Delft University of Technology

QuCAT

Quantum circuit analyzer tool in Python

Gely, Mario F.; Steele, Gary A.

DOI
10.1088/1367-2630/ab60f6

Publication date
2020

Document Version
Final published version

Published in
New Journal of Physics

Citation (APA)
Gely, M. F., & Steele, G. A. (2020). QuCAT: Quantum circuit analyzer tool in Python. *New Journal of Physics*, *22*(1), Article 013025. https://doi.org/10.1088/1367-2630/ab60f6

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG IOP Institute of Physics

PAPER • OPEN ACCESS

QuCAT: quantum circuit analyzer tool in Python

To cite this article: Mario F Gely and Gary A Steele 2020 *New J. Phys.* **22** 013025

View the article online for updates and enhancements.

This content was downloaded from IP address 131.180.131.188 on 10/03/2020 at 10:55

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

CrossMark

OPEN ACCESS

RECEIVED
26 September 2019

REVISED
3 December 2019

ACCEPTED FOR PUBLICATION
11 December 2019

PUBLISHED
20 January 2020

Original content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

QuCAT: quantum circuit analyzer tool in Python

Mario F Gely and Gary A Steele

Kavli Institute of NanoScience, Delft University of Technology, P O Box 5046, 2600 GA, Delft, The Netherlands

E-mail: mario.f.gely@gmail.com

Keywords: quantum, Josephson junction, superconducting circuits, normal mode analysis, Python software

Abstract

Quantum circuits constructed from Josephson junctions and superconducting electronics are key to many quantum computing and quantum optics applications. Designing these circuits involves calculating the Hamiltonian describing their quantum behavior. Here we present QuCAT, or 'Quantum Circuit Analyzer Tool', an open-source framework to help in this task. This open-source Python library features an intuitive graphical or programmatical interface to create circuits, the ability to compute their Hamiltonian, and a set of complimentary functionalities such as calculating dissipation rates or visualizing current flow in the circuit.

1. Introduction

Quantum circuits, constructed from superconducting electronics and involving one or more Josephson junctions, have steadily gained prominence in experimental and theoretical physics over the past twenty years. Foremost, they are one of the most successful platforms in the quest to build a quantum computer (Devoret and Schoelkopf 2013). The control that can be gained over their quantum state, and the flexibility in their design have also made these circuits an excellent test-bed to probe fundamental quantum effects (Gu *et al* 2017). They can also be coupled to other systems, such as atoms, spins, acoustic vibrations or mechanical oscillators, acting as a tool to measure and manipulate these systems at a quantum level (Xiang *et al* 2013).

Any application mentioned above generally translates to a desired Hamiltonian, which governs the physics of the circuit. The task of the quantum circuit designer is to determine which circuit components to use, how to inter-connect them, and calculate the corresponding Hamiltonian (Nigg *et al* 2012, Vool and Devoret 2017). Performing this task analytically can be time consuming or even challenging.

Here we present QuCAT, which stands for 'Quantum Circuit Analyzer Tool', an open-source Python framework to help in analyzing and understanding quantum circuits. We provide an easy interface to create and visualize circuits, either programmatically or through a graphical user interface (GUI). A Hamiltonian can then be generated for further analysis in QuTiP (Johansson *et al* 2012, 2013). The current version of QuCAT supports quantization in the basis of normal modes of the linear circuit (Nigg *et al* 2012), making it suited for the analysis of weakly anharmonic circuits with small losses. The properties of these modes: their frequency, dissipation rates, anharmonicity and cross-Kerr couplings can be directly calculated. The user can also visualize the current flows in the circuit associated with each normal mode. The library covers lumped element circuits featuring an arbitrary number of Josephson junctions, inductors, capacitors and resistors. Through equivalent lumped element circuits, certain distributed elements such as waveguide resonators can also be analyzed (see section B.3). The software relies on the symbolic manipulation of the circuits equations, making it reliable even for vastly different circuits and parameters. It also results in efficient parameter sweeps, as analytical manipulations need not be repeated for different circuit parameters. In a few seconds, circuits featuring 10 nodes (or degrees of freedom), corresponding to between 10 and 30 circuit elements can be simulated.

In the main section of this article, we cover the functionalities of the software. We start by showing how to create circuits, first using the GUI, then programmatically. We then demonstrate how to generate the corresponding Hamiltonian. Lastly, we show how to extract the characteristics of the circuit modes: frequencies, dissipation, anharmonicity and cross-Kerr coupling and present a tool to visualize these modes. This main

© 2020 The Author(s). Published by IOP Publishing Ltd on behalf of the Institute of Physics and Deutsche Physikalische Gesellschaft

```
import qucat
circuit = qucat.GUI('netlist.txt')
circuit.show()
```

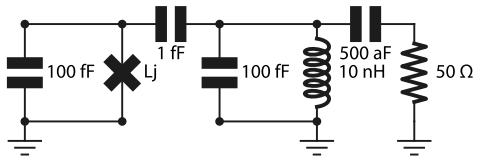

Figure 1. Construction of a circuit: code and output. The circuit used as an example in this section comprises of a transmon qubit on the left, coupled through a 1 fF capacitor to an *LC*-oscillator. Dissipation arises from the capacitive coupling of the *LC*-oscillator to a 50 Ω resistor on the right. After importing the `qucat` package, the `circuit` object is created manually through a graphical user interface (GUI) opened after calling `qucat.GUI('netlist.txt')`. All information necessary to construct the circuit is stored in the text file `netlist.txt`. After closing the GUI, this information is also stored in the variable `circuit`. The `show` method finally displays the circuit.

section will feature as an example the standard circuit of a transmon qubit coupled to a resonator (Koch *et al* 2007). In the appendices, we will first use QuCAT to analyze some recent experiments: a tuneable coupler (Kounalakis *et al* 2018), a multi-mode ultra-strong coupling circuit (Bosman *et al* 2017), a microwave optomechanics circuit (Ockeloen-Korppi *et al* 2016) and a Josephson-ring based qubit (Roy *et al* 2017). We then provide an overview of the circuit quantization method used and the algorithmic methods which implement it. The limitations of these methods regarding weak anharmonicity and circuit size will then be presented. Finally we will explain how to install QuCAT and we provide a summary of all its functions. More tutorials and examples are available on the QuCAT website https://qucat.org/.

2. Circuit construction

Any use of QuCAT will start with importing the `qucat` library

```
import qucat
```

One should then create a circuit. These are named `Qcircuit`, short for 'quantum circuit' in QuCAT. There are two ways of creating a `Qcircuit`: using the GUI, or programmatically.

2.1. Creating a circuit with the GUI

We first cover how to create a circuit with the GUI. This is done through this command

```
circuit = qucat.GUI('netlist.txt')
```

which opens the GUI. The GUI will appear as a separate window, which will block the execution of the rest of the Python script until the window is closed. The user can drag-in and drop capacitors, inductors, resistors or Josephson junctions, or grounds. These components can then be inter-connected with wires. Each change made to the circuit will be automatically be saved in the `'netlist.txt'` file. After closing the GUI, the `Qcircuit` object will be stored in the variable named `circuit` which we will use for further analysis.

2.2. Creating a circuit programmatically

Alternatively, one can create a circuit with only Python code. This is done by creating a list of circuit components with the functions `J`, `L`, `C` and `R` for junctions, inductors, capacitors and resistors respectively. For the circuit of figure 1:

```
circuit_components = [
qucat.C(0,1,100e-15), # transmon
qucat.J(0,1,'Lj'),
qucat.C(0,2,100e-15), # resonator
qucat.L(0,2,10e-9),
qucat.C(1,2,1e-15), # coupling capacitor
```

```
qucat.C(2,3,0.5e-15), # ext. coupl. cap.
qucat.R(3,0,50) # 50 Ohm load
]
```

All circuit components take as first two argument integers referring to the negative and positive node of the circuit components. Here 0 corresponds to the ground node for example. The third argument is either a float giving the component a value, or a string which labels the component parameter to be specified later. Doing the latter avoids performing the computationally expensive initialization of the `Qcircuit` object multiple times when sweeping a parameter. By default, junctions are parametrized by their Josephson inductance $L_j = \phi_0^2/E_j$ where $\phi_0 = \hbar/2e$ is the reduced flux quantum, and E_j (in Joules) is the Josephson energy.

Once the list of components is built, we can create a `Qcircuit` object via the `Network` function

```
circuit = Network(circuit_components)
```

as with a construction via the GUI, the `Qcircuit` object will be stored in the variable named `circuit` which we will use for further analysis.

3. Generating a Hamiltonian

The Hamiltonian of a Josephson circuit is given by

$$\hat{H} = \sum_m \hbar\omega_m \hat{a}_m^\dagger \hat{a}_m + \sum_j \sum_{n\geqslant 2} E_j \frac{(-1)^{n+1}}{(2n)!} \hat{\varphi}_j^{2n}. \tag{1}$$

It is written in the basis of its normal modes. These have an angular frequency ω_m and we write the operator which creates (annihilates) photons in the mode $\hat{a}_m^\dagger$ ($\hat{a}_m$). The cosine potential of each Josephson junction j with Josephson energy E_j has been Taylor expanded to order n for small values of its phase fluctuations $\hat{\varphi}_j$ across it. The phase fluctuations are a function of the annihilation and creation operators of the modes $\hat{\varphi}_j = \sum_m \varphi_{\mathrm{zpf},m,j}(\hat{a}_m^\dagger + \hat{a}_m)$. For a detailed derivation of this Hamiltonian, and the method used to obtain its parameters, see section B.

There are three different parameters that the user should fix

1. The set of modes to include.
2. For each of these modes, the number of excitations to consider.
3. The order of the Taylor expansion.

The more modes and excitations are included, and the higher Taylor expansion order, the more faithful the Hamiltonian will be to physical reality. The resulting increase in Hilbert space size will however make it more computationally expensive to perform further calculations. Typically, larger degrees of anharmonicity require a larger Hilbert space, with a fundamental limitation on the maximum anharmonicity due to the choice of basis. We expand on these topics in section E.2.

Such a Hamiltonian is generated through the method `hamiltonian`. More specifically, this function returns a QuTiP object (Johansson *et al* 2012, 2013), enabling an easy treatment of the Hamiltonian. All QuCAT functions use units of Hertz, so the function is actually returning $\hat{H}/h$.

As an example, we generate a Hamiltonian for the circuit of figure 1 at different values of the Josephson inductance and use QuTiP to diagonalize it and obtain the eigen-frequencies of the system. For a Josephson inductance of 8 nH this is achieved through the commands

```
H = circuit.hamiltonian(
    modes = [0,1],
    excitations = [10,12],
    taylor = 4,
    Lj = 8e-9)
E = H.eigenenergies() # Eigenenergies
```

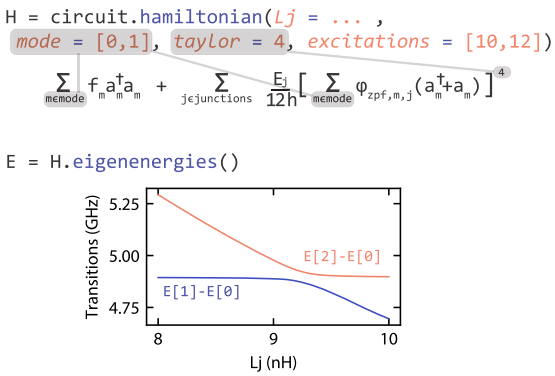

Figure 2. Hamiltonian generation is done by applying the `hamiltonian` method to the `circuit` variable defined in figure 1. The Hamiltonian is expressed in the basis of circuit normal modes m with frequencies $f_m = \omega_m/2\pi$, annihilation operators $\hat{a}_m$, and zero-point phase fluctuations $\varphi_{\mathrm{zpf},m,j}$ across junction j with Josephson energy E_j. The junction nonlinearities are expressed through a Taylor expansion of the cosine potentials, where the user chooses the degree of Taylor expansion. The other arguments are the list of modes to include, the number of excitations to consider for each of these modes, and any unspecified component value, here `Lj`. The returned Hamiltonian is a QuTiP object, giving the user access to an extensive set of tools for further analysis (Johansson *et al* 2012, 2013). As an example, we compute the eigenenergies of the Hamiltonian, and plot the two first transition frequencies, as a function of `Lj`.

(here in units of frequency) using the

QuTiP function eigenenergies

With `modes = [0,1]`, we are specifying that we wish to consider the first and second modes of the circuit. Modes are numbered with increasing frequency, so here we are selecting the two lowest frequency modes of the circuit. With `excitations = [10,12]`, we specify that for mode `0` (`1`) we wish to consider `10` (`12`) excitations. With `taylor = 4`, we are specifying that we wish to expand the cosine potential to fourth order, this is the lowest order which will give an anharmonic behavior. The unspecified Josephson inductance must now be fixed through a keyword argument `Lj = 8e-9`. Doing so avoids initializing the `Qcircuit` objects multiple times during parameter sweeps, as initialization is the most computationally expensive task. We calculate these energies with different values of the Josephson inductance, and the first two transition frequencies are plotted in figure 2, showing the typical avoided crossing seen in a coupled qubit-resonator system.

