Documentation:

Purpose:

This program was written in order to numerically determine the various properties of three qubit violation of the Svetlichny inequality. Generally speaking, it uses a minimization method, SLSQP to be specific[1], to determine which measurement bases yield the maximum violation of the Svetlichny inequality. These sets of measurement directions are then saved in a file dependent on the state of the system for later use.

Installation:

To install, first make sure to have python running smoothly, and make sure that you can import different modules. On a mac, this can be accomplished by installing the correct version of Xcode (found either in the App store or on the mac installation disk), and then installing the academic version of Enthought's suite of python programs.[2] On windows, the Enthought python suite seems to be sufficient, but this has only been tested on windows XP.

To install the actual maximization module, simply choose the correct version for the operating system (windows or mac/linux), navigate to the folder ~\S_violation in terminal/ command line and run the following command: *python setup.py install*. This will install the python code as a module called S_maximize_3q which can then be imported like any other module.

Also before running, ensure that the folders are set up properly. Because of the way the program calls files, the proper structure must be setup as follows:

- 3_qubit_S (or root, it can be renamed)
    - Data
        - "Trialname"- each trial must have a folder with the desired name within Data. This makes is so that large numbers of files can be moved around with ease.
    - Figures
        - AComp
        - VComp
        - SComp
    - Printout

Usage:

The purpose of creating this module was to have several useful functions that could be callable in other programs. Each one is called (assuming one has the line "*import S_maximize_3q as S3"* in the code) by using the *S3.function(*args)* terminology. A list of functions with a description of arguments and their usage follows. Also note, the output is listed in the order the function outputs it. So for example, to correctly record the output from GGHZ(2*pi), one would want to write psi,name=S3.GGHZ(2*pi).

Specific operating system notes:

- Mac/Linux:

[1] See http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms for information about the routine itself.

[2] Go to http://www.enthought.com/ and find the python distribution 7.3. Other distributions may work, but this is the one that the software was tested on.

    - While the program has not been tested using linux as of yet, due to the fact that both mac and linux call files in the same way, there is a very high probability that it will work the same. In either case, to create data or figures, the program must be run from the 3_qubit_S folder, or in whatever folder it was installed if a different folder was chosen. The program will look for the specific folder structure above and if it cannot find it, will yield an error.
- Windows:
    - While the program works in windows, there are some limitations. First, the windows variant at the moment can only create data, not produce any charts or print. Second, the windows machine only needs the folder structure
        - 3_qubit_S
            - Data
                - Trialnames
    - and each program will save data to the folder in which it is run. So to have a GGHZ trial, one must have a folder named /Data/GGHZ and run the program from within that folder.

Notation note:

This program can deal with state vectors parameterized by either one or two parameters. By this is meant a state such as the GGHZ state, *GGHZ=cos(θ) |000>+sin(θ) | 111>* , which is an example of a one parameter state, or the Top Plate state, TopPlate=12(cos(θ)|111>+sin(θ)|100>+cos(ϕ)|010>+sin(ϕ)|001>), which is an example of a two parameter state.

Function list:

Below is a list of all functions that can be called for mac/linux. Only the ones involved in creating data are used in Windows.

Bs.ijk	***MS1(theta)***
Fa(x,psi)	***MS2(theta)***
S_maximize(n,psi)	***W(theta,phi)***
GGHZ(theta)	
unique_search(xoptalist)	***DC_custom_2p(ord,n,ln,psif,name)***
vecprint(n,ln,name,param)	***p1call_file(name,n,q,ln)***
max_cull(speciallist,psi)	***p2call_file(name,n,q,j,ln)***
save_file_1p(name,n,q,ln,psi,speciallist,maxlist,xoptalist)	***p1tgl_theta_plot(name,n,ln,save,clear)***
save_file_2p(name,n,q,j,ln,psi,speciallist,maxlist,xoptalist)	***p1S_max_tgl_plot(name,n,ln,save,clear,color)***
ord_choose(ord,xoptalist,psi)	***p1S_max_theta_plot(name,n,ln,save,clear)***
DC(sc,ord,n,ln)	***p1Scomp_theta_plot(name,n,ln,compnum)***

