MÈTODES NUMÈRICS
Grau en Matemàtiques, 2013–2014.
Sessió introductòria 2.

(no és una pràctica a lliurar, però està plantejada com si ho fos)

1 Motivació del treball

Una empresa que es dedica a fer simulació numèrica us requereix que desenvolupeu unes utilitats per a poder calcular productes, combinacions lineals i normes de matrius en la línia de comandes Unix. Per exemple, suposant que A és una matriu 30×30 emmagatzemada al fitxer `A` i que v és un vector de 30 components emmagatzemat a `v`, es vol poder emmagatzemar el vector $u = Av$ en un fitxer `u` amb una comanda com

```
./mmul 30 30 1 A v.txt > u.txt
```

Degut al tipus d'encàrrecs que rep aquesta empresa, els programes elaborats pel seu personal sempre acaben fent càlculs amb matrius que ocupen gariebé tota la memòria, i és per això que, per tal de depurar els seus programes, volen un conjunt d'utilitats petites que es puguin cridar des de la línia de comandes.

A més de les utilitats, també volen una biblioteca amb rutines que permetin fer operacions matricials, preparada per a poder ser inclosa directament al seu codi. Finalment, també volen tant una rutina com una utilitat que permeti exportar matrius a Octave.

2 Requeriments tècnics

2.1 Definicions

Per a les utilitats, una matriu $A = (a_{i,j})_{i=1\div m, j=1\div n}$ serà una seqüència de text ASCII amb els seus coeficients, ordenats per **columnes**[1] i separats per qualsevol tipus d'espai en blanc (espai, salt de línia, tabulador o salt de pàgina). Així, per exemple, la matriu 2×3

$$A = \begin{pmatrix} 2.3 & -1.4 & 3.5 \\ 4.5 & -7.3 & -3.2 \end{pmatrix} \tag{1}$$

es pot representar mitjançant la seqüència de text

```
2.3 4.5 -1.4 -7.3
3.5 -3.2
```

Aquest text també podria estar en una sola línia:

```
2.3 4.5 -1.4 -7.3 3.5 -3.2
```

[1] Per compatibilitat amb Fortran i LAPACK.

Un vector de n components serà una matriu $n \times 1$ d'acord amb els convenis anteriors. Un vector fila de n components serà una matriu $1 \times n$.

Per a la biblioteca, una matriu $n \times m$ serà un vector unidimensional de tipus `double`, amb els coeficients de la matriu disposats per columnes. Per exemple, la declaració en C

```
double aa[9] = { 2.3, 4.5, -1.4, -7.3, 3.5, -3.2 };
```

defineix la matriu A de (1). Com abans, un vector de n components és una matriu $n \times 1$ d'acord amb els convenis anteriors, i un vector fila de n components es una matriu $1 \times n$.

Per a una matriu $A = (a_{i,j})_{i=1\div m, j=1\div n}$, es defineixen la seva norma–1 i la seva norma–infinit per

$$\|A\|_1 = \max_{j=1\div n} \sum_{i=1}^{m} |a_{i,j}|, \qquad \|A\|_\infty = \max_{i=1\div m} \sum_{j=1}^{n} |a_{i,j}|,$$

respectivament.

2.2 Productes a lliurar

2.2.1 Utilitats

A les següents especificacions sintàctiques, els claudàtors (`[ ]`) indiquen que l'argument que envolten és opcional.

Es requereixen:

1. Una utilitat que, donada una matriu A $m \times n$ i una matriu B $n \times p$, torni el seu producte AB. Ha de respondre a la crida

   ```
   ./mmul m n p [ A.txt [ B.txt ] ]
   ```

   Si no s'especifica el fitxer `B`, ha de llegir els seus coeficients per standard input. Si tampoc s'especifica el fitxer `A`, s'han de llegir per standard input primer els coeficients de A i després els de B. Ha de treure per standard output els coeficients de la matriu producte AB.

2. Una utilitat per sumar dues matrius $m \times n$, que respongui a la crida

   ```
   ./msum m n [ A.txt [ B.txt ] ]
   ```

3. Una utilitat per calcular una combinació lineal de dues matrius $m \times n$, $\alpha A + \beta B$, amb $\alpha, \beta \in \mathbb{R}$. Ha de respondre a la crida

   ```
   ./mcl m n alf bet [ A.txt [ B.txt ] ]
   ```

4. Una utilitat que calculi la norma–1 d'una matriu, que respongui a la crida

   ```
   ./mnrm1 m n [ A.txt ]
   ```

Ha de tornar la norma demanada per standard output.

