# TITPACK Ver. 2

## User's Manual

**Hidetoshi Nishimori**

## PREFACE

This manual explains the computer program package TITPACK Ver. 2 which is designed to numerically diagonalize the Hamiltonian of $S = 1/2$ quantum spin systems. The first version of TITPACK, released in 1985, has been used by many researchers in this field to investigate the ground-state properties. In the present new version, I have achieved significant improvements in speed, memory requirement, and portability by use of new coding techniques. I hope this new version will help us to enhance our understanding of quantum effects in spin systems.

I have been benefited by a number of useful suggestions since the release of Ver. 1. Especially, I thank here Dr. Masao Ogata at Institute for Solid State Physics, University of Tokyo, Mr. Misao Matsushita at Department of Physics, Chiba University (currently at CSK Research Institute), and the co-author of Ver. 1, Dr. Yoshihiro Taguchi at Department of Physics, Tokyo Institute of Technology. I also express my gratitude to Prof. Masatake Mori at Department of Physical Engineering, University of Tokyo, and Iwanami Shoten for approving the use of a part of programs appearing in Ref. 2.



April, 1991

Hidetoshi Nishimori

*Department of Physics*
*Tokyo Institute of Technology*
*Oh−okayama, Meguro−ku*
*Tokyo 152 − 8551*
*Japan*

nishimori@phys.titech.ac.jp

## CONTENTS

## §1. Introduction

TITPACK Ver. 2 is a set of programs to numerically diagonalize the following Hamiltonian with $S = 1/2$

$$H = -2 \sum_{<i,j>} J_{ij}(S_i^x S_j^x + S_i^y S_j^y + \Delta_{ij} S_i^z S_j^z) \ . \tag{1}$$

The four lowest eigenvalues and eigenstates can be calculated. If the matrix representation of the Hamiltonian has dimensionality small enough (roughly speaking, less than several hundred), all eigenvalues and eigenvectors can be calculated. The lattice structure and the range of interactions are arbitrary; the user can set these conditions by specifying the values of $J_{ij}$. No reduction in matrix size by use of translational or other symmetries is implemented because the program is designed for an arbitrary system including random cases. The Hamiltonian (1) is diagonalized in a subspace with a fixed value of $S_z^{\rm total}$.

The coding of TITPACK Ver. 2 follows the FORTRAN77 specifications with the exception of bit-wise logical operations: Function `IAND(i,j)` returns the result of a bit-wise AND operation of integers `i` and `j`; Function `IEOR(i,j)` yields the result of a bit-wise EXCLUSIVE OR of `i` and `j`. If your FORTRAN compiler does not support these functions, TITPACK Ver. 2 does not run as it is; you have to write these functions by assembly language.

**All real numbers in TITPACK Ver. 2 are of 8-byte length. The number of spins n must be less than 32 in TITPACK Ver. 2.**

### 1.1 Subroutines in TITPACK Ver. 2 and Their Functions

Subroutines for Large Matrices (Group L)
— To Diagonalize without Matrix Elements Stored

| Name | Function |
|---|---|
| **lnc1** | Eigenvalues by the Lanczos method |
| **lncv1** | Eigenvectors by the Lanczos method |
| **inv1** | Eigenvector by the inverse iteration method |
| **check1** | Precision check of eigenvalues and eigenvectors |

Subroutines for Mid-Size Matrices (Group M)
— To Store Nonzero Elements in Main Memory

| Name | Function |
|---|---|
| **elm2** | Calculation of nonzero matrix elements |
| **lnc2** | Eigenvalues by the Lanczos method |
| **lncv2** | Eigenvectors by the Lanczos method |
| **inv2** | Eigenvector by the inverse iteration method |
| **check2** | Precision check of eigenvalues and eigenvectors |

Subroutines for Small Matrices (Group S)
— To Store All Matrix Elements in Main Memory

| Name | Function |
|---|---|
| **elm3** | Calculation of matrix elements |
| **diag** | Eigenvalues and vectors by the Householder and inverse iteration methods |
| **check3** | Precision check of eigenvalues and eigenvectors |

Commonly Used Subroutines

| Name | Function |
|---|---|
| **sz** | Generation of spin configurations |
| **xcorr** | Correlation functions including $S_x$ and $S_y$ |
| **zcorr** | Correlation functions including $S_z$ |
| **orthg** | Degeneracy check by orthogonalization |

Subroutines Called Internally in the Above Routines

| Name | Function |
|---|---|
| **lnc1z** | Execution of the Lanczos process in Group L (eigenvalues) |
| **lncv1z** | Execution of the Lanczos process in Group L (eigenvectors) |
| **mltply** | Matrix-vector multiplication in Group L |
| **inv1z** | Execution of inverse iteration in Group L |
| **cg1** | Solution of linear equations by the Conjugate Gradient method in Group L |
| **lnc2z** | Execution of the Lanczos process in Group M (eigenvalues) |
| **lncv2z** | Execution of the Lanczos process in Group M (eigenvectors) |
| **inv2z** | Execution of inverse iteration in Group M |
| **cg2** | Solution of linear equations by the Conjugate Gradient method in Group M |
| **hshldr** | Tridiagonalization by the Householder method |
| **vec3** | Eigenvectors by inverse iteration and LU decomposition |
| **datack** | Error check of site data |
| **bisec** | Eigenvalues of tridiagonal matrix by the bisection method |
| **vec12** | Eigenvector of tridiagonal matrix by inverse iteration and LU decomposition |

The relationship between these subroutines will be shown in the following sections.

### 1.2 Choosing Subroutine Groups

The following convention is useful to determine which subroutine group among Large, Medium, or Small is appropriate for your purposes. First, calculate the matrix dimension `idim`. Let `n` denote the number of spins. For a given value of $S_z^{\rm total}$, the number of up-spins, iu, and the number of down spins, id, satisfy the relations iu+id=`n`, (iu−id)/2=$S_z^{\rm total}$. Then one obtains iu=`n`/2−$S_z^{\rm total}$, and the matrix dimension is calculated as the number of combinations to choose iu out of `n`, `idim`=${}_{\rm n}{\rm C}_{\rm iu}$.

If `idim` is smaller than 100, use Group S. When `idim` exceeds 100, use either Group M or L. Group M routines run faster than those in L. However, the former require more memory to store nonzero matrix elements; Group L routines evaluate necessary matrix elements at every step of diagonalization (thus leading to a significantly longer CPU time). The memory requirement for Group M is

$$(\text{idim} * \text{ibond} * 1.5 * 8)/1024^2 \quad \text{(MB)},$$

where `ibond` denotes the number of nonzero bonds ($J_{ij}$).

ENTER DIAGRAM NO.1 HERE

### 1.3 Process Diagram for Large Matrices (Group L)

ENTER DIAGRAM NO.2 HERE

### 1.4 Process Diagram for Mid-Size Matrices (Group M)

ENTER DIAGRAM NO.3 HERE

### 1.5 Process Diagram for Small Matrices (Group S)

ENTER DIAGRAM NO.4 HERE

### 1.6 Benchmark Test

The following is the list of necessary main memory space and CPU time on various computes, HITAC S-820/80 (vector processor), FACOM VP-2600 (vector processor), HITAC M-880/310 (general purpose scalar machine) and NEC EWS-4800/220 (work station). The Hamiltonian is the one-dimensional Heisenberg antiferromagnet with nearest neighbor interactions. The four lowest eigenvalues in the space $S_z^{\rm total} = 0$ and the ground-state eigenvector have been calculated with the initial condition `iv=idim/3` (see the item on `lnc1` in section 2.1 for details). All inner do loops have been automatically vectorized by the compiler on vector processors. It is observed that the Group M routines utilize vector processors more efficiently than those of Group L.

The memory requirement in other model systems are estimated as follows. The necessary memory space is proportional to `idim` (matrix dimension) in Group L, proportional to the product of `idim` and `ibond` (number of bonds) in Group M, and proportional to the square of `idim` in Group S. The CPU time is proportional to the product of `idim` and `ibond` in Groups L and M, and is proportional to the cube of `idim` in Group S.

**Legend**

I : Eigenvalues only by `lnc1` (Group L)
II : Eigenvalues only by `lnc2` (Group M)
III : Eigenvalues only by `diag` (Group S)
Iv : Eigenvalues and an eigenvector by `lnc1` and `lncv1` (Group L)
Iv′ : Eigenvalues and an eigenvector by `lnc1` and `inv1` (Group L)
IIv : Eigenvalues and an eigenvector by `lnc2` and `lncv2` (Group M)
IIv′ : Eigenvalues and an eigenvector by `lnc1` and `inv1` (Group M)
IIIv : Eigenvalues and an eigenvector by `diag` (Group S)

Main memory requirement (in MB; − : less than 1MB; ∗ : exceeding 1GB)

| n | idim(matrix size) | I | II | III | Iv | Iv′ | IIv | IIv′ | IIIv |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 70 | − | − | − | − | − | − | − | − |
| 12 | 924 | − | − | 6.6 | − | − | − | − | 6.6 |
| 16 | 12870 | − | 2.7 | ∗ | − | − | 2.8 | 3.0 | ∗ |
| 20 | 184756 | 3.5 | 48 | ∗ | 4.9 | 7.8 | 49 | 51 | ∗ |
| 24 | 2704156 | 52 | 701 | ∗ | 72 | 113 | 836 | 877 | ∗ |
| 26 | 10400600 | 198 | ∗ | ∗ | 277 | 436 | ∗ | ∗ | ∗ |

CPU time on S-820/80 (sec)
(− : under 1 sec; ∗ : unexecutable for insufficient memory)

| n | sz | I | II | III | Iv | Iv′ | IIv | IIv′ | IIIv |
|---|---|---|---|---|---|---|---|---|---|
| 8 | − | − | − | − | − | − | − | − | − |
| 12 | − | − | − | 2 | − | − | − | − | 2 |
| 16 | − | − | − | ∗ | − | 1 | − | − | ∗ |
| 20 | 5 | 10 | 1 | ∗ | 20 | 38 | 2 | 4 | ∗ |
| 24 | 80 | 225 | ∗ | ∗ | 448 | 839 | ∗ | ∗ | ∗ |
| 26 | 332 | 883 | ∗ | ∗ | 1758 | 2780 | ∗ | ∗ | ∗ |

CPU time on VP-2600 (sec)
(− : under 1 sec; ∗ : unexecutable for insufficient memory)

| n | sz | I | II | III | Iv | Iv′ | IIv | IIv′ | IIIv |
|---|---|---|---|---|---|---|---|---|---|
| 8 | − | − | − | − | − | − | − | − | − |
| 12 | − | − | − | 2 | − | − | − | − | 2 |
| 16 | − | − | − | ∗ | − | 1 | − | − | ∗ |
| 20 | 6 | 8 | 2 | ∗ | 15 | 27 | 3 | 5 | ∗ |
| 24 | 108 | 155 | ∗ | ∗ | 306 | 565 | ∗ | ∗ | ∗ |

CPU time on M-880/310 (sec)
(− : under 1 sec; ∗ : unexecutable for insufficient memory)

| n | sz | I | II | III | Iv | Iv′ | IIv | IIv′ | IIIv |
|---|---|---|---|---|---|---|---|---|---|
| 8 | − | − | − | − | − | − | − | − | − |
| 12 | − | − | − | 48 | − | − | − | − | 48 |
| 16 | 1 | 5 | 2 | ∗ | 11 | 21 | 4 | 11 | ∗ |
| 20 | 24 | 144 | 69 | ∗ | 290 | 503 | 131 | 283 | ∗ |

CPU time on EWS-4800/220 (sec)
(− : under 1 sec; ∗ : unexecutable for insufficient memory)

| n | sz | I | II | III | Iv | Iv′ | IIv | IIv′ | IIIv |
|---|---|---|---|---|---|---|---|---|---|
| 8 | − | − | − | − | − | − | − | − | − |
| 12 | − | 1 | 1 | 816 | 2 | 4 | 2 | 3 | 816 |
| 16 | 5 | 34 | 21 | ∗ | 68 | 128 | 41 | 95 | ∗ |
| 20 | 104 | 852 | ∗ | ∗ | 1710 | 2988 | ∗ | ∗ | ∗ |

## §2. Description of the Routines

The detailed explanation of subroutines of all Groups is given in this section. **All real numbers in TITPACK Ver. 2 are of 8-byte length. The number of spins n must be less than 32 in TITPACK Ver. 2.**

### 2.1 Subroutines for Large Matrices – Group L

In Group L, a matrix is tridiagonalized by the Lanczos method. Matrix elements are calculated in each step of the Lanczos iteration. The eigenvalues of the resulting tridiagonal matrix are obtained by the bisection method. Eigenvectors are calculated either by repeating the Lanczos process or by the inverse iteration method. In the latter case, each step of inverse iteration is executed by the Conjugate Gradient method (see Appendix B).

