Appendix A

Information on Matlab data processing

The Matlab scripts used in this work were developed by Johannes Thewes and me. They can be found in github:

`https://github.com/CarolinLueders/QST`
(Repository: QST, User: CarolinLueders)

A.1 Computation of quadratures from raw data

A homodyne measurement consists of a measurement without signal, in order to calibrate the number of photons of the LO, and a measurement with signal. These usually have file names of the form "01-xxmW-LOonly.raw" and "02-xxmW-LOwithDL.raw", respectively, and are stored in a folder called "raw-data". In the following, the file names will be referred to with the placeholders filenameLO and filenameSIG.
First, the quadratures are computed from the raw data. Therefore, in Matlab, one needs to be in the folder just above the folder "raw-data".
For a series of datasets, it is most convenient to run **seriesQuadratures**, which computes the quadratures for each dataset and saves them in a folder called "mat-data". This function uses the function **prepareData(filenameLO,filenameSIG)**, executing **computeQuadratures**, which integrates the data around the locations of the LO pulses to obtain the quadratures. Different options can be set:

- Detection channels to be calculated: e.g.
  seriesQuadratures('Channels',1:3).

- Piezo modulation usage, in order to cut the data into pieces according to the back and forth sweeping of the piezo:
  seriesQuadratures('Channels',1:3,'Piezo','yes'); (or 'Piezo','no')

- Numerical removal of correlations between adjacent quadratures. It should be used for any signal with random phase, but it does not work if the signal has a fixed phase with respect to the LO. It should be set for each of the three channels, e.g. **'CorrRemove',[true,true,true] or [false,false,false]**.

- Subtraction of an offset: This should be done for each set of 1000 pulses individually when there is a random phase between LO and signal (**'Offset',["local" "local" "local"]**) or for the whole dataset when there is a fixed phase (**'Offset', ["global" "global" "global"]**).

- An example for a signal with random phase:
  seriesQuadratures('Channels',1:3,'Offset',["local" "local" "local"], 'Piezo','yes','CorrRemove',[true,true,true]);

- An example for a signal with non-random phase:
  seriesQuadratures('Channels',1:3,'Offset',["global" "global" "global"], 'Piezo','yes','CorrRemove',[false,false,false]);

The computed quadratures X should be a matrix with size 999 x A x B, or 999 x A when no piezo was used, since B is the number of piezo path segments. The first dimension stems from the badges of 1000 pulses as which the data are stored. One pulse is removed after the numerical removal of correlations. If the length is significantly more or less than 999, this means the pulses are not recognized correctly, hinting at overexposure of the detectors.

A.2 Troubleshooting

If anything goes wrong, it might be helpful to look at the data itself.

- Load raw data using **data8bit = load8BitBinary(filename,'dontsave');** Plot this to see irregularities.

- Examine the variances of LOonly data with
  plotPointwiseVariance(data8bit(:,:,channel));
  This should show a peak for each LO pulse, but otherwise have a steady offset. If the offset is strangely varying, this might come from overexposure of the detectors.

- Plot the LOonly data of each channel and look if it is steady around zero. If it has a varying offset, this hints at mechanical instability.

- Plot the computed quadratures, e.g. **plot(X1(:),'.')** and **histogram(X1)**. For a signal with random phase, the quadratures should be distributed around zero. For a thermal signal, it should be a Gaussian distribution. For a coherent signal with random phase, the distribution should have two peaks symmetrical around zero.

A.3 Computation of $g^{(2)}(0)$

After the quadratures are computed as explained above and stored in the folder "mat-data", we can compute the second-order photon correlation and plot it versus time.

- Computation of photon number and $g^{(2)}(0)$ for a series of datasets, e.g.:
  g2SeriesFromQuadratures(nResolution,'Weight','yes','UseX','X1', 'Parameter','power');
  This function creates time-resolved plots of $g^{(2)}(0)$ and photon number for each dataset. It stores the time-resolved $g^{(2)}(0)$ values **g2vec**, the photon number values **ada** and the time vector **time** in a new folder, and it plots the average values of each dataset versus the parameter that was varied in the series. With **nResolution**, choose the number of quadratures that is used to compute one $g^{(2)}(0)$ value. Usually, 1000 is a good choice. 'UseX' sets which quadrature is used, e.g. 'X1' for channel 1. 'Parameter' describes which parameter is varied in the series. When setting 'Weight','yes', the $g^{(2)}(0)$ values are weighted with the corresponding photon numbers when the average is computed for each dataset.

- For a single quadrature dataset X, the time-resolved values can be computed via **[g2vec, ada, time] = g2(X, nResolution)**;

- Different averaging frequencies can be compared with **g2compareSamplingRate(X, filename)**, e.g. for the evaluation in Chapter 6.

A.4 Evaluation of 2 Channel Measurements of Husimi functions

As an example, let us assume that channels 1 and 2 were used for the measurement and in channel 1, the piezo was modulated.

