Librairie PETSc

PETSc (Portable Extensible Toolkit for Scientific computation) est une librairie de fonctions mathématiques pour la résolution des équations aux dérivées partielles. (Très utilisée en calcul parallèle
avec MPI -Message Passing Interface-). Elle est principalement réservée aux langages C, C++ et Fortran. Elle est développée par la
MCS Division
. On utilisera cette librairie lors des projets de MACS 2. Il est par ailleurs recommandé de programmer en PETSc sur Linux plutôt que sur Windows, c'est pour cela que nous n'expliquerons pas comment faire du PETSc sur Windows.

Le site : http://www-unix.mcs.anl.gov/petsc/petsc-as/

Documentation dans votre répertoire PETSc:
file:///-chemin-d'accès-au-répertoire-petsc-/docs/manualpages/singleindex.html

(adresse à taper dans votre navigateur web)

Installation sur linux

- 1. télécharger la dernière version ici , prendre la 2.3.2 qui est stable.

- 2. extraire vers un répertoire (conseil: mettre dans le répertoire /opt/)

- 3. Il faut d'abord export la variable PETSC_DIR :

$ export PETSC_DIR=/chemin-du-repertoire/petsc-2.X.X

- 4. Configurer petsc:

Voici un exemple de ./config, pour voir les options de configuration : **./config/configure.py –help**

```
$ cd $PETSC_DIR
$ ./config/configure.py --with-cc=gcc --with-fc=g77 --download-f-blas-lapack=1 --with-mpi=0 --with-shared
```

- Installation de la compilation en paralèlle: remplacer –with-mpi=0 par –download-mpich=1
- Si jamais vous avez mis petsc dans un dossier où vous n'avez pas les droits (i.e. ailleurs que votre home) alors vous pouvez taper sudo chown user $PETSC_DIR avec *user=votre-identifiant*
pour avoir les droits (seulement si nécessaire).

Si cela ne fonctionne pas : remplacer les "1" par des "yes"

- 5. lire quel PETSC_ARCH faire, si ce n'est pas écrit, essayer :

```
$ PETSC_ARCH=linux; export PETSC_ARCH
```

- 6. entrer:

```
$ make
$ make test
```

Voila si le make test vous affiche "succesfull" l'installation s'est bien déroulée.

Compilation

Pour compiler un programme C utilisant la librairie PETSc, il faut au préalable inclure dans le makefile la variable d'environnement du chemin d'accès au répertoire PETSc. Puis avant la construction des cibles, il faut appeler les instructions de PETSc. Enfin, il ne faut pas oublier d'inclure le chemin de la bibliothèque des fonctions de PETSc utilisées.

```
export PETSC_DIR=/-chemin-du-répertoire-/petsc-2.3.2-p10  #-variable d'environnement
include ${PETSC_DIR}/bmake/common/base    #-instructions petsc
                          #-> par exemple %.o: %.c  pour la création automatique des objets
                               #-> autre exemple : les macros des bibliothèques
projet: projet.o
 gcc -o projet  projet.o ${PETSC_KSP_LIB}  #-bibliothèque contenant les fonctions KSP (Mat et Vec)
```

Tutorial de démarrage

Tutorial de premiers pas en PETSc.
Vous apprendrez ici à écrire un programme de base utilisant la librairie PETSc. Vous trouverez

les commandes de bases pour la gestion des vecteurs et des matrices. Enfin, vous verrez comment résoudre un système de type Ax=b.
Bien sûr cette explication est non-exhaustive, elle a pour but de vous donnez un point de départ stable avant de vous lancer vous-mêmes.

I. Syntaxe

Pour utiliser les commandes et fonctions PETSc, il faut inclure un .header. (ex : *#include "petscvec.h"*).
Pour savoir quel fichier appeler, il faut utiliser l'index de son répertoire petsc :

file:///-chemin-d'accès-au-répertoire-petsc-/docs/manualpages/singleindex.html

En général, deux inclusions suffisent puisque certains .header en appellent d'autres. Il suffit de vérifier sur le site par exemple. Les " " indiquent que la librairie se trouve dans le répertoire courant (voir compilation).