4. Mode frequencies, dissipation rates, anharmonicities and cross-Kerr couplings

QuCAT can also return the parameters of the (already diagonal) Hamiltonian in first-order perturbation theory

$$\hat{H} = \sum_m \sum_{n\neq m} \left(\hbar\omega_m - A_m - \frac{\chi_{mn}}{2}\right)\hat{a}_m^\dagger \hat{a}_m - \frac{A_m}{2}\hat{a}_m^\dagger \hat{a}_m^\dagger \hat{a}_m \hat{a}_m - \chi_{mn}\hat{a}_m^\dagger \hat{a}_m \hat{a}_n^\dagger \hat{a}_n \tag{2}$$

valid for weak anharmonicity χ_{mn}, $A_m \ll \omega_m$. The physics of this Hamiltonian can be understood by considering that an excitation of one of the circuit modes may lead to current traversing a Josephson junction. This will change the effective inductance of the junction, hence changing its own mode frequency, as well as the mode frequencies of all other modes. This is quantified through the anharmonicity or self-Kerr A_m and cross-Kerr χ_{mn} respectively. When no mode is excited, vacuum-fluctuations in current through the junction give rise to shifted mode energies $\hbar\omega_m - A_m - \sum_n \chi_{mn}/2$.

In a circuit featuring resistors, these anharmonic modes will be dissipative. A mode m will lose energy at a rate κ_m. If these rates are specified in angular frequencies, the relaxation time $T_{1,m}$ of mode m is given by $T_{1,m} = 1/\kappa_m$. A standard method to include the loss rates in a mathematical description of the circuit is through the Lindblad equation (Johansson *et al* 2012), where the losses would be included as collapse operators $\sqrt{\kappa_m}\,\hat{a}_m$.

The frequencies, dissipation rates, and Kerr parameters can all be obtained via methods of the `Qcircuit` object. These methods will return numerical values, and we should always specify the values of symbolically defined circuit parameters as keyword arguments. Lists, or Numpy arrays, can be provided here making it easy to perform parameter sweeps. Additionally, initializing the circuit is the most computationally expensive operation, so this will be by far the fastest method to perform parameter sweeps.

We will assume that we want to determine the parameters of the Hamiltonian (2) for the circuit of figure 1 at different values of L_j. The values for L_j are stored as a Numpy array

```
Lj_list = numpy.linspace(11e-9,9e-9,101)
```

```
f,k,A,chi = circuit.f_k_A_chi(
    Lj = numpy.linspace(11e-9,9e-9))
```

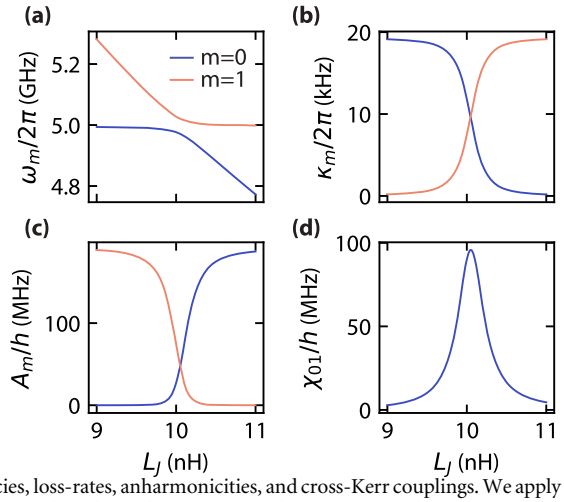

Figure 3. Extracting eigenfrequencies, loss-rates, anharmonicities, and cross-Kerr couplings. We apply the `f_k_A_chi` method to `circuit` defined in figure 1 to obtain a list of eigenfrequencies (`f`), loss-rates (`k`), anharmonicities (`A`), and cross-Kerr couplings (`chi`), for all the normal modes of the circuit. There is one unspecified variable in the circuit, the Josephson inductance L_j, which is here specified with a list of values. In (a), we plot the eigenfrequencies of the two first modes `f[0]` and `f[1]`. In (b), we plot the loss-rates of the same modes `k[0]` and `k[1]`, and in (c) their anharmonicities `A[0]` and `A[1]`. In (d), we plot the cross-Kerr coupling between modes `0` and `1`: `chi[0,1]`.

We can assign the frequency, dissipation rates, self-Kerr, and cross-Kerr parameters to the variables `f`, `k`, `A` and `chi` respectively, by calling

```
f = circuit.eigenfrequencies(Lj = Lj_list)
k = circuit.loss_rates(Lj = Lj_list)
A = circuit.anharmonicities(Lj = Lj_list)
chi = circuit.kerr(Lj = Lj_list)
```

or alternatively through a single function call:

```
f,k,A,chi = circuit.f_k_A_chi(Lj = Lj_list)
```

All values returned by these methods are given in Hertz, not in angular frequency. With respect to the conventional way of writing the Hamiltonian, which we have also adopted in (2), we thus return the frequencies as $\omega_m/2\pi$, the loss rates as $\kappa_m/2\pi$ and the Kerr parameters as A_m/h and χ_{mn}/h. Note that `f`, `k`, `A`, are arrays, where the index `m` corresponds to mode m, and modes are ordered with increasing frequencies. For example, `f[0]` will be an array of length 101, which stores the frequencies of the lowest frequency mode as `Lj` is swept from 11 to 9 nH. The variable `chi` has an extra dimension, such that `chi[m,n]` corresponds to the cross-Kerr between modes `m` and `n`, and `chi[m,m]` is the self-Kerr of mode `m`, which has the same value as `A[m]`. These generated values are plotted in figure 3.

We can also print these parameters in a visually pleasing way to get an overview of the circuit characteristics for a given set of circuit parameters. For a Josephson inductance of 9 nH, this is done through the command

```
circuit.f_k_A_chi(Lj = 10e-9, pretty_print
= True)
```

which will print

```
mode | freq.   | diss.    | anha.   |
 0   |4.99 GHz |9.56 kHz  |10.5 kHz |
 1   |5.28 GHz |94.3 Hz   |189 MHz  |
```

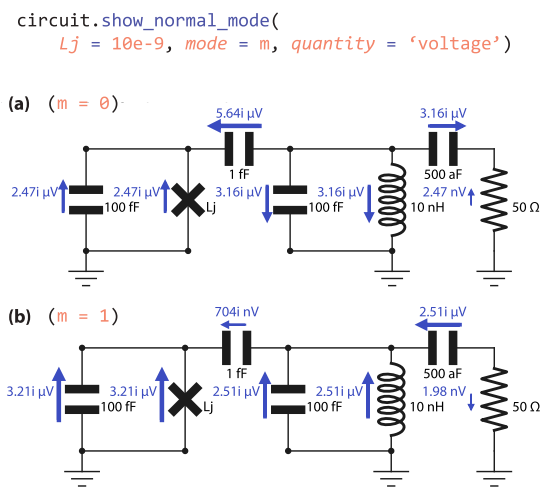

Figure 4. Visualizing normal modes. The `show_normal_mode` method overlays the circuit with arrows representing the voltage across components when the circuit is populated with a single-photon amplitude coherent state. The arrows are annotated with the value of the complex voltage oscillating across a component, where the direction of the arrow indicates the direction of a phase 0 for that component. The absolute value of this annotation corresponds to the zero-point fluctuations of the given quantity across the component. The length and thickness of the arrows scale with the magnitude of the voltage. `show_normal_mode` takes as argument any unspecified circuit parameter, here we specify `Lj = 10e-9` where the two modes undergo an avoided crossing. We plot each mode by specifying `mode = 0` or `mode = 1` and see that for mode `0`, the anti-symmetric mode, the voltage has opposite signs on each side of the coupling capacitor, leading to a larger voltage across the coupler (and hence a larger effective capacitance and lower frequency) than the symmetric mode.

```
Kerr coefficients
diagonal = Kerr
off-diagonal = cross-Kerr
mode |  0       |  1     |
 0   |10.5 kHz |        |
 1   |2.82 MHz |189 MHz |
```

We see that mode 1 is significantly more anharmonic than mode 0, whereas mode 0 has however a higher dissipation. We would expect that mode 1 is thus the resonance which has current fluctuations mostly located in the junction, whilst mode 0 is located on the other side to the coupling capacitor, where it can couple more strongly to the resistor.

Such interpretations can be verified by plotting a visual representation of the normal modes on top of the circuit as explained below. This can be done by plotting either the current, voltage, charge or flux distribution, overlaid on top of the circuit schematic. As shown in figure 4, this is done by adding arrows, representing one of these quantities at each circuit component and annotating it with the value of that component. The annotation corresponds to the complex amplitude, or phasor, of a quantity across the component, if the mode was populated with a single photon amplitude coherent state. The absolute value of this annotation corresponds to the contribution of a mode to the zero-point fluctuations of the given quantity across the component. The direction of the arrows indicates what direction we take for 0 phase for that component.

We note that an independantly developped Julia platform also allows the calculation of normal mode frequencies and dissipation rates for circuits (Scheer and Block 2018).

5. Outlook

We have presented QuCAT, a Python library to automatize and speed up the design process and analysis of superconducting circuits. By facilitating quick tests of different circuit designs, and helping develop an intuition

for the physics of quantum circuits, we also hope that QuCAT will enable users to develop even more innovative circuits.

Possible extensions of the QuCAT features could include black-box impedance components to model distributed components (Nigg *et al* 2012), more precisely modeling lossy circuits (Solgun *et al* 2014, Solgun and DiVincenzo 2015), handling static offsets in flux or charge through DC sources, additional elements such as coupled inductors or superconducting quantum interference devices (SQUIDS) and different quantization methods, enabling for example quantization in the charge or flux basis. The latter would extend QuCAT beyond the scope of weakly-anharmonic circuits.

In terms of performance, QuCAT would benefit from delegating analytical calculations to a more efficient, compiled language, with the exciting prospect of simulating large scale circuits (Kelly *et al* 2019). Note however that there is a strong limitation on the maximum Hilbert space size that one can simulate after extracting the Hamiltonian.

Acknowledgments

We acknowledge Marios Kounalakis for useful discussions and for reading the manuscript. This work was supported by the European Research Council under the European Union's H2020 program [grant numbers 681476-QOM3D, 732894-HOT, 828826-Quromorphic].

Author contributions

MFG developed QuCAT and the underlying theory and methods. MFG wrote this manuscript with input from GAS. GAS supervised the project.

Declarations of interest: none.

Appendix A. Applications

A.1. Designing a microwave filter

In this application we show how QuCAT can be used to design classical microwave components.
We study here a band pass filter made from two *LC* oscillators with the inductor inline and a capacitive shunt to ground. Such a filter can be used to stop a DC bias line from inducing losses, whilst being galvanically connected to a resonator, see for example (Viennot *et al* 2018). In this case we are interested in the loss rate κ of a *LC* resonator connected through this filter to a 50 Ω load, which could emulate a typical microwave transmission line. We want to study how κ varies as a function of the inductance L and capacitance C of its components.

The QuCAT `GUI` function can be used to open the GUI, the user will manually create the circuit, and upon closing the GUI a Qcircuit object is stored in the variable `filtered_cavity`. By calling the method `show`, we display the circuit as shown in figure A1(a). These steps are accomplished with the code

```
# Open the GUI and manually build the
  circuit
filtered_cavity = qucat.GUI('netlist.txt')
# Display the circuit
filtered_cavity.show()
```

We can then access the loss rates of the different circuit modes through the method `loss_rates`. Since the values of C and L were not specified in the construction of the circuit, their values have to be passed as keyword arguments upon calling `loss_rates`. For example, the loss rate for a 1 pF capacitor and 100 nH inductor is obtained through

```
# Loss rates of all modes
k_all = filtered_cavity.loss_rates(C = 1e-12, L = 100e-9)
```

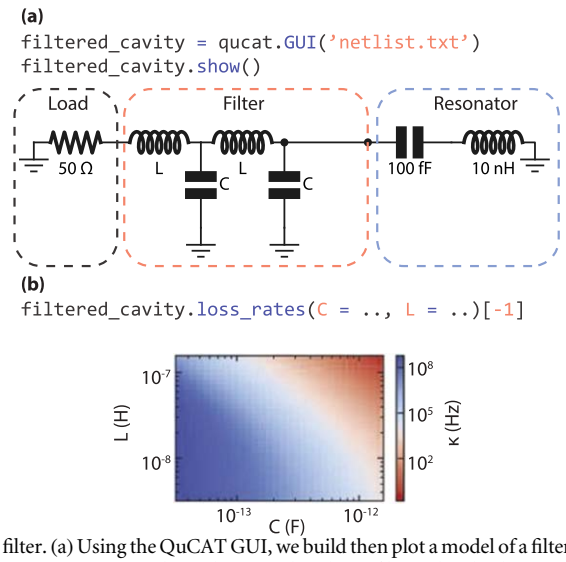

Figure A1. Design of a microwave filter. (a) Using the QuCAT GUI, we build then plot a model of a filtered cavity. A 50 Ω load, representing a cable, is connected to an *LC* resonator through two *LC* band pass filters. (b) The dissipation rate of the resonator is plotted as a function of inductance and capacitance of the filter using the `loss_rates` method.

```
# Resonator loss rate
k = k_all[-1]
```

Since the filter capacitance and inductance is large relative to the capacitance and inductance of the resonator, the modes associated with the filter will have a much lower frequency. We can thus access the loss rate of the resonator by always selecting the last element of the array of loss rates with the command `k_all[-1]`. The dissipation rates for different values of the capacitance and inductance are plotted in figure A1(b).

A.2. Computing optomechanical coupling

In this application, we show how QuCAT can be used for analyzing another classical system, that of microwave optomechanics. One common implementation of microwave optomechanics involves a mechanically compliant capacitor, or drum, embedded in one or many microwave resonators (Teufel *et al* 2011). One quantity of interest is the single-photon optomechanical coupling. This quantity is the change in mode frequency ω_m that occurs for a displacement x_{zpf} of the drum (the zero-point fluctuations in displacement)

$$g_0 = x_{\mathrm{zpf}} \frac{\partial \omega_m}{\partial x}. \tag{A1}$$

The change in mode frequency as the drum head moves $\partial\omega_m/\partial x$ is not straightforward to compute for complicated circuits. One such example is that of (Ockeloen-Korppi *et al* 2016), where two microwave resonators are coupled to a drum via a network of capacitances as shown in figure A2(a). Here, we will use QuCAT to calculate the optomechanical coupling of the drums to both resonator modes of the circuit.