The requirement of main memory space in Group L is much less than that in Groups M and S since one does not have to store matrix elements. The rough estimate of necessary memory is given as the memory space to store two and a half vectors (of 8-byte length) if you need only eigenvalues, and (2.5+`nvec`) vectors for `nvec` eigenvectors. One should remember that the routines in Group L require more CPU time than those in Group M because of matrix evaluation in the process of diagonalization. Sample data of memory and CPU time are listed in section 1.6.

**Sample Main Program**
(Four eigenvalues and the ground-state eigenvector with precision check and evaluation of correlation functions)

ENTER SAMPLE PROGRAM NO.1 HERE

ENTER SAMPLE PROGRAM NO.1 HERE.

(1)Generation of Spin Configurations

**call sz(n, idim, szval, list1, list2) (Equivalent routines szdy and sztn are faster)**

**Function**

Generate all spin configurations in the space determined by the number of spins `n` and $S_z^{\rm total}$=`szval`. The spin configurations are numbered consecutively according to their binary representations. A table to yield the spin configuration from the serial number (`list1`) and an inverse table to give the serial number from the spin configuration (`list2`) are generated. `sz` is used in all Groups and should be called prior to any other routines.

**Arguments**

**n**: (input; integer)

Number of spins.

**idim**: (input; integer)

Dimensionality of the space specified by `n` and `szval`, which is also the size of the matrix to be diagonalized. `idim` must exceed two. `idim` is calculated as follows. Let the number of up-spins be iu and down-spins id. These satisfy iu+id=`n` and (iu−id)/2=`szval`, from which iu=`n`/2−`szval`. `idim` is the number of combinations to choose iu out of `n`: `idim`=${}_{\rm n}{\rm C}_{\rm iu}$. For instance, if `n` =20 and `szval`=0.0d0, the number of up-spins is 10 and `idim` is equal to ${}_{20}{\rm C}_{10} = 184,756$.

**szval**: (input; real)

The value of $S_z^{\rm total}$. Must be larger than or equal to 0.0d0. If you wish to treat a space with negative $S_z^{\rm total}$, you must exchange up- and down-spins to translate $S_z^{\rm total}$ to $-S_z^{\rm total}$. For example, the space with $S_z^{\rm total} = -1.0{\rm d}0$ has the same eigenvalues with the space of $S_z^{\rm total} = 1.0{\rm d}0$. The eigenvectors of the former space are readily obtained from the latter by simply exchanging up-spins and down-spins.

**list1(idim)**: (output; integer)

`list1(j)` gives the spin configuration with the serial number `j` (explained below) in the space under consideration. The spin configuration is represented by the binary expression of `list1(j)` with 1 corresponding to an up-spin and 0 to a down-spin. The 0th bit represents the spin state of site 1, the 1st bit corresponds to that of site 2, and so on. The serial number is given in increasing order of the value of `list1(j)`. For instance, if `n`=4 and `szval`=0.0d0, there are two up-spins (represented by 1) and two down-spins (represented by 0). Thus `idim`=${}_4{\rm C}_2$=6, and `list1(1)`=$(0011)_{\rm binary}$=3, `list1(2)`=$(0101)_{\rm binary}$ = 5, $\cdots$, `list1(6)`=$(1100)_{\rm binary}$=12.

**list2(2, 0:2**15)**: (output; integer)

The inverse table of `list1` to give the serial number of a spin configuration. That is, if a spin configuration represented by a decimal number `i` has the serial number `j`, then `j` satisfies `j=list2(1,ia)+list2(2,ib)`. Here, `ia` is the lower `n`/2 bits of the integer `i (ia=IAND(i,k-1)` with `k=2**((n+1)/2))`, and `ib` is the upper `n`/2 bits of `i` (`ib` = `IAND(i,IEOR(2**n-1,k-1))/k`).

This is the two-dimensional search technique as explained in section 4.2. The decomposition of an integer into two parts, the upper and lower bits, helps to greatly reduce the memory space.

**Remarks**

`list1` and `list2` form the basis of matrix manipulations and must be transferred to other diagonalization routines. Vectorization rate by FORTRAN compiler for vector processors is not generally very good in `sz`, because `sz` sequentially generates spin configurations and checks their consistency with `idim`, `n`, and `szval` in a sequential manner. Hence, if you intend to diagonalize many matrices with the same `list1` and `list2`, *i.e.* with the same `n` and `szval`, it is better to perform `sz` only once and store `list1` and `list2` in an external device for later use. This remark does not apply if the I/O processing is not fast on your computer, like many workstations.

**call lnc1(n, idim, ipair, bondwt, zrtio, ibond, nvec, iv, E, itr, wk, ideclr, list1, list2)**

## Function

`lnc1` returns four lowest eigenvalues of a given Hamiltonian and spin configurations. It also calculates data necessary for eigenvector evaluation in `lncv1`.

## Arguments

**n**: (input; integer)

Number of spins.

**idim**: (input; integer)

Dimensionality of the matrix.

**ipair(ibond*2)**: (input; integer)

Lattice structure. The user specifies the lattice structure in terms of pairs of sites connected by nonzero bonds. For instance,

```
data ipair/1,2, 2,3, 3,4, 4,1, 1,3, 2,4/
```

means that site 1 is connected to site 2, 2 to 3,.., and 2 to 4.

**bondwt(ibond)**: (input; real)

Exchange interaction $J_{ij}$ of each bond $(ij)$. In the example in the above data statement, the exchange interactions $J_{12} = -1,\ J_{23} = -1.5,\ J_{34} = 0.3,\ J_{41} = -0.6,\ J_{13} = 0,\ J_{24} = 1$ are given by specifying `bondwt` as

```
data bondwt/-1.0d0, -1.5d0, 0.3d0, -0.6d0, 0.0d0, 1.0d0/
```

Note that the data sequence in the array `bondwt` should correspond to that in `ipair`. For instance, if you replace the above data statement by

```
data bondwt/-1.0d0, -1.5d0, 0.3d0, -0.6d0, 1.0d0, 0.0d0/,
```

then you will get results for $J_{13} = 1,\ J_{24} = 0$.

**zrtio(ibond)**: (input; real)

Anisotropy parameter $\Delta_{ij}$. The data sequence in this array should also follow that in `ipair`.

**ibond**: (input; integer)

Total number of nonzero bonds.

**nvec**: (input; integer)

Number of eigenvectors to calculate later in `lncv1`. Should be between 0 (no eigenvector) and 4. `lnc1` calculates data necessary for performing `lncv1` and writes the results in internal variables defined in `common block/vecdat/`. Thus, if you wish to calculate eigenvectors by `lncv1`, you should call `lnc1` beforehand with nonzero `nvec`.

**iv**: (input; integer)

`iv` specifies the initial vector of the Lanczos iteration. The `iv`th component of the initial vector is set to 1, and other components to 0. `iv` should be between 1 and `idim`. The numbering of the basis of a vector follows the convention explained in the item on the `sz` routine. The convergence rate of the Lanczos iteration is only weakly dependent on `iv` in most cases. Special initial conditions such as `iv`=1 or `iv=idim` may be exceptions because `iv`=1 and `iv=idim` represent very special spin configurations. Generally, intermediate numbers between these limits lead to reliable results. See also remarks below.

**E(4)**: (output; real)

Four lowest eigenvalues. The lowest two eigenvalues `E(1)` and `E(2)` have generally about 10 significant digits. The numbers of significant digits of the next two eigenvalues `E(3)` and `E(4)` depend on the Hamiltonian. I strongly suggest the user to check the reliability of these higher eigenvalues by the `check1` routine. `check1` is also useful to know the precise reliability of `E(1)` and `E(2)`.

**itr**: (output; integer)

The number of iterations for convergence of the Lanczos method. Given as a multiple of 5 above 20.

**wk(ideclr, 2)**: (working area; real)

**ideclr**: (input; integer)

Adjustable dimension of the working area. Must satisfy `ideclr`$\geq$`idim`.

**list1(idim)**: (input; integer)

Spin configuration given by `sz`.

**list2(2, 0:2**15)**: (input; integer)

Spin configuration given by `sz`.

**Remarks**

Convergence check starts at the 25th step and is performed every five steps. The Lanczos iteration is finished when `E(2)` converges with the relative precision of $10^{-13}$. The ground state energy `E(1)` is obtained with sufficient stability unless the initial vector is strictly orthogonal to the ground state. The other eigenvalues are sometimes strongly affected by rounding errors, leading to a spurious degeneracy such as `E(2)=E(3)` when `E(2)` is actually different from `E(3)`. This instability of higher eigenvalues is characteristic of the Lanczos method, and the user should be cautious about degeneracy of eigenvalues given by the Lanczos method. A safe way to know the true degree of degeneracy of an eigenvalue is to calculate several eigenvectors starting from different initial conditions and check their orthogonality by the `orthg` routine. This comment applies also to the case in which `E(1)` is different from `E(2)`; as seen in sample 10 of Appendix C, the ground state may be degenerate even when `E(1)` is not equal to `E(2)`.

**call lncv1(n, idim, ipair, bondwt, zrtio, ibond, nvec, iv, x, itr, wk, ideclr, list1, list2)**

## Function

Calculate eigenvectors by the Lanczos method. The user should execute `lnc1` beforehand with `nvec`$\neq$0 so that data used in `lncv1` are written in internal variables defined in `common block/vecdat/`.

## Arguments

Same as in `lnc1` except for `x`. The arguments `nvec, iv, itr` should assume the same values as in `lnc1`. The arguments specifying the Hamiltonian, `n, idim, ipair, bondwt, zrtio, ibond` must of course assume the same values as in `lnc1`.

**x(ideclr, nvec)**: (output; real)

Eigenvectors. The number of eigenvectors `nvec` must be in the range between 1 and 4. The components of the ground-state eigenvector are given from `x(1,1)` to `x(idim,1)`, those corresponding to `E(2)` are from `x(1,2)` to `x(idim,2)`, and so on. Each number in the array `x` represents the corresponding amplitude of the eigenvector. For instance, in the example in the item on `list1` in the `sz` routine,

$$\psi_{\text{ground}} = \mathrm{x}(1,1) \times (0011)_{\text{binary}} + \mathrm{x}(2,1) \times (0101)_{\text{binary}} \cdots$$

All wave functions are normalized to 1.

## Remarks

The ground-state eigenvector, from `x(1,1)` to `x(idim,1)`, usually have more than 6 significant digits. However, the precision is not very high for other eigenvectors. Precision check can be executed by `check1`. If a very accurate eigenvector is necessary, it is better to use `inv1` instead of `lncv1`.

The precision of the expectation value of a physical quantity often exceeds that of the components of an eigenvector. This comes from cancellation of errors in evaluating the expectation value. The best example is the energy, or the expectation value of the Hamiltonian; the bilinear form $\langle \psi H \psi \rangle$ is extremum at $\psi = \psi_{\text{eigenstate}}$, which implies stability of the expectation value against errors. This stability is often shared, empirically, by the expectation values of other quantities.

**call inv1(n, idim, ipair, bondwt, zrtio, ibond,**
**Eig, iv, x, wk, ideclr, list1, list2)**

## Function

Calculate the eigenvector corresponding to a given (approximate) eigenvalue `Eig` by the inverse iteration method. Each step of inverse iteration is performed by the Conjugate Gradient method. Only one eigenvector is calculated by `inv1`, in contrast to `lncv1` in which up to four vectors can be obtained in a single call. It is not necessary to execute `lnc1` with nonzero `nvec` beforehand to prepare internal variables, in contrast to `lncv1`, if the eigenvalue `Eig` is known by other methods.

## Arguments

Same as in `lncv1` except for `Eig, iv, x, wk`.

**Eig**: (input; real)

Eigenvalue. The eigenvector corresponding to `Eig` is calculated. It is not necessary that `Eig` has been obtained by `lnc1`; the eigenvector for any eigenvalue, including higher levels, can be calculated. The convergence is faster if the precision of the eigenvalue is better.

**iv**: (input; integer)

Initial vector of inverse iteration. See the item on `iv` in `lnc1`. It is not required to use the same `iv` as in `lnc1`, in contrast to `lncv1`.

**x(idim)**: (output; real)

Eigenvector. The amplitudes are given in the same order as in `lncv1`. See the explanation on `lncv1`.

**wk(ideclr, 4)**: (working area; real)

## Remarks

The components of an eigenvector obtained by `inv1` typically has 8 to 10 significant digits, which is better than those by `lncv1` by several digits. `check1` gives you an idea on precision of an eigenvector. Disadvantages in comparison with `lncv1` are: (i) Slower speed (see section 1.6). (ii) Larger working area. `inv1` requires `wk(ideclr,4)` while `lncv1` needs `wk(ideclr,2)`. (iii) Only one eigenvector is obtained in a single call. The user should determine which of `inv1` and `lncv1` to use depending upon his/her purposes.

