

目次

1. 概要	2
1. 1. はじめに	2
1. 2. 制限事項	2
2. コンパイルと実行方法	3
2. 1. コンパイル	3
2. 2. 実行方法	4
3. 係数行列と固有ベクトルの分割方法	5
3. 1. プロセスマップ	5
3. 3. 固有ベクトルデータの分割と分配規則.....	7
4. 複素対称行列用ハウスホルダ 3 重対角化変換プログラム PZSYTRD	9
4. 1. 引数仕様	9
4. 2. リターンコード	10
5. 固有ベクトルのハウスホルダ逆変換プログラム PZSYGHR	11
5. 1. 引数仕様	11
5. 2. リターンコード	12
付録 1. PZSYTRD, PZSYGHR 接続サンプルプログラム TEST	13
付録 2. 分散配置固有ベクトルマージプログラム MARGEV	16
付録 3. 残差判定プログラム RESID	18
付録 4. 係数行列作成プログラム MATGEN	20
付録 5. 係数行列分配プログラム MATDIV	21
付録 6. QR 法による固有値固有ベクトル求解プログラム SUB	22

1. 概要

本稿は分散メモリ並列計算機対応の**複素対称行列用ハウスホルダ 3 重対角化変換プログラム (PZSYTRD)**と固有ベクトルの**ハウスホルダ逆変換プログラム(PZSYGHR)**の利用法について説明します。

1. 1. はじめに

PZSYTRDは複素対称行列Cをハウスホルダ変換 $T = HCH^T$ によって3重対角行列Tへ変換するプログラムです。

PZSYGHRはQR法による固有値・固有ベクトル計算において必要となる初期固有ベクトルを作成するために単位行列にハウスホルダ逆変換を施すプログラムです。

2つのプログラムはScalapackの入出力インターフェースを踏襲しております。

また、ハウスホルダ変換の係数行列作成 $H = (\prod_{k=1}^n H_k), H_k = (I - \alpha u_k u_k^T)$ にはScalapack同様、多段変換型のブロック化アルゴリズムを採用しています。

1. 2. 制限事項

2つのプログラムには使用上、次の制限があります。

- (1) PZSYTRD, PZSYGHR は Fortran77 または Fortran90 で記述されたプログラムコードから呼び出されることを前提としております。
- (2) 指定できる MPI 数(NP)は 2 の偶数ベキ, すなわち, NP=1, 4, 16, 64, 256, 1024, 4096, ...となります。
- (3) 全体行列の分散配置する際に使用する小行列の**ブロックサイズ(NB)**は **8 以上の偶数**とします。詳細は 3 章を確認ください。行列データの**分配・分散**は利用者が行うものとしております。
- (4) PZSYTRD, PZSYGHR とともに Message Passing Interface (MPI) を利用しているため, MPI に対応した Fortran コンパイラを前提としております。

2. コンパイルと実行方法

PZSYTRD, PZSYGHR のコンパイルと実行方法について説明します。

2. 1. コンパイル

PZSYTRD, PZSYGHR を使用するためのライブラリ(libPZSYTRD.a)の作成する makefile の例を図 2-1 に示します。図 2-2 にはユーザプログラムとライブラリを接続する makefile の例を示します。図 2-1, 2-2 の例では, Fortran コンパイラに日立最適化 F90(SR16000/M1 向け)を利用する mpif90 を指定しています。

```
F90      = mpif90 -Os -64 -model=M1 -parallel -i,L
SRCDIR   = ./SRC
LIBNAME  = libPZSYTRD.a
IFILE    = pzsyttrd.o j5initzc.o j5pahous.o j5mvlwp.o ¥
          j5ranknb.o j5rngcom.o j5serial.o j5parank.o ¥
          pzsygthr.o j5inivec.o j5makvec.o

$(LIBNAME) : $(IFILE)
    -rm -f $(LIBNAME)
    ar -qv -X64 $(LIBNAME) $(IFILE)
    @echo "$(LIBNAME) is now up-to-date"

pzsyttrd.o : $(SRCDIR)/pzsyttrd.f
    $(F90) -c $(SRCDIR)/pzsyttrd.f
j5initzc.o : $(SRCDIR)/j5initzc.f
    $(F90) -c $(SRCDIR)/j5initzc.f
j5pahous.o : $(SRCDIR)/j5pahous.f
    $(F90) -c $(SRCDIR)/j5pahous.f
j5mvlwp.o : $(SRCDIR)/j5mvlwp.f
    $(F90) -c $(SRCDIR)/j5mvlwp.f
j5ranknb.o : $(SRCDIR)/j5ranknb.f
    $(F90) -c $(SRCDIR)/j5ranknb.f
j5rngcom.o : $(SRCDIR)/j5rngcom.f
    $(F90) -c $(SRCDIR)/j5rngcom.f
j5serial.o : $(SRCDIR)/j5serial.f
    $(F90) -c $(SRCDIR)/j5serial.f
j5parank.o : $(SRCDIR)/j5parank.f
    $(F90) -c $(SRCDIR)/j5parank.f

pzsygthr.o : $(SRCDIR)/pzsygthr.f
    $(F90) -c $(SRCDIR)/pzsygthr.f
j5inivec.o : $(SRCDIR)/j5inivec.f
    $(F90) -c $(SRCDIR)/j5inivec.f
j5makvec.o : $(SRCDIR)/j5makvec.f
    $(F90) -c $(SRCDIR)/j5makvec.f

.PRECIOUS : $(LIBNAME) ;

clean :
    rm -f *.o
```

図 2-1 ライブラリ libPZSYTRD.a 作成用 makefile 例

```

F90      = mpi f90 -O3 -x64 -model=M1 -parallel -i,L
LOADNAME = test
LD       = $(F90) -Wl,'-blpdata' -Wl,'-binitfini:poe_remote_main' -Wl,'-bmaxdata:22147483647'
IFILE    = test.o sub.o

LIB = -L./ -IPZSYTRD

$(LOADNAME) : $(IFILE)
    -rm -f $(LOADNAME)
    $(LD) -o $(LOADNAME) $(IFILE) $(LIB)
    @echo "$(LOADNAME) is now up-to-date"

test.o : ./test.f
    $(F90) -c ./test.f

sub.o : ./sub.f
    $(F90) -c ./sub.f

.PRECIOUS : $(LOADNAME) ;

clean :
    rm -f *.o

