

!YOUR TITLE ALL CAPS!

By Joel N. Johnson

A Dissertation

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in Applied Physics and Materials Science

Northern Arizona University

!Month YYYY!

Ryan O. Behunin, Ph.D., Co-Chair

John G. Gibbs, Ph.D., Co-Chair

Inès Montaña, Ph.D.

Jennifer S. Martinez, Ph.D.

Table of Contents

List of Tables	iv
List of Figures	v
Dedication	vii
Preface	ix
1 Introduction	1
1.1 Spontaneous Brillouin Scattering	1
1.2 Stimulated Brillouin Scattering	1
1.3 Phase-matching	2
1.4 Brillouin Gain of Materials	2
1.5 Raman Scattering	2
1.6 Raman-like Brillouin Modes	2
2 Foundational Experimental Techniques and Instrumentation	5
2.1 Experimental Techniques	5
2.1.1 ways we can direct light in a photonic system	6
2.1.2 photonic devices and diagrams	6
2.1.3 ways we can select and isolate signals	6
2.1.4 heterodyne detection and the role of the LO	6
2.1.5 loss in a photonic system	6
2.1.6 free space optics and beam alignment	6
2.1.7 special fiber types and properties	6
2.2 Optical Instrumentation	6
2.3 Electronic Instrumentation	6
2.4 Noise and Background Handling	7
2.5 Custom Software	7
2.5.1 Description of Python Script for CABS Data Collection	7
2.5.2 Description of Plotting Data in Go Program	8
3 Manuscript I: Laser cooling of traveling wave phonons in an optical fiber	9
3.1 Optomechanical Cooling and Heating	9
3.2 Cooling Platform: CS_2 -Liquid Core Optical Fiber	9
3.2.1 Optomechanical Properties	9
3.2.2 Fabrication	9
3.2.3 Fabrication Iterative Refinement	9
3.3 Intention of the Pump-Probe Experiment	9
3.4 Experimental Setup	9
3.4.1 Main Experiment	9
3.4.2 Pump-Probe Experiment	9
3.5 Results	9
3.5.1 Main Experiment Results	9

3.5.2	Pump-Probe Experiment Results	9
3.6	Discussion	9
3.6.1	Application to Ground State Cooling	9
3.6.2	Standardized Cooling Metric	9
3.6.3	Synchronous Achievement by Max Plank Group	9
3.6.3.1	Platform: Tapered chalcogenide Photonic Crystal Fiber	9
4	Manuscript II: A coherently stimulated phonon spectrometer	11
4.1	Abstract	11
4.2	Introduction	11
4.2.1	Theory of CABS	11
4.2.2	Phase-matching at short lengths	11
4.3	Methods	11
4.3.1	Theory of CABS	11
4.3.2	Phase-matching bandwidth	12
4.4	Results	12
4.4.1	Design of instrument	12
4.4.2	From fiber-coupled to micrometer-scale free-space	12
4.4.3	Relaxation of Phase-matching conditions	12
4.5	Discussion	12
4.6	Acknowledgements	12
4.7	Appendix	12
4.7.1	Equal contribution of P, S, Pr	12
5	Manuscript III: Brillouin-induced Raman modes	15
5.1	Abstract	15
5.2	Introduction	15
6	Manuscript IV: Nanoscale Brillouin scattering	17
6.1	Abstract	17
6.2	Introduction	17
7	Discussion & Conclusion	19
A	Acronyms	21
B	Code	23
B.1	Python Code for CABS Data Collection	23
B.2	Plotting Data In Go Program	28
C	Supplementary Information for Chapter 3: Manuscript I	91
	References	92

List of Tables

5.1	Table caption.	16
6.1	Table caption.	18

List of Figures

4.1	CABS measurement of 100um of CS2.	13
-----	---	----

Dedication

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Preface

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 1

Introduction

This is an inline citation, Boyd (2020). This is a parenthetical citation (Boyd, 2020). This is a figure reference (Figure ??). This is a section reference §??. This is a chapter reference with chapter spelled out: ???. This is an acronym definition American Geophysical Union (AGU). This is the second time I use the acronym in this section AGU. This is if I want to spell out the full acronym again American Geophysical Union (AGU). Define new acronyms in the acronyms.tex file.

1.1 Spontaneous Brillouin Scattering

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2 Stimulated Brillouin Scattering

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium

quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.3 Phase-matching

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.4 Brillouin Gain of Materials

1.5 Raman Scattering

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.6 Raman-like Brillouin Modes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a,

magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada
fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna
fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium
quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla,
malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus.
Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim
rutrum.

Chapter 2

Foundational Experimental Techniques and Instrumentation

This is an inline citation, Boyd (2020). This is a parenthetical citation (Boyd, 2020). This is a figure reference (Figure ??). This is a section reference §??. This is a chapter reference with chapter spelled out: ??. This is an acronym definition American Geophysical Union (AGU). This is the second time I use the acronym in this section AGU. This is if I want to spell out the full acronym again American Geophysical Union (AGU). Define new acronyms in the acronyms.tex file.

2.1 Experimental Techniques

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.1.1 ways we can direct light in a photonic system

2.1.2 photonic devices and diagrams

2.1.3 ways we can select and isolate signals

2.1.4 heterodyne detection and the role of the LO

2.1.5 loss in a photonic system

2.1.6 free space optics and beam alignment

2.1.7 special fiber types and properties

2.2 Optical Instrumentation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.3 Electronic Instrumentation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.4 Noise and Background Handling

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.5 Custom Software

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.5.1 Description of Python Script for CABS Data Collection

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.5.2 Description of Plotting Data in Go Program

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 3

Manuscript I: Laser cooling of traveling wave phonons in an optical fiber

3.1 Optomechanical Cooling and Heating

3.2 Cooling Platform: CS_2 -Liquid Core Optical Fiber

3.2.1 Optomechanical Properties

3.2.2 Fabrication

3.2.3 Fabrication Iterative Refinement

3.3 Intention of the Pump-Probe Experiment

3.4 Experimental Setup

3.4.1 Main Experiment

3.4.2 Pump-Probe Experiment

3.5 Results

3.5.1 Main Experiment Results

3.5.2 Pump-Probe Experiment Results

3.6 Discussion

3.6.1 Application to Ground State Cooling

3.6.2 Standardized Cooling Metric

3.6.3 Synchronous Achievement by Max Plank Group

3.6.3.1 Platform: Tapered chalcogenide Photonic Crystal Fiber

Chapter 4

Manuscript II: A coherently stimulated phonon spectrometer

Joel N. Johnson^{1,2}, Nils T. Otterstrom³, Peter T. Rakich⁴, Ryan O. Behunin^{1,2}

This is the Accepted Manuscript version of an article accepted for publication in Nature Photonics. Wiley Inc is not responsible for any errors or omissions in this version of the manuscript or any version derived from it. The Version of Record is available online at <https://doi.org/>.

4.1 Abstract

4.2 Introduction

State of brillouin microscopy Applications and usefulness Challenges: selection of backscattered signal conflated with Stokes field phase-matching requires probe wavelength to be exactly that of Stokes Wouldn't it be nice if we could break free of strict phase-matching requirements, therefore perfectly isolating the signal In this work

4.2.1 Theory of CABS

description of physics with scattered power equation

4.2.2 Phase-matching at short lengths

phase-matching bandwidth description with equation

4.3 Methods

4.3.1 Theory of CABS

full CABS theory arriving at scattered power

¹ Department of Applied Physics and Materials Science, Northern Arizona University, Flagstaff, AZ 86011, USA

² Center for Materials Interfaces in Research and Applications, Flagstaff, AZ 86011, USA

³ Sandia National Laboratory, 1515 Eubank Blvd SE, Albuquerque, NM 87123, USA

⁴ Department of Applied Physics, Yale University, New Haven, CT 06520, USA

4.3.2 Phase-matching bandwidth

phase-matching bandwidth theory

4.4 Results

4.4.1 Design of instrument

description of design figure: instrument apparatus design sensitivity measurements

4.4.2 From fiber-coupled to micrometer-scale free-space

figure: demonstration measurements 1mm uhna3 fiber 1mm CS2 bulk

comparison to stimulated brillouin and spontaneous brillouin?

4.4.3 Relaxation of Phase-matching conditions

figure: phase-matching peak vs pump-probe separation 1cm uhna3, CS2 peak vs pump-probe separation 1mm uhna3, CS2

4.5 Discussion

4.6 Acknowledgements

4.7 Appendix

4.7.1 Equal contribution of P, S, Pr

figure: P, S, Pr equal contributors

100 μm CS₂ CABS

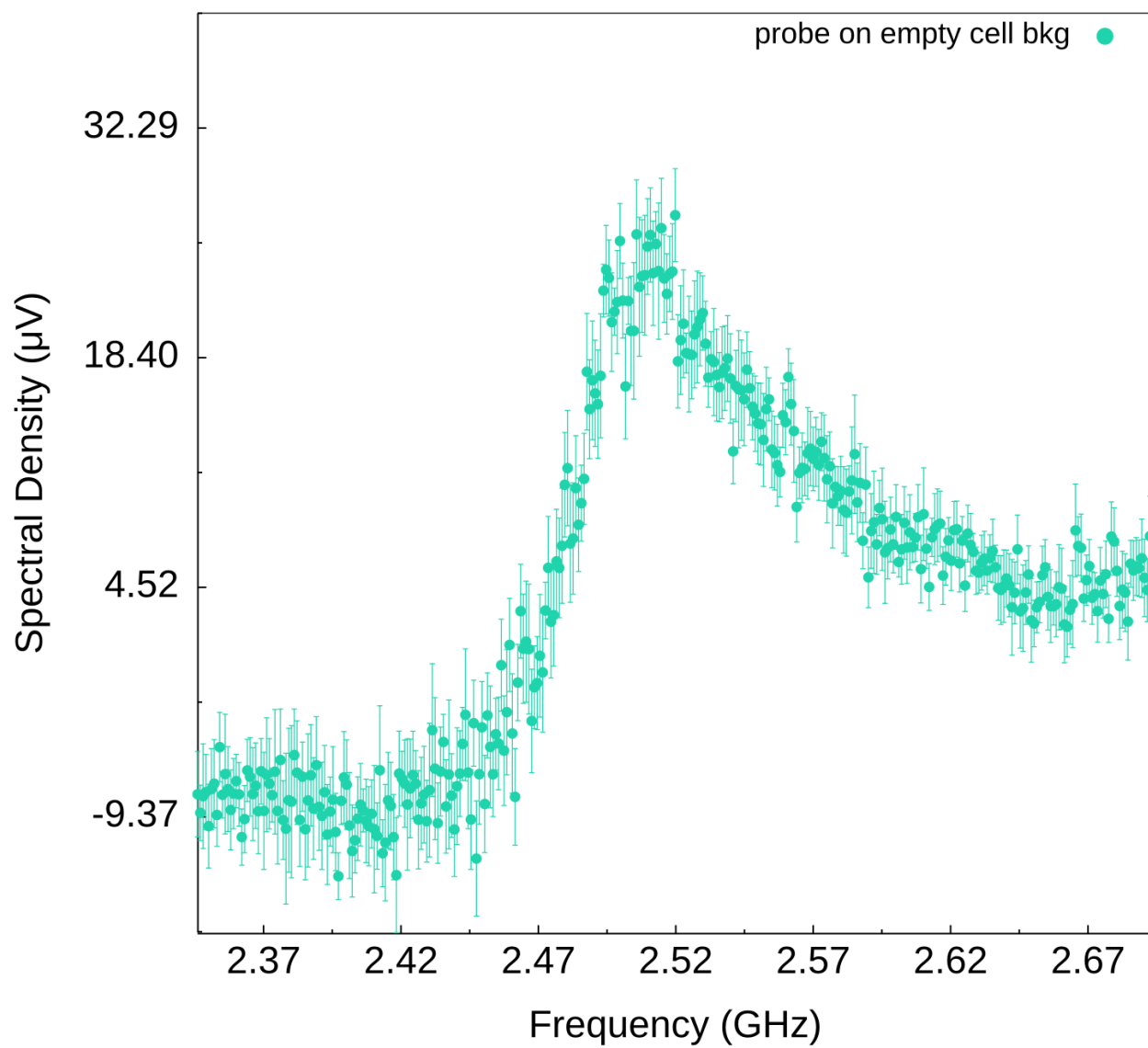


Figure 4.1: CABS measurement of 100 μm of CS₂.

Chapter 5

Manuscript III: Brillouin-induced Raman modes

5.1 Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

5.2 Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Table 5.1: Table caption.

	Parameter	Value	Description
Lookup Variables	lat	-85°–85°	Latitude (35 bins in 5° increments)
	ALBEDO	0.05–0.225	Bolometric albedo (6 bins in 0.035 increments)
	SLOPE	0°–90°	Surface slope (19 bins in 5° increments)
	SLOAZI	0°–360°	Surface azimuth (19 bins in 20° increments)
	DELLS	4°	L_s step size (90 bins spanning 0°–360°)
Thermal Parameters	EMISS	0.96	Emissivity
	thick	0.05	Upper layer thickness [m]
	DENSITY	1100	Upper layer density [kg/m ³]
	DENS2	1800	Lower layer density [kg/m ³]
	lbound	18	Interior heat flow [mW/m ²]
	PhotoFunc	0.045/albedo	Photometric function (Keihm-style)
Temperature-dependent parameters	SphUp0/SphLo0	602.88098583	Specific heat capacity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	SphUp1/SphLo1	235.98988249	
	SphUp2/SphLo2	-29.59742178	
	SphUp3/SphLo3	-3.78707193	
	ConUp0	0.00133644	Upper layer conductivity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	ConUp1	0.00073150	
	ConUp2	0.00033250	
	ConUp3	0.00005038	
	ConLo0	0.00634807	Lower layer conductivity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	ConLo1	0.00347464	
	ConLo2	0.00157938	
	ConLo3	0.00023930	
Model Setup Parameters	body	Moon	Target body
	k.style	Moon	Conductivity style (Moon for airless bodies)
	LKofT	T	Temperature-dependent conductivity
	FLAY	0.01	First layer thickness [m]
	RLAY	1.3	Layer thickness multiplier
	N1	26	Number of layers
	N24	288	Timesteps per day (5 min steps)
	DJUL	0	Start date

Chapter 6

Manuscript IV: Nanoscale Brillouin scattering

6.1 Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6.2 Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Table 6.1: Table caption.

	Parameter	Value	Description
Lookup Variables	lat	-85°–85°	Latitude (35 bins in 5° increments)
	ALBEDO	0.05–0.225	Bolometric albedo (6 bins in 0.035 increments)
	SLOPE	0°–90°	Surface slope (19 bins in 5° increments)
	SLOAZI	0°–360°	Surface azimuth (19 bins in 20° increments)
	DELLS	4°	L_s step size (90 bins spanning 0°–360°)
Thermal Parameters	EMISS	0.96	Emissivity
	thick	0.05	Upper layer thickness [m]
	DENSITY	1100	Upper layer density [kg/m ³]
	DENS2	1800	Lower layer density [kg/m ³]
	lbound	18	Interior heat flow [mW/m ²]
	PhotoFunc	0.045/albedo	Photometric function (Keihm-style)
Temperature-dependent parameters	SphUp0/SphLo0	602.88098583	Specific heat capacity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	SphUp1/SphLo1	235.98988249	
	SphUp2/SphLo2	-29.59742178	
	SphUp3/SphLo3	-3.78707193	
	ConUp0	0.00133644	Upper layer conductivity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	ConUp1	0.00073150	
	ConUp2	0.00033250	
	ConUp3	0.00005038	
	ConLo0	0.00634807	Lower layer conductivity expressed as 4th-order polynomial ($c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3$)
	ConLo1	0.00347464	
	ConLo2	0.00157938	
	ConLo3	0.00023930	
Model Setup Parameters	body	Moon	Target body
	k.style	Moon	Conductivity style (Moon for airless bodies)
	LKofT	T	Temperature-dependent conductivity
	FLAY	0.01	First layer thickness [m]
	RLAY	1.3	Layer thickness multiplier
	N1	26	Number of layers
	N24	288	Timesteps per day (5 min steps)
	DJUL	0	Start date

Chapter 7

Discussion & Conclusion

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Appendix A

Acronyms

AGU American Geophysical Union

Appendix B

Code

B.1 Python Code for CABS Data Collection

```
1 import csv
2 import visa
3 import time
4 import datetime
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import zhinst.ziPython, zhinst.utils
8 import os
9 import winsound
10
11 daq = zhinst.ziPython.ziDAQServer('localhost', 8005)
12 device = zhinst.utils.autoDetect(daq)
13
14 startTime = datetime.datetime.now()
15
16 dwell = .1
17 F_lockin = 45.0000E6
18 F_AOM = 40.000800e6
19 chan = 1
20 ch = str(chan-1)
21 rate=2e9 # rate, per millisecond
22 RBW = 100 # lock-in bandwidth
23 tc=1/(2*np.pi*RBW) # time constant
24 lockRange = .04
25 order = 8
26
27 exp_set = [
28     [['/', device, '/sigins/',ch,'/diff'], 0],
29     [['/', device, '/sigins/',ch,'/imp50'], 1],
30     [['/', device, '/sigins/',ch,'/ac'], 1],
31
32     # Want range as low as possible without clipping data
33     # (red Over light on Lock-in box)
34     [['/', device, '/sigins/',ch,'/range'], lockRange],
35     [['/', device, '/demods/',ch,'/order'], order],
36     [['/', device, '/demods/',ch,'/timeconstant'], tc/3.33],
37     [['/', device, '/demods/',ch,'/rate'], rate],
38     [['/', device, '/demods/',ch,'/adcselect'], chan-1],
39     [['/', device, '/demods/',ch,'/oscselect'], chan-1],
40     [['/', device, '/demods/',ch,'/harmonic'], 1],
41     [['/', device, '/oscs/',ch,'/freq'], F_lockin],
42 ]
43
44 daq.set(exp_set);
45 time.sleep(.001)
46
47 path = '/%/s/demods/%d/sample' % (device, 0) # (device, demod_index)
48 daq.subscribe(path)
49 daq.flush()
```

```

50 daq.sync()
51
52 def measureSynchronousFeedback(daq, device, channel, frequency):
53     c=str(channel-1) #return a string of an object. =channel-1=-1 for channel 0
54
55     # Poll the subscribed data from the data server. Poll will block and record
56     # for poll_length seconds.
57     daq.sync()
58     daq.flush()
59     poll_length = dwell # [s]
60     poll_timeout = 10 # [ms]
61     poll_flags = 0
62     poll_return_flat_dict = True
63     data = daq.poll(
64         poll_length,
65         poll_timeout,
66         poll_flags,
67         poll_return_flat_dict
68     )
69     assert data, ""poll() returned an empty data dictionary,
70                 did you subscribe to any paths?""
71
72     # Access the demodulator sample using the node's path.
73     sample = data[path] # Defines the sample as the data from defined path
74
75     # Calculate the demodulator's magnitude and adds it to the dict.
76     global sampleR
77     sampleR = np.abs(sample['x'] + 1j*sample['y']) #y-axis #magnitude
78
79     clockbase = float(daq.getInt('/%s/clockbase' % device))
80
81     # Convert timestamps from ticks to seconds via clockbase.
82     t = (sample['timestamp'] - sample['timestamp'][0])/clockbase
83
84     resources = visa.ResourceManager()
85     resources.list_resources()
86
87     #open the signal generator
88     #gen=resources.open_resource('USB0::0x03EB::0xAFFF::6C2-0A2A2000A-0374::INSTR')
89     gen=resources.open_resource('USB0::0x03EB::0xAFFF::6C2-0A2B2000A-0430::INSTR')
90
91     gen.write('FREQUENCY:MODE CW DUAL')
92     time.sleep(0.01)
93     gen.write('OUTPUT1 1')
94     time.sleep(0.01)
95     gen.write('OUTPUT2 1')
96
97     # run a cycle of frequencies for each output
98     Fstart = 9.0*1E9
99     Fstop = 9.28*1E9
100    F_step = 0.00500456*1E9
101
102    N_steps = np.int((Fstop-Fstart)/F_step)
103
104    NoRealizations = 5
105    dataR = [0]*N_steps # demod data
106    dataF = [0]*N_steps # Freq. axis
107    stdDevOfMeanR = [0]*N_steps # to hold standard deviation of dwell-time data
108    f2 = Fstart
109    F = Fstart
110
111
112    run = 1
113    folderName = startTime.strftime("%y-%m-%d ") + ""1cm UHNA3""
114    while os.path.exists(folderName + "/" + str(run) + "/signal.csv"):
115        run += 1
116    folderRunName = folderName + "/" + str(run)
117

