GSTpy User Guide

Robin Blume-Kohout, Erik Nielsen, John King Gamble, and Kenneth Rudinger

September 18, 2014

1 Overview

This document describes the design principles and basic usage of the GSTpy python codebase which computes Gate Set Tomography (GST) estimates. At the writing of this guide, the code is still being modified daily, and so while we expect most of the concepts presented here will remain unaltered into the future the exact syntax of the examples given may change. The most up to date and detailed documentation of the code are given by python doc-strings.

GSTpy defines two primary objects, GateSet and DataSet, and functions that generate or act on these types. While the code is structured into layers and modules internally (see section 4), the typical user will interact only with the top-most layers of GSTpy, which provide two independent interfaces:

- **Command-line interface.** The user creates JSON-format text files which, along with command line arguments specify a desired calculation or processing to a command-line-invokable python script. No python coding is required to use this interface.
- **IPython notebook interface.** The user imports the "ipythonGST" module into an IPython notebook and directly calls python functions to perform the desired processing.

For both interfaces the user must specify experiment data (to construct DataSet objects), gate definitions (to construct GateSet objects), and gate sequences (to specify special "fiducial" sequences) using the standard text format for these files (see section 5). The typical GSTpy workflow is given below, and is the same using either interface (though the specifics of how each step is carried out varies).

1. **Decide which gates will comprise your gate set.** This means deciding how many gates you'll have, what you want to call them (each should have a name of the form G¡lowercase¿ to be compatible with the input file formats), and what you expect each one to do in an ideal world. The same holds true for your preparation and measurement operations, except they should have rho¡number¿ and E¡number¿ names. All of this information gets put into "target gate set" file.
2. **Obtain on a set of "fiducial" gate sequences and "germ" gate sequences.** These are both sets of sequences of the gates in your gate set. Since these are typically given to you by us, we won't go into great detail here about how these are chosen. The salient point is that by obtaining experimental data for sequences of the form:

   $$\text{state-prep} + \text{fiducial-sequence} + \text{germ-to-power-sequence} + \text{fiducial-sequence} + \text{measure}$$

   allow the GST algorithms to estimate your gates particularly well. The fiducial sequences are the germ sequences are be specified gate string text files.
3. **Do the experiment.** Take data for the sequences described above and write results as a data-set format text file.
4. **Compute GST estimate.** Using all the files described up to this point, run GSTpy to compute estimates of the gates given the experimental data. There are several different algorithms each with numerous options which can be used for computing this estimate, which are selected through either JSON files (command-line interface) or directly using python (IPython interface). The algorithms and options are detailed in section 2.

5. **Analyze GST estimate.** Determine how well the estimation worked, and what types of gate errors are present. The types of analyses available in GSTpy are discussed in section 3.

The following two sections describe the options and specific steps for computing and analyzing GST estimates using GSTpy. Section 4 describes the code structure and section 5 specifies the expected format for the various input text files used by GSTpy.

2 Computing a GST estimate

There are several algorithms within GST for obtaining an estimate for the gate set. While the mechanics of each is different, the goal is the same: to take experimental data (contained in a DataSet object) and produce estimates for all the gates, state preparations, and measurement effects simultaneously. We will briefly describe and compare the different estimation methods in 2.1 and then provide concrete steps and examples for obtaining GST estimates using each interface in 2.2 and 2.3.

2.1 GST estimation methods

There are four main algorithms that can be used to obtain a gate set estimate.

Linear Gate Set Tomography (LGST) A process-tomography-like estimate for each gate is obtained using an effective preparation and measurement basis formed by the applying each fiducial gate sequence after and before the native preparation and measurement operations, respectively. This method uses only data for sequences of the form prep+fiducial+gate+fiducial+measure and runs very quickly since it only performs a linear inversion. This method is often used to generate an initial estimate of a gateset, since it is a non-iterative method that does not require an initial starting gateset. If data is generated by a Markovian process (gate set) and there is no sample error, the LGST estimate will match the gate set generating the data exactly.

Maximum Likelihood Estimation Gate Set Tomography (MLEGST) An iterative algorithm which seeks to maximize the likelihood function using a variety of local and/or global minimization methods. MLEGST does not require any particular gate sequences in the data set, but does require an initial starting point that can affect the final estimate when the likelihood function contains local minima. This susceptibility to local minima and the typically longer run-times make MLEGST a usually-inferior option compared to eLGST and LSGST.

Extended Linear Gate Set Tomography (eLGST) An iterative algorithm that uses the LGST process-tomography-like technique to estimate longer gate strings and minimize the least squares error between the action of sequences of gates from a gateset and the LGST estimates for those sequences. eLGST requires an initial gateset and requires data for each gate sequence used by the algorithm sandwiched by the full set of fiducial sequences. In return for the more structured data required, eLGST is the fastest iterative GST method for a given degree of estimation.