We start by reproducing the circuit of figure A2(a), excluding the capacitive connections on the far left and right. This is done via the GUI opened with the `qucat.GUI` function. Upon closing the GUI, the resulting Qcircuit is stored in the variable `OM`, and the `show` method is used to display the schematic of figure A2(a). These steps are accomplished with the code below

```
# Open the GUI and manually build the
  circuit
OM = qucat.GUI('netlist.txt')
# Display the circuit
OM.show()
```

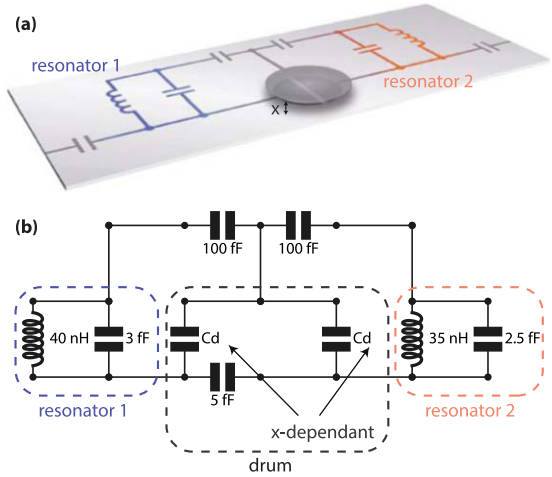

Figure A2. Example of an optomechanical system. (a) Schematic of the device, adapted from (Ockeloen-Korppi *et al* 2016) under a CC BY 3.0 license. Two resonators are connected through a network of capacitances and a mechanically compliant capacitor (drum). (b) QuCAT reconstruction of the circuit. By specifying a label for the mechanically compliant capacitances, we have the possibility to compute the eigenfrequencies ω_m with the method `eigenfrequencies` for small variation in $C_d(x)$. This enables an easy computation of the optomechanical coupling $\propto \mathrm{d}\omega_m/\mathrm{d}x$.

We use realistic values for the circuit components without trying to be faithful to (Ockeloen-Korppi *et al* 2016), the aim of this section is to illustrate a method to obtain g_0. Crucially, the mechanically compliant capacitors have been parametrized by the symbolic variable C_d. We can now calculate the resonance frequencies of the circuit with the method `eigenfrequencies` as a function of a keyword argument C_d.

The next step is to define an expression for C_d as a function of the mechanical displacement x of the drum head with respect to the immobile capacitive plate below it.

```
def Cd(x):
  # Radius of the drumhead
  radius = 10e-6
  # Formula for half a circular parallel
    plate capacitor
  return eps*pi*radius**2/x/2
```

where `pi` and `eps` have been set to the values of π and the vacuum permittivity respectively. We have divided the usual formula for parallel plate capacitance by 2 since, as shown in figure A2(a), the capacitive plate below the drum head is split in two electrodes. We are now ready to compute g_0. Following (Teufel *et al* 2011), we assume the rest position of the drum to be $D = 50$ nm above the capacitive plate below. And we assume the zero-point fluctuations in displacement to be $x_{\mathrm{zpf}} = 4$ fm. We start by differentiating the mode frequencies with respect to drum displacement using a finite differences formula

```
# drum-capacitor gap
D = 50e-9
# difference quotient
h = 1e-18
# derivative of eigenfrequencies
G = (OM.eigenfrequencies(Cd =
  Cd(D+h))-OM.eigenfrequencies(Cd =
  Cd(D)))/h
```

G is an array with values 2.3×10^{16} Hz m^{-1} and 3.6×10^{16} Hz m^{-1} corresponding to the lowest and higher frequency modes respectively. Multiplying these values with the zero-point fluctuations

```
# zero-point fluctuations
x_zpf = 4e-15
g_0 = G*x_zpf
```

yields couplings of 96 and 147 Hz. The lowest frequency mode thus has a 96 Hz coupling to the drum.

If we want to know to which part of the circuit (resonator 1 or 2 in figure A2) this mode pertains, we can visualize it by calling

```
OM.show_normal_mode(
  mode = 0,
  quantity = 'current',
  Cd = Cd(D))
```

and we find that the current is majoritarily located in the inductor of resonator 1.

A.3. Convergence in multi-mode cQED

In this section we use QuCAT to study the convergence of parameters in the first order Hamiltonian (equation (2)) of an ultra-strongly coupled multi-mode circuit QED system.

Using a length of coplanar waveguide terminated with engineered boundary conditions is a common way of building a high quality factor microwave resonator. One implementation is a $\lambda/4$ resonator terminated on one end by a large shunt capacitor, acting as a near-perfect short circuit for microwaves such that only a small amount of radiation may enter or leave the resonator. On the other end one places a small capacitance to ground: an open circuit. The shunt capacitor creates a voltage node, and at the open end the voltage is free to oscillate. This resonator hosts a number of normal modes, justifying its lumped element equivalent circuit: a series of LC oscillators with increasing resonance frequency (Gely *et al* 2017). Here, we study such a resonator with a transmon circuit capacitively coupled to the open end. In particular we consider this coupling to be strong enough for the circuit to be in the multi-mode ultra-strong coupling regime as studied experimentally in (Bosman *et al* 2017) and theoretically in (Gely *et al* 2017). The particularity of this regime is that the transmon has a considerable coupling to multiple modes of the resonator. It then becomes unclear how many of these modes to consider for a realistic modeling of the system. This regime is reached by maximizing the coupling capacitance of the transmon to the resonator and minimizing the capacitance of the transmon to ground. The experimental device accomplishing this is shown in figure A3(a), with its schematic equivalent in figure A3(b), and the lumped-element model in figure A3(c).

We will use QuCAT to track the evolution of different characteristics of the system as the number of considered modes N increases. For this application, programmatically building the circuit is more appropriate than using the GUI. We start by defining some constants

```
# fundamental mode frequency of the
  resonator
f0 = 4.603e9
w0 = f0*2.*numpy.pi
# characteristic impedance of the resonator
Z0 = 50
# Josephson energy (in Hertz)
Ej = 18.15e9
# Coupling capacitance
Cc = 40.3e-15
```

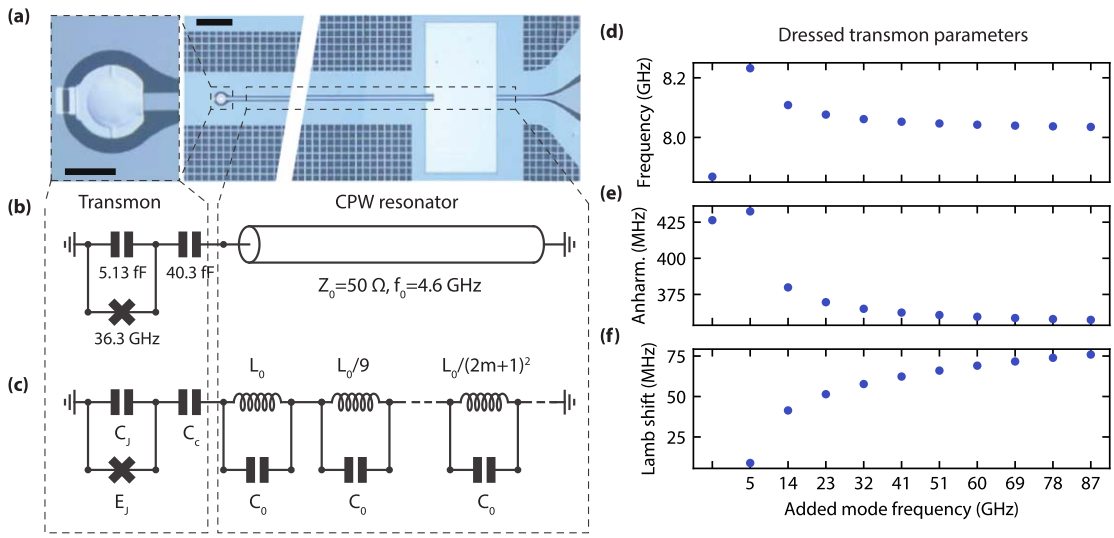

Figure A3. Convergence in multi-mode cQED. (a) Optical micrograph of the device studied in this example, adapted from (Bosman *et al* 2017) under a CC BY 4.0 license. Light-blue corresponds to superconductor, dark blue to an insulating substrate. On the left we see a vacuum-gap transmon: a capacitor plate suspended over the end of a coplanar-waveguide (CPW) resonator shorted to ground through two Josephson junctions. The scale bar corresponds to 30 μm. On the right a CPW $\lambda/4$ resonator, capacitively coupled to the transmon on one side and shorted to ground through a large shunt capacitor on the other. The scale bar corresponds to 100 μm. (b) Circuit schematic of the device. The CPW resonator hosts a number of modes, and is equivalent to a series assembly of *LC* oscillators shown in (c). This circuit is built programmatically in QuCAT, and the qubit parameters are extracted for different total numbers of modes. In (d) and (e) we plot the transmon mode frequency $\omega_t/2\pi$ and anharmonicity A_t/h, where t refers to the transmon-like mode, using the methods `eigenfrequencies` and `anharmonicities` respectively. In (f) we plot the shift defined in (Gely *et al* 2018) as the Lamb shift: the shift in transmon frequency (following equation (2)) due solely to the vacuum-fluctuations in the other modes $\frac{1}{2}\sum_{m\neq t}\chi_{t,m}$, obtained with the `kerr` method. These calculations allow the user to gauge how many modes are relevant to the physics of the circuit.

```
# Capacitance to ground
Cj = 5.13e-15


# Capacitance of all resonator modes
C0 = numpy.pi/4/w0/Z0
# Inductance of first resonator mode
L0 = 4*Z0/numpy.pi/w0
```

we can then generate a Qcircuit we name `mmusc`, as an example here with $N = 10$ modes.

```
# Initialize list of components for
  Transmon and coupling capacitor
netlist = [
  qucat.J(12,1,Ej,use_E = True),
  qucat.C(12,1,Cj),
  qucat.C(1,2,Cc)]


# Add 10 oscillators
for m in range(10):
```

```
  # Nodes of mth oscillator
  node_minus = 2 + m
  node_plus = (2 + m + 1)
  # Inductance of mth oscillator
  Lm = L0/(2*m + 1)**2
  # Add oscillator to netlist
  netlist = netlist + [
   qucat.L(node_minus,node_plus,Lm),
   qucat.C(node_minus,node_plus,C0)]


# Create Qcircuit
mmusc = qucat.Network(netlist)
```

Note that `12` is the index of the ground node.

We can now access some parameters of the system. Only the first mode of the resonator has a lower frequency than the transmon. The transmon-like mode is thus indexed as mode `1`. Its frequency is given by

```
mmusc.eigenfrequencies()[1]
```

and the anharmonicity of the transmon, computed from first order perturbation theory (see equation (2)) with

```
mmusc.anharmonicities()[1]
```

Finally the Lamb shift, or shift in the transmon frequency resulting from the zero-point fluctuations of the resonator modes, is given following equation (2) by the sum of half the cross-Kerr couplings between the transmon mode and the others

```
lamb_shift = 0
K = mmusc.kerr()
for m in range(10):
  if m! = 1:
   lamb_shift = lamb_shift + K[1][m]/2
```

These parameters for different total number of modes are plotted in figures A3(d)–(f).

From this analysis, we find that as we reach 10, the plotted parameters are converging. Surprisingly, adding even the highest modes significantly modifies the total Lamb shift of the Transmon despite large frequency detunings.

A.4. Modeling a tuneable coupler

In this section, we study the circuit of (Kounalakis *et al* 2018) where two transmon qubits are coupled through a tuneable coupler. This tuneable coupler is built from a capacitor and a SQUID. By flux biasing the SQUID, we change the effective Josephson energy of the coupler, which modifies the coupling between the two transmons. We will present how the normal mode visualization tool helps in understanding the physics of the device. Secondly, we will show how a Hamiltonian generated with QuCAT accurately reproduces experimental measurements of the device.