- First, the quadratures are computed with
  seriesQuadratures('Channels',1:2,'Offset',["local" "local" "local"], 'Piezo','yes','CorrRemove',[true,true,true]);

- For a series of measurements, e.g. a power series, the Husimi functions can be evaluated with

HusimiSeriesFromQuadratures([3,1,2],'LoadExistent',false, 'PlotErrorbars',true,'FitMethod','NLSQ-LAR','Scale',true, 'PlotOption',true,'MonteCarloError',true,'Parameter','power');
This function plots the Husimi function for each dataset, fits the Husimi function to the theoretical model of the displaced thermal state, and plots the coherent and thermal photon number and the quantum coherence derived from the fit. If there is a big variation of the photon number during one measurement, the Husimi function is derived for the instances with high and with low photon number separately.
Thereby, [3,1,2] assigns which channel was used for what. In general, [a,b,c] assigns the target channel a, which is not used here, the channel with piezo modulation b and the remnant channel c. 'LoadExistent' can be set true to speed up the evaluation when the orthogonal quadratures O1,O2 and the photon numbers have already been computed and saved in the quadrature datasets. 'FitMethod','NLSQ-LAR' sets the best method for fitting the Husimi function to the theoretical model. With 'Scale',true, the quadratures of channel 1 and 2 are scaled to account for a difference in their photon numbers. With 'MonteCarloError',true, an error is computed for all quantities. Set it to false to save time. With 'Parameter', set which parameter was varied during the series and is indicated in the filenames so it can be read out by Matlab ('no', if no parameter was varied). E.g. for a power series, filenames have the structure "02-xxmW-yymW-LOwithDL.raw", where xx is the excitation power and yy the LO power.

- Within this function, **plotHusimiAndCut** is used to plot and fit the Husimi function.

A.5 Evaluation of 3 Channel Measurements of regularized P functions

Let us assume that channel 3 is the target channel, channel 1 is the postselection channel with piezo modulation, and channel 2 is the remnant postselection channel. A series with different time delays of channel 3 was measured.

- Again, we start by computing the quadratures with
seriesQuadratures('Channels',1:3,'Offset',["local" "local" "local"], 'Piezo','yes','CorrRemove',[true,true,true]);

- Then, the series can be processed for different postselection parameters; i.e. we select different phase-space regions from the Husimi function in the two postselection channels and for each selection, we create a regularized P function from the target data, for each of the time delays.

In the first run, we use only one postselection region. In this run, the computed phases and orthogonal quadratures are stored in the quadrature datasets so they can be used in the next runs to save time. We start e.g. with (numbers are examples from a real measurement)
zeroDelay = 108.96; This is the delay line position that corresponds to zero time difference between the target channel and the other channels.
listOfParams = struct('Type',{'fullcircle'},'Position',{[7 0.4]});
This selects a ring in the Husimi function with a radius of 7 and a thickness of 0.4. Attention: When using the normalization for quadratures $\Delta X_{vacuum} = 1$, i.e. $A = 1$, these are later multiplied with $\sqrt{2}$.
makeSelectionPlots('table','ListOfParams',listOfParams,'Period',4, 'RecomputeTheta',true,'SaveTheta',true,'RecomputeOrth',true, 'saveOrth',true,'SavePostselection',true, 'ChannelAssignment',[3,1,2], 'ZeroDelay',zeroDelay,'Parameter','delay','RemoveModulation',true, 'Range',[0 20],'XUnit','ps','VaryAPS',false,'CorrRemove','yes', 'ZeroDelay',zeroDelay,'MeanNs',[13.8739,14.8419,9.3581]);

- In the next runs, the postselection is done for many more radii, whereby the phase and orthogonal quadratures are used that were computed and saved in the previous step. Therefore, we set some arguments as false, e.g.
  t = 0.4;
  listOfParams = struct('Type',{'fullcircle'},'Position',{[2 t],[3,t],[4,t],[5 t],[6 t],[7 t],[8 t]});
  makeSelectionPlots('table','ListOfParams',listOfParams,'Period', 4,'RecomputeTheta',false,'SaveTheta',false,'RecomputeOrth',false, 'saveOrth',false,'SavePostselection',true,'ChannelAssignment',[3,1,2], 'ZeroDelay',zeroDelay,'Parameter','delay','RemoveModulation',true, 'Range',[0 20],'XUnit','ps','VaryAPS',false,'CorrRemove','yes', 'ZeroDelay',zeroDelay,'MeanNs',[13.8739,14.8419,9.3581]);
  This function creates a table with postselected values, as well as storing the post-selected data in the folder “post-data”.
  'ChannelAssignment': This sets which channel is the target, which is the postselection channel where the piezo is modulated for the phase computation and which is the postselection channel without piezo modulation, according to [target,ps-piezo,ps-no-piezo].
  The ‘removeModulation’ option removes longterm drifts of the signal by scaling it to the mean photon number over all time delays set by 'MeanNs'. (The values have to be determined from the data). Also, random jumps of the signal outside a certain range of photon numbers set with 'Range' are removed.

- **makeSelectionPlots** uses **makeDelayPlots**, which in turn runs **series3Ch** for each set of postselection parameters. Also, **makeSelectionPlots** uses **plotSeriesPostselections**, which makes different types of plots if you set e.g. 'radiusPlots' instead of 'table' when running **makeSelectionPlots**.