Least Squares Gate Set Tomography (LSGST) An iterative algorithm that minimizes the χ^2 error between a gateset's predicted probabilities and the frequencies given by data. Similar to MLEGST, it does not require any particular gate sequences in the data set, but does require an initial starting point, though it appears to be much less sensitive to this point. It runs slightly slower than eLGST for a similar sized data set, but achieves a better estimate. The flexibility and balance of speed and accuracy make LSGST the best method for most cases.

In addition to these gate set estimation algorithms, there are two additional operations that are often performed on gate sets between or after applications of the estimation methods.

Gauge Optimization Performs a gauge transformation on a gate set in order to optimize some quantity. Common optimized quantities include distance-to-a-target-gateset and distance-to-CPTP. Since applying a gauge transformation does not affect the probabilities predicted by a gateset, a gauge optimization can be thought of as a relatively benign operation that is used to better see and understand a gateset, similar to looking at an object from a different angle.

Contraction Alters a gate set so that it falls within some space, often CPTP space. Unlike a gauge optimization, a contraction changes the gates in a way that affects the probabilities and as such is a more destructive operation that should be used only when necessary.

2.2 Obtaining an estimate using the command line

To obtain an estimate using the command line interface the user runs the `doGST` python script. This script takes as input two types of JSON-format text files which the user creates:

- **Algorithm JSON file.** Encodes dictionary in JSON format which specifies which GST method is used and parameters for that method. In particular, the `gst method` member of the dictionary specifies the name of the method used, and be set to one of the strings:
  - `Linear` - run LGST
  - `Sequential MLE` - run a single MLEGST estimation
  - `Ratchet MLE` - iterate over the (iterative) MLEGST estimation
  - `Extended Linear` - run eLGST
  - `Least Squares` - run LSGST
  - `Gauge Opt and Contract` - gauge optimize and/or contract a gateset
  - `do nothing` - don't run any algorithm – just load and write the dataset(s)

  Examples of algorithm JSON files can be found in the examples/ directory of GSTpy.
- **Dataset JSON file.** Encodes a dictionary in JSON format which specifies the target gate set, fiducial gate strings, and dataset. Each of these can be specified by simply referencing a text-format gateset, gate sequence, or dataset input file, respectively, by name. Examples of data set JSON files can be found in the examples/ directory of GSTpy.

After creating an algorithm and dataset JSON file GST can be run using the syntax:
`doGST --algorithms MyAlgSpecs.json --datasets MyDatasetSpecs.json`
which is equivalent to
`doGST -a MyAlgSpecs.json -d MyDatasetSpecs.json`
Multiple algorithm and/or dataset JSON files may be specified to run multiple algorithms on multiple datasets. Full usage can be obtained by `doGST --help`.

2.3 Obtaining an estimate using an IPython notebook

To obtain an estimate using the IPython notebook interface there are three easy steps:

1. **import the ipythonGST.py module** contained in the `ipython_notebooks` directory of GSTpy with a line like `import ipythonGST as GST`.
2. **Load Objects.** Use the `GST.loadGateset`, `GST.loadDataset`, and `GST.loadGatestringDict` functions to load a target gateset, dataset, and fiducial gate sequence list from files, respectively.
3. **Run GST.Core functions.** The sub-module `GST.Core` contains the top-level functions which execute the algorithms listed above. Examples include `doLGST`, `doMLEGST`, `doExLGST`, `doLSGST`, `doIterativeExLGST`, `doIterativeLSGST`, `optimizeGauge`, `contract`. Options to these methods are given as arguments to these functions which are documented in the pydoc strings that can be called using IPython.

Example IPython notebooks can be found in the `ipython_notebooks` directory of GSTpy.

3 Analyzing a GST estimate

After a gateset has been estimated, it can and should be run through a series of steps in order to determine whether the estimate can be trusted. If it is deemed to be a good estimate, it can be processed to more easily understand what the estimate implies about the gates begin performed, or in order to see trends in many runs (e.g. for statistical testing). Unlike the case for obtaining estimates, where most operations can be done using either the command line or IPython notebook interfaces, the analysis tools available using each of the interfaces are currently quite different. As such, this guide will separately describe the analysis/post-processing methods available from each interface. Since command-line output files can be read using the IPython interface, and the IPython interface can be used to generate command-line-formatted output files, generating an estimate using one interface does not limit one to that interface's analysis tools.

