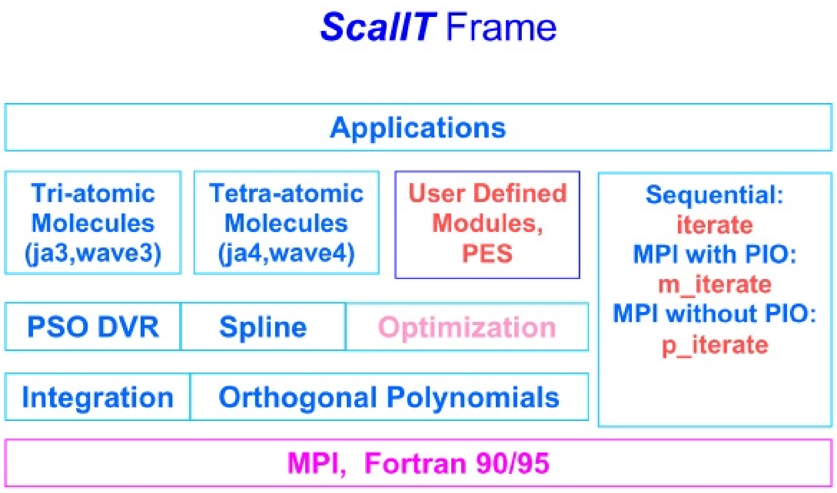
ScalIT (*Scal*able *IT*eration) Documentation
Version: June 2013

June 11, 2013

Top directory structure in *ScalIT* package:

bin/	Executable files
data/	V_{eff} spline data files for PSO-VBR calculation
include/	Header files used in *ScalIT* package
lib/	Libraries for *ScalIT* package
module/	*ScalIT* FORTRAN90 modules
src/	System Hamiltonian, potential energy surface, sequential and MPI version of iterate, etc.
Makefile	makefile
Makefile.setup	makefile setup



The *ScalIT* suite of parallel codes have been developed as a result of support from the following funding agencies:

US Department of Energy:	DE-FG03-02ER25534
National Science Foundation:	NSF CHE-1012662
NASA:	NNX13AJ49G

bin/ directory:

Iterate binaries

iterate	Sequential version
p_iterate	MPI version without parallel IO
m_iterate	MPI version with parallel IO

Wave function calculations

wave3 (wave3hs, wave3hs_e, wave3hs_o) (wave3jb, wave3jb_e, wave3jb_o)	Wave function of tri-atomic molecules in Hyper-spherical and Jacobi coordinate s

Rovibrational energy level calculations for tri-atomic systems.
The programs in each subdirectory:

xxvlr: DVR calculation of r in Jacobi coordinate.
xxvBr: DVR calculation of R in Jacobi coordinate.
[m,p]xx[_o,_e]: calculate Hamiltonian matrix, [**m**=MPI with parallel IO, **p**=MPI without parallel IO, **none**=sequential program], [**_o**: odd symmetry for permutation of atoms related to r, **_e**: even symmetry for permutation of atoms related to r]. [**xx**: name of the molecule]

ho2	HO_2

data/ directory:

Spline data files of xx *tri-atomic system for V_{eff} for PSO-DVR calculation.*

xx_vlr:	DVR calculation of r in Jacobi coordinate
xx_vBr:	DVR calculation of R in Jacobi coordinate

lib/ directory:

libosb.a	Library related to sequential version of OSB [iterate].
libmosb.a	Library related to MPI version of iterate [m_iterate,p_iterate].
libdosb.a	Library related to the generation of Hamiltonian matrix [dosb].

include/ directory:

OSB module

osb.interface.h	Interface of OSB module
osb.data.h	Define data used in OSB module
osb.io.h	Read/write data in OSB module
osb.print.h	Read input file and print related information of OSB module
osb.init.h	Initialize OSB module, mainly allocate memory for the data
osb.index.h	Transfer indices between 1D array and multi-dimension array
osb.diag.init.h	Block Jacobi diagonalization Initialization
osb.diag.h	Block Jacobi diagonalization
osb.progDiag.h	Driver to perform or test Block Jacobi diagonalization
osbw.progRes.h	Driver to perform or test the diagonal matrix after Block Jacobi diagonalization
osb.hx.h osb.hxcx.h osb.hxdx.h	Matrix-vector product H⋆x
osb.progHX.h	Driver to perform or test Matrix-vector product H⋆x
osb.hij.h osb.hij1.h osb.hijcx.h	Calculate matrix elements of OSBW matrix
osb.progHij.h	Driver to perform or test OSBW matrix
osb.pij.h	Calculate the inversion of OSBW matrix
osb.px.h osb.pxdx.h osb.mypxdx.h	Matrix-vector product P⋆x
osb.vx.h osb.vi.h osb.vi1.h	calculate transformation matrix V and V⋆x
osb.progPX.h	Driver to perform or test Matrix-vector product P⋆x
osb.osbw.h	Initialize OSBW preconditioner

osbw module

osbw.qmr.h osbw.qmrdx.h osbw.qmrcx.h	QMR implementation
osbw.progQMR.h	Driver to perform or test QMR algorithm
osbw.lanczos.h	Implementation of Lanczos algorithm
osbw.pist.h	Implementation of PIST algorithm
osbw.pistconv.h osbw.pistconverg.h	Convergence testing in PIST algorithm
osbw.progEig.h	Driver to perform or test eigenvalues based on PIST
osbw.polylan.h	Implementation of poly-nomial Lanczos algorithm
osbw.progLan.h	Driver to perform or test eigenvalues based on Lanczos
osbw.crp.h	CRP calculations based on PIST algorithm

p_osb_h

mosb3.data.h	Data and data structure used in MOSB3 module
mosb3.init.h	Initialization
mosb3.hx.h mosb3.hxdx.h mosb3.hxcx.h	H⋆x operations
mosb3.px.h mosb3.pxdx.h	P⋆x operations
mosb3.vxseq.h mosb3.vxseq_dx.h	V⋆x operations
mosb3.mxgrid.h mosb3.mxgrid_dx.h mosb3.mxgrid_cx.h mosb3.vxgrid.h	Matrix-vector product operations where vector is grid distributed.
mosb3.mxseq.h mosb3.mxseq_dx.h mosb3.mxseq_cx.h mosb3.vxgrid_dx.h	Matrix-vector product operations where vector is sequentially distributed.
mosb3.updatex.h	Update the vector x

