「ライブラリ利用：高性能プログラミング初級入門」 Part Ⅱ

2011年12月22日（木）

東京大学 情報基盤センター
伊藤祥司
itosho@cc.u-tokyo.ac.jp

当講習会の目標

1. 60分後には，線形方程式に対して，100種類近くのアルゴリズムで求解できるようになる．
2. 90分後には，線形方程式に対して，並列計算による求解ができるようになる．
3. 180分後には，反復解法に基づく求解アルゴリズムの特徴と注意点を理解している．

こんな夢のようなことを実現するのが当講習会！

その鍵となるのが・・・ Lisだ！

目　次

前半の内容

後半の内容

【　】：当資料中のスライド番号

0. 準備：Lisとテスト問題他の入手

◆ Lis [http://www.ssisc.org/lis/]

→ Downloadの「Version 1.2.xx」からソースコード，ユーザマニュアルをダウンロード
※ ソースコードは次画面の「ダウンロードページ」
→「Software License Agreement」へと進む．
ユーザ登録する／しないは各自の自由

※ 当資料，当講習会では，lis-1.2.49 版を前提として説明，実習します．

◆ テスト問題 Matrix Market [http://math.nist.gov/MatrixMarket]

→ BrowseやSearchの項目のリンクをクリック

◆ SESNA : Survey and Evaluation System for Numerical Algorithms [http://sesna.jp]
求解アルゴリズムの体系的性能評価システム
→ Matrix Market の一部問題を Lisの一部を用いて求解した結果情報のDBと求解結果の評価システム

1. HA8000での実習環境設定(1/3)

各自の$HOME下に実習用sampleファイルー式をコピー:

```
$ cd   [Enter]                                  # ホームディレクトリ($HOME)へ移動
$ cp -rp /home/t00003/sample  .                 # 実習用sampleファイルー式をコピー
$ ls -FR sample
sample:
library/  mat/  qsub_script/  setup.env  sh/  test/

sample/library:                                 # 今回の実習で使用するライブラリ等
lis-1.2.49.tar.gz          petsc-lite-3.1-p3.tar.gz
lis-manual-1.2.49_ja.pdf   petsc-manual-3.1.pdf

sample/mat:                                     # テスト用入力データ(係数行列, 右辺項データ)
494_bus.mtx      add20.rua        add32_rhs2.mtx  memplus.mtx        nos3.rsa
494_bus.rsa      add32.mtx        add32.rua       memplus_rhs1.mtx
add20.mtx        add32_rhs1.dat   fs_183_1.mtx    memplus.rua
add20_rhs1.mtx   add32_rhs1.mtx   fs_183_1.rua    nos3.mtx

sample/qsub_script:                             # JOB実行用スクリプト
qsub_mpi.nqs   qsub_omp.nqs   qsub_seq.nqs

sample/sh:                                      # 今回の実習で役立つユーティリティ
cdlis_kernel.sh*   morelis*

sample/test:                                    # Lis実行で使用するmainルーチン
test1_mod2.c   test1_mod.c   test1_org.c
```

以後 [Enter]キー
押下の記述は省略

- コマンド実行する部分は太字で表示してます(左端の$はUNIXのコマンドプロンプト).
- 画面表示は, 多少, 文字サイズを小さくしてます(当テキスト編集の都合上).
- 特に注意の無い限り, # 記号から右部分の青文字はコメントです.

1. HA8000での実習環境設定(2/3)

各自の$HOME/sample (~/sample)下で実習用環境をセットアップ:

```
$ cd sample
$ ls
library  mat  qsub_script  setup.env  sh  test

$ source setup.env                              # 実習環境のセットアップ
```

ログインの度に
実行する(☆1)

```
$ cat setup.env                                 # このような環境を設定
alias ls='ls -F'
alias lrt='ls -lrt'
alias cp='cp -i'
alias mv='mv -i'
alias rm='rm -i'
alias cdlis='source $HOME/sample/sh/cdlis_kernel.sh'
PATH=$PATH:$HOME/sample/sh
#
#   20100912 created  by itosho (ITOH, Shoji)
#
```


1. HA8000での実習環境設定(3/3)

各自のNFS下へ実習環境を構築(T2K TODAI固有の環境セットアップ):

```
$ cd                                            # ホームディレクトリ($HOME)へ移動
$ pwd
/home/{各自の利用者番号}

$ ln -s  /nfs/all/$USER   nfs                   # 各自のnfsのディレクトリをシンボリックリンク
                                                # /nfs/all/下への毎回の移動は面倒
   # 以後, {各自の利用者番号}を t00003 として説明

$ ls -l nfs
lrwxrwxrwx 1 t00003 t00 15 Sep 14 14:17 nfs -> /nfs/all/t00003/

$ cd  nfs
$ pwd                                           # /home配下であることを確認
/home/t00003/nfs
```


2. Lisインストールの準備作業(1/2)

Lis実習環境の準備(後半のtest1.cに関する作業は, 当実習ならではの事柄):

```
$ tar  zxf  $HOME/sample/library/lis-1.2.49.tar.gz
   # その他の方法の一例として:
  ( $ gunzip -c $HOME/sample/library/lis-1.2.49.tar.gz | tar xf - )

$ ls
lis-1.2.49/

$ cd  lis-1.2.49/test/
$ ls
defs*      etest4.c     matfiles.txt  spmvtest5.c  test4.c      test.sh*
etest1.c   etest4f.F    spmvtest1.c   test1.c      test4f.F
etest1f.F  etest5.c     spmvtest2.c   test1f.F     test5.c
etest2.c   Makefile.am  spmvtest3.c   test2.c      test6.c
etest3.c   Makefile.in  spmvtest4.c   test3.c      testmat.mtx
$ ls  $HOME/sample/test/
test1_mod2.c  test1_mod.c  test1_org.c              # test1_org.c と test1.c とは同一

$ cp  -p  $HOME/sample/test/test1_mod.c  .    # ここでは test1_mod.c のみコピー

$ mv      test1.c      test1_org.c
$ ln  -s  test1_org.c  test1.c                  # サンプルの test1.c を準備
$ ls  -l  test1.c
lrwxrwxrwx 1 t00003 t00 11 Sep 14 14:55 test1.c -> test1_org.c

$ cd  $HOME/nfs
```

2. Lisインストールの準備作業(2/2)

Lis逐次(seq)版の準備:

```
$ ls
lis-1.2.49/

$ cp -rp lis-1.2.49 lis-1.2.49_seq                    # Lis逐次(seq)版の環境準備

$ cd lis-1.2.49_seq
$ ls
aclocal.m4  config/       COPYING   INSTALL      NEWS    test/
AUTHORS     configure*    doc/      Makefile.am  README  win32/
ChangeLog   configure.in  include/  Makefile.in  src/

     # 今回の実習では，Intelコンパイラを使用する

$ source  /opt/itc/mpi/mpiswitch.sh  mpich-mx-intel11   # Intelコンパイラの環境設定
$ which icc ifort
/opt/intel/Compiler/11.0/074/bin/intel64/icc                  # Intelコンパイラのiccのパス
/opt/intel/Compiler/11.0/074/bin/intel64/ifort                # Intelコンパイラのifortのパス
```

ログインの度に実行する(☆2)

- 以後，コンパイラ等の処理系のバージョンアップに起因して，出力情報が異なる場合もありますが，それらは適宜読み替えて下さい．
- ただし，コンパイラ等のバージョンが異なることによる非互換などはあり得ます．

3. Lis逐次版のインストール(1/3)

Lis逐次(seq)版のconfigure:

```
$ pwd
/home/t00003/nfs/lis-1.2.49_seq
$ ./configure  --enable-saamg                          # SAAMG前処理を有効にする
  ・・・ 途中省略 ・・・

  ・・・ 出力の最後部 ・・・

Using
  C   Compiler      = icc
  C   Flags         =  -DHAVE_CONFIG_H -O3 -ansi_alias
  C   Libs          =
      Libs          =   -lm
  F90 Compiler      = ifort
  F90 Flags         = -O3 -fpp -g -DZERO_ORIGIN=1
  F90 Libs          =
  enable Fortran Wrapper          = no
  enable SA-AMG Preconditioner    = yes
  enable MPI                      = no
  enable OpenMP                   = no
  enable Quad Precision           = no
  MPICC                 =
  MPIF77                =
  MPIFC                 =
  MPILIBS               =
  MPIRUN                =
  MPFLAG                =
```