3.1 Analysis tools available using the command line

The command line tools deal with output files that are pickled `OutputData` objects (see `OutputData.py`). An `OutputData` object can contain any number of named `GateSet` and `DataSet` objects, as well as named python dictionaries of parameters. Generating estimates using `doGST` as described above creates one or more output files that are then used as input to the command-line post processing tools described below. There is typically one output file created per (algorithm,dataset) pair given to `doGST`. Typically, output files contain at least a `GateSet` named "final", the final estimated gateset after the algorithm fully completes, and a `DataSet` named "training", the dataset used by the algorithm. They may also contain gatesets named "target" (if a target gateset was specified) and "data generation" (if simulated data was used), as well as algorithm-dependent names of intermediate gatesets (for example, the final gateset before any contraction was performed).

3.1.1 Peeking at an output file using `inspectGST`

When all you want to do is determine what gatesets, datasets, and parameter dictionaries are contained in an output file (a pickled `OutputData` object written to disk), the `inspectGST` tool can be used. It can be used to print the gates (or their Choi matrices) of gatesets or the data contained in datasets from a single output file. Run `inspectGST --help` to show the usage.

3.1.2 Plotting output data using `plotGST`

When all you want to plot data, typically from numerous output files that were run using simulated data, `plotGST` is the tool for the job. It takes as input multiple output files and pools together the data contained in them. It computed user-specified x-axis and y-axis quantities and then renders the data in a specified graphical or non-graphical format. Note that despite its name, `plotGST` can be used just to bulk-compute values based on the estimates in many output files and save the results to a text, comma-separated-values, or pickled python pandas file. Multiple y-axis variables may be specified, and multiple values can be plotted on the same axes using the "–linekey" option. Full usage can be obtained by running `plotGST --help`. An important option is the "–info" flag which instead of computing and plotting anything causes `plotGST` to print a list of the available (computable) quantities that can be used after the "–xaxis", "–yaxis", and "–linekey" options.

3.1.3 Generating reports using `reportGST`

A latex-format report can be generated using one or more gatesets and datasets of a single output file using `reportGST`. This tool takes several options which determine how much output is displayed (e.g. "–gs-detail" and "–ds-detail"), but by and large displays a pre-determined set of gateset-dataset metrics and gateset gate decompositions for given gatesets and datasets. Other notable options are "–paperwidth" and "–paperheight" which specify the output paper size. These may be modified as necessary to get the report tables to display entirely (and not be clipped). Full usage information is obtained by running `reportGST --help`.

3.2 Analysis tools available using an IPython notebook

The analysis tools available from the IPython notebook interface are tailored to assessing whether a computed estimate can be trusted or not. Note that this is *not* the task of finding for error bars on the parameters comprising the estimate, but rather determining whether the parameters of the model are as a whole able to capture the physics inherent in the data. The analysis functions are located in the fine `AnalysisTools.py` under the GSTpy `src` directory, and can be divided into several main categories:

χ^2 functions Evaluate χ^2 using various objects. Includes `ChiSqFunc`, `ChiSq`, `TotalChiSquared`.

Matrix functions These functions evaluate a matrix whose rows and columns correspond to the effects $\{\langle E_i|\}$ and preparations $\{|\rho_i\rangle\}$, respectively. In many circumstances there exists only a single native state preparation $|\rho\rangle$ and measurement $\langle E|$, along with a set of fiducial strings $\{F_i\}$, such that $|\rho_i\rangle = F_i|\rho\rangle$ and $\langle E_i| = \langle E|F_i$, in which case the matrix is square and the rows and columns correspond to the fiducial gate strings. The functions in this category all take a gatestring G as an argument and differ only in what is computed as M_{ij}, the i,j-th entry of the returned matrix:

- `TotalCountMatrix` - M_{ij} = total number of samples for the $F_i + G + F_j$ string (in a specified `DataSet`).
- `CountMatrix` - M_{ij} = the number of outcomes for the $F_i + G + F_j$ string with the specified spam label (in a specified `DataSet`).
- `FrequencyMatrix` - M_{ij} = the frequency of outcomes for the $F_i + G + F_j$ string with the specified spam label (in a specified `DataSet`).
- `ProbabilityMatrix` - M_{ij} = the probability of the specified spam label outcome for the F_i+G+F_j string (using a specified `GateSet`).
- `ChiSqMatrix` - M_{ij} = the χ^2 value of the specified spam label outcome for the $F_i + G + F_j$ string (using a specified `GateSet` and `DataSet`).
- `DirectLGSTProbabilityMatrix` - similar to `ProbabilityMatrix` except the probabilities are obtained by directly estimating G using LGST.
- `DirectLGSTChiSqMatrix` - similar to `ChiSqMatrix` except the values are obtained by comparing the data with the LGST estimate of G instead of a separately given gateset.