Chaque programme utilisant PETSc doit contenir un variable globale, une méthode PETSc d'initialisation, et une de fermeture. Généralement, ce type de programme ressemble à :

```
  ...
static char help []  = "blablan";
...
int main(int argc,char **args)
{
 PetscInitialize(&argc,&args,(char *)0,help);
 ...
 PetscFinalize();
 return 0;
 }
```

Un objet de type PetscErrorCode est le plus souvent créé. A chaque appel d'une fonction de PETSc, on peut stocker le résultat de l'exécution dans cet objet (les méthodes PETSc ne renvoient rien sinon). Ainsi, lorsqu'une erreur est détectée, on peut la localiser directement. Son utilisation n'est pas nécessaire, mais elle se révèle assez rapidement utile.

```
  PetscErrorCode ierr;
...
ierr = PetscFinalize();CHKERRQ(ierr);
```

II. Création de vecteurs et de matrices

(Tout objet construit doit être détruit, cf gestion de la mémoire en C.)

Les vecteurs

Ils s'utilisent avec *petscvec.h*, objet de type *Vec*.
Trois commandes sont nécessaires à la création d'un vecteur, et une pour la destruction. Par exemple pour fabriquer un vecteur b de dimension dim, mettre la valeur val dans sa première coordonnée, le visionner, et le supprimer, il faut taper :

```
...
Vec b;
...
VecCreate(PETSC_COMM_WORLD,&b);
VecSetSizes(b,PETSC_DECIDE,dim);
VecSetFromOptions(b);
...
VecSetValue(b,0,val,INSERT_VALUES); //les tableaux commencent à l'indice 0
VecView(b,PETSC_VIEWER_STDOUT_SELF);
...
VecDestroy(b);
...
```

Les matrices

Ils s'utilisent avec *petscmat.h*, objet de type *Mat*.
De même, trois commandes sont nécessaires à la création d'une matrice, et une pour la destruction. Par exemple pour fabriquer une matrice A de dimension dim1*dim2, mettre la valeur val dans sa première ligne et deuxième colonne, la visionner, et la supprimer, il faut taper :

```
...
Mat A;
...
MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,dim1,dim2);
MatSetFromOptions(A);
...
MatSetValue(A,0,1,val,INSERT_VALUES);
MatView(A,PETSC_VIEWER_STDOUT_SELF);
...
MatDestroy(A);
...
```

III. Résolution d'un système Ax=b

(A matrice carrée, b et x vecteurs colonnes)

Quelques commandes utiles pour gérer les vecteurs :
* Pour créer un second vecteur de même taille que le premier, avec les mêmes options, sans écrire les trois lignes de création : *VecDuplicate(Vec , *Vec)*
* Pour remplir un vecteur avec une même valeur pour tous ses coefficients : *VecSet(Vec, PetscScalar)*
* Pour faire une multiplication matrice-vecteur classique : *MatMult(Mat, Vec , Vec)*
* Pour faire une addition vecteur-vecteur : *VecWAXPY(Vec, float, Vec, Vec)*
* Pour calculer la norme d'un vecteur : *VecNorm(Vec ,NormType)*, (pour la version 2.3.2-p10)

Dans notre programme :

```
  Vec b,x,c,diff;
Mat A;
float val=1.2,res;
...
VecDuplicate(b,&x);
VecSet(b,val);
...
MatMult(A,x,test);  //-- c=Ax
VecWAXPY(diff,-1,c,b);  //-- diff=-1*c+b
res = VecNorm(diff,NORM_INFINITY); //-version 2.3.2-p10
...
```

Pour résoudre un système Ax=b avec A une matrice carrée, b et x deux vecteurs colonnes, il est recommandé d'utiliser les commandes KSP de la librairie PETSc :

```
...
KSP ksp;
  ...
  KSPCreate(PETSC_COMM_WORLD,&ksp);
  KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
 //-- A est stockée dans KSP
  KSPSetTolerances(ksp,1.e-14,PETSC_DEFAULT,PETSC_DEFAULT,PETSC_DEFAULT);
 //-- la tolérance souhaitée
  KSPSetFromOptions(ksp);
  KSPSolve(ksp,b,x);  //-- le résultat est dans le vecteur x : Ax=b
```

Quelques commandes utiles de PETSc

VecGetArray

Récupérer les valeurs d'un vecteur : *VecGetArray(Vec, PetscScalar*)*
Il faur créer un tableau pour le stockage. (Il est impossible de récupérer une seule valeur)

```
...
Vec v;
PetscScalar *tab;
...
VecGetArray(v,&tab);
...
```

VecScale

Multiplier tous les coefficients d'un vecteur par un scalaire : *VecScale(Vec, float)*

```
...
Vec v;
float alpha;
...
VecScale(v,alpha);
...
```

Toutes les commandes sont répertoriées sur l'index du site , mais aussi sur la documentation dans votre répertoire PETSc

Un TD

FEM1D : projet de résolution EDP en 1D avec les éléments finis, en petsc avec interfacage Matlab ou Scilab (auteur: Cuvelier)