- Lisを正式にインストールする場合，--prefix オプションでディレクトリも指定．

赤文字の5項目をチェック

- configure実行後は，作業ディレクトリやパスを変更しないように(生成されるMakefile中では絶対パスが用いられている)．

3. Lis逐次版のインストール(2/3)

Lis逐次(seq)版のmake:

```
$ make

  ・・・  途中省略  ・・・

  ・・・ 出力の最後部 ・・・

/bin/sh ../libtool --mode=link ifort  -Vaxlib -nofor_main -o spmvtest5 -lm  spmvtest5.o -
L../src -llis -lm
libtool: link: ifort -Vaxlib -nofor_main -o spmvtest5 spmvtest5.o  -L/home/t00003/nfs/lis-
1.2.49_seq/src /home/t00003/nfs/lis-1.2.49_seq/src/.libs/liblis.a -lm
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_seq/test'
Making all in win32
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_seq/win32'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_seq/win32'
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_seq'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_seq'
```

最後部でErrorが出ていなければOK

3. Lis逐次版のインストール(3/3)

Lis逐次(seq)版のインストールチェック(make check):

```
$ make check

  ・・・ 出力の途中部分 ・・・

make  check-TESTS
make[2]: Entering directory `/nfs/all/t00003/lis-1.2.49_seq/test'
=== Running test test.sh

checking solvers...

number of processes = 1
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : BiCG 2
PRECON    : None
STORAGE   : CRS
lis_solve : normal end

BiCG: iter      = 15 iter_double = 15 iter_quad = 0
BiCG: times     = 2.660751e-04
BiCG: p_times   = 1.788139e-05 (p_c = 2.145767e-06 p_i = 1.573563e-05 )
BiCG: i_times   = 2.481937e-04
BiCG: residual = 6.399933e-15

checking eigensolvers...

 ・・・
```

赤文字の項目をチェック

下線の項目は計測時間なので，一致して無くても良い

3. Lis逐次版のインストール(補足)

当実習の中では，make installは実行しなくても利用可能：

$ make install

マニュアルには，最後に上記を実行するように記述されているが，
これは，正式にシステムにインストールする(例えば，/usr/local/下など)
場合に実行すれば良い．

この実習では，既に作成されている“test1 ”というプログラムを
用いて説明するため，特にインストール先を指定しないでも良い．
現状で，生成済みの関連ファイル群は下記のディレクトリ構成となっている：

```
$HOME/nfs/lis-1.2.49_seq/ 下:
        include/
             config.h, lis.h, lisf.h
        src/.libs/
             liblis.a
```

当実習では不要！

インストール先のデフォルトは /usr/local/ 下であるが，
それを変更する場合には，configure のステップまで戻って，

$ ./configure --prefix={インストールディレクトリ}

とする．

4. 補足事項：main関数の入替え

当実習ならではの事柄だが，反復法の収束判定に関する重要事項：

```
$ cd  test
$ ls -l test1*.c
lrwxrwxrwx 1 t00003 t00   11 Sep 14 16:43 test1.c -> test1_org.c      # test1.cのリンク先
-rw-r--r-- 1 t00003 t00 5309 Sep 12 16:01 test1_mod.c
-rw-r--r-- 1 t00003 t00 5157 Jun 18 10:48 test1_org.c

$ mv test1 test1_org ; ls -l test1_org                                # 必須ではないが
-rwxr-xr-x 1 t00003 t00 1669881 Sep 14 16:50 test1_org*

$ rm test1.c test1.o ; ln -s test1_mod.c test1.c                      # main関数の入替え
rm: remove symbolic link `test1.c'? y
rm: remove regular file `test1.o'? y
$ ls -l test1*.c
lrwxrwxrwx 1 t00003 t00   11 Sep 14 16:53 test1.c -> test1_mod.c      # test1.cのリンク先
-rw-r--r-- 1 t00003 t00 5309 Sep 12 16:01 test1_mod.c
-rw-r--r-- 1 t00003 t00 5157 Jun 18 10:48 test1_org.c

$ make
icc  -DHAVE_CONFIG_H -I. -I../include  -I.. -I../include   -O3 -ansi_alias  -c test1.c
/bin/sh ../libtool --mode=link ifort  -Vaxlib -nofor_main -o test1 -lm  test1.o -L../src
-llis -lm
libtool: link: ifort -Vaxlib -nofor_main -o test1 test1.o  -L/home/t00003/nfs/lis-
1.2.49_seq/src /home/t00003/nfs/lis-1.2.49_seq/src/.libs/liblis.a -lm

$ ls -l test1 test1_org
-rwxr-xr-x 1 t00003 t00 1669905 Sep 14 16:54 test1*
-rwxr-xr-x 1 t00003 t00 1669881 Sep 14 16:50 test1_org*
```


5. ジョブ実行スクリプトの用意(逐次版)

HA8000では，PBS(NQSの一種)でジョブ実行するスクリプトを使用する：

```
$ ls  $HOME/sample/qsub_script/
qsub_mpi.nqs  qsub_omp.nqs  qsub_seq.nqs

$ cp  -p $HOME/sample/qsub_script/qsub_seq.nqs .
$ cat qsub_seq.nqs
#@$-q   lecture
#@$-N   1
#@$-lm 2GB
#@$-lT 0:01:00
#@$-o   OUT            # 標準出力のファイル名
#@$-e   ERR            # 標準エラー出力のファイル名
#@$-s   /bin/bash      # スクリプト内のシェルを指定

cd $PBS_O_WORKDIR      # スクリプト内もカレントディレクトリへ移動する
pwd                    # pwdは本来は不要だが，確認のため

./test1 testmat.mtx 0 sol1.txt res1.txt                 # 求解1(デフォルト: -i bicg, -p none)
./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1       # 求解2
#./test1 ./mat/add32.mtx 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.mtx ./mat/add32_rhs1.dat sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.rua 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/nos3.mtx 2 sol.txt res.txt -i 5 -p 2

# see manual subsection 3.4
#
#   20100914 created  by itosho (ITOH, Shoji)
#
```

講習会後のキュー名．本日は tutorial (sampleは変更済み)

● スクリプトの拡張子は任意

● NQSのスクリプトでは左端の“ #@$ ”記号は，これに続くハイフンで始まる文字列がNQSオプションであることを宣言している．

● 赤文字の項目をチェック(NQSオプション“-N”でノード数を指定している)

5. HA8000でのジョブ実行(逐次版)

ジョブ実行(qsub)と実行中・実行後の状態確認方法：

```
$ qsub qsub_seq.nqs
Request 606870.batch1 submitted to queue: lecture.
        リクエストID(ジョブ固有の番号)
$ qstat                                              # ジョブ実行中の表示例
2010/09/14 (Tue) 19:30:26:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 253h 24m 34s)
   REQUEST      NAME      OWNER      QUEUE     PRI NODE E-TIME    MEM     STATE
606870.batch1   qsub_se   t00003     lecture   63    1     60s     28GB   RUNNING

$ qstat                                              # ジョブ終了後の表示例
2010/09/14 (Tue) 19:30:37:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 253h 24m 23s)
   REQUEST      NAME      OWNER      QUEUE     PRI NODE E-TIME    MEM     STATE
No requests.

$ type lrt
lrt is aliased to `ls -lrt'
$ lrt
 ･･･ 省略 ･･･
-rw------- 1 t00003 t00       0 Sep 14 19:30 ERR       # 標準エラー出力のファイル
-rw-r--r-- 1 t00003 t00    3242 Sep 14 19:30 sol2.txt  # 解ベクトル2(solution)
-rw-r--r-- 1 t00003 t00    3242 Sep 14 19:30 sol1.txt  # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 t00     208 Sep 14 19:30 res2.txt   # 相対残差2(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 t00     208 Sep 14 19:30 res1.txt   # 相対残差1(residual)のノルム収束履歴
-rw------- 1 t00003 t00     864 Sep 14 19:30 OUT       # 標準出力のファイル
```

5. 実行したジョブの結果確認(逐次版)

