!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 Montaño, Ph.D.

Jennifer S. Martinez, Ph.D.

Table of Contents

List of Tables

List of Figures

Dedication

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

Chapter 4

Manuscript II: A coherently stimulated phonon spectrometer

Joel N. Johnson[1,2], Nils T. Otterstrom[3], Peter T. Rakich[4], 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

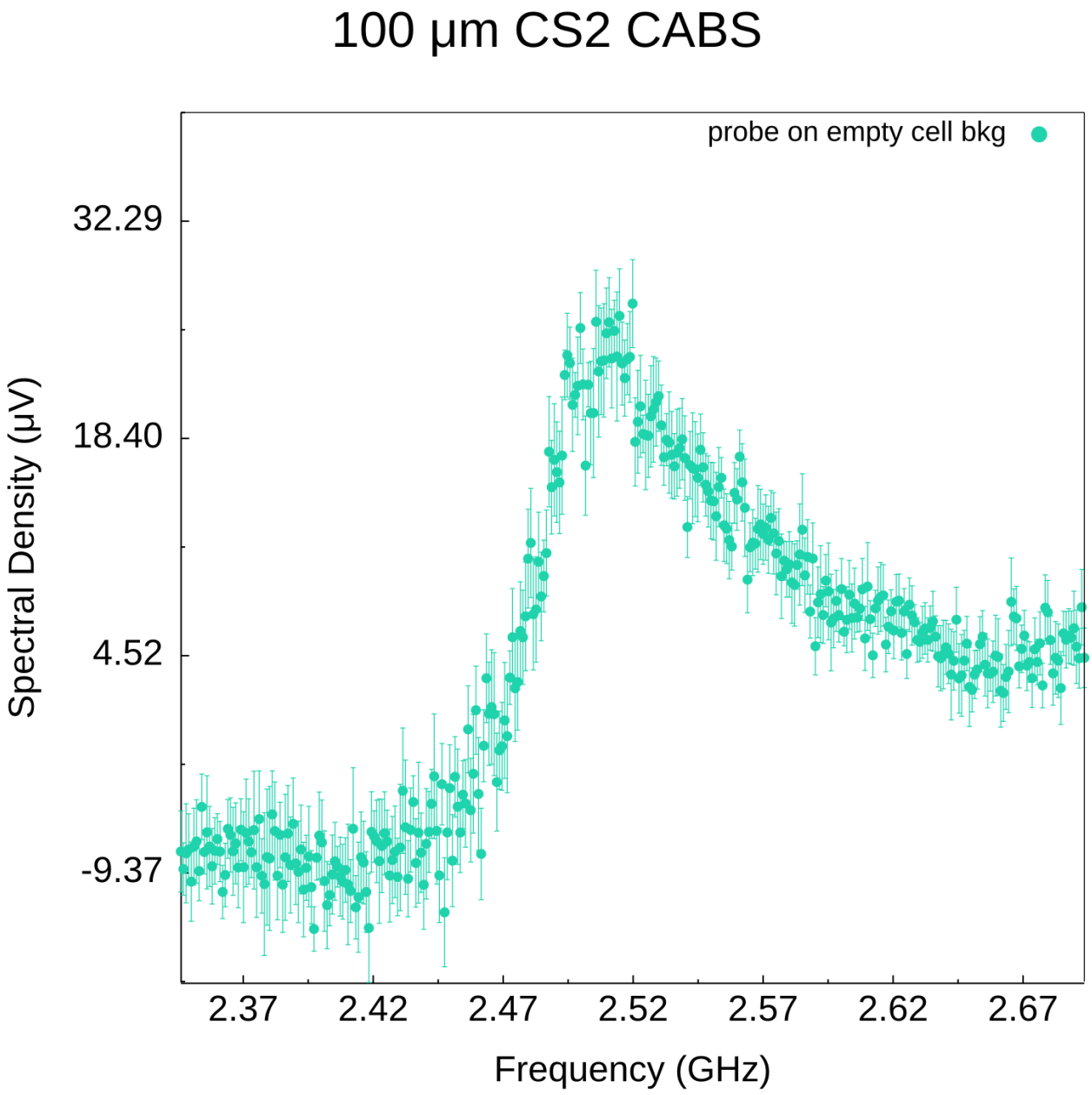

Figure 4.1: CABS measurement of 100um of CS2.

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 ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	SphUp1/SphLo1	235.98988249	
	SphUp2/SphLo2	-29.59742178	
	SphUp3/SphLo3	-3.78707193	
	ConUp0	0.00133644	Upper layer conductivity expressed as 4th-order polynomial ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	ConUp1	0.00073150	
	ConUp2	0.00033250	
	ConUp3	0.00005038	
	ConLo0	0.00634807	Lower layer conductivity expressed as 4th-order polynomial ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	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 ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	SphUp1/SphLo1	235.98988249	
	SphUp2/SphLo2	-29.59742178	
	SphUp3/SphLo3	-3.78707193	
	ConUp0	0.00133644	Upper layer conductivity expressed as 4th-order polynomial ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	ConUp1	0.00073150	
	ConUp2	0.00033250	
	ConUp3	0.00005038	
	ConLo0	0.00634807	Lower layer conductivity expressed as 4th-order polynomial ($c0 + c1 \cdot T + c2 \cdot T^2 + c3 \cdot T^3$)
	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

```
import csv
import visa
import time
import datetime
import numpy as np
import matplotlib.pyplot as plt
import zhinst.ziPython, zhinst.utils
import os
import winsound

daq = zhinst.ziPython.ziDAQServer('localhost', 8005)
device = zhinst.utils.autoDetect(daq)

startTime = datetime.datetime.now()

dwell = .1
F_lockin = 45.0000E6
F_AOM = 40.000800e6
chan = 1
ch   = str(chan-1)
rate=2e9 # rate, per millisecond
RBW = 100 # lock-in bandwidth
tc=1/(2*np.pi*RBW) # time constant
lockRange = .04
order = 8

exp_set = [
       [['/', device, '/sigins/',ch,'/diff'], 0],
       [['/', device, '/sigins/',ch,'/imp50'], 1],
       [['/', device, '/sigins/',ch,'/ac'], 1],

       # Want range as low as possible without clipping data
       # (red Over light on Lock-in box)
       [['/', device, '/sigins/',ch,'/range'], lockRange],
       [['/', device, '/demods/',ch,'/order'], order],
       [['/', device, '/demods/',ch,'/timeconstant'], tc/3.33],
       [['/', device, '/demods/',ch,'/rate'], rate],
       [['/', device, '/demods/',ch,'/adcselect'], chan-1],
       [['/', device, '/demods/',ch,'/oscselect'], chan-1],
       [['/', device, '/demods/',ch,'/harmonic'], 1],
       [['/', device, '/oscs/',ch,'/freq'], F_lockin],
   ]

daq.set(exp_set);
time.sleep(.001)

path = '/%s/demods/%d/sample' % (device, 0) # (device, demod_index)
daq.subscribe(path)
daq.flush()
```

```
daq.sync()

def measureSynchronousFeedback(daq, device, channel, frequency):
   c=str(channel-1) #return a string of an object. =channel-1=-1 for channel 0

# Poll the subscribed data from the data server. Poll will block and record
# for poll_length seconds.
    daq.sync()
    daq.flush()
    poll_length = dwell  # [s]
    poll_timeout = 10  # [ms]
    poll_flags = 0
    poll_return_flat_dict = True
    data = daq.poll(
        poll_length,
        poll_timeout,
        poll_flags,
        poll_return_flat_dict
    )
    assert data, """poll() returned an empty data dictionary,
                    did you subscribe to any paths?"""

    # Access the demodulator sample using the node's path.
    sample = data[path] # Defines the sample as the data from defined path

    # Calculate the demodulator's magnitude and adds it to the dict.
    global sampleR
    sampleR = np.abs(sample['x'] + 1j*sample['y']) #y-axis #magnitude

    clockbase = float(daq.getInt('/%s/clockbase' % device))

    # Convert timestamps from ticks to seconds via clockbase.
    t = (sample['timestamp'] - sample['timestamp'][0])/clockbase

resources = visa.ResourceManager()
resources.list_resources()

#open the signal generator
#gen=resources.open_resource('USB0::0x03EB::0xAFFF::6C2-0A2A2000A-0374::INSTR')
gen=resources.open_resource('USB0::0x03EB::0xAFFF::6C2-0A2B2000A-0430::INSTR')

gen.write('FREQUENCY:MODE CW DUAL')
time.sleep(0.01)
gen.write('OUTPUT1 1')
time.sleep(0.01)
gen.write('OUTPUT2 1')

# run a cycle of frequencies for each output
Fstart = 9.0*1E9
Fstop  = 9.28*1E9
F_step = 0.00500456*1E9

N_steps = np.int((Fstop-Fstart)/F_step)

NoRealizations = 5
dataR = [0]*N_steps # demod data
dataF = [0]*N_steps # Freq. axis
stdDevOfMeanR = [0]*N_steps # to hold standard deviation of dwell-time data
f2 = Fstart
F = Fstart


run = 1
folderName = startTime.strftime("%y-%m-%d ") + """1cm UHNA3"""
while os.path.exists(folderName + "/" + str(run) + "/signal.csv"):
    run += 1
folderRunName = folderName + "/" + str(run)

```

```
signalData = 1
takeBkrdData = 0

aveDataR = [0]*N_steps

if not os.path.exists(folderRunName):
    os.makedirs(folderRunName)

if not os.path.exists(folderName + "/meta.csv"):
    meta = open(folderName + "/meta.csv", "a")
    meta.write("""Date,Label,Sets,Start Time,End Time (hr:min:sec),Pump,Stokes,
                Probe,Frequency,Signal,Range,Dwell,Bandwidth,Data Rate,Order,
                Start Frequency,Stop Frequency,Step,Num Avgs,Pump Laser,Probe
                Laser,Probe Filter,Stokes Filter,Notes\n""")
    meta.close()

if takeBkrdData == 1:
    bkrdStartTime = datetime.datetime.now()

    bgDataR = np.zeros((NoRealizations, N_steps))

for kk in range(0,NoRealizations):
    F = Fstart
    print(kk)
    for jj in range(0,N_steps):
        f1 = F
        f2 = F + F_AOM - F_lockin
        time.sleep(0.1)
        gen.write("SOURce1: FREQuency:CW "+ str(f1)) # Hz
        time.sleep(1e-3)
        gen.write("SOURce2:FREQuency:CW "+ str(f2)) # Hz
        time.sleep(1e-3)
        measureSynchronousFeedback(daq, device, 1, F_lockin)
        time.sleep(1e-3)

        F=F+F_step

        dataF[jj] = Fstart+jj*F_step
        ff1 = f1*10**-6
        ff2 = f2*10**-6

        dataR[jj]=np.mean(sampleR)
        stdDevOfMeanR[jj] = np.std(sampleR)/np.sqrt(len(sampleR))

    #-----------------save run data (dataR and stdDevR)-----------------------#
    if signalData == 1:
        runSigDir = folderRunName + "/Runs/Signal/"
        if not os.path.exists(runSigDir):
            os.makedirs(runSigDir)

        csvfile = runSigDir + "Run " + str(kk) + ".csv"
        with open(csvfile, "w") as output:
            writer = csv.writer(output, delimiter=',')
            writer.writerow(['Sig','Std Dev'])
            for sig, std in zip(dataR, stdDevOfMeanR):
                writer.writerow([sig, std])

    elif takeBkrdData == 1:
        runBgDir = folderRunName + "/Runs/Background/"
        if not os.path.exists(runBgDir):
            os.makedirs(runBgDir)

        csvfile = runBgDir + "Run " + str(kk) + ".csv"
        with open(csvfile, "w") as output:
            writer = csv.writer(output, delimiter=',')
            writer.writerow(['Sig','Std Dev'])
            for sig, std in zip(dataR, stdDevOfMeanR):
                writer.writerow([sig, std])
```

```
    #-------------------------------------------------------------------------#

    aveDataR = (np.array(dataR)+kk*np.array(aveDataR))/(kk+1)

    if takeBkrdData == 1:

        bgDataR[kk] = dataR
        background = aveDataR

        plt.figure()
        plt.grid(True)
        plt.plot(dataF, background)
        plt.title('Background Demodulator data')
        plt.show()

    if takeBkrdData == 0:
        plt.figure()
        plt.grid(True)
        plt.plot(dataF, aveDataR-background)
        plt.title('Signal - Background Demodulator data')
        plt.xlabel('F')
        plt.ylabel('R')
        plt.show()

    #-----------------save subtracted run data (dataR)-------------------------#
    if signalData == 1:
        runSubtrDir = folderRunName + "/Runs/Subtracted/"
        if not os.path.exists(runSubtrDir):
            os.makedirs(runSubtrDir)

        csvfile = runSubtrDir + "Run " + str(kk) + ".csv"
        with open(csvfile, "w") as output:
            writer = csv.writer(output, lineterminator='\n')
            for val in dataR - bgDataR[kk]:
                writer.writerow([val])
    #-------------------------------------------------------------------------#

if signalData == 1:
    csvfile=folderRunName+"/signal.csv"
    with open(csvfile, "w") as output:
        writer=csv.writer(output, lineterminator='\n')
        for val in aveDataR-background:
            writer.writerow([val])

    csvfile=folderRunName+"/frequency.csv"
    with open(csvfile, "w") as output:
        writer=csv.writer(output, lineterminator='\n')
        for val in dataF:
            writer.writerow([val])

if not os.path.exists(folderRunName + "/timestamp.csv"):
    timestampf = open(folderRunName + "/timestamp.csv", "a")
    timestampf.write("""Date,Label,Run,Data,Start Time,End Time (hr:min:sec),
                    Pump,Stokes,Probe,Frequency,Signal,Range,Dwell,Bandwidth,
                    Data Rate,Order,Start Frequency, Stop Frequency,Step,Num
                    Avgs,Notes\n""")
    timestampf.close()

if takeBkrdData == 1:
    bkrdEndTime = datetime.datetime.now()
    timestampf = open(folderRunName + "/timestamp.csv", "a")
    timestampf.write(
        bkrdStartTime.strftime("%y-%m-%d,,") + str(run) + ",Background" +
        bkrdStartTime.strftime(",%H:%M:%S") +
        bkrdEndTime.strftime(",%H:%M:%S,,,,,,")
    )
    timestampf.write(
        str(lockRange) + "," + str(dwell) + "," + str(RBW) + "," +
```

```
        str("{:e}".format(rate)) + "," + str(order) + "," +
        str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
        str("{:e}".format(F_step)) + "," + str(NoRealizations) +",\n"
    )
    timestampf.close()

if signalData == 1:
    endTime = datetime.datetime.now()
    timestampf = open(folderRunName + "/timestamp.csv", "a")
    timestampf.write(
        startTime.strftime("%y-%m-%d,,") + str(run) + ",Signal" +
        startTime.strftime(",%H:%M:%S") + endTime.strftime(",%H:%M:%S,,,,,")
    )
    timestampf.write(str(run) + "/frequency.csv," + str(run) + "/signal.csv,")
    timestampf.write(
        str(lockRange) + "," + str(dwell) + "," + str(RBW) + "," +
        str("{:e}".format(rate)) + "," + str(order) + "," +
        str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
        str("{:e}".format(F_step)) +  "," + str(NoRealizations) +",\n"
    )
    timestampf.close()

    meta = open(folderName + "/meta.csv", "a")
    meta.write(
        startTime.strftime("%y-%m-%d,,") + str(run) +
        startTime.strftime(",%H:%M:%S") + endTime.strftime(",%H:%M:%S,,,,")
    )
    meta.write(str(run) + "/frequency.csv," + str(run) + "/signal.csv,")
    meta.write(
        str(lockRange) + "," + str(dwell) + "," + str(RBW) + "," +
        str("{:e}".format(rate)) + "," + str(order) + "," +
        str("{:e}".format(Fstart)) + "," + str("{:e}".format(Fstop)) + "," +
        str("{:e}".format(F_step)) +  "," + str(NoRealizations) +",,,,,\n"
    )
    meta.close()
```

