Developer's Guide for PINSPEC

April 17, 2013

1 Introduction

1.1 Intro to Developer's Guide

This document is meant to provide information in addition to the user guide hosted on GitHub's wiki pages. A number of instructions on how to install the required packages and PINSPEC, how to set up the python input files, and how to run the code can be found on the above link. Some of these info may be repeated in this document, but as a general rule of thumb users should refer to the wiki page for instructions related to using PINSPEC, and this document is reserved for info on developing PINSPEC.

While the rest of this section would provide a high-level introduction of how PINSPEC source codes are constructed, the next couple of sections in this document would be devoted to:

- Review some essential instructions for installing PINSPEC and running PINSPEC, see Section 2.
- Discuss how to manipulate the python codes, see Section 3.
- Discuss how to manipulate the C++ codes, see Section 4.

Keep in mind that the above info is in addition to the commenting in the source codes and Doxygen-generated files as hosted here: Doxygen generated documents.

1.2 Python to SWIG to C++

Fig. 1 illustrates the three major components of PINSPEC: the python codes, the SWIG interface, and the C++ source codes.

Simplified Wrapper and Interface Generator (SWIG) is open source software used in PINSPEC to wrap C++ functions for use with python. More generally, SWIG can be used to connect C/C++ with scripting languages such as Lua, Perl, PHP, Python, R, Ruby, Tcl and even non-scripting languages like C#, Java, JavaScript, Go, Modula-3, Ocaml, Octave, and Scheme[1].

More specifically, here is the flow of a typical simulation:

- Users inputs data in a python file;
- PINSPEC python source codes process input data, perform Doppler Broadening of the cross sections if requested;
- SWIG registers the data in C++ classes;
- C++ contains the actualy Monte Carlo kernel, and neutrons are simulated here;
- If any plotting is requested, generated results would be passed from C++ back to python using SWIG again and python would generate plots.

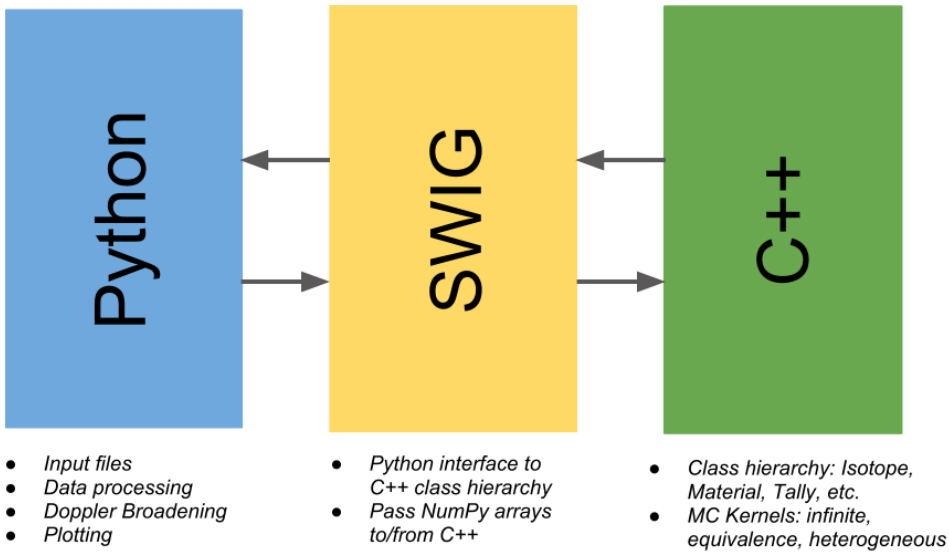

Figure 1: Python Interfacing C++ Through SWIG

[1]Reference: SWIG's wiki page, SWIG's homepage.

1.3 Major C++ Classes and Structures

C++ using the following major classes and structures:

1. Geometry: the umbrella object that knows the # neutrons per batch, # batches, # threads, the type of geometry etc. Geometry contains region, which contains material, which contains isotopes. There are three types of geometries:
   - **Infinite homogeneous model**: that is, the entire geometry is one homogenized material.
   - **Heterogeneous equivalent model**: that is, the region declared as Equivalent Fuel would be homogenized into one fuel region, and the region declared as Equivalent Moderator would be homogenized into one moderator region, and we use collision probability method to estimate the probability of neutrons traveling from one region to another. In this model neutrons only know whether they are in Equivalent Fuel or Equivalent Moderator and do not have an explicit (x, y, z).
   - **Full heterogeneous model**: this is the model where neutrons truly know their (x, y, z) locations. In this model, region objects contain surfaces.
2. Region: a region object knows its region name, the material inside, its region type, volume and buckling. More specifically, we support 6 region types:
   - **Infinite medium**: this region covers an infinite space.
   - **Equivalent fuel**: this region is a fuel region using the infinite equivalence model.
   - **Equivalent moderator**: this region is a moderator region using the infinite equivalence model.
   - **Bounded fuel**: this region is a fuel region using the true heterogeneous model.
   - **Bounded moderator**: this region is a moderator region using the true heterogeneous model.
   - **Bounded general**: this region is a generalized region bounded by surfaces and is used in the true heterogeneous model.