Matrix plotting functions General functions for plotting matrices and grids of matrices.

- `ColorBoxPlot` - plots a matrix as a grid of colored boxes, with colors corresponding to their values
- `NestedColorBoxPlot` - plots a grid of matrices as a grid of colored boxes, with colors corresponding to their values

χ^2 plotting functions Use the matrix plotting functions to plot the results of the χ^2 matrix functions.

- `ChiSqBoxPlot` - generates a color box plot of χ^2 values for a given set of germs and germ exponents. Useful keyword arguments are:
  - "sumUp" toggles whether the sum or the entire matrix of the χ^2 matrix corresponding to a single (germ, germ exponent) pair is displayed
  - "scale" sets the size of the plot
  - "boxLabels" toggles the numbers on the colored boxes; False makes plot generation much faster
  - "interactive" toggles interactive controls used to change the color scale and box labels
  - "histogram" toggles whether a histogram of the plotted data is displayed as well
  - "histboxes" specifies the number of histogram bins that are used
- `ZoomedChiSqBoxPlot` - generates an interactive color box plot of χ^2 values for a single (germ, germ exponent) pair selected by interactive drop-down controls. Useful when paired with `ChiSqBoxPlot` when "sumUp" equals True.
- `DirectLGSTChiSqBoxPlot` - same as `ChiSqBoxPlot` but plots Direct-LGST χ^2 results
- `ZoomedDirectLGSTChiSqBoxPlot` - same as `ChiSqBoxPlot` but plots Direct-LGST χ^2 results

4 The GSTpy code structure

This section describes at a high level the structure of the GSTpy code. For detailed descriptions of individual functions or classes, please refer to the docstrings contained in the code files themselves. The GSTpy code is arranged into logical groups or layers. From what is roughly a top to bottom ordering, we describe these groups/layers.

1. **Command-line drivers.** Files `doGST.py, inspectGST.py, plotGST.py, reportGST.py` serve as command-line interface drivers, specifying what command line arguments are available and how they are passed to lower levels.

2. **Command-line algorithm and dataset definitions** `GST_Methods.py` defines the algorithms available from the command line. An algorithm typically involves calling core GST functions in some sequence specified by the algorithm's parameters (input via the algorithm JSON file). `DataSetParams.py` specifies how to take a dataset JSON file and create one or more datasets from it, and includes the ability to generate simulated (fake) data if needed.

3. **IPython notebook interface** The files `ipythonCore.py, ipythonGST.py, ipythonMath.py` define which functions are available from the IPython interface, limiting the functions one can call from IPython to those that are well-documented and appropriate (e.g. not command-line interface specific).

4. **Core GST functions** `CoreFunctions.py` contains what you'd think it does – all the core functionality to compute GST estimates. All the LGST, LSGST, eLGST, MLEGST, gauge optimization, and contraction logic are located here.

5. **GST objects** `DataSet.py, GateSet.py, OutputData.py` define the objects used to encapsulate datasets, gatesets and output files (particularly when using the command line interface).

6. **File I/O** `Loaders.py` defines IPython accessible functions for loading datasets and gatesets from files. `StdInputParser.py` defines an object type for parsing text formatted gate strings and dataset file lines and defines GSTpy's accepted input grammar.

7. **Analysis** `AnalysisTools.py` defines IPython-accessible analysis functions. `ComputeMetrics.py` contains the routines used primarily by `plotGST` to compute various quantities related to gatesets and datasets.

8. **Utility functions** There are a number of python modules devoted to particular categories of utility functions:

   - `GateSetTools.py` contains functions that create and modify `GateSet` objects. For example, rotating or depolarizing a gateset.
   - `GateStringTools.py` contains functions for creating gate string lists from a more compact description. For example, this file contains a function to list all the gate strings using a specified set of gates with length less than or equal to a given integer.
   - `GST_Exceptions.py` defines several exception classes that are used within GSTpy.
   - `LatexUtil.py` contains functions for processing and rendering objects in LaTeX.
   - `LikelihoodFunctions.py` defines the log-likelihood function, its derivative, and penalty terms that were initially used solely within the log-likelihood function (but now are used elsewhere too).
   - `Optimize.py` is a wrapper around scipy.optimize.minimize to provide a more standard interface for performing various types of optimizations. For example, basinhopping can be called using the same function as BFGS using this module.
   - `UseMPI.py` is a tiny module used to isolate the logic for whether MPI should be turned on or off.
   - `Util.py` contains general utility functions

9. **Deprecated** The files `analyzeGST.py, StatVar.py, Metrics.py, UnitTests.py` are deprecated.

5 Input text file formats

Insert existing description of input format here?