We start by building the device shown in figure A4(a). More specifically, we are interested in the part of the device in the dashed box, consisting of the two transmons and the tuneable coupler. The other circuitry, the flux line, drive line and readout resonator could be included to determine external losses, or the dispersive coupling of the transmons to their readout resonator. We will omit these features for simplicity here. After opening the

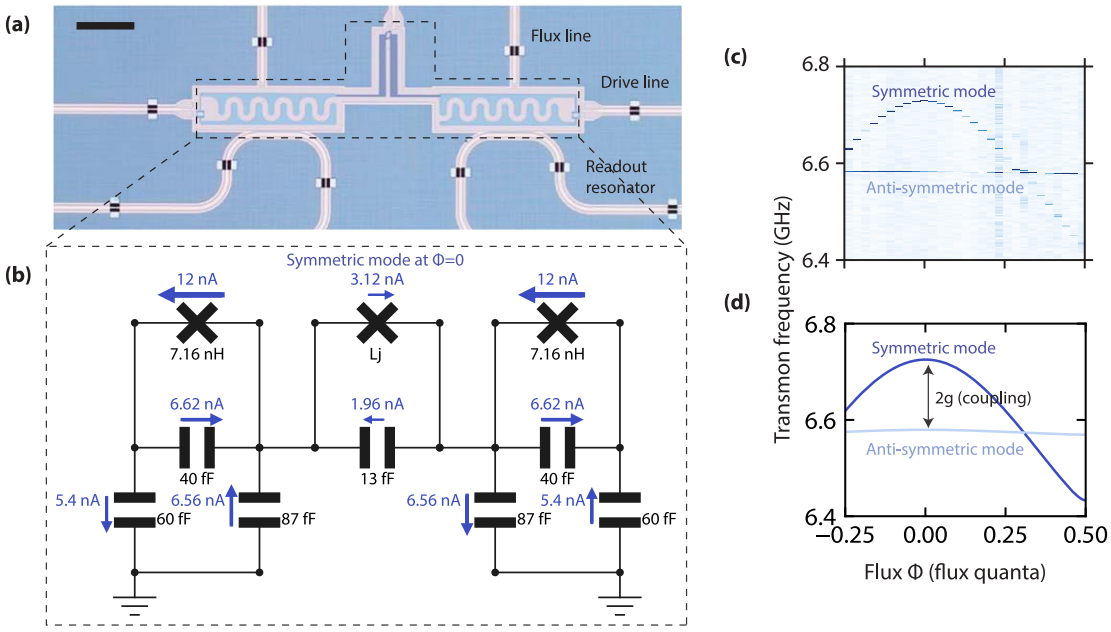

Figure A4. Tuneable coupler circuit analysis. (a) Optical micrograph of the device studied in this example, adapted from (Kounalakis *et al* 2018) under a CC BY 4.0 license. We will omit the flux lines, drive lines and readout resonators for simplicity in this example, and concentrate on the part of the device in the dashed box. The circuit consists of two near-identical transmon qubits coupled through a third 'coupler' transmon. Scale bar corresponds to 200 μm. (b) Equivalent lumped-element circuit constructed with the QuCAT GUI and displayed using the `show_normal_mode` method. This method has overlaid the circuit with the currents flowing through the components when the highest frequency mode is populated with a single-photon-amplitude coherent state. Most of the current is located in the resonantly coupled transmons rather than the coupler, and the fact that the coupled transmons are identical leads to the symmetry on each side of the coupler. This mode is called symmetric since the current in both coupled transmons flows in the same direction. The net current through the coupling junction makes the mode frequency sensitive to changes in the coupling junction inductance tuned with a SQUID. The change in symmetric mode frequency is shown in the experimental measure of the response frequencies in (c) (adapted from (Kounalakis *et al* 2018) under a CC BY 4.0 license), and in the diagonalization of the Hamiltonian generated from QuCAT in (d).

GUI with the `qucat.GUI` function, manually constructing the circuit, then closing the GUI, the resulting `Qcircuit` is stored in a variable `TC`.

```
TC = qucat.GUI('netlist.txt')
```

The inductance L_j of the junction which models the SQUID is given symbolically, and will have to be specified when calling `Qcircuit` functions. Since L_j is controlled through flux ϕ in the experiment, we define a function which translates ϕ (in units of the flux quantum) to L_j

```
def Lj(phi):
  # maximum Josephson energy
  Ejmax = 6.5e9
  # junction asymmetry
  d = 0.0769
  # flux to Josephson energy
  Ej = Ejmax*numpy.cos(pi*phi)
   *numpy.sqrt(1 + d**2
   *numpy.tan(pi*phi)**2)
  # Josephson energy to inductance
  return (hbar/2/e)**2/(Ej*h)
```

where pi, h, hbar, e were assigned the value of π, Plancks constant, Plancks reduced constant and the electron charge respectively.

By visualizing the normal modes of the circuit, we can understand the mechanism behind the tuneable coupler. We plot the highest frequency mode at $\phi = 0$, as shown in figure A4(b)

```
TC.show_normal_mode(mode = 2,
  quantity = 'current',
  Lj = Lj(0))
```

This mode is called symmetric since the currents flow in the same direction on each side of the coupler. This leads to a net current through the coupler junction, such that the value of L_j influences the oscillation frequency of the mode. Conversely, if we plot the anti-symmetric mode instead, where currents are flowing away from the coupler in each transmon, we find a current through the coupler junction and capacitor on the order of 10^{-21} A. This mode frequency should not vary as a function of L_j. When the bare frequency of the coupler matches the coupled transmon frequencies, the coupler acts as a band-stop filter, and lets no current traverse. At this point, both symmetric and anti-symmetric modes should have identical frequencies.

In figure A4(c) this effect is shown experimentally through a measure of the first transitions of the two nonlinear modes. One is tuned with flux (symmetric mode), the other barely changes (anti-symmetric mode). We can reproduce this experiment by generating a Hamiltonian with QuCAT and diagonalizing it with QuTiP for different values of the flux. For example, at 0 flux, the two first two transition frequencies f1and f2 can be generated from

```
# generate a Hamiltonian
H = TC.hamiltonian(Lj = Lj(phi = 0),
  excitations = [7, 7],
  taylor = 4,
  modes = [1, 2])
# diagonalize the Hamiltonian
ee=H.eigenenergies()
f1 = ee[1]-ee[0]
f2 = ee[2]-ee[0]
```

f1 and f2 is plotted in figure A4(d) for different vales of flux and closely matches the experimental data. Note that we have constructed a Hamiltonian with modes 1 and 2, excluding mode 0, which corresponds to oscillations of current majoritarily located in the tuneable coupler. One can verify this fact by plotting the distribution of currents for mode 0 using the show_normal_mode method.

This experiment can be viewed as two 'bare' transmon qubits coupled by the interaction

$$\hat{H}_{\text{int}} = g\sigma_x^L\sigma_x^R, \tag{A2}$$

where left and right transmons are labeled L and R and σ_x is the x Pauli operator. The coupling strength g reflects the rate at which the two transmons can exchange quanta of energy. If the transmons are resonant a spectroscopy experiment reveals a hybridization of the two qubits, which manifests as two spectroscopic absorption peaks separated in frequency by $2g$. From this point of view, this experiment thus implements a coupling which is tuneable from an appreciable value to near 0 coupling.

A.5. Studying a Josephson-ring-based qubit

In this section, we demonstrate the ability for QuCAT to analyze more complex circuits. The experiment of (Roy *et al* 2017) features a Josephson ring geometry, which is a Wheatstone-bridge-like circuit, typically difficult to analyze as it cannot be decomposed in series and parallel connections. We consider the coupling of this ring to two lossy modes of a cavity, bringing the total number of modes in the circuit to 5. We aim to understand the key feature of this circuit: that one qubit-like mode acts as a quadrupole with little coupling to the resonator modes.

The studied device consists of a 3D cavity (figure A5(a)) hosting a number of microwave modes, in which is positioned a chip patterned with the trimon circuit. The trimon circuit has four capacitive pads in a cross shape (figure A5(b)) which have an appreciable coupling between each other making up the capacitance of the trimon

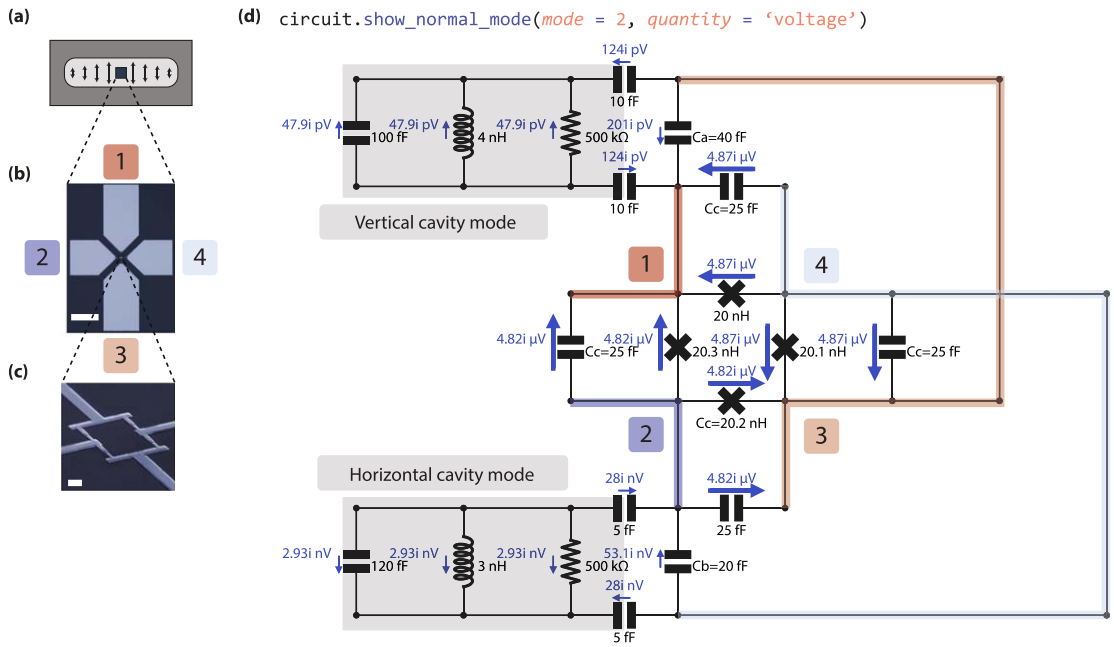

Figure A5. Trimon device and Purcell-decay-protected mode visualization. (a) Schematic of the cross-cut of a 3D microwave cavity. Dark gray shows metal whilst light gray show the hollowed out section forming the cavity. Arrows represent the electric field of the TE_{101}, or 'vertical' cavity mode. In the cavity is placed a chip hosting the trimon circuit shown in the optical micrograph (b). The circuit has 4 capacitive pads labeled from 1 to 4. These pads are connected by the Josephson junction ring shown in the scanning electron microscope image (c). Scale bars correspond 200 and 2 μm for panels (b) and (c) respectively. (d) Lumped-element equivalent circuit of the device constructed using the QuCAT GUI and displayed with `show_normal_mode`. The four pads of the trimon are color-coded to match (b). The capacitor C_a formed by pads 1 and 3 forms an electrical dipole which couples to a vertical cavity mode, and the capacitor C_b formed by pads 2 and 4 forms an electrical dipole which couples to modes with horizontal electric fields. The `show_normal_mode` overlays the voltage across different components if a single-photon amplitude coherent state was populating mode 2. This mode has a particularity that the voltage is concentrated across the junctions and their parallel capacitors without leading to a buildup of voltage across the capacitors C_a or C_b. This decouples mode 2 from the cavity mode decay (no Purcell effect) whilst the presence of voltage fluctuations across the junctions will ensure cross-Kerr coupling to the other modes of the system. Concerning panels (a)–(c): reprinted figures with permission from Roy *et al* (2017). Copyright 2017 by the American Physical Society.

qubit modes. The two vertically (horizontally) positioned pads will couple to modes of the 3D cavity featuring vertical (horizontal) electric fields. We will consider both a vertical and a horizontal cavity mode in our model. We number these pads from 1 to 4 as displayed in figure A5(b). Each pad is connected to its two nearest neighbors by a Josephson junction (figure A5(c)), forming a Josephson ring.

Using the QuCAT GUI, we build a lumped element model of this device, generating a `Qcircuit` object we store in the variable `trimon`.

```
trimon = qucat.GUI('netlist.txt')
```

The cavity modes are modeled as *RLC* oscillators with each plate of their capacitors capacitively coupled to a pad of the trimon circuit. The junction inductances are assigned different values, first to reflect experimental reality, but also to avoid infinities arising in the QuCAT analysis. Indeed, the voltage transfer function of this Josephson ring between nodes 1, 3 and nodes 2, 4 will be exactly 0, which will cause errors when initializing the `Qcircuit` object. Component parameters are chosen to only approximatively match the experimental results of (Roy *et al* 2017), the objective here is to demonstrate QuCAT features rather than accurately model the experiment.

The particularity of this circuit is that it hosts a quadrupole mode. It corresponds here to the second highest frequency mode and can be visualized by calling

```
trimon.show_normal_mode(
  mode = 2,
  quantity = 'voltage')
```

the result of which is displayed in figure A5(d). The voltage oscillations are majoritarily located in the junctions, indicating this is not a cavity mode, but a mode of the trimon circuit. Crucially, the polarity of voltages across the junctions is such that the total voltage between pads 1 and 3 and the total voltage across pads 2 and 4 is 0,

```
trimon.f_k_A_chi(pretty_print = True)

mode |   freq.   |   diss.   |   anha.   |
   0 | 4.23 GHz |   920 Hz | 68.8 MHz |
   1 | 5.14 GHz |   540 Hz |  102 MHz |
   2 | 7.09 GHz |  580 mHz |  194 MHz |
   3 | 7.78 GHz | 1.52 MHz |   654 Hz |
   4 | 8.31 GHz |  1.3 MHz |   313 Hz |
```

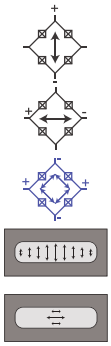

Figure A6. Other modes of the Trimon. Using the `f_k_A_chi` method together with the `pretty_print` option gives the user an overview of the different modes frequencies, dissipations rates and levels of anharmonicity. Here we have overlaid the output of the method with schematics of the corresponding trimon and cavity modes adapted from (Roy *et al* 2017). One can identify a mode to the schematic by observing where the currents or voltages are mostly located in the circuit using the `show_normal_mode` method as in figure A5(d). Since the only resistors of the circuit are located in the cavity modes, all dissipation in transmon modes 0 through 2are due to the Purcell effect. Mode 2 is better protected from this effect by 3 orders of magnitude with respect to the two other transmon modes. Concerning schematics: reprinted figures with permission from Roy *et al* (2017). Copyright 2017 by the American Physical Society.

warranting the name of 'quadrupole mode'. Due to the orientation of the chip in the cavity, the vertically and horizontally orientated cavity modes will only be sensitive to voltage oscillations across pads 1 and 3 or 2 and 4. This ensures that the mode displayed here is decoupled from the cavity modes, and from any loss channels they may incur. We can verify this fact by computing the losses of the different modes, and comparing the losses of mode 2 to the other qubit-like modes of the circuit. We perform this calculation by calling

```
trimon.f_k_A_chi(pretty_print = True)
```

which will calculate and return the loss rates of the modes, along with their eigenfrequencies, anharmonicities and Kerr parameters. Setting the keyword argument `pretty_print` to `True` prints a table containing all this information, which is shown in figure A6. To be succinct, we have not shown the table providing the cross-Kerr couplings. By using the `show_normal_mode` method to plot all the other modes of the circuit, and noting where currents or voltages are majoritarily located, we can identify each mode with the schematics provided in figure A6. The three lowest frequency modes are located in the trimon chip, and we notice that as expected the quadrupole mode 2 has a loss rate (due to resistive losses in the cavity modes) which is three orders of magnitude below the other two. Despite this apparent decoupling, the quadrupole mode will still be coupled to both cavity mode through the cross-Kerr coupling, given by twice the square-root of the product of the quadrupole and cavity mode anharmonicities.

