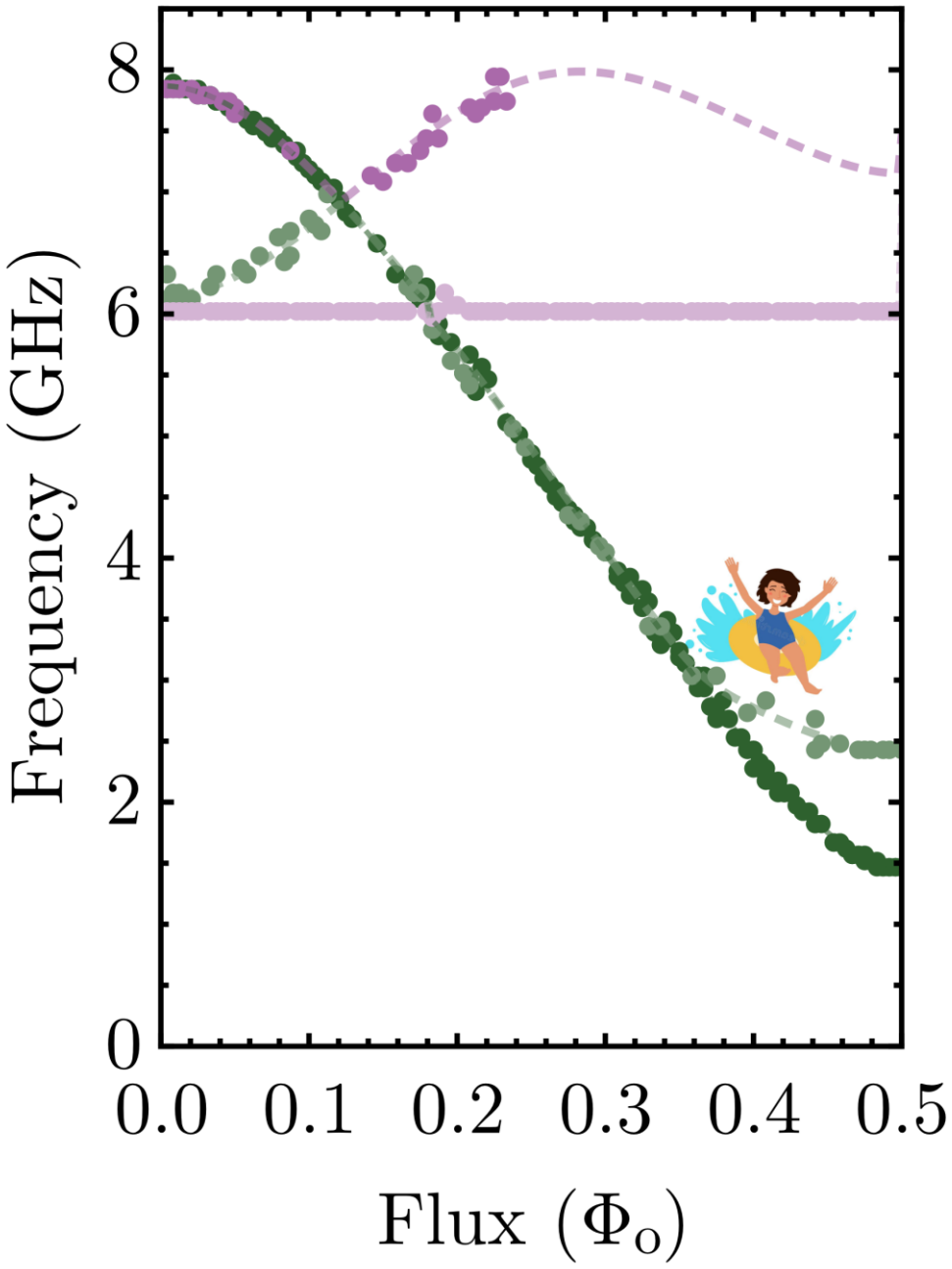


ReadMe – Hamiltonian Optimization

FIT THE FLUXONIUM SPECTRUM

Matilda Peruzzo, Gregory Szep | Hamiltonian Optimization | March 2021

Getting started

You need Julia 1.5+. Follow your platform specific instructions. In order to make the julia executable available from the command line / terminal follow the instructions on adding Julia to your PATH/environment.

Open a command line / terminal in a folder of your choice where you want to install this repo. Make sure Git is installed; if the command is not recognised then download and install it. Now you are ready to download and run this repo:

```
git clone git@github.com:gszep/hamiltonian-optimisation.git
cd hamiltonian-optimisation
julia run.jl
```

Although it is possible to re-run the optimisations by changing parameters in the run.jl script and then re-running from the command line, we recommend using opening run.jl with VSCode + Julia Extension to keep an interactive julia session open, saving you time.

