no estan massa interessades en la llibertat d'ús.

A diferència de molts altres casos el Programari Lliure disposa d'eines per substituir-lo i fins i tot superar-lo en funcionalitat. Les dues possibilitats més interessants són Octave i Python.

A continuació s'explorà l'ús d'Octave i Python per la construcció d'aplicacions híbrides i es visitaran alguns exemples pràctics per demostrar-ne la seva idoneïtat.

2. Octave

Octave busca la compatibilitat amb Matlab tot i que no era el seu objectiu principal fa un temps. És un projecte madur amb una llarga trajectòria i és multiplataforma. És part del projecte GNU i actualment es pot considerar compatible amb Matlab en un 95%. Està escrit amb C++ que és també el llenguatge natural per escriure extensions. Presenta, però, més dificultats per comunicar-se amb C i Fortran.

El seu principal punt fort és que, quan cal escriure una extensió, disposa d'una de les biblioteques per al Càlcul Numèric més interessants del programari lliure [API]. Això el fa especialment adequat per aplicar-lo a l'escriptura d'aplicacions híbrides.

2.1 Una aplicació senzilla

Aquesta és l'equació diferencial de l'atractor de Lorentz:

$$\begin{aligned}\dot{x} &= a(y - x) \\ \dot{y} &= x(b - z) - y \\ \dot{z} &= xy - cz\end{aligned}$$

Per integrar-la cal escriure-la com un arxiu .m:

```
function xdot=lorentz(t,x)
    a=10;b=28;c=8/3;
    xdot(1,1)=a*(x(2)-x(1));
    xdot(2,1)=x(1)*(b-x(3))-x(2);
    xdot(3,1)=x(1)*x(2)-c*x(3);
end
```

Ara la mateixa funció en C++ preparada per ser acoplada a Octave:

```
#include <octave/oct.h>
DEFUN_DLD (eqlorentz, args, ,
    "Ecuacion de Lorentz en C++") //1
{
    ColumnVector xdot (3); //2
    ColumnVector x (args(0).vector_value());
    int a=10;
    int b=28;
    double c=8./3;
    xdot(0) = a*(x(1)-x(0));
    xdot(1) = x(0)*(b-x(2))-x(1);
    xdot(2) = x(0)*x(1)-c*x(2);

    return octave_value (xdot); //3
}
```

L'esforç de tornar a escriure la funció amb C++ no és tan exagerat. Només cal conèixer unes poques funcions per crear valors i tornar-los a l'interpret. A continuació unes quantes aclaracions sobre el codi

1. Aquesta és la capçalera de la funció, DEFUN_DLD diu al compilador que no es tracta d'una funció qualsevol.
2. El tipus `ColumnVector`, un vector columna, és un dels tipus propis de la biblioteca d'Octave [\[API\]](#)
3. Una de les poques precaucions necessàries alhora d'escriure una extensió per Octave és que la funció ha de tornar un `octave_value`

Si la primera és comprensible per Octave directament la segona vol un procés de compilació i enllaç amb la biblioteca del programa:

```
$> mkoctfile eqlorentz.cpp
```

El procés ha generat un arxiu anomenat `eqlorentz.oct` que per Octave és equivalent a un .m amb la diferència de ser un binari. Si s'integra amb la primera versió de la funció:

```
>> x0=[1;1;1];
>> t=linspace(0,50,5000);
>> tic;x=lsode(@lorentz,x0,t);toc
```

S'arriba al resultat en un temps de 5.1 segons¹. Però si s'utilitza la nova versió en C++:

```
>> x0=[1;1;1];
>> t=linspace(0,50,5000);
>> tic;x=lsode(@eqlorentz,x0,t);toc
```

El temps del procés s’ha reduït a 0.36 segons! És un ordre de magnitud de millora i s’acosta molt als temps obtinguts amb un programa escrit només amb C++ amb un cost molt menor.

Queda demostrat, si més no amb un cas força senzill, els llenguatges interpretats són normalment entre un i dos ordres de magnitud més lents que els compilats però gràcies a escriure només una rutina amb C++ aquesta diferència s’ha reduït significativament.

Per escriure extensions a octave no es disposa de massa informació. Un bon punt de partida són [\[OCT\]](#) i [\[INT\]](#).