Appendix B. Circuit quantization overview

In this section we summarize the quantization method used in QuCAT, which is an expansion on the work of (Nigg *et al* 2012). This approach is only valid in the weak anharmonic limit, where charge dispersion is negligible. See Nigg *et al* (2012) or section E.2 for a detailed discussion of this condition.

The idea behind the quantization method is as follows. We first consider the 'linearized' circuit. This is a circuit where the junctions are replaced by their Josephson inductances $L_j = \phi_0^2/E_j$. Where E_j is the Josephson energy and the reduced flux quantum is given by $\phi_0 = \hbar/2e$. We determine the oscillation frequencies and dissipation rates of the different normal modes of this linearized circuit. Then, we calculate the amplitude of phase oscillations across each junction when a given mode is excited. This will determine how nonlinear each mode is. All this information will finally allow us to build a Hamiltonian for the circuit.

B.1. Circuit simplification to series of *RLC* resonators (Foster circuit)

The eigenfrequencies and nonlinearity of each mode is obtained by transforming the linearized circuit to a geometry we can easily analyze. We will first describe this process assuming there is only a single junction in the circuit, the case of multiple junctions will follow. We consider the example circuit of figure 1. After replacing the junction with its Josephson inductance, we determine the admittance $Y_j(\omega) = I_j(\omega)/V_j(\omega)$ evaluated at the nodes of the junction. This admittance is the inverse of the impedance measured at the nodes of the junction. It relates the amplitude $|V_j|$ and phase $\theta(V_j)$ of the voltage oscillating at frequency ω that would build up across the

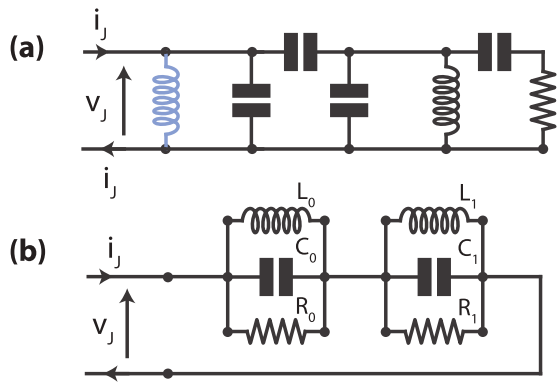

Figure B1. Example of equivalent circuit construction to prepare for quantization. We use the same example as used in figures 1–4. (a) The circuit is linearized by replacing the junction with an inductance L_j. The circuit is characterized at the nodes of the junction by its admittance Y_j. (b) In the limit of small dissipation, this circuit is equivalent to a series combination of *RLC* resonators.

junction if one would feed a current oscillating at ω with amplitude $|I_j|$ and phase $\theta(I_j)$ to one of its nodes through a infinite impedance current source. In figure B1(a) we show a schematic describing this quantity. In the case where all normal modes of the circuit have small dissipation rates, this circuit has an approximate equivalent shown in figure B1(b), consisting of a series of *RLC* resonators (Solgun *et al* 2014). By equivalent, we mean that the admittance Y_j of the circuit is approximatively equal to that of a series combination of *RLC* resonators

$$Y_j(\omega) \simeq \frac{1}{\sum_m 1/Y_m(\omega)} \tag{B1}$$

which each have an admittance

$$Y_m(\omega) = \frac{1}{\mathrm{i}L_m\omega} + \mathrm{i}C_m\omega + \frac{1}{R_m}. \tag{B2}$$

Each *RLC* resonator represents a normal mode of the circuit, with resonance frequency $\omega_m = 1/\sqrt{L_m C_m}$, and dissipation rate $\kappa_m = 1/R_m C_m$. Since this equivalent circuit comes from an extension of Foster's reactance theorem (Foster 1924) to lossy circuits, we call this the Foster circuit.

B.2. Hamiltonian of the Foster circuit

The advantage of this circuit form, is that it is easy to write its corresponding Hamiltonian following standard quantization methods (see Vool and Devoret 2017). In the absence of junction nonlinearity, it is given by the sum of the Hamiltonians of the independent harmonic *RLC* oscillators:

$$\sum_m \hbar\omega_m \hat{a}_m^\dagger \hat{a}_m. \tag{B3}$$

The annihilation operator $\hat{a}_m$ for photons in mode m is related to the expression of the phase difference between the two nodes of the oscillator

$$\hat{\varphi}_{m,j} = \varphi_{\mathrm{zpf},m,j}(\hat{a}_m + \hat{a}_m^\dagger),$$
$$\varphi_{\mathrm{zpf},m,j} = \frac{1}{\phi_0}\sqrt{\frac{\hbar}{2\omega_m C_m}}, \tag{B4}$$

where $\varphi_{\mathrm{zpf},m}$ are the zero-point fluctuations in phase of mode m. The total phase difference across the Josephson junction $\hat{\varphi}_j$ is then the sum of these phase differences $\hat{\varphi}_j = \sum_m \hat{\varphi}_{m,j}$, and we can add the Junction nonlinearity to the Hamiltonian

$$\hat{H} = \sum_m \hbar\omega_m \hat{a}_m^\dagger \hat{a}_m + E_j\left[1 - \cos\hat{\varphi}_j - \frac{\hat{\varphi}_j^2}{2}\right]. \tag{B5}$$

Since the linear part of the Hamiltonian corresponds to the circuit with junctions replaced by inductors, the linear part already contains the quadratic contribution of the junction potential $\propto\hat{\varphi}_j^2$, and it is subtracted from the cosine junction potential.

B.3. Calculating Foster circuit parameters

Both ω_m and κ_m can be determined from $Y(\omega)$ since we have $Y(\omega_m + \mathrm{i}\kappa_m/2) = 0$ for low loss circuits. This can be proven by noticing that the admittance Y_m of mode m has two zeros at

$$\begin{aligned}\zeta_m &= \frac{1}{\sqrt{L_m C_m}}\sqrt{1 - \frac{1}{4Q^2}} + \mathrm{i}\frac{1}{2R_m C_m} \\ &\simeq \omega_m + \mathrm{i}\kappa_m/2\end{aligned} \tag{B6}$$

and ζ_m^*. The approximate equality holds in the limit of large quality factor $Q_m = R_m\big/\sqrt{L_m/C_m} \gg 1$. From equation (B1) we see that the zeros of Y are exactly the zeros of the admittances Y_k. The solutions of $Y(\omega) = 0$, which come in conjugate pairs ζ_m and ζ_m^*, thus provide us with both resonance frequencies $\omega_m = \mathrm{Re}[\zeta_m]$ and dissipation rates $\kappa_m = 2\mathrm{Im}[\zeta_m]$.

Additionally, we need to determine the effective capacitances C_m in order to obtain the zero-point fluctuations in phase of each mode. We focus on one mode k, and start by rewriting the admittance in equation (B1) as

$$Y_j(\omega) = Y_k(\omega)\frac{1}{1 + \sum\limits_{m\neq k} Y_k(\omega)/Y_m(\omega)}. \tag{B7}$$

Its derivative with respect to ω is

$$Y_j'(\omega) = Y_k'(\omega)\frac{1}{1 + \sum\limits_{m\neq k} Y_k(\omega)/Y_m(\omega)} + Y_k(\omega)\frac{\partial}{\partial\omega}\left[\frac{1}{1 + \sum\limits_{m\neq k} Y_k(\omega)/Y_m(\omega)}\right]. \tag{B8}$$

Evaluating the derivative at $\omega = \zeta_k$, where $Y_k(\zeta_k) = 0$ yields

$$\begin{aligned}Y_j'(\zeta_k) = Y_k'(\zeta_k) &= \mathrm{i}C_m\left(1 + \frac{4Q^2}{(\mathrm{i} + \sqrt{4Q^2 - 1})^2}\right) \\ &\simeq \mathrm{i}2C_m \text{ for } Q_m \gg 1.\end{aligned} \tag{B9}$$

The capacitance is thus approximatively given by

$$C_m = \mathrm{Im}\,[Y_j'(\zeta_k)]/2. \tag{B10}$$

B.4. Multiple junctions

When more than a single junction is present, we start by choosing a single reference junction, labeled r. All junctions will be again replaced by their inductances, and by using the admittance Y_r across the reference junction, we can determine the Hamiltonian including the nonlinearity of the reference junction through the procedure described above.

In this section, we will describe how to obtain the Hamiltonian including the nonlinearity of all other junctions too

$$\hat{H} = \sum_m \hbar\omega_m \hat{a}_m^\dagger \hat{a}_m + \sum_j E_j\left[1 - \cos\hat{\varphi}_j - \frac{\hat{\varphi}_j^2}{2}\right], \tag{B11}$$

where $\hat{\varphi}_j$ is the phase across the jth junction. This phase is determined by first calculating the zero-point fluctuations in phase $\varphi_{\mathrm{zpf},m,r}$ through the reference junction r for each mode m given by equation (B4). For each junction j, we then calculate the (complex) transfer function $T_{jr}(\omega)$ which converts phase in the reference junction to phase in junction j. We can then calculate the total phase across a junction j with respect to the reference phase of junction r, summing the contributions of all modes and both quadratures of the phase

$$\hat{\varphi}_j = \sum_m \varphi_{\mathrm{zpf},m,r}[\mathrm{Re}\,(T_{jr}(\omega_m))(\hat{a}_m + \hat{a}_m^\dagger) - \mathrm{i}\ \mathrm{Im}\,(T_{jr}(\omega_m))(\hat{a}_m - \hat{a}_m^\dagger)]. \tag{B12}$$

The definition of phase (Vool and Devoret 2017) $\varphi_j(t) = \phi_0^{-1}\int_{-\infty}^t v_j(\tau)\,\mathrm{d}\tau$ where v_j is the voltage across junction j translates in the frequency domain to $\varphi_j(\omega) = \mathrm{i}\omega\phi_0^{-1}V_j(\omega)$. Finding the transfer function T_{jr} for phase is thus equivalent to finding a transfer function for voltage $T_{jr}(\omega) = V_j(\omega)/V_r(\omega)$. This is a standard task in microwave network analysis (see section D.3 for more details).

B.5. Further treatment of the Hamiltonian

The cosine potential in equation (B11) can be expressed in the Fock basis by Taylor expanding it around small values of the phase. This yields

$$\hat{H} = \sum_m \hbar\omega_m \hat{a}_m^\dagger \hat{a}_m + \sum_j \sum_{n\geqslant 2} E_j \frac{(-1)^{n+1}}{(2n)!}\left[\sum_m \varphi_{\mathrm{zpf},m,j}(\hat{a}_m^\dagger + \hat{a}_m)\right]^{2n} \tag{B13}$$

which is the form returned by the QuCAT `hamiltonian` method. By keeping only the fourth power in the Taylor expansion and performing first order perturbation theory, we obtain

$$\hat{H} = \sum_m \sum_{n\neq m}\left(\hbar\omega_m - A_m - \frac{\chi_{mn}}{2}\right)\hat{a}_m^\dagger \hat{a}_m - \frac{A_m}{2}\hat{a}_m^\dagger \hat{a}_m^\dagger \hat{a}_m \hat{a}_m - \chi_{mn}\hat{a}_m^\dagger \hat{a}_m \hat{a}_n^\dagger \hat{a}_n, \tag{B14}$$

where the anharmonicity or self-Kerr of mode m is

$$A_m = \sum_j A_{m,j} \tag{B15}$$

as returned by the `anharmonicites` method, where

$$A_{m,j} = \frac{E_j}{2}\varphi_{\mathrm{zpf},m,j}^4 \tag{B16}$$

is the contribution of junction j to the total anharmonicity of a mode m. The cross-Kerr coupling between mode m and n is

$$\chi_{mn} = 2\sum_j \sqrt{A_{m,j}A_{n,j}}\,. \tag{B17}$$

Both self and cross-Kerr parameters are computed by the `kerr` method. Note in equation (B14) that the harmonic frequency of the Hamiltonian is shifted by A_m and $\sum_{n\neq m}\chi_{nm}/2$. The former comes from the change in Josephson inductance induced by phase fluctuations of mode m. The latter is called the Lamb shift (Gely *et al* 2018) and is induced by phase fluctuations of the other modes of the circuit.

