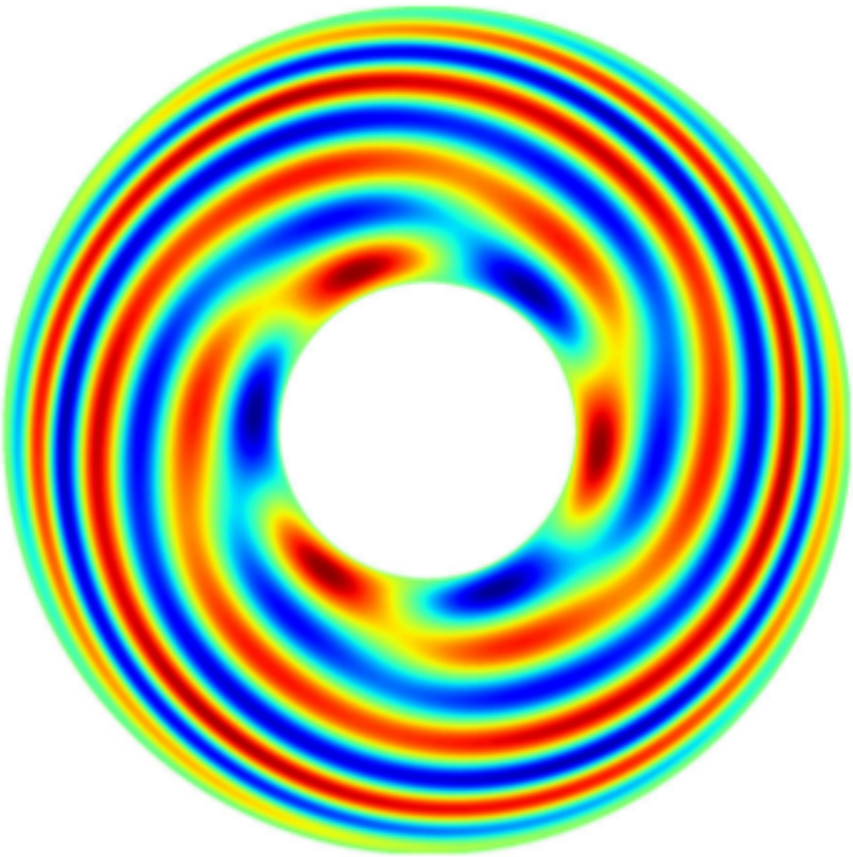
SINGE: User Manual (First Draft)

Jérémie Vidal
ISTerre/UGA

January 12, 2016



Chapter 1

Beginning

1.1 Description

SINGE (Spherical INertia-Gravity Eigenmode) is a code computing linear eigenmodes of incompressible, stratified fluids enclosed in a spherical cavity (full sphere or spherical shell). In addition to the Navier-Stokes equation it takes into account the co-density equation in the Boussinesq framework.

```
#
#                        / _,\
#                        \_\
#             ,,,,    _,_)  #      /)
#            (= =)D__/    __/     //
#           C/^__)/     _(    ___//
#             \_,/  -.    '-._/,--'
#       _\\_,  /            -//.
#        \_ \_/  -,._ _      ) )
#          \/    /    )    / /
#          \-__,/    (    ( (
#                     \.__,-)\_
#                      )\_ / -(
#                     / -(////
#                    ////
# SINGE means monkey in French!
```

SINGE uses finite differences (second order) in the radial direction and a spherical harmonic decomposition (pseudo-spectral). SINGE is written entirely in Python 3[1] with the Numpy/Scipy packages. It uses the Python version of the blazingly fast spherical harmonic transform library SHTns. Finally, it also relies on PETSc/SLEPc libraries (and their Python wrappers petsc4py/slepc4py) for efficiently solving the generalised eigenvalue problem in parallel, using direct or iterative methods. Thus SINGE efficiently runs on your laptop or on parallel clusters.

A post-processing program, based on the one developed in the XSHELLS code, is provided to extract useful data and to export fields to Matplotlib or Paraview.

SINGE is free software, distributed under the CeCILL Licence (compatible with GNU GPL): everybody is free to use, modify and contribute to the code.

[1] Compatible with Python 2.7

1.2 Requirements

1.2.1 For SINGE itself

SINGE should work on any Unix-like system (like GNU/Linux or MacOS X). To run SINGE, you have to to install:

- C++ and Fortran compilers with OpenMP support,
- the PETSc library,
- the SLEPc library,

and the following Python environment

- Python 3 (>= 3.2), including the scientific packages Numpy, Scipy and mpi4py,
- the SHTns library,
- the Python wrappers petsc4py and slepc4py.