DC_custom_1p(ord,n,ln,psif,name)	p1Scomp_tgl_plot(name,n,ln,compnum)
p1Veccomp_theta_plot(name,n,ln):	p1Veccomp_tgl_plot(name,n,ln):
p2Veccomp_tgl_plot(name,n,ln)	p2S_max_tgl_plot(name,n,ln,save,clear,color)
p2Scomp_tgl_plot(name,n,ln,compnum)	p1vecnumber_theta_plot(name,n,ln,save,clear,color)
p1Thetaccomp_theta_plot(name,n,ln)	p1vecnumber_tgl_plot(name,n,ln,save,clear,color)
p1Thetaccomp_tgl_plot(name,n,ln)	p2vecnumber_tgl_plot(name,n,ln,save,clear,color)
p1Scomp_compare_theta_plot(name,n,ln)	p2Thetaccomp_tgl_plot(name,n,ln)
p1Scomp_compare_tgl_plot(name,n,ln)	p2Scomp_compare_tgl_plot(name,n,ln)
p2S_max_theta_plot(name,n,ln,save,clear,color)	p2S_max_phi_plot(name,n,ln,save,clear,color)

Bs.ijk:

This calls a specific state. For example, Bs.ooo calls the state |000> or Bs.lol calls the state |101>, where each number refers to qubit A,B,C in that order, and 0,1 can be either ↑or ↓, excited or not, or some other two-state system. Either way, each of A,B,C is a qubit and therefore has only two possible states. The data is held in an [8,1] dimensional numpy array. Arguments such as Bs.ijk.conjugate() or Bs.ijk.transpose() work perfectly well, as these are full numpy arrays.

Fa(x,psi):

- Input:
  - x: a numpy array determining the paramaterization of the measurement basis for the expectation value of the Svetlichny operator.
  - psi: an [8,1] numpy array that represents the state of the system
- Output:
  - The expectation value of the Svetlichny operator, A(BP+B'P')+A'(BP'-B'P) where P=C+C' and P'=C-C', given a specific measurement basis determined by x
- Description and notes:
  - This function calculates the expectation value of the Svetlichny operator. It requires psi, a numpy array representing the state of the system, and x, a series of twelve values of theta which determine the measurement basis. Each measurement direction is broken up by the following parameterization with respect to theta and phi:

$$V=[sin(\theta)\,cos(\phi), sin(\theta)\,sin(\phi), cos(\theta)]$$

S_maximize(n,psi):

- Input

- n: an integer for the number of guesses.
- psi: an [8,1] numpy array that represents the state of the system

- Output:
  - The function outputs all the minimized parameters for the six different measurement basis (a,a’,b,b’,c,c’) in a numpy array, xoptalsit. Calling xoptalist[i] gives the x to input to Fa(x,psi).
- Description and notes
  - This function maximizes the previous function, Fa(x,psi), thereby maximizing the Svetlichny operator. It uses the SLSQP minimization method, bounds the values of x (the angles passed into Fa) between 0 and 2pi, and uses a random number as a starting guess for the minimization routine. As a result of the fact that the initial guess is inherently random, to ensure that the program does not get stuck in a local rather than a global minima, a series of guesses must be used. The value of n determines this, as it determines how many times the routine will run. While it is not quantitatively known what a good lower bound is, 25 guesses is safe, though there are few errors as low as 10.[3] The minimization routine is set to an accuracy of 1.0E-11, though it is not clear from the documentation that this means that the actual value of the minima is accurate to this, or if this is a value for internal computation. As before, psi is a numpy array for the state vector.

GGHZ(theta):
- Input:
  - theta: a float, determines the value of the parameter for the state.
- Output:
  - psi: an [8,1] numpy array that represents the state of the system
  - name: a string of the name of the state, “GGHZ” to be given to the saving functions
- Description and notes:
  - Four states have been explicitly programmed into the module, with the option of a custom state as desired. The first of these is the general Greenberger-Horne-Zellinger state, $|\psi\rangle = cos(\theta)\,|000\rangle + sin(\theta)\,|111\rangle$.

