Appendix B

GOFISH Documentation

B.1 Overview

GOFISH is an external minimizer for the Coulomb excitation analysis code GOSIA which implements Particle Swarm Optimization (PSO) [65]. PSO is a metaheuristic optimization scheme, meaning that it optimizes without regard to the gradient of the function being minimized. GOSIA minimizations often have many highly correlated parameters and local minima, which limits the effectiveness of gradient-based optimizers. This led to the desire to develop a non-gradient-based optimizer which could speed up and improve the analysis for Coulomb excitation experiments. Particle Swarm Optimization was chosen because it is highly robust to high-dimensional search spaces with many local optima, and because it can be very effectively parallelized to take advantage of high-performance computing. This appendix focuses on documentation of the GOFISH code. For more information about the use of the GOFISH code in this work, see chapter 5. The GOFISH code is hosted at `https://github.com/avamh672/gofish`.

GOFISH begins by initializing a number of "particles" randomly in the parameter space, and giving them random initial velocities in this space based on the user-defined bounds for each parameter. Reasonable bounds for all matrix elements, and particularly buffer state matrix elements, are important to achieving a good result. After the particles are initialized, GOSIA's **OP,POIN** is evaluated for each particle to compute the yields

associated with that particle's matrix elements, and a particle χ^2 value is computed by comparing the simulated yields with the experimentally observed ones. The particles' velocities are then updated based on equation 5.1. The default values of the inertial, cognitive, and social coefficients (0.7, 1.75, and 1.0, respectively) were selected based on the results of a hyperparameter search. The user may, however, specify alternative values for these and many other parameters in the config file. The topology used in GOFISH is adaptive; it starts with an *lbest* topology with 20 neighbors, and adds more connections as the number of iterations increases until a *gbest* topology is achieved. By default GOFISH performs 500 iterations of particle swarm optimization, though this number can also be specified by the user. When the maximum number of iterations is reached, it writes the best position it found and the χ^2 value associated with that position to an output file.

B.2 File Preparation

GOFISH runs with mostly standard GOSIA input files, with the exception of a few comments which must be added to make the file more easily parsed. Complete GOSIA input files executing the **OP,INTI** and **OP,POIN** commands should be generated. More information on how to generate the GOSIA input files can be found in the GOSIA user manual [45]. If simultaneous beam-target minimization is to be done, input files are required for both the beam and target nuclei. Note that in the case of simultaneous minimization, the files should still be prepared as GOSIA input files, not GOSIA2 input files. You will also need to include all of the other files required to run those GOSIA input files; it should be possible to run GOSIA on each of the input files in your working directory. Additionally, GOFISH requires the inclusion of a config file and a specially-formatted yields file (or files, in the case of simultaneous minimization) which list all of the transitions being simulated. Examples of all required files are included in the example directory of the Github repository for GOFISH [64].

B.2.1 The GOSIA Input File

In order to reduce the number of parameters which must be specified by the user, GOFISH uses the gosiaManager class to read the normalizing transition and upper limit, the literature constraints, and the relative scaling factors (if applicable) from the GOSIA input file or files. By flagging the relevant lines with comments, we can greatly simplify the functions required to parse the input file.

The following six lines, all in the **OP,YIEL** section, must be commented in each **OP,POIN** input file:

NS1,NS2 - This line must be commented with "!NS". GOFISH is not currently set up to handle different upper limit ratios or normalizing transitions for different experiments, so the input file should be prepared with only a single instance of these parameters.

UPL - This line must be commented with "!UPL".

NBRA - This line must be commented with "!BR".

NL - This line must be commented with "!NL".

NDL - This line must be commented with "!DL".

NAMX - This line must be commented with "!ME".

.

Additionally, if you doing beam- (or target-) only minimization, you will need to comment each line containing the variable **YNRM** with "!SCL", even for experiments which use independent scaling. The normalizing experiment numbers are read from the **EXPT** section of the GOSIA input file, and do not require any comments.