Appendix C. Algorithmic methods

There are three calculations to accomplish in order to obtain all the parameters necessary to write the circuit Hamiltonian. We need:

- the eigen-frequencies ω_m and loss rates κ_m fulfilling $Y_r(\zeta_m = \omega_m + \mathrm{i}\kappa_m/2) = 0$ where Y_r is the admittance across a reference junction
- the derivative of this admittance evaluated at ζ_m
- the transfer functions T_{jr} between junctions j and the reference junction r.

In this section, we cover the algorithmic methods used to calculate these three quantities.

C.1. Resonance frequency and dissipation rate

C.1.1. Theoretical background. In order to obtain an expression for the admittance across the reference junction, we start by writing the set of equations governing the physics of the circuit. We first determine a list of nodes, which are points at which circuit components connect. Each node, labeled n, is assigned a voltage v_n. We name $r_\pm$ the positive and negative nodes of the reference junction.

We are interested in the steady-state oscillatory behavior of the system. We can thus move to the frequency domain, with complex node voltages $|V_n(\omega)|\mathrm{e}^{\mathrm{i}(\omega t+\theta(V_n(\omega_n)))}$, fully described by their phasors, the complex numbers $V_n = |V_n(\omega)|\mathrm{e}^{\mathrm{i}\theta(V_n(\omega_n))}$. In this mathematical construct, the real-part of the complex voltages describes the voltage one would measure at the node in reality. Current conservation dictates that the sum of all currents arriving at any node n, from the other nodes k of the circuit should be equal to the oscillatory current injected at node n by a hypothetical, infinite impedance current source. This current is also characterized by a phasor I_n. This can be compactly written as

$$\sum_{k\neq n} Y_{nk}(V_n - V_k) = I_n, \tag{C1}$$

where k label the other nodes of the circuit and Y_{nk} is the admittance directly connecting nodes k and n. Note that in this notation, if a node k_1 can only reach node n through another node k_2, then $Y_{nk_1} = 0$. Inductors (with inductance L), capacitors (with capacitance C) and resistors (with resistance R) then have admittances $1/\mathrm{i}L\omega$, $\mathrm{i}C\omega$ and $1/R$ respectively.

Expanding equation (C1) yields

$$(\sum_{k\neq n} Y_{nk})\, V_n - \sum_{k\neq n} Y_{nk} V_k = I_n \tag{C2}$$

which can be written in matrix form as

$$\begin{pmatrix} \Sigma_{k\neq 0} Y_{0k} & -Y_{01} & \cdots & -Y_{0N} \\ -Y_{10} & \Sigma_{k\neq 1} Y_{1k} & \cdots & -Y_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ -Y_{N0} & -Y_{N1} & \cdots & \Sigma_{k\neq N} Y_{Nk} \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \\ \vdots \\ V_N \end{pmatrix} = \begin{pmatrix} I_0 \\ I_1 \\ \vdots \\ I_N \end{pmatrix}. \tag{C3}$$

Since voltage is the electric potential of a node relative to another, we still have the freedom of choosing a ground node. Equivalently, conservation of currents imposes that current exciting that node is equal to the sum of currents entering the others, there is thus a redundant degree of freedom in equation (C3). For simplicity, we will choose node 0 as ground. Since we are only interested in the admittance across the reference junction, we set all currents to zero, except the currents entering the positive and negative reference junction nodes: I_{r_+} and $I_{r_-} = -I_{r_+}$ respectively. The admittance is defined by $Y_r = I_{r_+}/(V_{r_+} - V_{r_-})$. The equations then reduce to

$$\mathbf{Y}\begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{pmatrix} = Y_r \begin{pmatrix} \vdots \\ 0 \\ V_{r_+} - V_{r_-} \\ 0 \\ \vdots \\ 0 \\ V_{r_-} - V_{r_+} \\ 0 \\ \vdots \end{pmatrix}, \tag{C4}$$

where $\mathbf{Y}$ is the admittance matrix

$$\mathbf{Y} = \begin{pmatrix} \Sigma_{k\neq 1} Y_{1k} & -Y_{12} & \cdots & -Y_{1N} \\ -Y_{21} & \Sigma_{k\neq 1} Y_{2k} & \cdots & -Y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -Y_{N1} & -Y_{N2} & \cdots & \Sigma_{k\neq N} Y_{Nk} \end{pmatrix}. \tag{C5}$$

For $Y_r = 0$, equation (C4) has a solution for only specific values of $\omega = \zeta_m$. These are the values which make the admittance matrix singular, i.e. which make its determinant zero

$$\mathrm{Det}\,[\mathbf{Y}(\zeta_m)] = 0. \tag{C6}$$

The determinant is a polynomial in ω, so the problem of finding $\zeta_m = \omega_m + \mathrm{i}\kappa_m/2$ reduces to finding the roots of this polynomial. Note that plugging ζ_m into the frequency domain expression for the node voltages yields $V_k(\zeta_m)\mathrm{e}^{\mathrm{i}\omega_m t}\mathrm{e}^{-\kappa_m t/2}$, such that the energy $\propto v_k(t)^* v_k(t) \propto \mathrm{e}^{-\kappa_m t}$ decays at a rate κ_m, which explains the division by two in the expression of ζ_m. Also note, that we would have obtained equation equation (C6) regardless of the choice of reference element.

C.1.2. Algorithm. We now describe the algorithm used to determine the solutions $\zeta_m = \omega_m + \mathrm{i}\kappa_m/2$ of equation (C6). As an example, we consider the circuit of figure C1(a) that a user would have built with the GUI.

The algorithm is as follows

1. Eliminate wires and grounds. In this case, nodes N_0, N_5 would be grouped under a single node labeled 0 and nodes N_1, N_2, N_3 would be grouped under node 1, we label node N_4 node 2, as shown in figure C1(b).
2. Compute the un-grounded admittance matrix. For each component present between the different couples of nodes, we append the admittance matrix with the components admittance. The matrix is then multiplied by ω such that all components are polynomials in ω, ensuring that the determinant is also a polynomial. In this example, the matrix is

$$\begin{pmatrix} \mathrm{i}C\omega^2 + 1/\mathrm{i}L & -\mathrm{i}C\omega^2 & -1/\mathrm{i}L \\ -\mathrm{i}C\omega^2 & \mathrm{i}C\omega^2 + \omega/R & -\omega/R \\ -1/\mathrm{i}L & -\omega/R & 1/\mathrm{i}L + \omega/R \end{pmatrix}. \tag{C7}$$

3. Choose a ground node. The node which has a corresponding column with the most components is chosen as the ground node (to reduce computation time). These rows and columns are erased from the matrix, yielding the final form of the admittance matrix

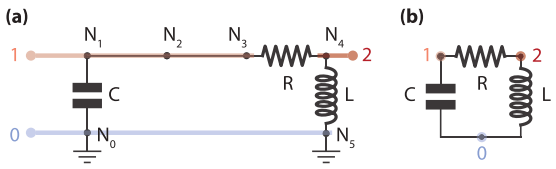

Figure C1. Example circuit to illustrate the mode frequency finder algorithm. (a) Example of a circuit built through the GUI by a user. (b) Application of the first step of the algorithm which removes the wires and grounds to obtain a minimal number of nodes without removing any components.

$$\mathbf{Y} = \begin{pmatrix} \mathrm{i}C\omega^2 + \omega/R & -\omega/R \\ -\omega/R & 1/\mathrm{i}L + \omega/R \end{pmatrix}. \tag{C8}$$

4. Compute the determinant. Even if the capacitance, inductance and resistance were specified numerically, the admittance matrix would still be a function of the symbolic variable ω. We thus rely on a symbolic Berkowitz determinant calculation algorithm (Berkowitz 1984, Kerber 2009) implemented in the Sympy library through the `berkowitz_det` function. In this example, one would obtain

$$\mathrm{Det}[\mathbf{Y}] = LC\omega^2 - \mathrm{i}RC\omega - 1. \tag{C9}$$

5. Find the roots of the polynomial. Whilst the above steps have to be performed only once for a given circuit, this one should be performed each time the user edits the value of a component. The root-finding is divided in the following steps as prescribed by (Press *et al* 2007).

   Diagonalize the polynomials companion matrix (Horn and Johnson 1985) to obtain an exhaustive list of all roots of the polynomial. This is implemented in the NumPy library through the `roots` function.

   Refine the precision of the roots using multiple iterations of Halley's gradient based root finder (Press *et al* 2007) until iterations do not improve the root value beyond a predefined tolerance given by the `Qcircuit` argument `root_relative_tolerance`. The maximum number of iterations that may be carried out is determined by the `Qcircuit` argument `root_max_iterations`. If the imaginary part relative to the real part of the root is lower than the relative tolerance, the imaginary part will be set to zero. The relative tolerance thus sets the highest quality factor that QuCAT can detect.

   Remove identical roots (equal up to the relative tolerance), roots with negative imaginary or real parts, 0-frequency roots, roots for which $Y_l{}'(\omega_m) < 0$ for all l, where Y_l is the admittance evaluated at the nodes of an inductive element l, and roots for which $Q_m <$ `Qcircuit.Q_min`. The user is warned of a root being discarded when one of these cases is unexpected.

The roots ζ_m obtained through this algorithm are accessed through the method `eigenfrequencies` which returns the oscillatory frequency in Hertz of all the modes $\mathrm{Re}[\zeta_m]/2\pi$ or `loss_rates` which returns $2\mathrm{Im}[\zeta_m]/2\pi$.

C.2. Derivative of the admittance

The zero-point fluctuations in phase $\varphi_{\mathrm{zpf},m,r}$ for each mode m across a reference junction r is the starting point to computing a Hamiltonian for the nonlinear potential of the Junctions. As expressed in equation (B4), this quantity depends on the derivative $Y_r{}'$ of the admittance Y_r calculated at the nodes of the reference element. In this section we first cover the algorithm used to obtain the admittance at the nodes of an arbitrary component. From this admittance we then describe the method to obtain the derivative of the admittance on which $\varphi_{\mathrm{zpf},m,r}$ depends Finally we describe how to choose a (mode-dependent) reference element.

C.2.1. Computing the admittance. Here we describe a method to compute the admittance of a network between two arbitrary nodes. We will continue using the example circuit of figure C1, assuming we want to compute the admittance at the nodes of the inductor.

1. Eliminate wires and grounds as in the resonance finding algorithm, nodes N_0, N_5 would be grouped under a single node labeled 0 and nodes N_1, N_2, N_3 would be grouped under node 1, we label node N_4 node 2. We thus obtain figure C3(a).

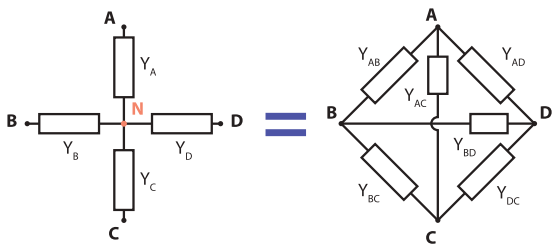

Figure C2. Star-mesh transform. A node N connected to nodes A, B, C,.. through admittances Y_A, Y_B, … can be eliminated if we interconnect nodes A, B, C,.. with impedances Y_{AB}, Y_{AC}, Y_{BC}, … given by $Y_{XY} = Y_X Y_Y / \sum_M Y_M$. We show the 5 node case, the initial network on the left is called the 'star', which is then transformed to the 'mesh' on the right, reducing the total number of nodes by 1.

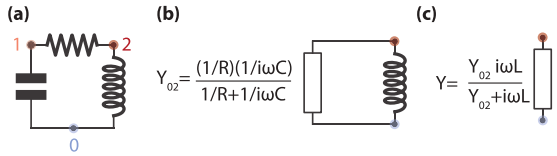

Figure C3. Example to illustrate the admittance calculation algorithm. (a) Example of a circuit built through the GUI by a user, after removal of all wires and grounds. (b) Application of the star-mesh transformation to remove node 1. (c) After each application of the star-mesh transformation, parallel connections are grouped together. Only the two nodes across which we want to compute the admittance remain, the admittance is that of the remaining 'admittance component'.

2. Group parallel connections. Group all components connected in parallel as a single 'admittance component' equal to the sum of admittances of its parts. In this way two nodes are either disconnected, connected by a single inductor, capacitor, junction or resistor, or connected by a single 'admittance component'.

3. Reduce the network through star-mesh transformations. Excluding the nodes across which we want to evaluate the admittance, we utilize the star-mesh transformation described in figure C2 to reduce the number of nodes in the network to two. If following a star-mesh transformation, two components are found in parallel, they are grouped under a single 'admittance component' as described previously. For a node connected to more than 3 other nodes the star-mesh transformation will increase the total number of components in the circuit. So we start with the least-connected nodes to maintain the total number of components in the network to a minimum. In this example, we want to keep nodes 0 and 2, but remove node 1, a start-mesh transform leads to the circuit of figure C3(b) then grouping parallel componnets leads to (c).

4. The admittance is that of the remaining 'admittance component' once the network has been completely reduced to two nodes.

The symbolic variables at this stage (Sympy `Symbols`) are ω, and the variables corresponding to any component with un-specified values.

C.2.2. Differentiating the admittance. The expression for the admittance obtained from the above algorithm will necessarily be in the form of multiple multiplication, divisions or additions of the admittance of capacitors, inductors or resistors. It is thus possible to transform Y to a rational function of ω

$$Y(\omega) = \frac{P(\omega)}{Q(\omega)} = \frac{p_0 + p_1\omega + p_2\omega^2 + \ldots}{q_0 + q_1\omega + q_2\omega^2 + \ldots} \tag{C10}$$

with the sympy function `together`. It is then easy to symbolically determine the derivative of Y, ready to be evaluated at ζ_m once the coefficients p_i and q_i have been extracted

$$\begin{aligned} Y'(\zeta_m) &= (P'(\zeta_m)Q(\zeta_m) - P(\zeta_m)Q'(\zeta_m)/Q(\zeta_m)^2 \\ &= (p_1 + 2p_2\zeta_m + \ldots)/(q_0 + q_1\zeta_m + \ldots), \end{aligned} \tag{C11}$$

taking advantage of the property $P(\zeta_m) \propto Y(\zeta_m) = 0$.