```

```

118 signalData = 1
119 takeBkrdData = 0
120
121 aveDataR = [0]*N_steps
122
123 if not os.path.exists(folderRunName):
124     os.makedirs(folderRunName)
125
126 if not os.path.exists(folderName + "/meta.csv"):
127     meta = open(folderName + "/meta.csv", "a")
128     meta.write("""Date,Label,Sets,Start Time,End Time (hr:min:sec),Pump,Stokes,
129               Probe,Frequency,Signal,Range,Dwell,Bandwidth,Data Rate,Order,
130               Start Frequency,Stop Frequency,Step,Num Avgs,Pump Laser,Probe
131               Laser,Probe Filter,Stokes Filter,Notes\n""")
132     meta.close()
133
134 if takeBkrdData == 1:
135     bkrdStartTime = datetime.datetime.now()
136
137     bgDataR = np.zeros((NoRealizations, N_steps))
138
139 for kk in range(0,NoRealizations):
140     F = Fstart
141     print(kk)
142     for jj in range(0,N_steps):
143         f1 = F
144         f2 = F + F_AOM - F_lockin
145         time.sleep(0.1)
146         gen.write("SOURCE1: FREQUENCY:CW "+ str(f1)) # Hz
147         time.sleep(1e-3)
148         gen.write("SOURCE2:FREQUENCY:CW "+ str(f2)) # Hz
149         time.sleep(1e-3)
150         measureSynchronousFeedback(daq, device, 1, F_lockin)
151         time.sleep(1e-3)
152
153         F=F+F_step
154
155         dataF[jj] = Fstart+jj*F_step
156         ff1 = f1*10**-6
157         ff2 = f2*10**-6
158
159         dataR[jj]=np.mean(sampleR)
160         stdDevOfMeanR[jj] = np.std(sampleR)/np.sqrt(len(sampleR))
161
162 #-----save run data (dataR and stdDevR)-----#
163 if signalData == 1:
164     runSigDir = folderRunName + "/Runs/Signal/"
165     if not os.path.exists(runSigDir):
166         os.makedirs(runSigDir)
167
168     csvfile = runSigDir + "Run " + str(kk) + ".csv"
169     with open(csvfile, "w") as output:
170         writer = csv.writer(output, delimiter=',')
171         writer.writerow(['Sig','Std Dev'])
172         for sig, std in zip(dataR, stdDevOfMeanR):
173             writer.writerow([sig, std])
174
175 elif takeBkrdData == 1:
176     runBgDir = folderRunName + "/Runs/Background/"
177     if not os.path.exists(runBgDir):
178         os.makedirs(runBgDir)
179
180     csvfile = runBgDir + "Run " + str(kk) + ".csv"
181     with open(csvfile, "w") as output:
182         writer = csv.writer(output, delimiter=',')
183         writer.writerow(['Sig','Std Dev'])
184         for sig, std in zip(dataR, stdDevOfMeanR):
185             writer.writerow([sig, std])

```

```

186 #-----#
187
188 aveDataR = (np.array(dataR)+kk*np.array(aveDataR))/(kk+1)
189
190 if takeBkrdData == 1:
191
192     bgDataR[kk] = dataR
193     background = aveDataR
194
195     plt.figure()
196     plt.grid(True)
197     plt.plot(dataF, background)
198     plt.title('Background Demodulator data')
199     plt.show()
200
201 if takeBkrdData == 0:
202     plt.figure()
203     plt.grid(True)
204     plt.plot(dataF, aveDataR-background)
205     plt.title('Signal - Background Demodulator data')
206     plt.xlabel('F')
207     plt.ylabel('R')
208     plt.show()
209
210 #-----save subtracted run data (dataR)-----#
211 if signalData == 1:
212     runSubtrDir = folderRunName + "/Runs/Subtracted/"
213     if not os.path.exists(runSubtrDir):
214         os.makedirs(runSubtrDir)
215
216     csvfile = runSubtrDir + "Run " + str(kk) + ".csv"
217     with open(csvfile, "w") as output:
218         writer = csv.writer(output, lineterminator='\n')
219         for val in dataR - bgDataR[kk]:
220             writer.writerow([val])
221 #-----#
222
223 if signalData == 1:
224     csvfile=folderRunName+"/signal.csv"
225     with open(csvfile, "w") as output:
226         writer=csv.writer(output, lineterminator='\n')
227         for val in aveDataR-background:
228             writer.writerow([val])
229
230     csvfile=folderRunName+"/frequency.csv"
231     with open(csvfile, "w") as output:
232         writer=csv.writer(output, lineterminator='\n')
233         for val in dataF:
234             writer.writerow([val])
235
236 if not os.path.exists(folderRunName + "/timestamp.csv"):
237     timestampf = open(folderRunName + "/timestamp.csv", "a")
238     timestampf.write("""Date,Label,Run,Data,Start Time,End Time (hr:min:sec),
239 Pump,Stokes,Probe,Frequency,Signal,Range,Dwell,Bandwidth,
240 Data Rate,Order,Start Frequency, Stop Frequency,Step,Num
241 Avgs,Notes\n""")
242     timestampf.close()
243
244 if takeBkrdData == 1:
245     bkrdEndTime = datetime.datetime.now()
246     timestampf = open(folderRunName + "/timestamp.csv", "a")
247     timestampf.write(
248         bkrdStartTime.strftime("%y-%m-%d,") + str(run) + ",Background" +
249         bkrdStartTime.strftime(",%H:%M:%S") +
250         bkrdEndTime.strftime(",%H:%M:%S,,,,,")
251     )
252     timestampf.write(
253         str(lockRange) + "," + str(dwell) + "," + str(RBW) + "," +

```

```

254         str("{:e}".format(rate)) + "," + str(order) + "," +
255         str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
256         str("{:e}".format(F_step)) + "," + str(NoRealizations) + ",\n"
257     )
258     timestampf.close()
259
260 if signalData == 1:
261     endTime = datetime.datetime.now()
262     timestampf = open(folderRunName + "/timestamp.csv", "a")
263     timestampf.write(
264         startTime.strftime("%y-%m-%d,") + str(run) + ",Signal" +
265         startTime.strftime(",%H:%M:%S") + endTime.strftime(",%H:%M:%S,,,,")
266     )
267     timestampf.write(str(run) + "/frequency.csv," + str(run) + "/signal.csv,")
268     timestampf.write(
269         str(lockRange) + "," + str(dwelling) + "," + str(RBW) + "," +
270         str("{:e}".format(rate)) + "," + str(order) + "," +
271         str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
272         str("{:e}".format(F_step)) + "," + str(NoRealizations) + ",\n"
273     )
274     timestampf.close()
275
276 meta = open(folderName + "/meta.csv", "a")
277 meta.write(
278     startTime.strftime("%y-%m-%d,") + str(run) +
279     startTime.strftime(",%H:%M:%S") + endTime.strftime(",%H:%M:%S,,,,")
280 )
281 meta.write(str(run) + "/frequency.csv," + str(run) + "/signal.csv,")
282 meta.write(
283     str(lockRange) + "," + str(dwelling) + "," + str(RBW) + "," +
284     str("{:e}".format(rate)) + "," + str(order) + "," +
285     str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
286     str("{:e}".format(F_step)) + "," + str(NoRealizations) + ",,,,,\n"
287 )
288 meta.close()

```

B.2 Plotting Data In Go Program

```
1 package main
2
3 import (
4     "image/color"
5     "github.com/Arafatk/glot"
6     "github.com/maorshutman/lm"
7     "encoding/csv"
8     "bufio"
9     "fmt"
10    "os"
11    "io"
12    "strconv"
13    "strings"
14    "math"
15    "gonum.org/v1/plot"
16    "gonum.org/v1/plot/plotter"
17    "gonum.org/v1/plot/vg"
18    "gonum.org/v1/plot/font"
19    "gonum.org/v1/plot/vg/draw"
20    "time"
21    "flag"
22    "log"
23 )
24
25 func main() {
26
27     cabs, lock, temp, slide, sinc, sample, coolingExperiment, note, length, csvToAvg := flags
28     ()
29
30     logpath := logpath(note)
31
32     date, label, setNums, startTime, endTime, asPowers, sPowers,
33     pumpPowers, stokesPowers, probePowers, filepath, sigFilepath, freqFilepath,
34     lockinRange, dwell, bandwidth, dataRate, order, startFrequency, stopFrequency,
35     step, numAverages, asNotes, sNotes, pumpLaser, probeLaser, probeFilter, stokesFilter,
36     notes := readMeta(
37         cabs, lock, temp, coolingExperiment,
38     )
39
40     logFile := logHeader(
41         cabs, lock, temp, slide, sample, coolingExperiment, note,
42         length,
43     )
44
45     if coolingExperiment != "" {
46
47          $\sigma$ as,  $\sigma$ s := avgCSVs(csvToAvg, asPowers)
48
49         ras, bas, rs, bs := getCoolingData(lock, filepath, label)
50
51         asLabel, basLabel, sLabel, bsLabel := getAllLabels(label)
52
53         setsToPlotRaw := []int{}
54         plotRaw(
55             setsToPlotRaw,
56             bas, ras, bs, rs,
57             basLabel, asLabel, bsLabel, sLabel,
58         )
59
60         s, as := subtractBackground(ras, bas, rs, bs, coolingExperiment)
61
62         setsToPlotSubtracted := []int{}
63         plotSubtracted(
64             setsToPlotSubtracted,
```

```

65     sLabel, asLabel,
66 )
67
68 setsToPlotSubtractedTogether := []int{}
69 plotSubtractedTogether(
70     setsToPlotSubtractedTogether,
71     as, s,
72     asLabel, sLabel,
73 )
74
75 binSets := []int{} // 0,4,8,12,15 // 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
76 if len(binSets) > 0 {
77     binMHz := 5.
78     as, s = bin(binSets, as, s, binMHz)
79 }
80
81 subtractedGrouped := []int{}
82 if len(subtractedGrouped) > 0 {
83     goPlotSubGrpd(
84         subtractedGrouped, s, as,  $\sigma$ s,  $\sigma$ as, sLabel, asLabel, logpath, sample,
85         coolingExperiment, slide,
86     )
87 }
88
89 fitSets := true
90 if fitSets {
91
92     var amp, wid, cen, c, gb,  $\Gamma$  float64
93
94     if sample == "LCOF" {
95         amp = 2.
96         wid = 0.1
97         cen = 2.275
98         c = .01
99         gb = 6 //  $W^{-1}m^{-1}$ 
100          $\Gamma$  = 98.65 //  $2*\text{math.Pi}$  // MHz
101     } else if sample == "UHNA3" {
102         amp = 12
103         wid = 0.1
104         cen = 9.18
105         gb = 0.6
106          $\Gamma$  = 100
107     } else {
108         sample = "[Unspecified] Sample"
109         amp = 5
110         wid = 0.1
111         cen = 2.25
112         gb = 0
113          $\Gamma$  = 0
114     }
115
116     var asAmps, asLinewidths []float64
117
118     fitAntiStokes := []int{0,1,2,3}
119     if len(fitAntiStokes) > 0 {
120
121         // as
122         header := fmt.Sprintf("\nAnti-Stokes\nSet \t Power \t\t Width \t\t Peak \t\t Center\n")
123         fmt.Printf(header)
124         logFile = append(logFile, header)
125
126         var asFits [][]float64
127         var asWidthLine [][]float64
128         var asWidthLines [][]float64
129         var asfwhm []float64
130
131         for i, set := range fitAntiStokes {

```

```

132     f := func(dst, guess []float64) {
133
134         amp, wid, cen, c := guess[0], guess[1], guess[2], guess[3]
135
136         for i := range as[set][0] {
137             x := as[set][0][i]
138             y := as[set][1][i]
139             dst[i] = (.25 * amp * math.Pow(wid, 2) / (math.Pow(x - cen, 2) + (.25 * math.
140 Pow(wid, 2))) - y) * (1./σas[set][i]) + c
141         }
142     }
143
144     jacobian := lm.NumJac{Func: f}
145
146     // Solve for fit
147     toBeSolved := lm.LMProblem{
148         Dim:      4,
149         Size:      len(as[set][0]),
150         Func:      f,
151         Jac:      jacobian.Jac,
152         InitParams: []float64{amp, wid, cen, c},
153         Tau:      1e-6,
154         Eps1:     1e-8,
155         Eps2:     1e-8,
156     }
157
158     results, _ := lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e
-16})
159
160     amp, wid, cen, c := results.X[0], math.Abs(results.X[1]), results.X[2], results.X
[3]
161
162     asfwhm = append(asfwhm, wid*1000)
163
164     str := fmt.Sprintf("%d \t %.2f mW \t %.2f MHz \t %.6f uV \t %.4f GHz\n", set,
asPowers[set], wid*1000, amp, cen)
165     fmt.Printf(str)
166     logFile = append(logFile, str)
167
168     // Create Lorentzian fit data according to solved fit parameters
169     df := .001
170     f0 := as[set][0][0]
171     fitPts := int((as[set][0][len(as[set][0]) - 1] - f0)/df) + 1
172     asFits = append(asFits, generateFitData(amp, wid, cen, c, f0, df, fitPts))
173
174     // Width lines
175     asWidthLine = [][]float64{{cen - wid/2, cen + wid/2},{amp/2, amp/2}}
176     asWidthLines = append(asWidthLines, asWidthLine)
177
178     // For height ratios
179     asAmps = append(asAmps, amp)
180
181     // For linewidths
182     asLinewidths = append(asLinewidths, asfwhm[i])
183 }
184
185 // goPlot as fits
186 goPlotasFits(
187     fitAntiStokes, as, asFits, asWidthLines, σas, asLabel, asfwhm, asNotes,
188     temp, slide, sample, logpath, coolingExperiment,
189 )
190
191 // goPlot power vs width
192 goPlotasPowerVsWid(
193     fitAntiStokes, asLabel, asNotes, asfwhm, temp, slide, sample, logpath,
194     coolingExperiment,
195 )

```



```

196 }
197
198 fitStokes := []int{0,1,2,3}
199 if len(fitStokes) > 0 {
200
201     header := "\nStokes\nSet \t Power \t\t Width \t\t Peak \t\t Center \n"
202     fmt.Printf(header)
203     logFile = append(logFile, header)
204
205     var sFits [][]float64
206     var sWidthLine []float64
207     var sWidthLines [][]float64
208     var ampRatios []float64
209     var sLinewidths []float64
210     var sfwhm []float64
211
212     for i, set := range fitStokes {
213
214         f := func(dst, guess []float64) {
215
216             amp, wid, cen := guess[0], guess[1], guess[2]
217
218             for i := range s[set][0] {
219                 x := s[set][0][i]
220                 y := s[set][1][i]
221                 dst[i] = (.25 * amp * math.Pow(wid, 2) / (math.Pow(x - cen, 2) + (.25 * math.
Pow(wid, 2))) - y) * (1./σs[set][i]) + c
222             }
223         }
224
225         jacobian := lm.NumJac{Func: f}
226
227         // Solve for fit
228         toBeSolved := lm.LMPProblem{
229             Dim: 3,
230             Size: len(s[set][0]),
231             Func: f,
232             Jac: jacobian.Jac,
233             InitParams: []float64{amp, wid, cen, c},
234             Tau: 1e-6,
235             Eps1: 1e-8,
236             Eps2: 1e-8,
237         }
238
239         results, _ := lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e
-16})
240
241         amp, wid, cen := results.X[0], math.Abs(results.X[1]), results.X[2]
242
243         sfwhm = append(sfwhm, wid*1000)
244
245         str := fmt.Sprintf("%d \t %.2f mW \t %.2f MHz \t %.6f uV \t %.4f GHz\n", set,
sPowers[set], wid*1000, amp, cen)
246         fmt.Printf(str)
247         logFile = append(logFile, str)
248
249         // Create Lorentzian fit data according to solved fit parameters
250         df := .001
251         f0 := s[set][0][0]
252         fitPts := int((s[set][0][len(s[set][0]) - 1] - f0)/df) + 1
253         sFits = append(sFits, generateFitData(amp, wid, cen, c, f0, df, fitPts))
254
255         // Width lines
256         sWidthLine = []float64{cen - wid/2, cen + wid/2, {amp/2, amp/2}}
257         sWidthLines = append(sWidthLines, sWidthLine)
258
259         if len(fitStokes) == len(fitAntiStokes) {
260             // For height ratio

```

```

261         ampRatios = append(ampRatios, amp/asAmps[i])
262     }
263
264     // For linewidth
265     sLinewidths = append(sLinewidths, sfwhm[i])
266 }
267 fmt.Printf("\n")
268 logFile = append(logFile, "\n")
269
270 goPlotsFits(
271     fitStokes, s, sFits, sWidthLines,  $\sigma$ s, sLabel, sfwhm, sNotes, temp, slide,
272     sample, logpath, coolingExperiment,
273 )
274
275 goPlotsPowerVsWid(
276     fitStokes, sLabel, sNotes, sfwhm, temp, slide, sample, logpath,
277     coolingExperiment,
278 )
279
280 eq := true
281 if len(fitAntiStokes) != len(fitStokes) {
282     eq = false
283 } else {
284     for i, v := range fitAntiStokes {
285         if v != fitStokes[i] {
286             eq = false
287             break
288         }
289     }
290 }
291 if eq {
292     var powers []float64
293     for i, v := range asPowers {
294         powers = append(powers, (v + sPowers[i])/2)
295     }
296     goPlotHeightRatios(
297         fitStokes, ampRatios, powers, sLabel, sample, logpath,
298         coolingExperiment, slide,
299     )
300
301      $\Gamma$ asEff,  $\Gamma$ sEff :=  $\Gamma$ eff(asPowers[len(asPowers)-1],  $\Gamma$ , length, gb, coolingExperiment)
302     goPlotLinewidths(
303         fitStokes,  $\Gamma$ asEff,  $\Gamma$ sEff, asLinewidths, sLinewidths, asPowers,
304         sPowers, sLabel, sample, logpath, coolingExperiment, slide,
305     )
306 } else {
307     str := fmt.Sprintf("Stokes & AntiStokes sets not equal\n" +
308         "(Height ratio and linewidth plots not produced)\n")
309     fmt.Printf(str)
310     logFile = append(logFile, str)
311 }
312 }
313 }
314 }
315
316 } else if cabs {
317
318     setsToPlotCABS := []int{0}
319
320     //setsToPlotCABS := rangeInt(0, 15)
321
322     normalized := []string{} // "Powers"
323     cabsData, sigUnit := getCABSDData(
324         setsToPlotCABS, lock, sigFilepath, freqFilepath, normalized,
325     )
326
327     sigmaMultiple := 1.
328     cabsData =  $\sigma$ CABS(

```

```

329     setsToPlotCABS, numAvgs, cabsData, sigUnit, sigmaMultiple, normalized,
330 )
331
332 if contains(normalized, "Powers") {
333     cabsData = normalizeByPowers(setsToPlotCABS, cabsData, pumpPowers, stokesPowers,
334     probePowers)
335     fmt.Println("*Data normalized by " + normalized[0] + "\n")
336     logFile = append(logFile, fmt.Sprintf("*Data normalized by %s*\n", normalized[0]))
337 }
338
339 // Fit data / Sinc
340 var initialParams []float64
341 switch sample {
342     case "CS2":
343         initialParams = []float64{25, 2.5, .08, 0} //amp, cen, wid, C
344     case "UHNA3":
345         initialParams = []float64{10, 9.14, .1, 0} //amp, cen, wid, C
346     default:
347         initialParams = []float64{1, 5, .1, 0}
348 }
349
350 optimizedParams := make([][]float64, len(cabsData))
351 phaseMatchPeaks := make([]float64, setsToPlotCABS[len(setsToPlotCABS)-1]+1)
352 pumpProbeSep := make([]float64, setsToPlotCABS[len(setsToPlotCABS)-1]+1)
353
354 for _, set := range setsToPlotCABS {
355     optimizedParams[set] = FitLorentzian(
356         // freq, sig,  $\sigma$ , guess
357         cabsData[set][0], cabsData[set][1], cabsData[set][2], initialParams,
358     )
359
360     phaseMatchPeaks[set] = optimizedParams[set][0]
361     probeValue, err := strconv.ParseFloat(probeLaser[set], 64)
362     if err != nil {
363         // handle error, maybe log it and/or return
364         log.Fatal("Failed to parse probeLaser:", err)
365     }
366
367     pumpValue, err := strconv.ParseFloat(pumpLaser[set], 64)
368     if err != nil {
369         // handle error, maybe log it and/or return
370         log.Fatal("Failed to parse pumpLaser:", err)
371     }
372
373     pumpProbeSep[set] = (probeValue - pumpValue) / .008
374 }
375
376 if sinc {
377     plotSinc(
378         setsToPlotCABS, [][]float64{pumpProbeSep, phaseMatchPeaks}, label,
379         sample, logpath, length, slide,
380     )
381 }
382
383 binCabsSets := []int{}
384 if len(binCabsSets) > 0 {
385     binMHz := 11.
386     logFile = logBinning(
387         logFile, binCabsSets, binMHz,
388     )
389     cabsData = binCabs(binCabsSets, cabsData, binMHz) // 3. combine above-calculated  $\sigma$  (
390     cabsData[set][2]) with binned  $\sigma$ . (only relevant if binned)
391 }
392
393 logFile = logPlots(