Note that SINGE has been written from scratch in Python 3, but it should be compatible with Python (>=2.7) thanks to the imports

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

at the beginning of each *.py file.

Finally, graphical outputs need

- the graphical package matplotlib, or GNU Octave,
- the XSHELLS code (xspp library). See the XSHELLS manual in the same directory for further details. A better solution will be find later...

The following item is not mandatory:

- a shared MPI library (with thread support). You can also let petsc downloading its own MPI library (recommended for new users).

1.3 Installation

Installing the required Python environment should not be difficult on Unix-like systems. For instance for Debian users (also Ubuntu and cie, Linux Mint...)

```
sudo apt-get install python3 python3-numpy python3-scipy
```

To install SHTNS with Python 3, the reader must refer to the webpage https://users.isterre.fr/nschaeff/SHTns/python.html

The tricky part is the installation of PETSc and SLEPc. In all cases, install first PETSc with complex scalars (mandatory because of the eigenvalue problem studied). Then it depends on the architecture of the computer that you are using... Please read the (exhaustive) documentation on the websites of PETSc and SLEPc.

The installation process also depends on the external packages you want to use, such as SUPERLU DIST or MUMPS. Most of the external packages depend on other packages, which must be installed first. Note that some installation configurations are also incompatible, such as MUMPS with 64-bits integers. If you want to install such external libraries, it is recommended for new users to let PETSc downloading and installing them (to avoid any compatibility problem).

Please find below some configuration options which should work on personal laptops:

```
# 32-bits with MUMPS, SuperLu, SuperLu_Dist
./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran
--with-scalar-type=complex --with-mumps=1 --with-superlu=1 --with-superlu_dist=1
--with-fortran-kernels=1 --with-debugging=no --download-mpich
--download-scalapack --download-metis --download-parmetis --download-mumps
--download-superlu --download-superlu_dist

# 32-bits with SuperLu_Dist, SuperLu
./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran
--with-scalar-type=complex --with-superlu=1 --with-superlu_dist=1
--download-mpich --download-scalapack --download-metis --download-parmetis
--download-superlu --download-superlu_dist

# 64-bits with PASTIX
./configure --with-64-bit-indices=1 --with-cc=gcc --with-cxx=g++
--with-fc=gfortran --with-scalar-type=complex --with-ptscotch=1 --with-pastix=1
--download-mpich --download-scalapack --download-metis --download-parmetis
--download-ptscotch --download-pastix

# 64-bits SuperLu_Dist
./configure --with-64-bit-indices=1 --with-cc=gcc --with-cxx=g++
--with-fc=gfortran --with-scalar-type=complex --with-superlu_dist=1
--with-fortran-kernels=1 --with-debugging=no --download-mpich
--download-scalapack --download-metis --download-parmetis  --download-superlu_dist

# SLEPc installation (same for 32 and 64-bits)
# With arpack
./configure --with-arpack
make
make install

# Otherwise
./configure
make
make install
```

Note that the debugging mode was disabled with

```
--with-fortran-kernels=1 --with-debugging=no
```

for faster simulations.

1.4 Configuration file `params.py`

There is a single configuration file, called `params.py`. It is read by (almost) all of the scripts. Beware: the `params.py` file must not be saved in the same directory as the scripts, but in your working directory. Otherwise, the `params.py` located in the scripts' directory will be loaded by default at each simulation, even if SINGE is launched from a different working directory with another `params.py`!

1.5 Running SINGE

There are two running modes in SINGE, one for free-eigenmodes computations and one to compute the onset of linear convection.

1.5.1 Free eigenmodes

Here are the steps the user must follow to compute free eigenmodes with SINGE.

1. Adapt the `params.py` file and copy it in the working directory (different from the directory containing the scripts).
2. Compute the A and B matrix (sequential) with the command

   ```
   python3 path_to_script/singe_matrix_eig.py
   ```

3. Call of the eigenvalue solver in parralel on *nb* mpi process with the command-line `options` (see later)

   ```
   mpiexec -n nb python3 path_to_script/singe_petsc_mpi.py options
   ```

4. Export into xshells format (sequential)

   ```
   python3 path_to_script/singe_to_xshells.py
   ```

1.5.2 Onset of thermal convection

Different from the other case.
Coming soon...