B.2 Plotting Data In Go Program

```
package main

import (
  "image/color"
  "github.com/Arafatk/glot"
  "github.com/maorshutman/lm"
  "encoding/csv"
  "bufio"
  "fmt"
  "os"
  "io"
  "strconv"
  "strings"
  "math"
  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/vg"
  "gonum.org/v1/plot/font"
  "gonum.org/v1/plot/vg/draw"
  "time"
  "flag"
  "log"
)

func main() {

  cabs, lock, temp, slide, sinc, sample, coolingExperiment, note, length, csvToAvg := flags
    ()

  logpath := logpath(note)

  date, label, setNums, startTime, endTime, asPowers, sPowers,
  pumpPowers, stokesPowers, probePowers, filepath, sigFilepath, freqFilepath,
  lockinRange, dwell, bandwidth, dataRate, order, startFrequency, stopFrequency,
  step, numAvgs, asNotes, sNotes, pumpLaser, probeLaser, probeFilter, stokesFilter,
  notes := readMeta(
    cabs, lock, temp, coolingExperiment,
  )

  logFile := logHeader(
    cabs, lock, temp, slide, sample, coolingExperiment, note,
    length,
  )

  if coolingExperiment != "" {

    σas, σs := avgCSVs(csvToAvg, asPowers)

    ras, bas, rs, bs := getCoolingData(lock, filepath, label)

    asLabel, basLabel, sLabel, bsLabel := getAllLabels(label)

    setsToPlotRaw := []int{}
    plotRaw(
      setsToPlotRaw,
      bas, ras, bs, rs,
      basLabel, asLabel, bsLabel, sLabel,
    )

    s, as := subtractBackground(ras, bas, rs, bs, coolingExperiment)

    setsToPlotSubtracted := []int{}
    plotSubtracted(
      setsToPlotSubtracted,
      s, as,
```

```
    sLabel, asLabel,
  )

  setsToPlotSubtractedTogether := []int{}
  plotSubtractedTogether(
    setsToPlotSubtractedTogether,
    as, s,
    asLabel, sLabel,
  )

  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
  if len(binSets) > 0 {
    binMHz := 5.
    as, s = bin(binSets, as, s, binMHz)
  }

  subtractedGrouped := []int{}
  if len(subtractedGrouped) > 0 {
    goPlotSubGrpd(
      subtractedGrouped, s, as, σs, σas, sLabel, asLabel, logpath, sample,
      coolingExperiment, slide,
    )
  }

  fitSets := true
  if fitSets {

    var amp, wid, cen, c, gb, Γ float64

    if sample == "LCOF" {
      amp = 2.
      wid = 0.1
      cen = 2.275
      c = .01
      gb = 6 // W^{-1}m^{-1}
      Γ = 98.65 //*2*math.Pi // MHz
    } else if sample == "UHNA3" {
      amp = 12
      wid = 0.1
      cen = 9.18
      gb = 0.6
      Γ = 100
    } else {
      sample = "[Unspecified] Sample"
      amp = 5
      wid = 0.1
      cen = 2.25
      gb = 0
      Γ = 0
    }

    var asAmps, asLinewidths []float64

    fitAntiStokes := []int{0,1,2,3}
    if len(fitAntiStokes) > 0 {

      // as
      header := fmt.Sprintf("\nAnti-Stokes\nSet  \t Power \t\t Width \t\t Peak \t\t Center
\n")
      fmt.Printf(header)
      logFile = append(logFile, header)

      var asFits [][][]float64
      var asWidthLine [][]float64
      var asWidthLines [][][]float64
      var asfwhm []float64

      for i, set := range fitAntiStokes {
```

```
132
133         f := func(dst, guess []float64) {
134
135           amp, wid, cen, c := guess[0], guess[1], guess[2], guess[3]
136
137           for i := range as[set][0] {
138             x := as[set][0][i]
139             y := as[set][1][i]
140             dst[i] = (.25 * amp * math.Pow(wid, 2) / (math.Pow(x - cen, 2) + (.25 * math.
    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       )
```

```
    }

    fitStokes := []int{0,1,2,3}
    if len(fitStokes) > 0 {

      header := "\nStokes\nSet \t Power \t\t Width \t\t Peak \t\t Center \n"
      fmt.Printf(header)
      logFile = append(logFile, header)

      var sFits [][][]float64
      var sWidthLine [][]float64
      var sWidthLines [][][]float64
      var ampRatios []float64
      var sLinewidths []float64
      var sfwhm []float64

      for i, set := range fitStokes {

        f := func(dst, guess []float64) {

          amp, wid, cen := guess[0], guess[1], guess[2]

          for i := range s[set][0] {
            x := s[set][0][i]
            y := s[set][1][i]
            dst[i] = (.25 * amp * math.Pow(wid, 2) / (math.Pow(x - cen, 2) + (.25 * math.Pow(wid, 2))) - y) * (1./σs[set][i]) + c
          }
        }

        jacobian := lm.NumJac{Func: f}

        // Solve for fit
        toBeSolved := lm.LMProblem{
          Dim:        3,
          Size:       len(s[set][0]),
          Func:       f,
          Jac:        jacobian.Jac,
          InitParams: []float64{amp, wid, cen, c},
          Tau:        1e-6,
          Eps1:       1e-8,
          Eps2:       1e-8,
        }

        results, _ := lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e-16})

        amp, wid, cen := results.X[0], math.Abs(results.X[1]), results.X[2]

        sfwhm = append(sfwhm, wid*1000)

        str := fmt.Sprintf("%d \t %.2f mW \t %.2f MHz \t %.6f uV \t %.4f GHz\n", set, sPowers[set], wid*1000, amp, cen)
        fmt.Printf(str)
        logFile = append(logFile, str)

        // Create Lorentzian fit data according to solved fit parameters
        df := .001
        f0 := s[set][0][0]
        fitPts := int((s[set][0][len(s[set][0]) - 1] - f0)/df) + 1
        sFits = append(sFits, generateFitData(amp, wid, cen, c, f0, df, fitPts))

        // Width lines
        sWidthLine = [][]float64{{cen - wid/2, cen + wid/2},{amp/2, amp/2}}
        sWidthLines = append(sWidthLines, sWidthLine)

        if len(fitStokes) == len(fitAntiStokes) {
          // For height ratio
```

```
            ampRatios = append(ampRatios, amp/asAmps[i])
          }

          // For linewidth
          sLinewidths = append(sLinewidths, sfwhm[i])
        }
        fmt.Printf("\n")
        logFile = append(logFile, "\n")

        goPlotsFits(
          fitStokes, s, sFits, sWidthLines, σs, sLabel, sfwhm, sNotes, temp, slide,
          sample, logpath, coolingExperiment,
        )

        goPlotsPowerVsWid(
          fitStokes, sLabel, sNotes, sfwhm, temp, slide, sample, logpath,
          coolingExperiment,
        )

        eq := true
        if len(fitAntiStokes) != len(fitStokes) {
          eq = false
        } else {
          for i, v := range fitAntiStokes {
            if v != fitStokes[i] {
              eq = false
              break
            }
          }
        }
        if eq {
          var powers []float64
          for i, v := range asPowers {
            powers = append(powers, (v + sPowers[i])/2)
          }
          goPlotHeightRatios(
            fitStokes, ampRatios, powers, sLabel, sample, logpath,
            coolingExperiment, slide,
          )

          ΓasEff, ΓsEff := Γeff(asPowers[len(asPowers)-1], Γ, length, gb, coolingExperiment)
          goPlotLinewidths(
            fitStokes, ΓasEff, ΓsEff, asLinewidths, sLinewidths, asPowers,
            sPowers, sLabel, sample, logpath, coolingExperiment, slide,
          )

        } else {
          str := fmt.Sprintf("Stokes & AntiStokes sets not equal\n" +
            "(Height ratio and linewidth plots not produced)\n")
          fmt.Printf(str)
          logFile = append(logFile, str)
        }
      }
    }

  } else if cabs {

    setsToPlotCABS := []int{0}

    //setsToPlotCABS := rangeInt(0, 15)

    normalized := []string{} // "Powers"
    cabsData, sigUnit := getCABSData(
      setsToPlotCABS, lock, sigFilepath, freqFilepath, normalized,
    )

    sigmaMultiple := 1.
    cabsData = σCABS(
```

```
    setsToPlotCABS, numAvgs, cabsData, sigUnit, sigmaMultiple, normalized,
  )

  if contains(normalized, "Powers") {
    cabsData = normalizeByPowers(setsToPlotCABS, cabsData, pumpPowers, stokesPowers,
  probePowers)
    fmt.Println("*Data normalized by " + normalized[0] + "*\n")
    logFile = append(logFile, fmt.Sprintf("*Data normalized by %s*\n", normalized[0]))

  }

  // Fit data / Sinc
  var initialParams []float64
  switch sample {
    case "CS2":
      initialParams = []float64{25, 2.5, .08, 0} //amp, cen, wid, C
    case "UHNA3":
      initialParams = []float64{10, 9.14, .1, 0} //amp, cen, wid, C
    default:
      initialParams = []float64{1, 5, .1, 0}
  }

  optimizedParams := make([][]float64, len(cabsData))
  phaseMatchPeaks := make([]float64, setsToPlotCABS[len(setsToPlotCABS)-1]+1)
  pumpProbeSep := make([]float64, setsToPlotCABS[len(setsToPlotCABS)-1]+1)

  for _, set := range setsToPlotCABS {

    optimizedParams[set] = FitLorentzian(
      // freq, sig, σ, guess
      cabsData[set][0], cabsData[set][1], cabsData[set][2], initialParams,
    )

    phaseMatchPeaks[set] = optimizedParams[set][0]
    probeValue, err := strconv.ParseFloat(probeLaser[set], 64)
    if err != nil {
      // handle error, maybe log it and/or return
      log.Fatal("Failed to parse probeLaser:", err)
    }

    pumpValue, err := strconv.ParseFloat(pumpLaser[set], 64)
    if err != nil {
      // handle error, maybe log it and/or return
      log.Fatal("Failed to parse pumpLaser:", err)
    }

    pumpProbeSep[set] = (probeValue - pumpValue) / .008
  }

  if sinc {

    plotSinc(
      setsToPlotCABS, [][]float64{pumpProbeSep, phaseMatchPeaks}, label,
      sample, logpath, length, slide,
    )
  }

  binCabsSets := []int{}
  if len(binCabsSets) > 0 {
    binMHz := 11.
    logFile = logBinning(
      logFile, binCabsSets, binMHz,
      )
    cabsData = binCabs(binCabsSets, cabsData, binMHz) // 3. combine above-calculated σ (
  cabsData[set][2]) with binned σ. (only relevant if binned)
  }

  logFile = logPlots(
```

```
      logFile, setsToPlotCABS, numAvgs, date, label, setNums, startTime, endTime,
      pumpPowers, stokesPowers, probePowers, lockinRange, dwell, bandwidth,
      dataRate, order, startFrequency, stopFrequency, step, pumpProbeSep,
      pumpLaser, probeLaser, probeFilter, stokesFilter, notes, optimizedParams,
    )
    plotCABS(
      setsToPlotCABS, cabsData, label, normalized, sample, sigUnit, logpath, length, slide,
    )
  }

  writeLog(logpath, logFile)
}

//------------------------------------------------------------------------------//

func flags() (
  bool, bool, bool, bool, bool, string, string, string, float64, int,
) {

  var cabs, lock, temp, slide, sinc bool
  var sample, coolingExperiment, note string
  var length float64
  var avg int

  flag.BoolVar(&cabs, "cabs", false, "CABS data")
  flag.BoolVar(&lock, "lockin", false, "lock-in data")
  flag.BoolVar(&temp, "temp", false, "contains temperature data in notes column")
  flag.BoolVar(&slide, "slide", false, "format figures for slide presentation")
  flag.BoolVar(&sinc, "sinc", false, "plot sinc^2 function (phase-matching data)")
  flag.StringVar(&sample, "sample", "", "sample: LCOF, UHNA3, CS2, Te, TeO2, glass slide")
  flag.StringVar(&coolingExperiment, "cooling", "", "Cooling data: pump-probe or pump-only")
  flag.StringVar(&note, "note", "", "note to append folder name")
  flag.Float64Var(&length, "len", 0, "length of sample in meters")
  flag.IntVar(&avg, "avg", 0, "number of CSV files to average")
  flag.Parse()

  if coolingExperiment != "" && cabs {
    fmt.Println("flag.Parse(): data flagged as both cooling and CABS.")
    os.Exit(1)
  }

  if sample == "LCOF" && length == 0 {
    fmt.Println("Specify length of sample in meters with -len=")
    os.Exit(1)
  }

  return cabs, lock, temp, slide, sinc, sample, coolingExperiment, note, length, avg
}

func logpath(
  note string,
) (
  string,
) {
  return "plots/" + time.Now().Format("2006-Jan-02") + "/" + time.Now().Format("15:04:05") +
    ": " + note
}

func readMeta(
  cabs, lock, temp bool,
  coolingExperiment string,
) (
  []string, []string, []string, []string, []string, []float64, []float64,
  []float64, []float64, []float64, []string, []string, []string,
  []float64, []float64, []float64, []float64, []float64, []float64, []float64, []float64,
  []int, []float64, []float64, []string, []string, []string, []string, []string,
) {

