getting_2015_calibration_AlanNew

April 7, 2022

1 Correspondence of `edges-cal` to `alans-pipeline`

1.1 Introduction

The purpose of this memo is to check how close the `edges-cal` pipeline reproduces `alans-pipeline` for the calibration parameter input done for the H2 case in the nature paper. Getting exact correspondence would be useful for a number of reasons, including the ability to check precisely how much difference certain parameter changes make.

We point towards Memo #198 for the exact known settings and inputs for Alan's pipeline in this case.

It is not trivial to reproduce the exact results. First of all, the code in the pipelines has slight differences throughout, and a different structure. Several "options" have been added to `edges-cal` to get it to perform more like alans-pipeline (note we use the `memo-199` tag for this memo -- functionally equivalent to the `nature-paper-case-h2` tag), but some of these are not on by default, because they are deemed to be less accurate than other methods. Secondly, while we know the exact raw input for the lab spectra, we do *not* know the exact raw input for the S11's (receiver, internal switch, loads, antenna or semi-rigid cable). Instead we have only calibrated S11's that were provided by Raul to Alan. While we would like to ensure that our S11 calibration matches Raul's and go directly from raw files, we cannot do that easily here since the provenance of these calibrated S11's is not available. However, we *have* elsewhere shown that given the exact same input as Raul, we do produce the same output calibration, so we are not too worried about inaccuracies here.

We thus construct our calibration by using the already-calibrated S11's used in Alan's pipeline, and thus our test here is whether, using these inputs, we obtain the same calibration.

Given these inputs, the only potential differences remaining between edges-cal and alans-pipeline must be either in the *modelling* of the S11's, or the iterative fitting procedure for the noise-wave parameters. For the first of these, we are able to do more stringent tests, as we modified `alans-pipeline` to output the exact modeled S11's for all quantities to which we compare in this notebook. While the resulting comparisons do not match to the level of machine noise, we do test whether using exactly Alan's *modeled* S11's directly makes a difference, and show that it does not.

1.2 Imports and Setup

[1]:
```
from edges_cal import CalibrationObservation
import numpy as np
from scipy.interpolate import InterpolatedUnivariateSpline as spline
import matplotlib.pyplot as plt
from edges_analysis.analysis.calibrate import LabCalibration
from edges_analysis.analysis.s11 import AntennaS11
from pathlib import Path
from edges_cal import modelling as mdl
from edges_cal import receiver_calibration_func as rcf
from edges_estimate.likelihoods import LinearFG
from edges_estimate.eor_models import AbsorptionProfile
from edges_estimate.fitting import SemiLinearFit
from edges_cal.tools import FrequencyRange
from yabf.core.samplers import run_map
from edges_cal import s11
import edges_cal as ec
import edges_analysis as ea
import edges_io as eio
from astropy import units as u
from edges_cal import reflection_coefficient as rc
import attr
from edges_cal.cal_coefficients import HotLoadCorrection
from edges_cal.spectra import LoadSpectrum
from numdifftools import Hessian
from datetime import datetime

%matplotlib inline
```

[2]:
```
print("Notebook last run on: ", datetime.now().strftime("%Y-%m-%d"))
```

```
Notebook last run on:  2022-04-07
```

[3]:
```
# Setup plotting for MNRAS
plt.rcParams['axes.facecolor'] = 'white'
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['figure.dpi'] = 600

single_width = 3.377  # inches
double_width = 7.03   # inches
```

[4]:
```
print(f"Using edges-io v{eio.__version__}")
print(f"Using edges-cal v{ec.__version__}")
print(f"Using edges-analysis v{ea.__version__}")
```

```
Using edges-io v4.1.2.dev2+g80276ea
```

```
Using edges-cal v6.1.0.post0.dev7+gdfef8b6
Using edges-analysis v4.1.1
```

1.3 Reading Alan's Data

In this notebook, we compare our calibration observation with data directly provided by Alan. Here we read in those data files.