1.5.3 SLEPc options

To call the SLEPc eigensolver, options must be specified by the user, such as the direct or implicit solver used, the linear algebra package used, the nature of the eigenvalue problem... For the moment, only a direct solver using a LU decomposition (sequential or parallel) has been tested. A complete description of these options can be found on the website of SLEPc and in the SLEPc user manual available on the website.

For that solver, `params.py` contains a set of predefined options:

- `nev`: minimal number of eigenvalues to compute,
- `ncv`: proxy for the size of the Krylov subspace. `ncv` $\geq$ `2nev` at least. 10 times is a good value. Note that the higher `ncv` is, the more the problem is memory-bound and the more the cpu time increases.

- `eig`: part of the spectrum to compute.
  - `'LM'`: Largest magnitude,
  - `'SM'`: Smallest magnitude,
  - `'LR'`: Largest real,
  - `'SR'`: Smallest magnitude,
  - `'LI'`: Largest imaginary,
  - `'SI'`: Smallest imaginary,
  - `'TM'`: Target magnitude,
  - `'TR'`: Target real,
  - `'TI'`: Target imaginary.
- τ: the target τ. Must be a complex (e.g. `-1e-3 - 0.015*1j`),
- `tol`: tolerance of reject eigenvalues. $1e-12$ is the default value.
- `maxit`: maximum number of iterations.

> **Example**
>
> ```
> nev   = 10
> ncv   = 100
> eig   = 'TM'
> tau   = -10**(-3) - 0.015*1j
> tol   = 10**(-12)
> maxit = 40
> ```

Then the user must specify some command-line `options`. Here is an example which works pretty well for the case studied in Vidal & Schaeffer (GJI,2015), using the shift and invert method with MUMPS

```
-eps_monitor_conv -eps_balance oneside
-st_type sinvert -st_pc_factor_mat_solver_package mumps
```

or with SUPERLU_DIST (only solver available for 64-bits integers)

```
-eps_monitor_conv -eps_balance oneside
-st_type sinvert -st_pc_factor_mat_solver_package superlu_dist
```

1.6 Outputs

Whatever the file `params.py`, Here are the outputs generated by SINGE:

- `A.mtx` and `B.mtx` are the sparse matrices of the eigenvalue problem

$$A\mathbf{X} = \lambda B\mathbf{X}; \tag{1.1}$$

- `Eigenval.txt`: list of the eigenvalues λ. The first column corresponds to the real part σ and the second to the imaginary part ω;

- `Real_Eigenvec.npy` and `Imag_Eigenvec.npy`: real and imaginary parts of the eigenvectors (temp files);
- `fieldX#.out`: field X (velocity or temperature) corresponding to the # eigenvalue in `Eigenval.txt` (XSHELLS format).

1.7 Citing

If you use SINGE for research work, you must cite (for the moment) the two articles

- J. Vidal & N. Schaeffer, *Quasi-geostrophic modes in the Earth's fluid core with an outer stably stratified layer*, Geophys. J. Int. **202**, 2182–2193, 10.1093/gji/ggv282 (2015)
- N. Schaeffer, *Efficient Spherical Harmonic Transforms aimed at pseudo-spectral numerical simulations*, Geochem. Geophys. Geosyst. **14**, 751-758, doi:10.1002/ggge.20071 (2013)

Chapter 2

Setting up the simulation with `params.py`

The file `params.par` is a Python file used as a parameter file. Example configuration files can be found in the `Examples` directory.

2.1 Modelling

2.1.1 Equations and controlling parameters

SINGE can solves the eigenvalue-eigenmode pairs of the Navier-Stokes equation in a rotating reference frame. Optionally it can include (i) a radial buoyancy force in the Boussinesq approximation, where the buoyancy obeys an advection-diffusion equation. Precisely, the following equations can be solved by SINGE:

$$\lambda \mathbf{u} + 2\,\Omega_0\,(\widehat{\mathbf{z}} \times \mathbf{u}) = -\nabla\pi + \nu\nabla^2\mathbf{u} + Ra\,\theta\,\mathbf{r}, \tag{2.1}$$

$$\lambda\theta + N^2(r)\,u_r = \kappa\nabla^2\theta \tag{2.2}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.3}$$