A.5.1 Computation of pattern functions for the regularized P function

Before the regularized P function can be reconstructed from the data, the pattern functions for the P function have to be computed once. In our case, this is done for a normalization of quadratures such that the fluctuations of a vacuum field $\Delta X_{vacuum} = 1$, i.e. $A = 1$. The Matlab function **mainCalcGridsAndPatterns(varargin)** creates all necessary coordinate grids and pattern functions and saves them for given parameters in a given directory. Here we used the following parameters, whereby 'R' is the filter parameter for the regularized P function:
mainCalcGridsAndPatterns('MaxQuad',20,'MaxX',20,'PhiStep',0.1,'R',0.7, 'XStep',1,'Resolution',0.25);
This function executes the following procedure:

- **[xGrid,phiGrid] = makeGridxAndPhi(maxX,xStep,phiStep);**
  This creates a grid of position intervals and phase intervals (x, phi), which are defined by the parameters 'PhiStep', 'MaxX' and 'XStep'. This is the coordinate grid to which the measured data is compared, being called old grid.

- **makeXGridAndPattern(xGrid,phiGrid,R,maxQuad,Resolution, directory);**
  For each position in the old grid, so-called X parameters (XGrid) (cf. Eq. (C.14)) and the corresponding pattern function (Eq. (C.14) and Eq. (C.15)) are computed and stored. Both are saved in a file called e.g. 'x-0.5-phi-0.1.mat', as for each position (x, phi) in the old grid, one file is created. For computing the X parameters, we define a new grid of quadratures, specified by 'maxQuad' and 'Resolution'. These new quadratures provide the phase-space coordinates on which the regularized P function is displayed.

A.5.2 Computing the P functions and plotting the results

Then, the regularized P functions are calculated from the postselected data, e.g. with these parameters:

- **zeroDelay=108.96; t = 0.4;**
  listOfParams = struct('Type',{'fullcircle'},'Position',{[2 t],[3,t],[4,t],[5 t],[6 t],[7 t],[8 t]});
  for iParams = 1:length(listOfParams)
  PFunctionDelaySeries('LoadExistent',true,'Plot',false,'ZeroDelay',

zeroDelay,’SelectionParameters’,listOfParams(iParams), ’RemoveModulation’,true,’Range’,[0 20],’MaxQuad’,20,’MaxX’,20, ’PhiStep’,0.1,’Rvalue’,0.7,’XStep’,1,’Resolution’,0.25);
end
Set ’Plot’,true, for plotting each P function. Also it is necessary to set the path where the pattern functions have been stored; see inside the Matlab script.

- Then, the expectation values from the P functions can be plotted versus delay via **plotPFunctionResults(listOfParams,’Quantity’,{’R’},’fitType’,’exp2sat1’, ’RemoveModulation’, true,’Range’,[0 20],’MaxQuad’,20,’MaxX’,20, ’PhiStep’,0.1,’Rvalue’,0.7,’XStep’,1,’Resolution’,0.25,’Norm’,1);**
  With ’Quantity’, set for which quantity the expectation value is plotted, e.g. {’R’} for the amplitude and {’circVa1’} for the circular variance. ’FitType’ can be set to various fit functions. Set it to ‘exp2’ when plotting the mean amplitude, so the offset can be fitted, or to ’envelopeExp2’ in order to fit the envelope of an oscillating curve. The results from the fits are stored into new files.

A.6 Evaluation of Dispersions

For the evaluation of spectroscopic dispersions, a few Matlab functions can be used.

- For a series of different positions on the sample e.g.:
  plotDispersionsPositionSeries(’ZeroPosition’,3.67,’ZoomE’,[1.59 1.64]);
  ’ZeroPosition’ gives the position of the sample edge. ’ZoomE’ gives the plotted energy range in eV.

- For a series of polarizations, polar plots are created via e.g.:
  plotDispersionsPolarisationSeries(’minY’,213,’xAperture’,142, ’GetTime’,’no’,’Fit’,’useOld’,’E0Old’,E0,’aOld’,a,’y0’,y0,’ModeK’, [-0.5 0.5],’ModeE’,[1.6103 1.6109],’AdjustEnergy’,true);
  Here, ’minY’, and ’xAperture’ give the calibration of the k space. ’GetTime’ toggles whether the intensity is divided by the acquisition time. ’Fit’, ’useOld’, ’E0Old’, E0, ’aOld’, a, ’y0’, y0 draws a parabola into the dispersion for which the parameters have to be determined from a dispersion below threshold. ’ModeK’ and ’ModeE’ define the rectangle in which the intensity is integrated. ’AdjustEnergy’ sets whether the energy of the rectangle is adapted to account for a red- or blue shift during the series.

- For a power series, an I-O curve is created via e.g.:
  plotDispersionsPowerSeries(’minY’,214,’xAperture’,142,’GetTime’, true,’Fit’,’useOld’,’E0Old’,E0,’aOld’,a,’y0’,y0,’ModeK’, [-0.5 0.5],’ModeE’,[1.6101 1.6111],’adjustEnergy’,true);