p_osb_pio	**MOSB module based on parallel IO, using mosb3 to perform real matrix-vector product operations**
mosb.data.h	Data and structures used in mosb module
mosb.io.h mosb.io10.h mosb.pio.h mosb.print.h	Read/write data
mosb.init.h	Initialization of mosb
mosb.util.h	Common subroutines
mosb.diaginit.h	Initialization of Block Jacobi diagonalization
mosb.diag.h	Block Jacobi diagonalization
mosb.progDiag.h	Driver to perform or test Block Jacobi diagonalization
mosb.hx.h mosb.hxdx.h mosb.hxcx.h	Matrix-vector product H⋆x
mosb.progHX.h	Driver to perform or test Matrix-vector product H⋆x
mosb.px.h mosb.pxdx.h	Matrix-vector product P⋆x
mosb.progPX.	Driver to perform or test Matrix-vector product P⋆x
mosbp.qmr.h mosbp.qmrcx.h mosbp.qmrdx.h	QMR implementation
mosbp.progQMR.h	Driver to perform or test QMR implementation
mosbp.lan.h mosbp.polylan.h	Lanczos implementation
mosbp.progLan.h	Driver to perform or test Lanczos implementaion
mosbp.pist.h mosbp.pistconverg.h mosbp.pistconv.h	PIST implementation
mosbp.progEig.h	Driver to perform or test eigenvalues
mosbw.data.h	Data used for OSBW preconditioener
mosbw.io.h	Read/write OSBW preconditioner data
mosbw.init.h	Initialization of OSBW preconditioner
mosbw.osbw.h	Preprocessing of OSBW
mosbw.hmat.h	Calculate OSBW matrix
mosbw.hij.h mosbw.hij1.h mosbw.hijcx.h mosbw.hijcx1.h	Calculate element of OSBW matrix.
mosbw.progHij.h	Driver to perform or test OSBW matrix.
mosbw.pij.h	Calculate the inversion of OSBW matrix
mosbw.vi.h mosbw.vx.h	Calculate the transformation matrix V and matrix-vector product V⋆x
mosbp.crp.h	CRP calculation based on QMR algorithm.

p_osb_nopio	**POSB module without parallel IO, using mosb3 to perform real matrix-vector product operations. Most subroutines are from MOSB directory.**
posb.io.h posb.io1.h posb.io2.h	
posb.diag.h	
posb.progHX.h	
posb.progPX.h	
posbw.hij.h posbw.hij1.h	
posbw.hijcx.h posbw.hijcx1.h	

module/ directory:

gauntmod.mod	Module for type of angular momentum coupling coefficients.
mja3.mod	MPI Module to generate Hamiltonian matrix for tri-atomic system with parallel IO.
mosbtype.mod	Module to define basic data structures used in MOSB/ POSB Modules.
pja3.mod	MPI Module to generate Hamiltonian matrix for tri-atomic system without parallel IO.
presinc.mod	Presinc Module.
integralpjm.mod	Gauss quadrature integral Module for PJM indices.
mja4.mod	MPI Module to generate Hamiltonian matrix for tetra-atomic system with parallel IO.
osb.mod	OSB Module.
pja4.mod	MPI Module to generate Hamiltonian matrix for tetra-atomic system without parallel IO.
threejmod.mod	
ja3.mod	Module to generate Hamiltonian matrix for tri-atomic system.
mosb.mod	MOSB Module (MPI version of OSB Module with parallel IO).
osbtype.mod	Module to define basic data structures used in OSB Module.
posb.mod	POSB Module (MPI version of OSB Module without parallel IO).
wave3hs.mod	Module to calculate tri-atomic system wavefunction in hyper-spherical coordinates.
ja4.mod	Module to generate Hamiltonian matrix for tetra-atomic system.
mosbp.mod	MOSBP Module.
osbw.mod	OSBW Module.
posbp.mod	POSBP Module.
wave3jb.mod	Module to calculate tri-atomic system wavefunction in Jacobi coordinates.

src/ directory:

drivers	**Testing drivers**
ja3	Testing drivers for tri-atomic system.
fitv.drv.f90	
mja3.drv.f90	
pot.drv.f90	
psolr.drv.f90	
ja3.drv.f90	
pja3.drv.f90	
psoBr.drv.f90	
ja4	Testing drivers for tetra-atomic system.
fitv.drv.f90	
mja4.drv.f90	
pot.drv.f90	
psolr1.drv.f90	
ja4.drv.f90	
pja4.drv.f90	
psoBr.drv.f90	
psolr2.drv.f90	

hamiltonians

ja3.f90	Calculate Hamiltonian matrix elements of tri-atomic system.
ja4.f90	Calculate Hamiltonian matrix elements of tetra-atomic system.
pja3.f90	Calculate Hamiltonian matrix elements of tri-atomic system: MPI version without parallel IO.
pja4.f90	Calculate Hamiltonian matrix elements of tetra-atomic system: MPI version without parallel IO.
mja3.f90	Calculate Hamiltonian matrix elements of tri-atomic system: MPI version with parallel IO.
mja4.f90	Calculate Hamiltonian matrix elements of tetra-atomic system: MPI version with parallel IO.

index

index_A2B2_A10.f90	index_A2B2_A1.f90	(j1,j2,j,K) indices for tetra-atomic molecule systems
index_A2B2_A1k0.f90	index_A2B2_A1k.f90	
index_A2B2_A2k0.f90	index_A2B2_A2k.f90	
index_A2B2.f90	index_ABCD0.f90	
index_ABCD.f90		
index_AB2e.f90	index_AB2o.f90	(j, K, m) indices for tri-atomic
index_ABC.f90		

iterate

Makefile	makefile
osb.f90 **osb**	The entry of OSB module
osbr.f90 **osbr**	The entry of OSB module
osbw.f90 **osbw**	The entry of OSBW module
osbtype.f90	Definitions of Data structures used in osb, osbr and osbw modules

p_osb_nopio **POSB module without parallel IO, using mosb3 to perform real matrix-vector product operations. Most subroutines are from MOSB directory.**

posb.f90
posbp.f90

p_osb_pio **MOSB module based on parallel IO, using mosb3 to perform real matrix-vector product operations**

mosb.f90	Entry of mosb module
mosbp.f90	

p_osb_h

mosb3.f90

prog/

osb.drv.f90	Generate program iterate.
posb.drv.f90	Generate program p_iterate.
mosb.drv.f90	Generate program m_iterate.
mosb1.drv.f90	Generate program m1_iterate.

p_type

mosbtype.f90 Definition of basic data structures used in MOSB / POSB

libdosb/ directory

ham/coeff

	Calculate the coefficients
cg.f90	Clebsch-Gondan coefficients
gaunt.f90 gauntmod.f90	Gaunt coefficients
lnFn.f90	ln(n!)
threej.f90 threejmod.f90	3-j symbols
sixj.f90	6-j symbols
ninej.f90	9-j symbols

ham/dvr

	DVR matrix calculations
advr.f90 ddvr.f90 sdvr.f90	General and common procedures to
dvr1.f90 dvr2.f90 dvrcx.f90	generate DVR points and matrices.
dvr.f90 dvrt.f90	
getv.f90	
mmt.f90	
mtm.f90	
dvrvx.f90 gauss_dvr.f90	
sinc1_dvr.f90 sinc2_dvr.f90 sinc.f90	Sinc DVR
sinc_xh.f90	

ham/fun

	DVR matrix calculations
amoeba.f90 linmin.f90	Several simple subroutines to get the
getRoot.f90	
mead.f90 powell.f90 simpleMin.f90 simplex.f90	minimization of the function
splint.f90 spline.f90	Subroutines for splining functions