```

図 2-2 ユーザプログラムとライブラリを接続する makefile 例

2. 2. 実行方法

PZSYTRD, PZSYGHR の実行には MPI が必須となりますので, poe(AIX 等)や mpirun(mpich 等)などの利用マシンにインストールされている実行バイナリをご使用ください。図 2-2 の例で作成したロードモジュール(test)を MPI プロセス 16 で実行する場合のコマンド例を以下に示します。poe では, MPI 実行プロセス数などを環境変数から取得しますので, ロードモジュール実行前に下記実行コマンド例に示す環境変数の設定を行ってください。mpirun では, MPI 実行プロセス数はコマンドオプション'-np'で指定します。

【poe の実行コマンド例】

```

> setenv MP_PROCS 16
> setenv MP_NODES 1
> setenv MP_TASKS_PER_NODE 1
> setenv MP_SHARED_MEMORY yes
> poe ./test

```

【mpirun のコマンド実行例】

```

> mpirun -np 16 ./test

```

3. 係数行列と固有ベクトルの分割方法

本章では、PZSYTRD、PZSYGHR で使用する係数行列と、PZSYGHR で使用する固有ベクトルの分割方法について説明します。

3. 1. プロセスマップ

係数行列を分割し部分行列に変換するためのプロセスマップについて説明します。プロセスマップは、行方向に連続した MPI プロセス番号を割り当てる Row Major 形式と、列方向に連続した MPI プロセス番号を割り当てる Column Major 形式があります。図 3-1 に、MPI プロセス数 16 の場合の Row Major 形式と Column Major 形式のプロセスマップを示します。

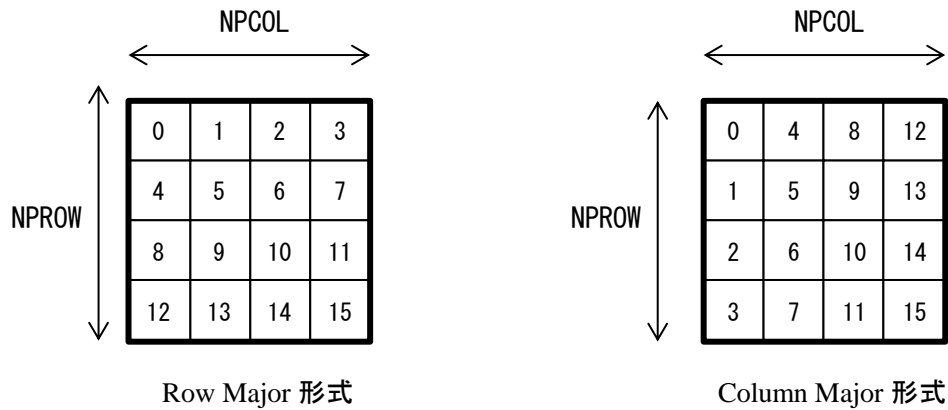


図 3-1 MPI プロセス数 16 の場合のプロセスマップ

プロセスマップの行方向のプロセス数を NPROW，列方向のプロセス数を NPCOL とし、 $NPROW=NPCOL=2$ べき数となるよう MPI プロセス数を選択してください。

各プロセスのプロセスマップ中の位置を行方向，列方向をそれぞれ IPROW，IPCOL とすると，以下の式で設定できます。式中にある IP は，MPI プロセス番号となります。

【Row Major 形式】

- ・ $IPROW = IP / NPCOL + 1$
- ・ $IPCOL = MOD(IP, NPCOL) + 1$

【Column Major 形式】

- ・ $IPROW = MOD(IP, NPROW) + 1$
- ・ $IPCOL = IP / NPROW + 1$

図 3-1 中のプロセス 6 番に着目すると，Row Major 形式では IPROW=2，IPCOL=3 に，Column Major 形式では IPROW=3，IPCOL=2 となります。

3. 2. 係数行列データの分割と分配規則

係数行列データの分割と分配規則について説明します。

はじめに、次元数 N の係数行列を、 $NB \times NB$ の小行列単位に分割します。 NB は 8 以上の偶数となるよう設定してください。この小行列単位に 3. 1. 節で定義したプロセスマップを 2D cyclic に適用します。図 3-2 に行列データの分割と Row Major 形式のプロセスマップ適用例を、図 3-3 にプロセスマップを適用した場合のプロセス 0 番と 6 番の部分行列を示します。図 3-3 中の NLB は以下の式で定義します。

