

rss_ringoccs: User Guide

V1.2

©Team Cassini at Wellesley College
Richard G. French*, Sophia R. Flury, Jolene W. Fong,
Ryan J. Maguire, and Glenn J. Steranka

**Cassini Radio Science Team Leader,
rfrench@wellesley.edu*

June 29, 2019

Contents

1	Introduction	1
1.1	Getting help	1
1.2	What is an RSS ring occultation?	2
1.3	Overview of Cassini RSS ring observations	2
1.4	Cassini RSS ring occultation observations on NASA's PDS	3
1.4.1	Raw RSS data files	3
1.4.2	Higher-level products	3
1.5	Required and recommended reading	4
1.5.1	Cassini Radio Science User's Guide	4
1.5.2	Marouf, Tyler, and Rosen (1986) - MTR86	4
1.5.3	Online Documentation	4
1.5.4	For more information...	4
2	Setting things up	4
2.1	System requirements	5
2.2	Downloading the <code>rss_ringoccs</code> repository from GitHub	5
2.3	Install Python 3 and required packages	5
2.3.1	Download and install <code>spiceypy</code>	6
2.3.2	Test <code>spiceypy</code>	6
2.4	Downloading Necessary Files	6
2.4.1	JPL/NAIF SPICE kernels	6
2.4.2	Cassini RSS raw data files	7
2.5	Hard Drive Space	8
2.6	Install A \LaTeX Compiler	9
3	Using <code>rss_ringoccs</code>	9
3.1	End-to-end pipeline outline	10
3.2	Conventions and hierarchy	12
3.3	End-to-end pipeline: A look at the Huygens ringlet	13
3.4	Quick-look method: Comparing profiles reconstructed at different resolutions	15
3.5	Batch Scripts	15
3.6	Profile Resolution	17
4	A detailed look at <code>rss_ringoccs</code>	19
4.1	RSR reader	19
4.2	Occultation geometry routines	19
4.3	Calibration routines	21
4.3.1	Frequency Offset	21
4.3.2	Power Normalization	23
4.4	Diffraction-limited profile routines	24
4.4.1	Threshold Optical Depth	26
4.5	Diffraction reconstruction routines	27
4.6	Utility routines	30
4.6.1	PDS3Reader	31
4.6.2	Label History	32
4.6.3	Science tools	32
4.7	Incoherent signal routines	32

5	Validation of <code>rss_ringoccs</code> algorithms	33
5.1	Comparison with results on PDS	33
5.1.1	Cassini RSS results	33
5.2	Effect of different sampling rates	37
6	Voyager 2 Uranus occultation	38
6.1	Changes to the procedure	39
6.1.1	Changes to reading raw data	39
6.1.2	Changes to Geometry	39
6.1.3	Changes to Calibration	40
6.1.4	Changes to DLP	41
6.1.5	Changes to Reconstruction	41
6.2	Validation of Uranus processing	41
7	Where to go from here	42
7.1	The Cassini RSS data catalog	42
7.2	Selecting an RSR file to process	42
7.3	Choosing a radial resolution	43
7.4	Execution time benchmarks	43
7.5	Licensing	44
7.6	Citing <code>rss_ringoccs</code>	44
	Acknowledgements	44
	A Meta-kernel file	45
	B Calibration Output Plots	45
	C Interactive Mode	46
	Acronyms	51
	Glossary	52

List of Figures

1	Ring opening angle vs. year	2
2	Earth View of Cassini During Rev007 and Rev054	3
3	Data processing pipeline	12
4	Huygens ringlet initial testfigure	14
5	Rev007E Maxwell ringlet at different resolutions	16
6	<i>Left:</i> Resolution dependence of τ_{TH} for Rev 007 E. <i>Right:</i> Effect of opening angle B on τ_{TH} as a function of SNR_0 for $\Delta\rho = 0.25$ km. Data are for all 1 kHz files prior to USO failure	26
7	Comparison of geometry parameters for Rev007 Egress X-band.	34
8	More comparisons of Rev007 geometry.	35
9	Comparison of Frequency Offset with PDS	36
10	Raw Power from Rev007 E X43	37
11	Comparison of Reconstructed Power	37
13	Comparison of our adapted <code>rss_ringoccs</code> processing of the Voyager 2 Uranus ring 5 egress occultation with that of Gresh et al. (1989)	42
14	Example of the frequency offset fit	46
15	Example of the free space power fit	47
12	Validation of 1 kHz processing using raw 16 kHz processing for Rev 007 E X band and Rev 125 E Ka band.	50

List of Tables

1	Hardware and operating systems	5
2	Python versions compatible with <code>rss_ringoccs</code>	5
3	Minor name changes in RSR files	8
4	1 kHz files missing from PDS	8
5	Glossary of parameters in the *GEO.TAB file	20
6	Glossary of data from the CAL file	22
7	Glossary of parameters in TAU file	24
8	Glossary of parameters in TAU file	27
9	Benchmarks for processing the Rev 007 E X43 1 kHz RSR file.	43

1 Introduction

The Cassini [Radio Science Subsystem \(RSS\)](#) was used during the Cassini orbital tour of Saturn to observe a superb series of ring occultations that resulted in high-resolution, high-SNR radial profiles of Saturn’s rings at three radio wavelengths: 13 cm (S band), 3.6 cm (X band), and 0.9 cm (Ka band). Radial optical depth profiles of the rings at 1- and 10-km resolution produced by Essam Marouf of the Cassini RSS team, using state-of-the-art signal processing techniques to remove diffraction effects, are available on the [NASA Planetary Data System \(PDS\)](#).¹ These archived products are likely to be quite adequate for many ring scientists, but for those who wish to generate their own diffraction-reconstructed ring profiles from Cassini RSS observations, we provide `rss_ringoccs`: a suite of Python-based analysis tools for radio occultations of planetary rings.²

The purpose of `rss_ringoccs` is to enable scientists to produce “on demand” radial optical depth profiles of Saturn’s rings from the raw RSS data, without requiring deep familiarity with the complex processing steps involved in calibrating the data and accounting for the effects of diffraction. The code and algorithms are extensively documented, providing a starting point for users who wish to test, refine, or optimize the straightforward methods we have employed. Our emphasis has been on clarity, sometimes at the expense of programming efficiency and execution time. `rss_ringoccs` does an excellent job of reproducing existing RSS processed ring occultation data already present on NASA’s PDS Ring-Moon Systems Node – detailed comparisons of representative examples of our results with those on the PDS are presented below in Section 5.1. However, we make no claim to having achieved the state-of-the-art in every respect. In a project this complex, bugs may well be present, and we encourage users to report any errors, augment our algorithms, and share these improvements so that they can be incorporated in future editions of `rss_ringoccs`.

This document provides an introduction to RSS ring occultations, directs users to required and recommended reading, describes in detail how to set up `rss_ringoccs`, and explains how to obtain RSS data files and auxiliary files required by the software. It provides an overview of the processing pipeline, from raw data to final high-resolution radial profiles of the rings, and guides users through a series of simple examples to illustrate the use of `rss_ringoccs`. The algorithms and code are validated by comparison with Voyager and Saturn RSS results on the PDS archive, and with analytical results. Finally, it includes practical considerations for users, and benchmarks of program execution time.

1.1 Getting help

`rss_ringoccs` is easy to install and use, but if you have questions along the way please don’t hesitate to get in touch with us. We recommend that you post an issue to the `rss_ringoccs` repository³ so that other users can join in the conversation, but you are also free to contact the lead author of the project, Richard G. French, at rfrench@wellesley.edu. For reference, we provide detailed documentation of the software online at <https://rss-ringoccs.readthedocs.io/en/master/>.

¹<https://pds-rings.seti.org/cassini/rss/index.html>

²`rss_ringoccs` may be found from https://github.com/NASA-Planetary-Science/rss_ringoccs

³https://github.com/NASA-Planetary-Science/rss_ringoccs/issues

1.2 What is an RSS ring occultation?

Simply put, an RSS ring occultation occurs when a radio signal transmitted from a spacecraft’s [High Gain Antenna \(HGA\)](#) passes through the rings on the way to a [Deep Space Network \(DSN\)](#) receiving antenna on Earth. The received signal at Earth is affected by interactions of the radio signal with the swarm of ring particles, including attenuation, scattering, Doppler-shifting of the signal, and diffraction. We refer to the process of compensating for diffraction to obtain the intrinsic radial optical depth profile of the rings as [diffraction reconstruction](#) or [Fresnel inversion](#), since the reconstruction process is based on the mathematical principles of Fresnel optics.

1.3 Overview of Cassini RSS ring observations

Over the course of the Cassini orbital tour of Saturn, the geometry of RSS ring occultations varied due both to changes in the orbiter’s trajectory and to the aspect of the rings as seen from Earth during Saturn’s orbit around the Sun. The opening angle of Saturn’s rings as a function of time as seen from Earth is shown below in **Fig. 1**.

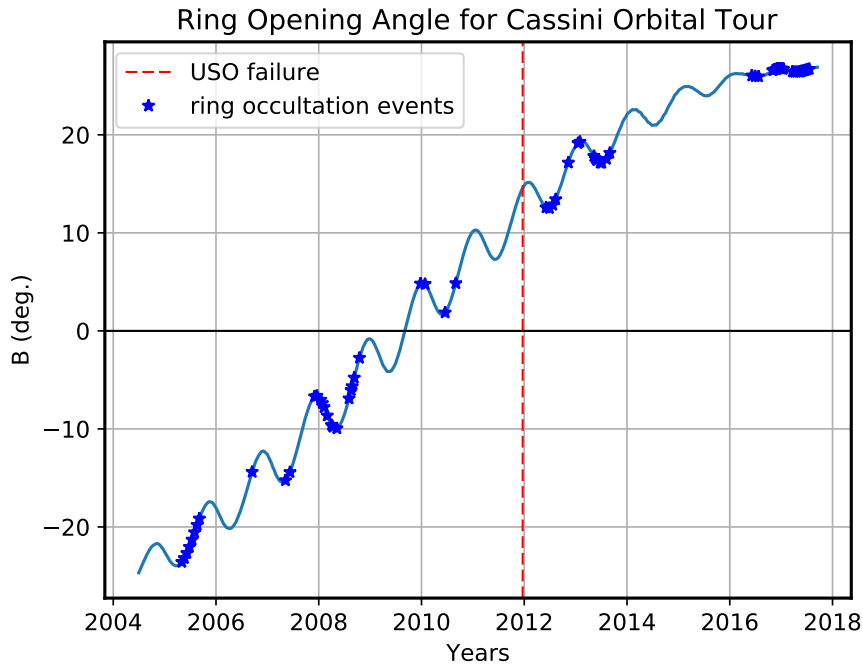


Fig. 1: Ring Opening angle (B) vs. Time. Dark line indicates an opening angle of $B = 0$ where the ring system is edge-on. In 2012, the on-board ultra-stable oscillator (USO) failed, complicating diffraction reconstruction thereafter.

Individual occultations are identified by the Cassini [rev number](#) n , corresponding roughly to the n^{th} orbit of Cassini around Saturn, during which the occultation occurred. During the [ingress](#) portion of an occultation, the orbital radius of the intercept point in the ring plane of the incident ray from the spacecraft decreases with time; the radius increases with time during the [egress](#) portion of an occultation. During a [diametric occultation](#), the ingress and egress portions of the occultation are interrupted by passage of the spacecraft behind the planet itself as seen from Earth, resulting in an [atmospheric occultation](#). The view from Earth of the ingress and egress portions of a diametric occultation on Rev 7 is

shown in **Fig. 2.1**. During a **chord occultation**, the ingress and egress occultations are contiguous. The Earth view of the chord occultation on Rev 54 is shown in **Fig. 2.2**.

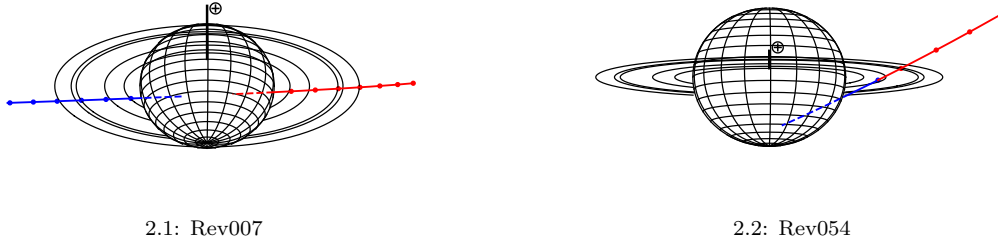


Fig. 2: Earth view of Cassini during the Rev007 and Rev054 ring occultation observations. The red solid lines represent the ingress portion of an occultation and the blue solid lines represent the egress portion. The blue and red dashed lines are the part of the occultation that is blocked by Saturn.

1.4 Cassini RSS ring occultation observations on NASA's PDS

There are two categories of Cassini RSS observations on the PDS: raw data in **Radio Science Receiver (RSR)** files that contain the digitized spacecraft signal as received at the DSN, and higher-level products (reduced data) that have been processed by the RSS team, including diffraction-reconstructed radial profiles of the optical depth of Saturn's rings and associated geometric and calibration information. **rss_ringoccs** processes raw RSR files and independently produces higher-level products that can be saved as files similar in form and content to those already on the PDS, but with a user-defined radial resolution.

1.4.1 Raw RSS data files

The raw data produced by the DSN that contain the original observations of all Cassini occultation observations are recorded in RSR files, described in more detail in the *Cassini Radio Science Users Guide* (Section 1.5.1). During Cassini RSS occultations, RSR files were typically recorded at two bandwidths: 1 kHz and 16 kHz – both types are available on the PDS for all Cassini RSS ring experiments. The **rss_ringoccs** package can handle either version, and they give nearly identical results, although the processing time for the 16 kHz files is significantly longer. We provide convenient scripts in **rss_ringoccs** (Section 2.4.2) to download both 1 kHz and 16 kHz RSR files before USO failure from the PDS archive.

1.4.2 Higher-level products

Essam Marouf of the Cassini RSS science team has produced a comprehensive set of higher-level products for ring occultation observations, available at https://pds-rings.seti.org/viewmaster/volumes/CORSS_8xxx/. This archive set contains 1- and 10-km resolution diffraction-reconstructed profiles from S-, X- and Ka-band observations of Revs 7 through 137. *Users interested in low-resolution ($\simeq 10$ km) ring profiles are advised to use the existing PDS results, rather than to use **rss_ringoccs** to produce them, since our software package does not implement the low-pass filtering techniques used for the PDS results to obtain the 10 km resolution profiles.*

1.5 Required and recommended reading

With this overview of RSS ring occultation observations and data in hand, we strongly recommend that all users familiarize themselves with several key documents before using the `rss_ringoccs` package. Our internal documentation of the `rss_ringoccs` code makes frequent reference to the following two documents:

1.5.1 Cassini Radio Science User’s Guide

The most complete practical introduction to Cassini RSS ring observations is contained in the *Cassini Radio Science User’s Guide*⁴ (Asmar et al. 2018). We regard this as *required reading*. Chapter 2 describes the *open-loop* RSR files that contain the raw RSS ring occultation data, and Chapter 3.3 summarizes the analysis steps to produce a diffraction reconstruction ring optical depth profile from the observations. *For the remainder of this guide, we will assume that all readers have familiarized themselves with this material.*

1.5.2 Marouf, Tyler, and Rosen (1986) - MTR86

The definitive reference for diffraction reconstruction of RSS occultations is Marouf et al. 1986: Marouf, Tyler and Rosen’s classic “Profiling Saturn’s rings by radio occultation.” We refer to this as MTR86. For copyright reasons, we cannot include MTR86 in this GitHub repository, but we highly recommend that scientists making use of radio occultation data have this paper readily at hand. It documents the Fresnel inversion method of diffraction reconstruction, complete with application to Voyager RSS occultation observations of Saturn’s rings. This is *recommended reading* for beginning users of `rss_ringoccs`, and *required reading* for anyone wishing to understand the inner workings of the `rss_ringoccs` software package.

1.5.3 Online Documentation

Documentation of functions, scripts, and modules which comprise `rss_ringoccs`, see the online docs at <https://rss-ringoccs.readthedocs.io/en/master/>.

1.5.4 For more information...

Readers interested in an overview of Cassini RSS instrumentation and science goals are encouraged to read “Cassini Radio Science” by Kliore et al. 2004. Scientific results making use of Cassini RSS occultation observations include Colwell et al. 2009, Moutamid et al. 2016, French et al. 2016a, French et al. 2016b, French et al. 2017 Marouf et al. 2011, Nicholson et al. 2014a, Nicholson et al. 2014b, Rappaport et al. 2009, and Thomson et al. 2007.

2 Setting things up

This section provides step-by-step instructions for setting up the `rss_ringoccs` package and associated data files. We assume that all users are familiar with basic unix commands and introductory-level Python.

⁴Available from <https://pds-rings.seti.org/cassini/rss/>

2.1 System requirements

The `rss_ringoccs` repository has been developed and tested on the following hardware, unix-based operating systems, and shells:

Hardware	Operating System	Shell	GB of RAM
MacBookPro, iMac	MacOS 10.13.4, 10.14.2	csh, bash	8, 16, and 32
Mac mini	Linux Ubuntu Budgie 16	bash	8
MacBookPro	Linux Ubuntu 16	bash	8
ThinkMate	Linux Debian	bash	32

Table 1: Hardware and operating systems

We strongly recommend that users run `rss_ringoccs` on a system with at least 16 GB of RAM (preferably 32 GB) to minimize disk-based memory swapping, which can significantly increase the run time when processing an entire occultation at high resolution.

2.2 Downloading the `rss_ringoccs` repository from GitHub

To download via git clone, open a terminal window and navigate to the directory you wish to run `rss_ringoccs` from and follow the command:

```
git clone https://github.com/NASA-Planetary-Science/rss_ringoccs.git
```

To download `rss_ringoccs` via ZIP file, visit https://github.com/NASA-Planetary-Science/rss_ringoccs and click on the green *Clone or Download* pull-down menu at the upper right and click on *Download ZIP*. `rss_ringoccs-master.zip` will appear in your local **Downloads** directory; unzip it and move it to your desired directory. Unzipping the file will create an `rss_ringoccs-master` folder, as opposed to the `rss_ringoccs` folder that appears when the repository is cloned.

2.3 Install Python 3 and required packages

`rss_ringoccs` has been developed under Python 3, in particular Python 3.5, 3.6, and 3.7. Our code has been tested under the following Python configurations:

Operating System	Distribution	Version	URL
MacOS 10.14.1	Enthought	3.5.2	https://www.enthought.com
MacOS 10.14.1	Anaconda	3.6.3	https://www.anaconda.com
MacOS 10.14.2	Anaconda	3.7.1	https://www.anaconda.com
Linux Ubuntu Budgie 16	Anaconda	3.6.3	https://www.ubuntu.com
Linux Ubuntu 16	Anaconda	3.6.3	https://www.ubuntu.com
Linux Debian	Anaconda	3.6.3	https://www.debian.org

Table 2: Python versions compatible with `rss_ringoccs`

Once the `rss_ringoccs` package has been downloaded, the user will need to install Python 3 and various dependencies. The easiest way to do this by running the setup script `rss_ringoccs.config.sh`, included in the download. This will install Python 3 using Anaconda, as well as all of the necessary packages, and update and source the `.bash_profile` and `.bashrc` files. For users who have already installed Python, this

script will update `conda` and then check that `spiceypy` and other packages exist on the machine. To run the setup script, navigate to the `rss_ringoccs` directory in a terminal and run the following:

```
./rss_ringoccs_config.sh
```

The installation process will begin, and may take several minutes.