Fluxonium theory

Starting with the fluxonium Hamiltonian:

$$H = 4\,E_C\hat{n}^2 + \frac{1}{2}E_L\hat{\phi}^2 - \left(\hat{\phi} - 2\pi\frac{\Phi_{ext}}{\Phi_0}\right)$$

To solve the hamiltonian we make use of the harmonic oscillators states as a basis as shown in Masluck's thesis.

This will require capping the number of levels used. An appropriate number of levels to use is around 20.

By following the previously cited source one can construct a Hamiltonian matrix, the eigenvalues of which will be the energy leves of the fluxonium. The transitions will be derived by subtracting the lowest level energy from the higher levels.

Solving the eigenvalue problem for different values of the external flux Φ_{ext} one can calculate the fluxonium spectrum.

In order to fit the resonator and coupling one must add to the Hamiltonian

$$H_{int} = \hbar\omega_R\hat{a}^\dagger\hat{a} - ig_C\hat{n}\left(\hat{a} - \hat{a}^\dagger\right) - g_L\hat{\phi}(\hat{a} + \hat{a}^\dagger)$$

To solve such a problem, we can again construct a Hamiltonian matrix. This time the total dimension of the matrix is going to be the number of fluxonium levels multiplied by the number of levels used for the resonator.

For the Hamiltonian matrix we do a tensor product between the previously established fluxonium Hamiltonian and an identity matrix with the dimentions chosen for the resonator. As for the coupling matrix we expand the number operator $\hat{n}$ and the phase operator $\hat{\phi}$ into the creation and annihilation operator of the qubit, $\hat{b}^\dagger$ and $\hat{b}$:

$$\hat{\phi} = \frac{1}{\sqrt{2}}\left(\frac{8E_C}{E_L}\right)^{\frac{1}{4}}\left(\hat{b}^\dagger + \hat{b}\right)$$

$$\hat{n} = \frac{i}{\sqrt{2}}\left(\frac{E_L}{8E_C}\right)^{\frac{1}{4}}\left(\hat{b}^\dagger - \hat{b}\right)$$

The creation and annihilation operators can be expressed in matrix form as:

$$\hat{a}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ \sqrt{1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sqrt{2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{3} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \sqrt{n} & 0 \end{pmatrix}$$

Where n is the dimensionality of the subsystem it acts on.

Once the Hamiltonian matrix is constructed the problem is once again solved by diagonalization and extraction of eigenvalues.

The Hamiltonian-optimization code

PSEUDO CODE

1. Start with data as a list of target points, each consisting of a flux (in units of π) and a corresponding frequency:

$$\begin{matrix} \phi_1 & \nu_1 \\ \phi_2 & \nu_2 \\ \phi_n & \nu_n \end{matrix}$$

2. Solve the Hamiltonian for a set of starting parameters.
3. The algorithm then calculates the difference between each target point and the qubit levels to fit. The number of qubit levels to fit is a parameter and is not the same as the total number of levels of the Hamiltonian but should be the same as the number of levels present in the data.
4. Each target point is assigned to a level based on which level it is less distant from.
5. The problem is optimized to minimize the difference between target points and the assigned qubit levels.

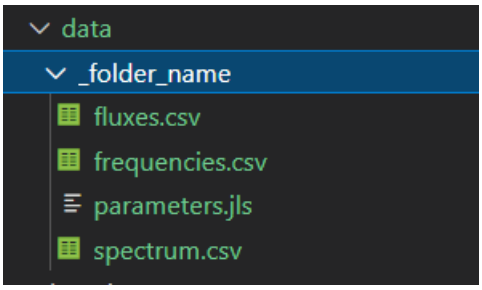
This process can be run for a model that only comprises of a qubit or a model that comprises of qubit and resonator.

IMPORTING THE DATA

The data can be imported either through uploading the vector of points directly or by uploading a color map where the change in intensity corresponds to the qubit level. These two methods are the difference between the two branches of the code.

Master branch

The data is to be placed in the **data** folder where a subfolder is to be placed with the qubit name. In this folder colormap data is to be placed under the name "spectrum.csv" (as a csv file).



When plotting the spectrum data the x and y axis will be constructed from the first and last elements of the fluxes.csv and frequencies.csv. These files can simply be the limit values of the spectrum.csv data.