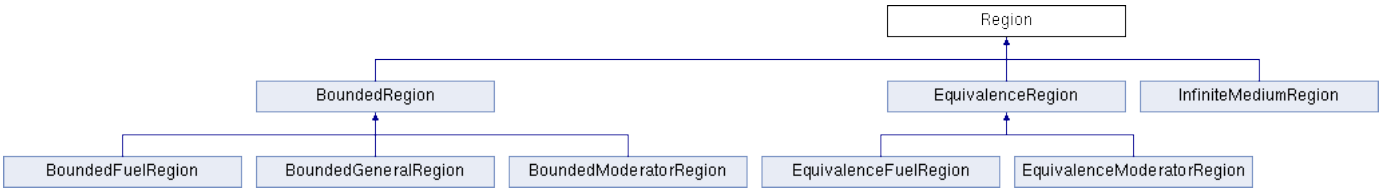

Figure 2: Inheritance Diagram of Region Class

3. Surface: Surface objects are used to bound regions in the true heterogeneous model. A surface object knows its name, ID #, its surface type (three sub-classes: x-plane, y-plane, z-cylinder), and its boundary condition (reflective, vacuum, interface).

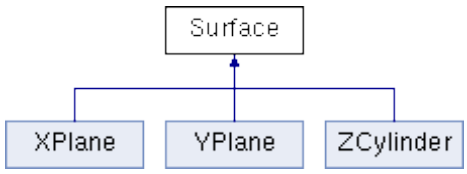

Figure 3: Inheritance Diagram of Surface Class

4. Material: an material object contains one or more isotope objects, and an material object is used to filled a region object. A material object knows basic info like its name, ID #, material density; when adding an isotope into a material, the material object needs to know the relative atomic mass associated with each isotope.

    - Note: each material and isotope isotope object needs to keep track of its number densities. Instead of having users to manually put in number densities, PINSPEC keeps track of them; notice that every time a new isotope is added, the number density of the material as well as other isotopes need to be updated as well. PINSPEC handles it by keeping a map of the isotopes with its corresponding relative atomic mass in each material object. So every time a new isotope is added, the material object loops through all existing isotopes and update the number density of the material and of each isotope.

5. Isotope: an isotope object can be considered as one of the most basic object in PINSPEC. We use isotope to build material, which in turn builds region, which builds geometry. An isotope object needs to have a name and atomic number, and PINSPEC would assign each isotope a unique identifier, calculate its $\alpha = \left(\frac{A-1}{A+1}\right)^2$, η, ρ for SLBW, averaged cosine of scattering angle $\bar{\mu} = \frac{2}{3A}$ etc.

    - We asks the user to provide isotope name in the format of: isotope's periodic table name (first letter capitalized), followed by its atomic number, with a hyphen in-between.

6. Neutron: a neutron structure knows its batch number (for running multiple batches and performing batch statistics), its previous energy (in eV) and its current energy (in eV), the region it is in, the material it is in, and the isotope it is in. For heterogeneous geometries, neutrons also know its (x, y, z) location, (u, v, w) velocity directions, and the closest surface. This neutron structure is being passed from class to class, and it contains the most essentially information that the other

classes would depend on to perform the simulation and generate the corresponding statistics.

7. Tally: PINSPEC uses a Tally class to keep the implementation of tally accumulation general. See Fig. 4 for the different sub-classes supported in the Tally class. There are also a TallyBank class that stores all the tallies, and a TallyFactory class that creates tallies. For more details, see Section 4.6 for an example on how to implement a new tally type.