If the configuration script fails in some fashion, or expected packages are not loaded, the user may instead manually install Python 3 and the required packages by visiting the URLs found in Table 2. A manual installation of either Anaconda3 or Enthought Canopy will provide all required dependencies, with the exception of the `spiceypy` package.

2.3.1 Download and install spiceypy

`rss_ringoccs` makes extensive use of JPL’s NAIF SPICE toolkit (Acton 1996), a set of software tools to calculate planetary and spacecraft positions, ring occultation geometry, and a host of useful calendar functions.⁵ Our software requires `spiceypy`, a Pythonized version of the NAIF toolkit, available from <https://github.com/AndrewAnnex/SpiceyPy>. The setup bash script detailed before should install `spiceypy` automatically, but `spiceypy` can also be installed manually by following the instructions on <https://spiceypy.readthedocs.io/en/master/installation.html>.

2.3.2 Test spiceypy

To test your installation of `spiceypy`, fire up Python in a terminal at the unix command line and at the `>>>` prompts, enter the following commands to confirm that `spiceypy` returns π and the speed of light c :

```
python
>>> import spiceypy
>>> print(spiceypy.pi(),spiceypy.clight())
3.141592653589793 299792.458
>>> exit()
```

2.4 Downloading Necessary Files

`rss_ringoccs` provides bash-based shell scripts to automate the retrieval of necessary files. For quick setup and testing (Section 3.3) purposes, run the following command in the `examples` directory:

```
./get_example_files.sh
```

For more shell scripts to retrieve a complete list of files and a detailed explanation of their contents, see below.

2.4.1 JPL/NAIF SPICE kernels

The `rss_ringoccs` package makes extensive use of SPICE data (*kernel* files) from JPL/NAIF that specify planetary and spacecraft ephemerides, planetary constants, and other

⁵See <https://naif.jpl.nasa.gov/naif/index.html>

essential information for computing the geometric circumstances of occultations.⁶ The `rss_ringoccs` distribution contains bash-based shell scripts to automate the retrieval of SPICE kernels from the NAIF website and store them in subdirectories under `rss_ringoccs/kernels/`, following the same directory structure as on the NAIF ftp site. Some of the kernel files are quite large, and will take some time (and significant disk space) to download. The size of the total set of kernel files is 897 Mb, so be sure that there is sufficient disk space available.

In order to compute the geometry of RSS occultations throughout the Cassini orbital tour, download a complete set of kernels by navigating to the `pipeline` directory and entering the following command:

```
./get_all_kernels.sh
```

The shell script detects whether a given kernel has already been downloaded, so you may interrupt this command if it hasn't run to completion in the time you have available, and repeat the command later, picking up the downloading process where it left off the previous time. However, if you stop the `get_kernels.sh` script while it is downloading a file, the file may be incomplete but will still be detected by future runs of the shell scripts as having been downloaded. You will need to delete incomplete files to restart the download. To check for incomplete files, look in the `lsk`, `pck`, and `spk` directories within the `rss_ringoccs-master/kernels/naif/CASSINI/kernels/` directory and in the `rss_ringoccs-master/kernels/naif/generic/kernels/` directory. The most likely incomplete files will be `.bsp` files located in the `spk/` directory where kernel files for each occultation set of spacecraft and planetary ephemerides are stored; however, it is best to check all kernel directories.

Once you have downloaded the complete set of kernels, you will not need to repeat this process unless JPL releases an updated set of Cassini trajectory files. We plan to update this documentation and the input files for `get_kernels.sh` if that occurs. The meta-kernel for the total set of Cassini and Saturn kernels is located in `../tables/e2e.kernels.ker`, which the user will need to reference when running `rss_ringoccs`.

2.4.2 Cassini RSS raw data files

The `rss_ringoccs` package requires local access to raw Cassini RSS data files (Section 1.4.1). The storage capacity on GitHub is not sufficient to allow even one sample RSR file to be part of the standard download. To download the full set of pre-USO failure 1 kHz files used in the archived `CORSS_8001` reconstructed data products, use the shell scripts provided in the `./pipeline/` directory to download the 1 kHz RSR files before and after USO failure (the scripts for these are `get_rsr_files_preUSOfailure.sh` and `get_rsr_files_postUSOfailure.sh`, respectively). These can be called by navigating to `./pipeline/` and executing the scripts as shown here:

```
cd rss_ringoccs/pipeline
./get_rsr_files_preUSOfailure.sh
./get_rsr_files_postUSOfailure.sh
```

⁶For detailed information about kernels, visit <https://naif.jpl.nasa.gov/naif/data.html>

This will take quite some time to execute, given the large volume (10 GB for just the pre-USO failure 1 kHz RSR files) of data to be transferred over the internet. Note that the following eight files on the PDS differ in name from the files used in `CORSS_8001`:

CORSS_8001 file	PDS file
S10EAOE2005_123_0740NNNK34D.1B1	S10SROE2005123_0740NNNK34RD.1B1
S10EAOE2005_123_0740NNNS43D.2B1	S10SROE2005123_0740NNNS43RD.2B1
S10EAOE2005_123_0740NNNX34D.1A1	S10SROE2005123_0740NNNX34RD.1A1
S10EAOE2005_123_0740NNNX43D.2A1	S10SROE2005123_0740NNNX43RD.2A1
S10EAOE2005_123_0229NNNS43D.2B1	S10SROI2005123_0230NNNS43RD.2B1
S12SROE2005177_2226NNNX14RD.2A1	S12SROE2005177_2225NNNX14RD.2A1
S12SROI2005177_1745NNNS14RD.2B1	S12SROI2005177_1740NNNS14RD.2B1
S12SROI2005177_1745NNNX14RD.2A1	S12SROI2005177_1740NNNX14RD.2A1

Table 3: PDS RSR file name changes

In addition, there are three 1 kHz files used by `CORSS_8001` that are currently not available on the PDS. In `../tables/rsr_1kHz_files_before_USO_failure.txt`, we replace these missing files with their 16 kHz equivalents, shown in Table 4.

1 kHz missing from PDS	16 kHz version
S10EAOI2005_123_0230NNNK26D.3B1	s10sroi2005123_0230nnnk26rd.3b2
S10EAOI2005_123_0230NNNX26D.3A1	s10sroi2005123_0230nnnx26rd.3a2
S10EAOE2005_123_0229NNNX43D.2A1	s10sroi2005123_0230nnnx43rd.2a1

Table 4: Left: 1 kHz files processed in `CORSS_8001` that are currently unavailable on the PDS. Right: 16 kHz equivalent of the missing files. Names are case-sensitive.

`rss_ringoccs` is currently able to process all pre-USO failure files but only some of the post-USO failure files.

2.5 Hard Drive Space

As a part of the hardware requirements for using `rss_ringoccs`, we describe here the requisite hard drive space for running the software. Although the software itself is relatively small in size ($\lesssim 5$ Mb), the files needed to produce science data products require a substantial amount of drive space. The full set of kernel files downloaded following Section 2.4.1 is 897 Mb in size. The full set of 1 kHz RSR files downloaded following Section 2.4.2 is 10.43 Gb in size. In total, the files required for all occultations observed before USO failure will require 11.35 Gb of drive space.

While this accounts for the initial space necessary for processing every pre-USO-failure occultation observation, the output files for each end-to-end pipeline data reduction will take up additional drive space. A single occultation (e.g., Rev 007 E X43) processed at 0.25 km DLP resolution and 1 km reconstruction resolution will have ≈ 520 Mb of output files. The output files containing the DLP and reconstructed profiles (the DLP and TAU files, respectively) are 61.2 Mb each when the profile covers the entire ring system, and these two output file types compose $> 90\%$ of the output file data volume. Because the DLP and TAU files are by far the largest, a smaller radial spacing / higher radial resolution will increase the output file size.

The duration of each occultation observation varies from rev to rev, often depending on the occultation geometry and the elevation angle of Saturn relative to Earth’s horizon at any given DSN station. Additionally, the number of bands and DSN stations at which a given occultation is observed also vary. As such, any given RSR file may not yield a complete radial profile resulting in smaller DLP and TAU files. Processing all 154 RSR files prior to USO failure (Section 3.5) will result in 22.66 GB of output files.

2.6 Install A L^AT_EX Compiler

To make full use of `rss_ringoccs`, you’ll need a LaTeX compiler to create the summary PDF files that accompany any given data set. We strongly recommend `pdflatex`, as this is the only compiler that `rss_ringoccs` has been tested on. MacTeX contains all of the necessary packages and more, and is recommended for users of MacOSX who wish to use L^AT_EX to its full capacity. A download can be found from <https://www.tug.org/mactex/mactex-download.html>. However, this is a very large download (a few Gigabytes). For users who want only the necessary compiler and associated binary files, BasicTeX will suffice. This can be downloaded from <https://www.tug.org/mactex/morepackages.html>. For linux users, enter one of the following commands:

```
sudo apt-get install texlive-full
```

or:

```
sudo apt-get install texlive-latex-base
```

This will again download either a plethora of packages for LaTeX and the necessary compiler, or simply the basic necessities. In an attempt to keep the LaTeX portion of the code simple, very few packages are used. A complete list, and how they appear in the code, is given as follows:

```
%-----Begin LaTeX Code-----%
\usepackage{geometry}
\geometry{a4paper, margin = 1.0in}
\usepackage{graphicx, float}
\usepackage{lscape}
\usepackage[english]{babel}
\usepackage[dvipsnames]{xcolor}
\usepackage[font={normalsize}, labelsep=colon]{caption}
```

In the event of a successful install of either package, all of these dependencies should now be available to the user. Should any error arise, a manual install can be done by visiting the CTAN website. For example, the `float` package can be found from <https://ctan.org/pkg/float>.

3 Using `rss_ringoccs`

Here, we provide an overview for using `rss_ringoccs`, including requisite information for writing your own scripts to process the data as well as directions for running example scripts.

3.1 End-to-end pipeline outline

For those interested in the the high-level structure of the processing pipeline, we offer this brief overview. An end-to-end script will require instantiating five separate Python classes in succession:

1. `rsr_inst = rss.RSRReader('RSR_filename.a2a')`

- Creates an instance of the `RSRReader` class and stores it in `rsr_inst`.

2. `geo_inst = rss.occgeo.Geometry(rsr_inst, planet, spacecraft, kernels)`

- Creates an instance of the `Geometry` class and stores it in `geo_inst`.
- Takes an `RSRReader` instance and user-specified planet, spacecraft, and kernel files.
- Calculates, among other things, the radial intercept point ρ where the Cassini spacecraft radio signal is occulted by the rings and the locations of gaps in the occultation profile.
- Produces `GEO*.TAB` and `GEO*.LBL` files.
- These and all subsequent output files are written to a user-specified output directory

3. `cal_inst = rss.calibration.Calibration(rsr_inst, geo_inst)`

- Creates an instance of the `Calibration` class and stores it in the variable `cal_inst`
- Takes the `RSRReader` (`rsr_inst`) and `Geometry` (`geo_inst`) instances
- This instance contains the calibrations necessary to convert the raw data into a diffraction-limited radial ring profile
- Calculates the observed frequency of the spacecraft signal to correct the real and imaginary components of the transmittance (I and Q), then estimates the intrinsic received power over the entire occultation
- Produces `CAL*.TAB` and `CAL*.LBL` files following the naming convention for the GEO files
- Produces frequency offset fit plots (`*FORFIT.PDF`) and free space power fit plots (`*FSPFIT.PDF`)

4. `dlp_inst = rss.calibration.DiffractionLimitedProfile(
 rsr_inst, dr, geo_inst, cal_inst)`

- Creates an instance of the `DiffractionLimitedProfile` class and stores it in the variable `dlp_inst`.
- Takes the `RSRReader`, `Geometry`, and `Calibration` instances.
- Contains as attributes the DLP as calibrated and reduced by the previous classes
- Optional input of radial sampling rate `dr_km_desired` in kilometers
- Calculates the normalized power P/P_0 and the diffraction-limited optical depth profile assuming $\tau = -\sin B \ln(P/P_0)$

- Produces DLP*.TAB and DLP*.LBL files with the same naming convention as the GEO and CAL files with the additional RRRR indicator.

5. `tau_inst = rss.diffrec.DiffractionCorrection(dlp_inst,res_km)`

- Creates an instance of the `DiffractionCorrection` class and stores it in the variable `tau_inst`
- takes the `DiffractionLimitedProfile` instance and a user-specified reconstruction resolution `res_km` in kilometers
- Calculates the optical depth profile with reconstruction by accounting for diffraction effects by means of Fresnel inversion at the user specified reconstruction resolution.
- Produces TAU*.TAB and TAU*.LBL files following the same naming convention as the DLP files except that RRRR here indicates the reconstruction resolution selected by the user when instantiating the `DiffractionCorrection()` class

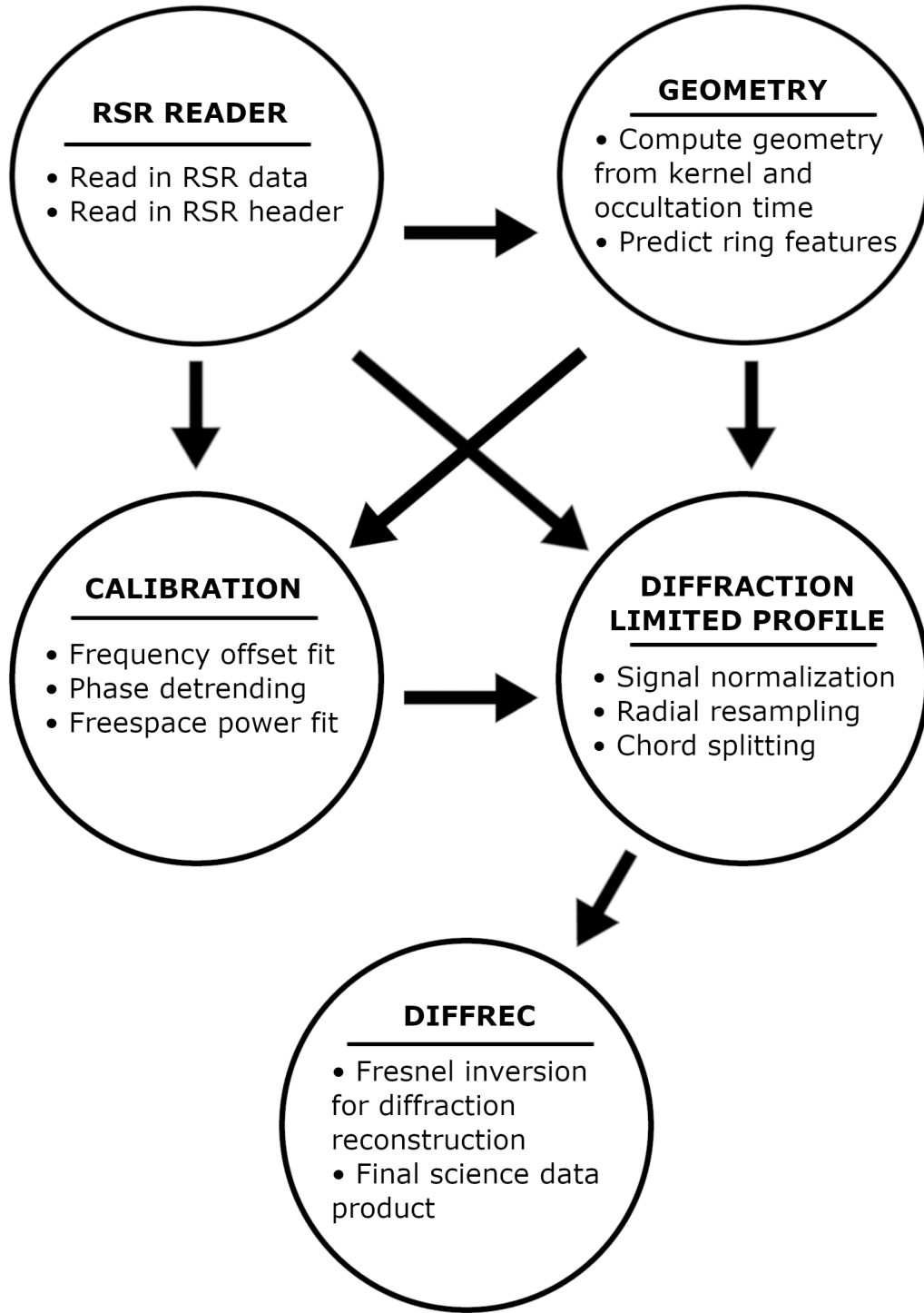


Fig. 3: Data processing pipeline. Each bubble is a separate object called by the `e2e*.py` scripts.

3.2 Conventions and hierarchy

Here, we list some conventions used in the `rss_ringoccs` directory structure/hierarchy and the names of directories and output files.

- No reference is made within the `rss_ringoccs` package to local directories outside of the top-level `rss_ringoccs/` hierarchy of directories.

- The directory structure under `rss_ringoccs/` *must* strictly follow that of the original download from the GitHub repository.
- For portability, all references within `rss_ringoccs` software to pathnames to other directories within `rss_ringoccs/` are relative, not absolute.
- Unless otherwise noted, all executable scripts and Python programs *must* be run from within the `rss_ringoccs/pipeline/` directory. (This is so that relative pathnames will point to the correct directories.)
- Output file names follow the format:
`RSS_OBSY_DOY_B##_D_INF_RRRRRM_YYYYMMDD_XXXX.EXT`
 - `OBSY` is the year the observation was made
 - `DOY` is the day of the year the observation was made
 - `B` is the wavelength band of the observation
 - `##_` is the DSN station number
 - `D` is the direction of the occultation
 - `INF` is a three-letter reference specifying the information stored within the file (`GEO` for the occultation geometry, `FOF` for the frequency offset, `CAL` for calibration, `DLP` for the DLP, and `TAU` for the diffraction-reconstructed optical depth profile)
 - `YYYYMMDD` is the year, month, and date on which the user ran the `rss_ringoccs` code
 - `XXXX` is the `XXXX`th run of `rss_ringoccs` on that date
 - `EXT` is the file extension
 - Only DLP and TAU files contain the `RRRRR` in the nomenclature. For the DLP files, `RRRRR` is the minimum reconstruction resolution (the so-called “DLP resolution” in MRT86) in meters while for TAU files this is the processing resolution selected by the user
 - The output LBL files match those from the PDS with minor changes.

With these caveats in mind, users are highly encouraged to write their own scripts to call upon and make full use of the `rss_ringoccs` package. To that end, we provide example scripts for both pipeline versions as well as specific portions of the pipeline.

3.3 End-to-end pipeline: A look at the Huygens ringlet

The “end-to-end” pipeline process is a set of steps that need to be performed only once when processing a given RSR file from scratch. For the initial run, users will need to supply an RSR file, a set of kernels to use, a radial spacing to resample to, and a reconstruction resolution (there are also default keyword inputs documented within each routine). The RSR extraction, geometry calculation, and frequency offset calculation steps are all automated; however, users may specify several options and parameters in advance by editing the input file `e2e.run_args.py` in the `./examples/` directory. These options include whether to process 16 kHz files, whether to write results to output files, whether to output to terminal, and whether the pipeline should enter an interactive

mode to customize the freespace power fitting. This interactive mode will display the automated fit results in a `matplotlib` plotting window and prompt the user for changes to the fit parameters and/or freespace regions.

At the end of the end-to-end run, if the `write_file` keyword argument is set to `True`, several data files will be generated: geometry files, calibration files, diffraction-limited profile files, reconstructed optical depth files, and a summary file. Each class instantiated in the end-to-end pipeline process corresponds to a specific set of output files that match those on the PDS. Once these files have been produced, users can use the quick-look process for subsequent runs.