B.2.2 The Yields File

In addition to the standard yields file read by GOSIA, GOFISH requires a yields file which lists the yields for every transition associated with the matrix elements in the GOSIA input. This yields file should be in the same format as the yields file described in section 7.33.1 of the GOSIA user manual [45], with the following modifications:

This yields file must include unobserved transitions whose matrix elements are included in the GOSIA input, in addition to the observed transitions. For unobserved transitions, the counts and uncertainty should be recorded as 0. GOFISH will interpret this as an unobserved transition and include it in the χ^2 calculation only if the computed yield exceeds the user-defined upper limit.

Transitions must be listed in the order in which they appear in the **OP,POIN** output.

Unresolved doublets must be formatted as described in section 7.33.1 of the GOSIA user manual [45]. They must be listed in the yields file in the place of the first of the two transitions to appear in the **OP,POIN** output. Additionally, using the same variable names as in the user manual, the first transition to appear in the **OP,POIN** output must be denoted by II1 and IF1, and the second by II2 and IF2. The sum of the two transitions' yields will be treated as a single observable when calculating the χ^2 in GOFISH, as they would be in GOSIA.

B.2.3 The Config File

GOFISH has many parameters which can be specified by the user in the config file which is passed to the program at startup. There are both mandatory parameters, which must be specified by the user or the program will raise an exception and terminate, and optional parameters, which have default values which may be overridden by the user. An example config file is shown in Figure B.1.

```
gosia = /mnt/home/hillma54/gosia/gosia
beamINTIinp = sn116_gos2.INTI.inp
beamPOINinp = sn116_gos2.inp
beamYields = sn116yields.yld
nBeamParams = 9
nThreads = 21
scratchDirectory = /mnt/scratch/hillma54/
simulMin = True
targetINTIinp = pt196_gos2.INTI.inp
targetPOINinp = pt196_gos2.inp
targetYields = pt196yields.yld
nTargetParams = 22
nParticles = 600
nIterations = 500
inertialCoeff = 0.7
cognitiveCoeff = 1.75
socialCoeff = 1.0
velocityStrategy = invert
boundaryStrategy = reflective
parallelAnnealing = True
annealinglayers = 5
annealingRates = 0.3,0.4,0.5,0.6,0.7
crossoverIterations = 100,200,300,400
```

Figure B.1: An example config file for GOFISH.

B.2.3.1 Mandatory Parameters

The following parameters must be included in the config file. Despite the naming convention of beam and target for the two nuclei, GOFISH makes no assumption about the kinematics of the problem. If two nuclei are being minimized simultaneously, the labels are arbitrary and either nucleus can be designated as beam or target without affecting the outcome of the fit. In the case where only a single nucleus is being fit, the beam variables should be used and the target ones not included, even if the nucleus being minimized is actually the target.

gosia - The path to the GOSIA executable. Note that there is a slightly modified version of GOSIA included in the Github repository for GOFISH which includes some additional information in the output file. This version of GOSIA should be used.

beamINTIinp - The path to a GOSIA input file for the first (or only) nucleus which executes **OP,INTI**.

beamPOINinp - The path to a GOSIA input file for the first which executes **OP,POIN**. Note that certain modifications must be made to this file, specified in Sec. B.2.

beamYields - The path to the GOSIA input file which contains the yields for the first nucleus. Note that this file must be formatted in a particular way, specified in the section on file preparation.

nBeamParams - Integer, must be set to the number of matrix elements to be minimized for the first nucleus.

nThreads - Integer, must be set to the number of threads to be used by the program. Note that since this program uses mpi4py, the number of threads specified here must match the number passed to the message passing interface at runtime. GOFISH uses a single director thread and uses the remaining threads to evaluate candidate solutions. It is therefore recommended that the number of evaluator threads divides evenly into the number of particles.

scratchDirectory - The path to the folder where GOFISH will generate subdirectories for running GOSIA in parallel. This folder can be empty; GOFISH will create the required subdirectories and copy over the necessary files.

simulMin - Boolean, must be set to either True or False. If set to True, GOFISH will perform a simultaneous minimization of the projectile and target matrix elements. If set to False, GOFISH will perform the minimization only for one set of matrix elements. Note that if simulMin is True, there are four additional parameters for the target which must be included in the config file.