ham/index

getm3.f90 index_3j0.f90	(j1,j2,j,K) indices for tetra-atomic
index_CF.f90 index_CFj.f90	molecule systems.
index_CG.f90 index_CGj.f90 index_CGmk.f90	
index_gaunt.f90 index_jm.f90 index_pjm.f90	
index_M1D.f90 index_nlm.f90	
index_node.f90	
index_M1D.f90	

ham/integral **Gauss quadrature integral frame**

gauIAssLagu.f90 gauIChev.f90
gauIHermit.f90 gauILagu.f90
gauILege.f90 integralDVR.f90
integralFmk.f90 integralPjm.f90
gauJacb.f90

ham/poly

chev.f90
gauHermit.f90
gauLege.f90
pj.f90
poly.f90
trigonom.f90
yjm.f90
gauChev.f90
gauLagu.f90
lagu.f90
pjm.f90
trigo.f90
yj.f90

ham/util	**Some utilities**
presinc.f90	Presinc module

libmosb/ directory
p_base

index_node.f90
vector_cx_mpi.f90
vector_mpi.f90

p_io

mio.f90 mio_seq.f90	Read/write real/complex data in binary format using MPI
mio_diag.f90 mio_grid.f90	parallel IO for extreme large data size (>GB),
mio_dist.f90	may be slower than the corresponding nio_xxx subroutines.
nio_seq.f90 nio_diag.f90	Read/write real/complex data in binary format using MPI parallel
nio_grid.f90	IO for small data size (<GB).
pio_seq.f90 pio_diag.f90	Read/write real/complex data in binary format without MPI
pio_grid.f90 pio_seqall.f90	parallel IO

p_iterate/ directory
diagmpi

hosbdiag_mpi.f90 hosbdiag_dx_mpi.f90	Do block Jacobi diagonalization
hosbdiag_cx_mpi.f90	
Jacobi_mpi.f90 jacobi_cx_mpi.f90	Jacobi diagonalization for each block

diagseq

hosbdiag_seq.f90 hosbdiag_dx_seq.f90	Do block Jacobi diagonalization
hosbdiag_cx_seq.f90	
hinitdep.f90 hinitxyz.f90	Initialization of matrix for block Jacobi diagonalization
jacobi.f90 jacobi_cx.f90	Jacobi diagonalization for each block
sumdiag.f90 vupdate.f90	Update the transformation matrices in Jacobi diagonalization

hpxseq — **New version of basic matrix-vector product operations**

hpx_cx.f90 hpx_dx.f90
hpx_cx_seq.f90
hpx_dx_seq.f90 hpx_seq.f90
hpx.f90

lanczos

lan_mpi.f90	Lanczos algorithm for real symmetric matrices
lanHij_mpi.f90	Calculate Lanczos matrices for Lanczos algorithm
lanEx_mpi.f90	Wrapper of Lanczos algorithm for real symmetric matrices
lan_dx_mpi.f90	Lanczos algorithm for complex Hermitian matrices
lanHij_dx_mpi.f90	Calculate complex Hermitian matrices for Lanczos algorithm
lanEx_dx_mpi.f90	Wrapper of Lanczos algorithm for complex Hermitian matrices
lan_cx_mpi.f90	

orth — Implementation of Gram-Schmidt and its modified version algorithms

gsorth_mpi.f90	For real vectors.
gsorth_cx_mpi.f90	For complex vectors
gsorth_sx_mpi.f90	For real/complex vectors

pist — **Base on modified Gram-Schmidt algorithm**

pist_mpi.f90 pist_cx_mpi.f90 pist_dx_mpi.f90 pist_sx_mpi.f90	Implementation of PIST algorithms for real, complex matrices
Pistf_mpi.f90 pistf_cx_mpi.f90 pistf_dx_mpi.f90 pistf_sx_mpi.f90	Same as pist_xx.f90, but also store PIST vectors in the files
pistHij_mpi.f90 pistHij_dx_mpi.f90 pistHij_cx_mpi.f90 pistHij_sx_mpi.f90	Calculate PIST matrix elements
pistEx_mpi.f90 pistEx_dx_mpi.f90 pistEx_cx_mpi.f90 pistEx_sx_mpi.f90	Wrapper of the corresponding PIST implementations.

pistgs — **Base on original Gram-Schmidt algorithm**

pist_mpi.f90 pist_cx_mpi.f90 pist_dx_mpi.f90 pist_sx_mpi.f90	Implementation of PIST algorithms for real, complex matrices
Pistf_mpi.f90 pistf_cx_mpi.f90 pistf_dx_mpi.f90 pistf_sx_mpi.f90	Same as pist_xx.f90, but also store PIST vectors in the files
pistHij_mpi.f90 pistHij_dx_mpi.f90 pistHij_cx_mpi.f90 pistHij_sx_mpi.f90	Calculate PIST matrix elements
pistEx_mpi.f90 pistEx_dx_mpi.f90 pistEx_cx_mpi.f90 pistEx_sx_mpi.f90	Wrapper of the corresponding PIST implementations.

qmr — **QMR implementation in MPI environment**

qmr_mpi.f90	Real version of QMR
qmrcx_mpi.f90	Complex version of QMR

util

getsimpos.f90 getga.f90 getpos.f90	Get the position information in global array distribution
recvnodes.f90 recvindex.f90	Get the data exchanging information for receiving data nodes
sendnodes.f90 sendindex.f90 seqdata.f90	Get the data exchanging information for sending data nodes
mvindex.f90 mvxindex.f90 vindex.f90	Get the information about the transformation matrix V

libosb/ directory

obase

osbtype.f90	Definition of basic data structures used in OSB
diag.f90	Wrapper of LAPACK diagonalization subroutines
mm.f90	Wrapper of BLAS matrix-matrix multiplication subroutines
matrix_cx.f90 matrix.f90 matrix_sx.f90 matrix_dx.f90	Commonly used matrix-matrix multiplication operations for both real and complex matrices.
vector.f90 vector_cx.f90	Commonly used vector-vector product operations for both real and complex vectors. Most similar to BLAS subroutines, and are seldom used for FORTRAN90 provides most of these functions. May be used for optimization of program later.
util.f90 / util_cx.f90	Some commonly used functions.