```

```

395     logFile, setsToPlotCABS, numAvgs, date, label, setNums, startTime, endTime,
396     pumpPowers, stokesPowers, probePowers, lockinRange, dwell, bandwidth,
397     dataRate, order, startFrequency, stopFrequency, step, pumpProbeSep,
398     pumpLaser, probeLaser, probeFilter, stokesFilter, notes, optimizedParams,
399 )
400 plotCABS(
401     setsToPlotCABS, cabsData, label, normalized, sample, sigUnit, logpath, length, slide,
402 )
403 }
404
405 writeLog(logpath, logFile)
406 }
407
408 //-----//
409
410 func flags() (
411     bool, bool, bool, bool, bool, string, string, string, float64, int,
412 ) {
413
414     var cabs, lock, temp, slide, sinc bool
415     var sample, coolingExperiment, note string
416     var length float64
417     var avg int
418
419     flag.BoolVar(&cabs, "cabs", false, "CABS data")
420     flag.BoolVar(&lock, "lockin", false, "lock-in data")
421     flag.BoolVar(&temp, "temp", false, "contains temperature data in notes column")
422     flag.BoolVar(&slide, "slide", false, "format figures for slide presentation")
423     flag.BoolVar(&sinc, "sinc", false, "plot sinc^2 function (phase-matching data)")
424     flag.StringVar(&sample, "sample", "", "sample: LCOF, UHNA3, CS2, Te, TeO2, glass slide")
425     flag.StringVar(&coolingExperiment, "cooling", "", "Cooling data: pump-probe or pump-only")
426     flag.StringVar(&note, "note", "", "note to append folder name")
427     flag.Float64Var(&length, "len", 0, "length of sample in meters")
428     flag.IntVar(&avg, "avg", 0, "number of CSV files to average")
429     flag.Parse()
430
431     if coolingExperiment != "" && cabs {
432         fmt.Println("flag.Parse(): data flagged as both cooling and CABS.")
433         os.Exit(1)
434     }
435
436     if sample == "LCOF" && length == 0 {
437         fmt.Println("Specify length of sample in meters with -len=")
438         os.Exit(1)
439     }
440
441     return cabs, lock, temp, slide, sinc, sample, coolingExperiment, note, length, avg
442 }
443
444 func logpath(
445     note string,
446 ) (
447     string,
448 ) {
449     return "plots/" + time.Now().Format("2006-Jan-02") + "/" + time.Now().Format("15:04:05") +
450         ": " + note
451 }
452
453 func readMeta(
454     cabs, lock, temp bool,
455     coolingExperiment string,
456 ) (
457     []string, []string, []string, []string, []string, []float64, []float64,
458     []float64, []float64, []float64, []string, []string, []string,
459     []float64, []float64, []float64, []float64, []float64, []float64, []float64,
460     []int, []float64, []float64, []string, []string, []string, []string, []string,
461 ) {

```

```

462 // Read
463 metaFile, err := os.Open("Data/meta.csv")
464 if err != nil {
465     fmt.Println(err)
466     os.Exit(1)
467 }
468
469 reader := csv.NewReader(metaFile)
470 meta, err := reader.ReadAll()
471 if err != nil {
472     fmt.Println(err)
473     os.Exit(1)
474 }
475
476 var date, label, set, startTime, endTime, filepath, sigFilepath, freqFilepath, notes []
477     string
478 var pumpLaser, probeLaser, probeFilter, stokesFilter []string
479 var asPowers, sPowers, asNotes, sNotes []float64
480 var pumpPowers, stokesPowers, probePowers []float64
481 var lockinRange, dwell, bandwidth, dataRate, order []float64
482 var startFrequency, stopFrequency, step []float64
483 var numAvgs []int
484 var dateCol, labelCol, setCol, startTimeCol, endTimeCol int
485 var pumpCol, stokesCol, probeCol, filepathCol int
486 var lockinRangeCol, dwellCol, bandwidthCol, dataRateCol, orderCol int
487 var startFrequencyCol, stopFrequencyCol, stepCol, numAvgsCol, notesCol int
488 var pumpLaserCol, probeLaserCol, probeFilterCol, stokesFilterCol int
489
490 for col, heading := range meta[0] {
491     switch heading {
492     case "Date":
493         dateCol = col
494     case "Label":
495         labelCol = col
496     case "Sets":
497         setCol = col
498     case "Start Time":
499         startTimeCol = col
500     case "End Time (hr:min:sec)":
501         endTimeCol = col
502     case "Pump":
503         pumpCol = col
504     case "Stokes":
505         stokesCol = col
506     case "Probe":
507         probeCol = col
508     case "Filepath":
509         filepathCol = col
510     case "Range":
511         lockinRangeCol = col
512     case "Dwell":
513         dwellCol = col
514     case "Bandwidth":
515         bandwidthCol = col
516     case "Data Rate":
517         dataRateCol = col
518     case "Order":
519         orderCol = col
520     case "Start Frequency":
521         startFrequencyCol = col
522     case "Stop Frequency":
523         stopFrequencyCol = col
524     case "Step":
525         stepCol = col
526     case "Pump Laser":
527         pumpLaserCol = col
528     case "Probe Laser":
529         probeLaserCol = col

```

```

529     case "Probe Filter":
530         probeFilterCol = col
531     case "Stokes Filter":
532         stokesFilterCol = col
533     case "Num Avgs":
534         numAvgsCol = col
535     case "Notes":
536         notesCol = col
537     }
538 }
539
540 for row, v := range meta {
541
542     if row > 0 {
543
544         date = append(date, v[dateCol])
545         label = append(label, v[labelCol])
546         set = append(set, v[setCol])
547         pumpLaser = append(pumpLaser, v[pumpLaserCol])
548         probeLaser = append(probeLaser, v[probeLaserCol])
549         stokesFilter = append(stokesFilter, v[stokesFilterCol])
550         probeFilter = append(probeFilter, v[probeFilterCol])
551         notes = append(notes, v[notesCol])
552
553         if numAvg, err := strconv.Atoi(v[numAvgsCol]); err == nil {
554             numAvgs = append(numAvgs, numAvg)
555         } else {
556             fmt.Println(err)
557             os.Exit(1)
558         }
559
560         if coolingExperiment != "" {
561             if strings.Contains(v[labelCol], "ras") {
562                 if v, err := strconv.ParseFloat(strings.Split(v[labelCol], " ")[0], 64); err ==
563                 nil {
564                     asPowers = append(asPowers, v)
565                 } else {
566                     fmt.Println(err)
567                     os.Exit(1)
568                 }
569             } else if strings.Contains(v[labelCol], "rs"){
570                 if v, err := strconv.ParseFloat(strings.Split(v[labelCol], " ")[0], 64); err ==
571                 nil {
572                     sPowers = append(sPowers, v)
573                 } else {
574                     fmt.Println(err)
575                     os.Exit(1)
576                 }
577             }
578
579             if lock {
580                 sigFilepath = append(sigFilepath, v[setCol] + "/signal.csv")
581                 freqFilepath = append(freqFilepath, v[setCol] + "/signal.csv")
582             } else {
583                 filepath = append(filepath, v[filepathCol])
584             }
585
586             if temp {
587                 if strings.Contains(v[labelCol], "as") {
588                     if asNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
589                         asNotes = append(asNotes, asNote)
590                     } else {
591                         fmt.Println(err)
592                         os.Exit(1)
593                     }
594                 } else {
595                     if sNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
596                         sNotes = append(sNotes, sNote)

```

```

595         } else {
596             fmt.Println(err)
597             os.Exit(1)
598         }
599     }
600 }
601 } else if cabs {
602
603     if v[pumpCol] == "" {
604         pumpPowers = append(pumpPowers, 0)
605     } else if v, err := strconv.ParseFloat(v[pumpCol], 64); err == nil {
606         pumpPowers = append(pumpPowers, v)
607     } else {
608         fmt.Println(err)
609         fmt.Println("readMeta pump string -> float error")
610         os.Exit(1)
611     }
612     if v[stokesCol] == "" {
613         stokesPowers = append(stokesPowers, 0)
614     } else if v, err := strconv.ParseFloat(v[stokesCol], 64); err == nil {
615         stokesPowers = append(stokesPowers, v)
616     } else {
617         fmt.Println(err)
618         fmt.Println("readMeta stokes string -> float error")
619         os.Exit(1)
620     }
621     if v[probeCol] == "" {
622         probePowers = append(probePowers, 0)
623     } else if v, err := strconv.ParseFloat(v[probeCol], 64); err == nil {
624         probePowers = append(probePowers, v)
625     } else {
626         fmt.Println(err)
627         fmt.Println("readMeta probe string -> float error")
628         os.Exit(1)
629     }
630
631     if lock {
632
633         startTime = append(startTime, v[startTimeCol])
634         endTime = append(endTime, v[endTimeCol])
635
636         sigFilepath = append(sigFilepath, v[setCol] + "/signal.csv")
637         freqFilepath = append(freqFilepath, v[setCol] + "/frequency.csv")
638
639         if v[lockinRangeCol] == "" {
640             lockinRange = append(lockinRange, 0)
641         } else if v, err := strconv.ParseFloat(v[lockinRangeCol], 64); err == nil {
642             lockinRange = append(lockinRange, v)
643         } else {
644             fmt.Println(err)
645             fmt.Println("readMeta lockinRange string -> float error")
646             os.Exit(1)
647         }
648         if v[dwellCol] == "" {
649             dwell = append(dwell, 0)
650         } else if v, err := strconv.ParseFloat(v[dwellCol], 64); err == nil {
651             dwell = append(dwell, v)
652         } else {
653             fmt.Println(err)
654             fmt.Println("readMeta dwell string -> float error")
655             os.Exit(1)
656         }
657         if v[bandwidthCol] == "" {
658             bandwidth = append(bandwidth, 0)
659         } else if v, err := strconv.ParseFloat(v[bandwidthCol], 64); err == nil {
660             bandwidth = append(bandwidth, v)
661         } else {
662             fmt.Println(err)

```

```

663         fmt.Println("readMeta bandwidth string -> float error")
664         os.Exit(1)
665     }
666     if v[dataRateCol] == "" {
667         dataRate = append(dataRate, 0)
668     } else if v, err := strconv.ParseFloat(v[dataRateCol], 64); err == nil {
669         dataRate = append(dataRate, v)
670     } else {
671         fmt.Println(err)
672         fmt.Println("readMeta dataRate string -> float error")
673         os.Exit(1)
674     }
675     if v[orderCol] == "" {
676         order = append(order, 0)
677     } else if v, err := strconv.ParseFloat(v[orderCol], 64); err == nil {
678         order = append(order, v)
679     } else {
680         fmt.Println(err)
681         fmt.Println("readMeta order string -> float error")
682         os.Exit(1)
683     }
684     if v[startFrequencyCol] == "" {
685         startFrequency = append(startFrequency, 0)
686     } else if v, err := strconv.ParseFloat(v[startFrequencyCol], 64); err == nil {
687         startFrequency = append(startFrequency, v)
688     } else {
689         fmt.Println(err)
690         fmt.Println("readMeta startFrequency string -> float error")
691         os.Exit(1)
692     }
693     if v[stopFrequencyCol] == "" {
694         stopFrequency = append(stopFrequency, 0)
695     } else if v, err := strconv.ParseFloat(v[stopFrequencyCol], 64); err == nil {
696         stopFrequency = append(stopFrequency, v)
697     } else {
698         fmt.Println(err)
699         fmt.Println("readMeta stopFrequency string -> float error")
700         os.Exit(1)
701     }
702     if v[stepCol] == "" {
703         step = append(step, 0)
704     } else if v, err := strconv.ParseFloat(v[stepCol], 64); err == nil {
705         step = append(step, v)
706     } else {
707         fmt.Println(err)
708         fmt.Println("readMeta step string -> float error")
709         os.Exit(1)
710     }
711 }
712 } else {
713     filepath = append(filepath, v[filepathCol])
714 }
715
716 if temp {
717     if strings.Contains(v[labelCol], "as") {
718         if asNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
719             asNotes = append(asNotes, asNote)
720         } else {
721             fmt.Println(err)
722             os.Exit(1)
723         }
724     } else {
725         if sNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
726             sNotes = append(sNotes, sNote)
727         } else {
728             fmt.Println(err)
729             os.Exit(1)
730         }
731     }

```



```

731     }
732   }
733 }
734 }
735 }
736
737 return date, label, set, startTime, endTime, asPowers, sPowers,
738 pumpPowers, stokesPowers, probePowers, filepath, sigFilepath, freqFilepath,
739 lockinRange, dwell, bandwidth, dataRate, order, startFrequency, stopFrequency,
740 step, numAves, asNotes, sNotes, pumpLaser, probeLaser, probeFilter, stokesFilter,
741 notes
742 }
743
744 func logHeader(
745     cabs, lock, temp, slide bool,
746     sample, coolingExperiment, note string,
747     length float64,
748 ) (
749     []string,
750 ) {
751
752     logFile := []string{}
753     if sample != "" {
754         logFile = append(logFile, "Sample: " + sample + "\n")
755     }
756     if note != "" {
757         logFile = append(logFile, "Runtime note: " + note + "\n")
758     }
759     if slide {
760         logFile = append(logFile, "Figures formatted for slide presentation\n")
761     }
762
763     fmt.Printf(logFile[0])
764
765     if coolingExperiment != "" {
766         logFile = append(logFile, "\n*Cooling Data: " + coolingExperiment + "\n")
767         fmt.Printf("\n*Cooling Data: " + coolingExperiment + "\n")
768     } else if cabs {
769         logFile = append(logFile, "\n*CABS Data*\n")
770         fmt.Printf("\n*CABS Data*\n")
771     }
772     if temp {
773         logFile = append(logFile, "\n*Temperature-dependent data*\n")
774         fmt.Printf("\n*Temperature-dependent data*\n")
775     }
776     if sample == "LCOF" {
777         str := fmt.Sprintf("\n*Liquid-core optical fiber sample*\n")
778         logFile = append(logFile, str)
779         fmt.Printf(str)
780     }
781     if lock {
782         str := fmt.Sprintf("\n*Data gathered from Lock-in*\n\n")
783         logFile = append(logFile, str)
784         fmt.Printf(str)
785     } else {
786         str := fmt.Sprintf("\n*Data gathered from Spectrum Analyzer*\n\n")
787         logFile = append(logFile, str)
788         fmt.Printf(str)
789     }
790
791     return logFile
792 }
793
794 func avgCSVs(
795     nAvg int,
796     powers []float64,
797 ) (
798     [][]float64, [][]float64,

```

```

799 ) {
800
801 // Peek at 0th file to get length
802 f, err := os.Open("Data/" + fmt.Sprintf(powers[0]) + "/rs0.csv")
803 if err != nil {
804     fmt.Println(err)
805     os.Exit(1)
806 }
807 peek, err := readCSV(f)
808 if err != nil {
809     fmt.Println(err)
810     os.Exit(1)
811 }
812
813 obs, ors, oras, obas, oas, os := make([][]float64, len(powers)),
814 make([][]float64, len(powers)), make([][]float64, len(powers)),
815 make([][]float64, len(powers)), make([][]float64, len(powers)),
816 make([][]float64, len(powers))
817 for i := range obs {
818     obs[i], ors[i], oras[i], obas[i], oas[i], os[i] = make([]float64, len(peek)-2),
819     make([]float64, len(peek)-2), make([]float64, len(peek)-2),
820     make([]float64, len(peek)-2), make([]float64, len(peek)-2),
821     make([]float64, len(peek)-2)
822 }
823
824 osString, oasString := make([][]string, len(powers)), make([][]string, len(powers))
825 for i := range osString {
826     osString[i], oasString[i] = make([]string, len(peek)-2), make([]string, len(peek)-2)
827 }
828
829 if nAvg > 0 {
830     for _, name := range []string{"bs", "rs", "ras", "bas"} {
831
832         for set, powFloat := range powers {
833
834             pow := fmt.Sprintf(powFloat)
835
836             var newDataCSV [][]string
837
838             // sigColsToAvg[nAvg][sig]
839             sigColsToAvg := make([][]float64, nAvg)
840             for k := range sigColsToAvg {
841                 sigColsToAvg[k] = make([]float64, len(peek)-1)
842             }
843
844             σCSV := make([][]string, 1)
845             σCSV[0] = make([]string, len(peek)-2)
846
847             for i := 0; i < nAvg; i++ {
848                 // Read
849                 f, err := os.Open("Data/" + pow + "/" + name + fmt.Sprintf(i) + ".csv")
850                 if err != nil {
851                     fmt.Println(err)
852                     os.Exit(1)
853                 }
854
855                 data, err := readCSV(f)
856                 newDataCSV = data
857
858                 for j := 1; j < len(data); j++ {
859
860                     s := strings.ReplaceAll(data[j][2], " ", "")
861
862                     sig, err := strconv.ParseFloat(s, 64)
863                     if err != nil {
864                         fmt.Println(err)
865                         os.Exit(1)
866                     }

```

```

867         sigColsToAvg[i][j-1] = sig
868     }
869 }
870
871
872 toAvg := make([]float64, nAvg)
873 averagedCol := make([]float64, len(sigColsToAvg[0]))
874  $\sigma$ Col := make([]float64, len(sigColsToAvg[0]))
875 for i := 0; i < len(sigColsToAvg[0]); i++ {
876     for j := 0; j < nAvg; j++ {
877         toAvg[j] = sigColsToAvg[j][i]
878     }
879     averagedCol[i] = avg(toAvg)
880      $\sigma$ Col[i] =  $\sigma$ (toAvg)
881 }
882
883 switch name {
884     case "bs":
885          $\sigma$ bs[set] =  $\sigma$ Col
886     case "rs":
887          $\sigma$ rs[set] =  $\sigma$ Col
888     case "ras":
889          $\sigma$ ras[set] =  $\sigma$ Col
890     case "bas":
891          $\sigma$ bas[set] =  $\sigma$ Col
892     default:
893         fmt.Println("error in switch statement with  $\sigma$  in avgCSVs\n\n")
894         os.Exit(1)
895 }
896
897 for i := 2; i < len(newDataCSV); i++ {
898     newDataCSV[i][2] = strconv.FormatFloat(averagedCol[i-2], 'f', -1, 64)
899      $\sigma$ CSV[0][i-2] = strconv.FormatFloat( $\sigma$ Col[i-2], 'f', -1, 64)
900 }
901
902 fData, err := os.Create("Data/" + pow + "/" + name + ".csv")
903 if err != nil {
904     fmt.Println(err)
905     os.Exit(1)
906 }
907
908 wData := csv.NewWriter(fData)
909 err = wData.WriteAll(newDataCSV)
910 if err != nil {
911     fmt.Println(err)
912     os.Exit(1)
913 }
914
915 f $\sigma$ , err := os.Create("Data/" + pow + "/" + name + " $\sigma$ .csv")
916 if err != nil {
917     fmt.Println(err)
918     os.Exit(1)
919 }
920
921 w $\sigma$  := csv.NewWriter(f $\sigma$ )
922 err = w $\sigma$ .WriteAll( $\sigma$ CSV)
923 if err != nil {
924     fmt.Println(err)
925     os.Exit(1)
926 }
927 }
928 }
929
930 for set := range powers {
931     for i := range  $\sigma$ s[0] {
932          $\sigma$ s[set][i] = math.Sqrt(math.Pow( $\sigma$ rs[set][i], 2) + math.Pow( $\sigma$ bs[set][i], 2))
933          $\sigma$ as[set][i] = math.Sqrt(math.Pow( $\sigma$ ras[set][i], 2) + math.Pow( $\sigma$ bas[set][i], 2))
934     }
935 }

```

```

935     osString[set][i] = fmt.Sprintf(os[set][i])
936     osasString[set][i] = fmt.Sprintf(osas[set][i])
937 }
938 }
939
940 fOs, err := os.Create("Data/os.csv")
941 if err != nil {
942     fmt.Println(err)
943     os.Exit(1)
944 }
945
946 wOs := csv.NewWriter(fOs)
947 err = wOs.WriteAll(osString)
948 if err != nil {
949     fmt.Println(err)
950     os.Exit(1)
951 }
952
953 fOas, err := os.Create("Data/oas.csv")
954 if err != nil {
955     fmt.Println(err)
956     os.Exit(1)
957 }
958
959 wOas := csv.NewWriter(fOas)
960 err = wOas.WriteAll(osasString)
961 if err != nil {
962     fmt.Println(err)
963     os.Exit(1)
964 }
965
966 } else {
967
968     fOs, err := os.Open("Data/os.csv")
969     if err != nil {
970         fmt.Println(err)
971         os.Exit(1)
972     }
973
974     osReader := csv.NewReader(fOs)
975     osString, err := osReader.ReadAll()
976     if err != nil {
977         fmt.Println(err)
978         os.Exit(1)
979     }
980
981     fOas, err := os.Open("Data/oas.csv")
982     if err != nil {
983         fmt.Println(err)
984         os.Exit(1)
985     }
986
987     oasReader := csv.NewReader(fOas)
988     osasString, err := oasReader.ReadAll()
989     if err != nil {
990         fmt.Println(err)
991         os.Exit(1)
992     }
993
994     for pow := range osString {
995         for i := range osString[pow] {
996             os[pow][i], err = strconv.ParseFloat(osString[pow][i], 64)
997             if err != nil {
998                 fmt.Println(err)
999                 os.Exit(1)
1000             }
1001
1002             osas[pow][i], err = strconv.ParseFloat(osasString[pow][i], 64)

```

```

1003         if err != nil {
1004             fmt.Println(err)
1005             os.Exit(1)
1006         }
1007     }
1008 }
1009
1010 }
1011
1012 return osas, os
1013 }
1014
1015 func rangeInt(
1016     start, end int,
1017 ) (
1018     []int,
1019 ) {
1020     nums := make([]int, end-start)
1021     for i := range nums {
1022         nums[i] = start + i
1023     }
1024     return nums
1025 }
1026
1027 func getCoolingData(
1028     lock bool,
1029     fileNames, labels []string,
1030 ) (
1031     [][][]float64, [][][]float64, [][][]float64, [][][]float64,
1032 ) {
1033
1034     var bas, bs, ras, rs [][][]float64
1035     sig := false
1036
1037     if lock {
1038
1039         // Assign data by name
1040         for i := 0; i < len(fileNames)/2; i++ {
1041
1042             if i == 0 || i%2 != 0 {
1043                 sig = true
1044             }
1045
1046             if strings.Contains(labels[i], "bas") {
1047                 bas = append(bas, getData(lock, sig, fileNames[i]))
1048             } else if strings.Contains(labels[i], "bs") {
1049                 bs = append(bs, getData(lock, sig, fileNames[i]))
1050             } else if strings.Contains(labels[i], "ras") {
1051                 ras = append(ras, getData(lock, sig, fileNames[i]))
1052             } else if strings.Contains(labels[i], "rs") {
1053                 rs = append(rs, getData(lock, sig, fileNames[i]))
1054             }
1055         }
1056     } else {
1057
1058         // Assign data by name
1059         for i, fileName := range fileNames {
1060
1061             if i == 0 || i%2 != 0 {
1062                 sig = true
1063             }
1064
1065             if strings.Contains(labels[i], "bas") {
1066                 bas = append(bas, getData(lock, sig, fileName))
1067             } else if strings.Contains(labels[i], "bs") {
1068                 bs = append(bs, getData(lock, sig, fileName))
1069             } else if strings.Contains(labels[i], "ras") {
1070                 ras = append(ras, getData(lock, sig, fileName))