targetINTIinp - The path to a GOSIA input file for the second nucleus which executes **OP,INTI**. This input is required only if simulMin is True.

targetPOINinp - The path to a GOSIA input file for the second nucleus which executes **OP,POIN**. Note that certain modifications must be made to this file, specified in Sec. B.2. This input is required only if simulMin is True.

targetYields - The path to the GOSIA input file which contains the yields for the second nucleus. Note that this file must be formatted in a particular way, specified in the section on file preparation. This input is required only if simulMin is True.

nTargetParams - Integer, must be set to the number of matrix elements to be minimized for the second nucleus. This input is required only if simulMin is True.

B.2.3.2 Optional Parameters

The following parameters may be optionally included in the config file. If they are not included, they will be set to the default values specified below.

nParticles - Integer, set to the number of particles to be included in the optimization. The default value is 600.

nIterations - Integer, set to the number of iterations for the optimization. The default

value is 500.

inertialCoeff - Float, set to the inertial coefficient for PSO. The default value is 0.7. This parameter is not used if parallelAnnealing is set to True.

cognitiveCoeff - Float, set to the cognitive coefficient for PSO. The default value is 1.75.

socialCoeff - Float, set to the social coefficient for PSO. The default value is 1.0.

velocityStrategy - String, sets the method for correcting the velocities of particles which leave the parameter boundaries. This parameter may be set to any strategy accepted for the PySwarms [112] velocityHandler class. The default value is "invert", which inverts and halves the vector each component of the velocity where the corresponding position vector component is outside the parameter bounds. Other options can be found in the PySwarms documentation.

boundaryStrategy - String, sets the method for correcting the positions of particles which leave the parameter boundaries. This parameter may be set to any strategy accepted for the PySwarms [112] boundaryHandler class. The default value is "reflective", which reflects the particles which exceed the parameter bounds. Other options can be found in the PySwarms documentation.

topology - String, sets the graph topology to be used for the particles. The default value is "star", which is the *gbest* topology. The other option implemented is "dynamic", which begins as a ring topology which slowly adds connections as the number of iterations increases until the graph is fully connected. The star topology converges faster than the dynamic topology, but the dynamic topology is less likely to prematurely converge to local optima. The parameters of the dynamic topology can be specified by the nNeighbors

and topologySwitchIterations parameters.

nNeighbors - Integer or list of integers, used to specify the number of neighbors to be used for the topology at various points of the optimization. Only used if topology is set to "dynamic". The default array is [20,30,40,60,80,120,160]. If the number of entries in this list is equal to the number of topologySwitchIterations, the topology will switch to the star topology at the last switch iteration. If only a single integer is passed here, and topologySwitchIterations is not included in the config file, GOFISH will use a ring topology with the specified number of numbers for the entire optimization.

topologySwitchIterations - Integer or list of integers, used to specify the iteration numbers at which the topology changes. Only used if the topology is set to "dynamic". The default array is [50,100,150,200,250,300,350]. The default array for this parameter has the same length as the default array for nNeighbors, so if you use the default values for both, the final topology, starting at the 350^{th} iteration, will be the star topology. If you wish to use a ring topology which does not increase the number of connections, do not include this parameter in the config file and pass only a single integer for nNeighbors.

parallelAnnealing - Boolean, enables the parallel annealing option described in Sec. B.5. Note that this option is significantly more computationally expensive and will drastically increase the run time of the program. Use of this feature was not found to improve performance, so we recommend that this parameterbe set to its default value of False. However, it remains implemented in the code in case a future user would like to expand on the concept.

annealingLayers - Integer, set to the number of layers to be used for parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is 5.

annealingRates - List of floats, set to the different inertial coefficients to be used for

the different layers in parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is [0.3,0.4,0.5,0.6,0.7].

crossoverIterations - List of integers, set to the iteration numbers at which layers will be swapped during parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is [100,200,300,400].

beamMultipletStates - List of integers, used to declare unresolved multiplets for beam transitions. Observables associated with the multiplet will be summed to produce a single observable for the χ^2 calculation. The default value for this parameter is an empty list. For information on how to format this input, see Sec. B.6.

targetMultipletStates - List of integers, used to declare unresolved multiplets for target transitions. Observables associated with the multiplet will be summed to produce a single observable for the χ^2 calculation. The default value for this parameter is an empty list. For information on how to format this input, see Sec. B.6.