oio

io.f90	Read/write real data in binary format and sequential access mode
io_cx.f90	Read/write complex data in binary format and sequential access mode
iodirect.f90	Read/write real data in binary format and direct access mode
iodirect_cx.f90	Read/write complex data in binary format and direct access mode

hpxdir **The old version of matrix-vector product operations**

dhpx_cx.f90 dhpx_dx.f90
dhpx.f90

iterate/ directory

diag/

hinitdep.f90 hinitxyz.f90	Initialization of matrix for block Jacobi diagonalization
hosbdiag.f90 hosbdiag_dx.f90 hosbdiag_cx.f90	Do block Jacobi diagonalization
jacobi.f90 jacobi_cx.f90	Jacobi diagonalization for each block
sumdiag.f90 vupdate.f90	Update the transformation matrices in Jacobi diagonalization

hpx

hpx_cx.f90 hpx_dx.f90 hpx.f90	Basic matrix-vector product operations

lanczos

lan.f90	Lanczos algorithm for real symmetric matrices
lanHij.f90	Calculate Lanczos matrices for Lanczos algorithm
lanEx.f90	Wrapper of Lanczos algorithm for real symmetric matrices
lan_dx.f90	Lanczos algorithm for complex Hermitian matrices
lanHij_dx.f90	Calculate complex Hermitian Lanczos matrices for Lanczos algorithm
lanEx_dx.f90	Wrapper of Lanczos algorithm for complex Hermitian matrices

orth

gsorth.f90	Implementation of Gram-Schmidt and its modified version algorithms for real vectors.
gsorth_cx.f90	Implementation of Gram-Schmidt and its modified version algorithms for complex vectors.
gsorth_sx.f90	Implementation of Gram-Schmidt and its modified version algorithms for real/complex vectors.

pist

pist.f90 pist_cx.f90 pist_dx.f90 pist_sx.f90	Implementation of PIST algorithms for real, complex matrices
pistf.f90 pistf_cx.f90 pistf_dx.f90 pistf_sx.f90	Same as pist_xx.f90, but also store PIST vectors in the files
pistHij.f90 pistHij_dx.f90 pistHij_cx.f90 pistHij_sx.f90	Calculate PIST matrix elements
pistEx.f90 pistEx_dx.f90 pistEx_cx.f90 pistEx_sx.f90	Wrapper of the corresponding PIST implementations.

qmr	**QMR implementation**
qmr.f90	Real version of QMR
qmrcx.f90	Complex version of QMR

prog/

osb.drv.f90	Generate osb for sequential program
testHij.f90	Driver program to test OSBW matrix
testHX.f90	Driver program to test H⋆x product
testPX.f90	Driver program to test P⋆x product

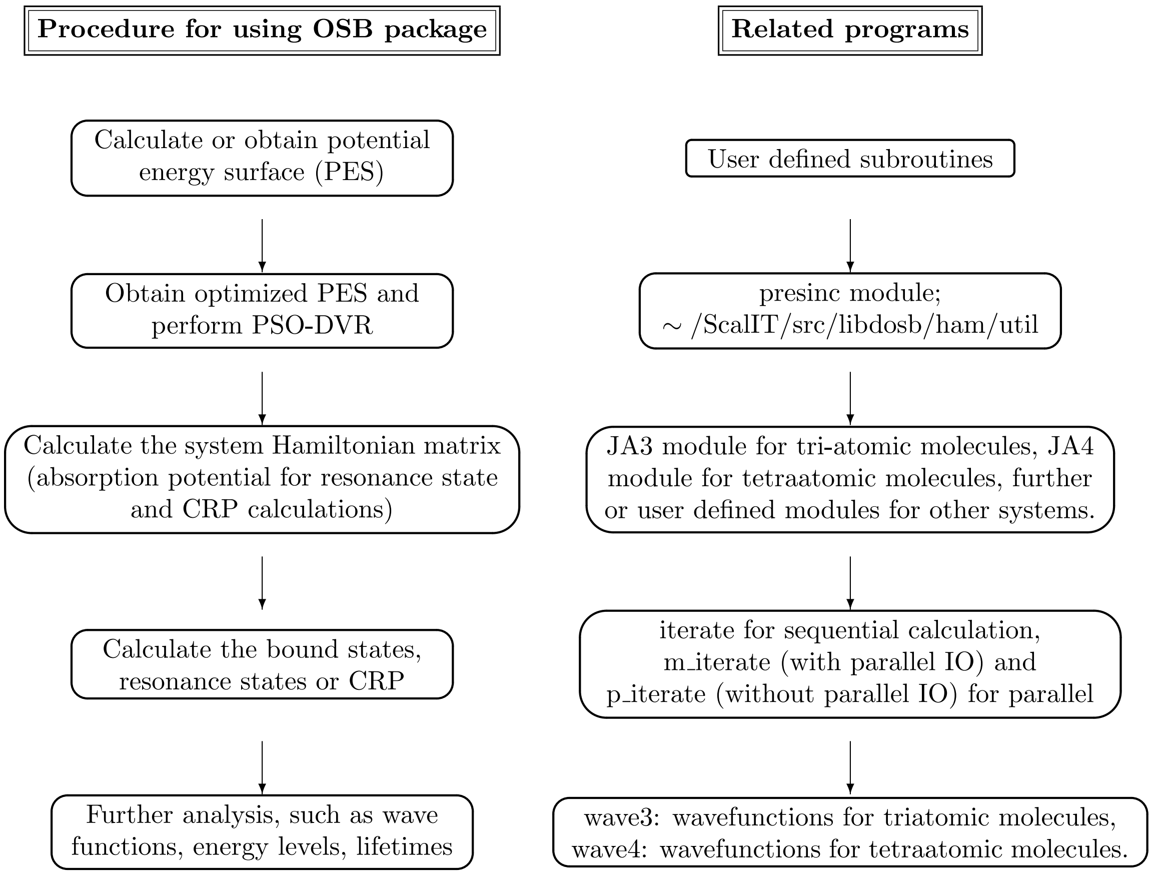
hpxrma	**New version of basic matrix-vector product operations based on RMA (not tested)**
comm_cx.f90 comm.f90	Basic RMA operations
hxga_cx.f90 hxga_dx.f90 hxga.f90	Matrix-vector products based on RMA

systems

pot_XX.f90	System specific subroutine for PES

wave3

wave3	wave3[hs,jb] modules to calculate wave functions in hyper-spherical and Jacobi coordinates for tri-atomic molecules.


Procedure for using OSB package
Related programs
Calculate or obtain potential energy surface (PES)
User defined subroutines
Obtain optimized PES and perform PSO-DVR
presinc module; ∼ /ScalIT/src/libdosb/ham/util
Calculate the system Hamiltonian matrix (absorption potential for resonance state and CRP calculations)
JA3 module for tri-atomic molecules, JA4 module for tetraatomic molecules, further or user defined modules for other systems.
Calculate the bound states, resonance states or CRP
iterate for sequential calculation, m_iterate (with parallel IO) and p_iterate (without parallel IO) for parallel
Further analysis, such as wave functions, energy levels, lifetimes
wave3: wavefunctions for triatomic molecules, wave4: wavefunctions for tetraatomic molecules.

Presinc module (Step I)

This is the input format for the 1st step, known as the presinc module. It is the module for creating PSODVRs for a given molecule. This step is only done once for every radial degree of freedom.

Typically, the input files are named “inputfilename.pin” with the output given as “outputfilename.pout”.

Here is an example that is given in the walkthrough explained in detail. It is for the HO_2 triatomic system. The filename is 1dDVRlr.pin.