```

```
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 []
        string
477   var pumpLaser, probeLaser, probeFilter, stokesFilter []string
478   var asPowers, sPowers, asNotes, sNotes []float64
479   var pumpPowers, stokesPowers, probePowers []float64
480   var lockinRange, dwell, bandwidth, dataRate, order []float64
481   var startFrequency, stopFrequency, step []float64
482   var numAvgs []int
483   var dateCol, labelCol, setCol, startTimeCol, endTimeCol int
484   var pumpCol, stokesCol, probeCol, filepathCol int
485   var lockinRangeCol, dwellCol, bandwidthCol, dataRateCol, orderCol int
486   var startFrequencyCol, stopFrequencyCol, stepCol, numAvgsCol, notesCol int
487   var pumpLaserCol, probeLaserCol, probeFilterCol, stokesFilterCol int
488
489   for col, heading := range meta[0] {
490     switch heading {
491     case "Date":
492       dateCol = col
493     case "Label":
494       labelCol = col
495     case "Sets":
496       setCol = col
497     case "Start Time":
498       startTimeCol = col
499     case "End Time (hr:min:sec)":
500       endTimeCol = col
501     case "Pump":
502       pumpCol = col
503     case "Stokes":
504       stokesCol = col
505     case "Probe":
506       probeCol = col
507     case "Filepath":
508       filepathCol = col
509     case "Range":
510       lockinRangeCol = col
511     case "Dwell":
512       dwellCol = col
513     case "Bandwidth":
514       bandwidthCol = col
515     case "Data Rate":
516       dataRateCol = col
517     case "Order":
518       orderCol = col
519     case "Start Frequency":
520       startFrequencyCol = col
521     case "Stop Frequency":
522       stopFrequencyCol = col
523     case "Step":
524       stepCol = col
525     case "Pump Laser":
526       pumpLaserCol = col
527     case "Probe Laser":
528       probeLaserCol = col
```

```
    case "Probe Filter":
      probeFilterCol = col
    case "Stokes Filter":
      stokesFilterCol = col
    case "Num Avgs":
      numAvgsCol = col
    case "Notes":
      notesCol = col
    }
  }

  for row, v := range meta {

    if row > 0 {

      date = append(date, v[dateCol])
      label = append(label, v[labelCol])
      set = append(set, v[setCol])
      pumpLaser = append(pumpLaser, v[pumpLaserCol])
      probeLaser = append(probeLaser, v[probeLaserCol])
      stokesFilter = append(stokesFilter, v[stokesFilterCol])
      probeFilter = append(probeFilter, v[probeFilterCol])
      notes = append(notes, v[notesCol])

      if numAvg, err := strconv.Atoi(v[numAvgsCol]); err == nil {
        numAvgs = append(numAvgs, numAvg)
      } else {
        fmt.Println(err)
        os.Exit(1)
      }

      if coolingExperiment != "" {
        if strings.Contains(v[labelCol], "ras") {
          if v, err := strconv.ParseFloat(strings.Split(v[labelCol], " ")[0], 64); err ==
    nil {
            asPowers = append(asPowers, v)
          } else {
            fmt.Println(err)
            os.Exit(1)
          }
        } else if strings.Contains(v[labelCol], "rs"){
          if v, err := strconv.ParseFloat(strings.Split(v[labelCol], " ")[0], 64); err ==
    nil {
            sPowers = append(sPowers, v)
          } else {
            fmt.Println(err)
            os.Exit(1)
          }
        }

        if lock {
          sigFilepath = append(sigFilepath, v[setCol] + "/signal.csv")
          freqFilepath = append(freqFilepath, v[setCol] + "/signal.csv")
        } else {
          filepath = append(filepath, v[filepathCol])
        }

        if temp {
          if strings.Contains(v[labelCol], "as") {
            if asNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
              asNotes = append(asNotes, asNote)
            } else {
              fmt.Println(err)
              os.Exit(1)
            }
          } else {
            if sNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
              sNotes = append(sNotes, sNote)
```

```
            } else {
              fmt.Println(err)
              os.Exit(1)
            }
          }
        }
      } else if cabs {

        if v[pumpCol] == "" {
          pumpPowers = append(pumpPowers, 0)
        } else if v, err := strconv.ParseFloat(v[pumpCol], 64); err == nil {
          pumpPowers = append(pumpPowers, v)
        } else {
          fmt.Println(err)
          fmt.Println("readMeta pump string -> float error")
          os.Exit(1)
        }
        if v[stokesCol] == "" {
          stokesPowers = append(stokesPowers, 0)
        } else if v, err := strconv.ParseFloat(v[stokesCol], 64); err == nil {
          stokesPowers = append(stokesPowers, v)
        } else {
          fmt.Println(err)
          fmt.Println("readMeta stokes string -> float error")
          os.Exit(1)
        }
        if v[probeCol] == "" {
          probePowers = append(probePowers, 0)
        } else if v, err := strconv.ParseFloat(v[probeCol], 64); err == nil {
          probePowers = append(probePowers, v)
        } else {
          fmt.Println(err)
          fmt.Println("readMeta probe string -> float error")
          os.Exit(1)
        }

        if lock {

          startTime = append(startTime, v[startTimeCol])
          endTime = append(endTime, v[endTimeCol])

          sigFilepath = append(sigFilepath, v[setCol] + "/signal.csv")
          freqFilepath = append(freqFilepath, v[setCol] + "/frequency.csv")

          if v[lockinRangeCol] == "" {
            lockinRange = append(lockinRange, 0)
          } else if v, err := strconv.ParseFloat(v[lockinRangeCol], 64); err == nil {
            lockinRange = append(lockinRange, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta lockinRange string -> float error")
            os.Exit(1)
          }
          if v[dwellCol] == "" {
            dwell = append(dwell, 0)
          } else if v, err := strconv.ParseFloat(v[dwellCol], 64); err == nil {
            dwell = append(dwell, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta dwell string -> float error")
            os.Exit(1)
          }
          if v[bandwidthCol] == "" {
            bandwidth = append(bandwidth, 0)
          } else if v, err := strconv.ParseFloat(v[bandwidthCol], 64); err == nil {
            bandwidth = append(bandwidth, v)
          } else {
            fmt.Println(err)
```

```
            fmt.Println("readMeta bandwidth string -> float error")
            os.Exit(1)
          }
          if v[dataRateCol] == "" {
            dataRate = append(dataRate, 0)
          } else if v, err := strconv.ParseFloat(v[dataRateCol], 64); err == nil {
            dataRate = append(dataRate, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta dataRate string -> float error")
            os.Exit(1)
          }
          if v[orderCol] == "" {
            order = append(order, 0)
          } else if v, err := strconv.ParseFloat(v[orderCol], 64); err == nil {
            order = append(order, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta order string -> float error")
            os.Exit(1)
          }
          if v[startFrequencyCol] == "" {
            startFrequency = append(startFrequency, 0)
          } else if v, err := strconv.ParseFloat(v[startFrequencyCol], 64); err == nil {
            startFrequency = append(startFrequency, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta startFrequency string -> float error")
            os.Exit(1)
          }
          if v[stopFrequencyCol] == "" {
            stopFrequency = append(stopFrequency, 0)
          } else if v, err := strconv.ParseFloat(v[stopFrequencyCol], 64); err == nil {
            stopFrequency = append(stopFrequency, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta stopFrequency string -> float error")
            os.Exit(1)
          }
          if v[stepCol] == "" {
            step = append(step, 0)
          } else if v, err := strconv.ParseFloat(v[stepCol], 64); err == nil {
            step = append(step, v)
          } else {
            fmt.Println(err)
            fmt.Println("readMeta step string -> float error")
            os.Exit(1)
          }

        } else {
          filepath = append(filepath, v[filepathCol])
        }

        if temp {
          if strings.Contains(v[labelCol], "as") {
            if asNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
              asNotes = append(asNotes, asNote)
            } else {
              fmt.Println(err)
              os.Exit(1)
            }
          } else {
            if sNote, err := strconv.ParseFloat(v[notesCol], 64); err == nil {
              sNotes = append(sNotes, sNote)
            } else {
              fmt.Println(err)
              os.Exit(1)
            }
```

```
          }
        }
      }
    }
  }

  return date, label, set, startTime, endTime, asPowers, sPowers,
  pumpPowers, stokesPowers, probePowers, filepath, sigFilepath, freqFilepath,
  lockinRange, dwell, bandwidth, dataRate, order, startFrequency, stopFrequency,
  step, numAvgs, asNotes, sNotes, pumpLaser, probeLaser, probeFilter, stokesFilter,
  notes
}

func logHeader(
  cabs, lock, temp, slide bool,
  sample, coolingExperiment, note string,
  length float64,
) (
  []string,
) {

  logFile := []string{}
  if sample != "" {
    logFile = append(logFile, "Sample: " + sample + "\n")
  }
  if note != "" {
    logFile = append(logFile, "Runtime note: " + note + "\n")
  }
  if slide {
    logFile = append(logFile, "Figures formatted for slide presentation\n")
  }

  fmt.Printf(logFile[0])

  if coolingExperiment != "" {
    logFile = append(logFile, "\n*Cooling Data: " + coolingExperiment + "*\n")
    fmt.Printf("\n*Cooling Data: " + coolingExperiment + "*\n")
  } else if cabs {
    logFile = append(logFile, "\n*CABS Data*\n")
    fmt.Printf("\n*CABS Data*\n")
  }
  if temp {
    logFile = append(logFile, "\n*Temperature-dependent data*\n")
    fmt.Printf("\n*Temperature-dependent data*\n")
  }
  if sample == "LCOF" {
    str := fmt.Sprintf("\n*Liquid-core optical fiber sample*\n")
    logFile = append(logFile, str)
    fmt.Printf(str)
  }
  if lock {
    str := fmt.Sprintf("\n*Data gathered from Lock-in*\n\n")
    logFile = append(logFile, str)
    fmt.Printf(str)
  } else {
    str := fmt.Sprintf("\n*Data gathered from Spectrum Analyzer*\n\n")
    logFile = append(logFile, str)
    fmt.Printf(str)
  }

  return logFile
}

func avgCSVs(
  nAvg int,
  powers []float64,
) (
  [][]float64, [][]float64,
```

```
) {

  // Peek at 0th file to get length
  f, err := os.Open("Data/" + fmt.Sprint(powers[0]) + "/rs0.csv")
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
  peek, err := readCSV(f)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  σbs, σrs, σras, σbas, σas, σs := make([][]float64, len(powers)),
  make([][]float64, len(powers)), make([][]float64, len(powers)),
  make([][]float64, len(powers)), make([][]float64, len(powers)),
  make([][]float64, len(powers))
  for i := range σbs {
    σbs[i], σrs[i], σras[i], σbas[i], σas[i], σs[i] = make([]float64, len(peek)-2),
    make([]float64, len(peek)-2), make([]float64, len(peek)-2),
    make([]float64, len(peek)-2), make([]float64, len(peek)-2),
    make([]float64, len(peek)-2)
  }

  σsString, σasString := make([][]string, len(powers)), make([][]string, len(powers))
  for i := range σsString {
    σsString[i], σasString[i] = make([]string, len(peek)-2), make([]string, len(peek)-2)
  }

  if nAvg > 0 {
    for _, name := range []string{"bs", "rs", "ras", "bas"} {

      for set, powFloat := range powers {

        pow := fmt.Sprint(powFloat)

        var newDataCSV [][]string

        // sigColsToAvg[nAvg][sig]
        sigColsToAvg := make([][]float64, nAvg)
        for k := range sigColsToAvg {
          sigColsToAvg[k] = make([]float64, len(peek)-1)
        }

        σCSV := make([][]string, 1)
        σCSV[0] = make([]string, len(peek)-2)

        for i := 0; i < nAvg; i++ {
          // Read
          f, err := os.Open("Data/" + pow + "/" + name + fmt.Sprint(i) + ".csv")
          if err != nil {
            fmt.Println(err)
            os.Exit(1)
          }

          data, err := readCSV(f)
          newDataCSV = data

          for j := 1; j < len(data); j++ {

            s := strings.ReplaceAll(data[j][2]," ","")

            sig, err := strconv.ParseFloat(s, 64)
            if err != nil {
              fmt.Println(err)
              os.Exit(1)
            }
```

```
            sigColsToAvg[i][j-1] = sig
          }
        }

        toAvg := make([]float64, nAvg)
        averagedCol := make([]float64, len(sigColsToAvg[0]))
        σCol := make([]float64, len(sigColsToAvg[0]))
        for i := 0; i < len(sigColsToAvg[0]); i++ {
          for j := 0; j < nAvg; j++ {
            toAvg[j] = sigColsToAvg[j][i]
          }
          averagedCol[i] = avg(toAvg)
          σCol[i] = σ(toAvg)
        }

        switch name {
          case "bs":
            σbs[set] = σCol
          case "rs":
            σrs[set] = σCol
          case "ras":
            σras[set] = σCol
          case "bas":
            σbas[set] = σCol
          default:
            fmt.Println("error in switch statement with σ in avgCSVs\n\n")
            os.Exit(1)
        }

        for i := 2; i < len(newDataCSV); i++ {
          newDataCSV[i][2] = strconv.FormatFloat(averagedCol[i-2], 'f', -1, 64)
          σCSV[0][i-2] = strconv.FormatFloat(σCol[i-2], 'f', -1, 64)
        }

        fData, err := os.Create("Data/" + pow + "/" + name + ".csv")
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }

        wData := csv.NewWriter(fData)
        err = wData.WriteAll(newDataCSV)
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }

        fσ, err := os.Create("Data/" + pow + "/" + name + "σ.csv")
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }

        wσ := csv.NewWriter(fσ)
        err = wσ.WriteAll(σCSV)
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }
      }
    }

    for set := range powers {
      for i := range σs[0] {
        σs[set][i] = math.Sqrt(math.Pow(σrs[set][i], 2) + math.Pow(σbs[set][i], 2))
        σas[set][i] = math.Sqrt(math.Pow(σras[set][i], 2) + math.Pow(σbas[set][i], 2))