3. Python

Python és un llenguatge consistent i madur que ha guanyat una fama tan enorme com merescuda durant la darrera dècada. Gràcies a mòduls dedicats al càlcul numèric i a tasques científiques com NumPy, SciPy, Matplotlib... S’ha situat en paral·lel a una eina tan potent com Matlab. Parteix d’un punt de vista complementari: Matlab és una biblioteca unida per un llenguatge de programació mentre que Python és un gran llenguatge dotat amb posterioritat de les biblioteques necessàries. Una bona introducció a l’ús d’aquest llenguatge per aplicacions científiques és [\[LAN\]](#)

Disposa d’eines molt potents tan per l’“extending” com pel “wrapping” com:

- La pròpia API, molt potent i ben documentada.
- Pyrex, un llenguatge de programació entre Python i C dissenyat per escriure extensions.
- SWIG, una eina per generar wrappers automàticament.
- WEAVE, per interpretar porcions de codi escrites amb C dins de funcions o classes escrites en Python.
- f2py, un generador de wrappers per Fortran que és capaç de convertir un mòdul de Fortran 95 en un mòdul de Python gairebé sense esforç.
- ctypes, una col·lecció de funcions que permeten a Python comunicar-se amb qualsevol biblioteca escrita amb C directament.

A continuació, un petit exemple de l’ús de WEAVE i de ctypes.

3.1 Una aplicació senzilla

La pràctica d’incrustar codi independent dins d’una funció s’anomena “inlining” i és la base de les extensions escrites mitjançant WEAVE. Es torna a escriure la funció de l’equació diferencial de l’atractor de Lorentz com a exemple il·lustratiu:

```
from numpy import empty,array #1
from scipy import weave
from scipy.weave import converters

def lorentz(x):
    xdot=empty(3) #1
    a=10
    b=28
    c=8./3.
    xdot[0] = a*(x[1]-x[0])
    xdot[1] = x[0]*(b-x[2])-x[1]
    xdot[2] = x[0]*x[1]-c*x[2]

    return xdot
```

```

def lorentz_weave(x):
    xdot=empty(3)
    code=""" #2
double a=10;
double b=28;
double c=8./3.;
xdot[0] = a*(x[1]-x[0]);
xdot[1] = x[0]*(b-x[2])-x[1];
xdot[2] = x[0]*x[1]-c*x[2];
return_val=1;
"""

    err = weave.inline(code, #3
                       ['x','xdot'],
                       type_converters=converters.blitz,
                       compiler='gcc')

    return xdot

```

La primera funció està escrita només amb Python i la segona conté inlinig. Ambdues es poden utilitzar de la mateixa manera, la transformació del codi es fa sense que l'usuari hi hagi d'intervenir.

1. Python ha d'importar els tipus numèrics necessaris. `empty` és una funció que reserva memòria per un array.
2. El codi en C s'introdueix com una cadena de caràcters.
3. L'inlining és bastant sofisticat: `weave` només necessita el codi, les variables d'entrada i sortida i el convertidor. `Weave` crearà l'extensió, la compilarà i l'enllaçarà automàticament.

Aquest exemple en concret és purament acadèmic perquè no hi ha cap guany en l'eficiència.

3.2 Un wrapper sense complicacions

Ctypes és una de les eines més interessants de Python ja que permet accedir directament a les biblioteques escrites amb C. Normalment la tasca d'escriure wrappers implica utilitzar el llenguatge amb el que s'ha escrit l'interpret i coneixer amb profunditat les seves propies biblioteques. En el cas de Python la tasca s'ha simplificat significativament com es pot comprobar en l'exemple següent:

```

from ctypes import c_int, POINTER #1
import numpy as np
from numpy.ctypeslib import load_library,ndpointer #1

def dgesv(N,A,B):
    A = np.asfortranarray(A.astype(np.float64)) #2
    B = np.asfortranarray(B.astype(np.float64))

    cN=c_int(N)
    NRHS=c_int(1)
    LDA=c_int(N)
    IPIV=(c_int * N)()
    LDB=c_int(N)
    INFO=c_int(1)

```

```

lapack=load_library('liblapack.so','/usr/lib/')#3

lapack.dgesv_.argtypes=[POINTER(c_int),
                        POINTER(c_int),
                        ndpointer(dtype=np.float64,
                                ndim=2,
                                flags='FORTRAN'),
                        POINTER(c_int), POINTER(c_int),
                        ndpointer(dtype=np.float64,
                                ndim=2,
                                flags='FORTRAN'),
                        POINTER(c_int),POINTER(c_int)]#4

lapack.dgesv_(cN,NRHS,A,LDA,IPIV,B,LDB,INFO)#5
return B

print dgesv(2,np.array([[1,2],[1,4]]),np.array([[1,3]]))

```

1. Per crear aquest 'wrapper' no només cal importar les biblioteques, també es necessitaran tipus concrets per comunicar-s'hi.
2. A continuació es creen les variables necessàries amb els arguments necessaris. En alguns casos caldrà convertir-les perquè no es produeixin errors de segmentació.
3. Aquesta és la màgia de ctypes. La biblioteca sencera, en aquest cas lapack, es converteix en una variable.
4. Per poder saber què va malament s'expliciten els tipus dels arguments que es passaran a la funció.
5. Finalment s'executa la funció.