MS1(theta):
- Input:
  - theta: a float, determines the value of the parameter for the state.
- Output:
  - psi: an [8,1] numpy array that represents the state of the system
  - name: a string of the name of the state, “MS1” to be given to the saving functions
- Description and notes:
  - The MS1 state is given as: $|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + cos(\theta)|101\rangle + sin(\theta)|111\rangle)$

MS2(theta):
- Input:
  - theta: a float, determines the value of the parameter for the state.
- Output:

[3] This was checked by running a known trial of the generalized Greenberger-Horne-Zeilinger state. It was found that around 5 there are noticeable differences from the theoretical, whereas around 10 or 15 there are no errors. The program does not, however, guarantee that all possible measurement bases will be correctly found, just that one that maximizes the expectation value of the Svetlichny operator.

- psi: an [8,1] numpy array that represents the state of the system
- name: a string of the name of the state, “MS2” to be given to the saving functions

- Description and notes:
  - The MS1 state is given as: $|\psi> = \frac{1}{\sqrt{2}}(|000> + cos(\theta)|110> + sin(\theta)|111>)$

W(theta,phi):

- Input:
  - theta: a float, determines the value of the first parameter for the state.
  - phi: a float, determines the value of the second parameter for the state.
- Output:
  - psi: an [8,1] numpy array that represents the state of the system
  - name: a string of the name of the state, “W” to be given to the saving functions
- Description and notes:
  - The W state is given as: $|\psi> = sin(\theta)cos(\phi)|100> + sin(\theta)sin(\phi)|010> + cos(\phi)|001>)$

unique_search(xoptalist):

- Input:
  - xoptalist: the list output from S_maximize(n,psi) which contains all the maximized measurement bases.
- Output:
  - speciallist: a list of all unique measurement bases.
- Description and notes:
  - As a result of the fact that the minimization routine requires many guesses, the same solution is generally recorded many times. In order to cut down on computational time later, the dot products are taken between each measurement basis, and if the dot product is close enough to one, it is ignored, and if not it is kept as a unique vector.

vecprint(n,ln,name,param):

- Input:
  - n: number of guesses for S_maximize
  - ln: the overall number of calculations for a parameterization of psi
  - name: name of the state
  - param: equals 1 for a one parameter state, equals 2 for a two parameter state
- Output:
  - none
- Description and notes:
  - This function prints out a file of the maximized vectors in the folder Printout. This is human-readable, but not python-readable. It is intended to be used to look for comparisons between different vectors of various different states.

max_cull(speciallist,psi):

- Input:
  - speciallist: a list of all unique measurement bases. This could be the raw list too, but it is intended to be used on the unique vectors.
  - psi: an [8,1] numpy array that represents the state of the system
- Output:
  - maxlist: a list of all maximized measurement bases from the inputted list

- Description and notes:
    - This routine will find the bases that have the maximum violation and output these in a special list. It is intended to be used after cutting out any duplicate vectors so that in the end, maxlist contains only unique directions of measurement which maximize <S>

save_file_1p(name,n,q,ln,psi,speciallist,maxlist,xoptalist):

- Input:
    - name: name of the state
    - n: number of guesses for S_maximize
    - q: the trial number for a given parameterization of psi.
    - ln: the overall number of calculations for a parameterization of psi
    - psi: an [8,1] numpy array that represents the state of the system
    - speciallist: the output of unique_search
    - maxlist: the output of max_cull
    - xoptalist: the output of S_maximize
- Output:
    - none
- Description and notes:
    - This function saves a file for the given inputs in the Data folder. The filename is of the following form:name+" 3 qubit pickled "+str(n)+" , "+str(q)+" , "+str(ln) +" .txt" This file is pickled, meaning it can be called later from python with ease. This comes with a cost, however, and it is the fact that the data cannot be read without unpickling it in python. It saves each file in a specific folder which has to be made beforehand: \Data\trialname

save_file_2p(name,n,q,j,ln,psi,speciallist,maxlist,xoptalist):