※ 編集の都合上，出力の一部を折り返し表示している．

```
$ cat OUT
/home/t00003/nfs/lis-1.2.49_seq/test
                    # 確認のpwdの出力
number of processes = 1
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : BiCG 2
PRECON    : None
STORAGE   : CRS
lis_solve : normal end

BiCG: iter      = 15 iter_double = 15
iter_quad = 0
BiCG: times     = 1.130104e-04
BiCG: p_times   = 1.811981e-05 (p_c =
9.536743e-07 p_i = 1.716614e-05 )
BiCG: i_times   = 9.489059e-05
BiCG: residual = 6.399933e-15

BiCG: rel_resid = 9.237507e-16
```

```
number of processes = 1
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : CGS 3
PRECON    : Jacobi
STORAGE   : CRS
lis_solve : normal end

CGS: iter      = 15 iter_double = 15
iter_quad = 0
CGS: times     = 1.420975e-04
CGS: p_times   = 3.552437e-05 (p_c =
9.536743e-07 p_i = 3.457069e-05 )
CGS: i_times   = 1.065731e-04
CGS: residual = 1.091864e-14

CGS: rel_resid = 1.575969e-15
```

```
./test1 testmat.mtx 0 sol1.txt res1.txt
# 求解1の結果（デフォルト: -i bicg, -p none）
```

```
./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1
# 求解2の結果
```


6. いま，Lisで何をやったのだろうか

```
$ ./test1
Usage: test1 matrix_filename rhs_setting solution_filename residual_filename [options]

matrix_filename:    係数行列のファイルを指定
rhs_setting:        右辺項ベクトルファイル，あるいは，自動生成機能（主にアルゴリズムの性能評価用途）のIDを指定
                      0 : 拡張Matrix Market形式（右辺ベクトルが行列データに含まれている）を用いる
                      1 : b = (1,1,...,1)^T   を用いる
                      2 : b = A ×(1,1,...,1)^T   を用いる
solution_filename:  解ベクトルのファイル名
residual_filename:  （アルゴリズム中のベクトルを用いた）相対残差ベクトルの2ノルムの履歴
[options]:          解法と前処理，その他のオプション指定

# 参考: マニュアル 「テストプログラム」の「test1」小節
```

```
$ more testmat.mtx
%%MatrixMarket matrix coordinate real general
100 100 460 1 0
1 2 -1.0000000000000000000e+000
1 11 -1.0000000000000000000e+000
1 1  4.0000000000000000000e+000
2 3 -1.0000000000000000000e+000
2 12 -1.0000000000000000000e+000
2 1 -1.0000000000000000000e+000
2 2  4.0000000000000000000e+000
 ...
```

参考1: マニュアル 付録「ファイル形式」の「拡張Matrix Market形式」小節
testmat.mtx も係数行列と右辺項の両方のデータを格納した拡張形式．
参考2:
http://math.nist.gov/MatrixMarket/formats.html

6. 色々な解法で解いてみよう

解法 デフォルトはBiCG	選択オプション { a | n} : a, nどちらでも良い	補助オプション [ ]内はデフォルト値
CG	-i {cg | 1}	
BiCG	-i {bicg | 2}	
CGS	-i {cgs | 3}	
BiCGSTAB	-i {bicgstab | 4}	
BiCGSTAB(l)	-i {bicgstabl | 5}	-ell [2]　解法の l の値
GPBiCG	-i {gpbicg | 6}	
TFQMR	-i {tfqmr | 7}	
Orthomin(m)	-i {orthomin | 8}	-restart [40]　リスタート m の値
GMRES(m)	-i {gmres | 9}	-restart [40]　リスタート m の値
Jacobi	-i {jacobi | 10}	
Gauss-Seidel	-i {gs | 11}	
SOR	-i {sor | 12}	-omega [1.9]　緩和係数ωの値(0<ω<2)

● 上記も含め，全22種類の解法が提供されている．

6. 色々な前処理を組合せてみよう

前処理 デフォルトは“なし”	選択オプション { a | n} : a, nどちらでも良い	補助オプション [ ]内はデフォルト値
なし	-p {none | 0}	
Jacobi	-p {jacobi | 1}	
ILU(k)	-p {ilu | 2}	-ilu_fill [0]　フィルインレベル k
SSOR	-p {ssor | 3}	-ssor_w [1.0]　緩和係数ωの値(0<ω<2)
Hybrid	-p {hybrid | 4}	-hybrid_i [sor]　線型方程式系解法，他
I+S	-p {is | 5}	-is_alpha [1.0]　前処理中の変数α，他
SAINV	-p {sainv | 6}	-sainv_drop [0.05]　ドロップ基準
SA-AMG	-p {saamg | 7}	-saamg_unsym [false]　非対称版選択，他
Crout ILU	-p {iluc | 8}	-iluc_drop [0.05]　ドロップ基準，他
ILUT	-p {ilut | 9}	-ilut_drop [0.05]　ドロップ基準，他
Additive Schwarz	-adds true	-adds_iter [1]　繰り返し回数

● 上記の補助オプションは一部のみ掲載している．詳細はマニュアル参照．

6. 色々なオプションを試してみよう

オプション [ ]内はデフォルト値	内容
-maxiter [1000]	最大反復回数
-tol [1.0e-12]	収束判定基準(相対残差)
-print [0]	残差の画面表示
-scale [0]	スケーリングの選択. ※ スケーリングされた係数行列，右辺項ベクトルは，元のデータに上書きされて，プログラムで実行される.
-initx_zeros [true]	初期ベクトル x_0 の設定(デフォルトは全要素の値が0)
-omp_num_threads [t]	実行スレッド数. t は最大スレッド数
-storage [0]	行列格納形式
-storage_block [2]	BSR，BSCのブロックサイズ

● 上記のオプションの内容の詳細はマニュアル参照.

6. 色々なデータで実行してみよう

```
$ pwd
/home/t00003/nfs/lis-1.2.49_seq/test
$ ls  $HOME/sample/
library/  mat/  qsub_script/  setup.env  sh/  test/

$ ln  -s  $HOME/sample/mat .
$ ls  -l  mat
lrwxrwxrwx 1 t00003 t00 23 Sep 14 19:39 mat -> /home/t00003/sample/mat/
$ ls mat/
494_bus.mtx     add20.rua       add32_rhs2.mtx  memplus.mtx       nos3.rsa
494_bus.rsa     add32.mtx       add32.rua       memplus_rhs1.mtx
add20.mtx       add32_rhs1.dat  fs_183_1.mtx    memplus.rua
add20_rhs1.mtx  add32_rhs1.mtx  fs_183_1.rua    nos3.mtx

参考までに，matの後にディレクトリを示す「/」を付け忘れると:
$ ls mat      # これは，☆1のsetup.env 影響
mat@
```

● 拡張子
mtx : Matrix Market形式
rua, rsa : Harwell-Boeing形式
dat : ベクトルのPLAIN形式
(拡張子自体は任意)

```
$ cat qsub_seq.nqs
#./test1 ./mat/add32.mtx 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.mtx ./mat/add32_rhs1.dat sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.rua 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/nos3.mtx 2 sol.txt res.txt -i 5 -p 2
```

● 各々を実行するときは，左端冒頭の「#」(コメントアウト)記号を削除する.

7. 入力データ形式について

● Matrix Market(MM)形式は，データの圧縮度は低いものの，行列のインデックスと値との対応を確認し易く，また，多くのアプリケーションでもMM形式をサポートしたりツール提供されているので，初級者や中級者にとっては都合が良いと思われる.

● Lisのtest1.cでは，MM形式のデータを読込み，CRS形式に変換する. MPIのデータ分散にも対応している.

● 対称行列の場合には，対角要素から下三角の要素のみが格納されている. この場合，ヘッダには，半角スペースに続いて「symmetric」の文字列が記載されている.
→ mat/下の「494_bus.mtx」「nos3.mtx」が該当

● 非対称行列の場合には，「general」の文字列が記載されている.
→ mat/下の「add{20,32}.mtx」「fs_183_1.mtx」「memplus.mtx」が該当

```
add32.mtx       : MMサイトで提供されているMM形式の係数行列データ
add32_rhs1.mtx  :      〃            MM形式の右辺項ベクトルデータ
add32.rua       :      〃            Harwell-Boeing(HB)形式の係数行列データ

 MMサイトで提供されている add32_rhs1.mtx のヘッダとフォーマットでは，Lisで読み取れないので，
 当実習では以下の2通りの修正データを用意してます. 問題点と修正内容は，各自で確認して下さい.
 add20, memplus等も同様ですが，サンプルではadd32のみ修正データを用意しています.

add32_rhs1.dat : add32_rhs1.mtxを元にPLAIN形式にしたファイル
add32_rhs2.mtx : Lisで読み取り可能なベクトル用拡張MM形式のデータ
```