```

```
        σsString[set][i] = fmt.Sprint(σs[set][i])
        σasString[set][i] = fmt.Sprint(σas[set][i])
      }
    }

    fσs, err := os.Create("Data/σs.csv")
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    wσs := csv.NewWriter(fσs)
    err = wσs.WriteAll(σsString)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    fσas, err := os.Create("Data/σas.csv")
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    wσas := csv.NewWriter(fσas)
    err = wσas.WriteAll(σasString)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

  } else {

    fσs, err := os.Open("Data/σs.csv")
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    σsReader := csv.NewReader(fσs)
    σsString, err := σsReader.ReadAll()
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    fσas, err := os.Open("Data/σas.csv")
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    σasReader := csv.NewReader(fσas)
    σasString, err := σasReader.ReadAll()
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    for pow := range σsString {
      for i := range σsString[pow] {
        σs[pow][i], err = strconv.ParseFloat(σsString[pow][i], 64)
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }

        σas[pow][i], err = strconv.ParseFloat(σasString[pow][i], 64)
```

```
        if err != nil {
          fmt.Println(err)
          os.Exit(1)
        }
      }
    }

  }

  return σas, σs
}

func rangeInt(
  start, end int,
) (
  []int,
) {
    nums := make([]int, end-start)
    for i := range nums {
        nums[i] = start + i
    }
    return nums
}

func getCoolingData(
  lock bool,
  fileNames, labels []string,
) (
  [][][]float64, [][][]float64, [][][]float64, [][][]float64,
) {

  var bas, bs, ras, rs [][][]float64
  sig := false

  if lock {

    // Assign data by name
    for i := 0; i < len(fileNames)/2; i++ {

      if i == 0 || i%2 != 0 {
        sig = true
      }

      if strings.Contains(labels[i], "bas") {
        bas = append(bas, getData(lock, sig, fileNames[i]))
      } else if strings.Contains(labels[i], "bs") {
        bs = append(bs, getData(lock, sig, fileNames[i]))
      } else if strings.Contains(labels[i], "ras") {
        ras = append(ras, getData(lock, sig, fileNames[i]))
      } else if strings.Contains(labels[i], "rs") {
        rs = append(rs, getData(lock, sig, fileNames[i]))
      }
    }
  } else {

    // Assign data by name
    for i, fileName := range fileNames {

      if i == 0 || i%2 != 0 {
        sig = true
      }

      if strings.Contains(labels[i], "bas") {
        bas = append(bas, getData(lock, sig, fileName))
      } else if strings.Contains(labels[i], "bs") {
        bs = append(bs, getData(lock, sig, fileName))
      } else if strings.Contains(labels[i], "ras") {
        ras = append(ras, getData(lock, sig, fileName))
```

```
      } else if strings.Contains(labels[i], "rs") {
        rs = append(rs, getData(lock, sig, fileName))
      }
    }
  }

  return ras, bas, rs, bs
}

func getCABSData(
  sets []int,
  lock bool,
  sigFileNames, freqFileNames, normalized []string,
) (
  [][][]float64, string,
) {

  // final form: cabsData[set][0: freq, 1: sig, 2: σ][rows of freq/sig/σ]
  var cabsData [][][]float64
  var sigUnit string

  if lock {

    var cabsDataPreUnit [][][]float64
    for i, v := range sigFileNames {
      lockData := getLockData(v, freqFileNames[i])
      cabsDataPreUnit = append(cabsDataPreUnit, lockData)
    }

    // if not going to be normalized
    if !contains(normalized, "Powers") {

      // Check for most appropriate signal unit (preference of larger)
      largestSig := 0.
      for set := range cabsDataPreUnit {
        for _, s := range sets {
          if set == s {
            for _, v := range cabsDataPreUnit[set][1] {
              if v > largestSig {
                largestSig = v
              }
            }
          }
        }

      }
      if largestSig > 1e-3 {
        sigUnit = "mV"
        for set := range cabsDataPreUnit {
          for i, v := range cabsDataPreUnit[set][1] {
            cabsDataPreUnit[set][1][i] = v*1e3
          }
          cabsData = append(cabsData, cabsDataPreUnit[set])
        }
      } else if largestSig > 1e-6 {
        sigUnit = "μV"
        for set := range cabsDataPreUnit {
          for i, v := range cabsDataPreUnit[set][1] {
            cabsDataPreUnit[set][1][i] = v*1e6
          }
          cabsData = append(cabsData, cabsDataPreUnit[set])
        }
      } else if largestSig > 1e-9 {
        sigUnit = "nV"
        for set := range cabsDataPreUnit {
          for i, v := range cabsDataPreUnit[set][1] {
            cabsDataPreUnit[set][1][i] = v*1e9
          }
```

```
          cabsData = append(cabsData, cabsDataPreUnit[set])
        }
      } else if largestSig > 1e-12 {
        sigUnit = "pV"
        for set := range cabsDataPreUnit {
          for i, v := range cabsDataPreUnit[set][1] {
            cabsDataPreUnit[set][1][i] = v*1e12
          }
          cabsData = append(cabsData, cabsDataPreUnit[set])
        }
      }
    } else {
      sigUnit = ""
      for set := range cabsDataPreUnit {
        cabsData = append(cabsData, cabsDataPreUnit[set])
      }
    }
  }

  return cabsData, sigUnit
}

func getData(
  lock, sig bool,
  csvName string,
) (
  [][]float64,
) {

  // Read
  f, err := os.Open("Data/" + csvName)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
  defer f.Close()
  dataStr, err := readCSV(f)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  // Separate, Strip, & Transpose
  var frequencyStrT, signalStrT []string

  if lock {
    if sig {
      for i := range dataStr {
        signalStrT = append(signalStrT, dataStr[i][0])
      }
    } else {
      for i := range dataStr {
        frequencyStrT = append(frequencyStrT, dataStr[i][0])
      }
    }
  } else {
    for i := 1; i < len(dataStr); i++ {
      frequencyStrT = append(frequencyStrT, strings.ReplaceAll(dataStr[i][0]," ",""))
      signalStrT = append(signalStrT, strings.ReplaceAll(dataStr[i][2]," ",""))
    }
  }

  // Convert to float
  var frequency, signal []float64

  for _, freqElem := range frequencyStrT {
    if freqValue, err := strconv.ParseFloat(freqElem, 64); err != nil {
      fmt.Println(err)
```

```
      os.Exit(1)
    } else {
      frequency = append(frequency, freqValue/1e9)
    }
  }

  for _, sigElem := range signalStrT {
    if sigValue, err := strconv.ParseFloat(sigElem, 64); err != nil {
      fmt.Println(err)
      os.Exit(1)
    } else {
      signal = append(signal, sigValue)
    }
  }

  if !lock {
    // Convert to Linear if dBm
    if dataStr[1][3] == " dBm" {

      // uV
      var uV []float64
      for _, dBm := range signal {
        uV = append(uV, math.Pow(10, 6)*math.Pow(10, dBm/10.))
      }
      return [][]float64{frequency, uV}

      /* nV
      var nV []float64
      for _, dBm := range signal {
        nV = append(nV, 1000*math.Pow(10, 6)*math.Pow(10, dBm/10.))
      }
      return [][]float64{frequency, nV}
      */

    } else if dataStr[1][3] == "  uV" {
      var nV []float64

      for _, uV := range signal {
        nV = append(nV, 1000*uV)
      }

      /* Convert to picovolts
      var pV []float64
      for _, uV := range signal {
        pV = append(pV, 1000*uV)
      }*/

      return [][]float64{frequency, nV}
      }

    fmt.Println("Warning: check units - not uV or dBm")
    return [][]float64{frequency, signal}
  } else {
    return [][]float64{frequency, signal}
  }
}

func getLockData(
  sigCSVName, freqCSVName string,
) (
  [][]float64,
) {

  // Read signal data
  sigf, err := os.Open("Data/" + sigCSVName)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
```

```
}
sigDataStr, err := readCSV(sigf)
if err != nil {
  fmt.Println(err)
  sigf.Close()
  os.Exit(1)
}

// Explicitly close the file (avoids too many files open error)
sigf.Close()

// Read frequency data
freqf, err := os.Open("Data/" + freqCSVName)
if err != nil {
  fmt.Println(err)
  os.Exit(1)
}
freqDataStr, err := readCSV(freqf)
if err != nil {
  fmt.Println(err)
  freqf.Close()
  os.Exit(1)
}

// Explicitly close the file (avoids too many files open error)
freqf.Close()

// Transpose
var freqStrT, sigStrT []string

for i := range sigDataStr {
  sigStrT = append(sigStrT, sigDataStr[i][0])
  freqStrT = append(freqStrT, freqDataStr[i][0])
}

// Convert to float
var frequency, signal []float64

for _, freqElem := range freqStrT {
  if freqValue, err := strconv.ParseFloat(freqElem, 64); err != nil {
    fmt.Println(err)
    os.Exit(1)
  } else {
    frequency = append(frequency, freqValue/1e9)
  }
}

for _, sigElem := range sigStrT {
  if sigValue, err := strconv.ParseFloat(sigElem, 64); err != nil {
    fmt.Println(err)
    os.Exit(1)
  } else {
    signal = append(signal, sigValue)
  }
}

/* Convert to pV
maxSig := 0.
sigUnit := "pV"
for i, v := range signal {
  signal[i] = v*1e9
  if signal[i] > maxSig {
    maxSig = signal[i]
  }
}

if maxSig > 1000. {
  // Convert to uV
```

```
    for i, v := range signal {
      signal[i] = v*1e-3
    }
    sigUnit = "uV"
  }*/


  return [][]float64{frequency, signal}
}

func readCSV(
  rs io.ReadSeeker,
) (
  [][]string, error,
) {
  // Skip first row (line)
  row1, err := bufio.NewReader(rs).ReadSlice('\n')
  if err != nil {
    return nil, err
  }
  _, err = rs.Seek(int64(len(row1)), io.SeekStart)
  if err != nil {
    return nil, err
  }

  // Read remaining rows
  r := csv.NewReader(rs)
  rows, err := r.ReadAll()
  if err != nil {
    return nil, err
  }
  return rows, nil
}

func logBinning(
  logFile []string,
  binCabsSets []int,
  binMHz float64,
) (
  []string,
) {

  for _, set := range binCabsSets {
    logFile = append(logFile, fmt.Sprintf("Run %d binned to %.3f MHz\n", set+1, binMHz))
  }

  return logFile
}

func logPlots(
  logFile []string,
  setsToPlotCABS, numAvgs []int,
  date, label, run, startTime, endTime []string,
  pumpPowers, stokesPowers, probePowers, lockinRange, dwell []float64,
  bandwidth, dataRate, order, startFrequency, stopFrequency, step, pumpProbeSep []float64,
  pumpLaser, probeLaser, probeFilter, stokesFilter, notes []string,
  optimizedParams [][]float64,
) (
  []string,
) {

  for _, set := range setsToPlotCABS {
    logFile = append(logFile, fmt.Sprintf("\nRun %s\n", run[set]))
    logFile = append(logFile, fmt.Sprintf("\tData taken: %s\n", date[set]))
    logFile = append(logFile, fmt.Sprintf("\tLabel: %s\n", label[set]))
    logFile = append(logFile, fmt.Sprintf("\tStart Time (h:m:s): %s\n", startTime[set]))
    logFile = append(logFile, fmt.Sprintf("\tEnd Time(h:m:s): %s\n", endTime[set]))
    logFile = append(logFile, fmt.Sprintf("\tPump Laser: %s nm\n", pumpLaser[set]))
```

```
    logFile = append(logFile, fmt.Sprintf("\tProbe Laser: %s nm\n", probeLaser[set]))
    logFile = append(logFile, fmt.Sprintf("\tPump-Probe Separation: %.2f GHz\n",
    pumpProbeSep[set]))
    logFile = append(logFile, fmt.Sprintf("\tStokes Filter: %s nm\n", stokesFilter[set]))
    logFile = append(logFile, fmt.Sprintf("\tProbe Filter: %s nm\n", probeFilter[set]))
    logFile = append(logFile, fmt.Sprintf("\tPump Power: %.3f mW\n", pumpPowers[set]))
    logFile = append(logFile, fmt.Sprintf("\tStokes Power: %.3f mW\n", stokesPowers[set]))
    logFile = append(logFile, fmt.Sprintf("\tProbe Power: %.3f mW\n", probePowers[set]))
    logFile = append(logFile, fmt.Sprintf("\tLock-in Range: %.2f\n", lockinRange[set]))
    logFile = append(logFile, fmt.Sprintf("\tDwell Time: %.9f s\n", dwell[set]))
    logFile = append(logFile, fmt.Sprintf("\tLock-in Bandwidth: %.2f Hz\n", bandwidth[set]))
    logFile = append(logFile, fmt.Sprintf("\tLock-in Bandwidth Order: %.0f\n", order[set]))
    logFile = append(logFile, fmt.Sprintf("\tData Sampling Rate: %.2f Sa/s\n", dataRate[set
    ]))
    logFile = append(logFile, fmt.Sprintf("\tStart Frequency: %.2f Hz\n", startFrequency[set
    ]))
    logFile = append(logFile, fmt.Sprintf("\tStop Frequency: %.2f Hz\n", stopFrequency[set])
    )
    logFile = append(logFile, fmt.Sprintf("\tStep Size: %.2f Hz\n", step[set]))
    logFile = append(logFile, fmt.Sprintf("\tNumber of Averages: %d\n", numAvgs[set]))
    logFile = append(logFile, fmt.Sprintf("\tData Collection Note: %s\n\n", notes[set]))
    logFile = append(logFile, fmt.Sprintf("\tFit Parameters:\n"))
    logFile = append(logFile, fmt.Sprintf("\t\tAmp: %v\n", optimizedParams[set][0]))
    logFile = append(logFile, fmt.Sprintf("\t\tCen: %v\n", optimizedParams[set][1]))
    logFile = append(logFile, fmt.Sprintf("\t\tWid: %v\n", optimizedParams[set][2]))
    logFile = append(logFile, fmt.Sprintf("\t\tC:   %v\n\n", optimizedParams[set][3]))
  }

  return logFile
}

func getAllLabels(
  label []string,
) (
  []string, []string, []string, []string,
) {

  var rasLabel, basLabel, rsLabel, bsLabel []string

  // Assign labels by verifying label
  for _, thisLabel := range label {
    if strings.Contains(thisLabel, "ras") {
      rasLabel = append(rasLabel, thisLabel)
    } else if strings.Contains(thisLabel, "bas") {
      basLabel = append(basLabel, thisLabel)
    } else if strings.Contains(thisLabel, "rs") {
      rsLabel = append(rsLabel, thisLabel)
    } else if strings.Contains(thisLabel, "bs") {
      bsLabel = append(bsLabel, thisLabel)
    }
  }

  return rasLabel, basLabel, rsLabel, bsLabel
}

func buildData(
  data [][]float64,
) (
  plotter.XYs,
) {

  xy := make(plotter.XYs, len(data[0]))

  for i := range xy {
    xy[i].X = data[0][i]
    xy[i].Y = data[1][i]
  }