where

- λ is the eigenvalue (see later);
- $\mathbf{u}$ is the velocity field;
- θ is a codensity (or temperature, or buoyancy) in the Boussinesq formulation;
- ν is the kinematic viscosity of the fluid and is set by the variable `nu` in `params.py`;
- κ is the thermal diffusivity of the fluid and is set by the variable `kappa` in `params.py`;
- Ω_0 is the amplitude of the rotation vector of the rotating reference frame, along the vertical unitary axis $\widehat{\mathbf{z}}$. It is set by the variable `Omega0` in `params.py`;
- $N^2(r) = \frac{1}{r}\frac{\mathrm{d}T_0}{\mathrm{d}r}$ is the Brunt-Väisälä frequency depending on the imposed base temperature profile $T_0(r)$. In `params.py`, it is defined by a function `def function N2r(r):` depending of the radial variable.
- π is the dynamic reduced pressure, which is eliminated by taking the curl of equation (2.1).

Note that it is up to the user to choose dimensional or non-dimensional control parameters. To disable the thermal equation, one must fix `Ra=0` in `params.py`.

The governing equations are recast as an generalised eigenvalue problem

$$\mathcal{A}\mathbf{X} = \lambda\mathcal{B}\mathbf{X}, \tag{2.4}$$

with $\mathcal{A}$ and $\mathcal{B}$ two linear operators and the eigenvalue

$$\lambda = \sigma + \mathrm{i}\,\omega, \tag{2.5}$$

where σ is the damping rate and ω the frequency of the eigenmode.

2.1.2 Internal representation of vector fields

Vector fields are represented internally using a poloidal/toroidal decomposition

$$\mathbf{u} = \nabla \times (T\mathbf{r}) + \nabla \times \nabla \times (P\mathbf{r}), \tag{2.6}$$

where $\mathbf{r}$ is the radial position vector, and T and P are the toroidal and poloidal scalars respectively. This decomposition ensures that the vector field $\mathbf{u}$ is divergence-free.

The scalar fields T and P for each radial shell are then decomposed on the basis of spherical harmonics. For that, the velocity and temperature fields are separated according to their equatorial symmetry:

- equatorially symmetric field such that

$$[u_r, u_\theta, u_\phi]\,(r,\theta,\phi) = [u_r, -u_\theta, u_\phi]\,(r,\pi-\theta,\phi), \tag{2.7}$$
$$\theta(r,\theta,\phi) = \theta(r,\pi-\theta,\phi). \tag{2.8}$$

  This symmetry is enabled with `sym='pos'` in `params.py`.

- antiequatorially symmetric field such that

$$[u_r, u_\theta, u_\phi]\,(r,\theta,\phi) = [-u_r, u_\theta, -u_\phi]\,(r,\pi-\theta,\phi), \tag{2.9}$$
$$\theta(r,\theta,\phi) = -\theta(r,\pi-\theta,\phi). \tag{2.10}$$

  This symmetry is enabled with `sym='neg'` in `params.py`.

2.1.3 Boundary conditions

Temperature (or buoyancy). Boundary conditions are either fixed temperature

$$\theta = 0 \tag{2.11}$$

or or fixed flux

$$\frac{\partial\theta}{\partial r} = 0. \tag{2.12}$$

Velocity. For the full sphere, the velocity field satisfies a regularity boundary condition at the center, imposed on the poloidal and toroidal scalars. At a shell boundary, the velocity field always satisfies the impermeability condition

$$\mathbf{u} \cdot \widehat{\mathbf{r}} = 0, \tag{2.13}$$

where $\widehat{\mathbf{r}}$ is the unitary radial vector. Then, it must satisfy either the stress-free condition

$$\widehat{\mathbf{r}} \times \left(\overline{\overline{\sigma}} \cdot \widehat{\mathbf{r}}\right) = \mathbf{0}, \tag{2.14}$$

where $\overline{\overline{\sigma}}$ is the stress tensor, or the no-slip boundary condition

$$\mathbf{u} \times \widehat{\mathbf{r}} = \mathbf{0}. \tag{2.15}$$

The inner and outer boundary conditions are set in `params.py` and allow to select independently the appropriate boundary conditions:

- `bc_o` and `bc_o` for the velocity field. bc_i = 0 for the full sphere, bc_i = 1 for stress free and bc_i = 2 for no-slip. BC at the outer boundary are similar.
- `bc_i_temp` and `bc_o_temp` for the temperature field. bc_i_temp = 0 for the full sphere, bc_i_temp = 1 for fixed flux and bc_i_temp = 2 for imposed temperature. BC at the outer boundary are similar.