$$\bullet NLB = (N - 1) / (NB * NPROW) + 1$$

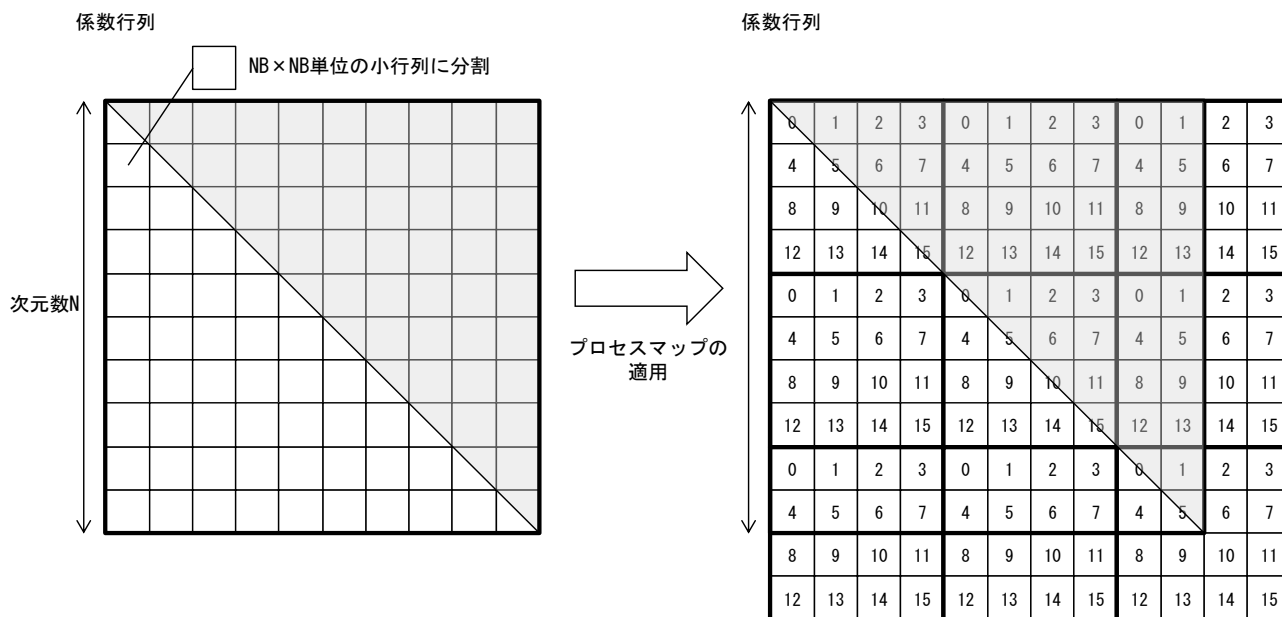


図 3-2 行列データの分割と Row Major 形式プロセスマップの適用例

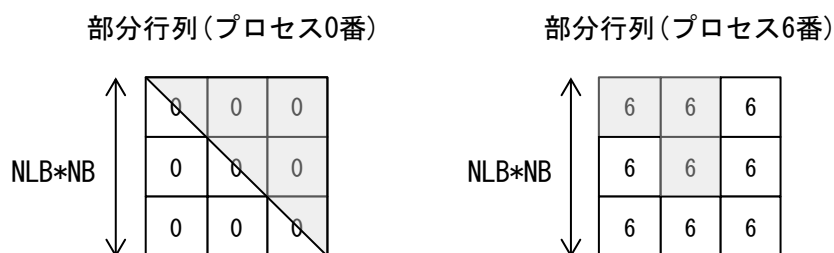


図 3-3 Row Major 形式のプロセスマップを適用した場合のプロセス 0 番と 6 番の部分行列

各 MPI プロセスが、係数行列のどの小行列を担当するかを算出する式を以下で定義します。下記式で算出する $IOWN$ と MPI プロセス番号が一致する場合、以下で定義する式を使用して係数行列から部分行列へのアドレス変換を行います。式中の I_A , J_A はそれぞれ、係数行列の行番号と列番号を示します。

【Row Major 形式】

$$\bullet IOWN = MOD((I_A - 1) / NB, NPROW) * NPROW + MOD((J_A - 1) / NB, NPCOL)$$

【Column Major 形式】

$$\bullet IOWN = MOD((J_A - 1) / NB, NPCOL) * NPCOL + MOD((I_A - 1) / NB, NPROW)$$

さらに、以下で算出される I_L , J_L に従い係数行列の要素を部分行列に格納すれば、各 MPI プロセスの部分行列が完成します。

$$\begin{aligned} \cdot I_L &= ((I_A - 1) / (NB * NPROW)) * NB + MOD(I_A - 1, NB) + 1 \\ \cdot J_L &= ((J_A - 1) / (NB * NPCOL)) * NB + MOD(J_A - 1, NB) + 1 \end{aligned}$$

3. 3. 固有ベクトルデータの分割と分配規則

固有ベクトルデータの分割と分配規則について説明します。

次元数 N の固有ベクトルを MPI プロセス数で分割し、先頭から連続した MPI プロセス番号に割り当てて分散して保持します。分散配置するために各 MPI プロセスでは、 N と以下の式で定義する $JVNUM$ で `Complex*16` 型の配列 $V(N, JVNUM)$ を確保してください。式中の NP は MPI プロセス数です。

$$\cdot JVNUM = (N - 1) / NP + 1$$

固有ベクトル

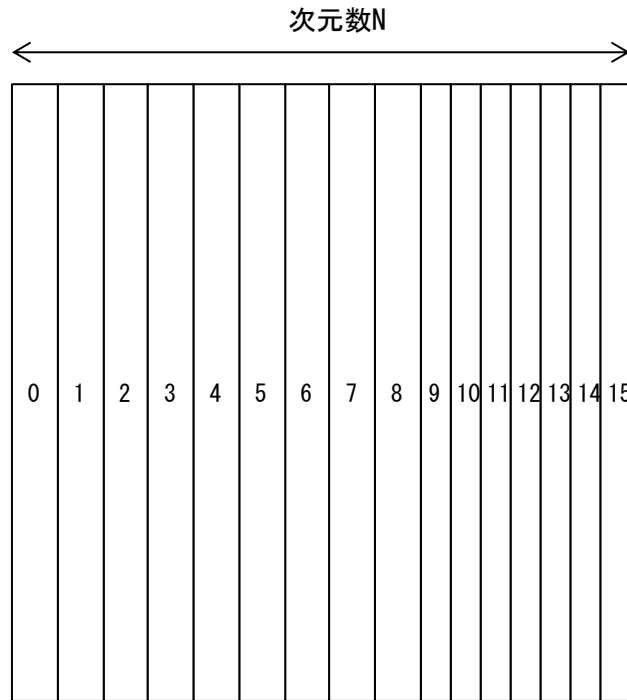


図 3-4 MPI プロセス数 16 の場合の固有ベクトル分割例

図 3-4 に MPI プロセス数 16 の場合の固有ベクトル分割例を示します。固有ベクトルのブロック分割式を以下で定義します。式中の IP は MPI プロセス番号です。各 MPI プロセスが保持する固有ベクトルの始点は 1, 終端は NV となります。