There are three categories of files that we read in, that enable comparisons at different levels of the calibration:

1. **Calibration Inputs:** Input files to the calibration pipeline:

- `s11_calibration_low_band_LNA25degC_2015-09-16-12-30-29_simulator2_long.txt` (available here), which contains the S11's of all necessary inputs to calibration (receiver, loads, semi-rigid cable),
- the four averaged spectrum files of the loads (i.e. the outputs of running `acqplot7amoon` in Alan's script).

2. **Calibration Output:**:

- `specal_final_case2.txt`, available here, which contains the full onise-wave parameter output, enabling direct comparison to our calibration outputs.
- `all_modeled_s11s.txt` -- output from the updated `edges2k` script, containing all the modeled S11s.

3. **Antenna and Field Data:** we read the Nature Paper H2 field data here to inspect the ramifications of any differences we have in our calibration. This consists of:

- `loss_file.txt`: the antenna loss computed by Alan's pipeline (used to decalibrate the NP data)
- `antenna_s11_file.txt`: the calibrated antenna S11 used by Alan's pipeline (used to decalibrate the NP data)
- `spe0.txt`: the final averaged spectrum from the H2 case, produced by Alan's pipeline.

Calibration Inputs

[5]:
```
alan_s11s = np.genfromtxt("alans-data/
↪s11_calibration_low_band_LNA25degC_2015-09-16-12-30-29_simulator2_long.txt")

alan_s11s = {
    'freq': alan_s11s[:, 0],
    'lna': alan_s11s[:, 1] + 1j*alan_s11s[:, 2],
    'ambient': alan_s11s[:, 3] + 1j*alan_s11s[:, 4],
    'hot_load': alan_s11s[:, 5] + 1j*alan_s11s[:, 6],
    'open': alan_s11s[:, 7] + 1j*alan_s11s[:, 8],
    'short': alan_s11s[:, 9] + 1j*alan_s11s[:, 10],
    'semi_rigid_s11': alan_s11s[:, 11] + 1j*alan_s11s[:, 12],
    'semi_rigid_s12': alan_s11s[:, 13] + 1j*alan_s11s[:, 14],
```

```
    'semi_rigid_s22': alan_s11s[:, 15] + 1j*alan_s11s[:, 16],
    'antsim1': alan_s11s[:, 17] + 1j*alan_s11s[:, 18],
    'antsim2': alan_s11s[:, 19] + 1j*alan_s11s[:, 20],
}
```

[6]:
```
alans_spec = {
    'ambient': np.genfromtxt("/home/smurray/data4/Projects/radio/EOR/Edges/
 ↪alans-pipeline/scripts/EdgesCalComp/spe_loadr.txt", usecols=(0, 1)),
    'hot_load': np.genfromtxt("/home/smurray/data4/Projects/radio/EOR/Edges/
 ↪alans-pipeline/scripts/EdgesCalComp/spe_hotr.txt", usecols=(0, 1)),
    'short': np.genfromtxt("/home/smurray/data4/Projects/radio/EOR/Edges/
 ↪alans-pipeline/scripts/EdgesCalComp/spe_shortr.txt", usecols=(0, 1)),
    'open': np.genfromtxt("/home/smurray/data4/Projects/radio/EOR/Edges/
 ↪alans-pipeline/scripts/EdgesCalComp/spe_openr.txt", usecols=(0, 1)),
}
mask = (alans_spec['ambient'][:, 0] >= 50.0) & (alans_spec['ambient'][:, 0] <=␣
 ↪100.0)
alans_spec = {
    n: v[mask] for n, v in alans_spec.items()
}
```

Calibration Output

[7]:
```
# Note that specal_final_case2.txt is equivalent to specal.txt
alan_cal = np.genfromtxt("alans-data/specal_final_case2.txt")
alan_cal_raw = alan_cal

_f = alan_cal[:, 1]
mask = (_f >= 50.0) & (_f <= 100.0)
_f = _f[mask]
alan_cal = alan_cal[mask]

# To make it easier to compare to our results, we spline it.
alan_cal = {
    'freq': _f,
    'lna_s11': alan_cal[:, 3] + 1j*alan_cal[:, 4],
    'scale': alan_cal[:, 6],
    'offset': alan_cal[:, 8],
    'tunc': alan_cal[:, 10],
    'tcos': alan_cal[:, 12],
    'tsin': alan_cal[:, 14],
    'weight': alan_cal[:, 16],
}
```