B.3 Running GOFISH

The GOFISH code relies on GOSIA for evaluating the candidate solutions generated by particle swarm optimization. Candidate solutions are evaluated by setting all matrix elements to the candidate values and then using GOSIA's **OP,POIN** to estimate the yields associated with those matrix elements. The yields are then compared to the experimental yields to generate a χ^2 value. Literature constraints, unresolved doublets, and unobserved transitions are included in the χ^2 in the same way as in GOSIA. Additionally, GOFISH allows for the declaration of unresolved multiplets consisting of up to six transitions. The transitions associated with the multiplet will be summed to produce a single observable for the purpose of computing the χ^2. Instructions on how to declare unresolved multiplets are outlined in Sec. B.6.

In order for GOFISH to run, all of the required inputs for GOSIA must be present

run directory, and all files must be formatted as described in the Sec. B.2. Prior to running GOFISH, you will also need to run the GOSIA **OP,INTI** input file (or files, for simultaneous minimization) to generate the "corrected" yields file (rather than correct the computed yields for the effect of the spline integration, GOSIA applies a correction to the experimental yields). Like GOSIA, GOFISH uses the corrected yields values for comparisons with the output of **OP,POIN**.

GOFISH handles multithreading through the Python library mpi4py [68, 113, 114, 115], which provides bindings for the Message Passing Interface (MPI) standard. To run GOFISH, execute the following command, replacing the variables in {} with the appropriate values:

mpirun -n {nThreads} python gofish.py {configFile} {batchNumber}

nThreads - The total number of threads to be used. This must match the number of threads specified in the config file. GOFISH reserves one thread as the director that executes the optimization. The other threads are used for evaluating the candidate solutions. It is recommended that the number of evaluator threads divides evenly into the number of particles, but this is not required. GOFISH will divide the particles as evenly as possible between the threads.

configFile - The path to the config file. The format of this file is described in the section on file preparation.

batchNumber - An integer, used only for naming the subdirectories created by GOFISH. If you are running multiple instances of GOFISH simultaneously, they must have different batch numbers so that they do not overwrite each other's files.

Particle swarm optimization is a stochastic algorithm and convergence is not guaranteed. As such, it is advisable to run multiple instances of GOFISH simultaneously to

improve the odds of finding the global minimum. In the work presented in this thesis, GOFISH optimizations were run in batches of 20. Our observed rate of convergence to the apparent global minimum is approximately 90%, so it is feasible to run with smaller batches if computationally limited. GOFISH is designed with batch submission in mind, and multiple instances of the code running in the same directory will not conflict with each other as long as the provided batch numbers are different. You must ensure that this is the case; if two instances of GOFISH are run with the same batch number at the same time, even if they are run from different directories, they may overwrite each other's temporary files.

When the GOFISH program terminates, it will produce a file called particleSwarmResult_{batchNumber}.csv which contains the best fit χ^2 value and matrix elements. The matrix elements will be listed in the order they are declared in the GOSIA input files, with the "beam" matrix elements (in the GOFISH naming convention) listed before the "target" ones. Because GOFISH relies on the output of **OP,INTI** to correct the measured γ-ray yields, the result is likely to be inaccurate unless your initial guess of the matrix elements was close to the optimal values. To account for this, we recommend running GOFISH several times in succession, each time using the best fit matrix elements from the previous fit for the next **OP,INTI** calculation. This process should be repeated until the solution converges. When the statistical variation within a batch is comparable to the variation between batches, the solution is ready for polishing. In the analysis of the ^{112,116,120}Sn Coulomb excitation, this step was performed three to four times for each isotope.