【 $IP < MOD(N, NP)$ の MPI プロセス】

$$\cdot NV = N / NP + 1$$

【 $IP \geq MOD(N, NP)$ の MPI プロセス】

$$\cdot NV = N / NP$$

各 MPI プロセスが担当する部分ベクトルの列番号 J_L は、以下で定義する式によって固有ベクトルの列番

号 J_G に変換できます。

【 $IP < \text{MOD}(N, NP)$ の MPI プロセス】

$$\cdot J_G = (N / NP + 1) * IP + J_L$$

【 $IP \geq \text{MOD}(N, NP)$ の MPI プロセス】

$$JPTR = (N / NP + 1) * \text{MOD}(N, NP) \text{ とし,}$$

$$\cdot J_G = JPTR + (N / NP) * (IP - \text{MOD}(N, NP)) + J_L$$

また、各 MPI プロセスが固有ベクトルを保持する範囲は始点 $JSTART$ と終点 $JEND$ で求められ、これを算出する式を以下で定義します。ただし、 $JSTART > JEND$ となる MPI プロセスは固有ベクトルを保持しません。

【 $IP < \text{MOD}(N, NP)$ の MPI プロセス】

$$\cdot JSTART = (N / NP + 1) * IP + 1$$

$$\cdot JEND = (N / NP + 1) * (IP + 1)$$

【 $IP \geq \text{MOD}(N, NP)$ の MPI プロセス】

$$JPTR = (N / NP + 1) * \text{MOD}(N, NP) \text{ とし,}$$

$$\cdot JSTART = JPTR + (N / NP) * (IP - \text{MOD}(N, NP)) + 1$$

$$\cdot JEND = JPTR + (N / NP) * (IP - \text{MOD}(N, NP) + 1)$$

4. 複素対称行列用ハウスホルダ 3 重対角化変換プログラム PZSYTRD

本章では、複素対称行列用ハウスホルダ 3 重対角化変換プログラム PZSYTRD の引数仕様とリターンコードについて説明します。

4. 1. 引数仕様

PZSYTRD の引数仕様を表 4-1 に示します。表 4-1 中の NB, NLB は以下の式で定義します。

- $NB = DESCA(6)$
- $NLB = (N - 1) / NB + 1$

表 4-1 PZSYTRD 引数仕様

引数	型	種別	引数の説明
UPLO	Character*1	INPUT	入力行列の格納形式。UPLO='L'のみ指定可能
N	Integer	INPUT	全体行列の次元数。N \geq 2
A(*)	Complex*16	INPUT / OUTPUT	(INPUT) : プロセスマップに従って分割した部分行列 (OUTPUT) : 3 重対角化で使用したリフレクタ情報
IA	Integer	INPUT	部分行列 A の行方向先頭アドレス。IA=1 のみ指定可能
JA	Integer	INPUT	部分行列 A の列方向先頭アドレス。JA=1 のみ指定可能
DESCA(9)	Integer	INPUT	ディスクリプタ <ul style="list-style-type: none"> • DESCA(6) 部分行列のブロックサイズ NB NB\geq8 かつ MOD(NB,2)=0 • DESCA(9) 部分行列 A の整合寸法 DESCA(9)\geqNLB*NB • DESCA(1)~DESCA(5), DESCA(7), DESCA(8) 値入力の必要なし
D(N)	Complex*16	OUTPUT	3 重対角行列の対角成分。全プロセスで同じ値が入る
E(N)	Complex*16	OUTPUT	3 重対角行列の副対角成分。全プロセスで同じ値が入る
TAU(N)	Complex*16	OUTPUT	ハウスホルダ変換の鏡像変換係数。 全プロセスで同じ値が入る。
WORK	Complex*16	WORK	値入力の必要なし (作業領域)
LWORK	Integer	INPUT	複素数型作業領域配列 WORK のサイズ。 LWORK \geq 12*NLB*NB+4*NB+4*NB*NLB*NB
INFO	Integer	OUTPUT	リターンコード
MAJOR	Character*1	INPUT	プロセスマップ方向。MAJOR='C'または MAJOR='R'のみ指定可能

4. 2. リターンコード

PZSYTRD のリターンコードを表 4-2 に示します。

表 4-2 PZSYTRD リターンコード

値	内容
0	正常終了したことを表す
-1	UPLO='L'が設定されていないことを表す
-2	$N \geq 2$ が設定されていないことを表す
-4	IA=1 が設定されていないことを表す
-5	JA=1 が設定されていないことを表す
-66	$\text{DESCA}(6) \geq 8$ かつ $\text{MOD}(\text{DESCA}(6), 2) = 0$ が設定されていないことを表す
-69	$\text{DESCA}(9) \geq \text{NLB} * \text{NB}$ が設定されていないことを表す
-11	$\text{LWORK} \geq 12 * \text{NLB} * \text{NB} + 4 * \text{NB} + 4 * \text{NB} * \text{NLB} * \text{NB}$ が設定されていないことを表す
-13	MAJOR='C'または MAJOR='R'が設定されていないことを表す
100	プロセスマップが正方かつ 1 辺が 2 べき数でないことを表す

5. 固有ベクトルのハウスホルダ逆変換プログラム PZSYGHR

本章では、固有ベクトルのハウスホルダ逆変換プログラム PZSYGHR の引数仕様とリターンコードについて説明します。

5. 1. 引数仕様

PZSYGHR の引数仕様を表 5-1 に示します。表 5-1 中の NB, NLB, NP, NPROW, JVNUM は以下の式で定義します。

- $NB = DESCA(6)$
- $NLB = (N - 1) / NB + 1$
- $NP = MPI$ プロセス数
- $NPROW =$ プロセスマップの 1 辺の長さ
- $JVNUM = (N - 1) / NP + 1$