The name of the folder is to be referenced in the code.

```
16      name = "_folder_name"
17      data_path = joinpath("data",name)
```

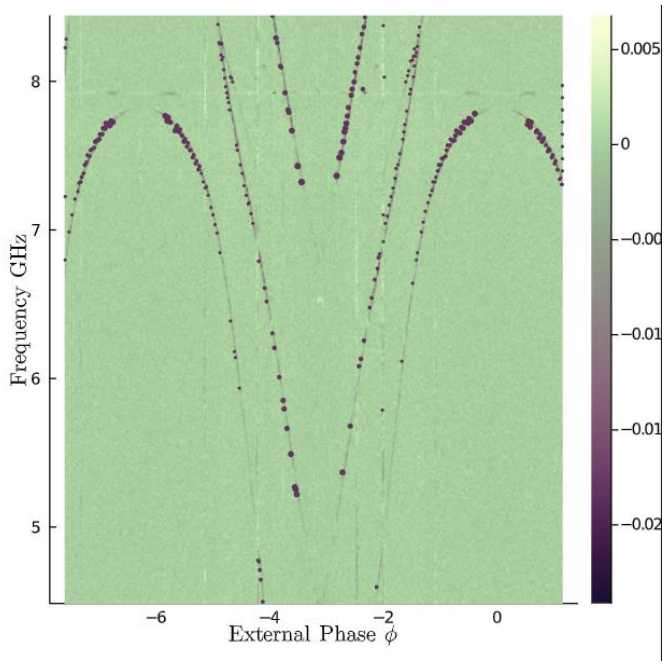
Targets are extracted from the spectrum image using morphological image processing techniques. Repeated dilations, to connect high value pixels representing measurements of the qubit transitions, and erosions, to remove noisy background pixels, are applied. An anisotropic version of these operations is used in an attempt to remove horizontal image artefacts.
The processing of the data is determined by the preprocessing parameters:

```
20      # preprocessing parameters can be updated here
21      threshold =.04,
22      downSample=4
23      , dilations=2, erosions=1,
24      frequencyLimits=(-Inf,6.5),maxTargets = 5000
25  )
```

These include:
Threshold: determines a lower bound for the height of the qubit peak.
Downsample: instructs the code to downsample the final data points.
Dilations/Erosions: number of dilations/erosions to to perform on input image in an effort to clean up noise and sharpen the observed spectral lines. The dilationDirection - by default vertical - and erosionDirection - by default horizontal - parameters are passed to the positional parameter "region" for the ImageMorphology.dilate and ImageMorphology.erode methods respectively.
Frequency limits: will cut off data at frequencies outside the specified window.
Max targets: determines what is the maximum amount of points the processing is allowed to save.

If the total number of points is less than one or more than the max_targets value the code will give an error and some of the processing parameters have to be changed. Once the code has been run once the preprocessing parameters will be saved in the qubit subfolder under the name *parameters.jl*. In future runs of the code if the line specifying a parameter is commented out the value of that parameter in *parameters.jl* will be used instead. Otherwise the corresponding value in *parameters.jl* will be overwritten.

Running the part of the code that extracts the data will return a contour plot of the data in spectrum.csv and points corresponding to the targets that were found as seen below.



Targets branch

In the case that one already has the targets the target branch can be used. For this option the data must be placed in a subfolder of the qubit folder called *uncoupled*. Here the targets are to be placed in two vectors: frequencies.csv and fluxes.csv. These can contain

data from multiple levels, what is important is that the order of the frequency values corresponds to the order of the flux values.

In both cases the data is weighted by the function $e^{-|\sin(\phi)|^2}$, the goal is to prioritize data around the flux sweet spots where the data is more constrictive.