Finally, it is recommended that solution polishing be performed on the final result. While gradient-based optimizers often struggle in search spaces with many local optima, they are quite good at fine-tuning the solutions produced by other algorithms. We recommend that you employ a gradient-based optimizer (either the one built into GOSIA or another external minimizer such as GOSIAFitter [69]) to polish the solution by using the GOFISH result as the initial guess for the gradient-based optimization and allowing only the matrix elements directly tied to observables to vary. The polished solution should be

very close to the GOFISH result; if it is not, you may need to run more iterations of the **OP,INTI** → GOFISH loop.

B.4 Checkpointing

The GOFISH code takes on the order of hours to run, even when utilizing a large amount of computational resources. To make the most efficient use of these resources, GOFISH employs a checkpoint system which writes all relevant program information to a series of files at the end of each iteration. The files will be stored in a folder called checkpoint_{batchNumber} which will be located in the GOFISH run directory. When a new instance of GOFISH is started, it will automatically check for the existence of a checkpoint directory with the correct batch number. If it exists, the program will read in the checkpointed information and continue the optimization from that point rather than beginning from scratch. If you wish to start a run from scratch, you must first ensure that no such checkpoint directory exists.

The checkpointing system was developed specifically for running on the scavenger queue at the Institute for Cyber-Enabled Research (ICER) at Michgan State University, although it works for jobs which are killed for any reason. The scavenger queue is a low-priority queue which submits jobs to otherwise unoccupied nodes, and cancels those jobs if a higher priority job requests those nodes. Cancelled jobs from the scavenger queue are automatically restarted when the resources become available again. Implementation of the checkpointing system allowed us to make highly efficient use of the scavenger queue by allowing cancelled jobs to be restarted with minimal loss of progress.

B.5 Parallel Annealing

GOFISH has an optional run mode which theoretically could be used to improve the quality of the fit, at the cost of increased computational time. In this mode, multiple swarms are created with different inertial coefficients. At predetermined points during the optimization, the swarms are swapped between layers, changing the inertial coefficient for

that swarm. In practice, this mode was found to significantly increase computational time (since more candidate solutions must be evaluated) with no improvement in convergence speed or solution quality. It is thus recommended that this feature be disabled by setting the parallelAnnealing parameter in the config file to False. This feature was included in the release of GOFISH in case a future user wishes to expand on this concept.

B.6 Unresolved Multiplets

GOSIA allows the user to define pairs of states as unresolved doublets. The observables for each transition in the doublet are summed together to produce a single observable for the purpose of computing the χ^2. GOSIA does not support the inclusion of multiplets with more than two transitions. However, since the declaration of a multiplet only affects the χ^2 calculation, and since GOFISH calculates its own χ^2, GOFISH is able to support the inclusion of unresolved multiplet states consisting of up to six transitions. If an unresolved multiplet is declared, all transitions in the multiplet will have their observables summed together to produce a single observable in the χ^2 calculation.

Unresolved multiplets are declared in the GOFISH config file. The GOSIA input files should be prepared as normal, treating the component transitions of the multiplet as individual observables. The config file has two optional inputs, beamMultipletStates and targetMultipletStates, which are used to declare multiplets.

The beamMultipletStates and targetMultipletStates take lists of integers as input. For an unresolved multiplet consisting of the $II1 \rightarrow IF1$, $II2 \rightarrow IF2$, $II3 \rightarrow IF3$, $II4 \rightarrow IF4$, $II5 \rightarrow IF5$, and $II6 \rightarrow IF6$ transitions, the input format is $MI1, MF1$, where

$$MI1 = 10000000000 * II6 + 100000000 * II5 + 1000000 * II4 + 10000 * II3 + 100 * II2 + II1$$

$$MF1 = 10000000000 * IF6 + 100000000 * IF5 + 1000000 * IF4 + 10000 * IF3 + 100 * IF2 + IF1 \tag{B.1}$$

In this equations, IIX and IFX are the initial and final level indices for transition X. For

multiplets with fewer than six transitions, the indices for nonexistent transitions should be taken to be 0. Thus for an unresolved triplet, the formulas would reduce to

$$MI1 = 10000 * II3 + 100 * I2 + II1 \tag{B.2}$$

$$MF1 = 10000 * IF3 + 100 * IF2 + IF1 \tag{B.3}$$

This format was selected to match the format GOSIA employs for unresolved doublets. If more than one multiplet is to be included, the input for beamMultipletStates or targetMultipletStates should take the form $MI1, MF1, MI2, MF2, MI3, MF3, ...,$ where $MI1$ and $MF1$ correspond to the first multiplet, $MI2$ and $MF2$ correspond to the second multiplet, and so on. The ordering of transitions within a multiplet is arbitrary; swapping $II1$ with $II2$ and $IF1$ with $IF2$ will not affect the results.