Also included in the pipeline's arguments file are various and sundry parameters for customizing the calibration steps and the profile ranges and resolutions. Detailed descriptions of which resolution values to choose and definitions of each type of resolution are given below.

As a demonstration of this end-to-end pipeline, we provide an example script which runs through the entire pipeline and examines the Huygens ringlet. To execute this script, follow the example below.

```
cd rss_ringoccs/examples
python e2e_run.py
```

This will automatically produce the multipanel plot shown in Fig. 4. The top row of plots show the raw power, optical depth, and phase profiles (i.e., the observed diffraction pattern) that were input to `DiffractionCorrection`, and the bottom row shows the reconstructed radial profiles in power, optical depth and phase.

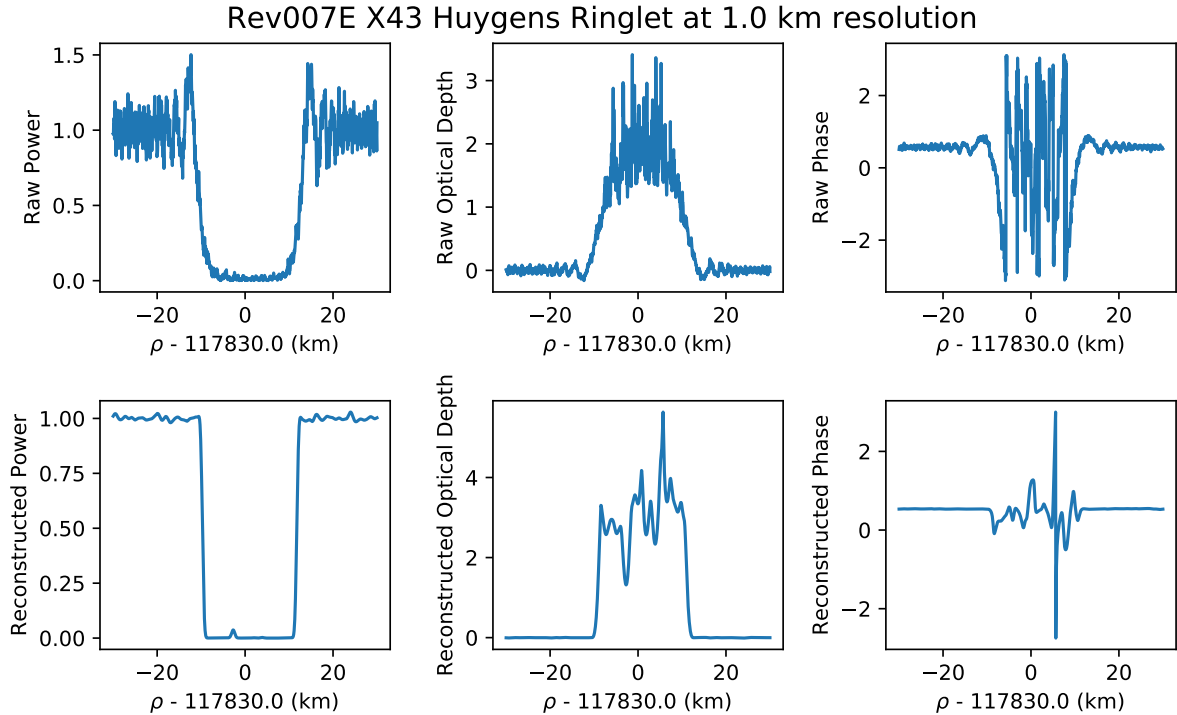


Fig. 4: Power, optical depth, and phase plots produced by `e2e_run.py`

3.4 Quick-look method: Comparing profiles reconstructed at different resolutions

The quick-look approach allows users to save computation time by utilizing pre-computed data files. For these runs, users start directly at the Fresnel inversion step of the pipeline, provided they have a set of `GEO*.TAB`, `CAL*.TAB`, and `DLP*.TAB` files. Users must specify the relative file path(s) and desired `*.TAB` files to the `tools.ExtractCSVData()` class to create a DLP instance like the one instantiated in the end-to-end pipeline from the `calibration.DiffractionLimitedProfile()` class. This DLP instance can then be passed to the `diffrec.DiffractionCorrection()` class.

Continuing with the RSR file downloaded in Section 2.4.2, we provide an example script to demonstrate diffraction-reconstructed optical depth profiles at different reconstruction resolutions for the Maxwell Ringlet. Before running, users will need to open the script in a text editor and change the `*_file` variables to include the appropriate file names as a string after the pre-specified path. For the first set of Rev007 files output by the `e2e_run.py` script, this might resemble

```
dir = '../output/Rev007/Rev007E/Rev007E_RSS_2005_123_X43_E/'
geo_file = dir+'RSS_2005_123_X43_E_GEO_20180926_0001.TAB'
cal_file = dir+'RSS_2005_123_X43_E_CAL_20180926_0001.TAB'
dlp_file = dir+'RSS_2005_123_X43_E_DLP_0100M_20180926_0001.TAB'
```

To execute the example quick-look script, follow the example below

```
cd rss_ringoccs/examples
python quick_look_run.py
```

This will produce optical depth profiles at four different reconstruction resolutions: 1 km, 0.75 km, 0.5 km, and 0.25 km for the Rev007 egress X band observation processed by the end-to-end script in Section 3.3. Running the script will produce the plot in Fig. 5 (not including the reference red line for PDS data).

3.5 Batch Scripts

To simplify and expedite the use of `rss_ringoccs` to process large numbers of files, we provide a single python script `e2e_batch.py` in `./pipeline/` that, when executed, will run the end-to-end pipeline for a list of files contained in a reference ASCII text file. The default list, `list_of_rsr_files_before_USO_failure_to_dl.txt`, is the same list of files used by `get_rsr_files.sh` to download all of the 1 kHz Cassini RSR files prior to the USO failure, and can be found in the `tables/` directory. This batch script, like the single-file script, reads in a file of parameters, `e2e_batch_args.py`, which specifies, among other things, radial sampling rates, calibration fit orders, profile range, and whether to output files. By modifying the contents of `e2e_batch_args.py`, users can produce on-demand, customized optical depth profiles for specific data sets and radial regions of interest. Each option within `e2e_batch_args.py` is documented in comments. For more information, we refer users to either §4 or the documentation online at <https://rss-ringoccs.readthedocs.io>.

The default end-to-end pipeline will jump into its interactive mode when the normalization of the received power to a nominal free-space level is poor. To avoid this interactive

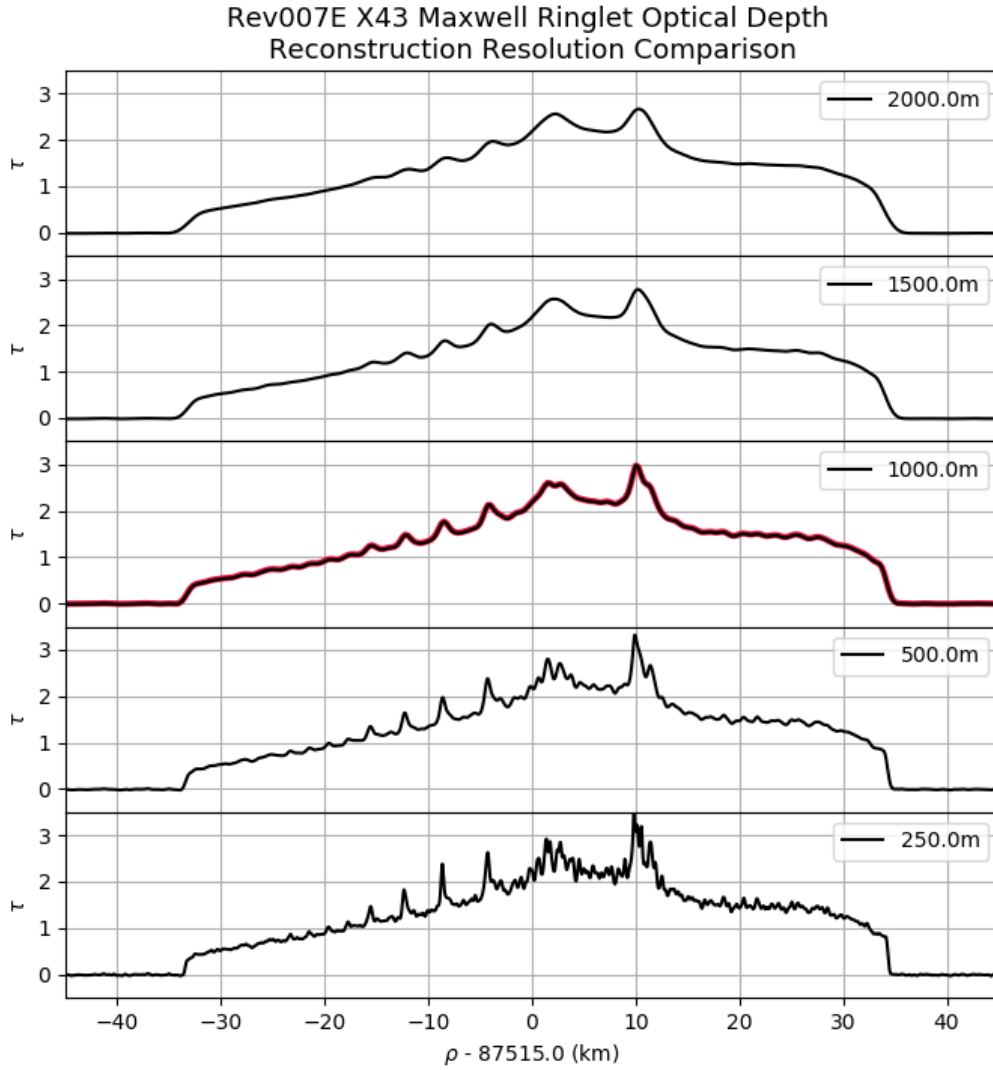


Fig. 5: Optical depth profile for the Maxwell ringlet from the Rev007E X43 occultation reconstructed at 2 km, 1 km, 0.5 km, and 0.25 km resolution. Solid black lines are the optical depth profile produced by the quick-look example script at various processing resolutions indicated by the plot text. For reference and validation, the solid red line is the 1 km reconstruction resolution profile obtained from the PDS3.

mode, one can alternatively run the script prepended with the unix **yes** command to automatically enter the string “yes” for all input prompts (thus accepting the default fit, should **rss_ringoccs** enter interactive mode). Running the batch script in this manner will resemble

```
yes | python e2e_batch.py
```

The end-to-end python script will automatically generate an error log file with a name such as **e2e_batch_YYYYMMDD-HHMMSS.err** to contain names of files that failed to process during the batch run as well as the resulting traceback error. If **verbose** keyword argument is set to **True**, some benign messages, such as:

```
1) WARNING (RSRReader): file size not the same as expected!
2) DETECTED INGRESS (resample_IQ.py): reversing arrays
```

will be printed to the terminal. These do not result in failed processing runs and are therefore not printed to the `.err` file.

One of these default parameters is `with16=False`, which skips 16 kHz files that have no corresponding 1 kHz files (see §2.4.2, Table 4). Set this to `True` only if the user’s computer has at least 16 GB RAM; these files will also take longer to process than a typical 1 kHz file.

3.6 Profile Resolution

As a practical matter, most users would like to know: “What is the highest achievable resolution for a given ring occultation, and how can I use `rss_ringoccs` to produce a radial profile at that resolution?” The answer is complicated by a variety of definitions of resolution – we clarify these below – and is also affected of course by the SNR of any given occultation, which depends on the wavelength of the observation, the observing conditions at the DSN, the size of the receiving antenna, the geometry and speed of the occultation, and host of other factors.

Our recommendation is that users begin with the 1-km reconstructions available on the PDS, to get a sense for the signal quality of a specific occultation of interest. As a next step, we provide an end-to-end script that uses `rss_ringoccs` to process the entire set of Cassini RSS occultation data (up to the point of the USO failure) at a diffraction-corrected resolution of 0.5 km. Users can compare these profiles with the 1.0 km results available on the PDS, and decide whether reconstruction at a different resolution is needed.

In cases where higher resolution is desired and is warranted by the SNR, we recommend that `rss_ringoccs` users produce diffraction-reconstructed radial profiles at successively finer resolutions (centered on a feature of interest over a limited radial range to reduce processing time). As the resolution is increased, the effects of noise will eventually limit the useful information that can be obtained for a particular science goal.

To carry this out in practice, it is important to understand the factors that affect the diffraction-reconstructed resolution. Prior to diffraction reconstruction, RSS observations are limited in resolution at a spatial scale comparable to the Fresnel scale $F \approx \sqrt{\lambda D/2}$, where λ is the wavelength of observation and D is the distance between the spacecraft and the ring plane. Geometrical effects such as the tilt of the ring plane result in multiplicative scale factors applied to F , when converted to an effective diffraction-limited resolution in the radial direction in the ring plane, but as a general rule, the effective Fresnel scale for RSS ring measurements is of order a few km for typical occultations.

The task of diffraction reconstruction is to improve on the diffraction-limited resolution of the observations by making use of both the intensity and phase of the received signal across the diffraction pattern. Intuitively, it makes sense that a high-resolution diffraction reconstruction will require finer resolution of the observed diffraction pattern, over a larger radial range (what we refer to as a *filter window* – see MTR86 for details), than a low-resolution reconstruction. For sub-km diffraction reconstruction at S-band (the longest of the three Cassini wavelengths), the filter window can extend for hundreds of km to

either side of the location of the feature of interest, and it must be sampled at a fine enough spatial scale to capture the diffraction fringes produced by the ring feature over the entire filter window.

From these considerations, it is clear that in order to achieve a given post-reconstruction resolution, there is an implied requirement on the resolution at which the diffraction-limited profile is calculated. To satisfy the sampling theorem, if the radial sampling of the diffraction-limited DLP profile is $\Delta\rho$, the subsequent diffraction reconstruction resolution can only have shortest resolvable wavelengths of Nyquist or higher (i.e., $\geq 2\Delta\rho$) – more on this in a moment! As a practical matter, it sometimes makes sense to produce a DLP profile once and for all at very high resolution (say, 0.05 km), use it to produce a diffraction-reconstructed profile at, say, 1 km resolution, and then to reuse this same DLP profile to explore the SNR properties of successively higher resolution diffraction reconstructions, consistent with the sampling theorem.

The original data are a time series sampled at regular intervals in time, but all of our processing is done in terms of ring plane radius. To achieve uniform radial sampling, we take account of the changing velocity and trajectory of the spacecraft and resample the phase-corrected complex signal at a user-defined spacing $\Delta\rho$ (passed to the `DiffractionLimitedProfile` as the positional argument `dr_km` when instantiating the class). We suggest a value for `dr_km` no smaller than 0.05 km.

To satisfy the sampling theorem, the subsequent diffraction reconstruction resolution can only have shortest resolvable wavelengths of Nyquist or higher (i.e., $\geq 2\Delta\rho$). However, the situation is complicated by a variety of definitions of diffraction reconstruction resolution. As MTR86 make clear (see Eqs. 10–14), resolution can be defined in terms of the filter window width W either as the null-to-null width of the main diffraction lobe, or as the equivalent width of the normalized impulse response profile – these definitions differ by a factor of two from each other. Uncertainty in the Fresnel scale can further degrade the effective resolution, as can limitations on the validity of the approximations made to compute the phase of the signal.

In our development of `rss_ringoccs`, we used the MTR86 Eq. 11 definition of effective resolution $\Delta R_W = 2F^2/W$ and geometrically scaled definition of F given by MTR86 Eq. 6. This is the definition used throughout MTR86 in the analysis of Voyager RSS data, and it matches our results when we apply diffraction reconstruction to the archived PDS Voyager RSS diffraction-limited profiles. We refer to this as the *processing resolution* ΔR within `rss_ringoccs`.

As it happens, Essam Marouf introduced yet another definition of resolution in his Cassini reconstructed profiles recently submitted to the PDS, based on applying a low-pass filter to the diffraction reconstruction. The relation between ΔR_W and this new PDS definition of resolution is given by

$$\Delta R_W = 0.75\Delta R_{\text{PDS}}$$

or

$$\Delta R_{\text{PDS}} = \frac{4}{3}\Delta R_W.$$

Given these two definitions, and a lack of information about the details of the low-pass filter used by Marouf, we faced a choice: should we retain the definition of resolution as derived in MTR86 and apparently used for Voyager RSS results, or should we be consistent with the recent PDS contributions by Marouf? We chose the latter approach, so that our `rss_ringoccs` runs could be directly compared to the PDS results at the same quoted post-reconstruction resolution. The user specifies a desired resolution ΔR_{PDS} for the reconstructed profile through the `res` positional argument when instantiating the `DiffractionCorrection` class. The resolution specified by `res` is then scaled within this class to the processing resolution ΔR by the factor of 0.75 described above.

In our interpretation of the sampling theorem, `rss_ringoccs` requires that $\Delta R_W > 2\Delta\rho$. In terms of the requested diffraction-corrected resolution ΔR_{PDS} , this places the following restriction on the pre- and post-diffraction reconstructed resolutions:

$$\Delta R_{\text{PDS}} > \frac{8}{3}\Delta\rho.$$

The factor of $8/3$ is both cumbersome and non-intuitive, but is the best compromise we could find. As specific example, for a DLP file produced at 0.05 km resolution ($\Delta\rho = 0.05$ km), the requested post-diffraction resolution ΔR_{PDS} (or `res`) must be no smaller than $8/3 \times 0.05 = 0.133$ km. (This is below the useful resolution of the majority of Cassini RSS ring observations.) If the user violates these conditions, an informative error message will be displayed, made more comprehensible (we hope!) by this extensive discussion.

4 A detailed look at `rss_ringoccs`

Here, we elaborate on the inner workings of `rss_ringoccs`, describing the methods and goals of each step of the software pipeline in order to make this software as accessible and transparent as possible. For documentation and definition of individual variables, functions, methods, and modules, we refer the user to our online reference available at <https://rss-ringoccs.readthedocs.io/en/master/>.

4.1 RSR reader

The `rsr_reader/` subpackage is used for extracting information from a given RSR file. The `RSRReader` class, when instantiated with a linked RSR file, extracts the raw complex signal I and Q from the RSR file from the PDS as well as some accompanying non-geometric meta-data stored in the RSR file header such as the DSN station, observation dates, sampling rate, start and end times of the observation, and the band of observation.

4.2 Occultation geometry routines

The routines in `occgeo/` depend heavily on the NAIF SPICE Toolkit and are geared towards reproducing all occultation geometry parameters that are documented within `Archived_Cassini_RSS_RingOcocs_2018`, or `CORSS.8001 v2`, submission (see Table 5). In addition to these parameters, `Geometry` also calculates the optical depth enhancement factor β and the effective ring opening angle B_{eff} .

Geometry supplies the rest of the pipeline Using orbital parameters compiled from literature and the computed ring azimuth and ring radius, **Geometry** uses the `find_gaps` method from the `cal_occ_geo` module to determine the locations of gaps in the ring system in the occultation data set. This is done iteratively by repeating the following steps, choosing the semimajor axis of the gap edge for the initial predicted edge radius:

1. Find the ring radius in the occultation data set closest to the predicted gap edge radius.
2. Find the ring azimuth and orbital time implied by the closest ring radius.
3. Predict the radius of the gap edges using compiled orbital properties and the implied ring azimuth and time.