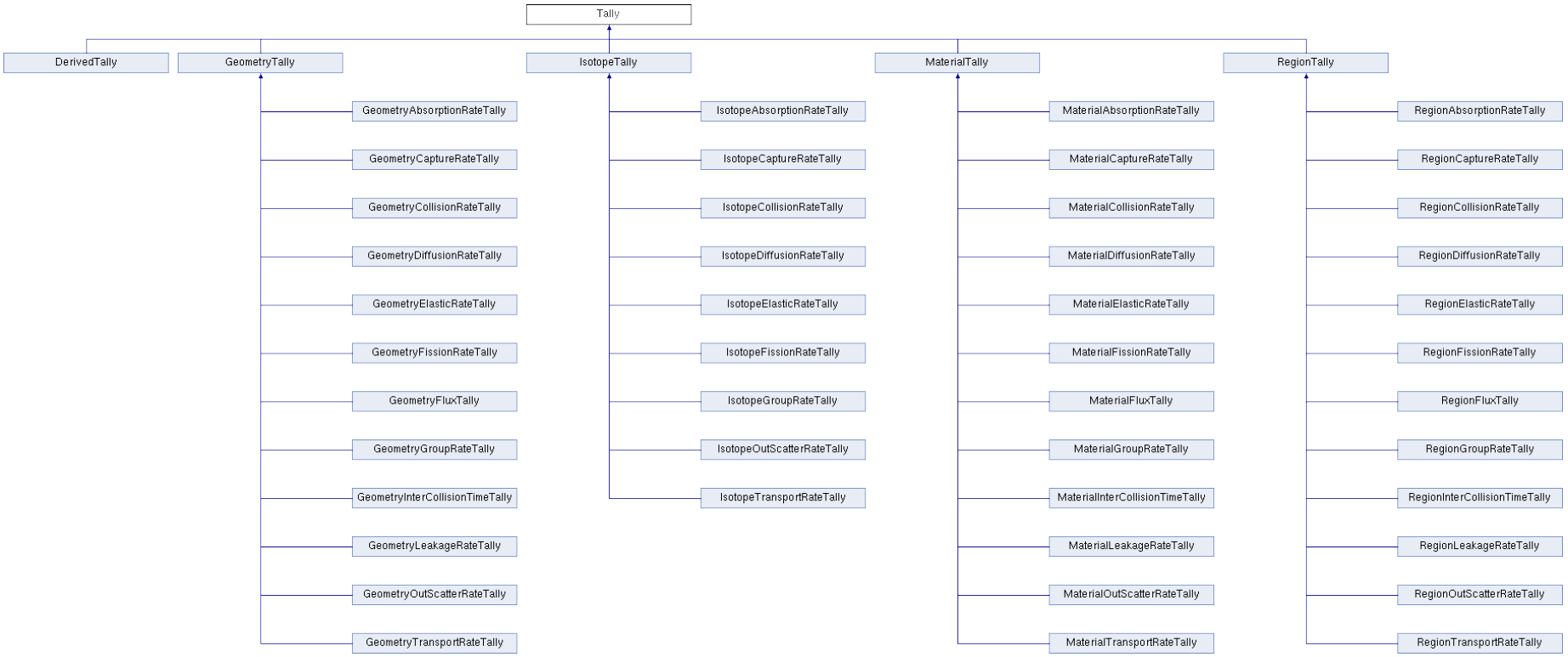

Figure 4: Inheritance Diagram of Tally Class

2 Installation and How To Run

An overview of the installation procedures can be found on the users' guide at PINSPEC's wiki page.

Table. 1 lists PINSPEC's package requirements and the third column provides installation suggestion for ubuntu users. Notes:

Package	Version #	Installation
Python	2.7	sudo apt-get install python2.7
matplotlib	1.2 or later	sudo apt-get install python-matplotlib
numpy	1.6 or later	sudo apt-get install python-numpy
scipy	0.9 or later	sudo apt-get install python-scipy
SWIG	–	sudo apt-get install swig
GIT	–	sudo apt-get install git

Table 1: PINSPEC Package Requirements and Installation Notes

- Ubuntu users:
  - If you have Ubuntu v12.10, the default installation (aka using the commands listed in Table 1) would get you the right versions.
  - If you have an earlier version of Ubuntu, and the packages you obtained through your Ubuntu release are earlier versions than the ones listed in Table. 1, we suggest you first install all the packages required, and try running PINSPEC, and python's error message would help you determine whether you need any newer version of software.
  - If you tried the previous bullet point and decided you have to build some of these packages from source, here are examples for pulling numpy and scipy from their git repositories:

    ```
    > git clone git://github.com/numpy/numpy.git numpy
    > git clone git://github.com/scipy/scipy.git scipy
    ```

    Then build using python's utilities:

    ```
    > python setup.py install
    ```

    For more details, see Scipy's installation page for instance.
- Other Linux users: use your corresponding package management system. For instance, in Gentoo you would use 'emerge' to get the above packages.
- Mac users: try install the listed packages using MacPort, and MacPort seems to be the Mac version of package management system, and it should make keeping track of dependencies and updating packages in the future easier.