(5)Precision Check of an Eigenvalue and an Eigenvector

**call check1(n, idim, ipair, bondwt, zrtio, ibond, x, v, Hexpec, list1, list2)**

**Function**

Return the result, `v`, of multiplication of the Hamiltonian to a given eigenvector `x`. The expectation value of the Hamiltonian by `x` is also returned in `Hexpec`. Furthermore, `check1` prints the value of `Hexpec` and the ratios `v(j)/x(j)` from `j=min(idim/3,13)` to `idim` in step of `max(1,idim/20)`. This ratio `v(j)/x(j)` should be equal to the eigenvalue if `x` is the correct eigenvector. Thus the actual values of the ratio give the reliability of the eigenvector.

**Arguments**

Same as in `lnc1` except for `x,v,Hexpec`.

**x(idim)**: (input; real)

Normalized eigenvector whose precision check is executed.

**v(idim)**: (output; real)

Result of multiplication of the Hamiltonian to `x`.

**Hexpec**: (output; real)

Expectation value of the Hamiltonian by the given eigenvector `x`.

**Remarks**

Precision of an eigenvalue is determined by comparison of `Hexpec` and the value given by `lnc1`. As mentioned in the item on `lncv1`, `Hexpec` is more reliable than the value from `lnc1`. At least, digits common between `Hexpec` and that from `lnc1` are significant. Reliability of the components of an eigenvector is determined by the ratio `v(j)/x(j)`. The user should remember that components with smaller amplitudes have less number of significant digits, leading to larger deviation of `v(j)/x(j)` from `Hexpec`. The components with smaller amplitudes have relatively small contribution to the expectation value of a physical quantity, and therefore, smallness of the number of significant digits is not necessarily a disturbing factor. For instance, when `x(10)=1.23456789d-10` and `x(20)=9.0d-20`, the final expectation value of a physical quantity is expected to be reliable up to approximately 9 digits if other components of `x` than `x(20)` have comparable reliability to `x(10)`.

(6)Two-Point Correlation Function Including $S_x, S_y$

**call xcorr(n, idim, npair, nbond, x, sxx, list1, list2)**

**Function**

Two-point correlation functions in the $xy$-plane are calculated for a given eigenvector `x`. This routine is used in all Groups, L, M, and S.

**Arguments**

Same as in `lnc1` except for `npair,nbond,x,sxx`.

**npair(nbond*2)**: (input; integer)

Pairs of sites for which two-point correlation functions are calculated. Give the site numbers in the same way as in `ipair` in `lnc1`.

**nbond**: (input; integer)

Number of pairs for which two-point correlation functions are calculated.

**x(idim)**: (input; real)

Normalized eigenvector.

**sxx(nbond)**: (output; real)

Values of two-point correlation functions. The value of $\langle S^x_{\mathrm{npair}(1)} S^x_{\mathrm{npair}(2)} \rangle$ is given in sxx(1), that of $\langle S^x_{\mathrm{npair}(3)} S^x_{\mathrm{npair}(4)} \rangle$ is in sxx(2), and so on. $\langle S^y_i S^y_j \rangle$ is equal to $\langle S^x_i S^x_j \rangle$ by symmetry of the Hamiltonian.

**Remark**

The pairs of sites specified by `npair` are independent of `ipair`. `xcorr` calculates correlation functions for any pairs of sites between 1 and `n`.

(7)Two-Point Correlation Function Including $S_z$

**call zcorr(n, idim, npair, nbond, x, szz, list1)**

**Function**

Two-point correlation functions along the $z$-axis are calculated for a given eigenvector `x`. This routine is used in all Groups, L, M, and S.

**Arguments**

Same as in `xcorr` except for `szz`.

**szz(nbond)**: (output; real)

Values of two-point correlation functions. The value of $\langle S^z_{\mathrm{npair}(1)} S^z_{\mathrm{npair}(2)} \rangle$ is returned in szz(1), and so on, similarly to `xcorr`.

**Remark**

Same as in `xcorr`.

**call orthg(idim, ideclr, ev, norm, idgn, numvec)**

## Function

Orthogonalize several vectors to check linear independence and degeneracy of an energy eigenvalue. The user can estimate degeneracy of an eigenvalue by calculating several eigenvectors, corresponding to the same eigenvalue, starting from different initial conditions `iv` in `lncv1` or `inv1`. The number of linearly independent vectors thus obtained gives degeneracy. (One should note that `orthg` is not used in the sample program shown in the first page of this section. See samples 5 and 10 in Appendix C.)

## Arguments

**idim**: (input; integer)

Matrix dimension.

**ideclr**: (input; integer)

Adjustable dimension of `ev`.

**ev(ideclr, numvec)**: (input and output; real)

Input: Vectors of which orthogonalization is executed. Output: Orthogonalized vectors.

**norm(numvec)**: (output; integer)

Norm of orthogonalized vectors. Given in 1 or 0. The norm of the first vector is in `norm(1)`, the second is in `norm(2)`, and so on.

**idgn**:(output; integer)

Degeneracy. The number of entry 1 in the array `norm`.

**numvec**:(input;integer)

Number of vectors to be checked.

## Remarks

`Orthg` regards a vector of norm less than $10^{-15}$ as a null vector after orthogonalization. `Orthg` calculates inner products of orthogonalized vectors to evaluate round-off errors. If the inner products are all less than $10^{-10}$, the process is returned normally to the main program. Otherwise, a warning message is issued. If the user tries to orthogonalize too many vectors or vectors with low precision, the latter case (a warning) may occur.

### 2.2 Subroutines for Mid-Size Matrices – Group M

Subroutines in this group first tridiagonalize a matrix by the Lanczos method with nonzero matrix elements calculated beforehand. Eigenvalues of the resulting tridiagonal matrix are calculated by the bisection method. Eigenvectors are obtained either by repeating the Lanczos iteration or by the inverse iteration method. In the latter case, each step of inverse iteration is executed by the Conjugate Gradient method.

Routines in Group M use nonzero matrix elements stored in main memory, and therefore require significantly larger memory space than those in Group L. Processing speed is much faster than in Group L, since it is not necessary to calculate matrix elements in each step of diagonalization. Therefore, it is better to use Group M than Group L as long as memory space to store nonzero elements is available. Examples of memory requirement and CPU time are listed in section 1.6.

**Sample Main Program**
(Four eigenvalues and the ground-state eigenvector with precision check and evaluation of correlation functions)

ENTER SAMPLE PROGRAM NO.6 HERE

ENTER SAMPLE PROGRAM NO.6 HERE.

**call elm2(n, idim, ipair, bondwt, zrtio, ibond,**
**elemnt, loc, ideclr, ic, list1, list2)**

**Function**

The locations and values of nonzero matrix elements are returned in `loc` and `elemnt` for a given Hamiltonian and spin configurations.

**Arguments**

Arguments `n,idim,ipair,bondwt,zrtio,ibond,list1,list2` have the same meaning as in `sz` and `lnc1` in section 2.1.

**elemnt(ideclr, ic)**: (output; real)

Values of nonzero matrix elements. The $k$th nonzero element in the $j$th row is `elemnt(j,k)`. The serial number $k$ within a row does not follow the number of column; instead it is the number of bond specified in `ipair`. That is, `elemnt(j,1)` is the value of nonzero element resulting from the operation of the off-diagonal term in the Hamiltonian $-2J_{ij}(S^x_{\mathrm{ipair}(1)}S^x_{\mathrm{ipair}(2)} + S^y_{\mathrm{ipair}(1)}S^y_{\mathrm{ipair}(2)})$ upon the spin configuration represented by `list1(j)`. Similarly, `elemnt(j,2)` is the off-diagonal term corresponding to `ipair(3)` and `ipair(4)`, and so on. The diagonal element is stored in `elemnt(j,ic)`.

**loc(ideclr, ic)**: (output; integer)

The location of nonzero matrix elements. The $k$th nonzero element in the $j$th row is located at the column `loc(j,k)`. In other words, the value of the matrix element at the $j$th row, and the `loc(j,k)`th column is `elemnt(j,k)`.

**ideclr**: (input; integer)

Adjustable dimension of `elemnt` and `loc`.

**ic**: (input; integer)

Enter `ibond+1`. This is the maximum number of nonzero elements in a row.

**call lnc2(elemnt, loc, idim, ideclr, ic, nvec, iv, E, itr, wk)**

**Function**

`lnc2` returns four lowest eigenvalues of a given matrix. It also calculates data necessary for eigenvector evaluation in `lncv2`. Matrix elements of the Hamiltonian must be calculated by `elm2` before `lnc2` is called.

**Arguments**

**elemnt(ideclr, ic)**: (input; real)

Values of nonzero matrix elements obtained in `elm2`. Elements of `elemnt` have their original values after the processing returns to the main routine from `lnc2`.

**loc(ideclr, ic)**: (input; real)

Locations of nonzero matrix elements obtained in `elm2`. Elements of `loc` have their original values after the processing returns to the main routine from `lnc2`.

**idim**: (input; integer)

Matrix dimension.

**ideclr**: (input; integer)

Adjustable dimension of `elemnt,loc,wk`.

**ic**: (input; integer)

Enter `ibond+1`. The maximum number of nonzero matrix elements in a row.

**nvec**: (input; integer)

Number of eigenvectors to calculate later in `lncv2`. Should be between 0 (no eigenvector) and 4. `lnc2` calculates data necessary for executing `lncv2` and writes the results in internal variables defined in `common block/vecdat/`. Thus, if you wish to calculate eigenvectors by `lncv2`, you should call `lnc2` beforehand with nonzero `nvec`.

**iv**: (input; integer)

`iv` specifies the initial vector of the Lanczos iteration. The `iv`th component of the initial vector is set to 1, and other components to 0. `iv` should be between 1 and `idim`. See also the item on `iv` in `lnc1`, section 2.1.

**E(4)**: (output; real)

Four lowest eigenvalues. The lowest two eigenvalues `E(1)` and `E(2)` have generally about 10 significant digits. The numbers of significant digits of the next two eigenvalues `E(3)` and `E(4)` depend on the Hamiltonian. I strongly suggest the user to check reliability of these higher eigenvalues by the `check2` routine. `check2` is also useful to know the precise reliability of `E(1)` and `E(2)`.

**itr**: (output; integer)

Number of iterations needed for convergence of the Lanczos method. Given in a multiple of 5 above 20.

**wk(ideclr, 2)**: (working area; real)

**Remark**

Same as in `lnc1`.

**call lncv2(elemnt, loc, idim, ideclr, ic, nvec, iv, x, itr, wk)**

## Function

Calculate eigenvectors by the Lanczos method. The user should execute `lnc2` beforehand with `nvec`$\neq$0 so that data used in `lncv2` are written in internal variables defined in `common block/vecdat/`.

## Arguments

Same as in `lnc2` except for `x`. Arguments `nvec, iv, itr` should assume the same values as in `lnc2`. Matrix elements in `elemnt,loc` must of course assume the same values as in `lnc2`.

**x(ideclr, nvec)**: (output; real)

Normalized eigenvectors. See `lncv1` in section 2.1 for details.

## Remark

Same as in `lncv1`.

**call inv2(elemnt, loc, idim, ideclr, ic, iv, Eig, x, wk)**

**Function**

Calculate the eigenvector corresponding to a given (approximate) eigenvalue `Eig` by the inverse iteration method. Each step of inverse iteration is executed by the Conjugate Gradient method. Only one eigenvector is calculated by `inv2`, in contrast to `lncv2` in which up to four vectors can be obtained in a single call. It is not necessary to execute `lnc2` with nonzero `nvec` beforehand to prepare internal variables, in contrast to `lncv2`, if the eigenvalue `Eig` is known by other methods.

**Arguments**

Same as in `lncv2` except for `Eig, iv, x, wk`.

**Eig**: (input; real)

Eigenvalue. The eigenvector corresponding to `Eig` is calculated. It is not necessary that `Eig` has been obtained by `lnc2`; the eigenvector of any eigenvalue, including higher levels, can be calculated. Convergence is faster if the precision of the eigenvector is better.

**iv**: (input; integer)

Initial vector of inverse iteration. See the item on `iv` in `lnc1` in section 2.1. It is not necessary that the value of `iv` be the same as in `lnc2`, in contrast to `lncv2`.

**x(idim)**: (output; real)

Normalized eigenvector. The amplitudes are given in the same way as in `lncv2`. See also explanations of `lncv1`.

**wk(ideclr, 4)**: (working area; real)

**Remark**

Same as in `inv1`.

(5)Precision Check of an Eigenvalue and an Eigenvector

**call check2(elemnt, loc, idim, ideclr, ic, x, v, Hexpec)**

**Function**

Return the result, `v`, of multiplication of the Hamiltonian to a given eigenvector `x`. The expectation value of the Hamiltonian by `x` is also returned in `Hexpec`. Furthermore, `check2` prints the value of `Hexpec` and the ratios `v(j)/x(j)` from `j=min(idim/3,13)` to `idim` in step of `max(1,idim/20)`. This ratio `v(j)/x(j)` should be equal to the eigenvalue if `x` is the correct eigenvector. Thus the actual values of the ratio give the reliability of the eigenvector.