8. Lis並列(OMP, MPI)版の準備作業

逐次版と同様，main関数のtest1.c を入替える(当実習ならではの事柄):

```
$ cd  $HOME/nfs/lis-1.2.49/test/                  # lis-1.2.49_seq ではないことに注意
$ ls -l test1*.c
lrwxrwxrwx 1 t00003 t00   11 Sep 14 14:55 test1.c -> test1_org.c
-rw-r--r-- 1 t00003 t00 5309 Sep 12 16:01 test1_mod.c
-rw-r--r-- 1 t00003 t00 5157 Jun 18 10:48 test1_org.c

$ rm test1.c ; ln -s test1_mod.c test1.c          # 逐次版同様に test1.c を入替える
rm: remove symbolic link `test1.c'? y
$ ls -l test1*.c
lrwxrwxrwx 1 t00003 t00   11 Sep 15 15:06 test1.c -> test1_mod.c
-rw-r--r-- 1 t00003 t00 5309 Sep 12 16:01 test1_mod.c
-rw-r--r-- 1 t00003 t00 5157 Jun 18 10:48 test1_org.c

$ ln  -s  $HOME/sample/mat .                      # ついでにmatもシンボリックリンク
$ cd  $HOME/nfs
$ cp -rp  lis-1.2.49  lis-1.2.49_omp              # LisのOpenMP(omp)版の環境準備
$ cp -rp  lis-1.2.49  lis-1.2.49_mpi              # LisのMPI(mpi)版の環境準備

$ which icc ifort
/opt/intel/Compiler/11.0/074/bin/intel64/icc      # Intelコンパイラのiccのパス
/opt/intel/Compiler/11.0/074/bin/intel64/ifort    # Intelコンパイラのifortのパス

    # Intelコンパイラの環境設定されていない場合には:
$ source  /opt/itc/mpi/mpiswitch.sh  mpich-mx-intel11   # Intelコンパイラの環境設定

$ cd  lis-1.2.49_omp
```

8. Lis並列(OMP)版のインストール(1/3)

Lis並列(omp)版のconfigure:

```
$ pwd
/home/t00003/nfs/lis-1.2.49_omp
$ ./configure  --enable-omp  --enable-saamg            # ここでompを指定する
  ・・・ 途中省略 ・・・

  ・・・ 出力の最後部 ・・・

Using
  C   Compiler      = icc
  C   Flags         =  -DHAVE_CONFIG_H -O3 -ansi_alias  -openmp
  C   Libs          =
      Libs          =   -lm
  F90 Compiler      = ifort
  F90 Flags         = -O3 -fpp -g -DZERO_ORIGIN=1 -openmp
  F90 Libs          =
  enable Fortran Wrapper         = no
  enable SA-AMG Preconditioner   = yes
  enable MPI                     = no
  enable OpenMP                  = yes
  enable Quad Precision          = no
  MPICC                =
  MPIF77               =
  MPIFC                =
  MPILIBS              =
  MPIRUN               =
  MPFLAG               =
```

● Lisを正式にインストールする場合，--prefix オプションでディレクトリも指定.

赤文字の6項目をチェック

● configure実行後は，作業ディレクトリやパスを変更しないように(生成されるMakefile中では絶対パスが用いられている).

8. Lis並列(OMP)版のインストール(2/3)

Lis並列(omp)版のmake:

```
$ make

  ・・・ 途中省略 ・・・

  ・・・ 出力の最後部 ・・・

/bin/sh ../libtool --mode=link ifort  -Vaxlib -openmp -nofor_main -o spmvtest5 -lm
spmvtest5.o -L../src -llis -lm
libtool: link: ifort -Vaxlib -openmp -nofor_main -o spmvtest5 spmvtest5.o  -
L/home/t00003/nfs/lis-1.2.49_omp/src /home/t00003/nfs/lis-1.2.49_omp/src/.libs/liblis.a -
lm
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_omp/test'
Making all in win32
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_omp/win32'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_omp/win32'
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_omp'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_omp'
```

最後部でErrorが出ていなければOK

8. Lis並列(OMP)版のインストール(3/3)

Lis並列(omp)版のインストールチェック(make check):

```
$ make check

  ・・・ 出力の途中部分 ・・・
make  check-TESTS
make[2]: Entering directory `/nfs/all/t00003/lis-1.2.49_omp/test'
=== Running test test.sh

checking solvers...

number of processes = 1
max number of threads = 16
number of threads = 2
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : BiCG 2
PRECON    : None
STORAGE   : CRS
lis_solve : normal end

BiCG: iter     = 15 iter_double = 15 iter_quad = 0
BiCG: times    = 1.162767e-03
BiCG: p_times  = 1.111031e-04 (p_c = 1.907349e-06 p_i = 1.091957e-04 )
BiCG: i_times  = 1.051664e-03
BiCG: residual = 8.589712e-15

BiCG: rel_resid = 1.239818e-15
 ・・・
```

赤文字の項目をチェック

下線の項目は計測時間なので，一致して無くても良い

8. ジョブ実行スクリプトの用意(OMP版)

HA8000では，PBS(NQSの一種)でジョブ実行するスクリプトを使用する:

```
$ cd test
$ ls  $HOME/sample/qsub_script/
qsub_mpi.nqs  qsub_omp.nqs  qsub_seq.nqs

$ cp  -p  $HOME/sample/qsub_script/qsub_omp.nqs .
$ cat qsub_omp.nqs
#@$-q  lecture
#@$-N  1
#@$-lm 2GB
#@$-lT 0:01:00
#@$-o  OUT            # 標準出力のファイル名
#@$-e  ERR            # 標準エラー出力のファイル名
#@$-s  /bin/bash      # スクリプト内のシェルを指定

export OMP_NUM_THREADS=4   # 環境変数によるスレッド数の指定
cd $PBS_O_WORKDIR     # スクリプト内もカレントディレクトリへ移動
pwd                   # pwdは本来は不要だが，確認のため

./test1 testmat.mtx 0 sol1.txt res1.txt                 # 求解1(デフォルト: -i bicg, -p none)
./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1       # 求解2
#./test1 ./mat/add32.mtx 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.mtx ./mat/add32_rhs1.dat sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.rua 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/nos3.mtx 2 sol.txt res.txt -i 5 -p 2

# see manual subsection 3.4
#   20100914 created  by itosho (ITOH, Shoji)
```

● スクリプトの拡張子は任意

● NQSのスクリプトでは左端の“ #@$ ”記号は，これに続くハイフンで始まる文字列がNQSオプションであることを宣言している.

● 赤文字の項目をチェック(NQSオプション“-N”でノード数を指定している)

8. HA8000でのジョブ実行(OMP版)

ジョブ実行(qsub)と実行中・実行後の状態確認方法:

```
$ qsub qsub_omp.nqs
Request 607327.batch1 submitted to queue: lecture.

$ qstat                                          # ジョブ実行中の表示例
2010/09/15 (Wed) 16:04:56:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 232h 50m  4s)
   REQUEST      NAME      OWNER      QUEUE      PRI NODE E-TIME    MEM    STATE
607327.batch1   qsub_om   t00003     lecture    63    1     60s     28GB  RUNNING

$ qstat                                          # ジョブ終了後の表示例
2010/09/15 (Wed) 16:05:06:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 232h 49m 54s)
   REQUEST      NAME      OWNER      QUEUE      PRI NODE E-TIME    MEM    STATE
No requests.

$ type lrt
lrt is aliased to `ls -lrt'
$ lrt
 ･･･ 省略 ･･･
-rw------- 1 t00003 t00        0 Sep 15 16:04 ERR       # 標準エラー出力のファイル
-rw-r--r-- 1 t00003 t00     3242 Sep 15 16:04 sol2.txt # 解ベクトル2(solution)
-rw-r--r-- 1 t00003 t00     3242 Sep 15 16:04 sol1.txt # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 t00      208 Sep 15 16:04 res2.txt  # 相対残差2(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 t00      208 Sep 15 16:04 res1.txt  # 相対残差1(residual)のノルム収束履歴
-rw------- 1 t00003 t00      962 Sep 15 16:04 OUT       # 標準出力のファイル
```


8. 実行したジョブの結果確認(OMP版)