- Input:
    - name: name of the state
    - n: number of guesses for S_maximize
    - q,j: the trial number for a given parameterization of psi.
    - ln: the overall number of calculations for a parameterization of psi
    - psi: an [8,1] numpy array that represents the state of the system
    - speciallist: the output of unique_search
    - maxlist: the output of max_cull
    - xoptalist: the output of S_maximize
- Output:
    - none
- Description and notes:
    - This function saves a file for the given inputs in the Data folder. The filename is of the following form:name+" 3 qubit pickled "+str(n)+" , "+str(q)+" , "+str(j) +" , "+str(ln)+" .txt" This file is pickled, meaning it can be called later from python with ease. This comes with a cost, however, and it is the fact that the data cannot be read without unpickling it in python. It saves each file in a specific folder which has to be made beforehand: \Data\trialname

ord_choose(ord,xoptalist,psi):

- Input:
    - ord: 0 or 1
    - xoptalist: the output from S_maximize
    - psi: an [8,1] numpy array that represents the state of the system

- Output:
    - none
- Description and notes:
    - This allows one to change whether max_cull or unique_search is called first. If ord==0, unique_search is called first. If ord==1, max_cull is called first

DC(sc,ord,n,ln):
- Input:
    - sc: chooses the state. 1 for GGHZ, 2 for MS1, 3 for MS2, 4 for W
    - ord: 0 or 1 to control ordchoose
    - n: number of guesses for S_maximize
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
    - none
- Description and notes:
    - This is a data creation routine which, for the for named states, will calculate and save data. ln describes how many steps there will be for the parameters of psi between [0,2pi].

DC_custom_1p(ord,n,ln,psif,name):
- Input:
    - ord: 0 or 1 to control ordchoose
    - n: number of guesses for S_maximize
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - psif: a function which yields a state psi, given a specific value of theta. An example is given in the notes.
    - name: a string that describes how the specific calculation should be named for saving and graphing.
- Output:
    - none.
- Description and notes:
    - This is a data creation routine which will calculate the value of <S> for any given state with only one parameter. It allows the user to define and name any possible state (provided it only uses three qubits). The following is an example of code for psif. S_maximize_3q is imported as Bs here.
        - `>>>def psif(theta):`
        - `>>>   psi=1.0/2.0**.5*(cos(theta)*(Bs.lll)+sin(theta)*(Bs.olo+Bs.ool))`
        - `>>>   return psi`

DC_custom_2p(ord,n,ln,psif,name):
- Input:
    - ord: 0 or 1 to control ordchoose
    - n: number of guesses for S_maximize
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - psif: a function which yields a state psi, given a specific value of theta. An example is given in the notes.
    - name: a string that describes how the specific calculation should be named for saving and graphing.

- Output:
    - none.
- Description and notes:
    - This is a data creation routine which will calculate the value of <S> for any given state with exactly two parameters. It allows the user to define and name any possible state (provided it only uses three qubits). The following is an example of code for psif. S_maximize_3q is imported as Bs here.
        - >>>def psif(theta,phi):
        - >>>   psi=1.0/2.0**.5*(cos(theta)*(Bs.lll)+sin(phi)*(Bs.olo+Bs.ool))
        - >>>   return psi

p1call_file(name,n,q,ln):

- Input:
    - name: name of the state desired
    - n: number of guesses
    - q: specific trial number
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
    - psi: an [8,1] numpy array that represents the state of the system
    - speciallist: the output of unique_search
    - maxlist: the output of max_cull
    - xoptalist: the output from S_maximize
- Description and notes:
    - This function will open and allow one to modify the elements contained in the output. This is the first step in analyzing or manipulating the data calculated for the previous tests, as this function unpickles the information to be used by python. Only works for one parameter states, and it calls files from the Data folder.

p2call_file(name,n,q,j,ln):

- Input:
    - name: name of the state desired
    - n: number of guesses
    - q: specific trial number
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
    - psi: an [8,1] numpy array that represents the state of the system
    - speciallist: the output of unique_search
    - maxlist: the output of max_cull
    - xoptalist: the output from S_maximize
- Description and notes:
    - This function will open and allow one to modify the elements contained in the output. This is the first step in analyzing or manipulating the data calculated for the previous tests, as this function unpickles the information to be used by python. Only works for two parameter states, and it calls files from the data folder.