5. Una utilitat que calculi la norma–infinit d'una matriu, que respongui a la crida

   ```
   ./mnrminf m n [ A.txt ]
   ```

6. Una utilitat que respongui a la crida

   ```
   ./moctexp m n nom [ A.txt ]
   ```

   A partir de la matriu dins el fitxer `A.txt` o passada per standard input d'acord amb les definicions anteriors, la utilitat `moctexp` ha de generar per standard output la matriu en el format adequat per tal que pugui ser importada a Octave mitjançant la comanda `load`. Per exemple, si la matriu (1) fos dins el fitxer `A.txt`, la crida

   ```
   ./moctexp 2 3 M A.txt
   ```

   hauria de bolcar el següent:

   ```
   # Created by moctexp
   # name: M
   # type: matrix
   # rows: 2
   # columns: 3
   2.3
   -1.4
   3.5
   4.5
   -7.3
   -3.2
   ```

   Atenció! Observeu que la matriu es bolca **per files**.

2.2.2 Biblioteca

La biblioteca que es requerix ha de constar de les següents rutines:

- Una funció amb prototipus

  ```
  void mmul (int m, int n, int p, double *aa, double *bb, double *cc);
  ```

  que torni, dins el vector apuntat per `cc`, el producte de les matrius emmagatzemades als vectors apuntats per `aa` i `bb`, que són $\mathtt{m} \times \mathtt{n}$ i $\mathtt{n} \times \mathtt{p}$, respectivament.

- Una funció amb prototipus

```
void mcl (int m, int n, double alf, double *aa, double bet, double *bb,
      double *cc);
```

que, donades les matrius $m \times n$ A i B, emmagatzemades als vectors apuntats per `aa` i `bb`, respectivament, amb $m =$ `m` i $n =$ `n`, i donats els coeficients $\alpha =$ `alf` i $\beta =$ `bet`, torni, dins el vector apuntat per `cc`, la matriu $C = \alpha A + \beta B$.

- Una funció amb prototipus

```
double mnrm1 (int m, int n, double *aa);
```

que torni via `return` la norma–1 de la matriu `m` $\times$ `n` guardada al vector apuntat per `aa`.

- Una funció anàloga a l'anterior, amb prototipus

```
double mnrminf (int m, int n, double *aa);
```

però per la norma–infinit en comptes de la norma–1.

- Una funció amb prototipus

```
void moctexp (int m, int n, double *aa, char *nom, FILE *fp);
```

que, donada la matriu `m`$\times$`n` A dins `aa[]`, la bolqui per l'stream `fp` en format per importar des d'Octave (vegeu més amunt), tot posant-li com a nom la tira de caràcters dins `nom`.

2.3 Etapes de desenvolupament

1.– Escriptura de la biblioteca.

2.– Validació de les rutines escrites, mitjançant testos a determinar per vosaltres (tot i que recomanable, aquest pas és opcional, donat que el punt 4 valida les rutines també).

3.– Escriptura de les utilitats.

4.– Validació de les utilitats. Feu proves amb matrius generades aleatòriament, i comproveu els resultats amb Octave.

3 Terminis i productes a lliurar

Aquesta pràctica no s'ha de lliurar, tot i que emprarem alguna rutina de la biblioteca a la tercera pràctica. Si no fos així, hauríeu de lliurar:

- Un fitxer `malg.c`, amb el codi de les rutines `mmul()`, `mcl()`, `mnrm1()` i `mnrminf()`.
- Un fitxer `malg.h`, amb els prototipus corresponents a les rutines anteriors.
- Fitxers `mmul.c`, `msum.c`, `mcl.c`, `mnrm1.c`, `mnrminf.c`, amb els programes principals de les utilitats corresponents. Les utilitats s'han de poder generar mitjançant La utilitat `mmul.c` s'ha de poder generar amb

  ```
  make mcl mnrminf mnrm1 mmul msum
  ```

  d'acord amb el fitxer `Makefile` que trobareu al campus virtual.