※ 編集の都合上，出力の一部を折り返し表示している.

```
$ cat OUT
/home/t00003/nfs/lis-1.2.49_omp/test
                    # 確認のpwdの出力
number of processes = 1
max number of threads = 16
number of threads = 4
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : BiCG 2
PRECON    : None
STORAGE   : CRS
lis_solve : normal end

BiCG: iter     = 15 iter_double = 15
iter_quad = 0
BiCG: times    = 6.790161e-04
BiCG: p_times  = 8.130074e-05 (p_c =
9.536743e-07 p_i = 8.034706e-05 )
BiCG: i_times  = 5.977154e-04
BiCG: residual = 7.177916e-15

BiCG: rel_resid = 1.036043e-15
```

```
./test1 testmat.mtx 0 sol1.txt res1.txt
# 求解1の結果(デフォルト: -i bicg, -p none)
```

```
number of processes = 1
max number of threads = 16
number of threads = 4
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : CGS 3
PRECON    : Jacobi
STORAGE   : CRS
lis_solve : normal end

CGS: iter     = 15 iter_double = 15
iter_quad = 0
CGS: times    = 7.081032e-04
CGS: p_times  = 1.065731e-04 (p_c =
9.536743e-07 p_i = 1.056194e-04 )
CGS: i_times  = 6.015301e-04
CGS: residual = 1.359740e-14

CGS: rel_resid = 1.962616e-15
```

```
./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1
# 求解2の結果
```


8. Lis並列(MPI)版のインストール(1/3)

Lis並列(mpi)版のconfigure:

```
$ pwd
/home/t00003/nfs/lis-1.2.49_mpi
$ ./configure  --enable-mpi  --enable-saamg          # ここでmpiを指定する
```

● Lisを正式にインストールする場合，--prefix オプションでディレクトリも指定.

```
 ･･･ 途中省略 ･･･

 ･･･ 出力の最後部 ･･･

Using
  C   Compiler      = mpicc
  C   Flags         =  -DHAVE_CONFIG_H -O3 -ansi_alias  -DUSE_MPI
  C   Libs          =
      Libs          =    -lm
  F90 Compiler      = mpif90
  F90 Flags         = -O3 -fpp -g -DZERO_ORIGIN=1 -DUSE_MPI
  F90 Libs          =
  enable Fortran Wrapper         = no
  enable SA-AMG Preconditioner   = yes
  enable MPI                     = yes
  enable OpenMP                  = no
  enable Quad Precision          = no
  MPICC                = mpicc
  MPIF77               = mpif90
  MPIFC                = mpif90
  MPILIBS              =
  MPIRUN               = mpirun
  MPFLAG               =
```

赤文字の10項目をチェック

● configure実行後は，作業ディレクトリやパスを変更しないように(生成されるMakefile中では絶対パスが用いられている).

8. Lis並列(MPI)版のインストール(2/3)

Lis並列(mpi)版のmake:

```
$ make

 ･･･ 途中省略 ･･･

 ･･･ 出力の最後部 ･･･

/bin/sh ../libtool --mode=link mpif90  -Vaxlib -nofor_main -o spmvtest5 -lm  spmvtest5.o -
L../src -llis -lm
libtool: link: mpif90 -Vaxlib -nofor_main -o spmvtest5 spmvtest5.o  -
L/home/t00003/nfs/lis-1.2.49_mpi/src /home/t00003/nfs/lis-1.2.49_mpi/src/.libs/liblis.a -
lm
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_mpi/test'
Making all in win32
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_mpi/win32'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_mpi/win32'
make[1]: Entering directory `/nfs/all/t00003/lis-1.2.49_mpi'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_mpi'
```

最後部でErrorが出ていなければOK

8. Lis並列(MPI)版のインストール(3/3)

Lis並列(mpi)版のインストールチェック(make check):

```
$ make check

   ･･･ 出力の最後部分 ･･･
checking solvers...
mpiexec: Error: PBS_JOBID not set in environment.  Code must be run from a
  PBS script, perhaps interactively using "qsub -I".
checking eigensolvers...
mpiexec: Error: PBS_JOBID not set in environment.  Code must be run from a
  PBS script, perhaps interactively using "qsub -I".

checking SAAMG preconditioner...
mpiexec: Error: PBS_JOBID not set in environment.  Code must be run from a
  PBS script, perhaps interactively using "qsub -I".
FAIL: test.sh
================================
1 of 1 test failed
Please report to devel@ssisc.org
================================
make[2]: *** [check-TESTS] Error 1
make[2]: Leaving directory `/nfs/all/t00003/lis-1.2.49_mpi/test'
make[1]: *** [check-am] Error 2
make[1]: Leaving directory `/nfs/all/t00003/lis-1.2.49_mpi/test'
make: *** [check-recursive] Error 1
$
```

MPIプログラムはqsubを用いないとシステム的に実行できないのでErrorになって当然

qsubで実行するスクリプトに「make check」を記述すれば，インストールチェックできる.

8. ジョブ実行スクリプトの用意(MPI版)

HA8000では，PBS(NQSの一種)でジョブ実行するスクリプトを使用する:

```
$ cd test
$ ls  $HOME/sample/qsub_script/
qsub_mpi.nqs  qsub_omp.nqs  qsub_seq.nqs

$ cp  -p  $HOME/sample/qsub_script/qsub_mpi.nqs .
$ cat qsub_mpi.nqs
#@$-q  lecture
#@$-N  4
#@$-lm 2GB
#@$-lT 0:01:00
#@$-o  OUT            # 標準出力のファイル名
#@$-e  ERR            # 標準エラー出力のファイル名
#@$-s  /bin/bash      # スクリプト内のシェルを指定

cd $PBS_O_WORKDIR     # スクリプト内もカレントディレクトリへ移動
pwd                   # pwdは本来は不要だが，確認のため

mpirun ./test1 testmat.mtx 0 sol1.txt res1.txt
mpirun ./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1
#mpirun ./test1 ./mat/add32.mtx 2 sol.txt res.txt -i 5 -p 2
#mpirun ./test1 ./mat/add32.mtx ./mat/add32_rhs1.dat sol.txt res.txt -i 5 -p 2
#mpirun ./test1 ./mat/add32.rua 2 sol.txt res.txt -i 5 -p 2
#mpirun ./test1 ./mat/nos3.mtx 2 sol.txt res.txt -i 5 -p 2

# see manual subsection 3.4
#   20100914 created  by itosho (ITOH, Shoji)
```

- スクリプトの拡張子は任意
- NQSのスクリプトでは左端の“ #@$ ”記号は，これに続くハイフンで始まる文字列がNQSオプションであることを宣言している.
- 赤文字の項目をチェック(NQSオプション“-N”でノード数を指定している)
- MPIジョブは mpirun コマンドの引数に実行プログラムを記述する.

8. HA8000でのジョブ実行(MPI版)

ジョブ実行(qsub)と実行中・実行後の状態確認方法:

```
$ qsub qsub_mpi.nqs
Request 607391.batch1 submitted to queue: lecture.

$ qstat                                          # ジョブ実行中の表示例
2010/09/15 (Wed) 17:22:58:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 231h 32m  2s)
   REQUEST      NAME      OWNER      QUEUE     PRI NODE E-TIME   MEM    STATE
607391.batch1   qsub_mp   t00003     lecture   63    4     60s    28GB  RUNNING

$ qstat                                          # ジョブ終了後の表示例
2010/09/15 (Wed) 17:23:24:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2010/09/25 (Sat)  8:55:00 (Remain: 231h 31m 36s)
   REQUEST      NAME      OWNER      QUEUE     PRI NODE E-TIME   MEM    STATE
No requests.

$ type lrt
lrt is aliased to `ls -lrt'
$ lrt
 ･･･ 省略 ･･･
-rw------- 1 t00003 t00       0 Sep 15 17:22 ERR       # 標準エラー出力のファイル
-rw------- 1 t00003 t00    3242 Sep 15 17:22 sol1.txt  # 解ベクトル1(solution)
-rw------- 1 t00003 t00     208 Sep 15 17:22 res1.txt  # 相対残差1(residual)のノルム収束履歴
-rw------- 1 t00003 t00    3242 Sep 15 17:23 sol2.txt  # 解ベクトル2(solution)
-rw------- 1 t00003 t00     208 Sep 15 17:23 res2.txt  # 相対残差2(residual)のノルム収束履歴
-rw------- 1 t00003 t00     864 Sep 15 17:23 OUT       # 標準出力のファイル
```


8. 実行したジョブの結果確認(MPI版)