NOTE:

- The following functions deal with graphing. To be able to see the graph in a dynamic form allowing for zoom, the pylab command show() must be added after calling the function. Note, however, that this command will stop computation, so only use it at the desired end.

p1tgl_theta_plot(name,n,ln,save,clear):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
  - save: whether or not the plot is to be saved. 1 for save, anything else for not save.
  - clear: whether the previous plot should be cleared or not. 1 for clear, anything else for not clearing.
- Output:
  - A line plot of the three-tangle with respect to the state parameter theta.
- Description and notes:
  - This function graphs, for a given state, the three tangle versus the state parameter theta. The way that python graphs, if the figure is not cleared, any previous graphs will be layered on top of one another, with the latest one being the top. This can be beneficial to compare various states, but it can also be detrimental in the fact that it makes for confusing graphs. Figures are saved to the folder Figures with an appropriate name, such as “GGHZ tgl vs theta, 25, 500.png” for a GGHZ trial of n=25, ln=500 and a graph of the three tangle versus theta. Saves to the folder \Figures.

p1S_max_tgl_plot(name,n,ln,save,clear,color):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
  - save: whether or not the plot is to be saved. 1 for save, anything else for not save.
  - clear: whether the previous plot should be cleared or not. 1 for clear, anything else for not clearing.
  - color: the color the graph should be.
- Output:
  - A scatter plot of the maximum Svetlichny violation with respect to the three tangle.
- Description and notes:
  - This function graphs, for a given state, the maximum Svetlichny violation versus the three tangle. Due to the fact that the scatter plot function does not change color depending on whether there are multiple plots, a color must be specified to be able to distinguish between various plots. The way that python graphs, if the figure is not cleared, any previous graphs will be layered on top of one another, with the latest one being the top. This can be beneficial to compare various states, but it can also be detrimental in the fact that it makes for confusing graphs. Figures are saved to the folder Figures with an appropriate name, such as “GGHZ tgl vs theta, 25, 500.png” for a GGHZ trial of n=25, ln=500

and a graph of the three tangle versus theta. This function only works with one parameter states. Saves to the folder \Figures.

p2S_max_tgl_plot(name,n,ln,save,clear,color):

- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - save: whether or not the plot is to be saved. 1 for save, anything else for not save.
    - clear: whether the previous plot should be cleared or not. 1 for clear, anything else for not clearing.
    - color: the color the graph should be.
- Output:
    - A scatter plot of the maximum Svetlichny violation with respect to the three tangle.
- Description and notes:
    - This function graphs, for a given state, the maximum Svetlichny violation versus the three tangle. This function only works for two parameter states. Saves to the folder \Figures.

p1S_max_theta_plot(name,n,ln,save,clear):

- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - save: whether or not the plot is to be saved. 1 for save, anything else for not save.
    - clear: whether the previous plot should be cleared or not. 1 for clear, anything else for not clearing.
- Output:
    - A line plot of the maximum Svetlichny violation with respect to the parameter theta.
- Description and notes:
    - This function graphs, for a given state, the maximum Svetlichny violation versus the state parameter theta. The way that python graphs, if the figure is not cleared, any previous graphs will be layered on top of one another, with the latest one being the top. This can be beneficial to compare various states, but it can also be detrimental in the fact that it makes for confusing graphs. Figures are saved to the folder \Figures with an appropriate name, such as “GGHZ tgl vs theta, 25, 500.png” for a GGHZ trial of n=25, ln=500 and a graph of the three tangle versus theta. This only works for one parameter states.

p1Scomp_theta_plot(name,n,ln,compnum):

- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].