The GOFISH yields file should be generated as if the transitions are resolved separately. The yields and uncertainties for each transition in the multiplet will be summed to create a single observable. It does not matter how the observed counts are distributed among the individual transitions, so for simplicity, it is recommended that for each GOSIA experiment, one transition in the multiplet declare the total yield and uncertainty of the unresolved feature for that experiment, and each other transition in the multiplet declare a yield and uncertainty of 0.

B.7 The GOSIAManager Class

The GOSIAManager class serves as the interface between GOSIA and GOFISH. GOSIA solves the coupled differential equations for Coulomb excitation to evaluate the candidate solutions, while GOFISH performs the optimization. The functions in the GOSIAmanager class allow us to parse the GOSIA input and output files to get information, and to create/modify the inputs for evaluating the candidate solutions. The available functions are as follows:

__init__ - The class initialization function takes the config file, detailed in the previous section, as an input. It parses the config file and assigns the user-specified parameters to the proper variables. It then parses the beamINTIinp and beamPOINinp files (and the target versions, if applicable) to get the names of several additional GOSIA input/output files. It also checks if all required arguments are included, and will raise an exception if they are not.

getMatrixElements - This function takes a GOSIA input file as the only argument. It will parse the given file and return a pandas [116, 117] dataframe containing the matrix elements listed in that file, with entries for multipolarity, inital and final states, and the upper and lower bounds assigned to that matrix element.

make_bst - This function creates the file from which GOSIA reads the matrix elements when OP,REST is called. It takes as arguments the name of the file and a list of the matrix element values, in the order they appear in the file. The name specified must match the name in the OP,FILE section of the relevant GOSIA input file.

read_bst - This function reads the file from which GOSIA reads the matrix elements when OP,REST is called. It takes the file name is an input and returns a list of the matrix elements.

getExperimentalObservables - This function populates four arrays and four dictionaries with information about the experimental observables by parsing the following files, which must be defined in the config: beamYields, beamINTIinp, and beamPOINinp. Additionally, if simulMin is set to True, it will also parse targetYields, targetINTIinp, and targetPOINinp. The four populated arrays are observables, uncertainties, beamExptMap, and targetExptMap, and the dictionaries are beamDoubletMap, targetDoubletMap, beamMultipletMap, and targetMultipletMap. The observables array contains the values for each experimental observable and literature constraint, and the uncertain-

ties array contains the corresponding uncertainties. The ordering of these arrays will be first the beam observables (in the order they appear in the GOSIA output file), followed by beam literature constraints, and then target observables and literature constraints if applicable. The beamExptMap and targetExptMap arrays are lists of tuples, with each tuple containing three values: the experiment number (as it appears in GOSIA) and the initial and final state numbers for the associated observable. For literature constraints, the tuple value is (0,0,0). The experiment maps are constructed from the beamYields and targetYields files, which must be formatted in a particular way, described in the section on file preparation. The beamDoubletMap and targetDoubletMap dictionaries store information about any unresolved doublets in the experimental observables. If no doublets were observed, the dictionaries will be empty. Otherwise, for each doublet, the key will be a tuple containing the initial and final states for the doublet transition associated with GOSIA variables II2 and IF2. The associated value will then be a tuple containing the "combined" initial and final states of the doublet, matching the format of the yields file. The beamMultipletMap and targetMultipletMap work similarly and contain information on unresolved multiplets with more than two transitions. Since this feature is not supported by GOSIA, these multiplets must be declared using the beamMultipletStates and targetMultipletStates parameters in the config file. For instructions on how to format these parameters, see Sec. B.6. All observables and uncertainties are read from the corrected yields file. The name of the corrected yields file is not a required input to this program. Instead it will be read from the OP,FILE section of the beamINTIinp (or targetINTIinp) file. Literature constrains are parsed from the beamPOINinp and targetPOINinp files. Parsing the constraints requires the inclusion of comments on specific lines in the GOSIA input files, described in Sec. B.2.