Com es pot comprobar en la darrera línia del codi s'ha utilitzat la funció **dgesv** de Lapack directament amb un tipus propi de Python (**array**). Possiblement aquest exemple sembli inútil ja que es disposa d'un operador que realitza la mateixa tasca però és com un regal per qui ha hagut d'escriure wrappers algun cop.

4. Conclusions

Les aplicacions híbrides són útils en el camp científic i tècnic i van guanyant popularitat. Matlab és el llenguatge més comú i planteja l'inconvenient que no és software lliure. Les possibles alternatives, Octave i Python, no només són vàlides per substituir Matlab sino que en alguns casos el poden superar.

Referències

[API] *Referència doxygen de liboctave* : <http://pareto.uab.es/mcreel/OctaveClassReference/html/index.html>

[LAN] Python Scripting for Computational Science, *Hans P. Langtangen*. 2004, Springer.

[INT] Introducció Informal a Matlab y Octave, *Guillem Borrell i Nogueras* . Disponible a: <http://servidor-da.aero.upm.es/>
[OCT] Wiki del projecte Octave. <http://wiki.octave.org/>

Apèndix A. Matlab

Matlab és un llenguatge de programació interpretat i interactiu, això significa que a diferència de C o Fortran no cal generar un executable sino que un programa anomenat intèrpret rep les ordres i les executa. El conjunt de llenguatge de programació, intèrpret i la biblioteca de funcions es considera una única aplicació i rep també el nom de Matlab. Així, fer servir Matlab es programar en Matlab.

La biblioteca de funcions és una de les més extenses i útils dins del camp del Càlcul Numèric i l'enginyeria i l'intèrpret es pot estendre mitjançant subrutines escrites amb C o Fortran.

Matlab té la gran virtut de convertir en trivials tasques senzilles com:

```
>> 2+2  
ans = 4
```

O de convertir en senzilles operacions que no ho són en absolut com per exemple aquesta integral d'una funció de Bessel

$$\int_0^{4.5} J_{2.5} dx$$

```
>> quad(@(x) besselj(2.5,x),0,4.5)  
ans = 1.1178
```

Amb Matlab també és senzill representar gràficament línies i superfícies.

Però Matlab no és perfecte; no disposa de suport per la programació orientada a objectes², no és modular ni plenament interactiu. Aquí no acaben els inconvenients, per ser un producte comercial sotmès a les estratègies de màrqueting de MathWorks el llenguatge no és estable i degut a problemes de compatibilitat amb versions anteriors pateix algunes inconsistències. Abans d'adquirir una llicència s'ha de meditar si realment val el que costa.

Com a llenguatge de programació té competidors directes, alguns d'ells molt seriosos. Cal destacar Mathematica, Scilab, IDL, R, Octave i una nova estrella emergent: Python gràcies als projectes NumPy i SciPy.

A.1 Una mica d'història

Matlab és un producte de [MathWorks](#). Fou dissenyat a finals dels anys setanta per Cleve Moler com una eina per evitar als seus estudiants haver d'aprendre Fortran. Va aparèixer com a producte comercial l'any 1984 i des d'aquell moment ha anat guanyant usuaris sobretot dins del camp del control lineal i no lineal i en l'enginyeria en general. La versió més recent quan s'escriuen aquestes línies és la R2007a.

A.2 Cal substituir Matlab?

Hi ha alguna necessitat de buscar un substitut lliure a Matlab? És una necessitat real o només es pot explicar amb arguments filosòfics? Alguns projectes de programari lliure argumenten que la llibertat d'ús és un motiu suficient com per no utilitzar mai programari propietari. Aquest argument no és vàlid per una empresa que vol un producte garantit i amb suport. Aquest sector, el que de debò genera la riquesa, necessita raons de pes per escollir un producte i descartar-ne un altre i més tenint en compte que parlem de quelcom tan útil i necessari com Matlab.

A.2.1 Matlab és un producte comercial

Matlab no és només un programa, és un llenguatge de programació. Moltes vegades sembla que MathWorks no pensa igual. No existeix cap definició formal que l'estandaritzi, només es disposa de la documentació i el propi funcionament de l'interpret. Això fa que no es pugui considerar un autèntic llenguatge de programació i els primers perjudicats són els usuaris. Per obtenir-ne la documentació és imprescindible adquirir una còpia del programa sencer.

La raó d'aquesta mancança és desconeguda però si es pensa amb mala fe sembla que hi ha un interès especial en impedir que hi hagi altres interprets compatibles amb l'original. Això ha succeït amb llenguatges com Java, Python, Lisp... Sembla contradictori tenint en compte que el veritable valor de Matlab és la seva enorme biblioteca de funcions i la mediocritat del seu llenguatge de programació.