C.2.3. Choice of reference element. For each mode m, we use as reference element r the inductor or junction which maximizes $\varphi_{\mathrm{zpf},m,r}$ as specified by equation (B4). This corresponds to the element where the phase

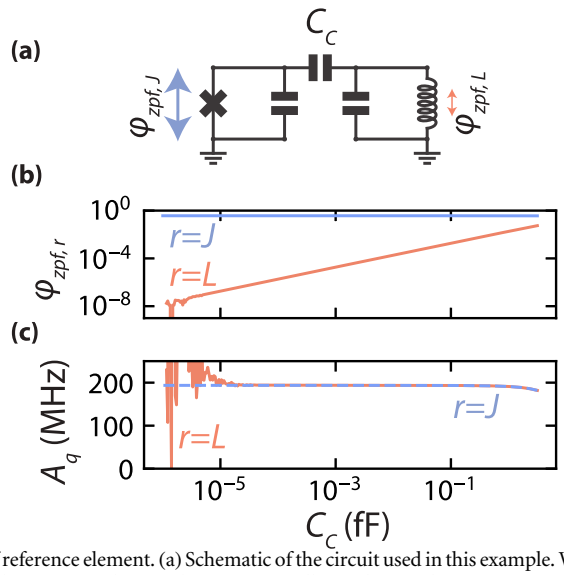

Figure C4. Impact of the choice of reference element. (a) Schematic of the circuit used in this example. We have used a 100 fF capacitances, a 10 nH inductor and a 8 nH Josephson inductance, we will vary the coupling capacitance. The zero-point fluctuations in phase $\varphi_{\mathrm{zpf},r}$ across the inductor ($r = L$) and junction ($r = J$) for most anharmonic mode are drawn on the schematic and plotted in (b) for different values of the coupling capacitor C_C. The phase oscillations associated with this mode are mostly located in the junction, so as the coupling capacitor is lowered, the amplitude of phase oscillations diminishes in the inductor. Below $C_C \sim 10^{-5}$ fF, numerical accuracies lead to unreliable values of the phase fluctuations in the inductor. This results in the anharmonicity of the qubit-like mode A_q, plotted in (c), to be incorrectly estimated if the inductor is chosen as a reference element and the anharmonicity is computed using equation (B12).

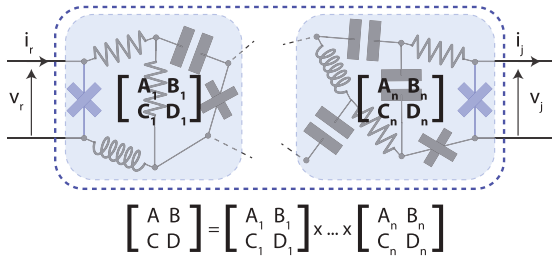

Figure C5. Visual summary of the notations and properties of the *ABCD* matrix applied to the calculation of T_{rj}. The transfer function $T_{rj} = 1/A$ is the inverse of the first coefficient of the *ABCD* matrix which relates the voltages and currents on either end of a network. These currents and voltages are defined as shown above, with the reference junction on the left and junction *j* on the right. Currents are defined as entering and exciting on the left and right respectively. If the circuit is constituted of a cascade of two port sub-networks, the product of the sub-network *ABCD* matrices are equal to the *ABCD* matrix of the total network.

fluctuations are majoritarily located. We find that doing so considerably increases the success of evaluating $Y'(\omega_m)$. As an example, we plot in figure C4 the zero-point fluctuations in phase $\varphi_{\mathrm{zpf},m,r}$ of the transmon-like mode, calculated for the circuit of figure 1, with the junction or inductor as reference element. What we find is that if the coupling capacitor becomes too small, resulting in modes which are nearly totally localized in either inductor or junction, choosing the wrong reference element combined with numerical inaccuracies leads to unreliable values of $\varphi_{\mathrm{zpf},m,r}$.

C.3. Transfer functions

In this section, we describe the method used to determine the transfer function T_{jr} between a junction *j* and the reference junction *r*. This quantity can be computed from the *ABCD* matrix (Pozar 2009). The *ABCD* matrix relates the voltages and currents in a two port network

$$\begin{pmatrix} V_r \\ I_r \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} V_j \\ I_j \end{pmatrix}, \tag{C12}$$

where the convention for current direction is described in figure C5. By constructing the network as in figure C5, with the reference junction on the left and junction *j* on the right, the transfer function is given by

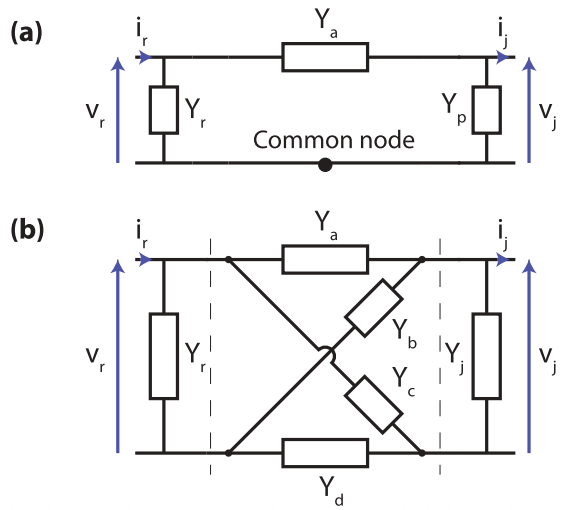

Figure C6. Networks after star-mesh reduction. The two non-trivial situations reached after applying star-mesh transformations to a network to obtain T_{rj}.

$$T_{jr}(\omega) = \frac{V_j(\omega)}{V_r(\omega)} = \frac{1}{A}. \tag{C13}$$

To determine A, we first reduce the circuit using star-mesh transformations (see figure C2), and group parallel connections as described in the previous section, until only the nodes of junctions r and j are left. If the junctions initially shared a node, the resulting circuit will be equivalent to the network shown in figure C6 (a). In this case,

$$A = \left(1 + \frac{Y_p}{Y_a}\right). \tag{C14}$$

If the junctions do not share nodes, the resulting circuit will be equivalent to the network shown in figure C6(b), where some admittances may be equal to 0 to represent open circuits. To compute the *ABCD* matrix of this resulting circuit, we make use of the property illustrated in figure C5: the *ABCD* matrix of a cascade connection of two-port networks is equal to the product of the *ABCD* matrices of the individual networks. We first determine the *ABCD* matrix of three parts of the network (separated by dashed line in figure C6) such that the *ABCD* matrix of the total network reads

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Y_r & 1 \end{bmatrix} \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ Y_j & 1 \end{bmatrix}. \tag{C15}$$

$$A = \tilde{A} + \tilde{B} Y_j, \tag{C16}$$

where the A and B coefficients of the middle part of the network are

$$\begin{aligned} \tilde{A} &= (Y_a + Y_b)(Y_c + Y_d)/(Y_a Y_d - Y_b Y_c) \\ \tilde{B} &= (Y_a + Y_b + Y_c + Y_d)/(Y_a Y_d - Y_b Y_c). \end{aligned} \tag{C17}$$

The *ABCD* matrix for the middle part of the circuit is derived in section 10.11 of (Arshad 2010), and the *ABCD* matrices for the circuits on either sides are provided in (Pozar 2009).

This method is also applied to calculate the transfer function to capacitors, inductors and resistors, notably to visualize the normal mode with the `show_normal_mode` function.

C.4. Alternative algorithmic methods

Since symbolic calculations are the most computationally expensive steps in a typical use of QuCAT, we cover in this section some alternatives to the methods previously described, and the reasons why they were not chosen.

C.4.1. Eigen-frequencies from the zeros of admittance. One could solve $Y_r(\omega) = 0$ where Y_r is the admittance computed as explained in section D.2. Providing good initial guesses for all values of the zeros ζ_m can be provided, a number of root-finding algorithms can then be used to obtain final values of ζ_m.

A set of initial guesses could be obtained by noticing that Y_r is a rational function of ω. Roots of its numerator are potentially zeros of Y, and a complete set of them is easy to obtain through a diagonalization

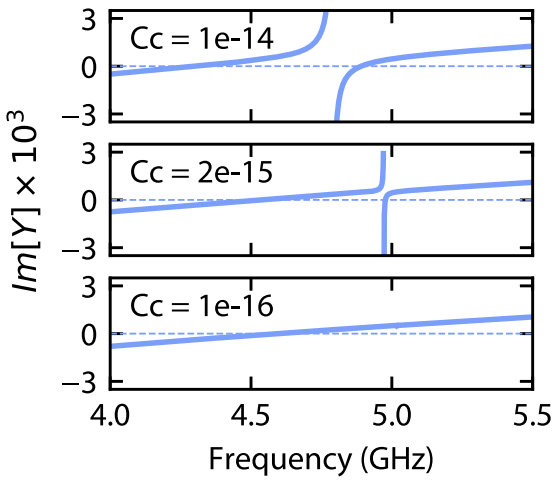

Figure C7. Determining zero-point fluctuations from differentiating the admittance in decoupled circuits. Imaginary part of the admittance Im[Y] across the junction of the circuit of figure 1 for different values of the coupling capacitor C_c and for $L_j = 12$ nH. Resonances correspond to the frequencies ω at which the admittance crosses 0, and the calculation of zero-point fluctuations depends on the derivative Im[Y'] at that point. As the resonator and transmon parts of the circuit decouple, Im[Y'] becomes larger, requiring a lower $\delta\omega$ if the admittance is to be determined through equation (C18). In extreme cases (see lower panel), when the derivative is very large, the smaller variation in the background slope may be mistaken for the slope at a resonance.

of the companion matrix as discussed before. Note that if these roots are roots of the denominator with equal or higher multiplicity, then they are not zeros of Y. They can, however, make good initial guesses of a root-finding algorithm run on Y_r. This requires a simplification of Y_r, as computed through star-mesh transforms, to its rational function form. We find this last step to be as computationally expensive as obtaining a determinant.

A different approach, which does not require using a root-finding algorithm on Y_r, is to simplify the rational-function form of Y such that the numerator and denominator share no roots. This can be done by using the extended Euclidian algorithm to find the greatest common polynomial divisor of the numerator and denominator. However, the numerical inaccuracies in the numerator and denominator coefficients may make this method unreliable.

The success of both of these approaches is dependent on determining a good reference component r, which may be mode-dependent (see figure C4). This reference component is difficult to pick at this stage, when the mode frequencies are unknown.

C.4.2. Finite difference estimation of the admittance derivative Rather than symbolically differentiating the admittance, one could use a numerical finite difference approximation, for example

$$Y_r{}'(\omega) \simeq \frac{Y_r(\omega + \delta\omega/2) - Y_r(\omega - \delta\omega/2)}{\delta\omega}. \tag{C18}$$

Y_r can be obtained through star-mesh reductions, or from a resolution of equation (C4).

But finding a good value of $\delta\omega$ is no easy task. As an example, we consider the circuit of figure 1, where we have taken as reference element the junction. As shown in figure C7, when the resonator and transmon decouple through a reduction of the coupling capacitor, a smaller and smaller $\delta\omega$ is required to obtain $Y_r{}'$ evaluated at the ζ_1 of the resonator-like mode. We have tried making use of Ridders method of polynomial extrapolation to try and reliably approach the limit $\delta x \to 0$ (Press *et al* 2007). However at small coupling capacitance, it always converges to the slower varying background slope of Y_r, without any way of detecting the error.

C.4.3. Transfer functions from the admittance matrix. Calculating the transfer function T_{ij} could alternatively be carried out through the resolution of the system of equations (C5). The difference in voltage of a reference elements nodes would first have to be fixed to the zero-point fluctuations computed with the method of section (D.2). These equations would have to be resolved at each change of system parameters and for each mode, with ω replaced in the admittance matrix by its corresponding value for a given mode. This is to be

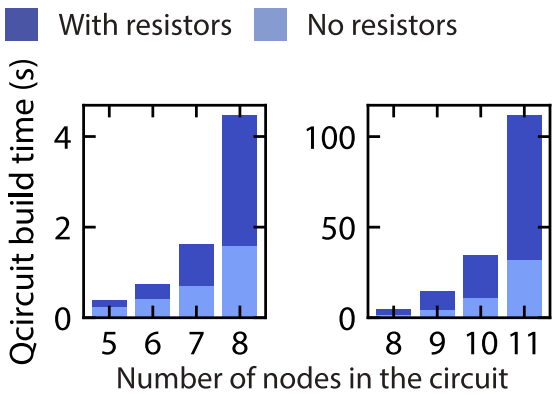

Figure D1. Computation time with increasing circuit size. On the vertical axis, we show the time necessary to initialize the `Qcircuit` object, which is the computationally expensive part of a typical QuCAT user case. This is plotted as a function of the number of nodes in the circuit. The test circuit used here is the multi-mode circuit of figure A3(c), optionally with a resistor in parallel of each capacitor. The number of nodes are increased by adding modes to the circuit. Most of the computational time is spent in the symbolic manipulations performed with the sympy library.

balanced against a single symbolic derivation of T_{ij} through star-mesh transformations, and fast evaluations of the symbolic expression for different parameters.