Symbol	Parameter Name	Geometry Attribute
t_{OET}	OBSERVED EVENT TIME	t_oet_spm_vals
t_{RET}	RING EVENT TIME	t_ret_spm_vals
t_{SET}	SPACECRAFT EVENT TIME	t_set_spm_vals
ρ	RING RADIUS	rho_km_vals
ϕ_{RL}	RING LONGITUDE	phi_rl_deg_vals
ϕ_{ORA}	OBSERVED RING AZIMUTH	phi_ora_deg_vals
B	OBSERVED RING ELEVATION	B_deg_vals
D	SPACECRAFT TO RING INTERCEPT DISTANCE	D_km_vals
V_{rad}	RING INTERCEPT RADIAL VELOCITY	rho_dot_kms_vals
V_{az}	RING INTERCEPT AZIMUTHAL VELOCITY	phi_rl_dot_kms_vals
F	FRESNEL SCALE	F_km_vals
R_{imp}	IMPACT RADIUS	R_imp_km_vals
r_x	SPACECRAFT POSITION X	rx_km_vals
r_y	SPACECRAFT POSITION Y	ry_km_vals
r_z	SPACECRAFT POSITIION Z	rz_km_vals
v_x	SPACECRAFT VELOCITY X	vx_kms_vals
v_y	SPACECRAFT VELOCITY Y	vy_kms_vals
v_z	SPACECRAFT VELOCITY Z	vz_kms_vals
θ_{EL}	OBSERVED SPACECRAFT ELEVATION	elev_deg_vals

Table 5: Glossary of parameters in *GEO.TAB file in PDS submission `Cassini.RSS.Ring.Profiles.2018.Archive` and their corresponding attribute names within the **Geometry** class.

This iterative process returns the gap edge radius once it converges on a solution (i.e., when the difference between two consecutive radius predictions is less than one meter) which occurs within five iterations or less. The inner and outer edges of each gap are then stored as an attribute of the **Geometry** object instance for future use.

To create an instance of **Geometry**, or `geo_inst`, users will need an instance of the **RSRReader** class (`rsr_inst`), a target planet, a target spacecraft, a set of kernels over the duration of the RSR file used in `rsr_inst`, and, optionally, a desired number of points per seconds for all calculations (the default is 1 point per second). For choosing an RSR file, which is the only input necessary for instantiating **RSRReader**, see Section 7.2.

4.3 Calibration routines

All of the routines to produce a calibrated diffraction pattern are in the `calibration/` directory in the `rss_ringoccs` package. Each of them performs a portion of the frequency and power calibration steps. Every step of the calibration process is handled by the `Calibration` class. When instantiated (which requires the appropriate instances of the `RSRReader` and `Geometry` classes), it calls the `FreqOffsetFit` class to compute the offset frequency needed for phase-correcting the measured complex signal. Then, the signal is phase-corrected using the `Calibration` class method `correct_IQ` following [Marouf et al. \(1986\)](#) and [Asmar et al. \(2018\)](#). The phase detrending function ψ is computed from the cumulative integral:

$$\psi = \int^t \hat{f}(\tau)_{offset} d\tau + \psi(t_0) \quad (1)$$

and used for detrending the complex signal such that:

$$I_c + iQ_c = [I_m + iQ_m] \exp(-i\psi) \quad (2)$$

([Asmar et al. 2018](#), Equations 17 and 18). Then, `Calibration` normalizes the phase-corrected signal, using the `Normalization` class to estimate the intrinsic spacecraft power \hat{P}_0 as it changes over the course of the occultation.⁷ Finally, all the results of the calibration are written out to an LBL and a TAB file following the naming conventions discussed above.

4.3.1 Frequency Offset

To estimate the offset frequency over the entire occultation, we compute the offset frequency $f(t)_{offset}$ directly from the raw data using a series of “rolling-window” FFTs. This is stored as an attribute of the `FreqOffsetFit` class for use by the phase detrending method in the `Calibration` class. `FreqOffsetFit` deviates from previous conventions of fitting the frequency offset residual for several reasons: (i) some RSR files contain NaNs in the file headers in place of the coefficients needed for calculating the polynomials that describe the predicted sky frequency, which prevented computing the residual frequency offset; (ii) computing the predicted sky frequency over the entire occultation added at least 5 seconds of computation time to each of the 151 RSR files at 1 kHz prior to USO failure; (iii) disagreements between the predicted and reconstructed sky frequency sometimes exceeded 2 MHz and introduced scalloping into the frequency offset, which prevented appropriate description of the frequency offset; (iv) systemically, there is essentially no difference between fitting the frequency offset and the residual frequency offset. Thus, deviating from past conventions not only simplified the pipeline but also enabled support for processing many more files than previously possible.

When instantiated, the `FreqOffsetFit` class begins by instantiating the `calc_freq_offset` class, which contains methods to calculate the frequency offset as a function of time from the raw data. It uses the time (`spm_vals`) and raw signal $I_m + iQ_m$ (`IQ_m`) attributes of `RSRReader` class (`rsr_inst`). Its positional arguments are an instance of the `RSRReader` class (`rsr_inst`) and two floats specifying the minimum and maximum SPM values to use as limits on the range of occultation data for which the offset frequency is computed (this

⁷Limits on the reliability of the normalized power are estimated in the form of threshold optical depth, which is computed by the `calc_tau_thres` class when instantiating `DiffractionLimitedProfile()`.

reduces computation time by eliminating the calculation of offset frequencies that are not useful for constraining the signal offset frequency). Its optional inputs are `dt_freq` for the FFT window size in seconds from which to calculate frequency offset (default is 2 seconds) and `verbose` for a Boolean specifying whether to print out intermediate steps to the terminal (default is `False`).

To calculate frequency offset from the data, we use the `numpy` FFT module to compute the frequency components of the raw measured complex signal for a window in time with a width of `dt_freq` and centered on time `spm_mid`. The submodule `calc_freq_offset` estimates the frequency corresponding to the peak in the power spectrum near the center of the bandwidth in two steps. First, it takes a discrete approach by applying a Hamming filter to the signal within the window, computing the discrete FFT, and obtaining the frequency associated with the maximum power ($|A^2|/N$). Second, to better constrain the frequency offset, `calc_freq_offset` takes a continuous approach, computing the continuous Fourier transform of the Hamming-filtered complex signal at a 0.001 Hz sampling in frequency over a 0.5 Hz window centered on the peak frequency found in the discrete Fourier transform. The frequency associated with the maximum power ($|A^2|/N$) of the continuous Fourier transform is taken to be the carrier frequency offset associated with the time `spm_mid`. The result is stored in an array. Then, the window center is shifted forward in SPM by the sampling resolution value, and the process is repeated. The first window is centered at the initial SPM value plus half with window width and are sampled at the keyword-specified resolution until the window center is half a window width from the end of the time series.

Symbol	Parameter Name	Calibration Attribute
t_{OET}	OBSERVED EVENT TIME	t_oet_spm_vals
$\hat{f}(t)_{sky}$	SKY FREQUENCY	f_sky_hz_vals
$\hat{f}(t)_{offset}$	OFFSET FREQUENCY	f_offset_fit_vals
\hat{P}_0	FREESPACE POWER	p_free_vals

Table 6: Glossary of calibration data in the CAL files.

The `FreqOffsetFit` class then calls `calc_f_sky_recon`, a pythonized version of Nicole Rappaport’s PREDICTS Fortran software, to compute the reconstructed sky frequency. This uses spacecraft ephemerides available in the kernel files and `spiceypy` to determine the implied line-of-sight Doppler shift of the intrinsic USO frequency due to the motion of the spacecraft relative to the observing station. This gives a reconstructed sky frequency, denoted as $f(t)_{dr}$.

With its method `create_mask`, `FreqOffsetFit` performs sigma clipping of the offset frequency to exclude data which are unreliable by creating a boolean mask array. The method begins by masking out all offset frequencies which more than five standard deviations away from the median frequency offset. Then, this method fits the masked data using the `calc_polyorder`. This method fits data with polynomials of iteratively increasing order, using the F-test statistic to evaluate whether a higher order polynomial provides a better fit to the frequency offset data. The method stops when increasing the polynomial order does not provide a better description or when the polynomial order exceeds

nine. The `create_mask` method uses this “best fit” polynomial to perform a second-order sigma clipping by excluding data which are more than five standard deviations away from the “best fit” polynomial. In order to exclude spurious inclusion, `create_mask` performs a “nearest-neighbor” comparisons. The first comparison examines the four nearest data points included by the boolean mask. If the absolute difference between the considered datum and each of its four neighbors exceeds 0.25 Hz, the datum is excluded. The second comparison assumes any datum with at least four excluded nearby data points should also be excluded. Finally, a boolean mask array is returned which ignores all unreliable data.

`FreqOffsetFit` uses its `fit_f_sky_offset` method to fit a polynomial (of order specified by the `calc_polyorder` method to prevent over-fitting) to the masked frequency offset data using the `numpy.polynomial` subpackage. The best-fit polynomial $\hat{f}(t)_{offset}$ is returned along with the reduced summed squared residuals as an estimate of the fit quality.

`FreqOffsetFit` computes $\hat{f}(t)_{offset}$ and stores both the $\hat{f}(t)_{dr}$ and $\hat{f}(t)_{offset}$ variables as attributes. For reference and visual assessment, `FreqOffsetFit` plots the total and sigma-clipped sets of $f(t)_{offset}$ and overplots $\hat{f}(t)_{offset}$. The plot is saved in the appropriate output directory and is named following the `*.TAB` and `*.LBL` conventions with the infix `FORFIT`.

4.3.2 Power Normalization

After phase compensation of the complex signal, the `Calibration` class instantiates the `Normalization` class. The objective of this class is to estimate the intrinsic spacecraft signal power \hat{P}_0 by fitting the observed power in free space regions with a low-order polynomial. A `Normalization` instance requires $I_c + iQ_c$ and an instance of the `Geometry` class (`geo_inst` in the pipeline scripts).

To begin, `Normalization` down-samples the corrected complex signal by a factor of 500. This serves to minimize the effect diffraction patterns in the free space regions might have on the overall prediction of \hat{P}_0 while expediting the fitting procedure.

Next, `Normalization` utilizes the ring gaps and other free space regions predicted by the `find_gaps` function in the `calc_occ_geo` submodule in units of SPM. Free space regions are then excluded or retained by comparing the median power within the SPM limits of the free space region to the maximum power *within* the ring system. Because extinction by the ring material causes a decrease in observed signal power, this effectively select the maximum power observed in gaps and divisions inside of the ring system. When the median power within a given free space region is 50% below or 25% above the maximum ring-system spacecraft signal power, the data from this free space region is excluded. After these comparisons and rejections are made, data from all remaining free space regions are culled for use in a polynomial fit.

Then, the `Normalization` method `fit_freespace_power` fits these reliable measurements of the intrinsic spacecraft signal in order to predict \hat{P}_0 over the entire occultation. The default fit is a third order polynomial; however, the fit order can be changed when instan-

tiating the `Calibration` class using the keyword argument `pnf_order` and the fit type can be set using the keyword argument `pnf_fittype` (default of “poly” for polynomial and “spline” for spline, piece-wise is on lien for future versions). Regardless of user input, if the number of free space regions is less than five, the fitting method forces the fit order to linear.

If the keyword `interact` is set to `True` when instantiating the `Calibration` instance, `Normalization` will enter its interactive mode after computing the best fit to the reliable free space data. In this mode, users can customize the free space regions, fit type, and fit order following prompts in the terminal. The command line will prompt the user to confirm interactive mode and subsequently display a plot in a `matplotlib.pyplot` window showing the best-fit polynomial to the default free space regions. Both the individual regions and the total profile will be displayed so that the user may assess the quality of the fit (in the same manner as the output fit plot, as shown in Figure 15). The command line will then prompt users whether to change included free space regions or revert to the defaults generated by the automated pipeline (see Appendix C for an example). If the user opts to change the free space regions, new definitions for the desired free space regions (specified as pairs of lower and upper SPM limits for each region) will need to be entered in the following format

[[30500,31785],[34080,34245],[35285,36000]]

to be used in the fit. If the new free space region limits are not entered in the appropriate format, the code will revert to the initial freespace regions predicted by `Geometry`.

Once the final fit has been computed, \hat{P}_0 and the boundaries of the free space regions used to compute the fit are stored as attributes. As with the frequency offset fit, this step also includes plotting the fit results for visual inspection and reference. \hat{P}_0 is plotted over the phase-compensated signal power for both the entire profile and for each individual free space region used in the fit. Naming convention follows that of the `*.TAB` and `*.LBL` files with the infix `FSPFIT`.

4.4 Diffraction-limited profile routines

Symbol	Parameter Name	NormDiff Attribute
ρ	RING RADIUS	rho_km_vals
$\Delta\rho_{IP}$	RADIUS CORRECTION DUE TO IMPROVED POLE	rho_corr_pole_km_vals
$\Delta\rho_{TO}$	RADIUS CORRECTION DUE TO TIMING OFFSET	rho_corr_timing_km_vals
ϕ_{RL}	RING LONGITUDE	phi_rl_rad_vals
ϕ_{ORA}	OBSERVED RING AZIMUTH	phi_ora_rad_vals
τ	NORMAL OPTICAL DEPTH	tau_norm_vals
ϕ	PHASE SHIFT	phase_rad_vals
τ_{TH}	NORMAL OPTICAL DEPTH THRESHOLD	tau_threshold_vals
t_{OET}	OBSERVED EVENT TIME	t_oet_spm_vals
t_{RET}	RING EVENT TIME	t_ret_spm_vals
t_{SET}	SPACECRAFT EVENT TIME	t_set_spm_vals
B	OBSERVED RING ELEVATION	B_rad_vals

Table 7: Glossary of optical depth, phase shift, and selected geometry parameters contained in the DLP files.

When instantiated, the `DiffractionLimitedProfile` class normalizes the power profile using the results from the `Calibration` step and resamples it with respect to uniformly-spaced ring radius at a sample spacing $\Delta\rho$. The `DiffractionLimitedProfile` class requires all previous class instances as arguments: the `RSRReader` class (`rsr_inst`), the `Geometry` class (`geo_inst`), and the `Calibration` class (`cal_inst`). The final positional argument `dr_km` specifies the radial sample spacing of the profile. Keyword arguments consist of the `verbose` – a boolean specifying verbose mode (default is `False`), `write_file` – a boolean specifying whether to write the DLP out to a CSV file (default is `True`), and `profile_range` – a 1×2 list setting the lower and upper limits of the DLP (default is `[65000, 150000]`; we highly recommended users proceed with these default limits).

When instantiated, the `DiffractionLimitedProfile` calls the `resample_IQ` module to resample the phase-detrended signal $I_c + iQ_c$ with respect to ring radius over the range `profile_range` at radial spacing of `dr_km`. In the event that the user-provided `dr_km` spacing is finer than the average raw radial spacing, the resampling routine will compute the lowest possible sample spacing and resample the complex signal to this spacing instead of `dr_km`. If this occurs, the user will be notified in the command line, and all documentation of the diffraction-limited profile will reflect the sample spacing used rather than the sample spacing specified by `dr_km`.

From the resampled detrended complex signal, `DiffractionLimitedProfile` computes the phase

$$\phi_c = \arctan(Q_c/I_c) \quad (3)$$

and normalized complex signal

$$\hat{T} = \hat{T}_R + i\hat{T}_I = (I_c + iQ_c)/\hat{P}_0, \quad (4)$$

the latter following from Equation 2 in [Marouf et al. \(1986\)](#) and Equation 20 in [Asmar et al. \(2018\)](#). It is from \hat{T} that the DLP normal optical depth τ_{DLP} is calculated by:

$$\tau_{DLP} = -\sin(|B|) \ln(\hat{T}) \quad (5)$$

Finally, `DiffractionLimitedProfile` calls the `calc_tau_thresh` submodule to compute the threshold optical depth τ_{TH} , a proxy for the maximum reliable value of τ_{DLP} .

For the DLP instance to be of use in the subsequent diffraction reconstruction step of the pipeline, $\dot{\rho}$ can only have one sign. Chord and proximal orbit occultations thus present a problem if not handled appropriately when creating the DLP object.

`DiffractionLimitedProfile` handles this issue by using a class method to return an object instance of itself for each direction of the occultation as indicated by the sign of $\dot{\rho}$. For the user, this means that instantiating the `DiffractionLimitedProfile` object class will always return two objects: one for ingress and one for egress (always in this order). For diametric occultations, the object corresponding to the direction *not* included in the observation will be a `None` object in place of a DLP instance. For chord occultations, the user will receive two DLP object instances, one for ingress and one for egress, split where a sign change in $\dot{\rho}$ occurs. Each DLP object can be passed individually to the diffraction reconstruction step in the pipeline.

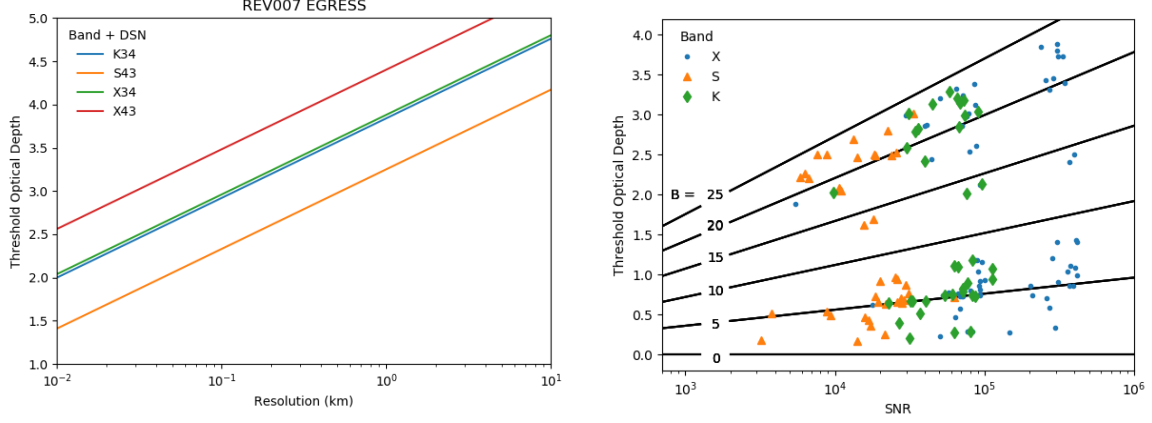


Fig. 6: *Left*: Resolution dependence of τ_{TH} for Rev 007 E. *Right*: Effect of opening angle B on τ_{TH} as a function of SNR_0 for $\Delta\rho = 0.25$ km. Data are for all 1 kHz files prior to USO failure

4.4.1 Threshold Optical Depth

The threshold optical depth is an estimate of the maximum reliable value of optical depth implied by the signal-to-noise ratio SNR_0 . The noise is estimated from the STFT of the entire observation of raw $I + iQ$. After computing the STFT, we average the power spectrum over the frequency ranges -450 to -200 Hz and 200 to 450 Hz so as to eliminate both the spacecraft carrier signal and the rolloff at the edges of the power spectrum. Then, we average the frequency-averaged power over the first and last thousand seconds of the observation to exclude possible contamination from scattered spacecraft signal while maintaining a large number of samples of the noise. The intrinsic signal \hat{P}_0 is assumed to be the fit to the freespace regions computed in the calibration step and stored as an attribute of the `DiffractionLimitedProfile` class. We consider the ratio of \hat{P}_0 and the average noise to be SNR_0 .