File name: 1dDVRlr.pin	Here is what each entry is, followed by what it means:
2 14578.4716590D0 6000 80 T	nType mass Nmax N useSP
1.0D0 11.0D0	Xmin Xmax
input/ho2lr.dat	outFile (**binary file**)
∼ / ScalIT/data/ho2/ho2_vlr.dat	(spFile)*

nType : int	: The type of SINC DVR, currently only two options are available: 1: sinc1 DVR, the range is [-Xmax:Xmax] If this option is chosen, only xmax must be specified in file. 2: sinc2 DVR, the range is [Xmin:Xmax] If this option is chosen, both Xmin and Xmax must be specified in file.
mass : double	: the mass (usually reduced) of the degree of freedom. The units of the mass is dependant upon the potential given. Be careful to choose them correctly, if wrong, all subsequent calculations will be wrong.
Nmax : int	: The maximum number of original SINC DVR points if sinc2 DVR is chosen. For sinc1 DVR, this is just the number of points for the range [0:Xmax], and the total points will be 2*Nmax+1.
N : int	: The number of eigenvalues to be calculated and shown. This number is the maximum number of PSODVR values you can have in subsequent steps.
useSP : bool	: Parameter for whether or not splining functions to calculate optimized potential. If used, then spFile must be specified. True: use splining functions, specify spFile False: don’t use splining functions, do not specify spFile
Xmin, Xmax : double	: The parameters to specify the DVR range for sinc1 or sinc2 DVRs
outFile : string	: Output filename to store intermediate DVR results, which can be used for further calculation.
spFile : string	: The data file providing optimized potential points for use when useSP=True.

The format of the data file is as follows:

```
Nmax
x(1)        V(1)
x(2)        V(2)
...         ...
...         ...
x(Nmax)     V(Nmax)
```

Here is the routine that reads in the input file if users are confused.
****

```
READ(*,*) nType, mass, Nmax, N, useSP
 IF (nType==1) THEN
 READ(*,*) Xmax
 Xmin=-Xmax
 maxNDVR = 2*Nmax + 1
 ELSE
 READ(*,*) Xmin, Xmax
 maxNDVR = Nmax
 END IF
 READ(*,'(A)') outFile
 IF(useSP) READ(*,'(A)') spFile
```

****

JA3 module (Step II: for triatomic molecule)

This is the input format that is used for the JA3 module, which is the second step of the procedure. It is assumed that you have already done step one, which is to create the PSO-DVR's for all coordinates that require them.

The input file name, standard for this step, is "filename.hin" and "filename.hout" for the output.

Here is an example of the J=0 HO_2 (Jacobi (r,R,theta)) input file for this step, in this example, there are no absorbing potentials. If absorbing potentials are included, the structure is more complicated.

File name: r22R24jmx56.hin	Here are what each entry is followed by what it means:
0 T 56 300	JTot parity jmax nGI
0 0 0 T 10.0D0	FcFlag CbFlag AbsFlag useSP Ecutoff
input/h0_og28.dat	fH0
input/h0gm_og28.dat	fH0gm
14578.471659D0 2.51D0 22	MASS(1) RE(1) NDVR(1)
input/ho2lr.dat	fVRlr
1781.041591D0 2.46D0 24	MASS(2) RE(2) NDVR(2)
input/ho2Br.dat	fVRBr
600 1	nDVR(3) ReFlag
input/hre.dat	fRE
~ / ScalIT/data/ho2/ho2_vlr.dat	fSpVRlr
~ / ScalIT/data/ho2/ho2_vBr.dat	fSpVRBr
...	...
...	...

JTot: int	: Total J quantum number
parity: bool	: Parity, true=even, false=odd
jmax: int	: The maximum j quantum number that will be used
nGI: int	: The number of integral points for Gauss quadrature
FcFlag : int	: Whether the coordinate(s) is (are) fixed. Available options are: 0: FCNONE, no coordinate is fixed 1: FCBR, Jacobi coordinate R is fixed 2: FCLR, Jacobi coordinate r is fixed 3: FCALL, bot (r,R) are fixed
CbFlag: int	: Whether the coordinates (r,R) are combined Available option are: 0: CBNONE, no Jacobi coordinates (r,R) are fixed 1: CBALL, All Jacobi coordinates (r,R) are fixed NOTE: if CBALL, then the input of osb becomes two layers instead of three (for triatomic), with the inside layer being (nDVRlr x nDVRBr)
AbsFlag: int	: Whether the absorption potential will be calculated. 0: only the polynomial absorption potential is implemented: $Ap = A_0((r-r_{min})/(r_{max}-r_{min}))^n$

useSP: bool	: How to calculate optimized potentials. Available options are: T: Use splining function to calculate optimized potentials V(r) and V(R). The optimized potentials are provided by the user at runtime in files spVRlr and spVRBr. F: Use analytical function to calculate optimized potentials V(r) and V(R). The user needs to proved subroutines to calculate these. These potentials go in the $(mol)_pot.f90 file under fitVBR and fitVlr subroutines respectively located in the ScalIT/pes directory.
Ecutoff: double	: Cutoff energy, works only if CbFlag != CBALL
fH0: string	: Filename to store H0 matrix for coordinates (r,R)
fH0gm: string	: Filename to store H0 matrix for coordinate (theta)

MASS(i): double	: Reduced mass for coordinate i:
	i= 1: r
	i= 2: R

RE(i): double	: The equilibrium values for coordinate i:
	i= 1: r
	i= 2: R

NDVR(I): int	: The final number of DVR points in coordinate i:
	i= 1: r
	i= 2: R

fVRlr: string	: The filename that stores all eigenvalues and eigenvectors for coordinate r. This is the output data file of the presinc module.
fVRBr: string	: The filename that stores all eigenvalues and eigenvectors for coordinate R. This is the output data file of the presinc module.
nDVR(3): int	: The final contracted size of the angular basis functions. This is a ceiling value. So if the combined basis is lower than this number, the data stored will be that number, and not this ceiling. If this is set to 0, then the data stored will be non-contracted. (TEST THIS!!!)
ReFlag: int	: Whether to store or extract the eigenvalues and eigenvectors from file cooresponding to coordinate theta at the equilibrium values (RE(1),RE(2)). The basis functions for coordinate theta were contracted from the original associated Legendre functions according to the eigenvalues at the equilibrium values (RE(1),RE(2)). Available options are: > 0: Store the eigenvalues and eigenvectors to file fRE. < 0: Extract the eigenvalues and eigenvectors from file fRE. = 0: Do not store or extract eigenvalues and eigenvectors.
fRE : string	: the filename to store the eigenvalues and eigenvectors corresponding to coordinate theta at the equilibrium values (RE(1),RE(2))
fSpVRlr: string	: The filename to provide the optimized potential V(r). Used only when useSp=T.
fSpVRBr: string	: The filename to provide the optimized potential V(R). Used only when useSp=T.

en(i): int	: the “n” parameter for Absorption potential i:
	i= 1: Either the only parameter input if AbsFlag=ABS_ONE or the first of two if AbsFlag=ABS_TWO.
	i= 2: The second paramter input only if Abs_Flag=ABS_TWO
A_0(i): double	: the A_0 parameter for Absorption potential i
Rabs0(i): double	: The r_{min} parameter for Absorption potential i
Rabs1(i): double	: The r_{max} parameter for Absorption potential i
fABS: string	: The filename to store absorption potentials.