**Arguments**

Same as in `lnc2` except for `x,v,Hexpec`.

**x(idim)**: (input; real)

Normalized eigenvector whose precision check is performed.

**v(idim)**: (output; real)

Result of multiplication of the Hamiltonian to `x`.

**Hexpec**: (output; real)

Expectation value of the Hamiltonian by the given eigenvector `x`.

**Remark**

Same as in `check1`.

### 2.3 Subroutines for Small Matrices – Group S

Subroutines in Group S tridiagonalize the Hamiltonian matrix by first calculating all matrix elements and storing them in the main memory. The matrix dimension must exceed two. Eigenvalues of a tridiagonal matrix are obtained by the bisection method. Eigenvectors are calculated by the inverse iteration method, each step of which is executed by the LU decomposition technique of linear equations.

Memory requirement is much larger than in Groups L and M because all matrix elements are stored. The required memory space is proportional to the square of the matrix dimension `idim`. This Group S is useful when `idim` is small (generally, less than 100), since the Lanczos method used in Groups L and M is sometimes unstable for small matrices. If the matrix dimension exceeds 100, Group S routines are generally slower and less memory-efficient than those in Groups L and M for the purpose of calculating several low-lying levels. Group S routines in TITPACK Ver. 2 are not optimized for vector processors; it might consume very large CPU time to calculate many eigenvalues and eigenvectors of a large matrix by Group S routines. See section 1.6 for sample data of memory and CPU time.

**Sample Main Program**
(Four eigenvalues and the ground-state eigenvector with precision check and evaluation of correlation functions)

ENTER SAMPLE PROGRAM NO.11 HERE

ENTER SAMPLE PROGRAM NO.11 HERE.

**call elm3(n, idim, ipair, bondwt, zrtio, ibond, elemnt, ideclr, list1, list2)**

**Function**

Return matrix elements in `elemnt` for a given Hamiltonian and spin configurations.

**Arguments**

See the items on `sz` and `lnc1` in section 2.1 for `n,idim,ipair,bondwt,zrtio,ibond, list1,list2`.

**elemnt(ideclr, idim)**: (output; real)

Matrix elements. The element at the *j*th row and *k*th column is `elemnt(j,k)`.

**ideclr**: (input; integer)

Adjustable dimension of `elemnt`.

(2)Eigenvalues by the Householder and Bisection Methods and Eigenvectors by the Inverse Iteration Method

**call diag(elemnt, ideclr, idim, E, v, ne, nvec, eps, wk, iwk)**

**Function**

Calculate an arbitrary number of eigenvalues and eigenvectors of a matrix, given all matrix elements.

**Arguments**

**elemnt(ideclr, idim)**: (input; real)

Matrix elements. Enter the values obtained in `elm3`. Entries of `elemnt` after execution of `diag` have meaningless values.

**ideclr**: (input; integer)

Adjustable dimension of `elemnt,vec,wk`.

**idim**: (input; integer)

Matrix dimension. Must be less than 2000.

**E(ne)**: (output; real)

Lower `ne` eigenvalues.

**v(ideclr, nvec)**: (output; real)

Components of eigenvectors. The components corresponding to `E(1)` are stored from `v(1,1)` to `v(idim,1)`, those to `E(2)` are from `v(1,2)` to `v(idim,2)`, and so on. Each value of components has the same meaning as in `lncv1`.

**ne**: (input; integer)

Number of eigenvalues to calculate. Must be between 1 and `idim`.

**nvec**: (input; integer)

Number of eigenvectors to calculate. Between 0 (no eigenvector) and `ne`.

**eps**: (input; real)

Allowed relative error.

**wk(ideclr, 8)**: (working area; real)

**iwk(ideclr)**: (working area; integer)

(3)Precision Check of an Eigenvalue and an Eigenvector

**call check3(elemnt, idim, ideclr, x, v, Hexpec)**

**Function**

Return the result, `v`, of multiplication of the Hamiltonian to a given eigenvector `x`. The expectation value of the Hamiltonian by `x` is also returned in `Hexpec`. Furthermore, `check2` prints the value of `Hexpec` and the ratios `v(j)/x(j)` from `j=min(idim/3,13)` to `idim` in step of `max(1,idim/20)`. This ratio `v(j)/x(j)` should be equal to the eigenvalue if `x` is the correct eigenvector. Thus the actual values of the ratio give the reliability of the eigenvector.

**Arguments**

Same as in `elm3` and `diag` except for `x,v,Hexpec`. See `check2` for `x,v,Hexpec`.

**Remark**

The user must call `elm3` just before calling `check3` since `diag` destroys the values in the array `elemnt`. See remarks on `check1` in section 2.1 for other comments.

## §3. User Errors

Processing is terminated at fatal errors, Exx. Messages with Wxx are warning-level errors. /S/ denotes the significance, and /A/ is the action to be taken by the user.

**E01**: Variable szval given to sz out of range

/S/ The value of `szval` given to the configuration-generation routine `sz` is out of allowed range.

/A/ Enter a value between 0 and `n`/2−1 to `szval`.

**E02**: Incorrect idim or n given to sz

/S/ The value of `idim` or `n` given to the routine `sz` does not match the values of `sz`.

/A/ Enter a value correctly calculated as explained in the item on `sz` in section 2.1.

**E03**: Incorrect data in ipair. Location : i j

/S/ The array `ipair` has an element either less than 1 or larger than `n`. The location of the wrong element is either `i` or `j`.

/A/ Check data given to `ipair`. In particular, check whether the number of data is correct (`ibond*2`).

**E04**: ndim given to bisec exceeds 2000

/S/ It has been found in `bisec` called by `diag` that the matrix dimension `idim` in `diag` exceeds 2000.

/A/ `diag` diagonalizes a matrix of dimension up to 2000. Specify a value within this limit, or use `lnc1` or `lnc2`.

**E05**: ne given to bisec out of range

/S/ It has been found in `bisec` called by `diag` that the number of eigenvalues `ne` given to `diag` is either less than 0 or larger than `idim`.

/A/ Specify `ne` between 0 and `idim`.

**E06**: Incorrect iv given to lnc1

/S/ The initial condition `iv` given to `lnc1` is either smaller than 1 or larger than `idim`.

/A/ Enter `iv` between 1 and `idim`.

**E07**: Tridiagonalization unsuccessful in lnc1. Beta(i) is too small at i=xx.

/S/ Tridiagonalization cannot be continued by the Lanczos iteration because the sub-diagonal element beta(i) at `xx` is nearly zero.

/A/ Try a different initial condition `iv`. Or, when the matrix dimension is smaller than about 100, use `diag`.

**E08**: Incorrect iv given to lnc2

/S/ The initial condition `iv` given to `lnc2` is either smaller than 1 or larger than `idim`.

/A/ Enter `iv` between 1 and `idim`.

**E09**: Tridiagonalization unsuccessful in lnc2. Beta(i) is too small at i=xx.

/S/ Tridiagonalization cannot be continued by the Lanczos iteration because the sub-diagonal element beta(i) at `xx` is nearly zero.

/A/ Try a different initial condition `iv`. Or, when the matrix dimension is smaller than about 100, use `diag`.

**E10**: nvec given to diag out of range

/S/ The number of vectors to calculate, `nvec`, given to `diag` is either smaller than 0 or larger than `ne`.

/A/ Enter `nvec` between 0 and `ne`.

**W01**: Wrong site number given to xcorr

/S/ A site number in pairs of sites given to `xcorr` is either smaller than 0 or larger than `n`, or a pair of sites have the same site number. Returns to the main program without processing.

/A/ Enter site numbers between 1 and `n` so that correct pairs are specified.

**W02**: Wrong site number given to zcorr

/S/ A site number in pairs of sites given to `zcorr` is either smaller than 0 or larger than `n`, or a pair of sites have the same site number. Returns to the main program without processing.

/A/ Enter site numbers between 1 and `n` so that correct pairs are specified.

**W03**: Number of vectors is less than 2 in orthg

/S/ The number of vectors `numvec` given to `orthg` is less than two.

/A/ Enter a value larger than or equal to two.

**W04**: Null vector given to orthg. Location is xx

/S/ The norm of the `xx`th vector given to `orthg` is too small (less than $10^{-20}$). Returns to the main program without orthogonalization.

/A/ Try orthogonalization excluding the null vector.

**W05**: Non-orthogonal vectors at xx yy. Overlap : zz. Unsuccessful orthogonalization.

/S/ Orthogonalization was unsuccessful by round-off errors in the sense that the resulting `xx`th and `yy`th vectors have significant overlap `zz`($\geq 10^{-10}$).

/A/ Accept round-off errors of `zz`, or try orthogonalization of smaller number of vectors.

**W06**: Wrong value given to nvec in lnc1. Only the eigenvalues are calculated.

/S/ The number of vectors `nvec` given to `lnc1` is either negative or larger than 4. Process is continued by assuming `nvec` =0.

/A/ Enter `nvec` between 0 and 4.

**W07**: lnc1 did not converge within 150 steps

/S/ The Lanczos iteration in `lnc1` did not converge within 150 steps to the level of relative precision of $10^{-13}$. Returns approximate eigenvalues.

/A/ Calculate the eigenvector using the returned approximate eigenvalue, and calculate a more reliable eigenvalue by `check1`. Or, if the matrix dimension is small (approximately, less than 100), use `diag`. Or, relax convergence criteria in the source code of `lnc1`: try longer steps than 150 (loop 100), or allow a larger relative error than $10^{-13}$ (just after calling `bisec`).

**W08**: nvec given to lncv1 out of range

/S/ The number of vectors `nvec` given to `lncv1` is either smaller than 1 or larger than 4. Process is returned to the main program without calculating eigenvectors.

/A/ Enter `nvec` between 0 and 4.

**W09**: Null vector given to check1

/S/ The eigenvector given to `check1` is almost null with a norm less than $10^{-30}$. Process is returned to the main program without checking eigenvectors.

/A/ Give a normalized vector. The output in `Hexpec` has a correct value only when the given eigenvector is normalized. The eigenvectors returned from `lncv1,lncv2,inv1, inv2,diag` are all normalized.

**W10**: Iterat in cg1 exceeds idim or 500. Approximate eigenvector returned. Iteration number in inv1 is xx.

/S/ The iteration number in the routine `cg1` called in `inv1` exceeded 500 or `idim` before reaching convergence. Returns an approximate eigenvector. The iteration number in `inv1` is `xx`.

/A/ Check precision of the eigenvector by `check1`. Or, try another initial vector `iv`, or use `diag` for a small matrix.

**W11**: inv1 did not converge

/S/ The inverse iteration did not reach convergence of relative precision $10^{-12}$ within 20 steps. An approximate eigenvector is returned.

/A/ Give a more accurate eigenvalue in `Eig`. Or, use `lncv1`.

**W12**: Wrong value given to nvec in lnc2. Only the eigenvalues are calculated.

/S/ The number of vectors `nvec` given to `lnc1` is either negative or larger than 4. Process is continued by assuming `nvec` =0.

/A/ Enter `nvec` between 0 and 4.

**W13**: lnc2 did not converge within 150 steps

/S/ The Lanczos iteration in `lnc2` did not converge within 150 steps to the level of relative precision of $10^{-13}$. Returns approximate eigenvalues.

/A/ Calculate the eigenvector using the returned approximate eigenvalue, and calculate a more reliable eigenvalue by `check2`. Or, if the matrix dimension is small (approximately, less than 100), use `diag`. Or, relax convergence criteria in the source code of `lnc2`: try longer steps than 150 (loop 100), or allow a larger relative error than $10^{-13}$ (just after calling `bisec`).

**W14**: nvec given to lncv2 out of range

/S/ The number of vectors `nvec` given to `lncv2` is either smaller than 1 or larger than 4. Process is returned to the main program without calculating eigenvectors.

/A/ Enter `nvec` between 0 and 4.

**W15**: Null vector given to check2

/S/ The eigenvector given to `check2` is almost null with a norm less than $10^{-30}$. Process is returned to the main program without checking eigenvectors.

/A/ Give a normalized vector. The output in `Hexpec` has a correct value only when the given eigenvector is normalized. The eigenvectors returned from `lncv1,lncv2,inv1, inv2,diag` are all normalized.

**W16**: itr in cg2 exceeds idim or 500. Approximate eigenvector returned. Iteration number in inv2 is xx.

/S/ The iteration number in the routine `cg2` called in `inv2` exceeded 500 or `idim` before reaching convergence. Returns an approximate eigenvector. The iteration number in `inv2` is `xx`.

/A/ Check precision of the eigenvector by `check2`. Or, try another initial vector `iv`, or use `diag` for a small matrix.