Appendix D. Performance and limitations

D.1. Number of nodes

In this section we ask the question: how big a circuit can QuCAT analyze? To address this, we first consider the circuit of figure A3(c), and secondly the same circuit with resistors added in parallel to each capacitor. As the number of $(R)LC$ oscillators representing the modes of a CPW resonator is increased, we measure the time necessary for the initialization of the Qcircuit object. This is typically the most computationally expensive part of a QuCAT usage, limited by the speed of symbolic manipulations in Sympy.

These symbolic manipulations include:

- calculating the determinant of the admittance matrix
- converting that determinant to a polynomial
- reducing networks through star-mesh transformations both for admittance and transfer function calculations
- rational function manipulations to prepare the admittance for differentiation.

Once these operations have been performed, the most computationally expensive step in a Qcircuit method is finding the root of a polynomial (the determinant of the admittance matrix) which typically takes a few milliseconds.

The results of this test are reported in figure D1. We find that relatively long computation times above 10 s are required as one goes beyond 10 circuit nodes. Due to an increased complexity of symbolic expressions, the computation time increases when resistors are included. For example, the admittance matrix of a non-resistive circuit will have no coefficients proportional to ω, only ω^2 and only real parts, translating to a polynomial in $\Omega = \omega^2$ which will have half the number of terms as a resistive circuit. However, we find that this initialization time is also greatly dependent on the circuit connectivity, and this test should be taken as only a rough guideline.

Making QuCAT compatible with the analysis of larger circuits will inevitably require the development of more efficient open-source symbolic manipulation tools. The development of the open-source C++ library SymEngine https://github.com/symengine/symengine, together with its Python wrappers, the symengine.py project https://github.com/symengine/symengine.py, could lead to rapid progress in this direction. An enticing prospect would then be able to analyze the large scale cQED systems underlying modern transmon-qubit-based quantum processors (Kelly *et al* 2019). One should keep in mind that an increase in circuit size translates to an increase in the number of degrees of freedom of the circuit and hence of the Hilbert space size needed for further analysis once a Hamiltonian has been extracted from QuCAT.

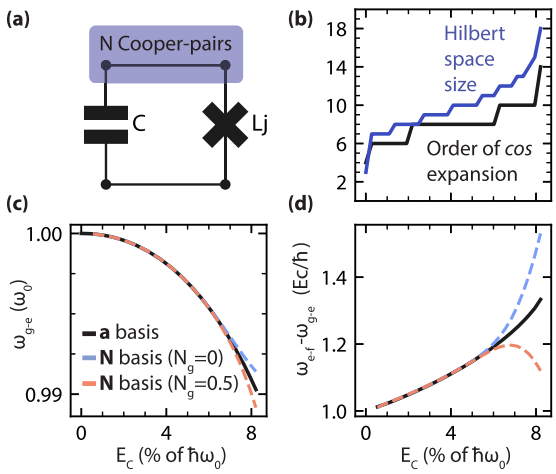

Figure D2. Applicability of the harmonic Fock basis. (a) Transmon or Cooper-pair-box circuit. (b) On the x-axis, we vary the approximate anharmonicity $E_C = e^2/2C$ with respect to the frequency $\omega_0 = 1/\sqrt{L_j C}$. For each value, we plot a Hilbert space size, and order of Taylor-expansion of the junction cosine-potential. Incrementing these values produces less than a 0.1 percent change in the first two transition frequencies obtained by diagonalizing the Hamiltonian. Beyond a relative anharmonicity of 8, convergence is no longer reached, even for Hilbert space sizes and Taylor expansions up to 100. (c) Frequency of the first transition ω_{g-e} obtained from Hamiltonian diagonalization, relative to the value expected from first order perturbation theory: $\omega_0 - E_C/\hbar$. (d) Anharmonicity $\omega_{e-f} - \omega_{g-e}$ obtained from Hamiltonian diagonalization relative to the value expected from first order perturbation theory $E_C/\hbar$. Black lines correspond to a diagonalization in the harmonic Fock basis (equation (B13)), blue and orange dashed lines correspond to a diagonalization of the Cooper-pair box Hamiltonian with gate charges of 0 and 1/2 respectively. The harmonic Fock basis provides reliable results up to approximatively 6 percent anharmonicity.

D.2. Degree of anharmonicity

In this section we study the limits of the current quantization method used in QuCAT. More specifically, we study the applicability of the basis used to express the Hamiltonian, that of Fock-states of harmonic normal modes of the linearized circuit. To do so, we use the simplest circuit possible (figure D2(a)), the parallel connection of a Josephson junction and a capacitor. As the anharmonicity of this circuit becomes a greater fraction of its linearized circuit resonance, the physics of the circuit goes from that of a Transmon to that of a Cooper-pair box (Koch *et al* 2007), and the Fock-state basis becomes inadequate. This test should be viewed as a guideline for the maximum acceptable amount for anharmonicity. We find that when the anharmonicity exceeds 6 percent of the eigenfrequency, a QuCAT generated Hamiltonian will not reliably describe the system.

In this test, we vary the ratio of Josephson inductance L_j to capacitance C, increasing the anharmonicity expected from first-order perturbation theory (see equation (B14)), called charging energy $E_C = e^2/2C$. The resonance frequency of the linearized circuit $\omega_0 = 1/\sqrt{L_j C}$ is maintained constant. For each different charging energy, we use the `hamiltonian` method to generate a Hamiltonian of the system. We are interested in the order of the Taylor expansion of the cosine potential, and the size of the Hilbert space, necessary to obtain realistic first and second transition frequencies of the circuit, named ω_{g-e} and ω_{e-f} respectively. To do so, we increase the order of Taylor expansion, and for each order we sweep through increasing Hilbert space sizes. In figure D2(b), we show the values of these parameters at which incrementing them would not change ω_{g-e} and ω_{e-f} by more than 0.1 percent. Beyond a relative anharmonicity $E_C/\hbar\omega_0$ of 8 percent, such convergence is no longer reached, even for cosine expansion orders and Hilbert space sizes up to 100.

Up to the point of no convergence, we compare the results obtained from the diagonalization in the harmonic Fock basis (Hamiltonian generated by QuCAT), with a diagonalization of the Cooper-pair box Hamiltonian. In regimes of higher anharmonicity, the system becomes sensitive to the preferred charge offset between the two plates of the capacitor N_g (expressed in units of Cooper-pair charge $2e$) imposed by the electric environment of the system. The Cooper-pair box Hamiltonian takes this into account

$$\hat{H}_{\mathrm{CPB}} = 4E_C\left(\sum_N |N\rangle\langle N| - N_g\right)^2 - \sum_N E_j(|N+1\rangle\langle N| + |N\rangle\langle N+1|), \tag{D1}$$

where $|N\rangle$ is the quantum state of the system where N Cooper-pairs have tunneled across the junction to the node indicated in figure D2(a). For more details on Cooper-pair box physics and the derivation of this Hamiltonian, refer to (Schuster 2007). We diagonalize this Hamiltonian in a basis of 41 $|N\rangle$ states.

We find that beyond 6 percent anharmonicity, the Cooper-pair box Hamiltonian becomes appreciably sensitive to N_g and diverges from the results obtained in the Fock basis. This corresponds to $E_j/E_C \simeq 35$ at which

the charge dispersion (the difference in frequency between 0 and 0.5 charge offset) is 4×10^{-5} and 1×10^{-3} for the first two transitions respectively.

Beyond 8 percent anharmonicity, one cannot reach convergence with the Fock basis and just before results diverge considerably from that of the Cooper-pair box Hamiltonian. This corresponds to $E_j/E_C \simeq 20$ at which the charge dispersion is 1.5×10^{-3} and 3×10^{-2} for the first two transitions respectively. A possible extension of the QuCAT Hamiltonian could thus include handling static offsets in charge and different quantization methods, for example quantization in the charge basis to extend QuCAT beyond the scope of weakly-anharmonic circuits.

Appendix E. Installing QuCAT and dependencies

The recommended way of installing QuCAT is through the standard Python package installer by running `pip install qucat` in a terminal. Alternatively, all versions of QuCAT, including the version currently under-development is available on github at https://github.com/qucat. After downloading or cloning the repository, one can navigate to the `src` folder and run `pip install .` in a terminal.

QuCAT and its GUI is cross-platform, and should function on Linux, MAC OS and Windows. QuCAT requires a version of Python 3, using the latest version is advised. QuCAT relies on several open-source Python libraries: Numpy, Scipy, Matplotlib, Sympy and QuTiP (Johansson *et al* 2012, 2013), installation of Python and these libraries through Anaconda is recommended. The performance of Sympy calculations can be improved by installing Gmpy2.

Appendix F. List of QuCAT objects and methods

QuCAT objects

`Network`—Creates a `Qcircuit` from a list of components

`GUI` —Opens a graphical user interface for the construction of a `Qcircuit`

`J` —Creates a Josephson junction object

`L`—Creates a inductor object

`C`—Creates a capacitor object

`R`—Creates a resistor object

Qcircuit methods

`eigenfrequencies` —Returns the normal mode frequencies

`loss_rates` —Returns the normal mode loss rates

`anharmonicities` —Returns the anharmonicities or self-Kerr of each normal mode

`kerr`—Returns the self-Kerr and cross-Kerr for and between each normal mode

`f_k_A_chi`—Returns the eigenfrequency, loss-rates, anharmonicity, and Kerr parameters of the circuit

`hamiltonian`—Returns the Hamiltonian of [B13]

Qcircuit methods *(only if built with GUI)*

`show` —Plots the circuit

`show_normal_mode` —Plots the circuit overlaid with the currents, voltages, charge or fluxes through each component when a normal mode is populated with a single-photon coherent state.

J, L, R, C methods

`zpf` —Returns contribution of a mode to the zero-point fluctuations in current, voltages, charge or fluxes.

J methods

`anharmonicity`—Returns the contribution of this junction to the anharmonicity of a given normal mode (equation (B16)).

Appendix G. Source code and documentation

The code used to generate the figures of this paper are available in Zenodo with the identifier (https://doi.org/10.5281/zenodo.3298107). Tutorials and examples, including those presented here are available on the QuCAT website at https://qucat.org/. The latest version of the QuCAT source code, is available to download or to contribute to at https://github.com/qucat.

ORCID iDs

Mario F Gely https://orcid.org/0000-0002-2861-2709

References

Arshad M 2010 *Network Analysis and Circuits* (New Delhi: Laxmi Publications)
Arute F *et al* 2019 *Nature* **574** 505–10
Berkowitz S J 1984 *Inf. Process. Lett.* **18** 147
Bosman S J, Gely M F, Singh V, Bruno A, Bothner D and Steele G A 2017 *npj Quantum Inf.* **3** 46
Devoret M H and Schoelkopf R J 2013 *Science* **339** 1169
Foster R M 1924 *Bell Syst. Tech. J.* **3** 259
Gely M F, Parra-Rodriguez A, Bothner D, Blanter Y M, Bosman S J, Solano E and Steele G A 2017 *Phys. Rev.* B **95** 245115
Gely M F, Steele G A and Bothner D 2018 *Phys. Rev.* A **98** 053808
Gu X, Kockum A F, Miranowicz A, Liu Y-X and Nori F 2017 *Phys. Rep.* **718** 1
Horn R A and Johnson C 1985 *Matrix Analysis* (Cambridge: Cambridge University Press)
Johansson J, Nation P and Nori F 2012 *Comput. Phys. Commun.* **183** 1760
Johansson J R, Nation P and Nori F 2013 *Comput. Phys. Commun.* **184** 1234
Kerber M 2009 *Int. J. Comput. Math.* **86** 2186
Koch J, Yu T M, Gambetta J, Houck A A, Schuster D I, Majer J, Blais A, Devoret M H, Girvin S M and Schoelkopf R J 2007 *Phys. Rev.* A **76** 042319
Kounalakis M, Dickel C, Bruno A, Langford N and Steele G 2018 *npj Quantum Inf.* **4** 38
Nigg S E, Paik H, Vlastakis B, Kirchmair G, Shankar S, Frunzio L, Devoret M H, Schoelkopf R J and Girvin S M 2012 *Phys. Rev. Lett.* **108** 240502
Ockeloen-Korppi C, Damskägg E, Pirkkalainen J-M, Heikkilä T, Massel F and Sillanpää M 2016 *Phys. Rev.* X **6** 041024
Pozar D M 2009 *Microwave Engineering* (New York: Wiley)
Press W H, Teukolsky S A, Vetterling W T and Flannery B P 2007 *Numerical Recipes: The Art of Scientific Computing* 3rd edn (Cambridge: Cambridge University Press)
Roy T, Kundu S, Chand M, Hazra S, Nehra N, Cosmic R, Ranadive A, Patankar M P, Damle K and Vijay R 2017 *Phys. Rev. Appl.* **7** 054025
Scheer M G and Block M B 2018 arXiv:1810.11510 [quant-ph]
Schuster D I 2007 *Circuit Quantum Electrodynamics* (New Haven, CT: Yale University Press)
Solgun F, Abraham D W and DiVincenzo D P 2014 *Phys. Rev.* B **90** 134504
Solgun F and DiVincenzo D P 2015 *Ann. Phys.* **361** 605
Teufel J D, Donner T, Li D, Harlow J W, Allman M S, Cicak K, Sirois A J, Whittaker J D, Lehnert K W and Simmonds R W 2011 *Nature* **475** 359
Viennot J J, Ma X and Lehnert K W 2018 *Phys. Rev. Lett.* **121** 183601
Vool U and Devoret M 2017 *Int. J. Circuit Theory Appl.* **45** 897
Xiang Z-L, Ashhab S, You J and Nori F 2013 *Rev. Mod. Phys.* **85** 623