    - compnum: whether the <S> operator will be broken into two (compnum==2) or four (compnum==4) components.
- Output:
    - Seven line plots are saved in the folder Figures for the max <S> violation, and for each component. For compnum==2, the Svetlichny operator is broken up into the following two componentes: A(BP+B'P') and A(BP'-B'P). If compnum==4, then it is broken up into ABP, AB'P', ABP', -AB'P'.
- Description and notes:
    - This outputs graphs showing the component's expectation value on the y axis and the state parameter theta on the x axis. Saves to the folder \Figures\SComp and only works for one parameter states.

p1Scomp_tgl_plot(name,n,ln,compnum):
- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - compnum: whether the <S> operator will be broken into two (compnum==2) or four (compnum==4) components.
- Output:
    - Seven line plots are saved in the folder Figures for the max <S> violation, and for each component. For compnum==2, the Svetlichny operator is broken up into the following two componentes: A(BP+B'P') and A(BP'-B'P). If compnum==4, then it is broken up into ABP, AB'P', ABP', -AB'P'.
- Description and notes:
    - This outputs graphs showing the component's expectation value on the y axis and the three tangle on the x axis. Saves to the folder \Figures\SComp and only works for one parameter states.

p2Scomp_tgl_plot(name,n,ln,compnum):
- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - compnum: whether the <S> operator will be broken into two (compnum==2) or four (compnum==4) components.
- Output:
    - Seven line plots are saved in the folder Figures for the max <S> violation, and for each component. For compnum==2, the Svetlichny operator is broken up into the following two componentes: A(BP+B'P') and A(BP'-B'P). If compnum==4, then it is broken up into ABP, AB'P', ABP', -AB'P'.
- Description and notes:
    - This outputs graphs showing the component's expectation value on the y axis and the three tangle on the x axis. Saves to the folder \Figures\SComp and only works for two parameter states.

p1Veccomp_theta_plot(name,n,ln):
- Input:
    - name: name of the state desired

- n: number of guesses
- ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].

- Output:
  - Eighteen scatter (and one line) plots are saved in the folder Figures for the max <S> violation, and for each vector component with respect to the parameter.
- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the vector component's magnitude on the y axis and the theta on the x axis. Only works for one parameter states. Saves to the folder \Figures\VComp.

p1Veccomp_tgl_plot(name,n,ln):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - Eighteen scatter (and one line) plots are saved in the folder Figures for the max <S> violation, and for each vector component with respect to the three tangle.
- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the vector component's magnitude on the y axis and the three tangle on the x axis. Only works for one parameter states. Saves to the folder \Figures\VComp.

p2Veccomp_tgl_plot(name,n,ln):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - Eighteen scatter (and one line) plots are saved in the folder Figures for the max <S> violation, and for each vector component with respect to the three tangle.
- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the vector component's magnitude on the y axis and the three tangle on the x axis. Only works for two parameter states. Saves to the folder \Figures\VComp.

p1vecnumber_theta_plot(name,n,ln,save,clear,color):

- Input:

    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - save: 1 or 0, depending on whether the graph is to be saved or not.
    - clear:1 or 0, depending on whether the previous plot is to be removed or not
    - color: the color the plot will be.
- Output:
    - One scatter plot
- Description and notes:
    - This outputs the number of maximized unique vectors found by S_minimize as a function of theta for a one parameter state. Saves to the folder \Figures.

p1vecnumber_tgl_plot(name,n,ln,save,clear,color):
- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - save: 1 or 0, depending on whether the graph is to be saved or not.
    - clear:1 or 0, depending on whether the previous plot is to be removed or not
    - color: the color the plot will be.
- Output:
    - One scatter plot
- Description and notes:
    - This outputs the number of maximized unique vectors found by S_minimize as a function of three tangle for a one parameter state. Saves to the folder \Figures.

p2vecnumber_tgl_plot(name,n,ln,save,clear,color):
- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
    - save: 1 or 0, depending on whether the graph is to be saved or not.
    - clear:1 or 0, depending on whether the previous plot is to be removed or not
    - color: the color the plot will be.
- Output:
    - One scatter plot
- Description and notes:
    - This outputs the number of maximized unique vectors found by S_minimize as a function of three tangle for a two parameter state. Saves to the folder \Figures.

p1Thetaccomp_theta_plot(name,n,ln):
- Input:
    - name: name of the state desired
    - n: number of guesses
    - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:

- Twelve scatter (and one line) plots are saved in the folder \Figures\AComp for the max <S> violation, and for each vector angle with respect to the theta.

- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the angle for a given vector in theta/pi notation on the y axis and the the state parameter on the x axis. Only works for one parameter states.

p1Thetaccomp_tgl_plot(name,n,ln):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - Twelve scatter (and one line) plots are saved in the folder \Figures\AComp for the max <S> violation, and for each vector angle with respect to the the three tangle.
- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the angle for a given vector in theta/pi notation on the y axis and the the three tangle on the x axis. Only works for one parameter states.

p2Thetaccomp_tgl_plot(name,n,ln):

- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - Thirteen scatter (and one line) plots are saved in the folder \Figures\AComp for the max <S> violation, and for each vector component with respect to the three tangle.
- Description and notes:
  - This outputs graphs with the same axes to be analyzed in an image editing program, such as gimp or photoshop. This allows one to remove the whitespace and compare each component with regards to the others, ensuring that one can see any data hidden behind other trials. Each graph shows the vector angle on the y axis and the three tangle on the x axis. Only works for two parameter states.

p1Scomp_compare_theta_plot(name,n,ln):

- Input:
  - name: name of the state desired
  - n: number of guesses

  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - One scatter plot.
- Description and notes:
  - This outputs a graph which breaks S into 2,4,8, and 1 components, and graphs them side by side with respect to theta. Each is colored differently. It is saved to \Figures\SComp. Only works on one parameter states.

p1Scomp_compare_tgl_plot(name,n,ln):
- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - One scatter plot.
- Description and notes:
  - This outputs a graph which breaks S into 2,4,8, and 1 components, and graphs them side by side,with respect to three tangle. Each is colored differently. It is saved to \Figures\SComp. Only works on one parameter states.

p2Scomp_compare_tgl_plot(name,n,ln):
- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - One scatter plot.
- Description and notes:
  - This outputs a graph which breaks S into 2,4,8, and 1 components, and graphs them side by side with respect to three tangle. Each is colored differently. It is saved to \Figures\SComp. Only works on two parameter states.

p2S_max_phi_plot(name,n,ln,save,clear,color):
- Input:
  - name: name of the state desired
  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - One scatter plot.
- Description and notes:
  - This outputs a graph which plots <S> max versus the second parameter of a state, where the entire first parameter is calculated over. Only works for two parameter states.

p2S_max_theta_plot(name,n,ln,save,clear,color):
- Input:
  - name: name of the state desired

  - n: number of guesses
  - ln: how many steps the parameterization of psi will be broken down into over the range of [0,2pi].
- Output:
  - One scatter plot.
- Description and notes:
  - This outputs a graph which plots <S> max versus the first parameter of a state, where the entire second parameter is calculated over. Only works for two parameter states.

Modification:

The module itself is written in python, but is compiled in cython for speed. While it may not be the most efficient code in the world (if you have a better understanding of computer programming, it would benefit from added efficiency) it does gain some benefit from compiling in C. At any rate, the easiest way to modify the program is to deal with the S_maximize_3q.pyx file. This is a file that cython turns into C code and as such can have both C and python structures.[4] After editing this, simply run *python setup.py build_ext -inplace* in the same folder as the setup.py file to complie the code. Strictly speaking the *-inplace* command is unnecessary, but this ensures that it is built in the local folder, and thus can be easily exported if necessary. In addition, if this documentation is incomplete, or if errors crop up, the above functions are all defined in the S_maximize_3q.pyx file.

[4] See http://docs.cython.org/ for more information about how cython can be used.