※ 編集の都合上，出力の一部を折り返し表示している.

```
$ cat OUT
/home/t00003/nfs/lis-1.2.49_mpi/test
                    # 確認のpwdの出力
number of processes = 4
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : BiCG 2
PRECON    : None
STORAGE   : CRS
lis_solve : normal end

BiCG: iter      = 15 iter_double = 15
iter_quad = 0
BiCG: times     = 6.489754e-04
BiCG: p_times   = 1.597404e-05 (p_c =
9.536743e-07 p_i = 1.502037e-05 )
BiCG: i_times   = 6.330013e-04
BiCG: residual = 7.396952e-15

BiCG: rel_resid = 1.067658e-15
```

```
number of processes = 4
100 x 100 matrix  460 entries
Initial vector x = 0
PRECISION : DOUBLE
SOLVER    : CGS 3
PRECON    : Jacobi
STORAGE   : CRS
lis_solve : normal end

CGS: iter      = 15 iter_double = 15
iter_quad = 0
CGS: times     = 6.649494e-04
CGS: p_times   = 3.385544e-05 (p_c =
9.536743e-07 p_i = 3.290176e-05 )
CGS: i_times   = 6.310940e-04
CGS: residual = 1.141642e-14

CGS: rel_resid = 1.647818e-15
```

`./test1 testmat.mtx 0 sol1.txt res1.txt`
求解1の結果(デフォルト: -i bicg, -p none)

`./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1`
求解2の結果

余談: 当実習に役立つユーティリティ1

cdlis: 他版のtestディレクトリ下へ移動する

```
$ ls  $HOME/sample/sh/
cdlis_kernel.sh*  morelis*

$ type cdlis
cdlis is aliased to `source $HOME/sample/sh/cdlis_kernel.sh'

$ pwd
/home/t00003/sample
$ cdlis seq                          # 引数のseqとは下記の赤文字で記したディレクトリ名の一部
/home/t00003/nfs/lis-1.2.49_seq/test # seqの部分がSEQ だったら, cdlisの引数はSEQ
$ pwd
/home/t00003/nfs/lis-1.2.49_seq/test
$ cdlis omp                          # ompも同様である.
/home/t00003/nfs/lis-1.2.49_omp/test
$ pwd
/home/t00003/nfs/lis-1.2.49_omp/test
```


余談: 当実習に役立つユーティリティ2

morelis: 他版のtestディレクトリ下のファイルの内容を参照するコマンド

```
$ type morelis
morelis is hashed (/home/t00003/sample/sh/morelis)
$ pwd
/home/t00003/nfs/lis-1.2.49_omp/test

$ more OUT
/home/t00003/nfs/lis-1.2.49_omp/test

number of processes = 1
 ...

$ morelis seq OUT
/home/t00003/nfs/lis-1.2.49_seq/test

number of processes = 1
 ...

$ pwd
/home/t00003/nfs/lis-1.2.49_omp/test
```


余談: test1コマンドは他のディレクトリでも利用可能

```
$ cdlis   seq
$ mkdir   $HOME/tmp
$ cp  -p  test1 qsub_seq.nqs testmat.mtx  $HOME/tmp

$ cd $HOME/tmp
$ ls
qsub_seq.nqs  test1*  testmat.mtx

$ $ qsub qsub_seq.nqs
Request 608801.batch1 submitted to queue: lecture.
$ lrt
total 1800
-rw-r--r-- 1 t00003 t00   19290 Oct 31  2007 testmat.mtx
-rwxr-xr-x 1 t00003 t00 1669905 Sep 14 16:54 test1*
-rw-r--r-- 1 t00003 t00     512 Sep 14 19:24 qsub_seq.nqs
-rw------- 1 t00003 t00       0 Sep 18 15:51 ERR
-rw------- 1 t00003 t00     844 Sep 18 15:51 OUT
-rw-r--r-- 1 t00003 t00    3242 Sep 18 15:51 sol1.txt
-rw-r--r-- 1 t00003 t00     208 Sep 18 15:51 res1.txt
-rw-r--r-- 1 t00003 t00    3242 Sep 18 15:51 sol2.txt
-rw-r--r-- 1 t00003 t00     208 Sep 18 15:51 res2.txt
```

目　次

9. Lis (a Library of Iterative Solvers for linear systems) の概観

大規模実疎行列の係数行列 (A):

線形方程式

$$Ax = b$$

標準固有値問題

$$Ax = \lambda x$$

これらを解くためのスケーラブルな反復解法ライブラリ
基本的にC言語向けのライブラリだが，Fortranのインタフェース(API)もあり

開発言語はC言語だが，一部SA-AMG前処理のみFortran90で記述されている．

この講習会では線形方程式の方を取り上げて説明します．

10. 線形方程式用Lisの概要

[http://www.ssisc.org]

Lis (a Library of Iterative Solvers for linear systems)

- 線形方程式求解アルゴリズムの逐次版，並列版（OpenMP, MPI）．
- 11種類のデータ形式をサポート. (CRS, CCS, MSR, DIA, …)
- 倍精度，4倍精度演算サポート
- Lis（小武守，藤井，西田，長谷川）
  22 解法 × 11 前処理 × 3 スケーリング（“前処理無し”，“スケーリング無し”も含める）

Component of Lis: [Lis-1.2.49 ver.]　(† Lis の特徴：国内研究者の成果，または，力を入れている)

反復解法
非定常反復解法
CG, BiCG, CGS, BiCGStab, BiCGStab(ℓ), GPBiCG†, TFQMR, Orthomin(m), GMRES(m) BiCGSafe†, CR, BiCR†, CRS†, BiCRStab†, GPBiCR†, BiCRSafe†, FGMRES(m), IDR(s), MINRES
定常反復解法
Jacobi, Gauss-Seidel, SOR

前処理
None, (Point) Jacobi, ILU(k), SSOR, Hybrid†, I+S†, SAINV, SA-AMG†, Crout ILU, ILUT, additive Schwarz

スケーリング（広義の前処理の一種）
None, $D^{-1}Ax = D^{-1}b.$
$D^{-1/2}AD^{-1/2}D^{1/2}\,x = D^{-1/2}b$

11. 当実習で用いた評価条件

Lisのオプション指定方法
（共にデフォルト値）

- 反復法の初期ベクトル:
  $x_0 = \mathbf{0}$
  `-initx_zeros true`
- 収束判定条件:
  $\|r_k\|_2 / \|b\|_2 \le 1.0 \times 10^{-12}$
  `-tol 1.0e-12`
- ベクトル r：アルゴリズム中の残差ベクトル，
  $r_{k+1} = r_k - \alpha_k A p_k$
  （CG法の例）
- 数値解が有効かどうかの判定：真の残差ベクトル
  $\hat{r} = b - A\hat{x}$,　$\hat{x}$：数値解
  $\|\hat{r}\|_2 / \|b\|_2$

12. 非定常反復法に関する盲点

1. 2つの残差ベクトルについて
   - 求解アルゴリズム中の残差ベクトル
   - 算出された数値解を元の方程式に代入した真の残差ベクトル
2. 残差ベクトルを用いた収束判定について
3. 前処理とスケーリングと収束判定との関係

参考文献：

[1] 伊藤祥司，杉原正顯，姫野龍太郎，クリロフ部分空間法に対する前処理方式と収束判定について，情報処理学会論文誌コンピューティングシステム(ACS)，Vol. 3，No. 2，pp. 9-19 (June 2010)．

[2] 伊藤祥司，杉原正顯，クリロフ部分空間法に対する前処理方向とライブラリ実装における注意点，日本応用数理学会2010年度年会，東京，9月，2010年．

12. 線形方程式求解のための反復解法とその収束性に関する評価の概要

非定常反復法の代表として共役勾配法
(CG: Conjugate Gradient method)

非定常反復法では，残差ベクトルに相当する情報も用いて求解アルゴリズムを構成している(収束判定の計算コスト面で有用である)

適当な初期値 x_0

$$p_0 = r_0 = b - Ax_0$$

収束判定

$$\text{for } k = 0,1,\cdots \text{ until } \|r_k\| \le \varepsilon \|b\|$$

$$\alpha_k = (p_k, r_k)/(p_k, Ap_k)$$

$$x_{k+1} = x_k + \alpha_k p_k$$

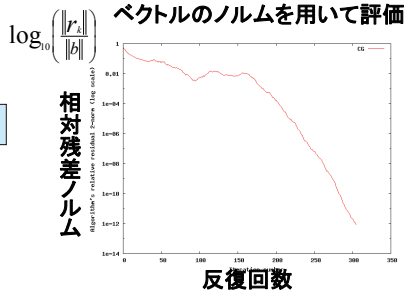
$$r_{k+1} = r_k - \alpha_k Ap_k$$ ← 残差を表すベクトル

$$\beta_k = -(r_{k+1}, Ap_k)/(p_k, Ap_k)$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

$$\text{end}$$

解へ収束する様子を残差ベクトルのノルムを用いて評価



12. 当実習で用いた評価条件(再掲)