```

```

1071     } else if strings.Contains(labels[i], "rs") {
1072         rs = append(rs, getData(lock, sig, fileName))
1073     }
1074 }
1075 }
1076
1077 return ras, bas, rs, bs
1078 }
1079
1080 func getCABSDData(
1081     sets []int,
1082     lock bool,
1083     sigFileNames, freqFileNames, normalized []string,
1084 ) (
1085     [][][]float64, string,
1086 ) {
1087
1088     // final form: cabsData[set][0: freq, 1: sig, 2:  $\sigma$ ][rows of freq/sig/ $\sigma$ ]
1089     var cabsData [][][]float64
1090     var sigUnit string
1091
1092     if lock {
1093
1094         var cabsDataPreUnit [][][]float64
1095         for i, v := range sigFileNames {
1096             lockData := getLockData(v, freqFileNames[i])
1097             cabsDataPreUnit = append(cabsDataPreUnit, lockData)
1098         }
1099
1100         // if not going to be normalized
1101         if !contains(normalized, "Powers") {
1102
1103             // Check for most appropriate signal unit (preference of larger)
1104             largestSig := 0.
1105             for set := range cabsDataPreUnit {
1106                 for _, s := range sets {
1107                     if set == s {
1108                         for _, v := range cabsDataPreUnit[set][1] {
1109                             if v > largestSig {
1110                                 largestSig = v
1111                             }
1112                         }
1113                     }
1114                 }
1115             }
1116
1117             if largestSig > 1e-3 {
1118                 sigUnit = "mV"
1119                 for set := range cabsDataPreUnit {
1120                     for i, v := range cabsDataPreUnit[set][1] {
1121                         cabsDataPreUnit[set][1][i] = v*1e3
1122                     }
1123                     cabsData = append(cabsData, cabsDataPreUnit[set])
1124                 }
1125             } else if largestSig > 1e-6 {
1126                 sigUnit = "μV"
1127                 for set := range cabsDataPreUnit {
1128                     for i, v := range cabsDataPreUnit[set][1] {
1129                         cabsDataPreUnit[set][1][i] = v*1e6
1130                     }
1131                     cabsData = append(cabsData, cabsDataPreUnit[set])
1132                 }
1133             } else if largestSig > 1e-9 {
1134                 sigUnit = "nV"
1135                 for set := range cabsDataPreUnit {
1136                     for i, v := range cabsDataPreUnit[set][1] {
1137                         cabsDataPreUnit[set][1][i] = v*1e9
1138                     }

```

```

1139         cabsData = append(cabsData, cabsDataPreUnit[set])
1140     }
1141 } else if largestSig > 1e-12 {
1142     sigUnit = "pV"
1143     for set := range cabsDataPreUnit {
1144         for i, v := range cabsDataPreUnit[set][1] {
1145             cabsDataPreUnit[set][1][i] = v*1e12
1146         }
1147         cabsData = append(cabsData, cabsDataPreUnit[set])
1148     }
1149 }
1150 } else {
1151     sigUnit = ""
1152     for set := range cabsDataPreUnit {
1153         cabsData = append(cabsData, cabsDataPreUnit[set])
1154     }
1155 }
1156 }
1157
1158 return cabsData, sigUnit
1159 }
1160
1161 func getData(
1162     lock, sig bool,
1163     csvName string,
1164 ) (
1165     [][]float64,
1166 ) {
1167
1168     // Read
1169     f, err := os.Open("Data/" + csvName)
1170     if err != nil {
1171         fmt.Println(err)
1172         os.Exit(1)
1173     }
1174     defer f.Close()
1175     dataStr, err := readCSV(f)
1176     if err != nil {
1177         fmt.Println(err)
1178         os.Exit(1)
1179     }
1180
1181     // Separate, Strip, & Transpose
1182     var frequencyStrT, signalStrT []string
1183
1184     if lock {
1185         if sig {
1186             for i := range dataStr {
1187                 signalStrT = append(signalStrT, dataStr[i][0])
1188             }
1189         } else {
1190             for i := range dataStr {
1191                 frequencyStrT = append(frequencyStrT, dataStr[i][0])
1192             }
1193         }
1194     } else {
1195         for i := 1; i < len(dataStr); i++ {
1196             frequencyStrT = append(frequencyStrT, strings.ReplaceAll(dataStr[i][0], " ", ""))
1197             signalStrT = append(signalStrT, strings.ReplaceAll(dataStr[i][2], " ", ""))
1198         }
1199     }
1200
1201     // Convert to float
1202     var frequency, signal []float64
1203
1204     for _, freqElem := range frequencyStrT {
1205         if freqValue, err := strconv.ParseFloat(freqElem, 64); err != nil {
1206             fmt.Println(err)

```

```

1207     os.Exit(1)
1208 } else {
1209     frequency = append(frequency, freqValue/1e9)
1210 }
1211 }
1212
1213 for _, sigElem := range signalStrT {
1214     if sigValue, err := strconv.ParseFloat(sigElem, 64); err != nil {
1215         fmt.Println(err)
1216         os.Exit(1)
1217     } else {
1218         signal = append(signal, sigValue)
1219     }
1220 }
1221
1222 if !lock {
1223     // Convert to Linear if dBm
1224     if dataStr[1][3] == " dBm" {
1225
1226         // uV
1227         var uV []float64
1228         for _, dBm := range signal {
1229             uV = append(uV, math.Pow(10, 6)*math.Pow(10, dBm/10.))
1230         }
1231         return [][]float64{frequency, uV}
1232
1233         /* nV
1234         var nV []float64
1235         for _, dBm := range signal {
1236             nV = append(nV, 1000*math.Pow(10, 6)*math.Pow(10, dBm/10.))
1237         }
1238         return [][]float64{frequency, nV}
1239         */
1240
1241     } else if dataStr[1][3] == " uV" {
1242         var nV []float64
1243
1244         for _, uV := range signal {
1245             nV = append(nV, 1000*uV)
1246         }
1247
1248         /* Convert to picovolts
1249         var pV []float64
1250         for _, uV := range signal {
1251             pV = append(pV, 1000*uV)
1252         }*/
1253
1254         return [][]float64{frequency, nV}
1255     }
1256
1257     fmt.Println("Warning: check units - not uV or dBm")
1258     return [][]float64{frequency, signal}
1259 } else {
1260     return [][]float64{frequency, signal}
1261 }
1262 }
1263
1264 func getLockData(
1265     sigCSVName, freqCSVName string,
1266 ) (
1267     [][]float64,
1268 ) {
1269
1270     // Read signal data
1271     sigf, err := os.Open("Data/" + sigCSVName)
1272     if err != nil {
1273         fmt.Println(err)
1274         os.Exit(1)

```



```

1275 }
1276 sigDataStr, err := readCSV(sigf)
1277 if err != nil {
1278     fmt.Println(err)
1279     sigf.Close()
1280     os.Exit(1)
1281 }
1282
1283 // Explicitly close the file (avoids too many files open error)
1284 sigf.Close()
1285
1286 // Read frequency data
1287 freqf, err := os.Open("Data/" + freqCSVName)
1288 if err != nil {
1289     fmt.Println(err)
1290     os.Exit(1)
1291 }
1292 freqDataStr, err := readCSV(freqf)
1293 if err != nil {
1294     fmt.Println(err)
1295     freqf.Close()
1296     os.Exit(1)
1297 }
1298
1299 // Explicitly close the file (avoids too many files open error)
1300 freqf.Close()
1301
1302 // Transpose
1303 var freqStrT, sigStrT []string
1304
1305 for i := range sigDataStr {
1306     sigStrT = append(sigStrT, sigDataStr[i][0])
1307     freqStrT = append(freqStrT, freqDataStr[i][0])
1308 }
1309
1310 // Convert to float
1311 var frequency, signal []float64
1312
1313 for _, freqElem := range freqStrT {
1314     if freqValue, err := strconv.ParseFloat(freqElem, 64); err != nil {
1315         fmt.Println(err)
1316         os.Exit(1)
1317     } else {
1318         frequency = append(frequency, freqValue/1e9)
1319     }
1320 }
1321
1322 for _, sigElem := range sigStrT {
1323     if sigValue, err := strconv.ParseFloat(sigElem, 64); err != nil {
1324         fmt.Println(err)
1325         os.Exit(1)
1326     } else {
1327         signal = append(signal, sigValue)
1328     }
1329 }
1330
1331 /* Convert to pV
1332 maxSig := 0.
1333 sigUnit := "pV"
1334 for i, v := range signal {
1335     signal[i] = v*1e9
1336     if signal[i] > maxSig {
1337         maxSig = signal[i]
1338     }
1339 }
1340
1341 if maxSig > 1000. {
1342     // Convert to uV

```

```

1343     for i, v := range signal {
1344         signal[i] = v*1e-3
1345     }
1346     sigUnit = "uV"
1347 }*/
1348
1349
1350 return [][]float64{frequency, signal}
1351 }
1352
1353 func readCSV(
1354     rs io.ReadSeeker,
1355 ) (
1356     [][]string, error,
1357 ) {
1358     // Skip first row (line)
1359     row1, err := bufio.NewReader(rs).ReadSlice('\n')
1360     if err != nil {
1361         return nil, err
1362     }
1363     _, err = rs.Seek(int64(len(row1)), io.SeekStart)
1364     if err != nil {
1365         return nil, err
1366     }
1367
1368     // Read remaining rows
1369     r := csv.NewReader(rs)
1370     rows, err := r.ReadAll()
1371     if err != nil {
1372         return nil, err
1373     }
1374     return rows, nil
1375 }
1376
1377 func logBinning(
1378     logFile []string,
1379     binCabsSets []int,
1380     binMHz float64,
1381 ) (
1382     []string,
1383 ) {
1384
1385     for _, set := range binCabsSets {
1386         logFile = append(logFile, fmt.Sprintf("Run %d binned to %.3f MHz\n", set+1, binMHz))
1387     }
1388
1389     return logFile
1390 }
1391
1392 func logPlots(
1393     logFile []string,
1394     setsToPlotCABS, numAves []int,
1395     date, label, run, startTime, endTime []string,
1396     pumpPowers, stokesPowers, probePowers, lockinRange, dwell []float64,
1397     bandwidth, dataRate, order, startFrequency, stopFrequency, step, pumpProbeSep []float64,
1398     pumpLaser, probeLaser, probeFilter, stokesFilter, notes []string,
1399     optimizedParams [][]float64,
1400 ) (
1401     []string,
1402 ) {
1403
1404     for _, set := range setsToPlotCABS {
1405         logFile = append(logFile, fmt.Sprintf("\nRun %s\n", run[set]))
1406         logFile = append(logFile, fmt.Sprintf("\tData taken: %s\n", date[set]))
1407         logFile = append(logFile, fmt.Sprintf("\tLabel: %s\n", label[set]))
1408         logFile = append(logFile, fmt.Sprintf("\tStart Time (h:m:s): %s\n", startTime[set]))
1409         logFile = append(logFile, fmt.Sprintf("\tEnd Time(h:m:s): %s\n", endTime[set]))
1410         logFile = append(logFile, fmt.Sprintf("\tPump Laser: %s nm\n", pumpLaser[set]))

```

```

1411     logFile = append(logFile, fmt.Sprintf("\tProbe Laser: %s nm\n", probeLaser[set]))
1412     logFile = append(logFile, fmt.Sprintf("\tPump-Probe Separation: %.2f GHz\n",
1413         pumpProbeSep[set]))
1413     logFile = append(logFile, fmt.Sprintf("\tStokes Filter: %s nm\n", stokesFilter[set]))
1414     logFile = append(logFile, fmt.Sprintf("\tProbe Filter: %s nm\n", probeFilter[set]))
1415     logFile = append(logFile, fmt.Sprintf("\tPump Power: %.3f mW\n", pumpPowers[set]))
1416     logFile = append(logFile, fmt.Sprintf("\tStokes Power: %.3f mW\n", stokesPowers[set]))
1417     logFile = append(logFile, fmt.Sprintf("\tProbe Power: %.3f mW\n", probePowers[set]))
1418     logFile = append(logFile, fmt.Sprintf("\tLock-in Range: %.2f\n", lockinRange[set]))
1419     logFile = append(logFile, fmt.Sprintf("\tDwell Time: %.9f s\n", dwell[set]))
1420     logFile = append(logFile, fmt.Sprintf("\tLock-in Bandwidth: %.2f Hz\n", bandwidth[set]))
1421     logFile = append(logFile, fmt.Sprintf("\tLock-in Bandwidth Order: %.0f\n", order[set]))
1422     logFile = append(logFile, fmt.Sprintf("\tData Sampling Rate: %.2f Sa/s\n", dataRate[set
1423 ]))
1423     logFile = append(logFile, fmt.Sprintf("\tStart Frequency: %.2f Hz\n", startFrequency[set
1424 ]))
1424     logFile = append(logFile, fmt.Sprintf("\tStop Frequency: %.2f Hz\n", stopFrequency[set]
1425 ))
1425     logFile = append(logFile, fmt.Sprintf("\tStep Size: %.2f Hz\n", step[set]))
1426     logFile = append(logFile, fmt.Sprintf("\tNumber of Averages: %d\n", numAvgs[set]))
1427     logFile = append(logFile, fmt.Sprintf("\tData Collection Note: %s\n\n", notes[set]))
1428     logFile = append(logFile, fmt.Sprintf("\tFit Parameters:\n"))
1429     logFile = append(logFile, fmt.Sprintf("\t\tAmp: %v\n", optimizedParams[set][0]))
1430     logFile = append(logFile, fmt.Sprintf("\t\tCen: %v\n", optimizedParams[set][1]))
1431     logFile = append(logFile, fmt.Sprintf("\t\tWid: %v\n", optimizedParams[set][2]))
1432     logFile = append(logFile, fmt.Sprintf("\t\tC: %v\n", optimizedParams[set][3]))
1433 }
1434
1435 return logFile
1436 }
1437
1438 func getAllLabels(
1439     label []string,
1440 ) (
1441     []string, []string, []string, []string,
1442 ) {
1443
1444     var rasLabel, basLabel, rsLabel, bsLabel []string
1445
1446     // Assign labels by verifying label
1447     for _, thisLabel := range label {
1448         if strings.Contains(thisLabel, "ras") {
1449             rasLabel = append(rasLabel, thisLabel)
1450         } else if strings.Contains(thisLabel, "bas") {
1451             basLabel = append(basLabel, thisLabel)
1452         } else if strings.Contains(thisLabel, "rs") {
1453             rsLabel = append(rsLabel, thisLabel)
1454         } else if strings.Contains(thisLabel, "bs") {
1455             bsLabel = append(bsLabel, thisLabel)
1456         }
1457     }
1458
1459     return rasLabel, basLabel, rsLabel, bsLabel
1460 }
1461
1462 func buildData(
1463     data [][]float64,
1464 ) (
1465     plotter.XYs,
1466 ) {
1467
1468     xy := make(plotter.XYs, len(data[0]))
1469
1470     for i := range xy {
1471         xy[i].X = data[0][i]
1472         xy[i].Y = data[1][i]
1473     }
1474

```

```

1475     return xy
1476 }
1477
1478 func buildErrors(
1479      $\sigma$  []float64,
1480 ) (
1481     plotter.Errors,
1482 ) {
1483
1484     error := make(plotter.Errors, len( $\sigma$ ))
1485
1486     for i := range error {
1487         error[i].Low, error[i].High =  $\sigma$ [i],  $\sigma$ [i]
1488     }
1489
1490     return error
1491 }
1492
1493 func plotRaw(
1494     sets []int,
1495     bas, ras, bs, rs [][]float64,
1496     basLabel, rasLabel, bsLabel, rsLabel []string,
1497 ) {
1498
1499     for i := range sets {
1500         dimensions := 2
1501         persist := true
1502         debug := false
1503         plot, _ := glot.NewPlot(dimensions, persist, debug)
1504
1505         plot.SetTitle("Raw")
1506         plot.SetXLabel("Frequency (GHz)")
1507         plot.SetYLabel("Signal (uV)")
1508
1509         plot.AddPointGroup(rasLabel[sets[i]], "points", ras[sets[i]])
1510         plot.AddPointGroup(basLabel[sets[i]], "points", bas[sets[i]])
1511         //plot.AddPointGroup(rsLabel[sets[i]], "points", rs[sets[i]])
1512         //plot.AddPointGroup(bsLabel[sets[i]], "points", bs[sets[i]])
1513     }
1514 }
1515
1516 func plotCABS(
1517     sets []int,
1518     cabsData [][]float64,
1519     label, normalized []string,
1520     sample, sigUnit, logpath string,
1521     length float64,
1522     slide bool,
1523 ) {
1524
1525     var l string
1526
1527     switch length {
1528     case 0.0:
1529         l = ""
1530     case 0.001:
1531         l = "1 mm"
1532     case 0.01:
1533         l = "1 cm"
1534     case 0.004:
1535         l = "4 mm"
1536     case 0.0000005:
1537         l = "500 nm"
1538     case 0.0001:
1539         l = "100  $\mu$ m"
1540     case 0.00001:
1541         l = "10  $\mu$ m"
1542     default:

```

```

1543     l = strconv.FormatFloat(length, 'f', 1, 64)
1544 }
1545
1546 type errorPoints struct {
1547     plotter.XYs
1548     plotter.YErrors
1549 }
1550
1551 title := l + " " + sample + " CABS"
1552 xlabel := "Frequency (GHz)"
1553 var ylabel string
1554 if contains(normalized, "Powers") {
1555     ylabel = "Normalized by Powers (V/W3)"
1556 } else {
1557     ylabel = "Spectral Density (" + sigUnit + ")"
1558 }
1559 legend := ""
1560
1561 /* Manual Axes
1562 xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes("CABS", sample, "")
1563 if err != nil {
1564     fmt.Println(err)
1565     os.Exit(1)
1566 }*/
1567
1568 // Auto Axes
1569 xmax := 0.
1570 xmin := cabsData[0][0][0]
1571 for _, set := range sets {
1572     if cabsData[set][0][0] < xmin {
1573         xmin = cabsData[set][0][0]
1574     }
1575     if cabsData[set][0][len(cabsData[set][0])-1] > xmax {
1576         xmax = cabsData[set][0][len(cabsData[set][0])-1]
1577     }
1578 }
1579
1580 xrange := []float64{xmin, xmax}
1581 xtick := 0.
1582 displayDigits := 2
1583 switch {
1584 case (xmax - xmin)/8 > 0.25:
1585     xtick = 0.5
1586     displayDigits = 1
1587 case (xmax - xmin)/8 > 0.1:
1588     xtick = 0.25
1589 case (xmax - xmin)/8 > 0.075:
1590     xtick = 0.075
1591 case (xmax - xmin)/8 > 0.05:
1592     xtick = 0.05
1593 case (xmax - xmin)/8 > 0.025:
1594     xtick = 0.025
1595 case (xmax - xmin)/8 > 0.01:
1596     xtick = 0.02
1597 case (xmax - xmin)/8 > 0.0075:
1598     xtick = 0.0075
1599 case (xmax - xmin)/8 > 0.005:
1600     xtick = 0.005
1601 case (xmax - xmin)/8 > 0.0025:
1602     xtick = 0.0025
1603 case (xmax - xmin)/8 > 0.001:
1604     xtick = 0.001
1605 }
1606
1607 firstTick := 0.
1608 for m := float64(int(xmin)); m <= xmin; m += xtick {
1609     firstTick = m
1610 }

```

```

1611 //fmt.Printf(strconv.FormatFloat(firstTick, 'f', 2, 64))
1612 //fmt.Printf(strconv.FormatFloat(xtick, 'f', 2, 64))
1613 xticks := []float64{}
1614 xtickLabels := []string{}
1615 for i := 0.; firstTick + xtick*i <= xmax - xtick/2; i++ {
1616     xticks = append(xticks, firstTick + xtick*i)
1617     if int(i)%2 != 0 {
1618         xtickLabels = append(xtickLabels, strconv.FormatFloat(firstTick + xtick*i, 'f',
1619             displayDigits, 64))
1620     } else {
1621         xtickLabels = append(xtickLabels, "")
1622     }
1623 }
1624
1625 ymax := 0.
1626 ymin := 10000.
1627 for _, set := range sets {
1628     for i, v := range cabsData[set][1] {
1629         if len(cabsData[set]) > 2 && v + cabsData[set][2][i]/2 > ymax {
1630             ymax = v + cabsData[set][2][i]/2
1631         } else if len(cabsData[set]) < 3 && v > ymax {
1632             ymax = v
1633         }
1634         if len(cabsData[set]) > 2 && v - cabsData[set][2][i]/2 < ymin {
1635             ymin = v - cabsData[set][2][i]/2
1636         } else if len(cabsData[set]) < 3 && v < ymin {
1637             ymin = v
1638         }
1639     }
1640     ymax += (ymax - ymin)/4 + (ymax - ymin)*float64(len(sets))/1000 //16
1641     ymin -= (ymax - ymin)/32
1642     yrange := []float64{ymin, ymax}
1643     ytick := ((ymax - ymin)/8)
1644     yticks := []float64{}
1645     ytickLabels := []string{}
1646     for i := 0.; i < 11; i++ {
1647         yticks = append(yticks, ytick*i + ymin)
1648         if int(i)%2 != 0 {
1649             ytickLabels = append(ytickLabels, strconv.FormatFloat(ytick*i + ymin, 'f', 2, 64))
1650         } else {
1651             ytickLabels = append(ytickLabels, "")
1652         }
1653     }
1654     yticks = append(yticks, ymax)
1655     ytickLabels = append(ytickLabels, "")
1656
1657     p, t, r := prepPlot(
1658         title, xlabel, ylabel, legend,
1659         xrange, yrange, xticks, yticks,
1660         xtickLabels, ytickLabels,
1661         slide,
1662     )
1663
1664     for _, set := range sets {
1665         pts := buildData(cabsData[set])
1666
1667         if len(cabsData[set]) > 2 {
1668             σErr := buildErrors(cabsData[set][2])
1669
1670             setPoints := errorPoints {
1671                 XYs: pts,
1672                 YErrors: plotter.YErrors(σErr),
1673             }
1674
1675             plotSet, err := plotter.NewScatter(setPoints)
1676             if err != nil {
1677