```

```
  return xy
}

func buildErrors(
   σ []float64,
) (
  plotter.Errors,
) {

  error := make(plotter.Errors, len(σ))

  for i := range error {
    error[i].Low, error[i].High = σ[i], σ[i]
  }

  return error
}

func plotRaw(
  sets []int,
  bas, ras, bs, rs [][][]float64,
  basLabel, rasLabel, bsLabel, rsLabel []string,
) {

  for i := range sets {
    dimensions := 2
    persist := true
    debug := false
    plot, _ := glot.NewPlot(dimensions, persist, debug)

    plot.SetTitle("Raw")
    plot.SetXLabel("Frequency (GHz)")
    plot.SetYLabel("Signal (uV)")

    plot.AddPointGroup(rasLabel[sets[i]], "points", ras[sets[i]])
    plot.AddPointGroup(basLabel[sets[i]], "points", bas[sets[i]])
    //plot.AddPointGroup(rsLabel[sets[i]], "points", rs[sets[i]])
    //plot.AddPointGroup(bsLabel[sets[i]], "points", bs[sets[i]])
  }
}

func plotCABS(
  sets []int,
  cabsData [][][]float64,
  label, normalized []string,
  sample, sigUnit, logpath string,
  length float64,
  slide bool,
) {

  var l string

  switch length {
  case 0.0:
    l = ""
  case 0.001:
    l = "1 mm"
  case 0.01:
    l = "1 cm"
  case 0.004:
    l = "4 mm"
  case 0.0000005:
    l = "500 nm"
  case 0.0001:
    l = "100 µm"
  case 0.00001:
    l = "10 µm"
  default:
```

```
    l = strconv.FormatFloat(length, 'f', 1, 64)
  }

  type errorPoints struct {
    plotter.XYs
    plotter.YErrors
  }

  title := l + " " + sample + " CABS"
  xlabel := "Frequency (GHz)"
  var ylabel string
  if contains(normalized, "Powers") {
    ylabel = "Normalized by Powers (V/W^3)"
  } else {
    ylabel = "Spectral Density (" + sigUnit + ")"
  }
  legend := ""

  /* Manual Axes
  xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes("CABS", sample, "")
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }*/

  // Auto Axes
  xmax := 0.
  xmin := cabsData[0][0][0]
  for _, set := range sets {
    if cabsData[set][0][0] < xmin {
      xmin = cabsData[set][0][0]
    }
    if cabsData[set][0][len(cabsData[set][0])-1] > xmax {
      xmax = cabsData[set][0][len(cabsData[set][0])-1]
    }
  }

  xrange := []float64{xmin, xmax}
  xtick := 0.
  displayDigits := 2
  switch {
    case (xmax - xmin)/8 > 0.25:
      xtick = 0.5
      displayDigits = 1
    case (xmax - xmin)/8 > 0.1:
      xtick = 0.25
    case (xmax - xmin)/8 > 0.075:
      xtick = 0.075
    case (xmax - xmin)/8 > 0.05:
      xtick = 0.05
    case (xmax - xmin)/8 > 0.025:
      xtick = 0.025
    case (xmax - xmin)/8 > 0.01:
      xtick = 0.02
    case (xmax - xmin)/8 > 0.0075:
      xtick = 0.0075
    case (xmax - xmin)/8 > 0.005:
      xtick = 0.005
    case (xmax - xmin)/8 > 0.0025:
      xtick = 0.0025
    case (xmax - xmin)/8 > 0.001:
      xtick = 0.001
  }

  firstTick := 0.
  for m := float64(int(xmin)); m <= xmin; m += xtick {
    firstTick = m
  }
```

```
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',
      displayDigits, 64))
1619    } else {
1620      xtickLabels = append(xtickLabels, "")
1621    }
1622  }
1623
1624  ymax := 0.
1625  ymin := 10000.
1626  for _, set := range sets {
1627    for i, v := range cabsData[set][1] {
1628      if len(cabsData[set]) > 2 && v + cabsData[set][2][i]/2 > ymax {
1629        ymax = v + cabsData[set][2][i]/2
1630      } else if len(cabsData[set]) < 3 && v > ymax {
1631        ymax = v
1632      }
1633      if len(cabsData[set]) > 2 && v - cabsData[set][2][i]/2 < ymin {
1634        ymin = v - cabsData[set][2][i]/2
1635      } else if len(cabsData[set]) < 3 && v < ymin {
1636        ymin = v
1637      }
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
1666    pts := buildData(cabsData[set])
1667
1668    if len(cabsData[set]) > 2 {
1669      σErr := buildErrors(cabsData[set][2])
1670
1671      setPoints := errorPoints {
1672        XYs: pts,
1673        YErrors: plotter.YErrors(σErr),
1674      }
1675
1676      plotSet, err := plotter.NewScatter(setPoints)
1677      if err != nil {
```

```
        fmt.Println(err)
        os.Exit(1)
      }

      // Error bars
      e, err := plotter.NewYErrorBars(setPoints)
      if err != nil {
        fmt.Println(err)
        os.Exit(1)
      }
      e.LineStyle.Color = palette(set, false, "")

      plotSet.GlyphStyle.Color = palette(set, false, "")
      plotSet.GlyphStyle.Radius = vg.Points(5) //3
      plotSet.Shape = draw.CircleGlyph{}

      p.Add(e, plotSet, t, r)

    } else {

      plotSet, err := plotter.NewScatter(pts)
      if err != nil {
        fmt.Println(err)
        os.Exit(1)
      }

      plotSet.GlyphStyle.Color = palette(set, false, "")
      plotSet.GlyphStyle.Radius = vg.Points(5) //3
      plotSet.Shape = draw.CircleGlyph{}

      p.Add(plotSet, t, r)
    }

    // Legend
    l, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    l.GlyphStyle.Color = palette(set, false, "")
    l.GlyphStyle.Radius = vg.Points(8) //6
    l.Shape = draw.CircleGlyph{}
    p.Legend.Add(label[set], l)
  }

  /* Guide Line
  guideLine := make(plotter.XYs, 2)

  guideLine[0].X = 12.1
  guideLine[0].Y = yrange[0]
  guideLine[1].X = 12.1
  guideLine[1].Y = yrange[1]

  plotGuideLine, err := plotter.NewLine(guideLine)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  plotGuideLine.Color = color.RGBA{R: 128, G: 128, B: 128, A: 255}

  p.Add(plotGuideLine)
  p.Legend.Add("12.1 GHz", plotGuideLine)
  */

  savePlot(p, "CABS", logpath)
}
```

```
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 "TeO2":
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}
```

```
      xtick := []float64{11.5, 11.6, 11.7, 11.8, 11.9, 12.0, 12.1, 12.2, 12.3, 12.4, 12.5}
      ytick := []float64{0, 300, 600, 900, 1200, 1500, 1800, 2100}
      xtickLabel := []string{"11.5", "", "11.7", "", "11.9", "", "12.1", "", "12.3", "", "
    12.5"}
      ytickLabel := []string{"", "300", "", "900", "", "1500", "", "2100"}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
    case "Sapphire":
      xrange := []float64{.100, .140}
      yrange := []float64{-50, 50}
      xtick := []float64{.100, .110, .120, .130, .140}
      ytick := []float64{-50, -25, 0, 25, 50}
      xtickLabel := []string{".100", "", ".120", "", ".140"}
      ytickLabel := []string{"-50", "", "0", "", "50"}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
  }
  case "fits":
    switch sample {
    case "LCOF":
      if coolingExperiment == "pump-only" {
        xrange := []float64{2.0, 2.5}
        yrange := []float64{0, 110}
        xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
        ytick := []float64{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140,
    150}
        xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
        ytickLabel := []string{"0", "", "20", "", "40", "", "60", "", "80", "", "100", ""}

        return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
      } else if coolingExperiment == "pump-probe" {
        xrange := []float64{2.0, 2.5}
        yrange := []float64{-.05, 2.5}
        xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
        ytick := []float64{0, 0.5, 1, 1.5, 2, 2.5}
        xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
        ytickLabel := []string{"0", "", "1", "", "2", ""}

        return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
      }
    case "UHNA3":
      xrange := []float64{1, 1.36}
      yrange := []float64{0, 17.5}
      xtick := []float64{1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4}
      ytick := []float64{0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5}
      xtickLabel := []string{"1", "", "1.1", "", "1.2", "", "1.3", "", "1.4"}
      ytickLabel := []string{"0", "", "5", "", "10", "", "15", ""}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
    case "[Unspecified Sample]":
      xrange := []float64{2, 2.5}
      yrange := []float64{0, 17.5}
      xtick := []float64{2, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5}
      ytick := []float64{0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5}
      xtickLabel := []string{"2", "", "2.1", "", "2.2", "", "2.3", "", "2.4", "", "2.5"}
      ytickLabel := []string{"0", "", "5", "", "10", "", "15", ""}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
    }
  case "pow vs wid":
    switch sample {
    case "LCOF":
      xrange := []float64{0, 300}
      yrange := []float64{90, 120}
      xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300}
      ytick := []float64{90, 92.5, 95, 97.5, 100, 102.5, 105, 107.5, 110, 112.5, 115, 117.5,
     120}
      xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200", "", "250", "",
```