Lisのオプション指定方法
(共にデフォルト値)

- 反復法の初期ベクトル:

$$x_0 = \mathbf{0}$$

`-initx_zeros true`

- 収束判定条件:

$$\|r_k\|_2 / \|b\|_2 \le 1.0 \times 10^{-12}$$

`-tol 1.0e-12`

- ベクトル r : アルゴリズム中の残差ベクトル，

$$r_{k+1} = r_k - \alpha_k Ap_k$$

(CG法の例)

- 数値解が有効かどうかの判定：真の残差ベクトル

$$\hat{r} = b - A\hat{x}, \quad \hat{x} : \text{数値解}$$

$$\|\hat{r}\|_2 / \|b\|_2$$

12. main関数での真の残差の算出

test1_org.c test1_mod.c との比較:

```
$ diff test1_org.c test1_mod.c
55a56
>       LIS_REAL                b_nrm2;    // addition
155a157,158
>       lis_vector_nrm2(b, &b_nrm2);    // addition
>
177a181
>               printf("%s: rel_resid = %e¥n¥n",solvername,resid/b_nrm2);  // addition
```

$\|b\|_2$ 用の変数を宣言

$\|b\|_2$ を算出

$\|\hat{r}\|_2 / \|b\|_2$ を算出

$$\hat{r} = b - A\hat{x}, \quad \hat{x} : \text{数値解}$$

```
$ cat OUT
/home/t00003/nfs/lis-1.2.49_seq/test
 ...
BiCG: residual = 6.399933e-15
BiCG: rel_resid = 9.237507e-16
```

$\|\hat{r}\|_2$

$\|\hat{r}\|_2 / \|b\|_2$

Lisのデフォルト (test1_org.c)

当講習会で併用 (test1_mod.c)

12. 前処理付き共役勾配法(PCG法)

前処理行列 K：$K \approx A$ と $K = LL^T$

(Lは下三角行列)

不完全コレスキー分解

$$(L^{-1}AL^{-T})(L^T\boldsymbol{x}) = (L^{-1}\boldsymbol{b}) \quad (1)$$

※ただし，実際には上記の変換は行わず，アルゴリズムで同値な演算を行う．

K が，どれくらい A を近似しているかが収束性向上のポイント

前処理の演算：

$\boldsymbol{q} = K^{-1}\boldsymbol{r}$ ($=L^{-T}L^{-1}\boldsymbol{r}$) に対して，

$\boldsymbol{y}=L^{-1}\boldsymbol{r}$（前進代入），および，

$\boldsymbol{q}=L^{-T}\boldsymbol{y}$（後退代入）を計算する

PCG法のアルゴリズム

・$\boldsymbol{x}_0$:適当な初期値を与える．

・$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 = K^{-1}\boldsymbol{r}_0$

for $k = 0, 1, \ldots,$ until $\|\boldsymbol{r}_k\| \le \varepsilon\|\boldsymbol{b}\|$ do

begin

$$\begin{aligned}
\alpha_k &= \frac{(K^{-1}\boldsymbol{r}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)} \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k \\
\beta_k &= \frac{(K^{-1}\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1})}{(K^{-1}\boldsymbol{r}_k, \boldsymbol{r}_k)} \\
\boldsymbol{p}_{k+1} &= K^{-1}\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k
\end{aligned}$$

end

12. CGS法と前処理付きCGS法の例

$A \approx K = K_L K_R$ 不完全LU分解

$Ax = b$

この変換自体は実施しない

$K_L^{-1}AK_R^{-1}K_R x = K_L^{-1}b$

$x_0, r_0 = b - Ax_0, r_0^{\#}, \beta_{-1} = 0,$

$k = 0,1,2,\cdots;$ Do

$p_k = r_k + \beta_{k-1}z_{k-1},$

$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1}u_{k-1}),$

$\alpha_k = \dfrac{\langle r_0^{\#}, r_k\rangle}{\langle r_0^{\#}, Au_k\rangle},$

$z_k = p_k - \alpha_k Au_k,$

$x_{k+1} = x_k + \alpha_k(p_k + z_k),$

$r_{k+1} = r_k - \alpha_k A(p_k + z_k),$

$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1}\rangle}{\langle r_0^{\#}, r_k\rangle},$

End Do

$x_0, r_0 = b - Ax_0, r_0^{\#}, \beta_{-1} = 0,$

$k = 0,1,2,\cdots;$ Do

$p_k = r_k + \beta_{k-1}z_{k-1},$

$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1}u_{k-1}),$

$\alpha_k = \dfrac{\langle r_0^{\#}, r_k\rangle}{\langle r_0^{\#}, AK^{-1}u_k\rangle},$

$z_k = p_k - \alpha_k AK^{-1}u_k,$

$x_{k+1} = x_k + \alpha_k K^{-1}(p_k + z_k),$

$r_{k+1} = r_k - \alpha_k AK^{-1}(p_k + z_k),$

$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1}\rangle}{\langle r_0^{\#}, r_k\rangle},$

End Do

12. CGS法とスケーリングを施した線形方程式に対するCGS法の例

$Ax = b$

この変換自体を実施する

$D = \mathrm{diag}(A)$ $D^{-1}Ax = D^{-1}b$ $\hat{A}\hat{x} = \hat{b}$

$x_0, r_0 = b - Ax_0, r_0^{\#}, \beta_{-1} = 0,$

$k = 0,1,2,\cdots;$ Do

$p_k = r_k + \beta_{k-1}z_{k-1},$

$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1}u_{k-1}),$

$\alpha_k = \dfrac{\langle r_0^{\#}, r_k\rangle}{\langle r_0^{\#}, Au_k\rangle},$

$z_k = p_k - \alpha_k Au_k,$

$x_{k+1} = x_k + \alpha_k(p_k + z_k),$

$r_{k+1} = r_k - \alpha_k A(p_k + z_k),$

$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1}\rangle}{\langle r_0^{\#}, r_k\rangle},$

End Do

$x_0, r_0 = \hat{b} - \hat{A}x_0, r_0^{\#}, \beta_{-1} = 0,$

$k = 0,1,2,\cdots;$ Do

$p_k = r_k + \beta_{k-1}z_{k-1},$

$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1}u_{k-1}),$

$\alpha_k = \dfrac{\langle r_0^{\#}, r_k\rangle}{\langle r_0^{\#}, \hat{A}u_k\rangle},$

$z_k = p_k - \alpha_k \hat{A}u_k,$

$x_{k+1} = x_k + \alpha_k(p_k + z_k),$

$r_{k+1} = r_k - \alpha_k \hat{A}(p_k + z_k),$

$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1}\rangle}{\langle r_0^{\#}, r_k\rangle},$

End Do

スケーリングの場合は，係数行列，右辺項とも，元とは異なる

12. Lisにおけるスケーリングの注意点

● Jacobiスケーリング：

$$D^{-1}Ax = D^{-1}b$$

● 対角スケーリング：

$$D^{-1/2}AD^{-1/2}D^{1/2}\,x = D^{-1/2}b$$

Lisでは，スケーリングすると，Aとbのデータは上書きされる．

例えば，Jacobiスケーリングのときの真の残差ベクトル：

$$D^{-1}b - D^{-1}A\hat{x}$$

となっており，本来求めるべき情報とは異なるので注意．

※ I+S前処理でもスケーリングを用いているので同様に注意．

`test1.c`では，元のAとbを保持，または，あらためて読込んで正確に判定する必要がある．

→ 後者で修正したものが

`$HOME/sample/test/test1_mod2.c`

「`rel_resid2=`」の出力が追加される．

12. CGS法の2つの異なる左前処理付きアルゴリズムと収束判定

見るからに
左前処理付きアルゴリズム

$x_0, r_0 = K^{-1}(b - Ax_0), r_0^{\#}, \beta_{-1} = 0,$
$k = 0,1,2,\cdots;$ Do
$p_k = r_k + \beta_{k-1} z_{k-1},$
$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1} u_{k-1}),$
$\alpha_k = \dfrac{\langle r_0^{\#}, r_k \rangle}{\langle r_0^{\#}, K^{-1}Au_k \rangle},$
$z_k = p_k - \alpha_k K^{-1}Au_k,$
$x_{k+1} = x_k + \alpha_k (p_k + z_k),$
$r_{k+1} = r_k - \alpha_k K^{-1}A(p_k + z_k),$
$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1} \rangle}{\langle r_0^{\#}, r_k \rangle},$
End Do

$K^{-1}Ax = K^{-1}b$

収束判定

× $\dfrac{\|r_k\|}{\|b\|} = \dfrac{\|K^{-1}(b - Ax_k)\|}{\|b\|} \le \varepsilon$

△ $\dfrac{\|r_k\|}{\|K^{-1}b\|} = \dfrac{\|K^{-1}(b - Ax_k)\|}{\|K^{-1}b\|} \le \varepsilon$

左側は係数行列の部分のみを変換したアルゴリズム.