Following Equations 24-26 in Marouf et al. (1986) and Equations 26 and 27 in Asmar et al. (2018), we compute the threshold optical depth using

$$\tau_{TH} = -\sin(|B|) \ln \left[\frac{C_\alpha \dot{\rho}}{2\text{SNR}_0 \Delta\rho} \right] \quad (6)$$

where B is the ring opening angle and $\dot{\rho}$ is the ring intercept radial velocity as computed by and attributed in the `Geometry` class, $C_\alpha = 2.41$ is chosen to correspond to 70% confidence⁸, and $\Delta\rho$ is the radial spacing. For the DLP class, this is simply the `dr_km` positional argument specified when instantiating the DLP class and has a default of 0.25 km. For the reconstructed profile, $\Delta\rho$ will depend on the window size W and Fresnel scale F , the so-called *processing resolution* $\Delta R = 2F^2/W$. We show the dependence of τ_{TH} on resolution in Figure 6 (cf. Fig. 8, Marouf et al. 1986). Also in Figure 6 is a demonstration of the effects of both opening angle and SNR on τ_{TH} for all 1 kHz RSR files prior to USO failure.

⁸From Marouf et al. (1986) Equation 25, this is given by the root-mean-square of the difference between the measured complex signal $X = T + n$ (where n is the additive noise from the receiver) and actual complex signal T , normalized with respect to the expected receiver noise P_N .

4.5 Diffraction reconstruction routines

Symbol	Parameter Name	Attribute Name
ΔR	RECONSTRUCTION RESOLUTION (KM)	res
ρ	RING RADIUS	rho_km_vals
$\Delta\rho_{IP}$	RADIUS CORRECTION DUE TO IMPROVED POLE	rho_corr_pole_km_vals
$\Delta\rho_{TO}$	RADIUS CORRECTION DUE TO TIMING OFFSET	rho_corr_timing_km_vals
ϕ_{RL}	RING LONGITUDE	phi_rl_rad_vals
ϕ_{ORA}	OBSERVED RING AZIMUTH	phi_rad_vals
τ	NORMAL OPTICAL DEPTH	tau_vals
ϕ	PHASE SHIFT	phase_rad_vals
τ_{TH}	NORMAL OPTICAL DEPTH THRESHOLD	tau_threshold_vals
t_{OET}	OBSERVED EVENT TIME	t_oet_spm_vals
t_{RET}	RING EVENT TIME	t_ret_spm_vals
t_{SET}	SPACECRAFT EVENT TIME	t_set_spm_vals
B	OBSERVED RING ELEVATION	B_rad_vals
w	WINDOW WIDTH FOR RECONSTRUCTION	w_km_vals
f_{sky}	SKY FREQUENCY	f_sky_hz_vals
F	FRESNEL SCALE	F_km_vals
D	SPACECRAFT-RIP DISTANCE	D_km_vals
\hat{T}	DIFFRACTED COMPLEX TRANSMITTANCE	T_hat_vals
T	RECONSTRUCTED COMPLEX TRANSMITTANCE	T_vals

Table 8: Glossary of parameters contained in RSS_2005_123_X43_E_TAU_01KM.TAB. See companion label (.LBL) files for description of the data.

All of the diffraction reconstruction tools are located within the **diffrec** subpackage of **rss_ringoccs**. One can find simple tools for diffraction reconstruction of the diffraction pattern contained in an instance of the **NormDiff** class, as well as more complex tools for modeling problems and comparing them with real-world data and geometry. The main dependency is the numpy package, but some functions also rely on tools found in the scipy package. The subpackage is broken into four submodules:

- **advanced_tools**
- **diffraction_correction**
- **special_functions**
- **window_functions**

The **diffraction_correction** submodule is the primary tool within **diffrec** and contains the **DiffractionCorrection** class, which is the main utility for creating diffraction-reconstructed profiles of radio occultation observations. The Python syntax is as follows:

```
In [1]: from rss_ringoccs import diffrec
In [2]: rec = diffrec.DiffractionCorrection(norm_inst, res)
```

Here, **norm_inst** is an instance of the **DiffractionLimitedProfile** class containing the diffracted data and **res** is the user-requested processing resolution of the reconstructed profiles in kilometers and expressed as a floating point number. There are several keywords in this class to allow the user to specify how the diffraction reconstruction will be performed.

A new feature of `rss.ringoccs` is the use of special polynomials within the diffraction reconstruction steps to greatly reduce computation time without sacrificing accuracy. The *Fresnel kernel*, $\psi(\rho, \rho_0; \phi, \phi_0)$, needs to be computed at its *stationary* value, ψ_s :

$$\frac{\partial \psi_s}{\partial \phi} = 0 \quad (7)$$

We expand ψ in a Taylor expansion about the point ϕ_0 , and obtain the following:

$$\psi = \sum_{n=0}^{\infty} \psi^{(n)}(\rho, \rho_0; \phi_0, \phi_0) \frac{(\phi - \phi_0)^n}{n!} \quad (8)$$

The notation $\psi^{(n)}$ represents the n^{th} partial derivative of ψ with respect to ϕ . The tuple $(\rho, \rho_0; \phi_0, \phi_0)$ is to indicate that the coefficients of the Taylor expansion are *functions of the other variables*, and are not constants. We apply Newton-Raphson to find the stationary value of ψ , and obtain in the first perturbation the following value:

$$\phi_s = \phi_0 - \frac{\psi'(\rho, \rho_0; \phi_0, \phi_0)}{\psi''(\rho, \rho_0; \phi_0, \phi_0)} \quad (9)$$

Where, again, derivatives are taken with respect to ϕ . Truncating Eqn. 8 at the quadratic term, and evaluating at $\phi = \phi_s$, We obtain:

$$\psi_s \approx \psi(\rho, \rho_0; \phi_0, \phi_0) - \frac{1}{2} \frac{\psi'(\rho, \rho_0; \phi_0, \phi_0)^2}{\psi''(\rho, \rho_0; \phi_0, \phi_0)} \quad (10)$$

The nature of ψ allows for the right-hand side of Eqn. 10 to be expressed exactly in terms of Legendre polynomials. Stopping the Legendre expansion at the quadratic gives the classic *Fresnel Approximation*. This is:

$$\psi_s = \frac{\pi}{2} \left(\frac{\rho - \rho_0}{F} \right)^2 \quad (11)$$

For historical reasons and due to the use of Legendre polynomials, this expansion is henceforth labelled as Fresnel-Legendre polynomials. `rss.ringoccs` allows for several different polynomial powers of this expansion by using the `psitype` keyword. This is a `string`, and the following inputs are allowed:

- ‘Fresnel’ - Classic Fresnel approximation.
- ‘Fresnel3’ - Legendre expansion up to the cubic term.
- ‘Fresnel4’ - Legendre expansion up to the quartic term.
- ‘Fresnel6’ - Legendre expansion up to the sextic term.
- ‘Fresnel8’ - Legendre expansion up to the octic term.
- ‘Full’ - No approximation, Newton-Raphson is applied.

For all but Rev133, the ‘Fresnel4’ option is more than adequate, and can reproduce the PDS results, as well as mimic the results obtained by using ‘full’. Unless overruled explicitly by the user, this is the default option. For Rev 133 observations, when the rings were observed nearly edge-on, a known problem with the cubic term of ψ persists, and the

Fresnel-Legendre polynomials cannot adequately capture this. This is, in part, because the Fresnel-Legendre expansion assumes that the first iterate of the Newton-Raphson approximation is sufficient, whereas in such extreme geometries it is not. Below is a detailed description of the example included in the `quick_look_run.py` script discussed in Section 3.4. As with the quick-look script itself, this example assumes that the user has available the requisite `*.TAB` files generated by the example end-to-end script covered in Section 3.3. The `ExtractCSVData` class covered in the next section can be used to extract information from the `*.TAB` files and reconstruct the DLP instance `norm_inst`.

```
In [1]: geo = "../Data/GEO.TAB"
In [2]: cal = "../Data/CAL.TAB"
In [3]: dlp = "../Data/DLP.TAB"
In [4]: from rss_ringoccs.tools import ExtractCSVData
In [5]: from rss_ringoccs import diffrec
In [6]: norm_inst = ExtractCSVData(geo, cal, dlp, verbose=False)
In [7]: tau_inst_f = diffrec.DiffractionCorrection(
...: norm_inst, 1.0, rng="all", psitype="fresnel")
Computation Time: 12.491324
```

Note the quite short (12 second) reconstruction time, due in part to the ‘Fresnel’ setting for `psitype` and 1 km resolution. Running `DiffractionCorrection` with `psitype=‘full’`:

```
In [8]: tau_inst_v = diffrec.DiffractionCorrection(
...: data, 1.0, psitype="full", verbose=True)
...: Things print out here...
Computation Time: 108.427886
In [9]: tau_inst = diffrec.DiffractionCorrection(
...: data, 1.0, psitype="full")
Computation Time: 76.470007
```

There are several things to note here. The computation of ψ is one of the slowest parts in the entire diffraction reconstruction algorithm. In particular, the computation of the *stationary phase* can be quite time-consuming. Since ‘Fresnel’ skips all of this, we see a substantial reduction in computation time. The data set that was used in this example comes from the Cassini rev007E occultation observation. We can check the validity of Fresnel approximation for this set by looking at the difference in the two reconstructions. Note that since the power is normalized to 1, this is both fractional and absolute error:

```
In [10]: import numpy as np
In [11]: np.max(np.abs(recf.power_vals-rec.power_vals))
Out[11]: 0.00016784579326811766
In [12]: np.mean(np.abs(recf.power_vals-rec.power_vals))
Out[12]: 3.5374768401331293e-06
```

The Fresnel approximation works very well here. The default setting of `psitype=‘Fresnel4’` is slightly slower than ‘Fresnel’, but still finishes execution in less than 20 seconds. For many occultations where ‘Fresnel’ fails, ‘Fresnel4’ still produces accurate results. Another thing to note is that there is a large discrepancy in the computation time for when `verbose=True` and `verbose=False` is set. Since `verbose` prints out pieces of information for every point that is being reconstructed, the Python interpreter needs to wait for the line to be printed before it can process the next point. In most cases this is not an issue, but when speed is crucial it may be better to leave `verbose` set to `False`.

4.6 Utility routines

Within the `tools/` subpackage one can find various `pds3*_series.py` scripts and routines to read and write PDS3-type data and label files.⁹ `rss_ringoccs` manages the file naming conventions and calling of the `pds3*_series.py` scripts by means of the `write_output_files.py` script.

Also included as a utility for the quick-look pipeline is the `ExtractCSVData`, a tool for extracting information from the `*.TAB` files and reconstructing the DLP instance `norm_inst`. This can be found in the `CSV_tools` submodule of the `tools` subpackage, all contained within `rss_ringoccs`. The steps for importing are shown below.

The `ExtractCSVData` imports user-specified `*.TAB` files produced by the end-to-end pipeline and uses them to construct an instance of the `NormDiff` class that can then be passed on for diffraction reconstruction as a part of the quick-look process. This way, the pipeline only needs to be run once on a given data set, and then the user may experiment with different resolutions, window functions, etc., on the diffracted data without repeated time consuming steps. For example, from the master directory of `rss_ringoccs`, one could use `ipython` to implement the following to construct a `DiffractionLimitedProfile` instance from `*.TAB` files output from a previous run:

⁹The products of these routines have not been thoroughly checked for PDS3 compliance.

```

cd ~/Research/rss_ringoccs-master
ipython
In [1]: path = '../output/Rev007/E/Rev007E_RSS_2005_123_X43_E/'
In [2]: geo = path+'RSS_2005_123_X43_E_GEO_20180926_0001.TAB'
In [3]: cal = path+'RSS_2005_123_X43_E_CAL_20180926_0001.TAB'
In [4]: dlp = path+'RSS_2005_123_X43_E_DLP_0100M_20180926_0001.TAB'
In [5]: from rss_ringoccs.tools.CSV_tools import ExtractCSVData
In [6]: from rss_ringoccs import diffrec
In [7]: dlp_inst = ExtractCSVData(geo, cal, dlp, verbose=True)
Extracting Data from CSV Files:
    Extracting Geo Data...
    Geo Data Complete.
    Extracting Cal Data...
    Cal Data Complete.
    Extracting DLP Data...
    DLP Data Complete
    Retrieving Variables...
    Computing Variables...
    Interpolating Data...
    Data Extraction Complete.
    Writing History...
    History Complete.
    Extract CSV Data Complete.

```

Also note that it is possible to combine *.TAB files from different end-to-end runs of `rss_ringoccs`, which is encouraged for users who want to examine the effect different calibration inputs might have on the final reconstructed optical depth profile.

The `CompareTau` tool, found in the `advanced_tools` submodule of the `diffrec` subpackage, imports user-specified *.TAB files produced by the end-to-end pipeline and runs the diffraction reconstruction at an additional user-specified spatial resolution for the purpose of comparing optical depth profiles for the same occultation with the only difference being the processing resolution.

4.6.1 PDS3Reader

The `PDS3Reader` function allows users to easily read in a set of PDS3-formatted data and label files without having to scroll through and examine a label file in detail (for column headers, etc.). To use the function, first verify that the desired data and label file are within the same directory. Then, follow the example below but replace `lbl_file` with the path to the desired label file.

```

In [1]: import rss_ringoccs as rss
In [2]: lbl_file = 'RSS_2005_123_X43_E_GEO_20180926_0001.LBL'
In [3]: geo = rss.tools.PDS3Reader(lbl_file, use_attr_names=False)

```

The `geo` variable will contain two classes as attributes: `series` and `keywords`. `series` refers to the actual data set, and calling `geo.series.__dict__.keys()` will print all the data column headers, as seen in the label file. By setting the keyword argument `use_attr_names` to `True`, these column header names will be replaced by the attribute name used in `rss_ringoccs` to represent the same variable (see Tables 5-7). To grab the data associated with the column header, simply call `geo.series.XX`, where `XX` is the header name. `keywords` contains all of the keywords listed in the label file, and the corresponding keyword values can be called in a similar fashion: `geo.keywords.XX`,

where **XX** is the keyword name. Definitions to these keywords should be available on the PDS data dictionary <https://pds.nasa.gov/tools/dd-search/>.

4.6.2 Label History

Each label file produced by `rss_ringoccs` will contain a `PROCESSING_HISTORY_TEXT` that lists user information (such as Python version, operating system, etc.) as well as an accumulated list of all positional and keyword arguments that were used to run the pipeline up to part where the label was created. Users can visually read this list and rerun the pipeline to reconstruct the same profile. Alternatively, users can use the `read_history` function, which will automatically read in the listed inputs and output a requested instance, if available. For example, a `*TAU.LBL` file will contain all of the inputs to the `RSRReader`, `Geometry`, `Calibration`, `DiffractionLimitedProfile`, and `DiffractionCorrection` classes. Using `read_history` on the `*TAU.LBL` will give users the option of returning an instance of any of the above classes.

This function is useful in particular for those who wish to reproduce a `dlp_inst` sampled at finer radial intervals.

4.6.3 Science tools

Some preliminary modeling tools are provided here for use with the data products from `rss_ringoccs`. The script `dtau_miescatt_partsize_grid.py` is one such tool. This script can be run from anywhere in the software directory structure and has no dependencies on other `rss_ringoccs` scripts. However, this tool does require the use of an external software, `PyMieScatt`, which can readily be installed using `pip` as shown here:

```
pip install pymiescatt
```

Following [Marouf et al. \(1983\)](#), this script calculates predictions for observed differential optical depth assuming a power law distribution of particle sizes. Users can specify the ring material index of refraction, range of particle sizes, and power law indices by editing the preamble in the script file. Running the script will output a CSV file and plot, both containing the differential optical depths predicted by Mie scattering physics for the given index of refraction, particle sizes and distributions. These predictions are intended for comparison with differential optical depths calculated from reconstructed normal optical depth profiles output by the `rss_ringoccs` pipeline.

4.7 Incoherent signal routines

All of the routines to produce the incoherent signal results are in the `rss_ringoccs/scatter/` directory in the `rss_ringoccs` package. One tool performs the STFT spectrogram calculations and stacking to improve SNR. The other is a tool for users to read in results from the spectrogram output file. The `spectrogram.py` script contains the object class `Scatter`. Instantiating this object requires instances of the `RSRReader`, `Geometry`, and `Calibration` classes. `Scatter` uses the `spectrogram` method from the `scipy.signal` module with a Hamming window to compute the discrete short-time Fourier transform of the phase-corrected complex signal over the entirety of the occultation. Then, its `stack_spec` method stacks the spectrogram slices to boost the SNR of the incoherent signal, which is only 1-2 dB above the noise threshold. Following `rss_ringoccs` file nomenclature, `Scatter`

outputs the spectrogram and ancillary information (e.g., radius, frequency, and time dimensions) to comma-delimited files. We provide the script `spectro_reader.py` as a tool for reading the output spectrogram and ancillary reference files to build the spectrogram, time, frequency, and radius arrays. As a demonstration for reading and visualizing the incoherent signal, we provide an example script `spectrogram_plot_example` in the `examples` directory which reads in scattered signal data products and normal optical depth profiles from the `output` directory and plots the results.

5 Validation of `rss_ringoccs` algorithms

Here we present tests and validations of the performance and output of `rss_ringoccs`. Perhaps most significantly, this includes a comparison of `rss_ringoccs` results with those published on the PDS as these results were processed using state-of-the-art methods following [Marouf et al. \(1986\)](#). This includes a demonstration of the results for Cassini and for Voyager, as well as reproduction of analytical solutions, results from raw data sampled at different rates, and results from DLPs sampled at different rates.

5.1 Comparison with results on PDS

5.1.1 Cassini RSS results

The results of `rss_ringoccs` match those in PDS CORSS_8001 extremely well. After completing the Huygens ringlet end-to-end example in Section 3.3, several data products will be created. In Figs. 7-8, we compare our output GEO file, plotted in blue, with the PDS GEO file, plotted in dashed red lines. These values are, for our purposes, essentially the same – the comparisons without grey dots indicating a fractional error match exactly to the precision listed in the files.

The calibration steps require more complex calculations and some user judgment and thus our results do not match the PDS as well as the geometry does, but overall these differences are small and almost unavoidable. A major part in the calibration step is to normalize the signal power such that it reaches unity within free-space regions, but the fit type and order rely heavily on user choices (see sec xx). So, even though Figure 9 looks as though the signal power outside of the ring system is at unity, indicating a good fit, if we zoom into a feature and compare our final signal power to that on the PDS, the unity background signals are slightly offset. This is seen in the Maxwell ringlet but not the Huygens ringlet.