JA4 module (Step II: for tetra-atomic molecule)

This is the input format that is used for the JA4 module, which is the second step of the procedure. It is assumed that you have already done step one, which is to create the PSO-DVR's for all coordinates that require them.

The input file name, standard for this step, is "filename.hin" and "filename.hout" for the output.

Here is an example of the a J=0 Ne_4 (Jacobi (r1,r2,R,theta1,theta2,phi)) input file for this step, in this example, there are no absorbing potentials. If absorbing potentials are included, the structure is more complicated.

File name: ne4.hin	Here are what each entry is followed by what it means:
0 T	JTol parity gmMethod
30 30 60 30 30 30	JMAX(theta1) JMAX(theta2) JMAX(phi) NGI(theta1) NGI(theta2) NGI(phi)
0 0 0 F 1.0	FcFlag CbFlag AbsFlag useSP Ecutoff
input/h0.dat	fH0
input/h0gm.dat	fH0GM
55.442105 1.1225 8	MASS(r1) RE(r1) NDVR(r1)
··/input/ne4vlr.dat	fVRlr1
55.442105 1.1225 8	MASS(r2) RE(r2) NDVR(r2)
··/input/ne4vlr.dat	fVRlr1
110.88421 0.05 14	MASS(R) RE(R) NDVR(R)
··/input/ne4vBr.dat	fVRlR
700 0	nDVR(angular) ReFlag
input/ne4hre.dat	fRe
~ /ScalIT/data/ne4/vlr1.dat	fSPVRlr1
~ /ScalIT/data/ne4/vlr2.dat	fSPVRlr2
~ /ScalIT/data/ne4/vBr.dat	fSPVRlR

JTot: int	: Total J quantum number
parity: bool	: Parity, true=even, false=odd
jmax: int	: The maximum j quantum number that will be used
nGI: int	: The number of integral points for Gauss quadrature
FcFlag: int	: Whether the coordinate(s) is (are) fixed. Available options are: 0: FCNONE, no coordinate is fixed 1: FCR1R2, Jacobi coordinates (r1,r2) are fixed 2: FCLBRR1, Jacobi coordinates (R,r1) are fixed 3: FCLBRR2, Jacobi coordinates (R,r2) are fixed 2: FCLBRR1, Jacobi coordinates R is fixed 3: FCALL, bot (r,R) are fixed

CbFlag: int	: Whether the coordinates (r,R) are combined Available option are: 0: CBNONE, no Jacobi coordinates are fixed 1: CBR1R2, Jacobi coordinates (r1, r2) are fixed 2: CBBRR1, Jacobi coordinates (R, r1) are fixed 3: CBBRR2, Jacobi coordinates (R, r2) are fixed 4: CBALL, All Jacobi coordinates (r1,r2,R) are fixed NOTE: if CBALL, then the input of osb becomes two layers instead of three (for triatomic), with the inside layer being (nDVRlr x nDVRBr) three layers instead of four (for tetramer)
AbsFlag : int	: Whether the absorption potential will be calculated. only the polynomial absorption potential is implemented: $\mathrm{Ap} = \mathrm{A}_0((\mathrm{r}\text{-}\mathrm{r}_{min})/(\mathrm{r}_{max}\text{-}\mathrm{r}_{min}))^n$ Available options are: 0: ABS_NONE, no absorption potential is calculated, in bound state calculations. 1: ABS_ONE, input absorption potential is calculated, in resonance state calculations 2: ABS_TWO, input and output absorption potential is calculated, in CRP calculations.
useSP: bool	: How to calculate optimized potentials. Available options are: T: Use splining function to calculate optimized potentials V(r1), V(r2), and V(R). The optimized potentials are provided in files fSPVRlr1, fSPVRlr2 and fSPVRlR. F: Use analytical function to calculate optimized potentials V(r) and V(R). The user needs to proved subroutines to calculate these. (NOTE: as of 10-5-2010, this has not been tested. -Corey Petty)
Ecutoff: double	: Cutoff energy, works only if CbFlag = CBALL
fH0: string	: Filename to store H0 matrix for coordinates (r1,r2,R)
fH0gm: string	: Filename to store H0 matrix for coordinate (angular)

MASS(i): double	: Reduced mass for coordinate i:
	i= 1: r1
	i= 2: r2
	i= 3: R

RE(i): double	: The equilibrium values for coordinate i:
	i= 1: r1
	i= 2: r2
	i= 3: R

NDVR(i): int	: The final number of DVR points in coordinate i:
	i= 1: r1
	i= 2: r2
	i= 3: R

fVRlr1: string	: The filename that stores all eigenvalues and eigenvectors for coordinate r1. This is the output data file of the presinc module.
fVRlr2: string	: The filename that stores all eigenvalues and eigenvectors for coordinate r2. This is the output data file of the presinc module.
fVRBr: string	: The filename that stores all eigenvalues and eigenvectors for coordinate R. This is the output data file of the presinc module.

nDVR(angular): int	: The final contracted size of the angular basis functions. This is a ceiling value. So if the combined basis is lower than this number, the data stored will be that number, and not this ceiling. If this is set to 0, then the data stored will be non-contracted. (TEST THIS!!!)
ReFlag: int	: Whether to store or extract the eigenvalues and eigenvectors from file cooresponding to coordinate theta at the equilibrium values (RE(1),RE(2)). The basis functions for coordinate theta were contracted from the original associated Legendre functions according to the eigenvalues at the equilibrium values (RE(1),RE(2)). Available options are: >0: Store the eigenvalues and eigenvectors to file fRE. <0: Extract the eigenvalues and eigenvectors from file fRE. $=0$: Do not store or extract eigenvalues and eigenvectors.
fRE: string	: The filename to store the eigenvalues and eigenvectors corresponding to angular coordinates at equilibrium values (RE(1),RE(2),RE(3)).
fSpVRlr1: string	: The filename to provide the optimized potential V(r1). Used only when useSp=T.
fSpVRlr2: string	: The filename to provide the optimized potential V(r2). Used only when useSp=T.
fSpVRlR: string	: The filename to provide the optimized potential V(R). Used only when useSp=T.
en(i): int	: the "n" parameter for Absorption potential i: i=1: Either the only parameter input if AbsFlag=ABS_ONE or the first of two if AbsFlag=ABS_TWO. i=2: The second parameter input only if Abs_Flag=ABS_TWO
A_0(i): double	: the A_0 parameter for Absorption potential i:
Rabs0(i): double	: The r_{min} parameter for Absorption potential i:
Rabs1(i): double	: The r_{max} parameter for Absorption potential i:
fABS: string	: The filename to store absorption potentials.

Iterate/p_iterate/m_iterate Module (Step III)

This is the breakdown for the input file of the iterate module in the *ScalIT* package. At this stage, it is asssumed we have already used the JA3/JA4 module to create the Hamiltonian.