**W17**: inv2 iteration did not converge

/S/ The inverse iteration did not reach convergence of relative precision $10^{-12}$ within 20 steps. An approximate eigenvector is returned.

/A/ Give a more accurate eigenvalue in `Eig`. Or, use `lncv2`.

**W18**: Null vector given to check3

/S/ The eigenvector given to `check3` is almost null with a norm less than $10^{-30}$. Process is returned to the main program without checking eigenvectors.

/A/ Give a normalized vector. The output in `Hexpec` has a correct value only when the given eigenvector is normalized. The eigenvectors returned from `lncv1,lncv2,inv1, inv2,diag` are all normalized.

## §4. Remarks on Coding Techniques in TITPACK Ver. 2

The basic algorithms used in TITPACK Ver. 2, such as the Lanczos method and the inverse iteration method, have been well-known for many years. A short account is found in Appendix B; the reader is referred to references for details. I give in this section a few remarks on techniques used in TITPACK Ver. 2 to actually code these algorithms.

### 4.1 Bit Operations

An iteration algorithm, such as the Lanczos method, often includes a multiplication of the Hamiltonian matrix to a vector. When matrix elements are not explicitly given, as in Group L routines, the matrix-vector multiplication is coded by using bit-wise logical operations of two integers. I explain here these operations; the reader is referred to the source code of the routine `mltply` since typical examples are found there.

First, consider the diagonal element, the $S_z$-part of the Hamiltonian. Let us denote the diagonal element corresponding to `ipair(k*2-1)`≡i1 and `ipair(k*2)`≡i2 by $H_k$:

$$H_k = -2J_{\mathrm{i}_1\mathrm{i}_2}\Delta_{\mathrm{i}_1\mathrm{i}_2}S^z_{\mathrm{i}_1}S^z_{\mathrm{i}_2}$$

When $H_k$ operates on the `j`th component of vector `v`, the result is $-0.5J_{\mathrm{i1,i2}}\Delta_{\mathrm{i1,i2}}$`v(j)` if the spin state of i1 is equal to that of i2, and is sign-reversed if the spins are antiparallel. To determine whether the two spins are parallel or not, the following method is useful. As explained in the item on `sz` in section 2.1, `j` is the serial number of a spin state represented by the binary form of `list1(j)`. To see the spin states of the sites i1 and i2, one first forms an integer `"is"`$(=1\times 2^{\mathrm{isite1}}+1\times 2^{\mathrm{isite2}})$ where isite1≡i1−1 and isite2≡i2−1. The binary representation of `"is"` has 1 in the two bits corresponding to i1 and i2, and 0 in other bits. By executing `IAND(is,list1(j))`, one is able to cut out two relevant bits from `list1(j)`. If the two spins are parallel, the result of `IAND`, denoted `ibit`, is either 0 (both are down spins, $0\times 2^{\mathrm{isite1}}+0\times 2^{\mathrm{isite2}}$) or `"is"` (both up). If the two spins are antiparallel, the result assumes one of the other values, $1\times 2^{\mathrm{isite1}}$ or $1\times 2^{\mathrm{isite2}}$. Thus, one branches the processing depending upon whether or not `ibit` is equal to either `"is"` or 0.

A similar method applies to off-diagonal elements of the Hamiltonian. If the two relevant spins are parallel, the operation of an off-diagonal term onto `v` can be skipped due to the special character of the $S = 1/2$ Hamiltonian. If the two spins are found to be antiparallel by the method described above, the operation of an off-diagonal element on `v` is to exchange the spin states of the two relevant bits. Function `IEOR` (Exclusive Or) is used to exchange spin states: `IEOR(k,l)` compares corresponding bits of `k` and `l`. If the corresponding bits are different, the same bit in `ir=IEOR(k,l)` is 1. Otherwise, the bit is 0. Therefore, if we denote a pair of bits in `k` and `l` by (j1, j2), the corresponding bit in `ir` is the reverse of j2 when j1=1, and is j2 itself when j1=0. Hence, in order

to exchange bits corresponding to i1 and i2 in `list1(j)`, it is sufficient to operate `IEOR` upon is and `list1(j)`. To complete the matrix-vector multiplication, one checks up the serial number `m` of the resulting new (spin-exchanged) configuration and adds the vector component `v(m)` to the $j$th component of the vector `v0` which is the storage space of the result of multiplication. The serial number of a spin configuration is determined by the two-dimensional search method explained in the next section.

### 4.2 Two-Dimensional Search of Serial Number

It is sometimes necessary to know the serial number of a spin configuration given in the binary representation of an integer. A simple way is to prepare an array `list2'(k)` which returns the serial number of a given spin configuration `k`. TITPACK Ver.1 used this method. However, for a large spin number `n`, the maximum integer `k` representing spin configurations increases exponentially, $2^{\mathrm{n}} - 1$, and accordingly the array `list2'` occupies a very large memory space.

It is also true that this simple method uses unnecessarily large memory: Not all of $2^{\mathrm{n}} - 1$ spin configurations are required to diagonalize a matrix in a space with a fixed $S_z^{\mathrm{total}}$. For instance, when `n`=20 and $S_z^{\mathrm{total}} = 0$, $2^{\mathrm{n}} - 1$ is equal to 1,048,575, while the number of actual spin configurations is the number of combinations to choose 10 up-spins out of 20 sites, ${}_{20}\mathrm{C}_{10} = 184,756$. The difference 1,048,575−184,756=863,819 is idling in `list2'`. Therefore it is desirable to release this unused area and, at the same time, to lower the limit of maximum number of the argument of `list2'(k)` from $2^{\mathrm{n}} - 1$ to a smaller number.

Masao Ogata (private communication) and H.Q. Lin (Ref. 1) solved this problem by splitting the binary representation of an integer (corresponding to a spin configuration) into two parts. I explain their idea in terms of a simple example. Let the spin number be `n`=6 and $S_z^{\mathrm{total}}$=`szval=0.0d0`. The number of up-spins is equal to that of down-spins, both 3. One gives a serial number to a spin configuration `i` in increasing order, and splits the binary representation of `i` into upper (`ia`) and lower (`ib`) parts. The table below shows the first six configurations.

| Serial No. | Configuration i(binary) | ib | ia | jb | ja | ja+jb |
|---|---|---|---|---|---|---|
| 1 | 000111 | 000 | 111 | 0 | 1 | 1 |
| 2 | 001011 | 001 | 011 | 1 | 1 | 2 |
| 3 | 001101 | 001 | 101 | 1 | 2 | 3 |
| 4 | 001110 | 001 | 110 | 1 | 3 | 4 |
| 5 | 010011 | 010 | 011 | 4 | 1 | 5 |
| 6 | 010101 | 010 | 101 | 4 | 2 | 6 |

One then gives serial numbers (`ja` and `jb`) to the upper (`ia`) and lower (`ib`) parts of spin configurations in the following way. Concerning the correspondence between `ia` and `ja`, `ja` starts from 1 and is increased by 1 as `ia` increases as long as `jb` is unchanged (and hence `ib` is unchanged). When `ib` is updated and so is `jb`, `ja` starts from 1 once again. This allocates a unique `ja` to a given `ia`. (It sometimes happens that different `ia`'s correspond to the same `ja`, which is not a disturbing factor.) Next, `jb` is first cleared to 0. When `ib` is renewed, the new value of `jb` is set to the value of `ja+jb` corresponding to the configuration just one configuration before. In this way, the serial numbers `ja` and `jb` of the upper and lower bits of a spin configuration are determined. The sum `ja+jb` gives the final serial number of the configuration.

One should prepare two arrays to give the serial numbers `ja` and `jb` of lower and upper bits, respectively, of spin configurations. If one treats spin configurations without splitting into two parts, one must declare an array `list2`$'$ of size $2^6 - 1$. When the splitting method is used, two arrays are needed, each of which is of size $2^3 - 1$. The difference between two methods is not necessarily large for `n`=6, but when `n` is as large as 26, one requires about $2^{26}$ of integer memory space while the other needs only $2 \times 2^{13}$. The former far exceeds the currently available memory capacity, whereas the latter poses no problem. In TITPACK Ver. 2, I declare two arrays as `list2(2,0:2**15)` assuming that the maximum number of spins treated is `n`=31 reflecting the fact that most computers express integers by 32 bits. To split an integer, `iexchg`, into upper and lower bits, one first prepares an integer `irght` which has 1's in the lower bits and 0's in all other bits and then operates `IAND` onto `iexchg` and `irght` to extract only lower bits. As for the upper ibits, one similarly prepares `ilft` with all upper bits being 1 and 0 otherwise. Then one takes `IAND` and divides the result by `ihfbit` which has 1 only at the mid-most bit, leading to a binary representation of upper bits shifted to the lower bits.

### 4.3 Lanczos Method with Two Vectors

It naively seems necessary to store three vectors to execute the Lanczos method. These are the current vector $\mathbf{v}_1$, the previous vector $\mathbf{v}_0$, and another vector $\mathbf{v}_2$ which is the result of matrix multiplication onto $\mathbf{v}_1$ (Appendix B). Misao Matsushita (private communication) pointed out that only two vectors are actually required since $\mathbf{v}_2$ appears only as a linear combination with $\mathbf{v}_0$. The difference between two and three vectors is significant when two vectors can be stored in main memory while three cannot be.

In the routine `mltply` of TITPACK Ver. 2, the input vector $\mathrm{v}_0$ corresponds to $-\beta_{i-1}\mathbf{v}_{i-1}$ in eq. (3) of Appendix B. The output $\mathrm{v}_0$ of `mltply` is $H\mathbf{v}_i - \beta_{i-1}\mathbf{v}_{i-1}$. `mltply` at the same time calculates $\langle \mathbf{v}_i H \mathbf{v}_i \rangle = \alpha_i$.

## §5. Quick Reference to Use Sample Programs

Samples of main programs are found in Appendix C which can be immediately used after changing several parameters. I hereby explain where to rewrite in these samples to adjust them to user's own problems. It is strongly recommended to read related articles in section 2.

1. Determine the parameters $J_{ij}, \Delta_{ij}$ and the lattice structure represented by $\langle i, j\rangle$ in the Hamiltonian
$$H = -2 \sum_{<i,j>} J_{ij}(S_i^x S_j^x + S_i^y S_j^y + \Delta_{ij} S_i^z S_j^z). \tag{1}$$
Site numbers should cover all integers from 1 to `n`. Also determine $S_z^{\rm total}$ in which to diagonalize the Hamiltonian. $S_z^{\rm total}$ must be non-negative.

2. Determine the matrix dimension `idim` which is calculated from `n` and $S_z^{\rm total}$. For a given value of $S_z^{\rm total}$, the number of up-spins, iu, and the number of down spins, id, satisfy the relations iu+id=`n`, (iu−id)/2=$S_z^{\rm total}$. Then one obtains iu=`n`/2−$S_z^{\rm total}$, and the matrix dimension is calculated as the number of combinations to choose iu out of `n`, `idim`=${}_{\rm n}{\rm C}_{\rm iu}$.

3. Use Group S routines if `idim` is smaller than 100. See sample 11. The user rewrites the following points.
   a. `n,idim,ibond` (number of bonds) in the `parameter` statement.
   b. `npair(2),sxx(1),szz(1)` in the `dimension` statement. If you wish to calculate correlation functions for $k$ pairs of sites, change 2 to `2*k` and 1 to `k`. Correspondingly, replace `npair(1)=1,npair(2)=2` above `call xcorr` by appropriate spin pairs. For example, if you wish to calculate correlations for a couple of pairs (2, 5) and (4, 7), you write `npair(1)=2,npair(2)=5, npair(3)=4, npair(4)=7`. It is further necessary to replace 1 in the arguments of `xcorr` and `zcorr` by `k`.
   c. Three data statements. Enter `ibond` values of $J_{ij}$ in `bondwt`. Sample 11 is the case with all $J$'s $-1$ (antiferromagnetic interactions). Enter `ibond` values of $\Delta_{ij}$ in `zrtio`. Sample 11 corresponds to the isotropic Heisenberg model which has all $\Delta$'s equal to 1. The lattice structure is specified by `ipair`. `ibond` pairs of sites in `ipair` describe which lattice site is connected to which. Sample 11 is an example of a one-dimensional model with a periodic boundary.
   d. Enter the value of $S_z^{\rm total}$ at the third argument of `sz` which is `0.0d0` in sample 11.
   e. Replace the value of `ne` (number of eigenvalues) and `nvec` (number of eigenvectors) above `call diag` with desired values. `ne` must not exceed `idim` in Group S. `nvec` should be between 0 and `ne`. If `nvec` is larger than 1, it is also necessary to rewrite `v(idim)` in the `dimension` statement as `v(idim,`value of `nvec)`. When precision check and evaluation of correlation function are performed, one should replace the

argument `v` in `check3,xcorr,zcorr` by `v(1,`vector no.`)`. The number of vector starts from 1 (ground state) and increases with increasing eigenvalues.