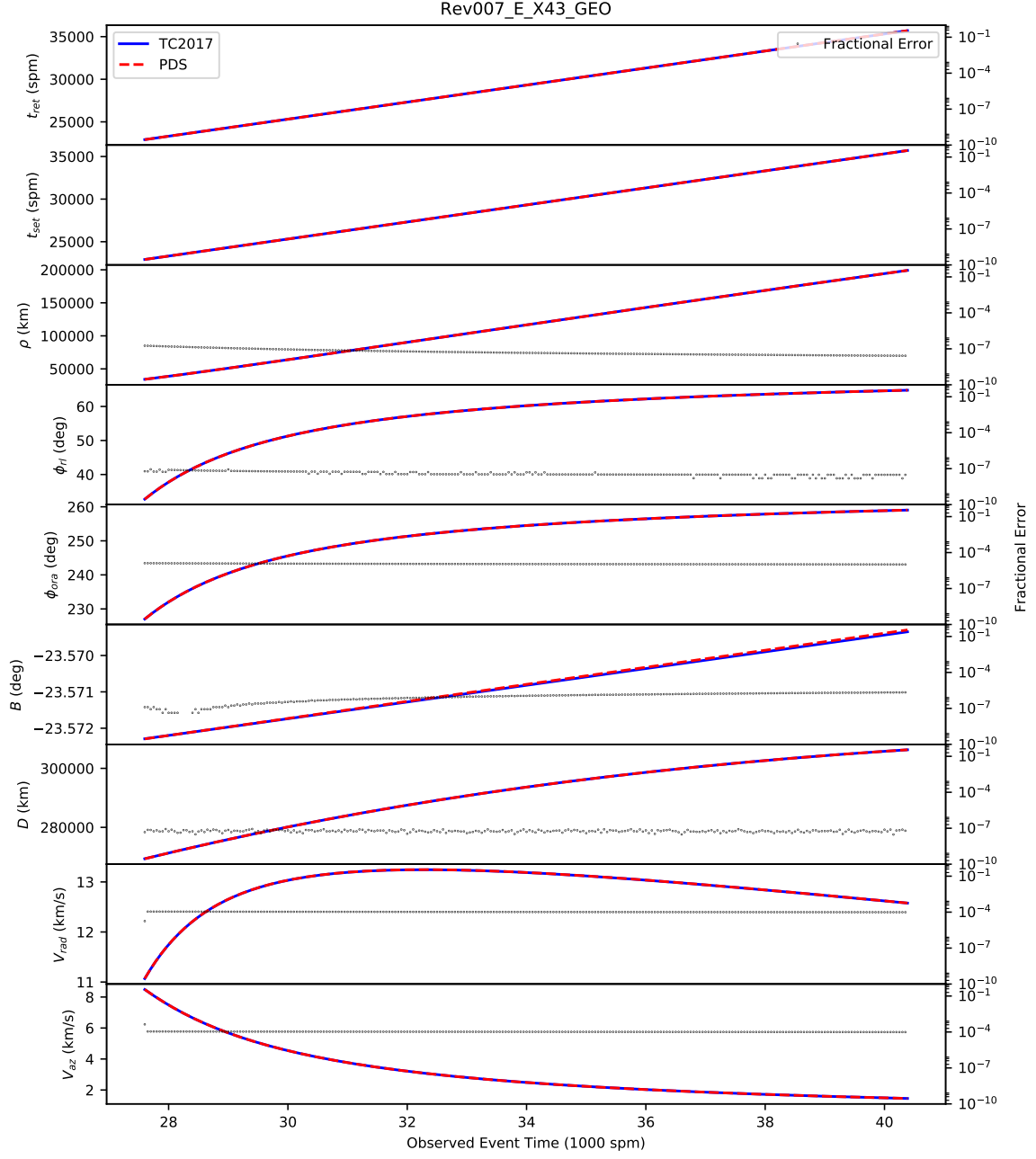


Fig. 7: Comparison of geometry parameters for Rev007 Egress X-band as viewed from DSS-43. PDS CORSS_8001 data product Rev007/Rev007E/Rev007E_RSS_2005_123_X43_E/RSS_2005_123_X43_E_GEO.TAB values are plotted in dashed red lines, with `rss_ringoccs` values overplotted in blue. Fractional error between the two are plotted on the right-hand y-axis in grey dots.

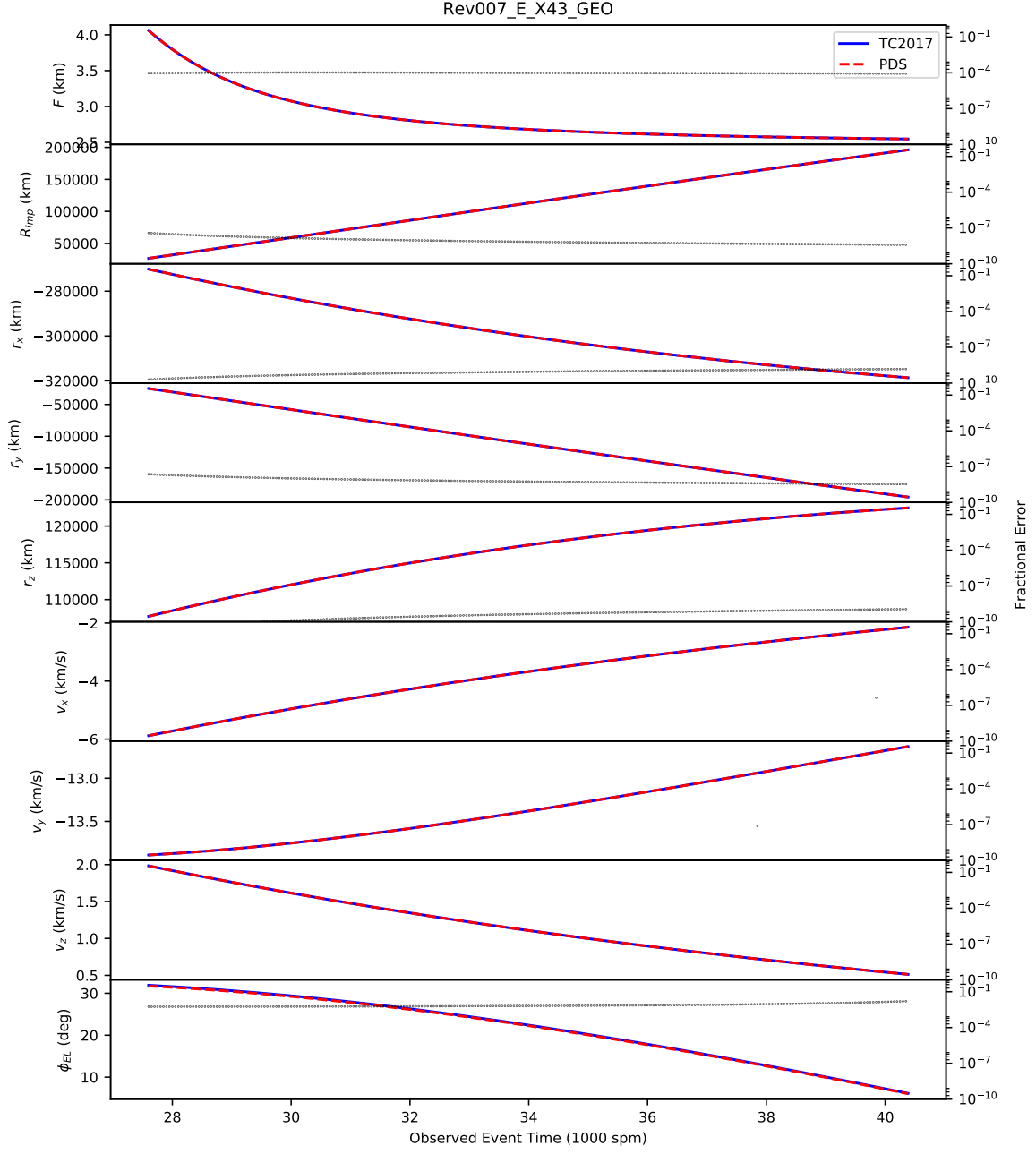


Fig. 8: Comparison of geometry parameters for Rev007 Egress X-band as viewed from DSS-43. PDS CORSS_8001 data product Rev007/Rev007E/Rev007E_RSS_2005_123_X43_E/RSS_2005_123_X43_E_GEO.TAB values are plotted in dashed red lines, with `rss_ringoccs` values overplotted in blue. Fractional error between the two are plotted on the right-hand y-axis in grey dots.

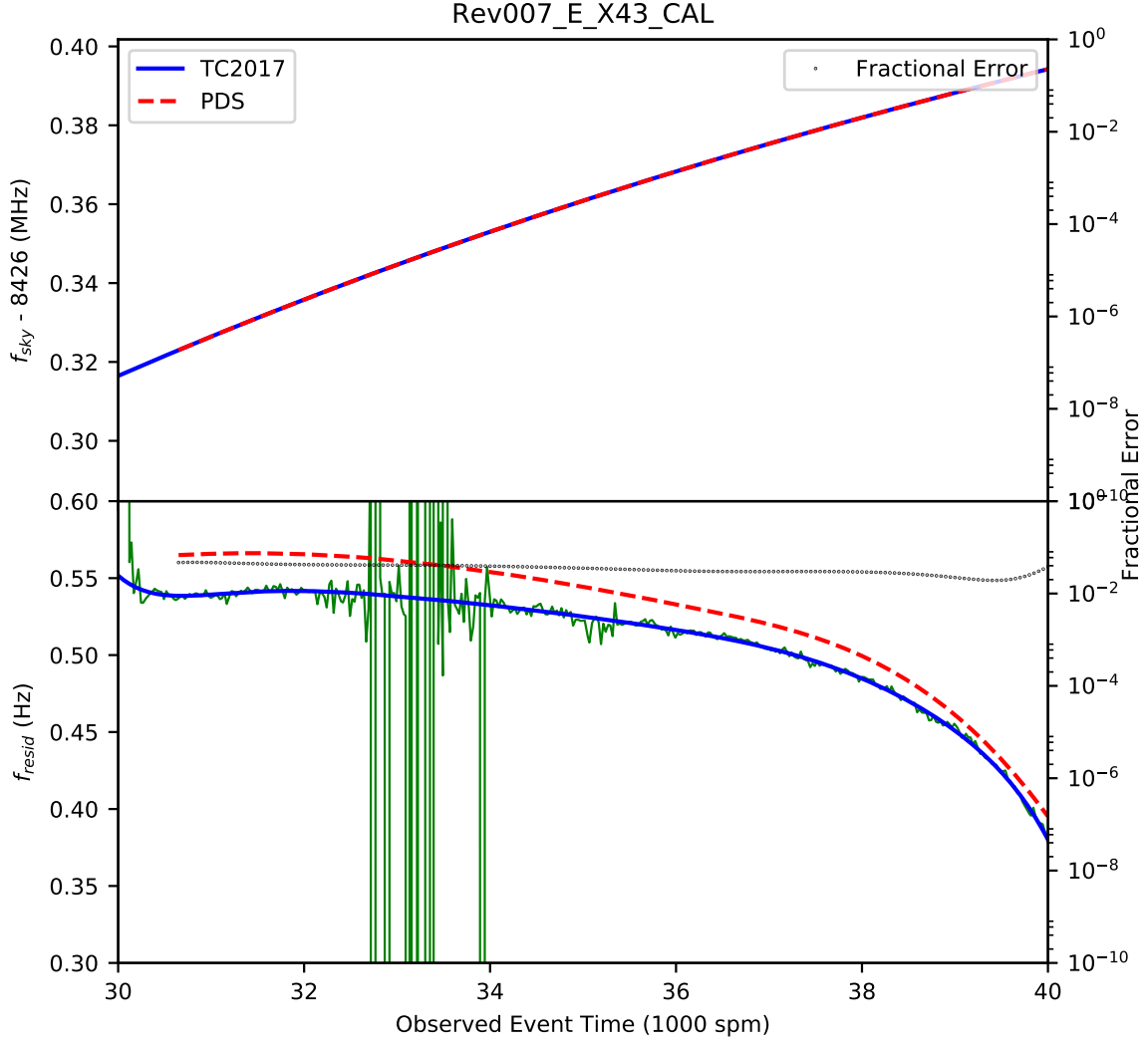


Fig. 9: Comparison of the Frequency Offset and Residual Frequency computed for the Rev007 E X43 data set.

If Fig. 9 we plot the output of the calibration steps of the pipeline for the Rev007 E X43 data set. Plotted with this are the results found on the PDS, as well as the fractional error between the two. Below in Fig. 10 we plot the raw power that is extracted from the RSR files:

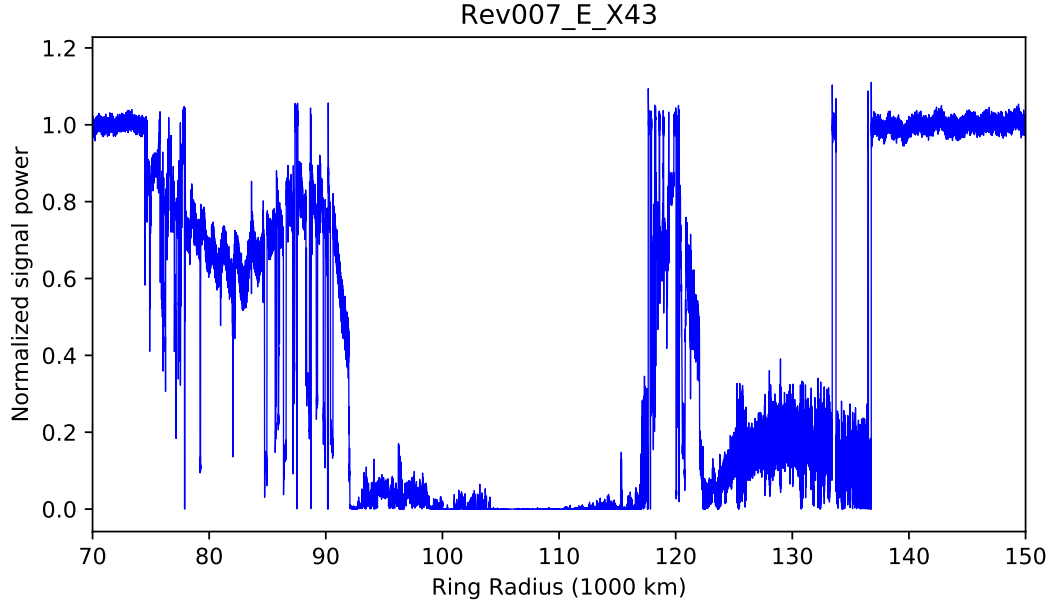
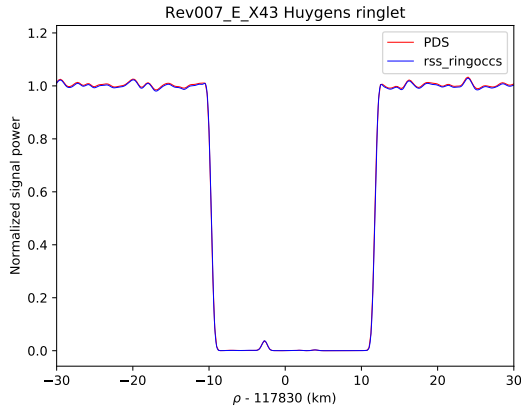
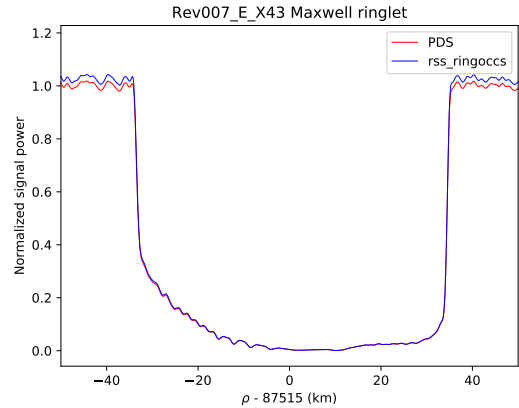


Fig. 10: Raw Power from Rev007 E X43

Finally, at the end of the pipeline we obtain the reconstructed profiles of the Saturnian ring system. It is important to note that the normalization step can be very subtle. As an example of this, we show the reconstruction of Rev007 E X43 in Fig 11 about the Huygens ringlet and the Maxwell ringlet. The default fit was used to create the normalized power. The fit and reconstruction worked extremely well for Huygens, but there is a slight discrepancy for the Maxwell ringlet. Adjusting the polynomials used in the spline fit will alleviate this error, and it is up to the user to decide when the fit is sufficient for their needs.



11.1: Huygens ringlet reconstructed normalized power.



11.2: Maxwell ringlet reconstructed normalized power.

Fig. 11: Comparison of reconstructed normalized power with PDS results. PDS results are plotted in red and can be found from Rev007/Rev007E/Rev007E_RSS_2005_123_X43_E/RSS_2005_123_X43_E-TAU_01KM.TAB). rss_ringoccs results are in blue.

5.2 Effect of different sampling rates

The raw data recorded at the DSN stations are sampled at 16 kHz. The high resolution of these data poses two difficulties: (i) each RSR data file is at least many hundreds of

megabytes in size for an ingress or egress diametric occultation (chord occultations are even larger), which can be cumbersome especially for users downloading a large number of RSR files; (ii) the `rss_ringoccs` processing time on a conventional laptop for each data file can be at least thirty minutes for diametric occultation (even longer for chord occultations), which is particularly computationally expensive for users looking to reduce data for numerous occultations. The PDS provides RSR data files with the raw data down-sampled to 1 kHz. These files are twelve to sixteen times smaller (lower limit of tens of megabytes instead of hundreds) and take 90 to 180 seconds for `rss_ringoccs` to process.

The computational benefits of utilizing the 1 kHz files over the 16 kHz files is clear. However, we must demonstrate that the down-sampled data yield results comparable to those of the raw high-resolution data. To begin this validation, we process 1 kHz and 16 kHz data files for X band at high ring opening angle ($B \approx 23^\circ$) and Ka band at low ring opening angle ($B \approx 5^\circ$). Then, we compare and contrast the 1 kHz and 16 kHz processing results from the calibration, DLP, and reconstruction steps as shown in Fig. 12.

The frequency offset and freespace power fits differ by less than 0.1% within the ring system, while the computed frequency offsets differ by less than 0.01%. The DLP optical depth values are typically within 0.01 dB of one another, although this is not the case for the part of the profile associated with the B ring. Any differences seen between the two profiles can be ascribed to changes in the diffraction pattern when the profile is down-sampled to the DLP radial sampling resolution. This assertion is substantiated by the similarity of the diffraction-corrected optical depth profiles. Because the diffraction reconstruction eliminates diffraction patterns in the profile, the excellent agreement of the two reconstructed profiles indicates that the disparity between the DLP optical depths is due to differences in the diffraction patterns.

Having been demonstrated to yield results comparable to those from the 16 kHz files, the 1 kHz files are recommended for processing instead of the 16 kHz files. Users can obtain 1 kHz files for the majority of occultations from the PDS with only a few exceptions. In these cases, we recommend the user decimate the 16 kHz file to 1 kHz at the start of the pipeline by setting the `decimate_16khz_to_1khz` keyword argument equal to `True`.

6 Voyager 2 Uranus occultation

The Voyager 2 X-band (3.6 cm) radio occultation of the Uranian rings on January 24-25, 1986 provides the highest resolution observations for the Uranian ring system to date. Definitive analysis of the Voyager 2 occultation was performed by [Gresh et al. \(1989\)](#) and the final results were archived at 50m resolution. Best Earth-based data are diffraction-limited to 1-2 km, inhibiting detailed analysis of rings. Voyager 2 radio occultation SNR is high enough to push resolution to 20m to reveal more internal structure of the Uranian rings, opening up possibilities for new science. This presents a unique opportunity to demonstrate the applicability of the `rss_ringoccs` reconstruction software to contexts outside of the Cassini radio occultations of Saturn’s rings with readily-available validation of the results.

6.1 Changes to the procedure

While the procedure for processing the Voyager 2 radio occultation data is comparable to the approach taken by `rss_ringoccs` for the Cassini radio occultations, there are some key differences which required adapting some components of the pipeline.

1. Data are stored in binary files which are not formatted in the same manner as the RSR files for Cassini radio occultations stored on the PDS. This requires a unique file reader.
2. The imaginary component of the complex signal has been lost and must be inferred from the real component.
3. The Uranian ring system is highly inclined. While enabling the high SNR necessary for high-resolution reconstruction, this changes the occultation geometry.
4. Each ring of Uranus has a different inclination and eccentricity, which requires a different treatment of the occultation geometry for each ring to appropriately calculate the ring intercept radius in each ring frame (for the Saturnian rings, `rss_ringoccs` computes the geometry for Saturn's equatorial reference frame). This requires processing each ring individually.
5. The oscillator on board the Voyager 2 spacecraft did not possess the same quality or precision as the USO on board the Cassini spacecraft. This requires adjustments to the phase corrections using the frequency offset, as well as an additional step in correcting the signal phase.
6. Each ring is accompanied in both radial directions by substantial amount of free space power, which eliminates the need for gap finding but does require fitting free space power local to each feature.
7. Output file documentation, i.e., the LBL files, must be adapted to reflect the occultation observation information and processing history.