FITTING THE DATA

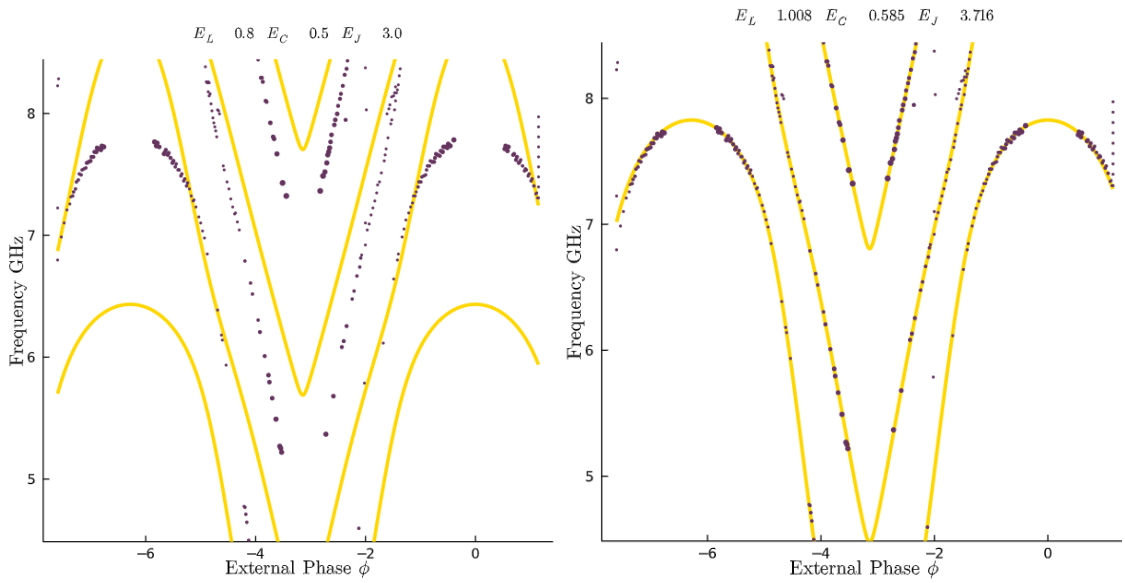
Fitting qubit only

As mentioned in the pseudocode the goal of the algorithm is to minimize the distance between the calculated level and the data points that are closest to it when the algorithm is initialized. The initial parameters and the number of levels to be considered are to be defined by the variables *parameters* and *nlevels* respectively.

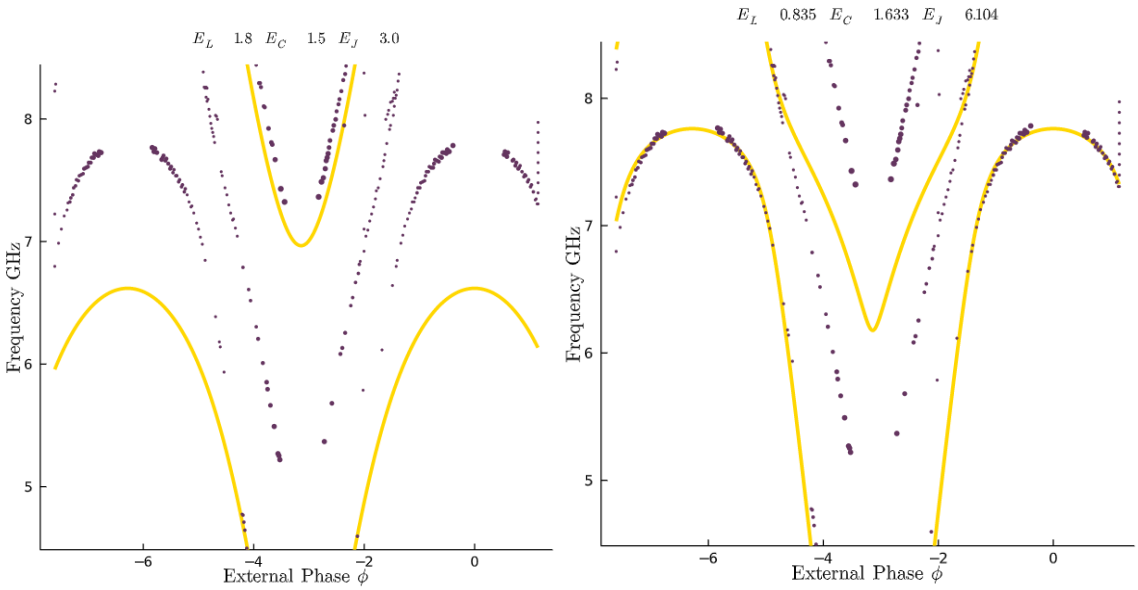
```
48      parameters = ( El=1.80,Ec=1.50,Ej=3.0, Gl=0.0,Gc=0.0,vr=NaN )
49      nlevels = 1:5
```

The optimization is done by way of a Nadler-Mead method.

Shown below is an example of good initial conditions and the result of the fitting.



Below is an example of bad initial conditions that lead the algorithm to fit to wrong parameters.



The previously shown plots are obtained by fitting the data to a model consisting only of the qubit Hamiltonian.