A.2.2 Matlab és car, molt car

Quant costa Matlab? Prou com perquè moltes empreses l'hagin de descartar. Aquest problema no es limita a petites empreses o consultories on el cost d'una única llicència pot significar una porció significativa dins del seu pressupost, també afecta a grans multinacionals.

Teras és una petita consultoria acabada de néixer. Un dels seus camps és el sector aeroespacial. Com qualsevol empresa del sector l'interessava adquirir Matlab per augmentar la seva productivitat. Ho va haver de descartar degut al preu desorbitat d'una única llicència, equivalent a la meitat del sou anual d'un consultor júnior.

Rolls Royce, multinacional que ocupa la segona posició mundial com a fabricant de motors per aviació comercial es troba en l'altre extrem del món empresarial. Per una empresa tan gran el volum de dades a compartir entre enginyers és enorme i és imprescindible que tots els qui estiguin involucrats en el projecte puguin obrir els arxius. Això implica que cada enginyer ha de disposar de una còpia de Matlab quan la necessiti. L'estimació del nombre de llicències necessàries era tan gran que es va descartar, actualment utilitzen fulles de càlcul amb la pèrdua de productivitat que implica.

A.2.3 Matlab a les universitats

La Universitat Politècnica de Madrid disposa d'un nombre no despreciable de llicències per a l'ús acadèmic. Les actualitza periòdicament perquè tant alumnes com personal docent i investigador disposin de l'última versió. En algunes escoles tècniques les pràctiques de càlcul numèric es realitzen amb Matlab.

Un alumne de segon o tercer curs només utilitza una part mínima de l'eina. És necessari que disposi d'un programa tan costós per introduir-se dins del món de la programació? Un intent de justificació és que cal que aprengui el que s'usa en la indústria però ja s'ha vist que la seva implantació no és exempta de dificultats.

És raonable fer-ho sacrificant el pressupost necessari per adquirir dos ordenadors? No seria més senzill instal·lar un programa gairebé idèntic com Octave? Probablement els responsables d'IT de moltes universitats mai s'hagin formulat la pregunta.

A.3 Matlab Vs. Octave

Malgrat els possibles inconvenients Matlab és un gran programa. L'entorn de treball està ben dissenyat i pot utilitzar-se en qualsevol dels tres sistemes operatius majoritaris. Les seves possibilitats de representació gràfica de dades són gairebé il·limitades, el compilador Matlab-C és realment potent i l'optimització JIT³ és efectiva en molts casos.

D'altra banda el seu parser és bastant ruc, fins i tot rebutja estructures sintàctiques que serien consistents dins del llenguatge; com que no és un projecte de programari lliure no hi ha manera d'ampliar-lo ni definir nous tipus; l'entorn de treball, escrit en Java, és molt inestable en GNU/Linux i finalment, i no per això menys important, és car.

Octave soluciona alguns dels inconvenients tècnics de Matlab mantenint-ne la compatibilitat. El seu entorn de treball és la consola del sistema que pot ser vist tant com una virtut com un inconvenient; el seu parser és més llest; l'interpret, tot i els seus problemes d'estabilitat, es renova amb més freqüència;

és molt fàcil d'ampliar amb C++ perquè permet l'accés a tota la seva maquinària interna i la seva biblioteca interna es pot utilitzar per construir programes en C++. És, definitivament, una eina més interessant per a un bon programador.

A.4 Conclusions

Una de les tentacions comuns dels usuaris habituals de Matlab és, com que és capaç de completar un ampli ventall de tasques, utilitzar-lo per absolutament tot. Tot significa crear entorns gràfics, escriure grans programes, analitzar estructures de dades complexes... Sembla raonable pensar que una eina no ha de servir per qualsevol feina. Si no es compleix cap de les condicions següents:

- Es fa servir Matlab per absolutament qualsevol tasca
- Cal un toolkit en concret i és impossible programar-lo en un temps raonable.
- Es disposa ja d'una llicència
- S'està obligat a utilitzar-lo

no hi ha cap motiu raonable per no provar Octave

¹ Matlab disposa de quelcom semblant a l'orientació a objectes però no pot considerar-se com a tal. Poden agrupar-se funcions de tal manera que el seu funcionament sigui semblant a com ho seria en un objecte però no és capaç de definir una classe.

² JIT són les inicials de Just In Time, una tecnologia d'optimització automàtica del codi que fa que el seu principal defecte pel que fa a rendiment, els bucles, ja no ho siguin tant.

³ Calculat amb un Athlon 2000 XP, ja obsolet tenint en compte que només disposa de 3dnow y SSE.