We provide an adaptation of our software with the v1.2 release of `rss_ringoccs` which addresses the majority of these issues.

6.1.1 Changes to reading raw data

The original 50 kHz occultation data were recorded on 9-track tapes when received by the DSN stations in Parkes and Canberra. Dick Simpson transferred these data to CDs, then to DVDs, and finally to local drive storage. To obtain these data files, submit a request to Dick French, reachable at rfrench@wellesley.edu.

We have designed an object class which, when instantiated, reads in the binary files from the local drive. Unfortunately, the imaginary component of the complex signal has been lost. To recover the complete complex signal, the Uranus file reader class uses a Hilbert transform to compute Q from the real component I stored in the binary file.

6.1.2 Changes to Geometry

The `Geometry` class continues to utilize `spiceypy`; however, this requires a different set of the kernel reference files to compute the occultation geometry. Most of these kernel files are provided with the v1.2 release. Those not included are large (> 100 Mb) and are

also not available from the PDS. These kernel files are available upon request from Dick French, reachable at rfrench@wellesley.edu.

What distinguishes the Uranus version of `Geometry` from that of `rss.ringoccs` is its processing of the occultation geometry for each ring occultation in its own reference frame rather than the host planet’s equatorial reference frame. This enables more accurate calculation of the ring intercept radius.

6.1.3 Changes to Calibration

Updates to the `Calibration` step involve two components of phase corrections. The frequency offset correction original to `rss.ringoccs` is estimated assuming the frequency offset falls in a 300 Hz window centered on 0 Hz with 0.5 Hz resolution in the discrete Fourier transform. For the Voyager 2 occultation, the frequency offset drifts over a 500 Hz window centered on -11 kHz. Additionally, the ring locations can be calculated from pre-existing orbital parameters, allowing us to mask the frequency offset to exclude any effects of the rings on the computed frequency offset from the fit to the frequency offset. Fitting the frequency offset is done before processing each individual ring because the frequency offset is not ring specific. Fitting the frequency offset could not be done with a single polynomial because of a cusp which occurs in both the ingress and egress occultations. Our solution to this was to split the frequency offset data into two sets where the cusp occurs in SPM and fit each set separately. While [Gresh et al. \(1989\)](#) fit each set of frequency offset values with a second order polynomials, we find the frequency offset is better described by a ninth order polynomial.

Each ring is then corrected by means of the phase detrending function as described in §4.3. Unfortunately, the oscillator on board Voyager 2 was not as stable as that on board Cassini, which means that the first-order phase correction used in `rss.ringoccs` is not sufficient. After each ring undergoes first-order phase correction and downsampling to 5 kHz (necessary for the desired 5 m DLP resolution), an additional phase correction is applied. First the phase is unwrapped to recover the phase drift. Due to both noise and to diffraction patterns present in the signal phase, the unwrapping is not always successful. Corrections to the phase unwrapping are made using parameters stored in an ancillary reference file `phase_unwrap_params.py`. We do not recommend users add to, remove, or otherwise change these parameters, but users may do so at their own risk.

Once the unwrapping corrections are made, we “fit” the unwrapped phase by convolving it with a 1 second boxcar kernel. This time-averaging is comparable to the [Gresh et al. \(1989\)](#) approach of applying a low-pass filter to the unwrapped phase to obtain a “fit”. Unlike the low-pass filter approach, our time-averaging does not fully preserve the diffraction pattern encoded in phase after subtracting the fit from the observed phase. This poses a problem because this diffraction pattern is necessary for reconstructing the original profile. To preserve the diffraction pattern, we fit a second order polynomial to the time-averaged unwrapped phase adjacent to the ring profile (where the diffraction pattern occurs in the phase) and replace the time-average “fit” over the diffraction pattern with the polynomial estimate, effectively assuming the polynomial is characteristic of the phase drift present in the diffraction pattern. Finally, we subtract the total fit from the unwrapped phase to correct for the second-order drift. This produces phase profiles

comparable to those shown in [Gresh et al. \(1989\)](#).

During our preliminary processing of the Voyager 2 data, we found that normalizing the power before resampling to radius (the order in `rss_ringoccs` would sometimes result in an average normalized free space power greater or less than unity. Performing the power normalization after resampling the data from observed event time to ring intercept radius eliminates this effect. After resampling the data to uniform sampling in radius, we mask out each ring region identified by orbital parameters along with an additional 15 km on either side of the profile to exclude the entire diffraction pattern from the fit. Then, we fit the free space power with a ninth order polynomial and divide the total diffraction-limited power by the fit to obtain the diffraction-limited profile.

6.1.4 Changes to DLP

As in `rss_ringoccs`, the DLP step primarily entails interpolating the Geometry and Calibration results to the uniform sampling in ring intercept radius. Threshold optical depth is calculated using noise power estimated from the STFT of the phase-corrected freespace power. Then, the reconstruction window size is estimated for the highest reconstruction resolution (20 m in this case) and Fresnel scale calculated by Geometry. To ensure reconstruction of the total ring profile with several kilometers of baseline freespace on either side of the profile, we pad the ring DLP profile output with additional data on either side of the profile to account for the window size necessary for reconstructing the end points. Limiting output to this small range of profile power dramatically improves processing time while reducing the output file size.

6.1.5 Changes to Reconstruction

The reconstruction routines remain unchanged from the v1.2 `rss_ringoccs` version of `diffrec`. The major differences are in the default inputs for calling the `DiffractionCorrection`. The `res_fac` is set to unity so that the reconstruction and processing resolutions are identical. The Allan deviation is set to the 2×10^{-13} value suggested by [Gresh et al. \(1989\)](#). The default processing window is set to a Kaiser-Bessel window with $\alpha = 2.5$.

6.2 Validation of Uranus processing

To validate our processing results, we compare ring reconstruction at 50 m to the 50 m reconstructions computed and archived by [Gresh et al. \(1989\)](#). While we do not find exact agreement, the profiles are comparable for each ring. Because the reconstructed profiles are comparable even in the freespace regions, we consider this comparison a successful validation of our method for processing the raw Voyager 2 data. Thus, our reconstruction of the raw Voyager 2 data can be considered comparable to that of [Gresh et al. \(1989\)](#) and can be reliably used to produced profiles at higher resolutions.

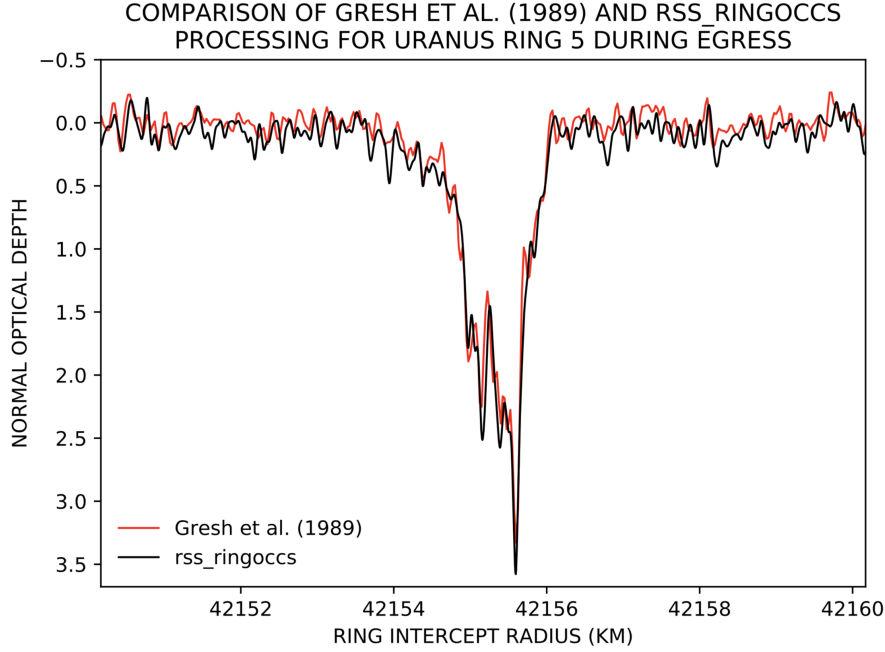


Fig. 13: Comparison of our adapted `rss_ringoccs` processing of the Voyager 2 Uranus ring 5 egress occultation with that of [Gresh et al. \(1989\)](#).

Any disagreement between the [Gresh et al. \(1989\)](#) profiles and ours is small and could be attributed to several systemic differences. To begin, [Gresh et al. \(1989\)](#) may not have recovered the imaginary component of the complex signal in the same manner as our pipeline. Their approach to phase corrections differs from ours and required three separate steps rather than two. It is also not explicitly clear which window the [Gresh et al. \(1989\)](#) processing used for their reconstruction. Furthermore, our pipeline does not account for an elliptical geometry or a cant angle during the reconstruction.

7 Where to go from here

7.1 The Cassini RSS data catalog

We have created a `CSV` file, located within the `tables` directory, with all 1 kHz Cassini ring occultation RSR files available on the PDS. This catalog includes information of each RSR file, such as wavelength/frequency band, observing station information, sampling rate, associated kernels, etc., as well as relevant geometry parameters that can help users determine which file they want to use. The column headers and definitions for this catalog are also available within the `tables/` directory.

7.2 Selecting an RSR file to process

Before using `rss_ringoccs`, we recommend users browse the Cassini RSS data catalog to find an appropriate RSR file to process. Factors such as elevation angle, antenna size, radius range, and ring opening angle can affect this decision.

First, users should establish that the RSR file covers the radius range of interest. Some occultations, such as chords, do not cover the entire ring system (see Figure 22.2). Next,

depending on the research goal, users should select an observation with the desired ring opening angle. A large ring opening angle means that the spacecraft signal has gone through less ring material, and a small ring opening angle means that the signal has gone through more ring material. The latter is useful for tenuous features, such as some C ring density waves, and the former is useful for observing optically thick regions, such as the B ring. Another factor to consider is the Earth receiving station. Different stations will record in different bands (S-, X-, or Ka-band) using different sized antennas (34m or 70m). The location of the DSN is also important, as Cassini could be lower in the sky at one station but higher in the sky at another – this affects the amount of atmosphere the signal must go through to reach the station. These station-related factors can affect the overall SNR as well as the background power drift.

7.3 Choosing a radial resolution

As discussed in Section 3.6, users must provide the shortest resolvable wavelengths to `rss_ringoccs` in the form of radial sample spacing in km for both the DLP and the reconstructed profile. The latter must always be at least 8/3 greater than the former; however, there are additional considerations in choosing a sample spacing.

In Figure 8, Marouf et al. (1986) show a relation between threshold optical depth τ_{TH} (Section 4.4.1) and processing resolution ΔR for X and S band. Because lower resolutions lead to larger SNR values, τ_{TH} increases with ΔR . If the intrinsic SNR is low—more often the case in Ka band or low-elevation observations with large Earth atmosphere extinction—a small radial sampling rate might not produce meaningful results because the profile normal optical depth frequently exceeds the threshold optical depth. In such cases, we recommend users consider the threshold optical depth when selecting the shortest resolvable wavelength.

7.4 Execution time benchmarks

The single-file end-to-end example script discussed in §3.3 includes a benchmark computation time for a typical 1 kHz RSR file for a diametric occultation. When run, `e2e_run.py` will print out the expected processing time for a single 1 kHz RSR file. We have conducted a series of benchmarks for various machines across a range of hardware and operating systems and various resolvable wavelengths. The results of these benchmark runs for a single 1 kHz file are listed below for reference. Note that processing time may vary based on other tasks being run by a machine at the time of running `rss_ringoccs`. For our own benchmarks, we find benchmark times can vary by up to $\approx 5\%$ for a given machine.

$\Delta\rho$ (km)	$4/3\Delta R$ (km)	Time (s)
MacBook Pro, 2.9 GHz Intel Core i7, 16GB RAM		
0.25	1.0	90
0.25	0.667	120
0.05	0.25	270
0.05	0.134	400

Table 9: Benchmarks for processing the Rev 007 E X43 1 kHz RSR file.

In addition to these individual benchmarks, we have run a benchmark of the batch processing script. A complete run of `e2e_batch.py` which processes all 1 kHz RSR files before USO failure requires 5.25 hrs to run on with a 2.9 GHz Intel Core i7 CPU with 16 GB of RAM and ≈ 10 hrs with a 2.7 GHz Intel Core i5 CPU and 8 GB RAM.

7.5 Licensing

`rss_ringoccs` is free/open-source software made available under the GNU General Public License. The following license is included with the top-level `__init__.py` file in the `rss_ringoccs` software package:

```
Copyright (C) 2018 Team Cassini at Wellesley College
```

```
This program is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation,
either version 3 of the License, or (at your option) any
later version.
```

```
This program is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU General Public License for more
details.
```

```
You should have received a copy of the GNU General Public
License along with this program. If not, see
http://www.gnu.org/licenses/.
```

```
This program is part of the rss_ringoccs repository hosted
at https://github.com/NASA-Planetary-Science/rss_ringoccs
and developed with the financial support of NASA's Cassini
Mission to Saturn.
```

7.6 Citing `rss_ringoccs`

If you use `rss_ringoccs` in your research as the basis of a publication, please consider citing the software package using the [DOI:10.5281/zenodo.2548947](https://doi.org/10.5281/zenodo.2548947)

Acknowledgements

Development of `rss_ringoccs` was supported by NASA/JPL as part of the Cassini Mission Closeout effort. Processing of the Voyager 2 Uranus occultations was supported by NASA program NNH14ZDA001N-PDART. The authors especially appreciate the support and encouragement of Linda Spilker and Kathryn Weld. We dedicate this work to the memory of Arv Kliore, the original Cassini RSS Team Leader and an example of wisdom and generosity for us all.

Appendices

A Meta-kernel file

A meta-kernel file contains a list of the complete pathnames to a list of individual kernel files. An example meta-kernel `Rev007.meta_kernel` has been provided in the `./examples/` directory for the Rev007 occultations. To create a meta-kernel for another occultation, follow the structure of the Rev007 meta-kernel example. Typically, the only files one needs to change are those with a `*.bsp` or `*.tpc` extension. All kernel files necessary for building the meta-kernel for a specific observation are listed in final column of the data catalog found in the `./tables/` directory. For example, the Rev014 occultation might have a meta-kernel with the following contents:

```
\begindata
PATH_VALUES = ( './kernels/naif/CASSINI/kernels/spk'
                './kernels/naif/CASSINI/kernels/lsk'
                './kernels/naif/generic_kernels/spk/planets'
                './kernels/naif/generic_kernels/spk/stations'
                './kernels/naif/generic_kernels/pck'
                './kernels/naif/CASSINI/kernels/pck'
                './kernels/local' )

PATH_SYMBOLS = ('A', 'B', 'C', 'D', 'E', 'F', 'G')

KERNELS_TO_LOAD = ( '$A/051011R_SCPSE_05245_05257.bsp'
                    '$B/naif0012.tls'
                    '$C/de430.bsp'
                    '$F/cpck110ct2005.tpc'
                    '$D/earthstns_itr93_050714.bsp'
                    '$F/earth_000101_180919_180629.bpc' )

\beginntext
These are the kernels used for Rev014
```

B Calibration Output Plots

After computing $\hat{f}(t)_{offset}$ from the frequency offset, the `FreqOffsetFit` class plots $f(t)_{offset}$, the $f(t)_{offset}$ values used to compute $\hat{f}(t)_{offset}$, and $\hat{f}(t)_{offset}$, along with labels specifying the polynomial order used in the fit, information identifying the revolution number, direction, wavelength band, DSN station, processing date, and processing serial number. `FreqOffsetFit` saves this plot in a PDF in the directory within `./output/` corresponding to the current Rev, band, and DSN station. This is the same directory as the `.LBL` and `.TAB` files output by `rss_ringoccs` if `write_file` is set to `True` (the recommended value). The output PDF is named following the same conventions as the `.LBL` and `.TAB` files, complete with processing date and serial number. We show an example of one such output plot in Figure 14 for the occultation Rev 007 in the egress direction as observed in the X-band from DSN station 43.

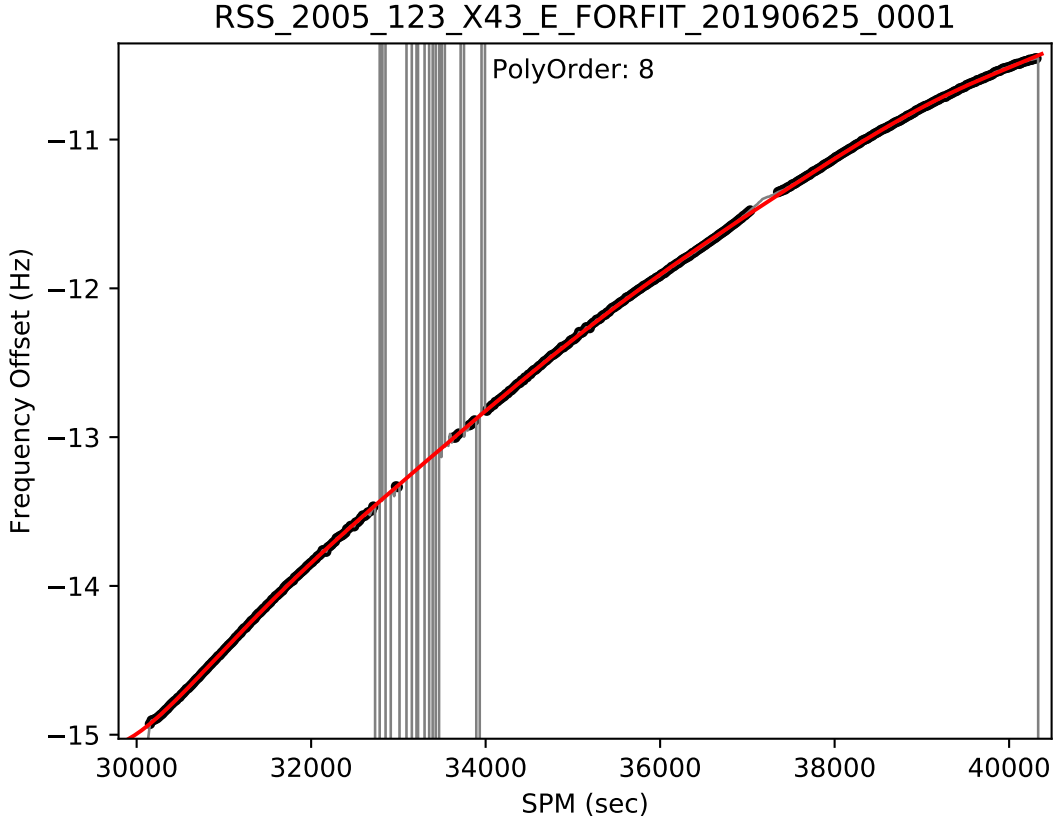


Fig. 14: Example of the frequency offset fit plot output by the `FreqOffsetFit` submodule of `Calibration` step. The total frequency offset $f(t)_{offset}$ is depicted in grey, the $f(t)_{offset}$ used to compute $\hat{f}(t)_{offset}$ in black, and the fit $\hat{f}(t)_{offset}$ in red.