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

```
      xrange := []float64{25, 200}
      yrange := []float64{50, 125}
      xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
      ytick := []float64{50, 62.5, 75, 87.5, 100, 112.5, 125}
      xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
      ytickLabel := []string{"50", "", "75", "", "100", "", "125"}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
    case "[Unspecified Sample]":
      xrange := []float64{25, 200}
      yrange := []float64{50, 125}
      xtick := []float64{0, 25, 50, 75, 100, 125, 150, 175, 200}
      ytick := []float64{50, 62.5, 75, 87.5, 100, 112.5, 125}
      xtickLabel := []string{"0", "", "50", "", "100", "", "150", "", "200"}
      ytickLabel := []string{"50", "", "75", "", "100", "", "125"}

      return xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, nil
    }
  }

  return []float64{}, []float64{}, []float64{}, []float64{}, []string{},
    []string{}, fmt.Errorf("func axes: no predefined axis for '%s'", plot)
}

func subtractBackground(
  ras, bas, rs, bs [][][]float64,
  coolingExperiment string,
) (
  [][][]float64, [][][]float64,
) {

  var s, as [][][]float64

  for i := range rs {
     s = append(s, subtract(bs[i], rs[i], false))
   }

  for i := range ras {
    as = append(as, subtract(bas[i], ras[i], false))
  }

  return s, as
}

func subtract(
  b, s [][]float64,
  sOutlier bool,
) (
  [][]float64,
) {

  //var shift float64

/*
  // Outlier applies to pump-only cooling data
  if sOutlier {
    shift = -(avg(s[1][:100]) - avg(b[1][:100]))
  } else {
    shift = -(avg(s[1][:100]) - avg(b[1][:100]))
  }
*/

  //shift = -(avg(s[1][:10]) - avg(b[1][:10]))
  //shift = -(s[1][0] - b[1][0])

  for i := range b[0] {
    s[1][i] = s[1][i] - b[1][i] //+ shift
  }
```

```
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[
          set]])
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[
          set]])
2053    }
2054  }
2055
2056  func goPlotSubGrpd(
2057    sets []int,
2058    s, as [][][]float64,
2059    σs, σas [][]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)"
```

```
  legend := ""

  if coolingExperiment == "pump-only" {
    legend = "Power"
  } else if coolingExperiment == "pump-probe" {
    legend = "Pump"
  }

  xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
    "fits", sample, coolingExperiment,
  )
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  p, t, r := prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xtick, ytick,
    xtickLabel, ytickLabel,
    slide,
  )

  p.Legend.Left = true
  p.Legend.XOffs = vg.Points(25)
  p.Legend.YOffs = vg.Points(-50)

  for _, set := range sets {

    asPts := buildData(as[set])
    σasErr := buildErrors(σas[set]) // σ[set][σi]

    antiStokes := errorPoints {
      XYs: asPts,
      YErrors: plotter.YErrors(σasErr),
    }

    // Make a scatter plotter and set its style.
    plotSet, err := plotter.NewScatter(antiStokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotSet.GlyphStyle.Color = palette(set, false, coolingExperiment)
    if slide {
      plotSet.GlyphStyle.Radius = vg.Points(5)
    } else {
      plotSet.GlyphStyle.Radius = vg.Points(3)
    }
    plotSet.Shape = draw.CircleGlyph{}

    // Error bars
    e, err := plotter.NewYErrorBars(antiStokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
    e.LineStyle.Color = palette(set, false, coolingExperiment)

    p.Add(e, t, r) // plotSet,

    // Legend
    l, err := plotter.NewScatter(antiStokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
```

```
    l.GlyphStyle.Color = palette(set, false, coolingExperiment)
    l.GlyphStyle.Radius = vg.Points(6)
    l.Shape = draw.CircleGlyph{}
    p.Legend.Add(strings.Trim(asLabel[set], " pras"), l)
  }

  savePlot(p, "Anti-Stokes Background Subtracted", logpath)

  // Stokes
  title = "Stokes"
  xlabel = "Frequency (GHz)"
  ylabel = "Spectral Density (uV)"

  if coolingExperiment == "pump-only" {
    legend = "Power"
  } else if coolingExperiment == "pump-probe" {
    legend = "Pump"
  }

  xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err = axes(
    "fits", sample, coolingExperiment,
  )
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  p, t, r = prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xtick, ytick,
    xtickLabel, ytickLabel,
    slide,
  )

  p.Legend.Left = true
  p.Legend.XOffs = vg.Points(25)
  p.Legend.YOffs = vg.Points(-50)

  for _, set := range sets {

    sPts := buildData(s[set])
    σsErr := buildErrors(σs[set]) // σ[set][σi]

    stokes := errorPoints {
      XYs: sPts,
      YErrors: plotter.YErrors(σsErr),
    }

    // Make a scatter plotter and set its style.
    plotSet, err := plotter.NewScatter(stokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotSet.GlyphStyle.Color = palette(set, true, coolingExperiment)
    if slide {
      plotSet.GlyphStyle.Radius = vg.Points(5)
    } else {
      plotSet.GlyphStyle.Radius = vg.Points(3)
    }
    plotSet.Shape = draw.CircleGlyph{}

    // Error bars
    e, err := plotter.NewYErrorBars(stokes)
    if err != nil {
      fmt.Println(err)
```

```
      os.Exit(1)
    }
    e.LineStyle.Color = palette(set, false, coolingExperiment)

    p.Add(e, t, r) // plotSet,

    // Legend
    l, err := plotter.NewScatter(stokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    l.GlyphStyle.Color = palette(set, true, coolingExperiment)
    l.GlyphStyle.Radius = vg.Points(6)
    l.Shape = draw.CircleGlyph{}

    p.Legend.Add(strings.Trim(sLabel[set], " rs"), l)
  }

  savePlot(p, "Stokes Background Subtracted", logpath)
}

func generateFitData(
  amp, wid, cen, c, f0, df float64,
  fitPts int,
) (
  [][]float64,
) {

  x := make([]float64, fitPts)
  for i := range x {
    x[i] = f0 + df*float64(i)
  }

  y := make([]float64, fitPts)
  for i := range x {
    // (amp*wid^2/((x-cen)^2+wid^2))
    y[i] = .25 * amp * math.Pow(wid, 2) / (math.Pow(x[i] - cen, 2) + (.25 * math.Pow(wid, 2)
    )) + c
  }

  return [][]float64{x, y}
}

func goPlotasFits(
  sets []int,
  as, fits, widthLines [][][]float64,
  σas [][]float64,
  labels []string,
  widths, notes []float64,
  temp, slide bool,
  sample, logpath, coolingExperiment string,
) {

  type errorPoints struct {
    plotter.XYs
    plotter.YErrors
  }

  title := " "
  xlabel := "Frequency (GHz)"
  ylabel := "Spectral Density (uV)"
  legend := ""

  if coolingExperiment == "pump-only" {
    legend = "Power"
  } else if coolingExperiment == "pump-probe" {
```

```
    legend = "Pump"
  }

  xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
    "fits", sample, coolingExperiment,
  )
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  p, t, r := prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xtick, ytick,
    xtickLabel, ytickLabel,
    slide,
  )

  p.Legend.Left = true
  p.Legend.XOffs = vg.Points(25)
  p.Legend.YOffs = vg.Points(-50)

  for i, set := range sets {

    pts := buildData(as[set])
    fit := buildData(fits[i])
    wid := buildData(widthLines[i])
    σasErr := buildErrors(σas[set]) // σ[set][σi]

    antiStokes := errorPoints {
      XYs: pts,
      YErrors: plotter.YErrors(σasErr),
    }

    // Plot points
    plotPts, err := plotter.NewScatter(antiStokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotPts.GlyphStyle.Color = palette(set, false, coolingExperiment)
    if slide {
      plotPts.GlyphStyle.Radius = vg.Points(5)
    } else {
      plotPts.GlyphStyle.Radius = vg.Points(3)
    }
    plotPts.Shape = draw.CircleGlyph{}

    // Plot fit
    plotFit, err := plotter.NewLine(fit)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotFit.LineStyle.Color = palette(set, true, coolingExperiment)
    plotFit.LineStyle.Width = vg.Points(3)

    // Width lines
    plotWid, err := plotter.NewLine(wid)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotWid.LineStyle.Color = palette(set, true, coolingExperiment)
    plotWid.LineStyle.Width = vg.Points(4)
```

```
    plotWid.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}

    // Y Error bars
    e, err := plotter.NewYErrorBars(antiStokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
    e.LineStyle.Color = palette(set, false, coolingExperiment)

    // Add set plots to p
    p.Add(e, t, r, plotFit) // , plotWid

    // Legend
    l, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    l.GlyphStyle.Color = palette(set, true, coolingExperiment)
    l.GlyphStyle.Radius = vg.Points(6)
    l.Shape = draw.CircleGlyph{}
    power := strings.Trim(labels[set], " pras")
    if temp {
      temp := strconv.FormatFloat(notes[set], 'f', -1, 64)
      p.Legend.Add(power + " @" + temp + "K", l)
    } else {
      p.Legend.Add(power, l)
    }

  }

  savePlot(p, "Anti-Stokes w Fits", logpath)
}

func plotSinc(
  sets []int,
  phaseMatchData [][]float64,
  label []string,
  sample, logpath string,
  length float64,
  slide bool,
) {

  pts := buildData(phaseMatchData)

  var l string

  switch length {
  case 0.0:
    l = ""
  case 0.001:
    l = "1 mm"
  case 0.01:
    l = "1 cm"
  case 0.004:
    l = "4 mm"
  case 0.0000005:
    l = "500 nm"
  case 0.00001:
    l = "10 µm"
  default:
    l = strconv.FormatFloat(length, 'f', 1, 64)
  }

  title := l + " " + sample + " Phase-Matching Bandwidth"
  xlabel := "Pump-Probe Separation (GHz)"
```

```
  ylabel := "Peak Spectral Density"
  legend := ""

  // Auto Axes
  xmin, xmax, ymin, ymax := 0., 0., 0., 0.
  for i, v := range phaseMatchData[0] {
    if v > xmax {
      xmax = v
    }
    if phaseMatchData[1][i] > ymax {
      ymax = phaseMatchData[1][i]
    }
  }

  xrange := []float64{xmin, xmax}
  yrange := []float64{ymin, ymax}

  xtick := 0.
  displayDigits := 2
  if (xmax - xmin)/8 > 5 {
    xtick = 5
  } else if (xmax - xmin)/8 > 2.5 {
    xtick = 2.5
  } else if (xmax - xmin)/8 > 1 {
    xtick = 1
  } else if (xmax - xmin)/8 > 0.5 {
    xtick = 0.5
  }
  firstTick := 0.
  for m := float64(int(xmin)); m <= xmin; m += xtick {
    firstTick = m
  }
  xticks := []float64{}
  xtickLabels := []string{}
  for i := 0.; firstTick + xtick*i <= xmax - xtick/2; i++ {
    xticks = append(xticks, firstTick + xtick*i)
    if int(i)%2 != 0 {
      xtickLabels = append(xtickLabels, strconv.FormatFloat(firstTick + xtick*i, 'f',
    displayDigits, 64))
    } else {
      xtickLabels = append(xtickLabels, "")
    }
  }

  ytick := ((ymax - ymin)/8)
  yticks := []float64{}
  ytickLabels := []string{}
  for i := 0.; i < 11; i++ {
    yticks = append(yticks, ytick*i + ymin)
    if int(i)%2 != 0 {
      ytickLabels = append(ytickLabels, strconv.FormatFloat(ytick*i + ymin, 'f', 2, 64))
    } else {
      ytickLabels = append(ytickLabels, "")
    }
  }
  yticks = append(yticks, ymax)
  ytickLabels = append(ytickLabels, "")

  p, t, r := prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xticks, yticks,
    xtickLabels, ytickLabels,
    slide,
  )

  scatter, err := plotter.NewScatter(pts)
  if err != nil {
    log.Fatalf("Could not create line for the first series: %v", err)
```

```
  }
  scatter.GlyphStyle.Color = palette(0, false, "")
  scatter.GlyphStyle.Radius = vg.Points(5) //3
  scatter.Shape = draw.CircleGlyph{}
  p.Add(scatter, t, r)

  savePlot(p, "Phase-Match", logpath)
}

func bin(
  sets []int,
  as, s [][][]float64,
  binMHz float64,
) (
  [][][]float64, [][][]float64,
) {

  binGHz := binMHz/1000
  nBins := int((as[0][0][len(as[0][0]) - 1] - as[0][0][0])/binGHz + 1)

  // [set][0: feq, 1: sig, 2: err][values]
  asBinned := make([][][]float64, len(as))
  for i := range asBinned {
    asBinned[i] = make([][]float64, 3)
    for j := range asBinned[i] {
      asBinned[i][j] = make([]float64, nBins)
    }
  }
  sBinned := make([][][]float64, len(s))
  for i := range sBinned {
    sBinned[i] = make([][]float64, 3)
    for j := range sBinned[i] {
      sBinned[i][j] = make([]float64, nBins)
    }
  }

  for _, set := range sets {

    asBound := as[set][0][0]
    sBound := s[set][0][0]

    for i := 0; i < nBins; i++ {
      asBound += binGHz
      sBound += binGHz

      var asSigsInBin, sSigsInBin []float64

      for j, f := range as[set][0] {
        if f < asBound && f > asBound - binGHz {
          asSigsInBin = append(asSigsInBin, as[set][1][j])
        }
      }
      asBinned[set][0][i] = asBound - (binGHz/2)
      asBinned[set][1][i] = avg(asSigsInBin)

      for j, f := range s[set][0] {
        if f < sBound && f > sBound - binGHz {
          sSigsInBin = append(sSigsInBin, s[set][1][j])
        }
      }
      sBinned[set][0][i] = sBound - (binGHz/2)
      sBinned[set][1][i] = avg(sSigsInBin)

      // Error for each binned point
      asBinned[set][2][i] = σ(asSigsInBin)
      sBinned[set][2][i] = σ(sSigsInBin)
    }
  }
```

```
  return asBinned, sBinned
}

func σCABS(
  setsToPlotCABS, numAvgs []int,
  cabsData [][][]float64,
  sigUnit string,
  sigmaMultiple float64,
  normalized []string,
) (
  [][][]float64,
) {

  N := "Number of Averages"

  if N == "Sampling Rate" {
    // Sampling rate = 1,842,000 usually
    // 1. figure σ from dwell-time σ in CSVs
    largestSet := setsToPlotCABS[0]
    for _, v := range setsToPlotCABS {
      if v > largestSet {
        largestSet = v
      }
    }

    stdDevBg := make([][][]float64, largestSet+1)
    stdDevSig := make([][][]float64, largestSet+1)

    for _, set := range setsToPlotCABS {

      stdDevBg[set] = make([][]float64, numAvgs[set])
      stdDevSig[set] = make([][]float64, numAvgs[set])

      for run := 0; run < numAvgs[set]; run++ {

        // Background
        // Open the CSV file for reading
        csvPath := "Data/" + fmt.Sprint(set+1) + "/Runs/Background/Run " + fmt.Sprint(run) +
    ".csv"
        file, err := os.Open(csvPath)
        if err != nil {
          fmt.Println("Error:", err)
          return cabsData
        }

        // Create a CSV reader
        reader := csv.NewReader(file)

        // Read all CSV records
        records, err := reader.ReadAll()
        if err != nil {
          fmt.Println("Error:", err)
          return cabsData
        }
        file.Close()

        // Iterate through the records (skip the first header row)
        for row, record := range records {
          if row == 0 {
            // Skip the header row
            continue
          }

          // Ensure there are at least 2 columns in the record
          if len(record) >= 2 {

            record[1] = strings.TrimSpace(record[1])
            // Convert the second column to a float64 and append to the slice
```

```
            values, err := strconv.ParseFloat(record[1], 64)
            if err != nil {
              fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
            } else {
              stdDevBg[set][run] = append(stdDevBg[set][run], values)
            }
          } else {
            fmt.Printf("Row %d does not have enough columns\n", row+1)
          }
        }

        // Signal
        // Open the CSV file for reading
        csvPath = "Data/" + fmt.Sprint(set+1) + "/Runs/Signal/Run " + fmt.Sprint(run) + ".
    csv"
        file, err = os.Open(csvPath)
        if err != nil {
          fmt.Println("Error:", err)
          return cabsData
        }

        // Create a CSV reader
        reader = csv.NewReader(file)

        // Read all CSV records
        records, err = reader.ReadAll()
        if err != nil {
          fmt.Println("Error:", err)
          return cabsData
        }
        file.Close()

        // Iterate through the records (skip the first header row)
        for row, record := range records {
          if row == 0 {
            // Skip the header row
            continue
          }

          // Ensure there are at least 2 columns in the record
          if len(record) >= 2 {

            record[1] = strings.TrimSpace(record[1])
            // Convert the second column to a float64 and append to the slice
            values, err := strconv.ParseFloat(record[1], 64)
            if err != nil {
              fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
            } else {
              stdDevSig[set][run] = append(stdDevSig[set][run], values)
            }
          } else {
            fmt.Printf("Row %d does not have enough columns\n", row+1)
          }
        }
      }
    }

    // combine σ for each freq across runs
    for _, set := range setsToPlotCABS {

      σCombinedAcrossRunsBg := make([]float64, len(stdDevBg[set][0]))
      σCombinedAcrossRunsSig := make([]float64, len(stdDevBg[set][0]))

      for i := range stdDevBg[set][0] {

        for run := 0; run < numAvgs[set]; run++ {

          // !assumes sig and background have same # of runs within a set
```

```
          σCombinedAcrossRunsBg[i] += math.Pow(stdDevBg[set][run][i], 2)
          σCombinedAcrossRunsSig[i] += math.Pow(stdDevSig[set][run][i], 2)
        }
      }

      // square root of sum of squares, divided by number of runs
      for i := range σCombinedAcrossRunsBg {

        σCombinedAcrossRunsBg[i] = math.Sqrt(σCombinedAcrossRunsBg[i])/float64(numAvgs[set])
        σCombinedAcrossRunsSig[i] = math.Sqrt(σCombinedAcrossRunsSig[i])/float64(numAvgs[set
    ])
      }

      // propagate errors through signal - background

      σSigMinusBg := make([]float64, len(σCombinedAcrossRunsSig))

      for i, v := range σCombinedAcrossRunsSig {

        σSigMinusBg[i] += math.Sqrt(math.Pow(v, 2) + math.Pow(σCombinedAcrossRunsBg[i], 2))

        // only scale to appropriate unit if not being normalized later
        if len(normalized) > 0 {
          switch sigUnit {
          case "mV":
            σSigMinusBg[i] *= 1e3
          case "uV":
            σSigMinusBg[i] *= 1e6
          case "nV":
            σSigMinusBg[i] *= 1e9
          case "pV":
            σSigMinusBg[i] *= 1e12
          }
        }

        // 1σ = 68.27%, 2σ = 95.45%, 3σ = 99.73%
        //σSigMinusBg[i] = σSigMinusBg[i]*2

      }

      // 2. tack errors onto cabsData
      // cabsData[set][0: freq, 1: sig, 2: σ][rows of freq/sig/σ]
      cabsData[set] = append(cabsData[set], σSigMinusBg)
    }
  } else if N == "Number of Averages" {

    // Number of Averages = 5 usually
    // Calculate σ across subtracted runs
    largestSet := setsToPlotCABS[0]
    for _, v := range setsToPlotCABS {
      if v > largestSet {
        largestSet = v
      }
    }

    runVals := make([][][]float64, largestSet+1)

    for _, set := range setsToPlotCABS {

      runVals[set] = make([][]float64, numAvgs[set])

      for run := 0; run < numAvgs[set]; run++ {

        // Open the CSV file for reading
        csvPath := "Data/" + fmt.Sprint(set+1) + "/Runs/Subtracted/Run " + fmt.Sprint(run) +
     ".csv"
        file, err := os.Open(csvPath)
        if err != nil {
```

```
          fmt.Println("Error:", err)
          return cabsData
        }

        // Create a CSV reader
        reader := csv.NewReader(file)

        // Read all CSV records
        records, err := reader.ReadAll()
        if err != nil {
          fmt.Println("Error:", err)
          return cabsData
        }
        file.Close()

        // Iterate through the records
        for row, record := range records {

        record[0] = strings.TrimSpace(record[0])
          // Convert the column to a float64 and append to the slice
          values, err := strconv.ParseFloat(record[0], 64)
          if err != nil {
            fmt.Printf("Error parsing value in row %d: %v\n", row+1, err)
          } else {
            runVals[set][run] = append(runVals[set][run], values)
          }
        }
      }

      transposedRunVals := transpose(runVals[set])

      stdDev := make([]float64, len(runVals[set][0]))

      for i, v := range transposedRunVals {

        stdDev[i] = σ(v)*sigmaMultiple
      }

      // only scale to appropriate unit if not being normalized later
      if !(len(normalized) > 0) {
        for i := range stdDev {
          switch sigUnit {
          case "mV":
            stdDev[i] *= 1e3
          case "µV":
            stdDev[i] *= 1e6
          case "nV":
            stdDev[i] *= 1e9
          case "pV":
            stdDev[i] *= 1e12
          default:
            fmt.Printf("Warning: Unrecognized signal unit '%s' for standard deviation
    scaling.\n", sigUnit)
          }
        }
      }

      cabsData[set] = append(cabsData[set], stdDev)

    }
  }

  return cabsData
}

func contains(
  list []string, a string,
) (
```

```
  bool,
) {
    for _, b := range list {
        if b == a {
            return true
        }
    }
    return false
}

func normalizeByPowers(
  setsToPlotCABS []int,
  cabsData [][][]float64,
  pumpPowers, stokesPowers, probePowers []float64,
) (
  [][][]float64,
) {

  for _, set := range setsToPlotCABS {

    // mW -> W
    pumpPowers[set] /= 1e3
    stokesPowers[set] /= 1e3
    probePowers[set] /= 1e3

    // sig
    for i := range cabsData[set][1] {

      cabsData[set][1][i] /= pumpPowers[set]*stokesPowers[set]*probePowers[set]
    }

    // σ
    for i := range cabsData[set][2] {

      cabsData[set][2][i] /= pumpPowers[set]*stokesPowers[set]*probePowers[set]
    }
  }

  return cabsData
}

func FitLorentzian(
  frequencies, signals, uncertainties, initialParams []float64,
) (
  []float64,
) {

  // Define the residual function
  resFunc := func(dst, params []float64) {
      r := residuals(params, frequencies, signals, uncertainties)
      for i := range r {
          dst[i] = r[i]
      }
  }

  // Define NumJac instance
  nj := &lm.NumJac{Func: resFunc}

  // Problem definition
  problem := lm.LMProblem{
      Dim:        4,
      Size:       len(frequencies),
      Func:       resFunc,
      Jac:        nj.Jac, // Use the numerical Jacobian
      InitParams: initialParams,
      Tau:        1e-6,
      Eps1:       1e-8,
      Eps2:       1e-8,
```

```
  }

  settings := &lm.Settings{Iterations: 1000, ObjectiveTol: 1e-16}

  result, err := lm.LM(problem, settings)
  if err != nil {
      log.Fatal("optimization failed:", err)
  }

  return result.X
}

func Lorentzian(
  f, A, f0, gamma, C float64,
) (
  float64,
) {
    return .25 * A * math.Pow(gamma, 2) / (math.Pow(f - f0, 2) + (.25 * math.Pow(gamma, 2))) + C
    //return (A / math.Pi) * (gamma / (math.Pow(f-f0, 2) + math.Pow(gamma, 2))) + C
}

func residuals(
  params, frequencies, signals, uncertainties []float64,
) (
  []float64,
) {

    A, f0, gamma, C := params[0], params[1], params[2], params[3]
    r := make([]float64, len(frequencies))

    for i, f := range frequencies {
        modelValue := Lorentzian(f, A, f0, gamma, C)
        if len(uncertainties) > 0 && uncertainties[i] != 0 {
            r[i] = (signals[i] - modelValue) / uncertainties[i]
        } else {
            r[i] = signals[i] - modelValue
        }
    }

    return r
}

func binCabs(
  setsToBin []int,
  cabsData [][][]float64,
  binMHz float64,
) (
  [][][]float64,
) {
  binGHz := binMHz/1000
  nBins := make([]int, len(cabsData))
  for set := range cabsData {
    yesBin := false
    for _, setToBin := range setsToBin {
      if set == setToBin {
        yesBin = true
      }
    }

    if yesBin {
      n := int((cabsData[set][0][len(cabsData[set][0]) - 1] - cabsData[set][0][0])/binGHz + 1)
      nBins[set] = n
    } else {
      nBins[set] = len(cabsData[set][0])
    }
  }
```

```
  // [set][0: feq, 1: sig, 2: err][values]
  cabsBinned := make([][][]float64, len(cabsData))
  for set := range cabsBinned {

    yesBin := false
    for _, setToBin := range setsToBin {
      if set == setToBin {
        yesBin = true
      }
    }

    if yesBin {
      cabsBinned[set] = make([][]float64, 3) // freq, sig, err
      for i := range cabsBinned[set] {
        cabsBinned[set][i] = make([]float64, nBins[set]) // len(cabsData[set][0])
      }
    } else {
      cabsBinned[set] = make([][]float64, 2) // freq, sig, err
      for i := range cabsBinned[set] {
        cabsBinned[set][i] = make([]float64, nBins[set])
      }
    }
  }

  for set := range cabsData {

    yesBin := false
    for _, setToBin := range setsToBin {
      if set == setToBin {
        yesBin = true
      }
    }

    if yesBin {
      cabsBound := cabsData[set][0][0]

      for i := 0; i < nBins[set]; i++ {
        cabsBound += binGHz

        var cabsSigsInBin []float64

        for j, f := range cabsData[set][0] {
          if f < cabsBound && f > cabsBound - binGHz {
            cabsSigsInBin = append(cabsSigsInBin, cabsData[set][1][j])
          }
        }
        cabsBinned[set][0][i] = cabsBound - (binGHz/2)
        cabsBinned[set][1][i] = avg(cabsSigsInBin)

        // Error for each binned point
        cabsBinned[set][2][i] = σCABSBins(cabsSigsInBin)
        }
      } else {
      for i, v := range cabsData[set][0] {
        cabsBinned[set][0][i] = v
        cabsBinned[set][1][i] = cabsData[set][1][i]
      }
    }
  }

  return cabsBinned
}

func transpose(
  slice [][]float64,
) (
  [][]float64,
```

```
) {

  rows := len(slice)
  cols := len(slice[0])

  transposed := make([][]float64, cols)

  for i := 0; i < cols; i++ {
    transposed[i] = make([]float64, rows)
    for j := 0; j < rows; j++ {
      transposed[i][j] = slice[j][i]
    }
  }

  return transposed
}

func σ(
  values []float64,
) (
  float64,
) {

  /* dBm -> uV
  for i, v := range values {
    values[i] = math.Pow(10, 6)*math.Pow(10, v/10.)
  }*/

  // Sum of squares of the difference
  dev := 0.
  for _, v := range values {
    dev += math.Pow(v - avg(values), 2)
  }
  n := float64(len(values))

  // Standard deviation of the mean
  return math.Sqrt((1/(n - 1) * dev))/math.Sqrt(n)
}

func σCABSBins(
  values []float64,
) (
  float64,
) {

  // Sum of squares of the difference
  dev := 0.
  for _, v := range values {
    dev += math.Pow(v - avg(values), 2)
  }
  n := float64(len(values))

  // Standard deviation of the mean
  return math.Sqrt((1/(n - 1) * dev))/math.Sqrt(n)
}

func avg(
  toAvg []float64,
) (
  float64,
) {

  sum := 0.
  for _, v := range toAvg {
    sum += v
  }
  return sum/float64(len(toAvg))
}
```

```
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    σs [][]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
```

```
  p, t, r := prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xtick, ytick,
    xtickLabel, ytickLabel,
    slide,
  )

  p.Legend.Left = true
  p.Legend.XOffs = vg.Points(25)
  p.Legend.YOffs = vg.Points(-50)

  for i, set := range sets {

    pts := buildData(s[set])
    fit := buildData(fits[i])
    wid := buildData(widthLines[i])
    σsErr := buildErrors(σs[set]) // σ[set][σi]

    stokes := errorPoints {
      XYs: pts,
      YErrors: plotter.YErrors(σsErr),
    }

    // Plot points
    plotPts, err := plotter.NewScatter(stokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotPts.GlyphStyle.Color = palette(set, false, coolingExperiment)
    if slide {
      plotPts.GlyphStyle.Radius = vg.Points(5)
    } else {
      plotPts.GlyphStyle.Radius = vg.Points(3)
    }
    plotPts.Shape = draw.CircleGlyph{}

    // Plot fit
    plotFit, err := plotter.NewLine(fit)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotFit.LineStyle.Color = palette(set, true, coolingExperiment)
    plotFit.LineStyle.Width = vg.Points(3)

    // Width lines
    plotWid, err := plotter.NewLine(wid)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotWid.LineStyle.Color = palette(set, true, coolingExperiment)
    plotWid.LineStyle.Width = vg.Points(4)
    plotWid.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}

    // Error bars
    e, err := plotter.NewYErrorBars(stokes)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
    e.LineStyle.Color = palette(set, false, coolingExperiment)

    // Add set plots to p
```

```
    p.Add(e, t, r, plotFit) // , plotWid

    // Legend
    l, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    l.GlyphStyle.Color = palette(set, true, coolingExperiment)
    l.GlyphStyle.Radius = vg.Points(6)
    l.Shape = draw.CircleGlyph{}
    power := strings.Trim(labels[set], " prs")
    if temp {
      temperature := strconv.FormatFloat(notes[set], 'f', -1, 64)
      p.Legend.Add(power + " @" + temperature + "K", l)
    } else {
      p.Legend.Add(power, l)
    }
  }

  savePlot(p, "Stokes w Fits", logpath)
}

func goPlotsPowerVsWid(
  sets []int,
  labels []string,
  notes, widths []float64,
  temp, slide bool,
  sample, logpath, coolingExperiment string,
) {

  title := "Stokes Pump Power vs Widths of Fits"
  xlabel := "Pump Power (mW)"
  ylabel := "Full Width Half Max (MHz)"
  legend := ""

  xrange, yrange, xtick, ytick, xtickLabel, ytickLabel, err := axes(
    "pow vs wid", sample, coolingExperiment,
  )
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  p, t, r := prepPlot(
    title, xlabel, ylabel, legend,
    xrange, yrange, xtick, ytick,
    xtickLabel, ytickLabel,
    slide,
  )

  for i, set := range sets {

    pts := make(plotter.XYs, 1)

    power := strings.Trim(labels[set], " mW prs")
    if pwr, err := strconv.ParseFloat(power, 64); err == nil {
      pts[0].X = pwr
    } else {
      panic(err)
    }
    pts[0].Y = widths[i]

    // Plot points
    plotPts, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
```

```
      os.Exit(1)
    }

    plotPts.GlyphStyle.Color = palette(set, true, coolingExperiment)
    plotPts.GlyphStyle.Radius = vg.Points(6)
    plotPts.Shape = draw.CircleGlyph{}

    // Dashed eye guide lines
    v := make(plotter.XYs, 2)
    h := make(plotter.XYs, 2)

    // Vertical
    v[0].X = pts[0].X
    v[0].Y = yrange[0]
    v[1].X = pts[0].X
    v[1].Y = pts[0].Y

    vDash, err := plotter.NewLine(v)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    vDash.LineStyle.Color = palette(set, true, coolingExperiment)
    vDash.LineStyle.Width = vg.Points(4)
    vDash.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}

    // Horizontal
    h[0].X = xrange[0]
    h[0].Y = pts[0].Y
    h[1].X = pts[0].X
    h[1].Y = pts[0].Y

    hDash, err := plotter.NewLine(h)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    hDash.LineStyle.Color = color.RGBA{R: 127, G: 127, B: 127, A: 255}
    hDash.LineStyle.Width = vg.Points(1)
    hDash.LineStyle.Dashes = []vg.Length{vg.Points(5), vg.Points(5)}

    // Add set plots to p
    p.Add(plotPts, t, r, vDash, hDash)
    if temp {
      temperature := strconv.FormatFloat(notes[set], 'f', -1, 64)
      p.Legend.Add(power + " mW @" + temperature + "K", plotPts)
    } else {
      p.Legend.Add(power + " mW")
    }
  }

  savePlot(p, "s Pow vs Wid", logpath)
}

func goPlotHeightRatios(
  sets []int,
  heightRatios, powers []float64,
  labels []string,
  sample, logpath, coolingExperiment string,
  slide bool,
) {

  title := " " // Height Ratios vs Power
  xlabel := "Pump Power (mW)"
  ylabel := "Stokes/Anti-Stokes Heights"
  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)
```

```
    yFit = append(yFit, m * x + b)
  }

  fit := buildData([][]float64{xFit, yFit})

  // Plot fit
  plotFit, err := plotter.NewLine(fit)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  p.Add(plotFit, t, r)

  plotFit.LineStyle.Color = color.RGBA{R: 127, G: 127, B: 127, A: 255}
  plotFit.LineStyle.Width = vg.Points(3)

  for i, set := range sets {

    pts := make(plotter.XYs, 1)

    pts[0].X = powers[set]
    pts[0].Y = heightRatios[i]

    // Plot points
    plotPts, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    plotPts.GlyphStyle.Color = color.RGBA{R: 27, G: 170, B: 139, A: 255}
    plotPts.GlyphStyle.Radius = vg.Points(6)
    plotPts.Shape = draw.CircleGlyph{}

    // Add set plots to p
    p.Add(plotPts)
    //p.Legend.Add(strings.Trim(labels[set], " prs"), plotPts)
  }

  savePlot(p, "height ratios", logpath)
}

func Γeff(
  maxPow float64,
  Γ, length, gb float64,
  coolingExperiment string,
) (
  [][]float64, [][]float64,
) {
  // Γ_as,eff = 2*pi*Γ*(1 + GPL/4)
  // Γ_s,eff = 2*pi*Γ*(1 - GPL/4)
  var pow []float64

  if coolingExperiment == "pump-only" {