右側は3種類の前処理方向の変換で，全て同じアルゴリズムとなる.しかも，左前処理にも関わらず，収束判定は下式の通り

収束判定

○ $\dfrac{\|r_k\|}{\|b\|} = \dfrac{\|b - Ax_k\|}{\|b\|} \le \varepsilon$

※ 両側・左・右前処理変換後の前処理付きアルゴリズムは一致する（一致性がある）

$x_0, r_0 = b - Ax_0, r_0^{\#}, \beta_{-1} = 0,$
$k = 0,1,2,\cdots;$ Do
$p_k = r_k + \beta_{k-1} z_{k-1},$
$u_k = p_k + \beta_{k-1}(z_{k-1} + \beta_{k-1} u_{k-1}),$
$\alpha_k = \dfrac{\langle r_0^{\#}, r_k \rangle}{\langle r_0^{\#}, AK^{-1}u_k \rangle},$
$z_k = p_k - \alpha_k AK^{-1}u_k,$
$x_{k+1} = x_k + \alpha_k K^{-1}(p_k + z_k),$
$r_{k+1} = r_k - \alpha_k AK^{-1}(p_k + z_k),$
$\beta_k = \dfrac{\langle r_0^{\#}, r_{k+1} \rangle}{\langle r_0^{\#}, r_k \rangle},$
End Do

12. Lisにおける前処理付きアルゴリズムの状況

Lisの非定常反復解法 cg(1)～gmres(9)は，右前処理系であり，アルゴリズムの収束判定も正常.

ただし，bicgsafe(13)～minres(22)については，伊藤の方では未確認なので，左前処理系にも関わらず収束判定が

$$\frac{\|r_k\|}{\|b\|} \le \varepsilon$$

と甘い場合があるかも知れないので注意.

※ 定常反復法（jacobi(10)～sor(12)）と非定常反復法とでは論点が異なるので注意.

13. test1 処理フローの概要(1)

```
int main(int argc, char* argv[] ) {
    lis_initialize

    lis_matrix_create                    係数行列 A をmalloc
    lis_vector_create                    ベクトル b, x をmalloc
    lis_input(A,b,x,argv[1]);            A(MMとHB形式), b(ファイル入力の場合),
                                         x(初期値)を読み込み

    lis_matrix_set_type                  行列データをCRSへ変換
    lis_matrix_convert                   〃
    右辺項 b の作成処理

    lis_solve(A,b,x,solver);             求解ルーチン

    lis_solver_get_timeex                タイマーの値をget
    lis_solver_get_residualnorm          真の残差ノルム算出

    lis_output_vector(x,LIS_FMT_MM,argv[3]);   解を出力
    lis_solver_output_rhistory(solver, argv[4]);   残差履歴を出力

    lis_finalize();
    return 0;
}
```

MMとHB以外の形式は，マニュアル5節「行列格納形式」のサンプルを参照

13. test1 処理フローの概要(2)

```
int lis_solve( ) {
    lis_precon_create                    前処理演算の設定
    lis_solve_kernel(A, b, x, solver, precon);   求解のための事前処理
}
```

```
int lis_solve_kernel( ) {
    test1の引数をチェック

    x0を設定

    if(I+S前処理選択) スケーリング処理        I+S前処理は要スケーリング
    else (スケーリング選択) スケーリング処理   CG法ではSYMM_DIAG
                                         ※1 scalingでは A, b のデータを上書き

    lis_solver_execute                   求解（例えば，lis_gmresを呼び出す）

    y=D^{-1/2}x          「SYMM_DIAG(対角scaling)」&&「I+S前処理以外」のとき実行

    lis_vector_nrm2(t,&nrm2);            真の残差ベクトルの2ノルム（絶対残差）
                                         ※2 scaling後の系に対して算出(cf. ※1)
}
```

14. PETScインストールの準備作業

PETSc利用環境の準備:

```
$ cd $HOME/nfs

$ tar  zxf  $HOME/sample/library/petsc-lite-3.1-p3.tar.gz
$ ls
lis-1.2.49/  lis-1.2.49_mpi/  lis-1.2.49_omp/  lis-1.2.49_seq/  petsc-3.1-p3/

$ mv  petsc-3.1-p3  petsc-lite-3.1-p3      # lite版と明示しておく方が良いと思う
```

PETScでは, Intelコンパイラの環境設定してもconfigureでは, gcc, gfortranが設定されるため, ここではGNUコンパイラを用いることにする.

```
$ cd petsc-lite-3.1-p3
bin/   config/     include/  projects/  TAGS        zope/
conf/  configure@  makefile  src/       tutorials/
$ ls -l configure
lrwxrwxrwx 1 t00003 t00 19 Sep 19 13:35 configure -> config/configure.py*
```

●PETScのホームページ
http://www.mcs.anl.gov/petsc/
●PETScのソース:
http://www.mcs.anl.gov/petsc/download/index.html
●PETScのインストール手順:
http://www.mcs.anl.gov/petsc/documentation/installation.html

14. PETScインストール

PETScのインストール:

```
$ ./configure --with-cc=gcc --with-fc=gfortran --download-f-blas-lapack=1
--download-mpich=1
================================================================================
             Configuring PETSc to compile on your system
================================================================================
      ･･･ 途中省略 ･･･（10分位かかる）
xxx=============================================================================xxx
   Configure stage complete. Now build PETSc libraries with:
   make PETSC_DIR=/nfs/all/t00003/petsc-lite-3.1-p3 PETSC_ARCH=linux-gnu-c-debug all
xxx=============================================================================xxx

$ make all test
    ･･･ 途中省略 ･･･  （10分位かかる）
Completed building libraries
=========================================
Now to check if the libraries are working do:
make PETSC_DIR=/nfs/all/t00003/petsc-lite-3.1-p3 PETSC_ARCH=linux-gnu-c-debug test
=========================================
Running test examples to verify correct installation
C/C++ example src/snes/examples/tutorials/ex19 run successfully with 1 MPI process
C/C++ example src/snes/examples/tutorials/ex19 run successfully with 2 MPI processes
Fortran example src/snes/examples/tutorials/ex5f run successfully with 1 MPI process
Completed test examples
```

● これら2種類の環境変数が重要

14. PETScの使用要領

サンプルプログラムからLisのtest1のような機能を自作する:

```
$ pwd
/home/t00003/nfs/petsc-lite-3.1-p3
$ ls
bin/     configure@          include/            make.log@  src/        zope/
conf/    configure.log@      linux-gnu-c-debug/  projects/  TAGS
config/  externalpackages/   makefile            RDict.log  tutorials/

$ ls src/mat/examples/tests
ex100.c   ex113.c  ex126f.F   ex21.c   ex37.c  ex51.c   ex63f.F  ex77.c   ex95.c
   ･･･ 途中省略 ･･･
ex104.c   ex117.c  ex12.c     ex27.c   ex41.c  ex55.c   ex67f.F  ex80.c   ex9.c
ex105f.F  ex118.c  ex13.c     ex28.c   ex42.c  ex56.c   ex68.c   ex81.c   makefile
ex106.c   ex119.c  ex14.c     ex2.c    ex43.c  ex57.c   ex6.c    ex85f.F  output/
ex107.c   ex11.c   ex15.c     ex30.c   ex44.c  ex58.c   ex70.c   ex86.c
   ･･･ 以下省略 ･･･
```

例えば, Matrix Market形式の行列を入力するプログラム例は ex72.c である.
この「tests」下の ex72.c, makefile をPETSC_DIR下にコピーして make する.
ただし, 元々の「makefile」を上書きしないようmakefile2 とコピーした例で以下説明

● その際のスクリプト例:

```
#!/bin/bash

export PETSC_DIR="/home/t00003/nfs/petsc-lite-3.1-p3"
export PETSC_ARCH="linux-gnu-c-debug"

make -f makefile2 ex72
```

● PETScの詳細についてはマニュアル参照