4. When `idim` exceeds 100, determine which of Groups L or M to use. Group M routines run faster than those in Group L. If the following memory requirement of Group M is satisfied on your computer,

$$\text{idim} \times \text{ibond(number of bonds)} \times 1.5 \times 8/1024^2 \quad \text{(MB)}$$

   it is better to use Group M.

5. To calculate eigenvalues and the ground-state eigenvector by Group L:

   Use sample 1 or sample 2. If it is necessary to obtain eigenvectors with very high precision (typically, more than 10 significant figures), use of sample 2 is recommended. For most purposes sample 1 is sufficient. Each of these sample programs should be rewritten following the same remarks as in item 3 above.

6. To calculate eigenvalues and `nvec` eigenvectors by Group L:

   Use sample 3. Rewrite the value of `nvec` in the `parameter` statement in addition to the points in item 3 above. Sample 3 executes precision check of the last vector (no. `nvec`). As mentioned in the item on `lncv1` and `inv1` in section 2.1, the precision of higher-level eigenvectors obtained by `lncv1` is not very high.

7. To calculate a higher-level eigenvector with sufficient reliability:

   Use sample 4. In addition to the points in the above item 3, one should replace the argument `E(3)` of `inv1` with the desired energy level.

8. To check degeneracy of an eigenvalue:

   Follow sample 5 to call `orthg`. In sample 5, one calculates two eigenvectors starting from different initial conditions and store these in `v`. The eigenvectors in `v` are passed to `orthg`, and the output in `idgn` indicates the degeneracy. The user has to rewrite the points in item 3 above as well as to replace 2 in `v(idim,2)` and `norm(2)` by the expected maximum number of degeneracy (denoted here `idg`). Furthermore, `idim/2` in loop 10 must be replaced by `idim/idg` and the last argument of `orthg` is changed to `idg`.

9. To use Group M routines:

   Refer to samples 6 to 10. Sample 1 for Group L corresponds to sample 6 for M, and similarly, sample j to sample (j+5). Rewrite the same points as in samples 1 to 5. The only difference is that, in sample 10, the maximum possible number of degeneracy is assumed to be 3 in contrast to 2 in sample 5. Thus 2 in item 8 should be read as 3 to use sample 10.

## §6. Distribution and Other Remarks

The source code of TITPACK Ver. 2 is available at

`http://www.stat.phys.titech.ac.jp/~nishi/`

The copyright of TITPACK Ver. 2 belongs to the present author. There is no restriction on personally copying and using it for academic purposes. I only ask the user to acknowledge the use of TITPACK Ver. 2 with the present author's name explicitly indicated when publishing results in an oral or a written form. Since TITPACK Ver. 2 is not a commercial software, I do not assume responsibility for for losses or damages resulting from bugs or other reasons.

## References


1. H.Q. Lin, Phys. Rev. B**42** (1990) 6561.
2. M. Mori, *Programming For Numerical Calculations By FORTRAN77* (in Japanese) (Iwanami Shoten) 1988.
3. H. Togawa, *Numerical Calculations Of Matrices* (in Japanese) (Ohm Sha) 1971.
   B.N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice Hall) 1980.
   J.H. Wilkinson and C. Reinsch, *Linear Algebra* (Springer) 1971.
4. E.R. Gagliano, E. Dagotto, A. Moreo and F.C. Alcaraz, Phys. Rev. B**34** (1986) 1677.

## Appendix A. Internal Routines

Functions and arguments of internally-called routines are explained here. Although these routines are not explicitly referred to by the user, it may be helpful to give a short account for those who rewrite the source code to further optimize the program for their own problems. **All real numbers in TITPACK Ver. 2 are of 8-byte length. The number of spins n must be less than 32 in TITPACK Ver. 2.**

(1)Execution of the Lanczos Process in Group L (Eigenvalues)

**call lnc1z(n, idim, ipair, bondwt, zrtio, ibond,**
**nvec, iv, E, itr, v1, v0, list1, list2)**

**Function**

Routine `lnc1` only checks input parameters and passes the actual processing to `lnc1z`. The reason to code in this way is to decrease the number of arguments for working area which the user has to indicate explicitly (Ref. 2); `lnc1z` has two arguments `v0,v1` for working area while `lnc1` has only one `wk`. In the present case, the total number of arguments the user has to write remains unchanged because there appears additional argument `ideclr` to specify the adjustable dimension of working area `wk` in `lnc1`. However, in many other routines such as `inv1`, the number of arguments the user has to write actually decreases by this technique, which would lead to less trivial bugs. Thus, to be consistent with other routines, I have used this method also in `lnc1/lnc1z`.

**Arguments**

Same as in `lnc1` except that arguments `v1` and `v0` replace `wk` and `ideclr` in `lnc1`. Both `v1(idim)` and `v0(idim)` are real.

(2)Execution of the Lanczos Process in Group L (Eigenvectors)

**call lncv1z(n, idim, ipair, bondwt, zrtio, ibond,**
**nvec, iv, x, ideclr, itr, v1, v0, list1, list2)**

**Function**

Execute the calculation of eigenvectors by receiving arguments from `lncv1`.

**Arguments**

Same as in `lncv1` except that arguments `v1` and `v0` replace `wk` and `ideclr` in `lncv1`. Both `v1(idim)` and `v0(idim)` are real.

(3)Matrix-Vector Multiplication Used in Lanczos Iteration in Group L

**call mltply(n, idim, ipair, bondwt, zrtio, ibond, v1, v0, prdct, list1, list2)**

**Function**

Return $H\times$`v1`+`v0` for input vectors `v1` and `v0` as well as the expectation value `Hexpec` of the Hamiltonian by `v1`.

**Arguments**

Same as in `lnc1` except for `v0,v1,prdct`.

**v1**: (input; real)

Normalized input data vector.

**v0**: (input/output; real)

Input: data vector. Output: $H \times$`v1`+`v0`.

**prdct**: (output; real)

Expectation value of the Hamiltonian by `v1`.

(4)Execution of Inverse Iteration in Group L

---

**call inv1z(n, idim, ipair, bondwt, zrtio, ibond,**
**Eig, iv, x, b, p, r, y, list1, list2)**

---

**Function**

Execute inverse iteration. Actual execution is performed in this routine to decrease the number of arguments the user writes in `inv1`, similarly to `lnc1z`.

**Arguments**

Same as in `inv1` except that `b,p,r,y` replace `wk,ideclr` in `inv1`.

**b(idim), p(idim), r(idim), y(idim)**: (working area; real)

(5)Solution of Linear Equation System by the Conjugate Gradient Method

---

**call cg1(n, idim, ipair, bondwt, zrtio, ibond,**
**Eig, x, b, p, r, y, itr, list1, list2)**

---

**Function**

Calculate the improved approximate eigenvector $\psi_i$ in the inverse iteration process $(H - \mathrm{Eig})^{-1}\psi_{i-1} = \psi_i$ by solving the system of linear equations $(H - \mathrm{Eig})\psi_i = \psi_{i-1}$. The Conjugate Gradient method is used to solve linear equations.

**Arguments**

Same as in `inv1` except that `iv,x,wk,ideclr` in `inv1` are replaced by `x,b,p,r,y,itr`.

**x(idim)**: (output; real)

New approximate eigenvector. $\psi_i$ in the above explanation of function.

**b(idim), p(idim), r(idim), y(idim)**: (working area; real)

**itr**: (output; integer)

Number of steps executed for convergence. Given in a multiple of 5. The convergence condition is that the norm of residual in the Conjugate Gradient method is smaller than $1.0 \times 10^{-9}$ of the norm of the right hand side $(\psi_{i-1})$ of linear equations.

**Remark**

The maximum number of iteration is the smaller one of 500 and `idim`. Empirically, no convergence is expected beyond this limit. If convergence is not reached within this limit, it is better to try a different initial condition `iv` or to use a different routine.

(6)Execution of Diagonalization in Group M

The relation between execution routines `lnc2z,lncv2z,inv2z,cg2` in Group M and user routines `lnc2,lncv2,inv2` is the same as the relation of routines in Group L (such as `lnc1z` vs. `lnc1`). Only usage is shown below.

**call lnc2z(elemnt, loc, idim, ideclr, ic, nvec, iv, E, itr, v1, v0)**

Both `v1(idim),v0(idim)` are real.

**call lncv2z(elemnt, loc, idim, ideclr, ic, nvec, iv, x, itr, v1, v0)**

Both `v1(idim),v0(idim)` are real.

**call inv2z(elemnt, loc, idim, ideclr, ic, iv, Eig, x, b, p, r, y)**

`b(idim),p(idim),r(idim),r(idim)` are all real.

**call cg2(elemnt, loc, idim, ideclr, ic, Eig, x, b, p, r, y, itr)**

See the item on `cg1` for `x,b,p,r,y`. Other arguments have the same meaning as in `inv2z`.

(7)Tridiagonalization by the Householder Method in Group S

**call hshldr(elemnt, ideclr, idim, alpha, beta, c, w, p, q)**

**Function**

Tridiagonalize a matrix by the Householder method, given all matrix elements. The main part of coding is based on Ref. 2.

**Arguments**

Arguments `elemnt,ideclr,idim` have the same meaning as in `diag`.

**alpha(idim)**: (output; real)

Diagonal elements.

**beta(idim)**: (output; real)

Subdiagonal elements.

**c(idim)**: (output; real)

Normalization factor used in the calculation of eigenvectors.

**w(idim), p(idim), q(idim)**: (working area; real)

(8)Eigenvector of Tridiagonal Matrix in Original Basis in Group S

**call vec3(E, elemnt, ideclr, idim, ne, nvec,**
**di, bl, bu, bv, cm, lex, alpha, beta, c, v)**

**Function**

Calculate the eigenvector of a given tridiagonal matrix by the inverse iteration method, and transform the result to the representation by the original (pre-tridiagonalized) basis. Each step of inverse iteration is executed by solving a system of linear equations based on LU decomposition of a matrix. The main part of coding is based on Ref. 2.

**Arguments**

`elemnt,ideclr,idim` have the same meaning as in `diag`.

**E(ne)**: (input; real)

Eigenvalues. Enter those obtained in `bisec`.

**ne**: (input; integer)

Number of input eigenvalues. Must not exceed `idim`.

**nvec**: (input; integer)

Number of eigenvectors to calculate. Between 1 and `ne`.

**di(idim), bl(idim), bu(idim), bv(idim), cm(idim)**: (working area; real)

**lex(idim)**: (working area; integer)

**alpha(idim)**: (input; real)

Diagonal elements of tridiagonal matrix.

**beta(idim)**: (input; real)

Subdiagonal elements.

**c(idim)**: (input; real)

Normalization factor to transform to the original representation. Enter the data calculated in `hshldr`.

**v(ideclr, nvec)**: (output; real)

Eigenvectors. The eigenvector corresponding to `E(1)` is given in `v(1,1)`,$\cdots$,`v(idim,1)`, that to `E(2)` is in `v(1,2)`,$\cdots$,`v(idim,2)`, and so on. Represented in the original basis generated in `sz`.

(9)Error Check of Site Data

**call datack(ipair, ibond, n)**

**Function**

Check if the data in `ipair` include numbers out of the range between 1 and `n`. If found, terminate execution issuing an error message.

**Arguments**

All arguments have the same meaning as in `lnc1`.

(10)Eigenvalues of Tridiagonal Matrix by the Bisection Method

**call bisec(alpha, beta, ndim, E, ne, eps)**

**Function**

Calculate lower `ne` eigenvalues by the bisection method, given a tridiagonal matrix. The main part of coding is based on Ref. 2.

**Arguments**

**alpha(ndim)**: (input; real)

Diagonal elements of a tridiagonal matrix.

**beta(ndim)**: (input; real)

Subdiagonal elements. The last entry (`beta(ndim)`) may have an arbitrary value.

**ndim**: (input; integer)

Size of matrix.

**E(ne)**: (output; real)

Lowest `ne` eigenvalues.

**eps**: (input; real)

Allowed relative error.

(11)Eigenvector of Tridiagonal Matrix in Tridiagonal Basis in Groups L and M

**call vec12(E, ndim, nvec, di, bl, bu, bv, cm, lex)**

**Function**

Calculate the eigenvector of a tridiagonal matrix by the inverse iteration method. Each step of inverse iteration is executed by solving a system of linear equations based on LU decomposition. In contrast to `vec3`, no transformation to the original basis is carried out. The result is written in internal variables defined in `common block/vecdat/` and is used later by `lncv1,lncv2` to calculate the eigenvector in the original representation. Elements of the tridiagonal matrix are passed from `lnc1,lnc2` through `common block/vecdat/`. The main part of coding is based on Ref. 2.

**Arguments**

`nvec,di,bl,bu,bv,cm,lex` have the same meaning as in `vec3`.