2.2 Spatial discretization

2.2.1 Radial grid

As XSHELLS, SINGE uses second order finite differences in radius. The total number of radial grid intervals (number of points - 1) is defined in `params.py` by the variable `N`. The radial extent of both velocity and temperature fields is set using `r0` and `rf` variables, determing the radius of the first and last shells. The `NR` grid points will be distributed between radii corresponding to the minimum and maximum of these values. `reg` must be equal to `'irreg'` (resp. `'reg'`) for an irregular (resp. a regular) radial mesh.

The irregular grid(recommanded choice) refines the number of points in the boundary layers, and this refinement can be controlled by the two variables `nin` and `nout`, the first and the second being the number of points reserved for the inner and outer boundary layer respectively, reinforcing the normal refinement.

> **Example** The following lines in `params.py` define a grid from 0 to 1, with a total of 241 radial grid points, with 10 and 5 points reserved to the refinement of the inner and outer boundary layer respectively.
>
> ```
> reg   = 'irreg'
> r0    = 0
> rf    = 1
> error = 2
> N     = 240
> nin   = 10
> nout  = 5
> ```

In the case of no-slip boundary condition, it is necessary to have enough points in the boundary layer not to induced a wrong Ekman pumping in the bulk. Following Dormy (PhD

thesis, 1997), 10 points in the BL is a sufficient condition. To ensure it, the user may adjust `nin` and `nout`. A good (maybe too restrictive) condition is to choose them such that the variable `hmin` (which is printed in the terminal) is 10 times smaller than $\sqrt{E}$, where E is the Ekman number.

2.2.2 Angular grid and spherical harmonic truncation

SINGE uses spherical harmonics to represent fields of a given azimutal symmetry m in the spherical volume

$$f(r,\theta,\phi) = \sum_{\ell=m}^{L} f_\ell^m(r) Y_\ell^m(\theta,\phi) \tag{2.16}$$

where Y_ℓ^m is the spherical harmonic of degree ℓ and order m. Numerically, the expansion stores only $m = 0$ and the given m in longitude, with a truncation of the spherical harmonic degree at maximum degree `Lmax`. In `params.py`, `Lmax` is arbitrary but such that `Lmax` $>$ `m`. Note that the true `Lmax` is modified by SINGE and may be `Lmax` $\pm$ 1, in order to have the same number of unknowns for poloidal and toroidal scalars satisfying the equatorial symmetry condition specified by the user.

The angular grid (spanning the co-latitude θ and longitude ϕ) consists of `nphi` regularly spaced points in longitude, and `nlat` gauss nodes in latitude which are internally specified by SHTNS.

Finally, note that only half of the number of spherical harmonic degrees is actually stored, because of the chosen equatorial symmetry.

2.3 Eigenvalue solver

Following Rieutord (1997) and Dintrans & Rieutord (1999), the eigenvalue problem (2.4) is not solved by SINGE. Instead, a spectral transform is applied, namely the shift and invert method. The eigenvalue problem (2.4) is converted into the eigenvalue problem

$$(\mathcal{A} - \tau\mathcal{B})^{-1}\mathcal{B}\mathbf{X} = \theta\mathbf{X}, \tag{2.17}$$

where τ is the target eigenvalue and θ the new eigenvalue linked to λ by

$$\theta = \frac{1}{\lambda - \tau}. \tag{2.18}$$

This transformation is effective for finding eigenvalues near τ, since the eigenvalues θ of the operator that are largest in magnitude correspond to the eigenvalues λ of the original problem that are closest to the shift τ in absolute value. Note that the eigenvectors remain unchanged.

Note that there are other spectral transforms, such as the Cayley transform, which can be used with the corresponding command-line option. Please refer to the SLEPc manual for further details.

Chapter 3

Post-processing with xspp

Fields are stored in binary files (see 1.6), using the custom format developped for XSHELLS. They can be handled after the simulation by the `xspp` command line program.

3.1 Using the xspp command-line tool

Compile the program by typing `make xspp`. Invoking it without arguments (by running `./xspp`) will print a help screen including the commands and their syntax.