表 5-1 PZSYGHR 引数仕様

引数	型	種別	引数の説明
UPLO	Character*1	INPUT	入力行列の格納形式。UPLO='L'のみ指定可能
N	Integer	INPUT	全体行列の次元数。N \geq 2
A(*)	Complex*16	INPUT	3 重対角化で使用したリフレクタ情報。 PZSYTRD の結果を入れればよい
IA	Integer	INPUT	部分行列 A の行方向先頭アドレス。IA=1 のみ指定可能
JA	Integer	INPUT	部分行列 A の列方向先頭アドレス。JA=1 のみ指定可能
DESCA(9)	Integer	INPUT	ディスクリプタ <ul style="list-style-type: none"> • DESCA(6) 部分行列 A のブロックサイズ NB NB\geq8 かつ MOD(NB,2)=0 • DESCA(9) 部分行列 A の整合寸法 DESCA(9)\geqNLB*NB • DESCA(1)~DESCA(5), DESCA(7), DESCA(8) 値入力の必要なし
TAU(N)	Complex*16	INPUT	ハウスホルダ変換の鏡像変換係数。PZSYTRD の結果を入れればよい。
V(N,*)	Complex*16	OUTPUT	固有ベクトル逆変換結果
WORK	Complex*16	WORK	値入力の必要なし（作業領域）
LWORK	Integer	INPUT	複素数型作業領域配列 WORK のサイズ LWORK \geq NB*NPROW*NLB*NB+2*NLB*NB*NB+4*JVNUM
INFO	Integer	OUTPUT	リターンコード
MAJOR	Character*1	INPUT	プロセスマップ方向。MAJOR='C'または MAJOR='R'のみ指定可能

5. 2. リターンコード

PZSYGHR のリターンコードを表 5-2 に示します。

表 5-2 PZSYGHR リターンコード

値	内容
0	正常終了したことを表す。
-1	UPLO='L'が設定されていないことを表す。
-2	$N \geq 2$ が設定されていないことを表す。
-4	IA=1 が設定されていないことを表す。
-5	JA=1 が設定されていないことを表す。
-66	$\text{DESCA}(6) \geq 8$ かつ $\text{MOD}(\text{DESCA}(6), 2) = 0$ が設定されていないことを表す。
-69	$\text{DESCA}(9) \geq \text{NLB} * \text{NB}$ が設定されていないことを表す。
-10	$\text{LWORK} \geq \text{NB} * \text{NPROW} * \text{NLB} * \text{NB} + 2 * \text{NLB} * \text{NB} * \text{NB} + 4 * \text{JVNUM}$ が設定されていないことを表す。
-12	MAJOR='C'または MAJOR='R'が設定されていないことを表す。
100	プロセスマップが正方かつ 1 辺が 2 べき数でないことを表す。

付録 1. PZSYTRD, PZSYGHR 接続サンプルプログラム TEST

```

PROGRAM TEST
C
  IMPLICIT REAL*8 (A-H, O-Z)
C
  INCLUDE 'mpif.h'
C
  COMPLEX*16, ALLOCATABLE :: A(:), V(:), D(:), E(:), TAU(:)
  COMPLEX*16, ALLOCATABLE :: WORK(:)
  INTEGER N, IA, JA, DESCA(9), LWORK, INFO
  CHARACTER*1 UPLO, MAJOR
C
  COMPLEX*16, ALLOCATABLE :: AORG(:), A2(:), V2(:), ZWK(:)
  INTEGER NP, IP, NPROW, NPCOL, NB, NLB
  INTEGER LDA, LDA2, NGB, JVNUM
  INTEGER I, JV, IERR, IPRINT
C
  DOUBLE PRECISION GF1, GF2
  DOUBLE PRECISION TS, TE
C
  CALL MPI_INIT(IERR)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NP, IERR)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, IP, IERR)
C
  DO I=0, 16
    NPROW=2**I
    IF (NPROW*NPROW .EQ. NP) THEN
      NPCOL=NPROW
      GO TO 10
    END IF
  ENDDO
10 CONTINUE
C
  READ (31, *)
  READ (31, *) UPLO, MAJOR
  READ (31, *)
  READ (31, *) N, DESCA(6)
  READ (31, *)
  READ (31, *) IA, JA
C
  NB=DESCA(6)
  NGB=(N-1)/NB+1
  NLB=(NGB-1)/NPROW+1
  DESCA(9)=NLB*NB
  LDA=DESCA(9)
  LDA2=N+1
C
  GF1=1.0D0*16/3*N*N*N*1.0E-9
  GF2=1.0D0*4*N*N*N*1.0E-9