```

```

1678         fmt.Println(err)
1679         os.Exit(1)
1680     }
1681
1682     // Error bars
1683     e, err := plotter.NewErrorBars(setPoints)
1684     if err != nil {
1685         fmt.Println(err)
1686         os.Exit(1)
1687     }
1688     e.LineStyle.Color = palette(set, false, "")
1689
1690     plotSet.GlyphStyle.Color = palette(set, false, "")
1691     plotSet.GlyphStyle.Radius = vg.Points(5) //3
1692     plotSet.Shape = draw.CircleGlyph{}
1693
1694     p.Add(e, plotSet, t, r)
1695
1696 } else {
1697
1698     plotSet, err := plotter.NewScatter(pts)
1699     if err != nil {
1700         fmt.Println(err)
1701         os.Exit(1)
1702     }
1703
1704     plotSet.GlyphStyle.Color = palette(set, false, "")
1705     plotSet.GlyphStyle.Radius = vg.Points(5) //3
1706     plotSet.Shape = draw.CircleGlyph{}
1707
1708     p.Add(plotSet, t, r)
1709 }
1710
1711 // Legend
1712 l, err := plotter.NewScatter(pts)
1713 if err != nil {
1714     fmt.Println(err)
1715     os.Exit(1)
1716 }
1717
1718 l.GlyphStyle.Color = palette(set, false, "")
1719 l.GlyphStyle.Radius = vg.Points(8) //6
1720 l.Shape = draw.CircleGlyph{}
1721 p.Legend.Add(label[set], l)
1722 }
1723
1724 /* Guide Line
1725 guideLine := make(plotter.XYs, 2)
1726
1727 guideLine[0].X = 12.1
1728 guideLine[0].Y = yrange[0]
1729 guideLine[1].X = 12.1
1730 guideLine[1].Y = yrange[1]
1731
1732 plotGuideLine, err := plotter.NewLine(guideLine)
1733 if err != nil {
1734     fmt.Println(err)
1735     os.Exit(1)
1736 }
1737
1738 plotGuideLine.Color = color.RGBA{R: 128, G: 128, B: 128, A: 255}
1739
1740 p.Add(plotGuideLine)
1741 p.Legend.Add("12.1 GHz", plotGuideLine)
1742 */
1743
1744 savePlot(p, "CABS", logpath)
1745 }

```

```

1746
1747 func axes(
1748     plot, sample, coolingExperiment string,
1749 ) (
1750     []float64, []float64, []float64, []float64, []string, []string, error,
1751 ) {
1752
1753     switch plot {
1754     case "CABS":
1755         switch sample {
1756         case "UHNA3":
1757             xrange := []float64{9, 9.3}
1758             yrange := []float64{-10, 150}
1759             xtick := []float64{9, 9.05, 9.1, 9.15, 9.2, 9.25, 9.3}
1760             ytick := []float64{0, 50, 100, 150}
1761             xtickLabel := []string{"9", "", "9.1", "", "9.2", "", "9.3"}
1762             ytickLabel := []string{"0", "", "100", ""}
1763
1764             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1765         case "CS2":
1766             xrange := []float64{2.3, 2.8}
1767             yrange := []float64{0, 60}
1768             xtick := []float64{2.3, 2.35, 2.4, 2.45, 2.5, 2.55, 2.6, 2.65, 2.7, 2.75, 2.8}
1769             ytick := []float64{0, 10, 20, 30, 40, 50, 60}
1770             xtickLabel := []string{"2.3", "", "2.4", "", "2.5", "", "2.6", "", "2.7", "", "2.8"}
1771             ytickLabel := []string{"0", "", "20", "", "40", "", "60", ""}
1772
1773             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1774         case "glass slide":
1775             xrange := []float64{10.5, 11.5}
1776             yrange := []float64{0, 15}
1777             xtick := []float64{10.5, 10.6, 10.7, 10.8, 10.9, 11, 11.1, 11.2, 11.3, 11.4, 11.5}
1778             ytick := []float64{0, 3, 6, 9, 12, 15}
1779             xtickLabel := []string{"10.5", "", "10.7", "", "10.9", "", "11.1", "", "11.3", "", "11.5"}
1780             ytickLabel := []string{"0", "3", "6", "9", "12", "15"}
1781
1782             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1783         case "Tapered Fiber":
1784             xrange := []float64{8.78, 9.48}
1785             yrange := []float64{0, 150}
1786             xtick := []float64{8.78, 8.88, 8.98, 9.08, 9.18, 9.28, 9.38, 9.48}
1787             ytick := []float64{0, 25, 50, 75, 100, 125, 150}
1788             xtickLabel := []string{"8.78", "", "8.98", "", "9.18", "", "9.38", ""}
1789             ytickLabel := []string{"0", "25", "50", "75", "100", "125", "150"}
1790
1791             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1792         case "Te":
1793             xrange := []float64{3.2, 5}
1794             yrange := []float64{0, 24}
1795             xtick := []float64{3.2, 3.4, 3.6, 3.8, 4, 4.2, 4.4, 4.6, 4.8, 5}
1796             ytick := []float64{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24}
1797             xtickLabel := []string{"3.2", "", "3.6", "", "4", "", "4.4", "", "4.8", ""}
1798             ytickLabel := []string{"", "2", "", "6", "", "10", "", "14", "", "18", "", "22", ""}
1799
1800             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1801         case "Te02":
1802             xrange := []float64{11.9, 12.4}
1803             yrange := []float64{0, 2.5}
1804             xtick := []float64{11.9, 12.0, 12.1, 12.2, 12.3, 12.4}
1805             ytick := []float64{0, .25, .5, .75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5}
1806             xtickLabel := []string{"", "", "", "", "", "", ""}
1807             ytickLabel := []string{"", "0.25", "", "0.75", "", "1.25", "", "1.75", "", "2.25", ""}
1808
1809             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1810         case "No Sample":
1811             xrange := []float64{11.5, 12.5}
1812             yrange := []float64{0, 2100}

```



```

1813     xtick := []float64{11.5, 11.6, 11.7, 11.8, 11.9, 12.0, 12.1, 12.2, 12.3, 12.4, 12.5}
1814     ytick := []float64{0, 300, 600, 900, 1200, 1500, 1800, 2100}
1815     xtickLabel := []string{"11.5", "", "11.7", "", "11.9", "", "12.1", "", "12.3", "", "
12.5"}
1816     ytickLabel := []string{"", "300", "", "900", "", "1500", "", "2100"}
1817
1818     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1819 case "Sapphire":
1820     xrange := []float64{.100, .140}
1821     yrange := []float64{-50, 50}
1822     xtick := []float64{.100, .110, .120, .130, .140}
1823     ytick := []float64{-50, -25, 0, 25, 50}
1824     xtickLabel := []string{".100", "", ".120", "", ".140"}
1825     ytickLabel := []string{"-50", "", "0", "", "50"}
1826
1827     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1828 }
1829 case "fits":
1830     switch sample {
1831     case "LCOF":
1832         if coolingExperiment == "pump-only" {
1833             xrange := []float64{2.0, 2.5}
1834             yrange := []float64{0, 110}
1835             xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
1836             ytick := []float64{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140,
1837 150}
1838             xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
1839             ytickLabel := []string{"0", "", "20", "", "40", "", "60", "", "80", "", "100", ""}
1840
1841             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1842         } else if coolingExperiment == "pump-probe" {
1843             xrange := []float64{2.0, 2.5}
1844             yrange := []float64{-0.05, 2.5}
1845             xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
1846             ytick := []float64{0, 0.5, 1, 1.5, 2, 2.5}
1847             xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
1848             ytickLabel := []string{"0", "", "1", "", "2", ""}
1849
1850             return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1851         }
1852     case "UHNA3":
1853         xrange := []float64{1, 1.36}
1854         yrange := []float64{0, 17.5}
1855         xtick := []float64{1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4}
1856         ytick := []float64{0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5}
1857         xtickLabel := []string{"1", "", "1.1", "", "1.2", "", "1.3", "", "1.4"}
1858         ytickLabel := []string{"0", "", "5", "", "10", "", "15", ""}
1859
1860         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1861     case "[Unspecified Sample]":
1862         xrange := []float64{2, 2.5}
1863         yrange := []float64{0, 17.5}
1864         xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
1865         ytick := []float64{0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5}
1866         xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
1867         ytickLabel := []string{"0", "", "5", "", "10", "", "15", ""}
1868
1869         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1870     }
1871 case "pow vs wid":
1872     switch sample {
1873     case "LCOF":
1874         xrange := []float64{0, 300}
1875         yrange := []float64{90, 120}
1876         xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300}
1877         ytick := []float64{90, 92.5, 95, 97.5, 100, 102.5, 105, 107.5, 110, 112.5, 115, 117.5,
120}
1878         xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200", "", "250", "",

```

```

1878     "300"}
1879     ytickLabel := []string{"90", "", "95", "", "100", "", "105", "", "110", "", "115", "",
1880     "120"}
1881
1882     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1883 case "UHNA3":
1884     xrange := []float64{0, 200}
1885     yrange := []float64{90, 130}
1886     xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1887     ytick := []float64{90, 95, 100, 105, 110, 115, 120, 125, 130}
1888     xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1889     ytickLabel := []string{"90", "", "100", "", "110", "", "120", "", "130"}
1890
1891     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1892 case "[Unspecified Sample]":
1893     xrange := []float64{0, 200}
1894     yrange := []float64{90, 130}
1895     xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1896     ytick := []float64{90, 95, 100, 105, 110, 115, 120, 125, 130}
1897     xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1898     ytickLabel := []string{"90", "", "100", "", "110", "", "120", "", "130"}
1899
1900     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1901 }
1902 case "height ratios":
1903     switch sample {
1904     case "LCOF":
1905         xrange := []float64{0, 300}
1906         yrange := []float64{1, 1.5}
1907         xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300}
1908         ytick := []float64{1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4, 1.45, 1.5}
1909         xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200", "", "250", "",
1910         "300"}
1911         ytickLabel := []string{"1", "", "1.1", "", "1.2", "", "1.3", "", "1.4", "", "1.5"}
1912
1913         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1914     case "UHNA3":
1915         xrange := []float64{0, 200}
1916         yrange := []float64{90, 130}
1917         xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1918         ytick := []float64{90, 95, 100, 105, 110, 115, 120, 125, 130}
1919         xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1920         ytickLabel := []string{"90", "", "100", "", "110", "", "120", "", "130"}
1921
1922         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1923     case "[Unspecified Sample]":
1924         xrange := []float64{0, 200}
1925         yrange := []float64{90, 130}
1926         xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1927         ytick := []float64{90, 95, 100, 105, 110, 115, 120, 125, 130}
1928         xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1929         ytickLabel := []string{"90", "", "100", "", "110", "", "120", "", "130"}
1930
1931         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1932     }
1933 case "linewidths":
1934     switch sample {
1935     case "LCOF":
1936         xrange := []float64{0, 300}
1937         yrange := []float64{90, 110}
1938         xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300}
1939         ytick := []float64{90, 92.5, 95, 97.5, 100, 102.5, 105, 107.5, 110}
1940         xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200", "", "250", "",
1941         "300"}
1942         ytickLabel := []string{"90", "", "95", "", "100", "", "105", "", "110"}
1943
1944         return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1945     case "UHNA3":

```

```

1942     xrange := []float64{25, 200}
1943     yrange := []float64{50, 125}
1944     xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1945     ytick := []float64{50, 62.5, 75, 87.5, 100, 112.5, 125}
1946     xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1947     ytickLabel := []string{"50", "", "75", "", "100", "", "125"}
1948
1949     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1950 case "[Unspecified Sample]":
1951     xrange := []float64{25, 200}
1952     yrange := []float64{50, 125}
1953     xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
1954     ytick := []float64{50, 62.5, 75, 87.5, 100, 112.5, 125}
1955     xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
1956     ytickLabel := []string{"50", "", "75", "", "100", "", "125"}
1957
1958     return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
1959 }
1960 }
1961
1962 return []float64{}, []float64{}, []float64{}, []float64{}, []string{},
1963 []string{}, fmt.Errorf("func axes: no predefined axis for '%s'", plot)
1964 }
1965
1966 func subtractBackground(
1967     ras, bas, rs, bs [][]float64,
1968     coolingExperiment string,
1969 ) (
1970     [][]float64, [][]float64,
1971 ) {
1972
1973     var s, as [][]float64
1974
1975     for i := range rs {
1976         s = append(s, subtract(bs[i], rs[i], false))
1977     }
1978
1979     for i := range ras {
1980         as = append(as, subtract(bas[i], ras[i], false))
1981     }
1982
1983     return s, as
1984 }
1985
1986 func subtract(
1987     b, s [][]float64,
1988     sOutlier bool,
1989 ) (
1990     [][]float64,
1991 ) {
1992
1993     //var shift float64
1994
1995     /*
1996     // Outlier applies to pump-only cooling data
1997     if sOutlier {
1998         shift = -(avg(s[1][:100]) - avg(b[1][:100]))
1999     } else {
2000         shift = -(avg(s[1][:100]) - avg(b[1][:100]))
2001     }
2002     */
2003
2004     //shift = -(avg(s[1][:10]) - avg(b[1][:10]))
2005     //shift = -(s[1][0] - b[1][0])
2006
2007     for i := range b[0] {
2008         s[1][i] = s[1][i] - b[1][i] //+ shift
2009     }

```

```

2010
2011     return s
2012 }
2013
2014 func plotSubtracted(
2015     sets []int,
2016     s, as [][]float64,
2017     sLabel, asLabel []string,
2018 ) {
2019
2020     for _, set := range sets {
2021         dimensions := 2
2022         persist := true
2023         debug := false
2024         plot, _ := glot.NewPlot(dimensions, persist, debug)
2025
2026         plot.SetTitle("Background Subtracted")
2027         plot.SetXLabel("Frequency (GHz)")
2028         plot.SetYLabel("Signal (uV)")
2029
2030         plot.AddPointGroup(strings.Trim(sLabel[sets[set]], " rs") + " s", "points", s[sets[set]
2031         plot.AddPointGroup(strings.Trim(asLabel[sets[set]], " ras") + " as", "points", as[sets[
2032     ]}
2033 }
2034
2035 func plotSubtractedTogether(
2036     sets []int,
2037     as, s [][]float64,
2038     asLabel, sLabel []string,
2039 ) {
2040
2041     dimensions := 2
2042     persist := true
2043     debug := false
2044     plot, _ := glot.NewPlot(dimensions, persist, debug)
2045
2046     plot.SetTitle("Background Subtracted")
2047     plot.SetXLabel("Frequency (GHz)")
2048     plot.SetYLabel("Signal (uV)")
2049
2050     for _, set := range sets {
2051         //plot.AddPointGroup(strings.Trim(sLabel[sets[set]], " rs") + " s", "points", s[sets[set]
2052         plot.AddPointGroup(strings.Trim(asLabel[sets[set]], " ras") + " as", "points", as[sets[
2053     ]}
2054 }
2055
2056 func goPlotSubGrpd(
2057     sets []int,
2058     s, as [][]float64,
2059     os, oas [][]float64,
2060     sLabel, asLabel []string,
2061     logpath, sample, coolingExperiment string,
2062     slide bool,
2063 ) {
2064
2065     type errorPoints struct {
2066         plotter.XYs
2067         plotter.YErrors
2068     }
2069
2070     // Anti-Stokes
2071     title := "Anti-Stokes"
2072     xlabel := "Frequency (GHz)"
2073     ylabel := "Spectral Density (uV)"

```

```

2074 legend := ""
2075
2076 if coolingExperiment == "pump-only" {
2077     legend = "Power"
2078 } else if coolingExperiment == "pump-probe" {
2079     legend = "Pump"
2080 }
2081
2082 xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
2083     "fits", sample, coolingExperiment,
2084 )
2085 if err != nil {
2086     fmt.Println(err)
2087     os.Exit(1)
2088 }
2089
2090 p, t, r := prepPlot(
2091     title, xlabel, ylabel, legend,
2092     xrange, yrange, xtick, ytick,
2093     xtickLabel, ytickLabel,
2094     slide,
2095 )
2096
2097 p.Legend.Left = true
2098 p.Legend.XOffs = vg.Points(25)
2099 p.Legend.YOffs = vg.Points(-50)
2100
2101 for _, set := range sets {
2102
2103     asPts := buildData(as[set])
2104     σasErr := buildErrors(σas[set]) // σ[set][σi]
2105
2106     antiStokes := errorPoints {
2107         XYs: asPts,
2108         YErrors: plotter.YErrors(σasErr),
2109     }
2110
2111     // Make a scatter plotter and set its style.
2112     plotSet, err := plotter.NewScatter(antiStokes)
2113     if err != nil {
2114         fmt.Println(err)
2115         os.Exit(1)
2116     }
2117
2118     plotSet.GlyphStyle.Color = palette(set, false, coolingExperiment)
2119     if slide {
2120         plotSet.GlyphStyle.Radius = vg.Points(5)
2121     } else {
2122         plotSet.GlyphStyle.Radius = vg.Points(3)
2123     }
2124     plotSet.Shape = draw.CircleGlyph{}
2125
2126     // Error bars
2127     e, err := plotter.NewYErrorBars(antiStokes)
2128     if err != nil {
2129         fmt.Println(err)
2130         os.Exit(1)
2131     }
2132     e.LineStyle.Color = palette(set, false, coolingExperiment)
2133
2134     p.Add(e, t, r) // plotSet,
2135
2136     // Legend
2137     l, err := plotter.NewScatter(antiStokes)
2138     if err != nil {
2139         fmt.Println(err)
2140         os.Exit(1)
2141     }

```

```

2142     l.GlyphStyle.Color = palette(set, false, coolingExperiment)
2143     l.GlyphStyle.Radius = vg.Points(6)
2144     l.Shape = draw.CircleGlyph{}
2145     p.Legend.Add(strings.Trim(asLabel[set], " pras"), 1)
2146 }
2147
2148
2149 savePlot(p, "Anti-Stokes Background Subtracted", logpath)
2150
2151 // Stokes
2152 title = "Stokes"
2153 xlabel = "Frequency (GHz)"
2154 ylabel = "Spectral Density (uV)"
2155
2156 if coolingExperiment == "pump-only" {
2157     legend = "Power"
2158 } else if coolingExperiment == "pump-probe" {
2159     legend = "Pump"
2160 }
2161
2162 xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err = axes(
2163     "fits", sample, coolingExperiment,
2164 )
2165 if err != nil {
2166     fmt.Println(err)
2167     os.Exit(1)
2168 }
2169
2170 p, t, r = prepPlot(
2171     title, xlabel, ylabel, legend,
2172     xrange, yrange, xtick, ytick,
2173     xtickLabel, ytickLabel,
2174     slide,
2175 )
2176
2177 p.Legend.Left = true
2178 p.Legend.XOffs = vg.Points(25)
2179 p.Legend.YOffs = vg.Points(-50)
2180
2181 for _, set := range sets {
2182
2183     sPts := buildData(s[set])
2184     osErr := buildErrors(os[set]) //  $\sigma[set][\sigma]$ 
2185
2186     stokes := errorPoints {
2187         XYs: sPts,
2188         YErrors: plotter.YErrors(osErr),
2189     }
2190
2191     // Make a scatter plotter and set its style.
2192     plotSet, err := plotter.NewScatter(stokes)
2193     if err != nil {
2194         fmt.Println(err)
2195         os.Exit(1)
2196     }
2197
2198     plotSet.GlyphStyle.Color = palette(set, true, coolingExperiment)
2199     if slide {
2200         plotSet.GlyphStyle.Radius = vg.Points(5)
2201     } else {
2202         plotSet.GlyphStyle.Radius = vg.Points(3)
2203     }
2204     plotSet.Shape = draw.CircleGlyph{}
2205
2206     // Error bars
2207     e, err := plotter.NewYErrorBars(stokes)
2208     if err != nil {
2209         fmt.Println(err)