> **Example** The following will display information about the file `fieldU.job` (resolution, precision, time of the snapshot, ...):
>
> `./xspp fieldU.job`
>
> To compute the energy and maximum value of the curl of the field:
>
> `./xspp fieldU.job curl nrj max`
>
> To extract the field values along a line spanning the x-axis from $x = -1$ to $x = 0.8$, and also display total energy of field:
>
> `./xspp fieldU.job line -1,0,0 0.8,0,0 nrj`
>
> Add two fields and save the result to a new file (the first file will set the resolution for the result):
>
> `./xspp fieldT_0004.job + fieldT0.job save fieldT_total_0004.job`
>
> Extract only a given range of spherical harmonic coefficients (2 to 31) and computes the corresponding energy:
>
> `./xspp fieldB.job llim 2:31 nrj`

Note that `xspp` is not parallelized using MPI, so that for very big cases you might run out of memory (although it can operate out-of-core – without actually loading the whole file in memory – in some cases). As a workaround you can always reduce the spherical harmonic truncation while reading your big files with the `llim` option (see example avobe).

3.2 Extract and plot 2D slices

One of the most common usage for `xspp` is to extract two-dimensional slices of the 3D data stored in spectral representation in the field files. Four types of 2D slices are available:

- Meridian cuts (a plane containing the z-axis), with the `merid` command;
- Equatorial cuts (the plane $z = 0$), with the `equat` command;
- Surface data (on a sphere of given radius r), with the `surf` command;
- Disc cuts (an arbitrary plane), with the `disc` command;

When these commands are given to `xspp`, it will write text files corresponding to the required cuts. They can then be loaded and displayed using matlab, octave or python with matplotlib (see next sections).

> **Example** A meridian cut at $\phi = 0$:
>
> `./xspp fieldU.job merid`
>
> An equatorial cut, and a meridian cut at $\phi = 45$degrees, of the vorticity (curl of U)
>
> `./xspp fieldU.job curl equat merid 45`
>
> Extract the field at the spherical surface closest to $r = 0.9$, using only the symmetric components.
>
> `./xspp fieldU.job sym 0 surf 0.9`
>
> Make a cut at $z = 0.7$, using 200 azimuthal points, with field truncated at harmonic degree 60:
>
> `./xspp fieldU.job llim 0:60 disc 200 0,0,0.7`

3.2.1 plotting with matlab/octave

Matlab or Octave scripts are located in the `matlab` dierectory. There are scripts to load and plot cuts obtained with `xspp`.

> **Example** Produce a meridian cut with `xspp`:
>
> `./xspp fieldU.job merid`
>
> Then, from octave (or matlab), load and plot the ϕ-component of the field in this meridional slice:
>
> ```
> > [Vphi,r,theta] = load_merid('o_Vp.0');
> > plot_merid(r,theta,Vphi)
> ```

3.2.2 plotting with python/matplotlib

The python module `xsplot` is provided to load and display cuts produced by `xspp`. It can be used interactively or within scripts. Such Python scripts using matplotlib and xsplot are located in the `matplotlib` dierectory, and can be called from command line. `xsplot` can also be used directly from command line and will guess the type of cut of your file and display it accordingly.

The python module should be installed by calling `make install`, or explicitly `python setup.py install -user` from the `python` subdirectory, to install it for the current user only.

> **Example** Produce a meridian and an equatorial cut with `xspp`:
>
> ```
> ./xspp fieldU.job merid equat
> ```
>
> From command prompt, quickly load and plot all three components in cylindrical coordinates of the field in this meridional slice, as well as in the equatorial plane.
>
> ```
> xsplot o_Vs.0 o_Vp.0 o_Vz.0 o_equat
> ```
>
> Alternatively, from an Ipython interpreter (or notebook, or script), load and plot the ϕ-component of the field in the meridional and equatorial slices:
>
> ```
> > import xsplot
> > r,theta,Vphi = xsplot.load_merid('o_Vp.0')
> > xsplot.plot_merid(r,theta,Vphi)
> > s,phi,Vs,Vp,Vz = xsplot.load_disc('o_equat')
> > xsplot.plot_disc(s,phi, Vp)
> ```

3.3 3D visualization with paraview

From the paraview website: *ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques.*

Full spatial fields can be saved to XDMF format, which can be loaded by paraview. Note that the HDF5 library is required for this to work, and must be found by the `configure` script. If so, Simply run:

```
make xspp
./xspp fieldB_0004.job hdf5 B_cartesian.h5
```

The file `B_cartesian.h5.xdmf` describes the cartesian components of the vector field B on a spherical grid that can be read directly by paraview (if prompted for a loader, select 'XDMF').

3.4 Advanced post-processing using pyxshells

For more complex post-processing, xspp may not be enough. The python module `pyxshells` allows you to quickly write your own scripts to work directly with the spectral fields stored in the `field` files output by SINGE, cast them to spatial domain, and so on...