```

PZSYTRD, PZSYGHR の呼び出し前に,
MPI_INIT を CALL

DESCA(9) の設定と行列分配を行う場合に必要な,
プロセスマップの行数 NPROW と列数 NPCOL を算出

係数行列の分配, 固有ベクトルのマージで
必要となるパラメータを設定

```

      IF (IP .EQ. 0) THEN
        WRITE (6, *) '*****'
        WRITE (6, *) '**** PZSYTRD, PZSYGHR TEST ****'
        WRITE (6, *) '*****'
        WRITE (6, *) '*** INPUT PARAMETER'
        WRITE (6, *) '*** NUM OF MPI = ', NP
        WRITE (6, *) '*** UPLO      = ', UPLO
        WRITE (6, *) '*** MAJOR    = ', MAJOR
        WRITE (6, *) '*** N        = ', N
        WRITE (6, *) '*** NB       = ', NB
        WRITE (6, *) '*** IA       = ', IA
        WRITE (6, *) '*** JA       = ', JA
        WRITE (6, *)
      END IF

C
      JVNUM=(N-1)/NP+1
      LWORK=MAX(12*NLB*NB+4*NB+4*NB*NLB*NB,
      *      NB*NPROW*NLB*NB+2*NLB*NB*NB+4*JVNUM)
C
      ALLOCATE (A (LDA*NLB*NB), V (N*JVNUM), D (N), E (N), TAU (N))
      ALLOCATE (WORK (LWORK))
C
      ALLOCATE (AORG (N*N), A2 (LDA2*N), V2 (N*N))
      ALLOCATE (ZWK ((N+1)*(N/10+5)))
C
      CALL MATGEN (AORG, N)
      CALL MATDIV (MAJOR, AORG, A, LDA, NB, N, IP, NPROW, NPCOL)
C
      CALL MPI_BARRIER (MPI_COMM_WORLD, IERR)
      TS=MPI_WTIME ()
      CALL PZSYTRD (UPLO, N, A, IA, JA, DESCA, D, E, TAU, WORK, LWORK, INFO, MAJOR)
      CALL MPI_BARRIER (MPI_COMM_WORLD, IERR)
      TE=MPI_WTIME ()
      IF (INFO .NE. 0) THEN
        WRITE (6, *) '!!!!!!! PZSYTRD ERROR, INFO =', INFO
        GO TO 9999
      END IF
      IF (IP .EQ. 0) THEN
        WRITE (6, 2000) TE-TS, GF1/(TE-TS)
      END IF
C
      CALL MPI_BARRIER (MPI_COMM_WORLD, IERR)
      TS=MPI_WTIME ()
      CALL PZSYGHR (UPLO, N, A, IA, JA, DESCA, TAU, V, WORK, LWORK, INFO, MAJOR)
      CALL MPI_BARRIER (MPI_COMM_WORLD, IERR)
      IF (INFO .NE. 0) THEN
        WRITE (6, *) '!!!!!!! PZSYGHR ERROR, INFO =', INFO
        GO TO 9999
      END IF
      TE=MPI_WTIME ()
      IF (IP .EQ. 0) THEN
        WRITE (6, 2100) TE-TS, GF2/(TE-TS)
        WRITE (6, *)
      END IF

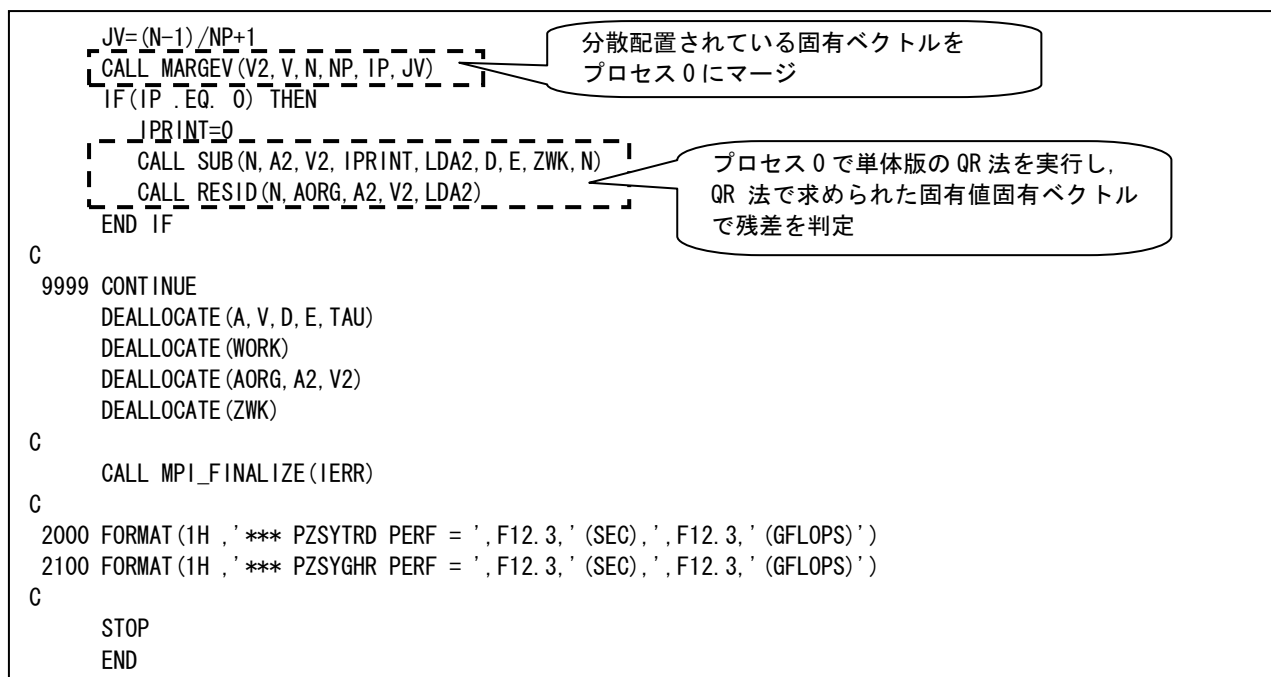
```

PZSYTRD, PZSYGHR の LWORK サイズの大きいほうを LWORK とし, WORK を共有

MATGEN で係数行列を作成し, MATDIV で係数行列の分配を行う

PZSYTRD を CALL

PZSYGHR を CALL



付録 2. 分散配置固有ベクトルマージプログラム MARGEV

```

SUBROUTINE MARGEV(V2, V, N, NP, IP, JV)
C
  IMPLICIT NONE
C
  INCLUDE 'mpif.h'
C
  INTEGER N
  COMPLEX*16 V2(N, *), V(N, *)
  INTEGER NP, IP, JV
C
  COMPLEX*16 VREC(N, JV)
  INTEGER JVBLK(NP+1)
  INTEGER JMOD, JGB
C
  INTEGER I, J, JS, JE, J1, ISIZE, ISEN, JT
  INTEGER ITAG, REQ_S, REQ_R, IERR
  INTEGER SWK(MPI_STATUS_SIZE), RWK(MPI_STATUS_SIZE)
C
  JGB=N/NP
  JMOD=MOD(N, NP)
C
  JVBLK(1)=1
  DO J=1, NP-1
    IF (J-1 .LT. JMOD) THEN
      JT=1
    ELSE
      JT=0
    END IF
    IF (JGB+JT .GT. 0) THEN
      JVBLK(J+1)=JVBLK(J)+JGB+JT
    ELSE
      JVBLK(J+1)=N+1
    END IF
  ENDDO
  JVBLK(NP+1)=N+1