```

```

2210     os.Exit(1)
2211 }
2212 e.LineStyle.Color = palette(set, false, coolingExperiment)
2213
2214 p.Add(e, t, r) // plotSet,
2215
2216 // Legend
2217 l, err := plotter.NewScatter(stokes)
2218 if err != nil {
2219     fmt.Println(err)
2220     os.Exit(1)
2221 }
2222
2223 l.GlyphStyle.Color = palette(set, true, coolingExperiment)
2224 l.GlyphStyle.Radius = vg.Points(6)
2225 l.Shape = draw.CircleGlyph{}
2226
2227 p.Legend.Add(strings.Trim(sLabel[set], " rs"), l)
2228 }
2229
2230 savePlot(p, "Stokes Background Subtracted", logpath)
2231 }
2232
2233 func generateFitData(
2234     amp, wid, cen, c, f0, df float64,
2235     fitPts int,
2236 ) (
2237     [][]float64,
2238 ) {
2239
2240     x := make([]float64, fitPts)
2241     for i := range x {
2242         x[i] = f0 + df*float64(i)
2243     }
2244
2245     y := make([]float64, fitPts)
2246     for i := range x {
2247         // (amp*wid^2/((x-cen)^2+wid^2))
2248         y[i] = .25 * amp * math.Pow(wid, 2) / (math.Pow(x[i] - cen, 2) + (.25 * math.Pow(wid, 2)
2249         )) + c
2250     }
2251
2252     return [][]float64{x, y}
2253 }
2254
2255 func goPlotasFits(
2256     sets []int,
2257     as, fits, widthLines [][][]float64,
2258     oas [][]float64,
2259     labels []string,
2260     widths, notes []float64,
2261     temp, slide bool,
2262     sample, logpath, coolingExperiment string,
2263 ) {
2264
2265     type errorPoints struct {
2266         plotter.XYs
2267         plotter.YErrors
2268     }
2269
2270     title := " "
2271     xlabel := "Frequency (GHz)"
2272     ylabel := "Spectral Density (uV)"
2273     legend := ""
2274
2275     if coolingExperiment == "pump-only" {
2276         legend = "Power"
2277     } else if coolingExperiment == "pump-probe" {

```

```

2277     legend = "Pump"
2278 }
2279
2280 xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
2281     "fits", sample, coolingExperiment,
2282 )
2283 if err != nil {
2284     fmt.Println(err)
2285     os.Exit(1)
2286 }
2287
2288 p, t, r := prepPlot(
2289     title, xlabel, ylabel, legend,
2290     xrange, yrange, xtick, ytick,
2291     xtickLabel, ytickLabel,
2292     slide,
2293 )
2294
2295 p.Legend.Left = true
2296 p.Legend.XOffs = vg.Points(25)
2297 p.Legend.YOffs = vg.Points(-50)
2298
2299 for i, set := range sets {
2300
2301     pts := buildData(as[set])
2302     fit := buildData(fits[i])
2303     wid := buildData(widthLines[i])
2304     sigmaErr := buildErrors(sigma[set]) //  $\sigma[set][\sigma i]$ 
2305
2306     antiStokes := errorPoints {
2307         XYs: pts,
2308         YErrors: plotter.YErrors(sigmaErr),
2309     }
2310
2311     // Plot points
2312     plotPts, err := plotter.NewScatter(antiStokes)
2313     if err != nil {
2314         fmt.Println(err)
2315         os.Exit(1)
2316     }
2317
2318     plotPts.GlyphStyle.Color = palette(set, false, coolingExperiment)
2319     if slide {
2320         plotPts.GlyphStyle.Radius = vg.Points(5)
2321     } else {
2322         plotPts.GlyphStyle.Radius = vg.Points(3)
2323     }
2324     plotPts.Shape = draw.CircleGlyph{}
2325
2326     // Plot fit
2327     plotFit, err := plotter.NewLine(fit)
2328     if err != nil {
2329         fmt.Println(err)
2330         os.Exit(1)
2331     }
2332
2333     plotFit.LineStyle.Color = palette(set, true, coolingExperiment)
2334     plotFit.LineStyle.Width = vg.Points(3)
2335
2336     // Width lines
2337     plotWid, err := plotter.NewLine(wid)
2338     if err != nil {
2339         fmt.Println(err)
2340         os.Exit(1)
2341     }
2342
2343     plotWid.LineStyle.Color = palette(set, true, coolingExperiment)
2344     plotWid.LineStyle.Width = vg.Points(4)

```



```

2345     plotWid.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
2346
2347     // Y Error bars
2348     e, err := plotter.NewYErrorBars(antiStokes)
2349     if err != nil {
2350         fmt.Println(err)
2351         os.Exit(1)
2352     }
2353     e.LineStyle.Color = palette(set, false, coolingExperiment)
2354
2355     // Add set plots to p
2356     p.Add(e, t, r, plotFit) // , plotWid
2357
2358     // Legend
2359     l, err := plotter.NewScatter(pts)
2360     if err != nil {
2361         fmt.Println(err)
2362         os.Exit(1)
2363     }
2364
2365     l.GlyphStyle.Color = palette(set, true, coolingExperiment)
2366     l.GlyphStyle.Radius = vg.Points(6)
2367     l.Shape = draw.CircleGlyph{}
2368     power := strings.Trim(labels[set], " pras")
2369     if temp {
2370         temp := strconv.FormatFloat(notes[set], 'f', -1, 64)
2371         p.Legend.Add(power + " @" + temp + "K", l)
2372     } else {
2373         p.Legend.Add(power, l)
2374     }
2375
2376 }
2377
2378 savePlot(p, "Anti-Stokes w Fits", logpath)
2379 }
2380
2381 func plotSinc(
2382     sets []int,
2383     phaseMatchData [][]float64,
2384     label []string,
2385     sample, logpath string,
2386     length float64,
2387     slide bool,
2388 ) {
2389
2390     pts := buildData(phaseMatchData)
2391
2392     var l string
2393
2394     switch length {
2395     case 0.0:
2396         l = ""
2397     case 0.001:
2398         l = "1 mm"
2399     case 0.01:
2400         l = "1 cm"
2401     case 0.004:
2402         l = "4 mm"
2403     case 0.0000005:
2404         l = "500 nm"
2405     case 0.00001:
2406         l = "10 μm"
2407     default:
2408         l = strconv.FormatFloat(length, 'f', 1, 64)
2409     }
2410
2411     title := l + " " + sample + " Phase-Matching Bandwidth"
2412     xlabel := "Pump-Probe Separation (GHz)"

```

```

2413 ylabel := "Peak Spectral Density"
2414 legend := ""
2415
2416 // Auto Axes
2417 xmin, xmax, ymin, ymax := 0., 0., 0., 0.
2418 for i, v := range phaseMatchData[0] {
2419     if v > xmax {
2420         xmax = v
2421     }
2422     if phaseMatchData[1][i] > ymax {
2423         ymax = phaseMatchData[1][i]
2424     }
2425 }
2426
2427 xrange := []float64{xmin, xmax}
2428 yrange := []float64{ymin, ymax}
2429
2430 xtick := 0.
2431 displayDigits := 2
2432 if (xmax - xmin)/8 > 5 {
2433     xtick = 5
2434 } else if (xmax - xmin)/8 > 2.5 {
2435     xtick = 2.5
2436 } else if (xmax - xmin)/8 > 1 {
2437     xtick = 1
2438 } else if (xmax - xmin)/8 > 0.5 {
2439     xtick = 0.5
2440 }
2441 firstTick := 0.
2442 for m := float64(int(xmin)); m <= xmax; m += xtick {
2443     firstTick = m
2444 }
2445 xticks := []float64{}
2446 xtickLabels := []string{}
2447 for i := 0.; firstTick + xtick*i <= xmax - xtick/2; i++ {
2448     xticks = append(xticks, firstTick + xtick*i)
2449     if int(i)%2 != 0 {
2450         xtickLabels = append(xtickLabels, strconv.FormatFloat(firstTick + xtick*i, 'f',
2451             displayDigits, 64))
2452     } else {
2453         xtickLabels = append(xtickLabels, "")
2454     }
2455 }
2456 ytick := ((ymax - ymin)/8)
2457 yticks := []float64{}
2458 ytickLabels := []string{}
2459 for i := 0.; i < 11; i++ {
2460     yticks = append(yticks, ytick*i + ymin)
2461     if int(i)%2 != 0 {
2462         ytickLabels = append(ytickLabels, strconv.FormatFloat(ytick*i + ymin, 'f', 2, 64))
2463     } else {
2464         ytickLabels = append(ytickLabels, "")
2465     }
2466 }
2467 yticks = append(yticks, ymax)
2468 ytickLabels = append(ytickLabels, "")
2469
2470 p, t, r := prepPlot(
2471     title, xlabel, ylabel, legend,
2472     xrange, yrange, xticks, yticks,
2473     xtickLabels, ytickLabels,
2474     slide,
2475 )
2476
2477 scatter, err := plotter.NewScatter(pts)
2478 if err != nil {
2479     log.Fatalf("Could not create line for the first series: %v", err)

```

```

2480 }
2481 scatter.GlyphStyle.Color = palette(0, false, "")
2482 scatter.GlyphStyle.Radius = vg.Points(5) //3
2483 scatter.Shape = draw.CircleGlyph{}
2484 p.Add(scatter, t, r)
2485
2486 savePlot(p, "Phase-Match", logpath)
2487 }
2488
2489 func bin(
2490     sets []int,
2491     as, s [][]float64,
2492     binMHz float64,
2493 ) (
2494     [][]float64, [][]float64,
2495 ) {
2496
2497     binGHz := binMHz/1000
2498     nBins := int((as[0][0][len(as[0][0]) - 1] - as[0][0][0])/binGHz + 1)
2499
2500     // [set][0: freq, 1: sig, 2: err][values]
2501     asBinned := make([][]float64, len(as))
2502     for i := range asBinned {
2503         asBinned[i] = make([]float64, 3)
2504         for j := range asBinned[i] {
2505             asBinned[i][j] = make(float64, nBins)
2506         }
2507     }
2508     sBinned := make([][]float64, len(s))
2509     for i := range sBinned {
2510         sBinned[i] = make(float64, 3)
2511         for j := range sBinned[i] {
2512             sBinned[i][j] = make(float64, nBins)
2513         }
2514     }
2515
2516     for _, set := range sets {
2517
2518         asBound := as[set][0][0]
2519         sBound := s[set][0][0]
2520
2521         for i := 0; i < nBins; i++ {
2522             asBound += binGHz
2523             sBound += binGHz
2524
2525             var asSigsInBin, sSigsInBin []float64
2526
2527             for j, f := range as[set][0] {
2528                 if f < asBound && f > asBound - binGHz {
2529                     asSigsInBin = append(asSigsInBin, as[set][1][j])
2530                 }
2531             }
2532             asBinned[set][0][i] = asBound - (binGHz/2)
2533             asBinned[set][1][i] = avg(asSigsInBin)
2534
2535             for j, f := range s[set][0] {
2536                 if f < sBound && f > sBound - binGHz {
2537                     sSigsInBin = append(sSigsInBin, s[set][1][j])
2538                 }
2539             }
2540             sBinned[set][0][i] = sBound - (binGHz/2)
2541             sBinned[set][1][i] = avg(sSigsInBin)
2542
2543             // Error for each binned point
2544             asBinned[set][2][i] =  $\sigma$ (asSigsInBin)
2545             sBinned[set][2][i] =  $\sigma$ (sSigsInBin)
2546         }
2547     }

```

```

2548     return asBinned, sBinned
2549 }
2550
2551 func  $\sigma$ CABS(
2552     setsToPlotCABS, numAvgs []int,
2553     cabsData [][]float64,
2554     sigUnit string,
2555     sigmaMultiple float64,
2556     normalized []string,
2557 ) (
2558     [][]float64,
2559 ) {
2560
2561     N := "Number of Averages"
2562
2563     if N == "Sampling Rate" {
2564         // Sampling rate = 1,842,000 usually
2565         // 1. figure  $\sigma$  from dwell-time  $\sigma$  in CSVs
2566         largestSet := setsToPlotCABS[0]
2567         for _, v := range setsToPlotCABS {
2568             if v > largestSet {
2569                 largestSet = v
2570             }
2571         }
2572
2573         stdDevBg := make([][]float64, largestSet+1)
2574         stdDevSig := make([][]float64, largestSet+1)
2575
2576         for _, set := range setsToPlotCABS {
2577
2578             stdDevBg[set] = make([]float64, numAvgs[set])
2579             stdDevSig[set] = make([]float64, numAvgs[set])
2580
2581             for run := 0; run < numAvgs[set]; run++ {
2582
2583                 // Background
2584                 // Open the CSV file for reading
2585                 csvPath := "Data/" + fmt.Sprintf(set+1) + "/Runs/Background/Run " + fmt.Sprintf(run) +
2586                 ".csv"
2587                 file, err := os.Open(csvPath)
2588                 if err != nil {
2589                     fmt.Println("Error:", err)
2590                     return cabsData
2591                 }
2592
2593                 // Create a CSV reader
2594                 reader := csv.NewReader(file)
2595
2596                 // Read all CSV records
2597                 records, err := reader.ReadAll()
2598                 if err != nil {
2599                     fmt.Println("Error:", err)
2600                     return cabsData
2601                 }
2602                 file.Close()
2603
2604                 // Iterate through the records (skip the first header row)
2605                 for row, record := range records {
2606                     if row == 0 {
2607                         // Skip the header row
2608                         continue
2609                     }
2610
2611                     // Ensure there are at least 2 columns in the record
2612                     if len(record) >= 2 {
2613                         record[1] = strings.TrimSpace(record[1])
2614                         // Convert the second column to a float64 and append to the slice

```

```

2615         values, err := strconv.ParseFloat(record[1], 64)
2616         if err != nil {
2617             fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
2618         } else {
2619             stdDevBg[set][run] = append(stdDevBg[set][run], values)
2620         }
2621     } else {
2622         fmt.Printf("Row %d does not have enough columns\n", row+1)
2623     }
2624 }
2625
2626 // Signal
2627 // Open the CSV file for reading
2628 csvPath = "Data/" + fmt.Sprint(set+1) + "/Runs/Signal/Run " + fmt.Sprint(run) + ".
csv"
2629 file, err = os.Open(csvPath)
2630 if err != nil {
2631     fmt.Println("Error:", err)
2632     return cabsData
2633 }
2634
2635 // Create a CSV reader
2636 reader = csv.NewReader(file)
2637
2638 // Read all CSV records
2639 records, err = reader.ReadAll()
2640 if err != nil {
2641     fmt.Println("Error:", err)
2642     return cabsData
2643 }
2644 file.Close()
2645
2646 // Iterate through the records (skip the first header row)
2647 for row, record := range records {
2648     if row == 0 {
2649         // Skip the header row
2650         continue
2651     }
2652
2653     // Ensure there are at least 2 columns in the record
2654     if len(record) >= 2 {
2655
2656         record[1] = strings.TrimSpace(record[1])
2657         // Convert the second column to a float64 and append to the slice
2658         values, err := strconv.ParseFloat(record[1], 64)
2659         if err != nil {
2660             fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
2661         } else {
2662             stdDevSig[set][run] = append(stdDevSig[set][run], values)
2663         }
2664     } else {
2665         fmt.Printf("Row %d does not have enough columns\n", row+1)
2666     }
2667 }
2668 }
2669 }
2670
2671 // combine  $\sigma$  for each freq across runs
2672 for _, set := range setsToPlotCABS {
2673
2674      $\sigma$ CombinedAcrossRunsBg := make([]float64, len(stdDevBg[set][0]))
2675      $\sigma$ CombinedAcrossRunsSig := make([]float64, len(stdDevBg[set][0]))
2676
2677     for i := range stdDevBg[set][0] {
2678
2679         for run := 0; run < numAvgs[set]; run++ {
2680
2681             // !assumes sig and background have same # of runs within a set

```

```

2682          $\sigma$ CombinedAcrossRunsBg[i] += math.Pow(stdDevBg[set][run][i], 2)
2683          $\sigma$ CombinedAcrossRunsSig[i] += math.Pow(stdDevSig[set][run][i], 2)
2684     }
2685 }
2686
2687 // square root of sum of squares, divided by number of runs
2688 for i := range  $\sigma$ CombinedAcrossRunsBg {
2689      $\sigma$ CombinedAcrossRunsBg[i] = math.Sqrt( $\sigma$ CombinedAcrossRunsBg[i])/float64(numAvgs[set])
2690      $\sigma$ CombinedAcrossRunsSig[i] = math.Sqrt( $\sigma$ CombinedAcrossRunsSig[i])/float64(numAvgs[set]
2691 ])
2692 }
2693
2694 // propagate errors through signal - background
2695
2696  $\sigma$ SigMinusBg := make([]float64, len( $\sigma$ CombinedAcrossRunsSig))
2697
2698 for i, v := range  $\sigma$ CombinedAcrossRunsSig {
2699      $\sigma$ SigMinusBg[i] += math.Sqrt(math.Pow(v, 2) + math.Pow( $\sigma$ CombinedAcrossRunsBg[i], 2))
2700
2701     // only scale to appropriate unit if not being normalized later
2702     if len(normalized) > 0 {
2703         switch sigUnit {
2704             case "mV":
2705                  $\sigma$ SigMinusBg[i] *= 1e3
2706             case "uV":
2707                  $\sigma$ SigMinusBg[i] *= 1e6
2708             case "nV":
2709                  $\sigma$ SigMinusBg[i] *= 1e9
2710             case "pV":
2711                  $\sigma$ SigMinusBg[i] *= 1e12
2712         }
2713     }
2714 }
2715
2716 // 1 $\sigma$  = 68.27%, 2 $\sigma$  = 95.45%, 3 $\sigma$  = 99.73%
2717 //  $\sigma$ SigMinusBg[i] =  $\sigma$ SigMinusBg[i]*2
2718
2719 }
2720
2721 // 2. tack errors onto cabsData
2722 // cabsData[set][0: freq, 1: sig, 2:  $\sigma$ ][rows of freq/sig/ $\sigma$ ]
2723 cabsData[set] = append(cabsData[set],  $\sigma$ SigMinusBg)
2724 }
2725 } else if N == "Number of Averages" {
2726
2727     // Number of Averages = 5 usually
2728     // Calculate  $\sigma$  across subtracted runs
2729     largestSet := setsToPlotCABS[0]
2730     for _, v := range setsToPlotCABS {
2731         if v > largestSet {
2732             largestSet = v
2733         }
2734     }
2735
2736     runVals := make([][][]float64, largestSet+1)
2737
2738     for _, set := range setsToPlotCABS {
2739         runVals[set] = make([][]float64, numAvgs[set])
2740
2741         for run := 0; run < numAvgs[set]; run++ {
2742             // Open the CSV file for reading
2743             csvPath := "Data/" + fmt.Sprint(set+1) + "/Runs/Subtracted/Run " + fmt.Sprint(run) +
2744             ".csv"
2745             file, err := os.Open(csvPath)
2746             if err != nil {
2747

```

```

2748         fmt.Println("Error:", err)
2749         return cabsData
2750     }
2751
2752     // Create a CSV reader
2753     reader := csv.NewReader(file)
2754
2755     // Read all CSV records
2756     records, err := reader.ReadAll()
2757     if err != nil {
2758         fmt.Println("Error:", err)
2759         return cabsData
2760     }
2761     file.Close()
2762
2763     // Iterate through the records
2764     for row, record := range records {
2765
2766         record[0] = strings.TrimSpace(record[0])
2767         // Convert the column to a float64 and append to the slice
2768         values, err := strconv.ParseFloat(record[0], 64)
2769         if err != nil {
2770             fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
2771         } else {
2772             runVals[set][run] = append(runVals[set][run], values)
2773         }
2774     }
2775 }
2776
2777 transposedRunVals := transpose(runVals[set])
2778
2779 stdDev := make([]float64, len(runVals[set][0]))
2780
2781 for i, v := range transposedRunVals {
2782
2783     stdDev[i] =  $\sigma(v)$ *sigmaMultiple
2784 }
2785
2786 // only scale to appropriate unit if not being normalized later
2787 if !(len(normalized) > 0) {
2788     for i := range stdDev {
2789         switch sigUnit {
2790             case "mV":
2791                 stdDev[i] *= 1e3
2792             case "µV":
2793                 stdDev[i] *= 1e6
2794             case "nV":
2795                 stdDev[i] *= 1e9
2796             case "pV":
2797                 stdDev[i] *= 1e12
2798             default:
2799                 fmt.Printf("Warning: Unrecognized signal unit '%s' for standard deviation
2800 scaling.\n", sigUnit)
2801         }
2802     }
2803 }
2804
2805 cabsData[set] = append(cabsData[set], stdDev)
2806 }
2807 }
2808
2809 return cabsData
2810 }
2811
2812 func contains(
2813     list []string, a string,
2814 ) (

```

```

2815     bool,
2816 ) {
2817     for _, b := range list {
2818         if b == a {
2819             return true
2820         }
2821     }
2822     return false
2823 }
2824
2825 func normalizeByPowers(
2826     setsToPlotCABS []int,
2827     cabsData [][]float64,
2828     pumpPowers, stokesPowers, probePowers []float64,
2829 ) (
2830     [][]float64,
2831 ) {
2832
2833     for _, set := range setsToPlotCABS {
2834
2835         // mW -> W
2836         pumpPowers[set] /= 1e3
2837         stokesPowers[set] /= 1e3
2838         probePowers[set] /= 1e3
2839
2840         // sig
2841         for i := range cabsData[set][1] {
2842             cabsData[set][1][i] /= pumpPowers[set]*stokesPowers[set]*probePowers[set]
2843         }
2844
2845         //  $\sigma$ 
2846         for i := range cabsData[set][2] {
2847             cabsData[set][2][i] /= pumpPowers[set]*stokesPowers[set]*probePowers[set]
2848         }
2849     }
2850 }
2851
2852 return cabsData
2853 }
2854
2855 func FitLorentzian(
2856     frequencies, signals, uncertainties, initialParams []float64,
2857 ) (
2858     []float64,
2859 ) {
2860
2861     // Define the residual function
2862     resFunc := func(dst, params []float64) {
2863         r := residuals(params, frequencies, signals, uncertainties)
2864         for i := range r {
2865             dst[i] = r[i]
2866         }
2867     }
2868 }
2869
2870 // Define NumJac instance
2871 nj := &lm.NumJac{Func: resFunc}
2872
2873 // Problem definition
2874 problem := lm.LMProblem{
2875     Dim:      4,
2876     Size:     len(frequencies),
2877     Func:     resFunc,
2878     Jac:      nj.Jac, // Use the numerical Jacobian
2879     InitParams: initialParams,
2880     Tau:      1e-6,
2881     Eps1:     1e-8,
2882     Eps2:     1e-8,