getPOINobservables - This function takes as arguments a GOSIA OP,POIN output file, the associated exptMap, and the associated doubletMap and multipletMap. It parses the GOSIA output file and returns the computed observables for the input matrix elements. The computed observables will be a list in the same format and order as the

observables array returned by getExperimentalObservables. Transition pairs which are included as unresolved doublets in the doubletMap or unresolved multiplets in the multipletMap will have their yields combined into a single observable.

getUpperLimits - This function takes as arguments the beamExptMap, targetExptMap (even if empty), and the observables array, and returns an upper limit for each unobserved transition included in the optimization. It does this by parsing beamPOINinp (and targetPOINinp, if applicable) and reading the upper limit settings specified in the file. Parsing this information requires the inclusion of comments on specific lines in the GOSIA input file(s), described in the section on file preparation.

createSubDirectories - This function creates a sub-directory in the user-defined scratch directory and copies all of the files required for executing the GOSIA **OP,POIN** input file. This is done so that multiple instances of GOSIA can be run simultaneously, in different directories so that they are not writing to the same output files. This function requires an integer argument, which will be used in the naming of the subdirectory.

removeSubDirectories - This function is used for removing a sub-directory, and allows GOFISH to clean up its temporary files when they are no longer needed. It requires an integer argument which matches the integer used to create the subdirectory in the first place.

runGosia - This function instantiates a subprocess to run GOSIA in the current working directory. The thread will wait for the subprocess to terminate before executing any more lines of code. This function requires the GOSIA input file to be passed as an input.

runGosiaInDir - This function instantiates a subprocess to run GOSIA in a user-specified directory. As with runGosia, the thread will wait for the subprocess to terminate

before executing any more lines of code. The function requires the GOSIA input file and the path to the run directory to be passed as input.

getScalingFactors - This function parses the user-defined scaling factors from the GOSIA **OP,POIN** input file. It does not require any arguments. It will return two lists, the first containing the normalizing experiment number for each experiment (the value of the variable LN from GOSIA's EXPT section) and the second containing the normalization constants for each experiment (the variable YNRM from GOSIA's **OP,YIEL** section). The latter requires comments on specific lines of the GOSIA input file, described in the section on file preparation. This function is only called if simulMin is false. If it is true, scaling factors are instead constrained by the second set of yields.

getAverageAngle - This function parses the average angle for each experiment from the **EXPT** section of the GOSIA **OP,POIN** input file. It does not require any arguments. This information is used to apply corrections to the scaling factors produced by the getScalingFactors function so that we match the calculations done by GOSIA.

getThetaCM - This function computes the center of mass scattering angle from the experimental parameters and a LAB frame particle scattering angle. It takes as arguments the LAB frame scattering angle, the beam mass number, the target mass number, the beam energy, and the excitation energy (taken to be the energy of the first excited state). It is used by the getDsig function.

getRecoilThetaLAB - This function computes the target recoil angle from the center of mass scattering angle and experimental parameters. It takes as arguments the center of mass scattering angle, the beam mass number, the target mass number, the beam energy, and the excitation energy. It is used by the getDsig function.

getDsig - This function takes no arguments and returns a list of the DSIG values for

each experiment. The DSIG parameter is used by GOSIA for calculating scaling factors for coupled experiments. The parameter is calculated and used in the same way as in GOSIA.

B.8 Required Libraries

The following libraries must be installed for GOFISH:

mpi4py [68, 113, 114, 115]

numpy [118]

pandas [116, 117]

PySwarms [112]

GOFISH also requires the following four libraries which are included in the Python standard library:

os

shutil

subprocess

sys