After computing \hat{P}_0 from available freespace regions, the `Normalization` class plots power, the power values used to compute \hat{P}_0 , and \hat{P}_0 , along with labels specifying the fit type, fit order, and information identifying the revolution number, direction, wavelength band, DSN station, processing date, and processing serial number. `Normalization` saves this plot in a PDF in the directory within `./output/` corresponding to the current Rev, band, and DSN station. This is the same directory as the `.LBL` and `.TAB` files output by `rss_ringoccs` if `write_file` is set to `True` (the recommended value). The output PDF is named following the same conventions as the `.LBL` and `.TAB` files, complete with processing date and serial number. We show an example of one such output plot in Figure 15 for the occultation Rev 007 in the egress direction as observed in the X-band from DSN station 43.

C Interactive Mode

When the `interact` keyword argument is set to `True` when instantiating the `Calibration` object class, the free space power fitting method will enter interactive mode to receive user input. After the initial automated fit to the free space regions, `rss_ringoccs` will plot the results of the fit (in the same manner as Figure 15) in a `matplotlib` window and prompt the user for input in the terminal.

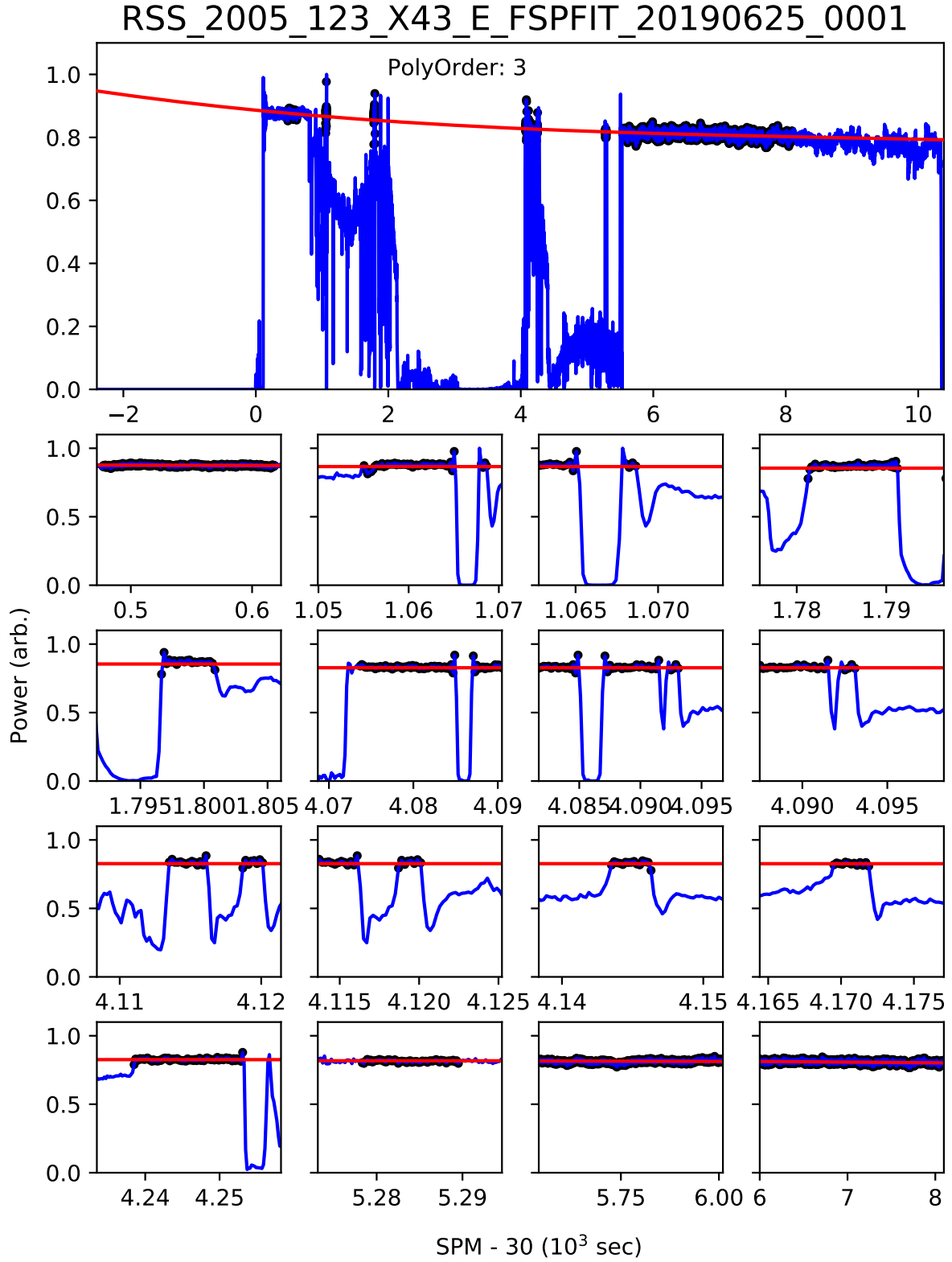


Fig. 15: Example of the free space power fit plot output by the `Normalization` submodule of `Calibration` step. Phase-detrended power is shown in blue, the power used to compute the fit \hat{P}_0 to the free space power in black, and the fit \hat{P}_0 in red. Top panel is the total power profile. Each subpanel is a closeup of one of the free space regions used in the fit. Subpanels shown in order of time observed (SPM). The number of subpanels will vary depending on the number of freespace regions used in the fit.

Here, we explain the input at each prompt and then show an example of a single iteration of the interactive mode with possible responses.

```

Do you want to continue with this fit? (y/n): n

Do you want to change the freespace regions? y

Last used freespace gaps in SPM:
[[30477.0, 30618.833698196322],
 [31054.928062871928, 31065.345792559005],
 [31067.770358800513, 31068.930045219375],
 [31780.866566933848, 31791.436274129595],
 [31796.580066071256, 31801.05616095102],
 [34073.68113956013, 34085.4943434775],
 [34086.68935725464, 34091.74708080809],
 [34092.49215200969, 34093.327481330285],
 [34113.3838723948, 34116.39786534125],
 [34118.619852169526, 34120.21853934005],
 [34143.358694145434, 34146.38873649045],
 [34169.41951728726, 34172.05701601253],
 [34238.49233676398, 34253.26244631217],
 [35278.146362236446, 35289.59914281318],
 [35545.59294280804, 36005.16144320606],
 [36005.16144320606, 38092.48430556758]]

Do you want to revert to the default freespace gaps? n
Please input new freespace gaps in SPM and press enter twice:
[[30477, 30618],[31055, 31065],[34087, 34091],[34114, 34116],
 [35546, 36005],[36006, 38092]]

[[30477, 30618],
 [31055, 31065],
 [34087, 34091],
 [34114, 34116],
 [35546, 36005],
 [36006, 38092]]

What fit type would you like to use? (poly/spline): poly

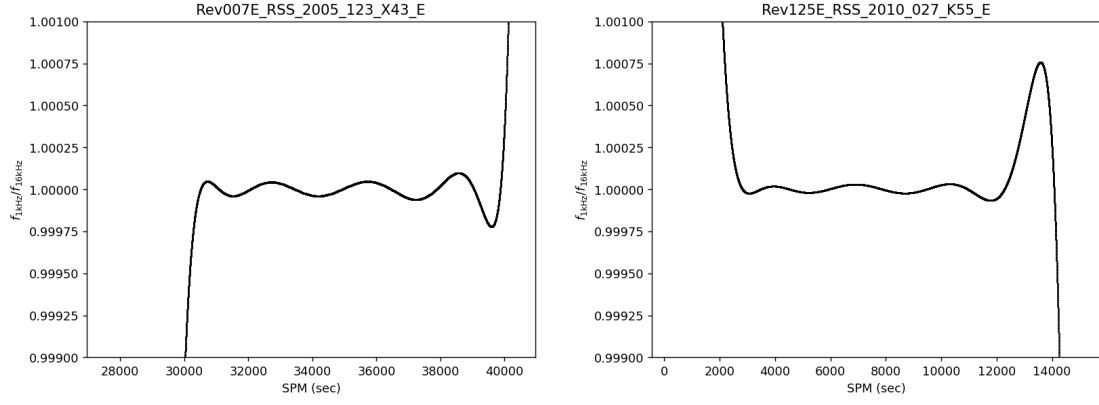
What fit order would you like to use?: 3

Do you want to continue with this fit? (y/n): n

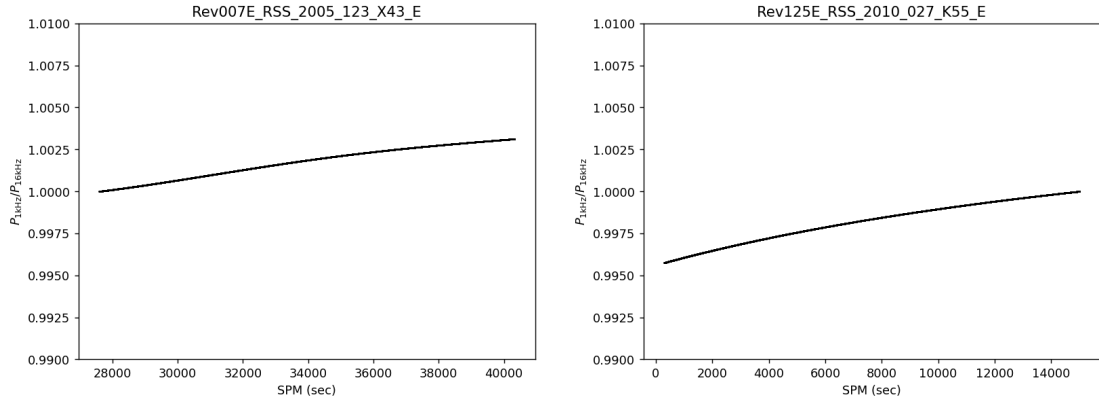
```

1. Entering `y` will end interactive mode and accept the current fit. Type `n` to continue interactive mode and change the fit properties.
2. `n` will skip the prompt for new free space regions. `y` will continue to the prompt for reverting to original free space regions.
3. `y` will revert the free space regions to those predicted by the `Geometry` class and skip the prompt for new free space regions. `n` will continue to a prompt for new, user-defined free space regions.
4. User now enters a $2 \times N$ list of floats specifying the desired free space regions in SPM (units of seconds). N must be greater than or equal to five in order for the free space fitting to properly select the data in the free space regions. For clarity and verification, the entered freespace regions will be printed to the terminal after the user input is parsed.

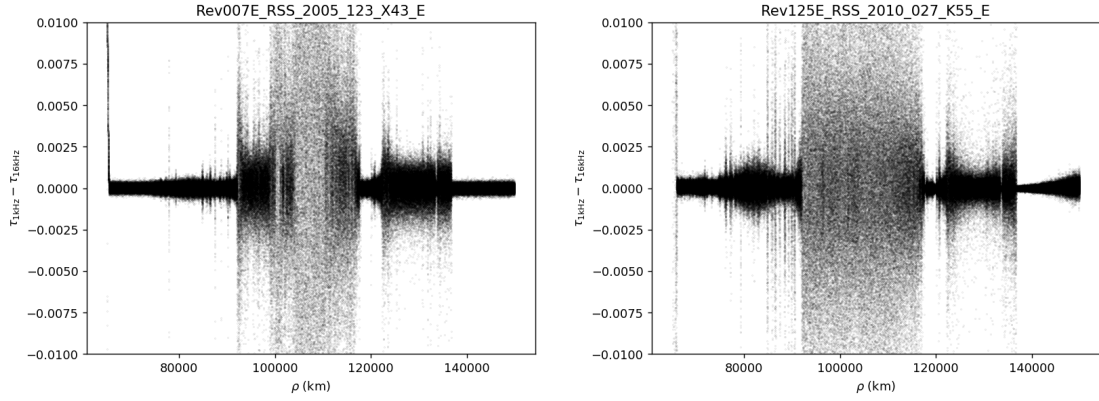
5. Enter the type of fit to use for the freespace power. Two fit types are available, a spline (**spline**) and a polynomial (**poly**); however, we highly recommend the polynomial fit.
6. Enter a whole number between 1 and 9 for polynomial fit or between 1 and 5 for spline fit.
7. A plot of the new fit made based on the user input will appear in a **matplotlib** window. The interactive mode will return to the initial prompt about whether to accept the fit or to continue with interactive mode. Return to step 1.



12.1: Ratio of 1 kHz to 16 kHz Frequency Offset Residual Fits.



12.2: Ration of 1 kHz to 16 kHz Free Space Power Fits.



12.3: Difference between 1 kHz and 16 kHz DLP optical depths.

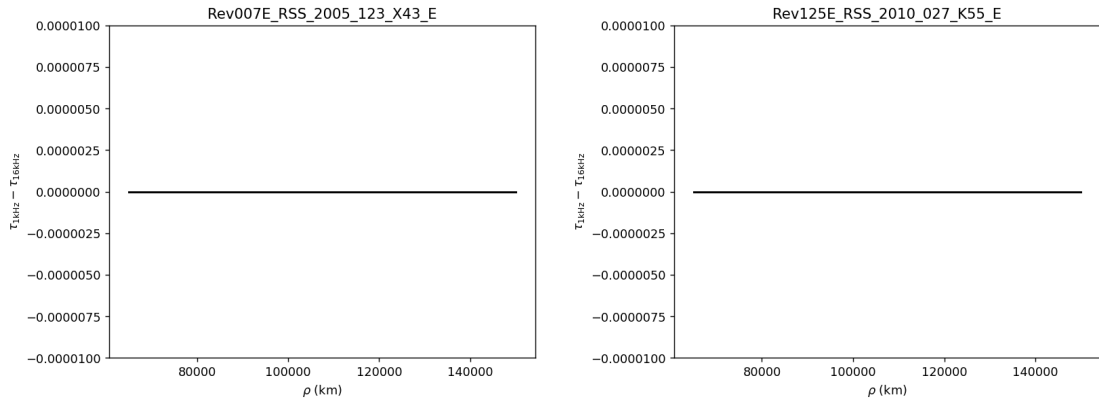


Fig. 12: Validation of 1 kHz processing using raw 16 kHz processing for Rev 007 E X band and Rev 125 E Ka band.

Acronyms

DSN Deep Space Network. [2](#)

HGA High Gain Antenna. [2](#)

NASA National Aeronautic and Space Administration. [1](#)

PDS Planetary Data System. [1](#)

RSR Radio Science Receiver. [3](#), [4](#)

RSS Radio Science Subsystem. [1](#)

Glossary

Atmospheric occultation

Disappearance or reappearance of a source after the signal has passed through the atmosphere of a planet or satellite. [2](#)

Chord occultation

Ring occultation. [3](#)

Deep Space Network

NASA's complex of Earth-based antennas, used to communicate with spacecraft. [2](#)

Diametric occultation

An occultation geometry in which the path of the complete occultation extends from ring ansa to ring ansa, passing behind the planet at mid-occultation. [2](#)

Diffraction Reconstruction

Retrieval of radial optical depth profile of Saturn's rings responsible for observed diffraction pattern. [2](#)

Egress

Exit phase of a ring or atmosphere occultation. [2](#)

Fresnel inversion

Retrieval of intrinsic optical depth profile of the rings by using a Fresnel transform to correct for the effects of diffraction in the observed signal. [2](#)

High Gain Antenna

Highly directional main spacecraft antenna for communications and radio science. [2](#)

Ingress

Entry phase of a ring or atmosphere occultation. [2](#)

NASA

National Aeronautics and Space Administration. [1](#)

Planetary Data System

Long-term archive of digital data products returned from NASA's planetary missions, and from other kinds of flight and ground-based data acquisitions. [1](#)

Radio Science Receiver

An open-loop receiver used in NASA's Deep Space Network (DSN) facilities. [3](#), [4](#)

Radio Science Subsystem

A subsystem placed on board a spacecraft for radio science purposes. [1](#)

Rev number

The number of times the Cassini spacecraft has orbited Saturn. [2](#)

References

- Acton, C. (1996), ‘Ancillary Data Services of NASA’s Navigation and Ancillary Information Facility’, *Planetary and Space Science* **44**, 65–70.
- Asmar, S. W., French, R. G., Marouf, E. A., Schinder, P., Armstrong, J. W., Tortora, P., Iess, L., Anabtawi, A., Kliore, A. J., Parisi, M., Zannoni, M., & Kahan, D. (2018), *Cassini Radio Science User’s Guide*, v1.1 edn, NASA Jet Propulsion Laboratory.
- Colwell, J., Nicholson, P., Tiscareno, M., Murray, C., French, R. & Marouf, E. (2009), ‘The Structure of Saturn’s Rings’, *Saturn from Cassini-Huygens* p. 375.
- French, R., McGhee-French, C., Lonergan, K., Sepersky, T., Jacobson, R., Nicholson, P., Hedman, M., Marouf, E. & Colwell, J. (2017), ‘Noncircular features in Saturn’s rings IV: Absolute radius scale and Saturn’s pole direction’, *Icarus* **290**, 14–45.
- French, R., Nicholson, P., Hedman, M., Hahn, J., McGhee-French, C., Colwell, J., Marouf, E. & Rappaport, N. (2016a), ‘Deciphering the embedded wave in Saturn’s Maxwell ringlet’, *Icarus* **279**, 62–77.
- French, R., Nicholson, P., McGhee-French, C., Lonergan, K., Sepersky, T., Hedman, M., Marouf, E. & Colwell, J. (2016b), ‘Noncircular features in Saturn’s rings III: The Cassini Division’, *Icarus* **274**, 131–162.
- Gresh, D. L., Marouf, E. A., Tyler, G. L., Rosen, P. A. & Simpson, R. A. (1989), ‘Voyager radio occultation by Uranus’ rings I. Observational results’, *Icarus* **78**(1), 131–168.
- Kliore, A., Anderson, J., Armstrong, J., Asmar, S., Hamilton, C., Rappaport, N., Wahlquist, H., Ambrosini, R., Flasar, F., French, R., Iess, L., Marouf, E. & Nagy, A. (2004), ‘Cassini Radio Science’, *Space Science Reviews* **115**, 1–70.
- Marouf, E., French, R., Rappaport, N., Wong, K., McGhee-French, C. & Anabtawi, A. (2011), ‘Uncovering of small-scale quasi-periodic structure in Saturn’s ring C and possible origin’, *EPSC-DPS Joint Meeting* **265**.
- Marouf, E., Tyler, L. & Rosen, P. (1986), ‘Profiling Saturn’s rings by radio occultation’, *Icarus* **68**, 120–166.
- Marouf, E., Tyler, L., Zebker, H., Simpson, R. & Eshleman, V. (1983), ‘Particle Size Distributions in Saturn’s Rings from Voyager 1 Radio Occultation’, *Icarus* **54**, 189–211.
- Moutamid, M. E., Nicholson, P., French, R., Tiscareno, M., Murray, C., Evans, M., French, C., Hedman, M. & Burns, J. (2016), ‘How Janus’ orbital swap affects the edge of Saturn’s A ring?’, *Icarus* **279**, 125–140.
- Nicholson, P., French, R. & Colwell, M. H. E. M. J. (2014a), ‘Noncircular features in Saturn’s rings I: The edge of the B ring’, *Icarus* **227**, 152–175.

- Nicholson, P., French, R., McGhee-French, C., Hedman, M., Marouf, E., Colwell, J., Lonergan, K. & Sepersky, T. (2014b), ‘Noncircular features in Saturn’s rings II: The C ring’, *Icarus* **241**, 373–396.
- Rappaport, N., Longaretti, P., French, R., Marouf, E. & McGhee, C. (2009), ‘A procedure to analyze nonlinear density waves in Saturn’s rings using several occultation profiles’, *Icarus* **199**.
- Thomson, F., Marouf, E., Tyler, G., French, R. & Rappoport, N. (2007), ‘Periodic microstructure in Saturn’s rings A and B’, *Geophysical Research Letters* **34**, L23203.