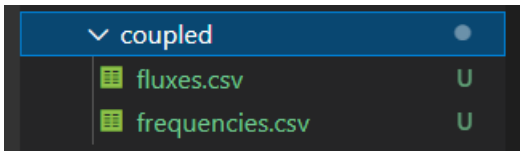
The following is the part of the code relevant to the optimization and saving the fitted parameters.

```
52      result = optimize(
53
54          x->loss(fluxonium, merge(parameters,(El=x[1],Ec=x[2],Ej=x[3])),
55          targets; nlevels=nlevels),
56
57          [ parameters.El, parameters.Ec, parameters.Ej ],
58
59          NelderMead(), Optim.Options(iterations=10^4))
60
61      # update parameters
62      fluxonium_parameters = merge((El=NaN,Ec=NaN,Ej=NaN),result.minimizer)
63      parameters = merge(parameters,fluxonium_parameters)
```

Fitting the qubit and resonator

In the case of qubits not in the ultrastrong coupling regime (where the coupling *g* is small with respect to the qubit frequency) it is easiest to fit the qubit separately to the resonator. When fitting the resonator coupling the process is similar to fitting the qubit.

The data is to be placed in a subfolder called *coupled* in the data folder under the specific qubit name.



The data saved in this folder follows the convention of the previous section. This can contain data of the resonator frequency vs flux as well as qubit level data. For the best result the resonator data is highly recommended. As the qubit parameters will be taken to be the ones found in the previous section the qubit level data is not strictly necessary but it helps narrow down the potential solutions.

Shown is the code responsible for the optimization with indicator to important details.

```
##################################### optimisation
parameters = merge(parameters,(Gl=0.130,Gc=-0.391,vr=5.975 ))
nlevels_coupled = 1:3
result = optimize(

    #x->loss(system, merge(parameters,(Gl=x[1],Gc=x[2],vr=x[3])),
    x->loss(system, merge(parameters,(Gl=x[1],Gc=x[2])),
    targets; nlevels=nlevels_coupled, coupled=true),

    [ parameters.Gl, parameters.Gc],
    #[ parameters.Gl, parameters.Gc, parameters.vr ],
    NelderMead(), Optim.Options(iterations=10^4)
)

# update parameters
coupling_parameters = merge((Gl=NaN,Gc=NaN),result.minimizer)
#coupling_parameters = merge((Gl=NaN,Gc=NaN,vr=NaN),result.minimizer)
```

Levels considered

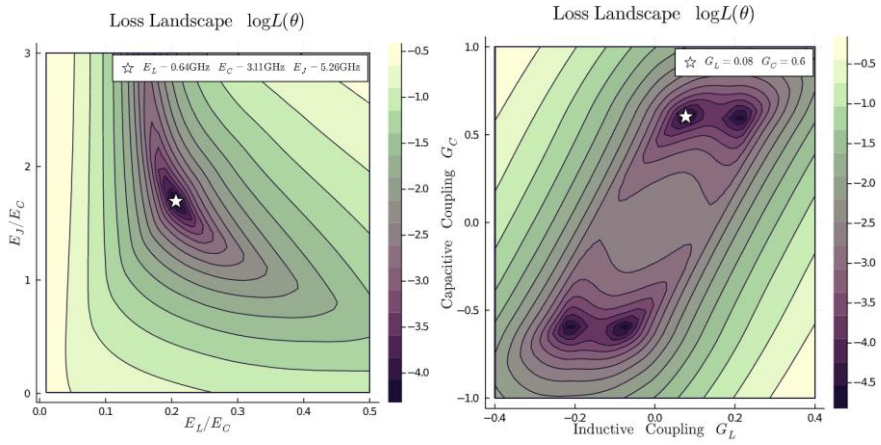
Initial parameters

Uses full Hamiltonian

The code shown will only fit coupling strengths Gl and Gc, to fit the resonator comment the lines with the cloud symbol next to them and uncomment the lines with a speech bubble next to them.

PARAMETER SWEEPS

The code adds the possibility to sweep parameters to look at the uncertainty landscape. The parameter sweeps are plotted in 2D and hence are done for E_J/E_C vs E_L/E_C in the case of the single qubit and for g_C vs g_L for the coupled case. This means that E_C and υ_R are fixed to whatever is already saved under the variable *parameters*.



Pictured above are examples of the two types of parameter sweeps. The star identifies the minimum found. This minimum however doesn't always represent a global minimum. In the case of the coupled system it is possible that multiple minima are found.
Due to symmetries in the Hamiltonian by changing the sign of both g_C and g_L the result of the Hamiltonian remains unchanged. In addition when only resonator data is fed into the code it is possible that additional solutions appear, these are often local minima and may not appear if the data includes qubit levels as well.

SAVING FIGURES

At specific points in the code there is the possibility of saving a figure of the fit or of the loss landscape. These will be saved under the figures folder.

```
75    # save final figure when happy
76    savefig(joinpath("figures",name*".pdf"))
```

```
88    # save final figure when happy
89    savefig(joinpath("figures",name*".uncertainty.pdf"))
```