    pow = []float64{0, maxPow}
  } else if coolingExperiment == "pump-probe" {
    pow = []float64{-10, maxPow}
  }

  ΓasEff := []float64{Γ, Γ*(1 + gb*pow[1]*.001*length/(4*2*math.Pi))}
  ΓsEff := []float64{Γ, Γ*(1 - gb*pow[1]*.001*length/(4*2*math.Pi))}

  fmt.Printf("\nΓasEff: %.4f\n", ΓasEff[1])
  fmt.Printf("ΓsEff: %.4f\n", ΓsEff[1])

  return [][]float64{pow, ΓasEff}, [][]float64{pow, ΓsEff}
}
```

```
3561
3562 func goPlotLinewidths(
3563   sets []int,
3564   Γ asEff, Γ sEff [][]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})
```

```
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  m, b = results.X[0], results.X[1]

  var asyFit []float64
  var asxFit []float64

  // Create function according to solved fit parameters
  for _, set := range sets {
    var x float64

    x = asPowers[set]

    asxFit = append(asxFit, x)
    asyFit = append(asyFit, m * x + b)
  }

  asfit := buildData([][]float64{asxFit, asyFit})

  // Plot as fit
  asPlotFit, err := plotter.NewLine(asfit)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  // s linear fit

  f = func(dst, guess []float64) {

    var x float64
    m, b := guess[0], guess[1]

    for i, set := range sets {

      x = sPowers[set]
      y := sLinewidths[i]

      dst[i] = m * x + b - y
    }
  }

  jacobian = lm.NumJac{Func: f}

  // Solve for fit
  toBeSolved = lm.LMProblem{
    Dim:        2,
    Size:       len(sets),
    Func:       f,
    Jac:        jacobian.Jac,
    InitParams: []float64{m, b},
    Tau:        1e-6,
    Eps1:       1e-8,
    Eps2:       1e-8,
  }

  results, err = lm.LM(toBeSolved, &lm.Settings{Iterations: 100, ObjectiveTol: 1e-16})
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  m, b = results.X[0], results.X[1]

  var syFit []float64
```

```
  var sxFit []float64

  // Create function according to solved fit parameters
  for _, set := range sets {
    var x float64

    x = sPowers[set]

    sxFit = append(sxFit, x)
    syFit = append(syFit, m * x + b)
  }

  sfit := buildData([][]float64{sxFit, syFit})

  // Plot s fit
  sPlotFit, err := plotter.NewLine(sfit)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  sPlotFit.LineStyle.Color = color.RGBA{R: 201, G: 104, B: 146, A: 255}
  sPlotFit.LineStyle.Width = vg.Points(3)

  asPlotFit.LineStyle.Color = color.RGBA{R: 99, G: 124, B: 198, A: 255}
  asPlotFit.LineStyle.Width = vg.Points(3)

  // Plot Γeff
  ΓasEffPlot, err := plotter.NewLine(buildData(ΓasEff))
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  ΓasEffPlot.LineStyle.Color = color.NRGBA{R: 99, G: 124, B: 198, A: 125} // R: 0, G: 89, B:
     128, A: 255
  ΓasEffPlot.LineStyle.Width = vg.Points(3)
  ΓasEffPlot.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}

  ΓsEffPlot, err := plotter.NewLine(buildData(ΓsEff))
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  ΓsEffPlot.LineStyle.Color = color.NRGBA{R: 201, G: 104, B: 146, A: 125} // R: 0, G: 89, B:
     128, A: 255
  ΓsEffPlot.LineStyle.Width = vg.Points(3)
  ΓsEffPlot.LineStyle.Dashes = []vg.Length{vg.Points(15), vg.Points(5)}

  p.Add(asPlotFit, t, r, sPlotFit, ΓasEffPlot, ΓsEffPlot)
  p.Legend.Add("Anti-Stokes Fit", asPlotFit)
  p.Legend.Add("Γ as,eff", ΓasEffPlot)
  p.Legend.Add("Stokes Fit", sPlotFit)
  p.Legend.Add("Γ s,eff", ΓsEffPlot)

  // as points
  for i, set := range sets {

    pts := make(plotter.XYs, 1)

    pts[0].X = asPowers[set]
    pts[0].Y = asLinewidths[i]

    // Plot points
    asPlotPts, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
```

```
      os.Exit(1)
    }

    asPlotPts.GlyphStyle.Color = color.RGBA{R: 99, G: 124, B: 198, A: 255}
    asPlotPts.GlyphStyle.Radius = vg.Points(6)
    asPlotPts.Shape = draw.CircleGlyph{}

    // Add set plots to p
    p.Add(asPlotPts)
  }

  // s points
  for i, set := range sets {

    pts := make(plotter.XYs, 1)

    pts[0].X = sPowers[set]
    pts[0].Y = sLinewidths[i]

    // Plot points
    sPlotPts, err := plotter.NewScatter(pts)
    if err != nil {
      fmt.Println(err)
      os.Exit(1)
    }

    sPlotPts.GlyphStyle.Color = color.RGBA{R: 201, G: 104, B: 146, A: 255}
    sPlotPts.GlyphStyle.Radius = vg.Points(6)
    sPlotPts.Shape = draw.CircleGlyph{}

    // Add set plots to p
    p.Add(sPlotPts)
  }

  savePlot(p, "linewidths", logpath)
}

func prepPlot(
  title, xlabel, ylabel, legend string,
  xrange, yrange, xtick, ytick []float64,
  xtickLabels, ytickLabels []string,
  slide bool,
) (
  *plot.Plot,
  *plotter.Line, *plotter.Line,
) {

  p := plot.New()
  p.BackgroundColor = color.RGBA{A:0}
  p.Title.Text = title
  p.Title.TextStyle.Font.Typeface = "liberation"
  p.Title.TextStyle.Font.Variant = "Sans"

  p.X.Label.Text = xlabel
  p.X.Label.TextStyle.Font.Variant = "Sans"
  p.X.LineStyle.Width = vg.Points(1.5)
  p.X.Min = xrange[0]
  p.X.Max = xrange[1]
  p.X.Tick.LineStyle.Width = vg.Points(1.5)
  p.X.Tick.Label.Font.Variant = "Sans"

  xticks := []plot.Tick{}
  for i, v := range xtick {
    xticks = append(xticks, plot.Tick{Value: v, Label: xtickLabels[i]})
  }

  p.X.Tick.Marker = plot.ConstantTicks(xticks)
  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)
```

```
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  // Right
  r[0].X = xrange[1]
  r[0].Y = yrange[0]
  r[1].X = xrange[1]
  r[1].Y = yrange[1]

  rAxis, err := plotter.NewLine(r)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  return p, tAxis, rAxis
}

func palette(
  brush int,
  dark bool,
  coolingExperiment string,
) (
  color.RGBA,
) {

  if coolingExperiment == "pump-probe" {
    if dark {
      darkColor := make([]color.RGBA, 16)
      darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
      darkColor[1] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
      darkColor[2] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
      darkColor[3] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
      darkColor[4] = color.RGBA{R: 7, G: 150, B: 189, A: 255}
      darkColor[5] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
      darkColor[6] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
      darkColor[7] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
      darkColor[8] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
      darkColor[9] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
      darkColor[10] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
      darkColor[11] = color.RGBA{R: 194, G: 140, B: 86, A: 255}
      darkColor[12] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
      darkColor[13] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
      darkColor[14] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
      darkColor[15] = color.RGBA{R: 194, G: 140, B: 86, A: 255}

      return darkColor[brush]
    }

    col := make([]color.RGBA, 16)
    col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
    col[1] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
    col[2] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
    col[3] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
    col[4] = color.RGBA{R: 11, G: 191, B: 222, A: 255}
    col[5] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
    col[6] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
    col[7] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
    col[8] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
    col[9] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
    col[10] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
    col[11] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
    col[12] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
    col[13] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
    col[14] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
    col[15] = color.RGBA{R: 255, G: 182, B: 110, A: 255}
```

```
    return col[brush]

  } else if coolingExperiment == "pump-only" {
    if dark {
      darkColor := make([]color.RGBA, 21)
      darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
      darkColor[4] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
      darkColor[8] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
      darkColor[12] = color.RGBA{R: 183, G: 139, B: 89, A: 255}
      darkColor[15] = color.RGBA{R: 18, G: 102, B: 99, A: 255}
      darkColor[1] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
      darkColor[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
      darkColor[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
      darkColor[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
      darkColor[2] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
      darkColor[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
      darkColor[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
      darkColor[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
      darkColor[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
      darkColor[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
      darkColor[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
      darkColor[16] = color.RGBA{R: 36, G: 117, B: 100, A: 255}
      darkColor[17] = color.RGBA{R: 117, G: 85, B: 41, A: 255}
      darkColor[18] = color.RGBA{R: 86, G: 17, B: 22, A: 255}
      darkColor[19] = color.RGBA{R: 17, G: 39, B: 91, A: 255}
      darkColor[20] = color.RGBA{R: 54, G: 12, B: 66, A: 255}

      return darkColor[brush]
    }

    col := make([]color.RGBA, 21)
    col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
    col[4] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
    col[8] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
    col[12] = color.RGBA{R: 255, G: 193, B: 122, A: 255}
    col[15] = color.RGBA{R: 27, G: 150, B: 146, A: 255}
    col[1] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
    col[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
    col[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
    col[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
    col[2] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
    col[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
    col[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
    col[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
    col[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
    col[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
    col[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
    col[16] = color.RGBA{R: 36, G: 117, B: 100, A: 255}
    col[17] = color.RGBA{R: 117, G: 85, B: 41, A: 255}
    col[18] = color.RGBA{R: 86, G: 17, B: 22, A: 255}
    col[19] = color.RGBA{R: 17, G: 39, B: 91, A: 255}
    col[20] = color.RGBA{R: 54, G: 12, B: 66, A: 255}

    return col[brush]
  }

  if dark {
    darkColor := make([]color.RGBA, 17)
    darkColor[0] = color.RGBA{R: 27, G: 170, B: 139, A: 255}
    darkColor[1] = color.RGBA{R: 201, G: 104, B: 146, A: 255}
    darkColor[2] = color.RGBA{R: 99, G: 124, B: 198, A: 255}
    darkColor[12] = color.RGBA{R: 183, G: 139, B: 89, A: 255}
    darkColor[15] = color.RGBA{R: 18, G: 102, B: 99, A: 255}
    darkColor[4] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
    darkColor[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
    darkColor[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
    darkColor[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
```

```
    darkColor[8] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
    darkColor[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
    darkColor[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
    darkColor[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
    darkColor[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
    darkColor[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
    darkColor[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
    darkColor[16] = color.RGBA{R: 255, G: 102, B: 102, A: 255}

    return darkColor[brush % len(darkColor)]
  }

  col := make([]color.RGBA, 17)
  col[0] = color.RGBA{R: 31, G: 211, B: 172, A: 255}
  col[1] = color.RGBA{R: 255, G: 122, B: 180, A: 255}
  col[2] = color.RGBA{R: 122, G: 156, B: 255, A: 255}
  col[12] = color.RGBA{R: 255, G: 193, B: 122, A: 255}
  col[15] = color.RGBA{R: 27, G: 150, B: 146, A: 255}
  col[4] = color.RGBA{R: 188, G: 117, B: 255, A: 255}
  col[5] = color.RGBA{R: 234, G: 156, B: 172, A: 255}
  col[6] = color.RGBA{R: 1, G: 56, B: 84, A: 255}
  col[7] = color.RGBA{R: 46, G: 140, B: 60, A: 255}
  col[8] = color.RGBA{R: 140, G: 46, B: 49, A: 255}
  col[9] = color.RGBA{R: 122, G: 41, B: 104, A: 255}
  col[10] = color.RGBA{R: 41, G: 122, B: 100, A: 255}
  col[11] = color.RGBA{R: 122, G: 90, B: 41, A: 255}
  col[3] = color.RGBA{R: 91, G: 22, B: 22, A: 255}
  col[13] = color.RGBA{R: 22, G: 44, B: 91, A: 255}
  col[14] = color.RGBA{R: 59, G: 17, B: 66, A: 255}
  col[16] = color.RGBA{R: 255, G: 102, B: 102, A: 255}

  return col[brush % len(col)]
}

func savePlot(
  p *plot.Plot,
  name, logpath string,
) {

  date := time.Now()

  // Make current date folder if it doesn't already exist
  if _, err := os.Stat("plots/" + date.Format("2006-Jan-02")); os.IsNotExist(err) {
    if err := os.Mkdir("plots/" + date.Format("2006-Jan-02"), 0755); err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
  }

  // Make current time folder if it doesn't already exist
  if _, err := os.Stat(logpath); os.IsNotExist(err) {
    if err := os.Mkdir(logpath, 0755); err != nil {
      fmt.Println(err)
      fmt.Println(logpath)
      os.Exit(1)
    }
  }

  path := logpath + "/" + name

  if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".png"); err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".svg"); err != nil {
    fmt.Println(err)
    os.Exit(1)
```

```
  }

  if err := p.Save(15*vg.Inch, 15*vg.Inch, path + ".pdf"); err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
}

/*func pngsToGIF(
  pngPaths []string,
  gifPath string,
) (
  error,
) {

    var frames []*image.Paletted
    var delays []int

    for _, fname := range pngPaths {
        // Open the PNG file
        f, err := os.Open(fname)
        if err != nil {
            return err
        }

        // Decode the PNG
        img, err := png.Decode(f)
        if err != nil {
            return err
        }
        f.Close()

        // Convert the image to Paletted
        palettedImage := image.NewPaletted(img.Bounds(), ipalette.Plan9)
        idraw.Draw(palettedImage, img.Bounds(), img, image.Point{}, idraw.Over)

        frames = append(frames, palettedImage)
        delays = append(delays, 10) // Add a delay for this frame
    }

    // Save as a GIF
    outFile, err := os.Create(gifPath)
    if err != nil {
        return err
    }
    defer outFile.Close()

    return gif.EncodeAll(outFile, &gif.GIF{
        Image: frames,
        Delay: delays,
    })
}*/

func normalizeFit(
  fit []float64,
) (
  []float64,
) {

  var shift float64 = (fit[0] + fit[599])/2

  for i := range fit {
    fit[i] = fit[i] - shift
  }
  return fit
}

func writeLog(
```

```
  logpath string,
  logFile []string,
) {

  date := time.Now()

  // Make current date folder if it doesn't already exist
  if _, err := os.Stat("plots/" + date.Format("2006-Jan-02")); os.IsNotExist(err) {
    if err := os.Mkdir("plots/" + date.Format("2006-Jan-02"), 0755); err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
  }

  // Make current time folder if it doesn't already exist
  if _, err := os.Stat(logpath); os.IsNotExist(err) {
    if err := os.Mkdir(logpath, 0755); err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
  }

  txt, err := os.Create(logpath + "/log.txt")
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }

  w := bufio.NewWriter(txt)
  defer w.Flush()
  for _, line := range logFile {
    if _, err := w.WriteString(line); err != nil {
      fmt.Println(err)
      os.Exit(1)
    }
  }
}
```

Appendix C

Supplementary Information for Chapter 3: Manuscript I

References

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