2.1 A Special Note About Python Version and Running Python

- Python2 vs. Python3: because of the syntax incompatibility between python2 and python3, PINSPEC only supports python2 (more specifically it is tested mostly with python2.7). Developers could choose to update the python source codes into python3 syntax in the future if python3 gets more popular; though for now the easiest thing to do for users with python3 would be to install a copy of python2.7 as well, and every command that has 'python' in it, using the syntax 'python2.7' would request the correct version of python.

- We provide a setup.py file in the release that would build PINSPEC using distutils. To run in Linux:

  ```
  > python setup.py install --user
  ```

  The '–user' flag is handy for two reasons: a) you do not need root access later to run etc; b) you do not want to build PINSPEC into the default directories on your machine that keep your real programs.

- Mac users may run into problems with running our python setup.py file (located in the main directory). It is recommended to use the env prefix as in the following example for users with gcc-4.7:

  ```
  > env CC=gcc-mp-4.7 python setup.py install --user
  ```

2.2 Additional Tools

This section contains some notes on useful tools that developers might find beneficial. Notice this is by all means an incomplete list and would be updated from time to time.

1. Version control/source code management. We choose to use GIT (and use gitHub as our host for repositories), so it is recommended that the developers use git as well. However if you really insist, there might be a way to set up such that you can continue to use SVN, which is supported by GIT now, though we would be at the risk of losing our commit log messages.

2. Secure Copy (SCP) is useful in securely transferring files between two hosts. Two hosts can be: your local machine, and the server that you might run PIN-SPEC on. More specifically, the general command for copying a file from a local folder to a remote folder is:

   ```
   > scp <local folder path> <account>@<ip>:<remote folder path>
   ```

   For coying files from a remote folder to a local machine, you just need to need to swap the second and third argument.

3. SSHFS enables mounting of a remote filesystem on a local machine. That is, you can mount a remote directory (say, the one you have on a server) onto your laptop, and you can access it graphically using your file management system (e.g., click open a file, edit it, etc). Here are the three common commands:

   - To mount:

     ```
     > sshfs -0 idmap=user <account>@<ip>:<remote folder path> <local folder
     ```

   - To see what sftp subsystems are mounted:

     ```
     > ps aux | grep -i sftp | grep -v grep
     ```

   - To unmount:

     ```
     > fusermount -u <local folder path>
     ```

4. To be continued.

3 Manipulating Python Inputs

3.1 Utilizing Tally Arithmetic

In writing the python input file, users can utilize the tally arithmetics implemented and create essentially a new tally type by 'adding,' 'subtracting,' 'multiplying,' and 'dividing' two existing tallies together.

There are three tally arithmetics supported in PINSPEC using operator overload; we use addition as an example:

1. 'new_ tally = tally1 + tally2': after error checking (e.g., tallies have computed batch statistics, tally1 and tally2 have the same # of energy bins), a new tally (of derived type) of the correct length will be initialized, and entry i in new_tally would be the addition of tally1[i] + tally2[i].

2. 'new_tally = tally1 + 3.5': a new tally (of derived type) will be initialized to have the same length as tally1, and for every entry i, new_tally[i] = tally1[i] + 3.5. Notice PINSPEC only supports tally + constant, not constant + tally.

3. 'new_tally = tally1.addFloats(numpy.array([1,2]))': instead of the second argument being a single number, users can choose to pass in an array of numbers as well using numpy. In addition to addFloats(..), PINSPEC also supports addIntegers(..), or addDoubles(..).

For more details, see Tally.cpp's operator overloading functions.

If the tally you want cannot be implemented this way, see Section. 4.6 for suggestions on how to implement your own tally types.

3.2 Example: Add A New Isotope

The PINSPEC package comes with a library of common isotopes' ENDF/B-VII cross sections for capture, elastic scattering and fission cross sections.

- To view the isotopes available, developers can do a 'ls' in PINSPEC/pinspec/xs-lib.
- To add a provided isotope to your simulation, call Isotope construct with the isotope name (formatted as the isotope's periodic table element name followed by hyphen followed by atomic mass). For instance,

```
U235 = Isotope('U-235')
O16 = Isotope('O-16')
Zr90 = Isotope('Zr-90')
```

If an isotope of desire is not included in the xs folder, following the instruction in the next sub-section on how to define your own isotope cross section, and construct isotope object the same way as if the isotope is a PINSPEC pre-defined one.