**E(4)**: (input; real)

Four lowest eigenvalues obtained in `bisec`.

**ndim**: (input; integer)

Matrix size.

## Appendix B. Lanczos, Inverse Iteration, and Conjugate Gradient Methods

Routines in groups L and M use the Lanczos method to tridiagonalize a matrix and the inverse iteration method combined with the Conjugate Gradient method to calculate eigenvectors. In this Appendix, I give a short account of these methods. The reader is referred to Ref. 3 and other textbooks of numerical techniques for details.

### B.1 Lanczos Method

It is instructive first to explain the Lanczos method as an improvement of the power method. The power method is a simple way to calculate the eigenvalue of a matrix with the largest absolute value; one starts from an arbitrary initial vector $\mathbf{v}_0$ and multiplies the matrix $H$ repeatedly until the resulting vector converges to the desired eigenvector. More precisely, when the eigenvalues and eigenvectors of an $m$-dimensional matrix $H$ are $E_j, \psi_j (j = 1, \cdots, m)$, the expansion of the initial vector reads

$$\mathbf{v}_0 = \sum_{j=1}^{m} a_j \psi_j$$

Multiplying this initial vector $k$ times by $H$, one obtains

$$\mathbf{v}_k \equiv H^k \mathbf{v}_0 = \sum_{j=1}^{m} a_j E_j^k \psi_j$$

It is apparent that the relative weight of the eigenvector corresponding to the eigenvalue with the largest absolute value increases exponentially with $k$ among terms appearing in the above sum.

Acceleration of convergence over the simple power method is achieved by subtracting components of previous vectors $(\mathbf{v}_{k-1}, \mathbf{v}_{k-2}, \cdots)$ from $\mathbf{v}_k$ so that one can eliminate the effects of the arbitrarily chosen initial vector as rapidly as possible. This subtraction of components of previous vectors is incidentally equivalent to tridiagonalization. Let us explain this point.

If the tridiagonal matrix $T$ is obtained from the original matrix $H$ by a transformation matrix $V$, one has the relation $T = V^{-1}HV$, or $VT = HV$. Let the column vectors of $V$ be $\mathbf{v}_1, \mathbf{v}_2, \cdots$, and the diagonal elements of $T$ be $\alpha_1, \alpha_2, \cdots$, and the subdiagonal elements $\beta_1, \beta_2, \cdots$. Then the relation $VT = HV$ is written as

$$\begin{aligned}
H\mathbf{v}_1 &= \alpha_1 \mathbf{v}_1 + \beta_1 \mathbf{v}_2 \\
H\mathbf{v}_2 &= \beta_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \beta_2 \mathbf{v}_3 \\
H\mathbf{v}_3 &= \beta_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \beta_3 \mathbf{v}_4 \\
&\cdots \\
H\mathbf{v}_{m-1} &= \beta_{m-2} \mathbf{v}_{m-2} + \alpha_{m-1} \mathbf{v}_{m-1} + \beta_{m-1} \mathbf{v}_m \\
H\mathbf{v}_m &= \beta_{m-1} \mathbf{v}_{m-1} + \alpha_m \mathbf{v}_m
\end{aligned} \tag{1}$$

If one rewrites (1) into a form to calculate $\mathbf{v}_k$ successively,

$$\begin{aligned}
\mathbf{v}_2 &= (H\mathbf{v}_1 - \alpha_1\mathbf{v}_1)/\beta_1 \\
\mathbf{v}_3 &= (H\mathbf{v}_2 - \beta_1\mathbf{v}_1 - \alpha_2\mathbf{v}_2)/\beta_2 \\
\mathbf{v}_4 &= (H\mathbf{v}_3 - \beta_2\mathbf{v}_2 - \alpha_3\mathbf{v}_3)/\beta_3 \\
&\cdots \\
\mathbf{v}_m &= (H\mathbf{v}_{m-1} - \beta_{m-2}\mathbf{v}_{m-2} - \alpha_{m-1}\mathbf{v}_{m-1})/\beta_{m-1}
\end{aligned} \tag{2}$$

The last equation in (2) corresponds the second last of (1). The requirement that the last relation of (1) is compatible with that of (2) leads to the vanishing value of $\mathbf{v}_{m+1}$ when (2) is formally extended to $m+1$. In order to have a vanishing $(m+1)$th vector in the series of $\mathbf{v}_k$, it is sufficient to choose $\mathbf{v}_k$ so that it is orthogonal to all previous vectors $\mathbf{v}_{k-1}, \mathbf{v}_{k-2}, \cdots$, since $(m+1)$ vectors cannot be orthogonal to each other in the $m$-dimensional space. It is useful here to regard (2) as an iterative orthogonalization process by subtracting components of previous vectors. Actually, it turns out that (Ref. 3) by choosing

$$\begin{aligned}
\alpha_i &= \mathbf{v}_i^T H\mathbf{v}_i \\
\beta_i &= \parallel H\mathbf{v}_i - \beta_{i-1}\mathbf{v}_{i-1} - \alpha_i\mathbf{v}_i \parallel
\end{aligned} \tag{3}$$

all $\mathbf{v}_k$ are orthogonal to each other. Since this process (2) and (3) can be regarded as an improvement of the power method, it is not necessary to execute all of $(m-1)$ steps in (2) if one wishes to evaluate only several low-lying eigenvalues; one may calculate the eigenvalues of the intermediate tridiagonal matrix of dimension $l(< m)$ by the bisection method to see whether or not the low-lying eigenvalues have reached sufficiently converged values.

As for the eigenvectors, one first calculates the eigenvectors of the tridiagonal matrix when convergence of eigenvalues is confirmed. One then transforms the eigenvectors of the tridiagonal matrix into the original representation by use of the transformation matrix $V$. That is, the eigenvectors in the original representation are obtained by summing up the products of the components $c_1, c_2, \cdots$ of the eigenvector in the tridiagonal representation and the column vectors $\mathbf{v}_1, \mathbf{v}_2, \cdots$ of $V$: $c_1\mathbf{v}_1 + c_2\mathbf{v}_2 \cdots$. However, it is difficult to store the sequence generated by (2) $\mathbf{v}_1, \mathbf{v}_2, \cdots$ until convergence since convergence usually results after tens of iterations (and tens of the vectors should be stored in memory). A simple way out is to store $c_1, c_2, \cdots$ (real numbers) in the first run, and repeat the Lanczos process to generate $\mathbf{v}_1, \mathbf{v}_2, \cdots$ and sum up the products of the vectors and $c_1, c_2, \cdots$ (Ref. 4). This method is used in routines `lncv1,lncv2`.

### B.2 Inverse Iteration and Conjugate Gradient Methods

Inverse iteration is a method to calculate the eigenvector corresponding to a given approximate eigenvalue $E_a$. One starts from an arbitrary initial vector $\mathbf{v}_0$ and repeatedly multiplies $(H - E_a)^{-1}$. Let the expansion of the initial vector by the eigenvectors of $H$ be

$$\mathbf{v}_0 = \sum_{j=1}^{m} a_j \psi_j$$

Then, $k$ multiplications of $(H - E_a)^{-1}$ lead to

$$\mathbf{v}_k \equiv (H - E_a)^{-k} \mathbf{v}_0 = \sum_{j=1}^{m} a_j (E_j - E_a)^{-k} \psi_j$$

This equation indicates that the contribution of the eigenvector corresponding to the eigenvalue closest to $E_a$ becomes exponentially large as $k$ increases. Convergence is faster for a better approximate eigenvalue; if one gives an eigenvalue obtained in `lnc1,lnc2` to `inv1,inv2`, the iteration usually converges after a few steps.

If one tries to directly execute multiplication of $(H - E_a)^{-1}$, one has to obtain the inverse matrix, which is in general a difficult task. However, the relation

$$\mathbf{v}_k = (H - E_a)^{-1} \mathbf{v}_{k-1}$$

is equivalent to

$$(H - E_a)\mathbf{v}_k = \mathbf{v}_{k-1}$$

which may be regarded as a system of linear equations to calculate $\mathbf{v}_k$, given $\mathbf{v}_{k-1}$. Thus standard techniques can be used.

The Conjugate Gradient method solves linear equations of large scale by regarding the equation $M\mathbf{x} - \mathbf{b} = 0$ as the extremization of a bilinear form $f(\mathbf{x}') \equiv (\mathbf{r}, M^{-1}\mathbf{r})$ of the residual $\mathbf{r} \equiv \mathbf{b} - M\mathbf{x}'$, where $\mathbf{x}'$ is an approximate solution. Here $(\ ,\ )$ denotes inner product. To extremize $f(\mathbf{x}')$, one iteratively improves the approximate solution $\mathbf{x}'$ by changing $\mathbf{x}'$ to the direction with the steepest gradient on the landscape of $f(\mathbf{x}')$. The point where $f(\mathbf{x}')$ is extremum along this direction may be chosen as the next improved approximation. This is the Steepest Decent method. The Conjugate Gradient method is its improvement in that the direction along which a new approximation is sought is chosen within a space orthogonal to all previous modification vectors.

## Appendix C. Sample Programs

Here are listed samples of main programs to be used in various situations. The model Hamiltonian in these samples is the `n`=16 Heisenberg antiferromagnet with nearest neighbor interactions in one dimension unless mentioned otherwise. $S_z^{\rm total}$ is 0.

**Sample 1**: Eigenvalues by `lnc1` and the ground-state eigenvector by `lncv1`. Precision check by `check1`, and two-point correlation functions by `xcorr,zcorr`.

**Sample 2**: Eigenvalues by `lnc1` and the ground-state eigenvector by `inv1`. Precision check by `check1`, and two-point correlation functions by `xcorr,zcorr`

**Sample 3**: Three lowest eigenvectors by `lncv1` and precision check of the third one.

**Sample 4**: The third lowest eigenvector by `inv1` and its precision check.

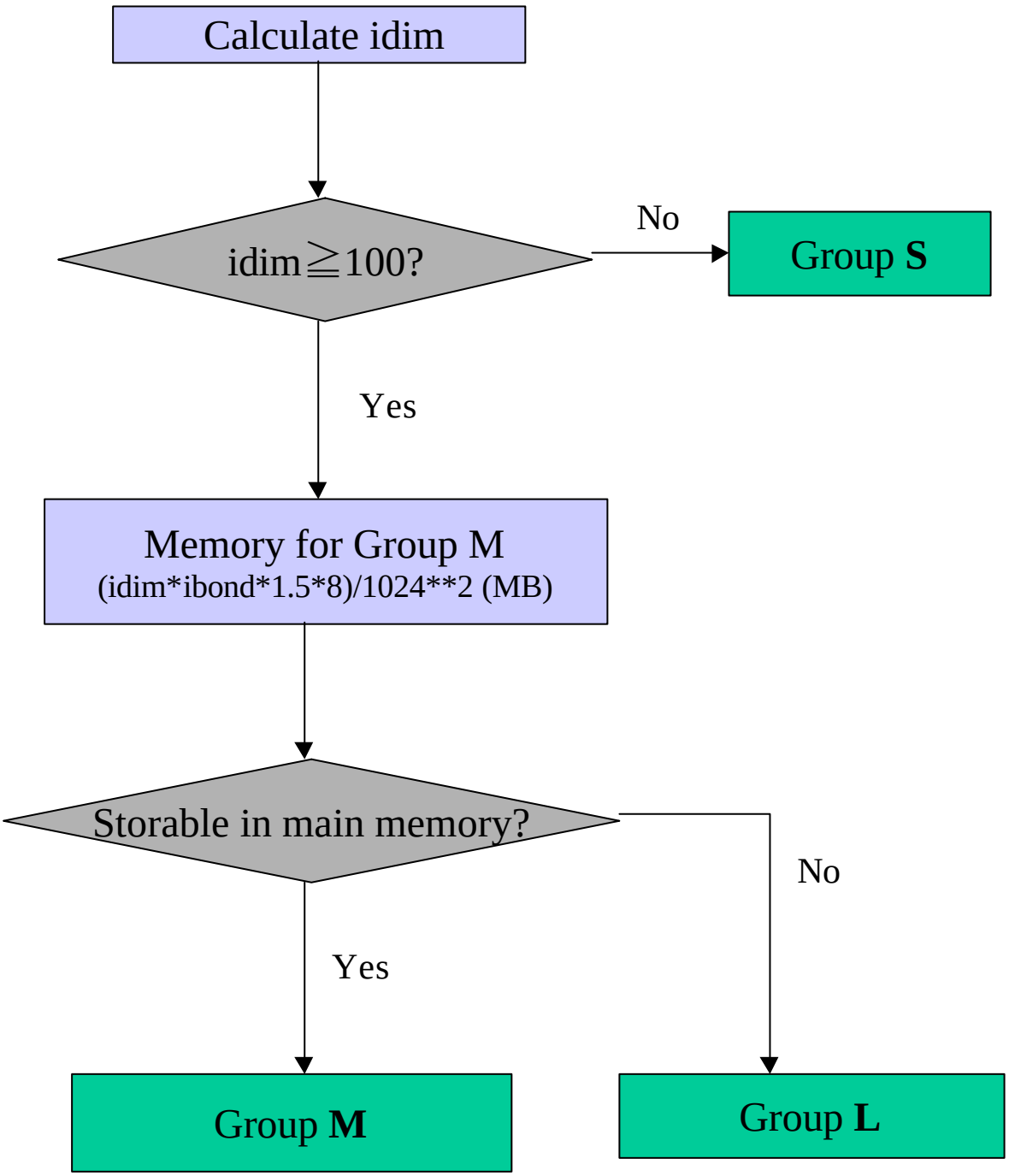
**Sample 5**: Degeneracy check of the lowest level by starting from two different initial conditions.