NOTE: sOSBW:mAlpha, the variable that the documentation says is the beta variable for OSBD, when using OSBW Preconditioning, becomes the number of desired states in the Wyatt Window.

Here is the example of an input file that I have used:

File name:	Here are the actual names of the inputs, followed by what they are:
3 22 24 28	sF sN1 sN2 sN3
F F F	sDep1 sDep2 sDep3
1 4	sJOB sOSB
F T F F	sCX sNDVR sST sAP
1000 1.0D-5 1000 1.0D-3	sBJ1 sBJ2 sQMR1 sQMR2
-0.070 1.0D-5 50 10 200 30 5	SConv:E0 SConv:DE SConv:nStart SConv:nStep SConv:nMax SConv:nE0 SConv:nGap
-0.070 0.001 10.0 100	sOSBW:mE0 sOSBW:mDE sOSBW:mBeta sOSBW:nCnt
0 0 0 0 0	sHOSB sVOSB sHW sVX SPT
∼ /input/h0_og28.dat	fH0
∼ /input/h0gm_og28.dat	(fOUTH)*
	(fRES)
	(fDep(1))
	(fDep(2))
	(fDep(3))
	(fAPP)
	(fAPR)
	(fHOSB)
	(fVOSB)
	(fEig)
	(fHW)
	(fVX)
	(fPT)

NOTE: This is the general format, parenthesis items are optional, starred parenthesis are the ones in the above file. They are in order of how they are read in. This is based on the implementation of *ScalIT* on Robinson as of 10-1-2010, which is based off of the version that we took off the archived file osb_03_06_2006.tar.gz from alpha2 that was used on Jazz cluster.

Definition of variables:

sF: (int)	: Number of layers
sN(1:sF): (int)	: Dimension size of each layer (innermost to outermost)
	note: last is AFTER angular basis truncation.
Dep(1:sF): (bool)	: Whether coordinate dependence exists
sJOB: (int)	: The type of job. Currently only two options:
	1: bound state calculations
	2,3: resonance state calculations
	4,5,6: CRP calculations
sOSB: (int)	: Control paramter for OSB preconditioning
	Choices 1: Uses simple OSB Preconditioner:
	$P = (Eig - E0)^{-1}$
	2: Uses the first OSBD Preconditioner:
	$P = (Eig - E0)^{-1}$ or mDE^{-1}
	3: Uses the second OSBD Preconditioner:
	$P = (Eig - E0)^{-1}$ or $P = [(1-\beta) \star DE + \beta(E - E0)]^{-1}$
	4: Uses the full OSBW Preconditioner:
	$P = (Eig - E0)^{-1}$ or $P = Hp^{-1}$
	BP Comments: 3 above is a more complex version of OSBD,
	with linear + complex functional form.
	ignore mDE value, whereas other choices ignore β value.
sCX: (bool)	: Whether the matrices are complex
	T: if matrices are complex
	F: if matrices are real
sNDVR: (bool)	: Whether DVR is used for outermost layer
	T: DVR is used for outermost layer
	F: normal basis functions are used
sST: (bool)	: Whether the off-diagonal matrices of HOSB are stored in memory.
	T: the off-diagonal elements of HOSB are stored in memory
	F: they are not stored in memory
	NOTE: If sOSB=4, this must be T for posb module (Wenwu)
	Test this!!
sAP: (bool)	: Whether absorption potentials exist.
sBJ1: (int)	: Max # of iterations for Block Jacobi
sBJ2: (double)	: Convergence criteria for Block Jacobi

sQMR1: (int)	: Max # of iterations for QMR
sQMR2: (double)	: Convergence criteria for QMR
sConv::	Convergence parameters for PIST
E0: (double)	: central energy of interested area (atomic units)
DE: (double)	: convergence criteria (atomic units) rms error? worst error?
nStart: (int)	: The original size of PIST matrix
nStep: (int)	: The increment step of PIST matrix for each convergence testing.
nMax: (int)	: The maximum size of PIST matrix
nE0: (int)	: # of interested states half above E0 always?
nGap: (int)	: the size difference of two PIST matrices used for convergence testing
sOSBW::	“Parameters for OSBD and OSBW preconditioners, OSBD, OSBW are applied only for the states whose Eig0 is in the range of $[mE0 - mDE, mE0 + mDE]$. In OSBW, for those states within $[mE0 - mDE, mE0 + mDE]$ ”Wyatt Window”, only the (off-diagonal) matrix elements are calculated for those $\|Eig0(i) - Eig0(j)\| < mDDE$ if sPC=OSBW2. If sPC= OSBW2, all matrix elements are calculated. “cd ho - Wenwu Chen (ScalIT/iterate/osb/doc.txt) BP comments on above: First, we have to figure out which sPC value does what, because there is a typo above. The first sPC value, I think, does COMBINED OSBD and OSBW, with mDE used for OSBD, and mDDE used for OSBW. The second value than does pure OSBW, on a window defined by MDE. This is a guess.
mE0: (double)	: central energy of preconditioner windows (Wyatt window)
mDE: (double)	: width of preconditioner windows (CP Comment: notice plural ”windows”)
mBeta: (double)	: Beta parameter For use with modified OSBD preconditioner: $P = [(1 - \beta) \star DE + \beta(E - E0)]^{-1}$
nCnt: (int)	: Window size for OSBW preconditioner
sHC: (int)	: Control parameter for H⋆X
sPC: (int)	: Control parameter for P⋆X (See “ScalIT/osb/osb/doc.txt”) BP Comments: pure OSBW vs. combined OSBW/OSBD (guess). Probably only used for sOSB=4.

THE OPTIONS OF THE NEXT 5 WILL BE BELOW THEIR DEFINITION:

sHOSB: (int)	: Whether to store HOSB off-diagonal matrices
sVOSB: (int)	: Whether to store: 1.) VOSB transformation matrices 2.) The diagonal elements after Block-Jacobi
sHW: (int)	: Whether to store HW0 and OSBW matrix
sVX: (int)	: Whether to store PIST initial vectors
sPT: (int)	: Whether to store PIST final vectors

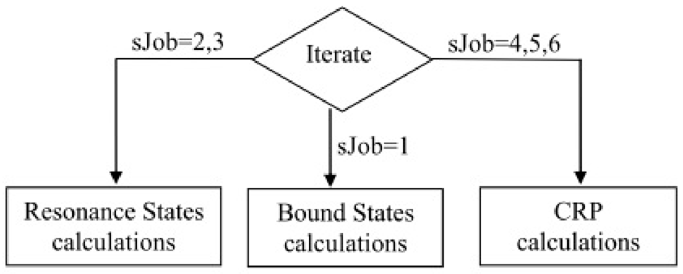
CORRESPONDING DATA FILES:

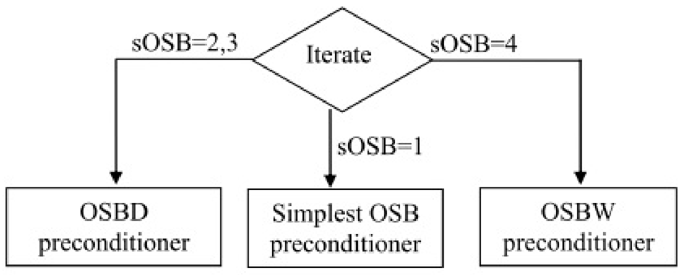
fHOSB: (string)	: sHOSB data file
fVOSB: (string)	: VOSB transforms data file
fEig: (string)	: diagonal elements after Block-Jacobi data file
fHW: (string)	: HW0 and OSBW data file
fVX: (string)	: PIST initial vector data file
fPT: (string)	: PIST final vector data

OPTIONS:
if integer is:

	> 0: store the related data in corresponding file
	$= 0$: don't store or extract the data from files
	< 0: Extract the related data from corresponding files
fDep(i): (string)	: Filename of coordinate dependance data, this should be the output of the previous JA3 module.
fAPP: (string)	: Filename for storage of product absorption potential
fAPR: (string)	: Filename for storage of reactact absorption potential

(SNDVR) “RES” here means ”residual potential”.





wave3jb Module (Step III)

Input file format for wave3jb module
@ read parameters from: ScalIT/wave3/wave3jb.io @
@ print out wave function from: ScalIT/wave3/wave3.io @

```
#!/bin/bash ScalIT/bin/wave3/wave3jb_e < t2.sin > t2.sout
```

An example of input file, this input file is for Ne3 J=0, calculating only the 15th state wavefunction of a previously run calulation. Furthermore, it only plots calculates the R and theta degrees of freedom with $r = r_{eq}$:

File name:	Here are the variable names in their correct location
0 T	JTol Parity
20.0 20.0 20.0	Mass(1) Mass(2) Mass(3)
-1 T F -1	NState gType sType kNum
15	(NSInd(1) NSInd(2) NSInd(i))*
'input/j0jb.dat'	fOut
1 1.22246 1.22246	NR(1) RanMin(1) RanMax(1)
400 0.0 2.5	NR(2) RanMin(2) RanMax(2)
100 0.0 1.0	NR(3) RanMin(3) RanMax(3)
6000 40 'input/vlr.dat'	NMax(1) NS(1) fVlr
6000 60 'input/vBr.dat'	NMax(2) NS(2) fVBr
60 31 'input/hre.dat'	jmax NS(3) fVTh
'input/j0pt.dat'	fVP

Asterisked quantities are situational

int:: JTol	:: Total quantum number J
bool:: parity	:: Parity, true=even parity; false=odd parity
double:: Mass(1:3)	:: Mass of atoms, in (amu)

int:: NState	:: Number of interested states
	if Nstate is negative:
	add asterisked items in formated input file,
	read the indices of —Nstate— interested states:
	read(fd,⋆) (NSInd(ind), ind=1,—NState—)
	if Nstate is positive:
	remove asterisked item in formated input file,
	NSInd(ind)=1, 2, 3,..., NState

bool:: gType	:: T (true) theta in radian; F(false): cos(theta)
bool:: sType	:: T (true) output saved in binary format, F(false) output saved in ascii format, in output file 'fOut' 1st col. 2nd col. 3rd col. 4th col. r R theta wave-function

int:: NSInd(i=1,NState)	:: used only for NState < 0, interested states indices if NState > 0, remove this line
int:: kNum	:: option for choosing which k-value wavefunction to calculate

knum	gType=T	gType=F
-2	calWFP1	calWFP2
-1(default)	calWFS1	calWFS2
0-mMax	calWFM1(kNum)	calWFM2(kNum)

The above are defined as:

calWFP1	: Does something with all k-values (not sure yet)
calWFP2	: Does something with all k-values (not sure yet)
calWFS1	: Sum of all k-values
calWFS2	: Sum of all k-values
calWFM1(kNum)	: kth value wavefunction slice
calWFM2(kNum)	: kth value wavefunction slice
mMax	: IF(JTot $>$ jmax) THEN mMax = jmax ELSE mMax = JTot

NOTE: 1.) picking a kNum outside of the range of mMax will choose the default option (-1)
2.) The value given is the wavefunction, NOT the density.
3.) It may be necessary to divide by the sum of all probabilities to get a normalized quantity
Example: The sum (kNum = -1) of Probability in HO_2 J=110 is 23.813 for all k-slices. Therefore each individual k-slice may need to be divided by that number. C.P.

string:: fOut	:: the file name storing all wave functions
int:: NR(i) i=1,3	:: Grid sizes for r, R, and theta !! Indices: 1–r, 2-R, 3-theta
double:: RanMin(i) i=1,3	:: Starting point for r, R, and theta
double:: RanMmax(i) i=1,3	:: Ending point for r, R, and theta depending on wave function as a function of what two variables of r, R, and theta, set the fixed coordinate at equilibrium value, RanMin(i)=RanMax(i)=equilibrium value, and the corresponding NR(i)=1, where i is index of the fixed coordinate.
int:: NMax(i) i=1,2	:: The maximum number of original sinc DVR points for r, R
int:: NS(i, i=1,2)	:: The final number of basis functions for r, R, which is determined in the Hamiltonian calculation (Second step). Note: if this number is greater than NMAX, then NMAX is taken instead.
int:: jmax	:: The maximum j quantum number,
int:: NS(3)	:: dimension size of angle (theta) this is determined in the Hamiltonian calculation (Second step).
String:: fVlr, fVBr	:: The file names storing all eigenvalues and eigenvectors for r and R fVlr and fVBr are output files of presinc calculation (First step).

String:: fVTh	:: The file name storing all eigenvalues and eigenvectors for theta NOTE: If truncating theta, the 'hre.dat' output file from Hamiltonian calculation is used (Second step). If not truncating theta, then a dummy file is used. I typically use the 'h0gm.dat' output file from the Hamiltonian calculation (Second step) because the 'hre.dat' file is not outputted with no truncation is used.
String:: fVP	:: The file name storing PIST final vectors. fVP is output file of osb calculation (Third step). NOTE: this part of the code cannot be run in parallel if you plan to create wavefunctions. If so, the fVP file is not properly written, more investigation is needed.

******* copy from ver_11_17_2010/ScalIT/wave3/wave3jb.io *********
*** This is the source code that reads in the input file above****

```
READ(fd, *) JTol, parity
READ(fd, *) Mass(1:3)
READ(fd, *) NState, gType, sType, kNum
IF (NState>0) THEN
DO ind=1, NState
NSInd(ind)=ind
END DO
ELSE
NState=-NState
read(fd,*) (NSInd(ind), ind=1,NState)
END IF
READ(fd,*) fOut
DO i=1, 3
READ(fd, *) NR(i), RanMin(i), RanMax(i)
IF (RanMin(i)>RanMax(i)) THEN
tmp=RanMin(i); RanMin(i)=RanMax(i); RanMax(i)=tmp
END IF
END DO
READ(fd,*) NMax(1), NS(1), fVlr
READ(fd,*) NMax(2), NS(2), fVBr
READ(fd,*) jmax, NS(3), fVTh
READ(fd,*) fVP
```