```

各 MPI プロセスの固有ベクトル開始位置
リストを作成


```

      ISIZE=N*JV
      DO ISEN=0, NP-1
        IF (IP .EQ. ISEN) THEN
          CALL MPI_ISEND(V, ISIZE, MPI_DOUBLE_COMPLEX,
*          0, ITAG, MPI_COMM_WORLD, REQ_S, IERR)
        END IF
        IF (IP .EQ. 0) THEN
          CALL MPI_IRECV(VREC, ISIZE, MPI_DOUBLE_COMPLEX,
*          ISEN, ITAG, MPI_COMM_WORLD, REQ_R, IERR)
          CALL MPI_WAIT(REQ_R, RWK, IERR)
          JS=JVBLK(ISEN+1)
          IF (JS .GT. N) THEN
            GO TO 2000
          END IF
          JE=JVBLK(ISEN+2)-1
          DO J=JS, JE
            J1=J-JS+1
            DO I=1, N
              V2(I, J)=VREC(I, J1)
            ENDDO
          ENDDO
        END IF
2000    CONTINUE
        IF (IP .EQ. ISEN) THEN
          CALL MPI_WAIT(REQ_S, SWK, IERR)
        END IF
        CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
      ENDDO
C
      RETURN
      END

```

ISEN に等しい MPI プロセスが、分散配置された固有ベクトルをプロセス 0 に send

ISEN に等しい MPI プロセスが send した固有ベクトルを、プロセス 0 で recv

recv した固有ベクトルを、全体の固有ベクトルへ復元

付録 3. 残差判定プログラム RESID

```

SUBROUTINE RESID(N, ZZ, ZG, ZX, LDA)
  IMPLICIT REAL*8(A-H, O-Y)
  IMPLICIT COMPLEX*16(Z)
  DIMENSION ZG(LDA, *), ZX(N, *)
  DIMENSION ZZ(N, *)

```

```

C

```

```

  COMPLEX*16 CWK(N)
  COMPLEX*16 CSUM
  COMPLEX*16 S2

```

```

*

```

```

C+++++CHECK1(NORMALIZE)

```

```

  SMAX=0.0D0
  DO K=1, N
    S=0.0D0
    DO I=1, N
      S=S+DREAL(ZX(I, K)*ZX(I, K))
    ENDDO
    S=DSQRT(S)
    IF (S.GT.SMAX) SMAX=S
  ENDDO
  WRITE(6, *) ' MAX. NORMALIZE NORM =', SMAX

```

$S = \sqrt{V \cdot V}$ から、固有ベクトルの正規性をチェック

```

C+++++CHECK2(UNITARY)

```

```

  SMAX=0.0D0
  DO K=1, N
    DO J=K+1, N
      S=0.0D0
      DO I=1, N
        S=S+ZX(I, J)*ZX(I, K)
      ENDDO
      S=DSQRT(ABS(S))
      IF (S.GT.SMAX) SMAX=S
    ENDDO
  ENDDO
  WRITE(6, *) ' MAX. UNITARY NORM =', SMAX

```

$S = \sqrt{V_i \cdot V_j}$ から、固有ベクトルの直交性をチェック

```

RMAX=-1.0D5
KMAX=0
DO K=1, N
  DO I=1, N
    CSUM=DCMPLX(0.0D0, 0.0D0)
    DO J=1, N
      CSUM=CSUM+ZZ(J, I)*ZX(J, K)
    ENDDO
    CWK(I)=CSUM
  ENDDO
  S2=0.0D0
  DO I=1, N
    CWK(I)=ZG(K, K)*ZX(I, K)-CWK(I)
    S2=S2+CWK(I)*CONJG(CWK(I))
  ENDDO
  RS1=CDABS(ZG(K, K))
  RS2=DSQRT(DREAL(S2))
  WRITE(6,*) ' COMPLEX RELATIVE RESID=', K, RS2/RS1
  IF (RS2/RS1 .GE. RMAX) THEN
    RMAX=RS2/RS1
    KMAX=K
  END IF
ENDDO
WRITE(6,*) ' ====='
WRITE(6,*) ' MAX COMPLEX RELATIVE RESID=', KMAX, RMAX
WRITE(6,*) ' ====='

```

$S=AV-\lambda V$ から、固有ベクトルの
残差をチェック

C

```

RETURN
END

```

付録 4. 係数行列作成プログラム MATGEN

```
      SUBROUTINE MATGEN(A, N)
C
      IMPLICIT REAL*8 (A-H, O-Z)
C
      COMPLEX*16 A(N, N)
C
      DO J=1, N
        A(J, J)=DCMPLX(J*1.0D0, J*1.0D0)
        DO I=J+1, N
          A(I, J)=DCMPLX(I*1.0D0, I*1.0D0)
          A(J, I)=A(I, J)
        ENDDO
      ENDDO
C
      RETURN
      END
```

付録 5. 係数行列分配プログラム MATDIV

```

SUBROUTINE MATDIV(MAJOR, AORG, A, LDA, NB, N, IP, NPROW, NPCOL)
C
C   IMPLICIT REAL*8 (A-H, O-Z)
C
C   CHARACTER*1 MAJOR
C   COMPLEX*16 A(LDA, *), AORG(N, N)
C
DO J=1, N
DO I=1, N
  IF (MAJOR .EQ. 'C') THEN
    IOWN=MOD((J-1)/NB, NPCOL)*NPCOL+MOD((I-1)/NB, NPROW)
  ELSE IF (MAJOR .EQ. 'R') THEN
    IOWN=MOD((I-1)/NB, NPROW)*NPROW+MOD((J-1)/NB, NPCOL)
  END IF
  IF (IP .EQ. IOWN) THEN
    IL=((I-1)/(NB*NPROW))*NB + MOD(I-1, NB) + 1
    JL=((J-1)/(NB*NPCOL))*NB + MOD(J-1, NB) + 1
    IF (J .LE. I) THEN
      A(IL, JL)=AORG(I, J)
    ELSE
      A(IL, JL)=DCMPLX(0.0D0, 0.0D0)
    END IF
  END IF
ENDDO
ENDDO
C
RETURN
END