**Samples 6 – 10**: Same processing as in samples 1 – 5 using Group M routines. The only difference is that degeneracy check is for `n`=15.

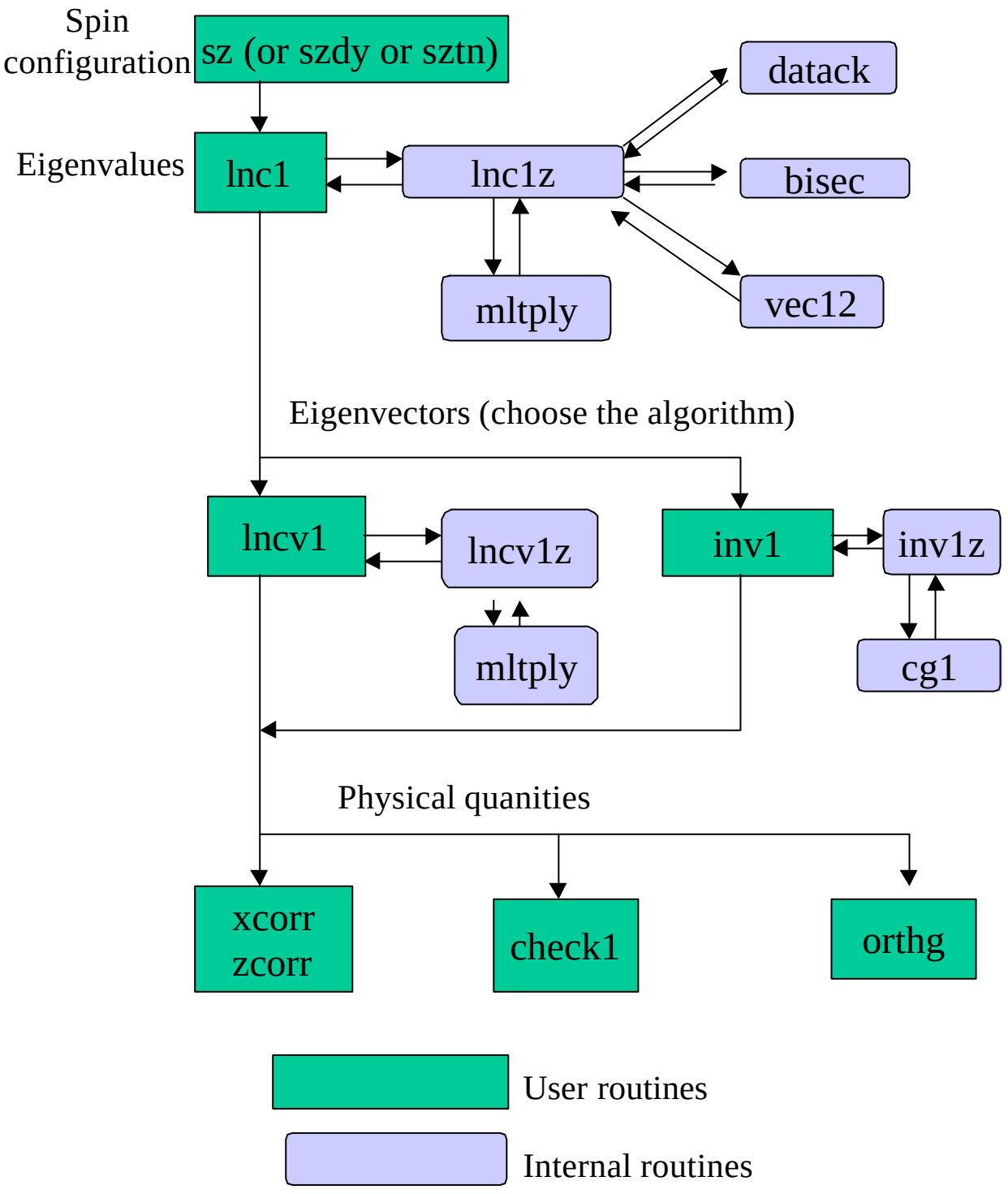
**Sample 11**: The same processing for `n`=8 as in sample 1 using `diag`.

# Source List

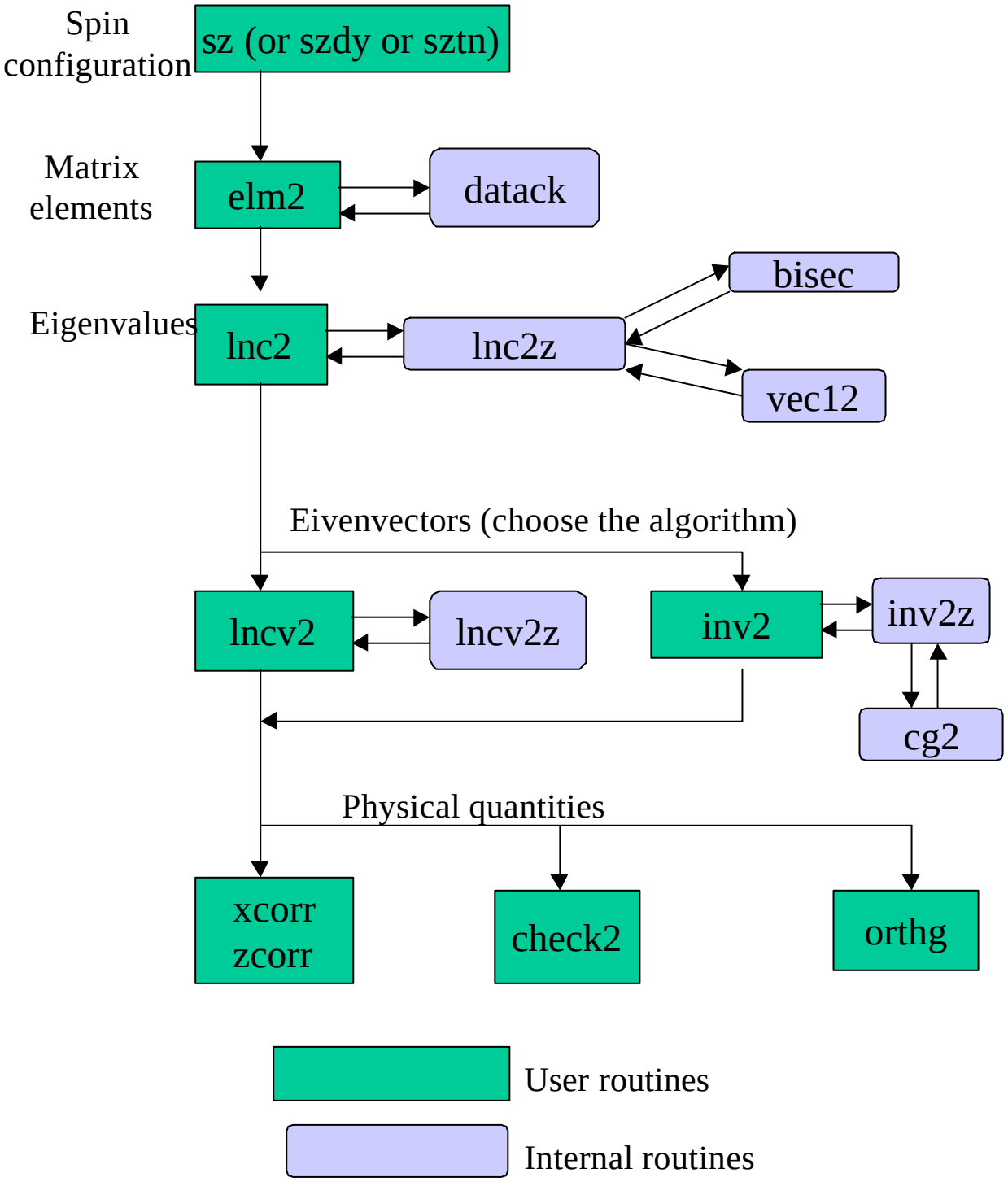
## 1.2 Choosing Subroutine Groups



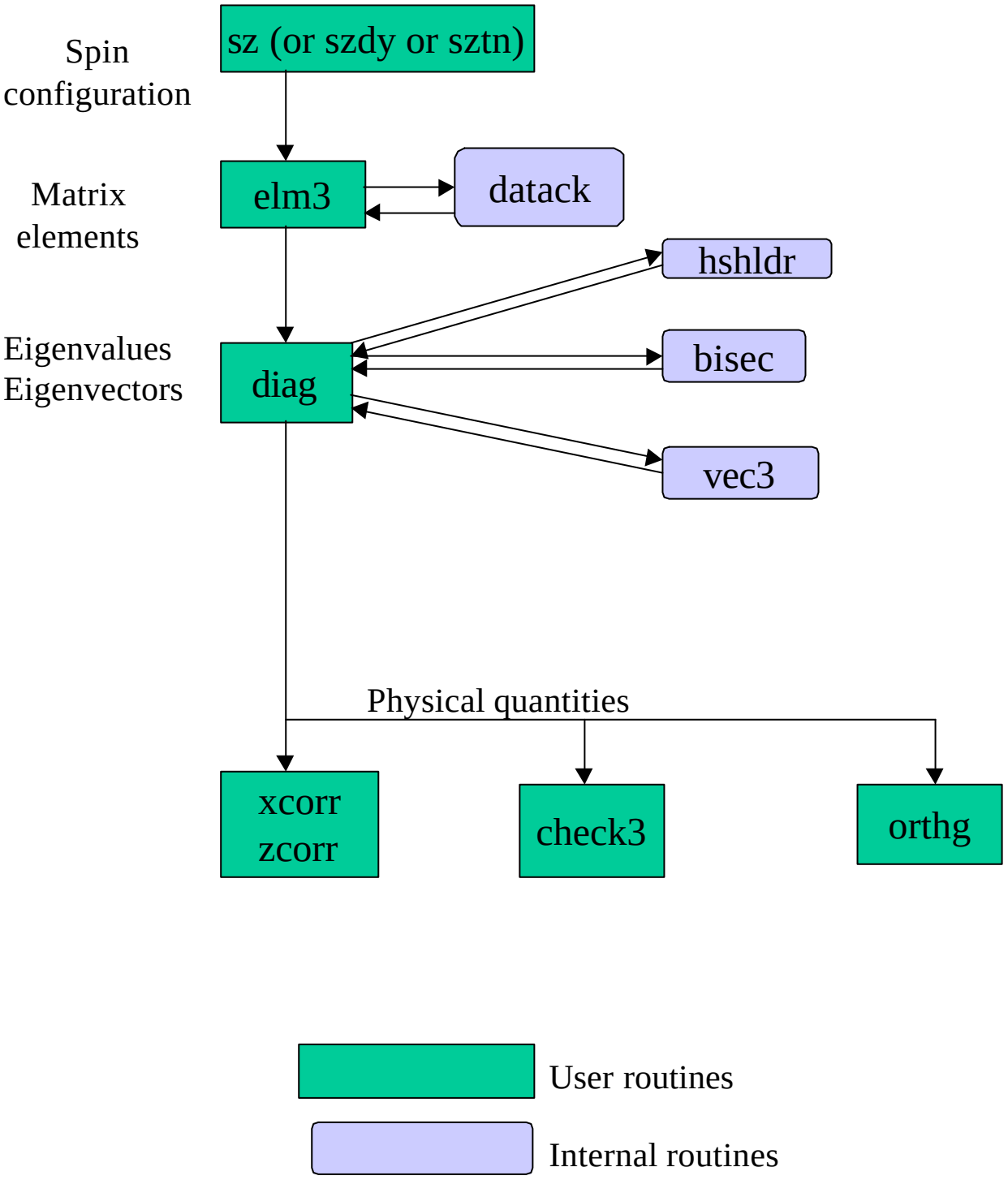
# 1.3 Process Diagram for Large Matrices (Group L)



## 1.4 Process Diagram for Mid-size Matrices (Group M)



## 1.5 Process Diagram for Small Matrices (Group S)



2.1 Subroutines for Large Matrices --Group L
(2) Eigenvalues by the Lanczos method
Connection example for ipair(ibond*2)