3.3 Example: Add A New Cross Section

It is also possible to add to this library and use other isotopes from the ENDF/B-VII cross section library. To do this, visit the National Nuclear Data Center and follow the steps below:

- Select the element you wish to add to the library.
- On the right hand side, select the isotope.
- On the right hand side, select the 'plot' link next to the cross section desired ((n,elastic), (n,gamma) or (n,total fission)). A new tab will appear.
- On the right hand side, click "view evaluated data".
- At the top of the page, scroll over the link called "Text". Right click and select 'Save Link As..'. This will give you the option of saving the cross section file as a text file.
- Save the above file in the xs-lib folder (PINSPEC/pinspec/xs-lib), and title it with the element abbreviation, mass number, and isotope designation. For example, the capture cross section (n,gamma) for Boron 10 should be saved as B-10-capture.txt.
- Then you can use it as any pre-defined isotope types.

4 Manipulating C++ Codes

4.1 Create A New Source File

If you create a new source file, make sure do the following:

1. Add it to *setup.py*
2. If you need to pass data from the python input file into the C file, or you would like to retrieve the results generated by C++ and plot or print it using python, see Section 4.2.
3. Make sure you update the doxygen commenting accordingly, and run doxygen by:

   ```
   doxygen docs/Doxyfile
   ```

   See more details in Section. 4.3.
4. If you add in any additional functions, it is nice to update the test suites accordingly.
5. Two examples are shown about how to add a new tally and a new surface.

4.2 Passing Data Between Python and C++

If you would like to gain access to an array of data, you need to do so through SWIG. Currently we have Geometry.i to keep track of the data being passed between the two programs, and to add your own, for instance, you need to do something similar to the following:

```
%apply (float* ARGOUT_ARRAY1, int DIM1) {(float* xs, int num_xs)}
```

The above means that a (float *xs, int num_xs) argument in C++ can be translated into an array ARGOUT_ARRAY1 of length DIM1.

Examples can be found in Geometry.i which contains numpy template.

4.3 Doxygen Commenting

To generate html and pdf from source code commenting, run Doxygen by the following command:

```
doxygen docs/doxygen/Doxyfile
```

If you modify the source code, please update the Doxygen commenting accordingly. For instance, here are some styling suggestions:

1. For any class structure (including super-class and sub-class), use @class followed by class name, header file name, and header file path. Examples:

   ```
   @class Surface Surface.h "pinspec/src/Surface.h"
   @class XPlane Surface.h "pinspec/src/Surface.h"
   @class YPlane Surface.h "pinspec/src/Surface.h"
   ```

   In the above examples, Surface is the super-class, and XPlane and YPlane are sub-classes.

2. For any structure, try using @brief followed by a one-line short description, as well as @details followed by a longer description if needed.

3. For any structure, use @param followed by parameter name and description to comment on the inputs of a method, and use @return followed by variable name and description to comment on the outputs of a method.

4.4 Update Test Suites

4.5 Example: Implement A New Surface

4.6 Example: Implement A New Tally

4.7 Example: Implement A Monte Carlo Variance Reduction Technique

The main kernel that performs the Monte Carlo simulation is contained in Geometry.cpp as runMonteCarloSimulation(). First this function does error checking to make sure that the geometry and region types are compatible. Then it initializes tallies using tally bank.

Algorithm 1 High Level Monte Carlo Kernel

```
while not hit precision do
  for each batch do
    for each neutron inside the batch do
      Initialize neutrons: sampling energy from fission spectrum, place it in fuel.
      while neutron is alive do
        Find neutron's region.
        Call collideNeutron method in that region.
        Call tallyBank to tally that collision.
      end while
    end for
  end for
end while
```

How collideNeutron(..) method works:

- When it is called for a region, depends on the region subclass type, collideNeutron(..) may perform slightly different things. For instance, a Infinite Medium Region would just call collideNeutron(..) for the material inside; an Equivalence Region would figure out whether the neutron is in fuel or moderator right now, calculate PFF or PMF, and sample next location, and call collideNeutron in that material; whereas in Bounded Regions, collideNeutron(..) perform a true random walk, update neutron's location, and sample a collision type in the new material.
- When it is called in a material, it samples an isotope, and call collideNeutron(..) for that isotope.
- When it is called in an isotope, the method samples a collision type and generate the neutron's outgoing energy based on the geometry type. For instance, in infinite homogeneous and heterogeneous equivalence model, we perform asymptotic elastic downscattering above 4eV and perform thermal scattering below 4eV, and consider the direction of traveling to be uniformaly. Whereas in Heterogeneous geometry, we sample real ϕ and update neutron's velocity directions u, v, w