```



```

2883 }
2884
2885 settings := &lm.Settings{Iterations: 1000, ObjectiveTol: 1e-16}
2886
2887 result, err := lm.LM(problem, settings)
2888 if err != nil {
2889     log.Fatal("optimization failed:", err)
2890 }
2891
2892 return result.X
2893 }
2894
2895 func Lorentzian(
2896     f, A, f0, gamma, C float64,
2897 ) (
2898     float64,
2899 ) {
2900     return .25 * A * math.Pow(gamma, 2) / (math.Pow(f - f0, 2) + (.25 * math.Pow(gamma, 2)))
2901         + C
2902     //return (A / math.Pi) * (gamma / (math.Pow(f-f0, 2) + math.Pow(gamma, 2))) + C
2903 }
2904
2905 func residuals(
2906     params, frequencies, signals, uncertainties []float64,
2907 ) (
2908     []float64,
2909 ) {
2910     A, f0, gamma, C := params[0], params[1], params[2], params[3]
2911     r := make([]float64, len(frequencies))
2912
2913     for i, f := range frequencies {
2914         modelValue := Lorentzian(f, A, f0, gamma, C)
2915         if len(uncertainties) > 0 && uncertainties[i] != 0 {
2916             r[i] = (signals[i] - modelValue) / uncertainties[i]
2917         } else {
2918             r[i] = signals[i] - modelValue
2919         }
2920     }
2921
2922     return r
2923 }
2924
2925 func binCabs(
2926     setsToBin []int,
2927     cabsData [][]float64,
2928     binMHz float64,
2929 ) (
2930     [][]float64,
2931 ) {
2932     binGHz := binMHz/1000
2933     nBins := make([]int, len(cabsData))
2934     for set := range cabsData {
2935         yesBin := false
2936         for _, setToBin := range setsToBin {
2937             if set == setToBin {
2938                 yesBin = true
2939             }
2940         }
2941
2942         if yesBin {
2943             n := int((cabsData[set][0][len(cabsData[set][0]) - 1] - cabsData[set][0][0])/binGHz +
2944                 1)
2945             nBins[set] = n
2946         } else {
2947             nBins[set] = len(cabsData[set][0])
2948         }
2949     }
2950 }

```

```

2949
2950 // [set][0: freq, 1: sig, 2: err][values]
2951 cabsBinned := make([][]float64, len(cabsData))
2952 for set := range cabsBinned {
2953
2954     yesBin := false
2955     for _, setToBin := range setsToBin {
2956         if set == setToBin {
2957             yesBin = true
2958         }
2959     }
2960
2961     if yesBin {
2962         cabsBinned[set] = make([][]float64, 3) // freq, sig, err
2963         for i := range cabsBinned[set] {
2964             cabsBinned[set][i] = make([]float64, nBins[set]) // len(cabsData[set][0])
2965         }
2966     } else {
2967         cabsBinned[set] = make([][]float64, 2) // freq, sig, err
2968         for i := range cabsBinned[set] {
2969             cabsBinned[set][i] = make([]float64, nBins[set])
2970         }
2971     }
2972 }
2973
2974 for set := range cabsData {
2975
2976     yesBin := false
2977     for _, setToBin := range setsToBin {
2978         if set == setToBin {
2979             yesBin = true
2980         }
2981     }
2982
2983     if yesBin {
2984         cabsBound := cabsData[set][0][0]
2985
2986         for i := 0; i < nBins[set]; i++ {
2987             cabsBound += binGHz
2988
2989             var cabsSigsInBin []float64
2990
2991             for j, f := range cabsData[set][0] {
2992                 if f < cabsBound && f > cabsBound - binGHz {
2993                     cabsSigsInBin = append(cabsSigsInBin, cabsData[set][1][j])
2994                 }
2995             }
2996             cabsBinned[set][0][i] = cabsBound - (binGHz/2)
2997             cabsBinned[set][1][i] = avg(cabsSigsInBin)
2998
2999             // Error for each binned point
3000             cabsBinned[set][2][i] =  $\sigma$ CABSbins(cabsSigsInBin)
3001         }
3002     } else {
3003         for i, v := range cabsData[set][0] {
3004             cabsBinned[set][0][i] = v
3005             cabsBinned[set][1][i] = cabsData[set][1][i]
3006         }
3007     }
3008 }
3009
3010 return cabsBinned
3011 }
3012
3013 func transpose(
3014     slice [][]float64,
3015 ) (
3016     [][]float64,

```

```

3017 ) {
3018
3019     rows := len(slice)
3020     cols := len(slice[0])
3021
3022     transposed := make([][]float64, cols)
3023
3024     for i := 0; i < cols; i++ {
3025         transposed[i] = make([]float64, rows)
3026         for j := 0; j < rows; j++ {
3027             transposed[i][j] = slice[j][i]
3028         }
3029     }
3030
3031     return transposed
3032 }
3033
3034 func  $\sigma$ (
3035     values []float64,
3036 ) (
3037     float64,
3038 ) {
3039
3040     /* dBm -> uV
3041     for i, v := range values {
3042         values[i] = math.Pow(10, 6)*math.Pow(10, v/10.)
3043     }*/
3044
3045     // Sum of squares of the difference
3046     dev := 0.
3047     for _, v := range values {
3048         dev += math.Pow(v - avg(values), 2)
3049     }
3050     n := float64(len(values))
3051
3052     // Standard deviation of the mean
3053     return math.Sqrt((1/(n - 1) * dev))/math.Sqrt(n)
3054 }
3055
3056 func  $\sigma$ CABSBins(
3057     values []float64,
3058 ) (
3059     float64,
3060 ) {
3061
3062     // Sum of squares of the difference
3063     dev := 0.
3064     for _, v := range values {
3065         dev += math.Pow(v - avg(values), 2)
3066     }
3067     n := float64(len(values))
3068
3069     // Standard deviation of the mean
3070     return math.Sqrt((1/(n - 1) * dev))/math.Sqrt(n)
3071 }
3072
3073 func avg(
3074     toAvg []float64,
3075 ) (
3076     float64,
3077 ) {
3078
3079     sum := 0.
3080     for _, v := range toAvg {
3081         sum += v
3082     }
3083     return sum/float64(len(toAvg))
3084 }

```

```

3085
3086 func goPlotasPowerVsWid(
3087     sets []int,
3088     labels []string,
3089     notes, widths []float64,
3090     temp, slide bool,
3091     sample, logpath, coolingExperiment string,
3092 ) {
3093
3094     title := " "
3095     xlabel := "Pump Power (mW)"
3096     ylabel := "FWHM (MHz)"
3097     legend := ""
3098
3099     xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
3100         "pow vs wid", sample, coolingExperiment,
3101     )
3102     if err != nil {
3103         fmt.Println(err)
3104         os.Exit(1)
3105     }
3106
3107     p, t, r := prepPlot(
3108         title, xlabel, ylabel, legend,
3109         xrange, yrange, xtick, ytick,
3110         xtickLabel, ytickLabel,
3111         slide,
3112     )
3113
3114     for i, set := range sets {
3115
3116         pts := make(plotter.XYs, 1)
3117
3118         power := strings.Trim(labels[set], " mW pras")
3119         if pwr, err := strconv.ParseFloat(power, 64); err == nil {
3120             pts[0].X = pwr
3121         } else {
3122             fmt.Println(err)
3123             os.Exit(1)
3124         }
3125         pts[0].Y = widths[i]
3126
3127         // Plot points
3128         plotPts, err := plotter.NewScatter(pts)
3129         if err != nil {
3130             fmt.Println(err)
3131             os.Exit(1)
3132         }
3133
3134         plotPts.GlyphStyle.Color = palette(set, true, coolingExperiment)
3135         plotPts.GlyphStyle.Radius = vg.Points(6)
3136         plotPts.Shape = draw.CircleGlyph{}
3137
3138         // Dashed eye guide lines
3139         v := make(plotter.XYs, 2)
3140         h := make(plotter.XYs, 2)
3141
3142         // Vertical
3143         v[0].X = pts[0].X
3144         v[0].Y = yrange[0]
3145         v[1].X = pts[0].X
3146         v[1].Y = pts[0].Y
3147
3148         vDash, err := plotter.NewLine(v)
3149         if err != nil {
3150             fmt.Println(err)
3151             os.Exit(1)
3152         }

```

```

3153
3154     vDash.LineStyle.Color = palette(set, true, coolingExperiment)
3155     vDash.LineStyle.Width = vg.Points(4)
3156     vDash.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
3157
3158     // Horizontal
3159     h[0].X = xrange[0]
3160     h[0].Y = pts[0].Y
3161     h[1].X = pts[0].X
3162     h[1].Y = pts[0].Y
3163
3164     hDash, err := plotter.NewLine(h)
3165     if err != nil {
3166         fmt.Println(err)
3167         os.Exit(1)
3168     }
3169
3170     hDash.LineStyle.Color = color.RGBA{R: 127, G: 127, B: 127, A: 255}
3171     hDash.LineStyle.Width = vg.Points(1)
3172     hDash.LineStyle.Dashes = []vg.Length{vg.Points(5), vg.Points(5)}
3173
3174     // Add set plots to p
3175     p.Add(plotPts, t, r, vDash, hDash)
3176     /*if temp {
3177         temperature := strconv.FormatFloat(notes[set], 'f', -1, 64)
3178         p.Legend.Add(power + " mW @" + temperature + "K", plotPts)
3179     } else {
3180         p.Legend.Add(power + " mW", plotPts)
3181     }*/
3182 }
3183
3184 savePlot(p, "as Pow vs Wid", logpath)
3185 }
3186
3187 func goPlotsFits(
3188     sets []int,
3189     s, fits, widthLines [][][]float64,
3190     os [][]float64,
3191     labels []string,
3192     widths, notes []float64,
3193     temp, slide bool,
3194     sample, logpath, coolingExperiment string,
3195 ) {
3196
3197     type errorPoints struct {
3198         plotter.XYs
3199         plotter.YErrors
3200     }
3201
3202     title := " " // sample + " Stokes"
3203     xlabel := "Frequency (GHz)"
3204     ylabel := "Spectral Density (uV)"
3205     legend := ""
3206
3207     if coolingExperiment == "pump-only" {
3208         legend = "Power"
3209     } else if coolingExperiment == "pump-probe" {
3210         legend = "Pump"
3211     }
3212
3213     xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
3214         "fits", sample, coolingExperiment,
3215     )
3216     if err != nil {
3217         fmt.Println(err)
3218         os.Exit(1)
3219     }
3220

```

```

3221 p, t, r := prepPlot(
3222     title, xlabel, ylabel, legend,
3223     xrange, yrange, xtick, ytick,
3224     xtickLabel, ytickLabel,
3225     slide,
3226 )
3227
3228 p.Legend.Left = true
3229 p.Legend.XOffs = vg.Points(25)
3230 p.Legend.YOffs = vg.Points(-50)
3231
3232 for i, set := range sets {
3233
3234     pts := buildData(s[set])
3235     fit := buildData(fits[i])
3236     wid := buildData(widthLines[i])
3237     σsErr := buildErrors(σs[set]) // σ[set][σi]
3238
3239     stokes := errorPoints {
3240         XYs: pts,
3241         YErrors: plotter.YErrors(σsErr),
3242     }
3243
3244     // Plot points
3245     plotPts, err := plotter.NewScatter(stokes)
3246     if err != nil {
3247         fmt.Println(err)
3248         os.Exit(1)
3249     }
3250
3251     plotPts.GlyphStyle.Color = palette(set, false, coolingExperiment)
3252     if slide {
3253         plotPts.GlyphStyle.Radius = vg.Points(5)
3254     } else {
3255         plotPts.GlyphStyle.Radius = vg.Points(3)
3256     }
3257     plotPts.Shape = draw.CircleGlyph{}
3258
3259     // Plot fit
3260     plotFit, err := plotter.NewLine(fit)
3261     if err != nil {
3262         fmt.Println(err)
3263         os.Exit(1)
3264     }
3265
3266     plotFit.LineStyle.Color = palette(set, true, coolingExperiment)
3267     plotFit.LineStyle.Width = vg.Points(3)
3268
3269     // Width lines
3270     plotWid, err := plotter.NewLine(wid)
3271     if err != nil {
3272         fmt.Println(err)
3273         os.Exit(1)
3274     }
3275
3276     plotWid.LineStyle.Color = palette(set, true, coolingExperiment)
3277     plotWid.LineStyle.Width = vg.Points(4)
3278     plotWid.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
3279
3280     // Error bars
3281     e, err := plotter.NewYErrorBars(stokes)
3282     if err != nil {
3283         fmt.Println(err)
3284         os.Exit(1)
3285     }
3286     e.LineStyle.Color = palette(set, false, coolingExperiment)
3287
3288     // Add set plots to p

```

```

3289     p.Add(e, t, r, plotFit) // , plotWid
3290
3291     // Legend
3292     l, err := plotter.NewScatter(pts)
3293     if err != nil {
3294         fmt.Println(err)
3295         os.Exit(1)
3296     }
3297
3298     l.GlyphStyle.Color = palette(set, true, coolingExperiment)
3299     l.GlyphStyle.Radius = vg.Points(6)
3300     l.Shape = draw.CircleGlyph{}
3301     power := strings.Trim(labels[set], " prs")
3302     if temp {
3303         temperature := strconv.FormatFloat(notes[set], 'f', -1, 64)
3304         p.Legend.Add(power + " @" + temperature + "K", l)
3305     } else {
3306         p.Legend.Add(power, l)
3307     }
3308 }
3309
3310 savePlot(p, "Stokes w Fits", logpath)
3311 }
3312
3313 func goPlotsPowerVsWid(
3314     sets []int,
3315     labels []string,
3316     notes []float64,
3317     temp, slide bool,
3318     sample, logpath, coolingExperiment string,
3319 ) {
3320
3321     title := "Stokes Pump Power vs Widths of Fits"
3322     xlabel := "Pump Power (mW)"
3323     ylabel := "Full Width Half Max (MHz)"
3324     legend := ""
3325
3326     xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
3327         "pow vs wid", sample, coolingExperiment,
3328     )
3329     if err != nil {
3330         fmt.Println(err)
3331         os.Exit(1)
3332     }
3333
3334     p, t, r := prepPlot(
3335         title, xlabel, ylabel, legend,
3336         xrange, yrange, xtick, ytick,
3337         xtickLabel, ytickLabel,
3338         slide,
3339     )
3340
3341     for i, set := range sets {
3342
3343         pts := make(plotter.XYs, 1)
3344
3345         power := strings.Trim(labels[set], " mW prs")
3346         if pwr, err := strconv.ParseFloat(power, 64); err == nil {
3347             pts[0].X = pwr
3348         } else {
3349             panic(err)
3350         }
3351         pts[0].Y = widths[i]
3352
3353         // Plot points
3354         plotPts, err := plotter.NewScatter(pts)
3355         if err != nil {
3356             fmt.Println(err)

```

```

3357     os.Exit(1)
3358 }
3359
3360 plotPts.GlyphStyle.Color = palette(set, true, coolingExperiment)
3361 plotPts.GlyphStyle.Radius = vg.Points(6)
3362 plotPts.Shape = draw.CircleGlyph{}
3363
3364 // Dashed eye guide lines
3365 v := make(plotter.XYs, 2)
3366 h := make(plotter.XYs, 2)
3367
3368 // Vertical
3369 v[0].X = pts[0].X
3370 v[0].Y = yrange[0]
3371 v[1].X = pts[0].X
3372 v[1].Y = pts[0].Y
3373
3374 vDash, err := plotter.NewLine(v)
3375 if err != nil {
3376     fmt.Println(err)
3377     os.Exit(1)
3378 }
3379
3380 vDash.LineStyle.Color = palette(set, true, coolingExperiment)
3381 vDash.LineStyle.Width = vg.Points(4)
3382 vDash.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
3383
3384 // Horizontal
3385 h[0].X = xrange[0]
3386 h[0].Y = pts[0].Y
3387 h[1].X = pts[0].X
3388 h[1].Y = pts[0].Y
3389
3390 hDash, err := plotter.NewLine(h)
3391 if err != nil {
3392     fmt.Println(err)
3393     os.Exit(1)
3394 }
3395
3396 hDash.LineStyle.Color = color.RGBA{R: 127, G: 127, B: 127, A: 255}
3397 hDash.LineStyle.Width = vg.Points(1)
3398 hDash.LineStyle.Dashes = []vg.Length{vg.Points(5), vg.Points(5)}
3399
3400 // Add set plots to p
3401 p.Add(plotPts, t, r, vDash, hDash)
3402 if temp {
3403     temperature := strconv.FormatFloat(notes[set], 'f', -1, 64)
3404     p.Legend.Add(power + " mW @" + temperature + "K", plotPts)
3405 } else {
3406     p.Legend.Add(power + " mW")
3407 }
3408 }
3409
3410 savePlot(p, "s Pow vs Wid", logpath)
3411 }
3412
3413 func goPlotHeightRatios(
3414     sets []int,
3415     heightRatios, powers []float64,
3416     labels []string,
3417     sample, logpath, coolingExperiment string,
3418     slide bool,
3419 ) {
3420
3421     title := " " // Height Ratios vs Power
3422     xlabel := "Pump Power (mW)"
3423     ylabel := "Stokes/Anti-Stokes Heights"
3424     legend := " "

```



```

3425
3426 xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
3427     "height ratios", sample, coolingExperiment,
3428 )
3429 if err != nil {
3430     fmt.Println(err)
3431     os.Exit(1)
3432 }
3433
3434 p, t, r := prepPlot(
3435     title, xlabel, ylabel, legend,
3436     xrange, yrange, xtick, ytick,
3437     xtickLabel, ytickLabel,
3438     slide,
3439 )
3440
3441 // Linear fit line
3442
3443 // Fit parameter guesses
3444 m := 5.
3445 b := .0125
3446
3447 f := func(dst, guess []float64) {
3448
3449     var x float64
3450     m, b := guess[0], guess[1]
3451
3452     for i, set := range sets {
3453
3454         x = powers[set]
3455         y := heightRatios[i]
3456
3457         dst[i] = m * x + b - y
3458     }
3459 }
3460
3461 jacobian := lm.NumJac{Func: f}
3462
3463 // Solve for fit
3464 toBeSolved := lm.LMProblem{
3465     Dim:      2,
3466     Size:     len(sets),
3467     Func:     f,
3468     Jac:      jacobian.Jac,
3469     InitParams: []float64{m, b},
3470     Tau:      1e-6,
3471     Eps1:     1e-8,
3472     Eps2:     1e-8,
3473 }
3474
3475 results, err := lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e-16})
3476 if err != nil {
3477     fmt.Println(err)
3478     os.Exit(1)
3479 }
3480
3481 m, b = results.X[0], results.X[1]
3482
3483 var yFit []float64
3484 var xFit []float64
3485
3486 // Create function according to solved fit parameters
3487 for _, set := range sets {
3488     var x float64
3489
3490     x = powers[set]
3491
3492     xFit = append(xFit, x)

```

```

3493     yFit = append(yFit, m * x + b)
3494 }
3495
3496 fit := buildData([][]float64{xFit, yFit})
3497
3498 // Plot fit
3499 plotFit, err := plotter.NewLine(fit)
3500 if err != nil {
3501     fmt.Println(err)
3502     os.Exit(1)
3503 }
3504
3505 p.Add(plotFit, t, r)
3506
3507 plotFit.LineStyle.Color = color.RGBA{R: 127, G: 127, B: 127, A: 255}
3508 plotFit.LineStyle.Width = vg.Points(3)
3509
3510 for i, set := range sets {
3511     pts := make(plotter.XYs, 1)
3512
3513     pts[0].X = powers[set]
3514     pts[0].Y = heightRatios[i]
3515
3516     // Plot points
3517     plotPts, err := plotter.NewScatter(pts)
3518     if err != nil {
3519         fmt.Println(err)
3520         os.Exit(1)
3521     }
3522
3523     plotPts.GlyphStyle.Color = color.RGBA{R: 27, G: 170, B: 139, A: 255}
3524     plotPts.GlyphStyle.Radius = vg.Points(6)
3525     plotPts.Shape = draw.CircleGlyph{}
3526
3527     // Add set plots to p
3528     p.Add(plotPts)
3529     //p.Legend.Add(strings.Trim(labels[set], " prs"), plotPts)
3530 }
3531
3532 savePlot(p, "height ratios", logpath)
3533 }
3534
3535 func Γeff(
3536     maxPow float64,
3537     Γ, length, gb float64,
3538     coolingExperiment string,
3539 ) (
3540     [][]float64, [][]float64,
3541 ) {
3542     // Γ_as,eff = 2*pi*Γ*(1 + GPL/4)
3543     // Γ_s,eff = 2*pi*Γ*(1 - GPL/4)
3544     var pow []float64
3545
3546     if coolingExperiment == "pump-only" {
3547         pow = []float64{0, maxPow}
3548     } else if coolingExperiment == "pump-probe" {
3549         pow = []float64{-10, maxPow}
3550     }
3551
3552     ΓasEff := []float64{Γ, Γ*(1 + gb*pow[1]*.001*length/(4*2*math.Pi))}
3553     ΓsEff := []float64{Γ, Γ*(1 - gb*pow[1]*.001*length/(4*2*math.Pi))}
3554
3555     fmt.Printf("\nΓasEff: %.4f\n", ΓasEff[1])
3556     fmt.Printf("ΓsEff: %.4f\n", ΓsEff[1])
3557
3558     return [][]float64{pow, ΓasEff}, [][]float64{pow, ΓsEff}
3559 }
3560 }