```

3.2. 節の分配式に従い、担当する MPI プロセスを算出

係数行列の I, J を部分行列 A のローカルアドレスに変換し、要素を格納

付録 6. QR 法による固有値固有ベクトル求解プログラム SUB

```

SUBROUTINE SUB (N, ZG, ZX, IPRINT, LDA, D, E, ZWK, NOT)
C -----
  IMPLICIT REAL*8 (A-H, O-Y)
  IMPLICIT COMPLEX*16 (Z)
  REAL*8 TOLER, ANORM, EPS
  DIMENSION ZG (LDA, *), ZX (N, *), ZWK (NOT+1, NOT/10+5)
  COMPLEX*16 D (N), E (N)

  TOLER=1. 5D-60

200 CONTINUE

  DO I=1, N
    ZWK (I, 1)=D (I)
    ZWK (I+1, 2)=E (I)
  ENDDO

C
  WRITE (6, *) ' *** TRIDIAGONAL MATRIX ***'
  DO I=1, N
    WRITE (6, *) I, '      DIAG=', ZWK (I, 1)
    WRITE (6, *) I, ' SUB-DIAG=', ZWK (I+1, 2)
  ENDDO

  ANORM=0. DO
  ZTA = (0. DO, 0. DO)
  ZWK (N+1, 2) = (0. DO, 0. DO)
  DO 110 K=1, N
    ZT=ABS (ZWK (K, 1)) + ZTA
    TA=ABS (ZWK (K+1, 2))
    ZT=ZT+TA
    IF (DBLE (ZT) . GT. ANORM) ANORM=DBLE (ZT)
110 CONTINUE
  EPS = ANORM*TOLER
  WRITE (6, *) ' --- TRIDIAGONAL NORM =', ANORM

  ZAMU=(0. DO, 0. DO)
  ZWK (1, 2)=(0. DO, 0. DO)
  M=N
15 IF (M. EQ. 0) GO TO 21
  I = M-1
  M1= I
  K = I
  IF (ABS (ZWK (K+1, 2)) . GT. EPS) GO TO 16
  ZAMU=(0. DO, 0. DO)
  M = K
  GO TO 15

16 I=I-1
  IF (ABS (ZWK (I+1, 2)) . LE. EPS. OR. I. EQ. 0) GO TO 18
  K=I
  GO TO 16

18 ZRAMDA=(0. DO, 0. DO)
  IF (ABS (ZWK (M, 1) - ZAMU) . LT. 0. 5DO*ABS (ZWK (M, 1)) . OR. M1. EQ. K)
& ZRAMDA=ZWK (M, 1) + 0. 5DO*ZWK (M1+1, 2)
  ZAMU = ZWK (M, 1)
  WRITE (6, *) ' ZRAMDA=', ZRAMDA
  ZWK (K, 1) = ZWK (K, 1) - ZRAMDA
  ZBETA = ZWK (K+1, 2)

```

```

DO 19 J=K, M1
  ZAO=ZWK(J, 1)
  ZA1=ZWK(J+1, 1)-ZRAMDA
  ZBO=ZWK(J+1, 2)
  ZT=SQRT(ZAO**2+ZBETA**2)
  ZCOSINE=ZAO/ZT
  ZWK(J, 4)=ZCOSINE
  ZSINE=ZBETA/ZT
  ZWK(J, 5)=ZSINE
  ZWK(J, 1)=ZCOSINE*ZAO+ZSINE*ZBETA
  ZWK(J+1, 1)=-ZSINE*ZBO+ZCOSINE*ZA1
  ZWK(J+1, 2)=ZCOSINE*ZBO+ZSINE*ZA1
  ZBETA=ZWK(J+2, 2)
  ZWK(J+2, 2)=ZCOSINE*ZBETA
19 ZWK(J+2, 3)=ZSINE*ZBETA
  ZWK(K, 2) = (0. D0, 0. D0)
  ZWK(K+1, 3) = (0. D0, 0. D0)

DO 20 J=K, M1
  ZSINE=ZWK(J, 5)
  ZCOSINE=ZWK(J, 4)
  ZAO=ZWK(J, 1)
  ZBO=ZWK(J+1, 2)
  ZWK(J, 2)=ZWK(J, 2)*ZCOSINE+ZWK(J+1, 3)*ZSINE
  ZWK(J, 1)=ZAO*ZCOSINE+ZBO*ZSINE+ZRAMDA
  ZWK(J+1, 2)=-ZAO*ZSINE+ZBO*ZCOSINE
  ZWK(J+1, 1)=ZWK(J+1, 1)*ZCOSINE
DO 20 I=1, N
  ZX0=ZX(I, J)
  ZX1=ZX(I, J+1)
  ZX(I, J)=ZX0*ZCOSINE+ZX1*ZSINE
20 ZX(I, J+1)=-ZX0*ZSINE+ZX1*ZCOSINE
  ZWK(M, 1)=ZWK(M, 1)+ZRAMDA
GO TO 15
21 DO 120 M=1, N
120 ZG(M, M)=ZWK(M, 1)
  IF (IPRINT.EQ.0) RETURN    !OUTPUT LEVEL

DO 170 I=1, N, 4
  I6=MIN0(I+3, N)
  WRITE(6, 601) (ZG(J, J), J=I, I6)
  DO 160 J=1, N
160 WRITE(6, 602) J, (ZX(J, J9), J9=I, I6)
170 CONTINUE

C -----
601 FORMAT(1H0, 4X, 4(2X, '(', E13. 6, E15. 6, ')'))
602 FORMAT(1X, '(', I3, ')', F13. 6, 7F16. 6)
  RETURN
END

```