4 Apèndix: carreres contra Octave i cridar BLAS

Existeix un paquet de software per càlcul científic anomenat Matlab que és usat per enginyers i científics arreu del món. Consta de diverses toolboxes per a diferents aplicacions científiques i tècniques. Per sota hi ha un llenguatge de programació (que també es s'anomena Matlab) orientat a matrius[2] que està molt pensat per fer càlculs complexos amb poques línies de codi. Octave és un paquet de software del projecte GNU que implementa el llenguatge de Matlab. A partir d'ara ens referirem a Octave, però tot el que veurem és vàl·lid per Matlab.

Octave és un llenguatge interpretat. Això permet escriure petits programes amb molta facilitat i provar-los ràpidament, ja que ens estalviem el procés de compilació. Com a contrapartida, l'execució és molt més lenta, i és per això que, per a aplicacions que requereixen un gran esforç de càlcul, els llenguatges compilats com C o Fortran són més adients (o, de vegades, l'única opció).

Podeu comprovar aquest fet molt fàcilment comparant el temps d'execució de les següents línies de codi Octave

```
n=2000000
t0=cputime() ; a=0; for i=1:n a+=1 ; endfor ; t1=cputime();
t1-t0
```

contra el programa en C de la figura 1. Suposant que l'executable corresponent es diu `suma`, podeu esbrinar el temps d'execució teclejant a una terminal:

```
time ./suma 1000000000
```

Observareu que el programa C va de l'ordre de 1000 vegades més ràpid.

Això no sempre és així. Suposem que volem multiplicar dues matrius quadrades grosses. Podeu prendre mesures del temps d'execució de la rutina `mmul()` amb el programa

[2]Matlab significa "Matrix Laboratory".

```
#include <stdio.h>

int main (int argc, char *argv[]) {
   double a; int i, n;
   if (argc<2
          || sscanf(argv[1], "%d", &n)!=1
      ) {
      fprintf(stderr,"%s n \n", argv[0]);
      return -1;
   }
   a=0; for (i=0; i<n; i++) a+=1;
   printf("%.16G\n", a);
   return 0;
}
```

Figura 1: Programa per comparar C i Octave amb operacions no matricials.

`testmmul.c` que trobareu al campus virtual. Podeu generar l'executable corresponent amb `make testmmult`. Per a això, necessitareu els fitxers `cronometre.c` i `cronometre.h` que també heu de baixar del campus virtual. Una crida d'exemple és:

```
./testmmult 700=n 0=iblas
```

Incrementeu el paràmetre `n` fins que l'execució duri alguns segons.

Podeu comprovar fàcilment que el codi equivalent en Octave és molt més ràpid:

```
n=700 ; a=rand(n,n); b=rand(n,n);
t0=cputime(); c=a*b; t1=cputime(); t1-t0
```

El motiu és que, per tal de fer operacions amb matrius, Octave (i Matlab també) crida una biblioteca anomenada BLAS (Basic Linear Algebra Subprograms).

Aquesta biblioteca està escrita en Fortran (trobareu al campus virtual una quick reference card de l'interfície Fortran), però existeix una interfície C. El fitxer de capçalera corresponent és `/usr/include/cblas.h`. Si compareu amb la quick reference card, veureu que a cada rutina se li afegeix el prefix `cblas_` i, a les rutines que treballen amb matrius, un primer argument `order` que especifica si estan guardades per files (`CblasRowMajor`) o columnes (`CblasColMajor`). Com que nosaltres les guardem per columnes, sempre hem de fer servir `CblasColMajor` per a `order`. Així, podem cridar rutines d'aquesta biblioteca des dels nostres programes en C. De fet, `testmmult` ho fa si el crideu amb el paràmetre `iblas` igual a 1:

```
./testmmult 700=n 1=iblas
```

Podreu comprovar que ara fins i tot és una mica més ràpid que Octave.

El motiu pel qual la rutina `cblas_dgemm` és molt més ràpida que `mmul` és que està optimitzada d'acord amb l'estructura de la memòria caché del processador, i a més fa servir

les extensions vectorials SSE2 (Streaming SIMD Extensions 2) del vostre processador. Això passa amb totes les rutines de BLAS.