```

```

3561
3562 func goPlotLinewidths(
3563     sets []int,
3564     asEff, rsEff [][]float64,
3565     asLinewidths, sLinewidths, asPowers, sPowers []float64,
3566     labels []string,
3567     sample, logpath, coolingExperiment string,
3568     slide bool,
3569 ) {
3570
3571     title := " " // Linewidths vs Power
3572     xlabel := "Power (mW)"
3573     ylabel := "Dissipation Rate (MHz)"
3574     legend := ""
3575
3576     xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
3577         "linewidths", sample, coolingExperiment,
3578     )
3579     if err != nil {
3580         fmt.Println(err)
3581         os.Exit(1)
3582     }
3583
3584     p, t, r := prepPlot(
3585         title, xlabel, ylabel, legend,
3586         xrange, yrange, xtick, ytick,
3587         xtickLabel, ytickLabel,
3588         slide,
3589     )
3590
3591     p.Legend.Left = true
3592     p.Legend.XOffs = vg.Points(37.5)
3593     p.Legend.YOffs = vg.Points(12.5)
3594
3595     // as linear fit
3596     // linewidth fit parameter guesses
3597     m := 0.01
3598     b := 87.5
3599
3600     f := func(dst, guess []float64) {
3601
3602         var x float64
3603         m, b := guess[0], guess[1]
3604
3605         for i, set := range sets {
3606
3607             x = asPowers[set]
3608             y := asLinewidths[i]
3609
3610             dst[i] = m * x + b - y
3611         }
3612     }
3613
3614     jacobian := lm.NumJac{Func: f}
3615
3616     // Solve for fit
3617     toBeSolved := lm.LMProblem{
3618         Dim:      2,
3619         Size:      len(sets),
3620         Func:      f,
3621         Jac:      jacobian.Jac,
3622         InitParams: []float64{m, b},
3623         Tau:      1e-6,
3624         Eps1:     1e-8,
3625         Eps2:     1e-8,
3626     }
3627
3628     results, err := lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e-16})

```

```

3629 if err != nil {
3630     fmt.Println(err)
3631     os.Exit(1)
3632 }
3633
3634 m, b = results.X[0], results.X[1]
3635
3636 var asyFit []float64
3637 var asxFit []float64
3638
3639 // Create function according to solved fit parameters
3640 for _, set := range sets {
3641     var x float64
3642
3643     x = asPowers[set]
3644
3645     asxFit = append(asxFit, x)
3646     asyFit = append(asyFit, m * x + b)
3647 }
3648
3649 asfit := buildData([][]float64{asxFit, asyFit})
3650
3651 // Plot as fit
3652 asPlotFit, err := plotter.NewLine(asfit)
3653 if err != nil {
3654     fmt.Println(err)
3655     os.Exit(1)
3656 }
3657
3658 // s linear fit
3659
3660 f = func(dst, guess []float64) {
3661
3662     var x float64
3663     m, b := guess[0], guess[1]
3664
3665     for i, set := range sets {
3666
3667         x = sPowers[set]
3668         y := sLinewidthths[i]
3669
3670         dst[i] = m * x + b - y
3671     }
3672 }
3673
3674 jacobian = lm.NumJac{Func: f}
3675
3676 // Solve for fit
3677 toBeSolved = lm.LMProblem{
3678     Dim:      2,
3679     Size:     len(sets),
3680     Func:     f,
3681     Jac:      jacobian.Jac,
3682     InitParams: []float64{m, b},
3683     Tau:      1e-6,
3684     Eps1:     1e-8,
3685     Eps2:     1e-8,
3686 }
3687
3688 results, err = lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e-16})
3689 if err != nil {
3690     fmt.Println(err)
3691     os.Exit(1)
3692 }
3693
3694 m, b = results.X[0], results.X[1]
3695
3696 var syFit []float64

```

```

3697 var sxFit []float64
3698
3699 // Create function according to solved fit parameters
3700 for _, set := range sets {
3701     var x float64
3702
3703     x = sPowers[set]
3704
3705     sxFit = append(sxFit, x)
3706     syFit = append(syFit, m * x + b)
3707 }
3708
3709 sfit := buildData([][]float64{sxFit, syFit})
3710
3711 // Plot s fit
3712 sPlotFit, err := plotter.NewLine(sfit)
3713 if err != nil {
3714     fmt.Println(err)
3715     os.Exit(1)
3716 }
3717
3718 sPlotFit.LineStyle.Color = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3719 sPlotFit.LineStyle.Width = vg.Points(3)
3720
3721 asPlotFit.LineStyle.Color = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3722 asPlotFit.LineStyle.Width = vg.Points(3)
3723
3724 // Plot Γeff
3725 ΓasEffPlot, err := plotter.NewLine(buildData(ΓasEff))
3726 if err != nil {
3727     fmt.Println(err)
3728     os.Exit(1)
3729 }
3730
3731 ΓasEffPlot.LineStyle.Color = color.NRGBA{R: 99, G: 124, B: 198, A: 125} // R: 0, G: 89, B:
3732 128, A: 255
3733 ΓasEffPlot.LineStyle.Width = vg.Points(3)
3734 ΓasEffPlot.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
3735
3736 ΓsEffPlot, err := plotter.NewLine(buildData(ΓsEff))
3737 if err != nil {
3738     fmt.Println(err)
3739     os.Exit(1)
3740 }
3741
3742 ΓsEffPlot.LineStyle.Color = color.NRGBA{R: 201, G: 104, B: 146, A: 125} // R: 0, G: 89, B:
3743 128, A: 255
3744 ΓsEffPlot.LineStyle.Width = vg.Points(3)
3745 ΓsEffPlot.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}
3746
3747 p.Add(asPlotFit, t, r, sPlotFit, ΓasEffPlot, ΓsEffPlot)
3748 p.Legend.Add("Anti-Stokes Fit", asPlotFit)
3749 p.Legend.Add("Γ as,eff", ΓasEffPlot)
3750 p.Legend.Add("Stokes Fit", sPlotFit)
3751 p.Legend.Add("Γ s,eff", ΓsEffPlot)
3752
3753 // as points
3754 for i, set := range sets {
3755     pts := make(plotter.XYs, 1)
3756
3757     pts[0].X = asPowers[set]
3758     pts[0].Y = asLinewidthths[i]
3759
3760 // Plot points
3761 asPlotPts, err := plotter.NewScatter(pts)
3762 if err != nil {
3763     fmt.Println(err)

```

```

3763     os.Exit(1)
3764 }
3765
3766 asPlotPts.GlyphStyle.Color = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3767 asPlotPts.GlyphStyle.Radius = vg.Points(6)
3768 asPlotPts.Shape = draw.CircleGlyph{}
3769
3770 // Add set plots to p
3771 p.Add(asPlotPts)
3772 }
3773
3774 // s points
3775 for i, set := range sets {
3776
3777     pts := make(plotter.XYs, 1)
3778
3779     pts[0].X = sPowers[set]
3780     pts[0].Y = sLinewidths[i]
3781
3782     // Plot points
3783     sPlotPts, err := plotter.NewScatter(pts)
3784     if err != nil {
3785         fmt.Println(err)
3786         os.Exit(1)
3787     }
3788
3789     sPlotPts.GlyphStyle.Color = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3790     sPlotPts.GlyphStyle.Radius = vg.Points(6)
3791     sPlotPts.Shape = draw.CircleGlyph{}
3792
3793     // Add set plots to p
3794     p.Add(sPlotPts)
3795 }
3796
3797 savePlot(p, "linewidths", logpath)
3798 }
3799
3800 func prepPlot(
3801     title, xlabel, ylabel, legend string,
3802     xrange, yrange, xtick, ytick []float64,
3803     xtickLabels, ytickLabels []string,
3804     slide bool,
3805 ) (
3806     *plot.Plot,
3807     *plotter.Line, *plotter.Line,
3808 ) {
3809
3810     p := plot.New()
3811     p.BackgroundColor = color.RGBA{A:0}
3812     p.Title.Text = title
3813     p.Title.TextStyle.Font.Typeface = "liberation"
3814     p.Title.TextStyle.Font.Variant = "Sans"
3815
3816     p.X.Label.Text = xlabel
3817     p.X.Label.TextStyle.Font.Variant = "Sans"
3818     p.X.LineStyle.Width = vg.Points(1.5)
3819     p.X.Min = xrange[0]
3820     p.X.Max = xrange[1]
3821     p.X.Tick.LineStyle.Width = vg.Points(1.5)
3822     p.X.Tick.Label.Font.Variant = "Sans"
3823
3824     xticks := []plot.Tick{}
3825     for i, v := range xtick {
3826         xticks = append(xticks, plot.Tick{Value: v, Label: xtickLabels[i]})
3827     }
3828
3829     p.X.Tick.Marker = plot.ConstantTicks(xticks)
3830     p.X.Padding = vg.Points(-8) // -12.5

```

```

3831
3832 p.Y.Label.Text = ylabel
3833 p.Y.Label.TextStyle.Font.Variant = "Sans"
3834 p.Y.LineStyle.Width = vg.Points(1.5)
3835 p.Y.Min = yrange[0]
3836 p.Y.Max = yrange[1]
3837 p.Y.Tick.LineStyle.Width = vg.Points(1.5)
3838 p.Y.Tick.Label.Font.Variant = "Sans"
3839
3840 yticks := []plot.Tick{}
3841 for i, v := range ytick {
3842     yticks = append(yticks, plot.Tick{Value: v, Label: ytickLabels[i]})
3843 }
3844
3845 p.Y.Tick.Marker = plot.ConstantTicks(yticks)
3846 p.Y.Padding = vg.Points(-6) // -0.5
3847
3848 p.Legend.TextStyle.Font.Variant = "Sans"
3849 p.Legend.Top = true
3850 p.Legend.XOffs = vg.Points(-25)
3851 p.Legend.YOffs = vg.Points(25)
3852 p.Legend.Padding = vg.Points(10)
3853 p.Legend.ThumbnailWidth = vg.Points(50)
3854 p.Legend.Add(legend)
3855
3856 if slide {
3857     p.Title.TextStyle.Font.Size = 80
3858     p.Title.Padding = font.Length(80)
3859
3860     p.X.Label.TextStyle.Font.Size = 56
3861     p.X.Label.Padding = font.Length(40)
3862
3863     p.X.Tick.Label.Font.Size = 56
3864
3865     p.Y.Label.TextStyle.Font.Size = 56
3866     p.Y.Label.Padding = font.Length(40)
3867
3868     p.Y.Tick.Label.Font.Size = 56
3869
3870     p.Legend.TextStyle.Font.Size = 56
3871 } else {
3872     p.Title.TextStyle.Font.Size = 50
3873     p.Title.Padding = font.Length(50)
3874
3875     p.X.Label.TextStyle.Font.Size = 36
3876     p.X.Label.Padding = font.Length(20)
3877
3878     p.X.Tick.Label.Font.Size = 36
3879
3880     p.Y.Label.TextStyle.Font.Size = 36
3881     p.Y.Label.Padding = font.Length(20)
3882
3883     p.Y.Tick.Label.Font.Size = 36
3884
3885     p.Legend.TextStyle.Font.Size = 28
3886 }
3887
3888 // Enclose plot
3889 t := make(plotter.XYs, 2)
3890 r := make(plotter.XYs, 2)
3891
3892 // Top
3893 t[0].X = xrange[0]
3894 t[0].Y = yrange[1]
3895 t[1].X = xrange[1]
3896 t[1].Y = yrange[1]
3897
3898 tAxis, err := plotter.NewLine(t)

```

```

3899     if err != nil {
3900         fmt.Println(err)
3901         os.Exit(1)
3902     }
3903
3904     // Right
3905     r[0].X = xrange[1]
3906     r[0].Y = yrange[0]
3907     r[1].X = xrange[1]
3908     r[1].Y = yrange[1]
3909
3910     rAxis, err := plotter.NewLine(r)
3911     if err != nil {
3912         fmt.Println(err)
3913         os.Exit(1)
3914     }
3915
3916     return p, tAxis, rAxis
3917 }
3918
3919 func palette(
3920     brush int,
3921     dark bool,
3922     coolingExperiment string,
3923 ) (
3924     color.RGBA,
3925 ) {
3926
3927     if coolingExperiment == "pump-probe" {
3928         if dark {
3929             darkColor := make([]color.RGBA, 16)
3930             darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
3931             darkColor[1] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3932             darkColor[2] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3933             darkColor[3] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
3934             darkColor[4] = color.RGBA{R: 7, G: 150, B: 189, A: 255}
3935             darkColor[5] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3936             darkColor[6] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3937             darkColor[7] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
3938             darkColor[8] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
3939             darkColor[9] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3940             darkColor[10] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3941             darkColor[11] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
3942             darkColor[12] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
3943             darkColor[13] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3944             darkColor[14] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3945             darkColor[15] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
3946
3947             return darkColor[brush]
3948         }
3949
3950         col := make([]color.RGBA, 16)
3951         col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
3952         col[1] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
3953         col[2] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
3954         col[3] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
3955         col[4] = color.RGBA{R: 11, G: 191, B: 222, A: 255}
3956         col[5] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
3957         col[6] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
3958         col[7] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
3959         col[8] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
3960         col[9] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
3961         col[10] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
3962         col[11] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
3963         col[12] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
3964         col[13] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
3965         col[14] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
3966         col[15] = color.RGBA{R: 255, G: 182, B: 110, A: 255}

```



```

3967     return col[brush]
3968
3969
3970 } else if coolingExperiment == "pump-only" {
3971     if dark {
3972         darkColor := make([]color.RGBA, 21)
3973         darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
3974         darkColor[4] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
3975         darkColor[8] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
3976         darkColor[12] = color.RGBA{R: 183, G: 139, B: 89, A: 255}
3977         darkColor[15] = color.RGBA{R: 18, G: 102, B: 99, A: 255}
3978         darkColor[1] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
3979         darkColor[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
3980         darkColor[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
3981         darkColor[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
3982         darkColor[2] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
3983         darkColor[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
3984         darkColor[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
3985         darkColor[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
3986         darkColor[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
3987         darkColor[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
3988         darkColor[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
3989         darkColor[16] = color.RGBA{R: 36, G: 117, B: 100, A: 255}
3990         darkColor[17] = color.RGBA{R: 117, G: 85, B: 41, A: 255}
3991         darkColor[18] = color.RGBA{R: 86, G: 17, B: 22, A: 255}
3992         darkColor[19] = color.RGBA{R: 17, G: 39, B: 91, A: 255}
3993         darkColor[20] = color.RGBA{R: 54, G: 12, B: 66, A: 255}
3994
3995         return darkColor[brush]
3996     }
3997
3998     col := make([]color.RGBA, 21)
3999     col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
4000     col[4] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
4001     col[8] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
4002     col[12] = color.RGBA{R: 255, G: 193, B: 122, A: 255}
4003     col[15] = color.RGBA{R: 27, G: 150, B: 146, A: 255}
4004     col[1] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
4005     col[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
4006     col[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
4007     col[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
4008     col[2] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
4009     col[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
4010     col[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
4011     col[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
4012     col[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
4013     col[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
4014     col[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
4015     col[16] = color.RGBA{R: 36, G: 117, B: 100, A: 255}
4016     col[17] = color.RGBA{R: 117, G: 85, B: 41, A: 255}
4017     col[18] = color.RGBA{R: 86, G: 17, B: 22, A: 255}
4018     col[19] = color.RGBA{R: 17, G: 39, B: 91, A: 255}
4019     col[20] = color.RGBA{R: 54, G: 12, B: 66, A: 255}
4020
4021     return col[brush]
4022 }
4023
4024 if dark {
4025     darkColor := make([]color.RGBA, 17)
4026     darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
4027     darkColor[1] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
4028     darkColor[2] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
4029     darkColor[12] = color.RGBA{R: 183, G: 139, B: 89, A: 255}
4030     darkColor[15] = color.RGBA{R: 18, G: 102, B: 99, A: 255}
4031     darkColor[4] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
4032     darkColor[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
4033     darkColor[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
4034     darkColor[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}

```

```

4035     darkColor[8] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
4036     darkColor[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
4037     darkColor[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
4038     darkColor[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
4039     darkColor[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
4040     darkColor[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
4041     darkColor[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
4042     darkColor[16] = color.RGBA{R: 255, G: 102, B: 102, A: 255}
4043
4044     return darkColor[brush % len(darkColor)]
4045 }
4046
4047 col := make([]color.RGBA, 17)
4048 col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
4049 col[1] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
4050 col[2] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
4051 col[12] = color.RGBA{R: 255, G: 193, B: 122, A: 255}
4052 col[15] = color.RGBA{R: 27, G: 150, B: 146, A: 255}
4053 col[4] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
4054 col[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
4055 col[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
4056 col[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
4057 col[8] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
4058 col[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
4059 col[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
4060 col[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
4061 col[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
4062 col[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
4063 col[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
4064 col[16] = color.RGBA{R: 255, G: 102, B: 102, A: 255}
4065
4066     return col[brush % len(col)]
4067 }
4068
4069 func savePlot(
4070     p *plot.Plot,
4071     name, logpath string,
4072 ) {
4073
4074     date := time.Now()
4075
4076     // Make current date folder if it doesn't already exist
4077     if _, err := os.Stat("plots/" + date.Format("2006-Jan-02")); os.IsNotExist(err) {
4078         if err := os.Mkdir("plots/" + date.Format("2006-Jan-02"), 0755); err != nil {
4079             fmt.Println(err)
4080             os.Exit(1)
4081         }
4082     }
4083
4084     // Make current time folder if it doesn't already exist
4085     if _, err := os.Stat(logpath); os.IsNotExist(err) {
4086         if err := os.Mkdir(logpath, 0755); err != nil {
4087             fmt.Println(err)
4088             fmt.Println(logpath)
4089             os.Exit(1)
4090         }
4091     }
4092
4093     path := logpath + "/" + name
4094
4095     if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".png"); err != nil {
4096         fmt.Println(err)
4097         os.Exit(1)
4098     }
4099
4100     if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".svg"); err != nil {
4101         fmt.Println(err)
4102         os.Exit(1)

```

```

4103 }
4104
4105 if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".pdf"); err != nil {
4106     fmt.Println(err)
4107     os.Exit(1)
4108 }
4109 }
4110
4111 /*func pngsToGIF(
4112     pngPaths []string,
4113     gifPath string,
4114 ) (
4115     error,
4116 ) {
4117
4118     var frames []*image.Paletted
4119     var delays []int
4120
4121     for _, fname := range pngPaths {
4122         // Open the PNG file
4123         f, err := os.Open(fname)
4124         if err != nil {
4125             return err
4126         }
4127
4128         // Decode the PNG
4129         img, err := png.Decode(f)
4130         if err != nil {
4131             return err
4132         }
4133         f.Close()
4134
4135         // Convert the image to Paletted
4136         palettedImage := image.NewPaletted(img.Bounds(), ipalette.Plan9)
4137         idraw.Draw(palettedImage, img.Bounds(), img, image.Point{}, idraw.Over)
4138
4139         frames = append(frames, palettedImage)
4140         delays = append(delays, 10) // Add a delay for this frame
4141     }
4142
4143     // Save as a GIF
4144     outFile, err := os.Create(gifPath)
4145     if err != nil {
4146         return err
4147     }
4148     defer outFile.Close()
4149
4150     return gif.EncodeAll(outFile, &gif.GIF{
4151         Image: frames,
4152         Delay: delays,
4153     })
4154 }*/
4155
4156 func normalizeFit(
4157     fit []float64,
4158 ) (
4159     []float64,
4160 ) {
4161
4162     var shift float64 = (fit[0] + fit[599])/2
4163
4164     for i := range fit {
4165         fit[i] = fit[i] - shift
4166     }
4167     return fit
4168 }
4169
4170 func writeLog(

```

```

4171 logpath string,
4172 logFile []string,
4173 ) {
4174
4175     date := time.Now()
4176
4177     // Make current date folder if it doesn't already exist
4178     if _, err := os.Stat("plots/" + date.Format("2006-Jan-02")); os.IsNotExist(err) {
4179         if err := os.Mkdir("plots/" + date.Format("2006-Jan-02"), 0755); err != nil {
4180             fmt.Println(err)
4181             os.Exit(1)
4182         }
4183     }
4184
4185     // Make current time folder if it doesn't already exist
4186     if _, err := os.Stat(logpath); os.IsNotExist(err) {
4187         if err := os.Mkdir(logpath, 0755); err != nil {
4188             fmt.Println(err)
4189             os.Exit(1)
4190         }
4191     }
4192
4193     txt, err := os.Create(logpath + "/log.txt")
4194     if err != nil {
4195         fmt.Println(err)
4196         os.Exit(1)
4197     }
4198
4199     w := bufio.NewWriter(txt)
4200     defer w.Flush()
4201     for _, line := range logFile {
4202         if _, err := w.WriteString(line); err != nil {
4203             fmt.Println(err)
4204             os.Exit(1)
4205         }
4206     }
4207 }

```

Appendix C

Supplementary Information for Chapter 3: Manuscript I

References

Boyd, R. W. 2020, Nonlinear optics (Academic press)