[8]:
```
modeled_s11s = np.genfromtxt(
```

```
    '/home/smurray/data4/Projects/radio/EOR/Edges/alans-pipeline/scripts/
↪all_modeled_s11s.txt',
    dtype=[
        ('freq', float),
        ('lna_re', float),
        ('lna_im', float),
        ('ambient_re', float),
        ('ambient_im', float),
        ('hot_load_re', float),
        ('hot_load_im', float),
        ('open_re', float),
        ('open_im', float),
        ('short_re', float),
        ('short_im', float),
        ('sr_s11_re', float),
        ('sr_s11_im', float),
        ('sr_s12_re', float),
        ('sr_s12_im', float),
        ('sr_s22_re', float),
        ('sr_s22_im', float),
    ]
)
modeled_s11s = modeled_s11s[mask]
```

Antenna and Field Data

[9]:
```
alan_sky_data = np.genfromtxt("alans-data/spe0.txt")
flg = alan_sky_data[:, 12]>0
alan_sky_data = {
    'freq': alan_sky_data[:, 1][flg],
    't_ant': alan_sky_data[:, 3][flg],
    't_sky': alan_sky_data[:, 6][flg],
}
```

[10]:
```
loss = np.genfromtxt("alans-data/loss_file.txt")
loss_fq = loss[:, 0]
loss_temp = spline(loss_fq, loss[:, 2])(alan_sky_data['freq'])
bmcorr = spline(loss_fq, loss[:, -1])(alan_sky_data['freq'])
loss = spline(loss_fq, loss[:, 1])(alan_sky_data['freq'])
```

[11]:
```
alan_ant_s11 = np.genfromtxt('alans-data/antenna_s11_file.txt')
alan_ant_s11 = alan_ant_s11[:, 1] + 1j*alan_ant_s11[:, 2]
```

1.4 Definition of Calibration Observation

Here we define our calibration observation, in which we ensure it matches Alan's pipeline as closely as possible. We first define the observation directly based on data on disk in the calibration, knowing that this will not have the correct S11's. However, this gives us a chance to explicitly test how close our S11s are to Alan's. While all of them do not line up with his, we do not expect that this is a problem with our S11 calibration, but rather that we haven't specified the parameters of the S11 calibration (or indeed the raw data) exactly the same. Keeping the comparisons here is just meant to provide a starting point for future investigations.

As we define our observation below, we are careful to note non-default parameters that had to be set specifically to match Alan's pipeline.

[12]:
```
# This defines the directory in which the raw calibration data is stored
# The 'run_num' parameter selects all six runs of the Receiver S11 to be␣
 ↪calibrated
# then averaged. These are almost certainly not the exact raw inputs that were␣
 ↪used
# by Alan.
calio = eio.io.CalibrationObservation(
    "/data5/edges/data/CalibrationObservations/Receiver01/
 ↪Receiver01_25C_2015_09_02_040_to_200MHz/",
    run_num={"receiver_reading": (1,2,4,5,6,8)},
    repeat_num=1,
)

calobs = CalibrationObservation.from_io(
    calio,
    f_low=50.0*u.MHz,     # Sets the final frequency range
    f_high=100.0*u.MHz,   # used in calibration
    cterms=6,
    wterms=5,
    freq_bin_size=8,      # Equivalent to Alan's "smooth" parameter
    spectrum_kwargs= {
        "default": {
            "f_low": 40*u.MHz,             # Frequency range for reading/
 ↪smoothing
            "f_high": 110*u.MHz,           # the spectra, from which a subset␣
 ↪is used for calibration
            "t_load": 300,
            "t_load_ns": 1000,
            "ignore_times_percent": 7200,  # Since it's >100, sets the number␣
 ↪of seconds to ignore
            'frequency_smoothing': 'gauss',# Smooth with gaussian filter like␣
 ↪Alan. Prefer 'bin' usually.
            'time_coordinate_swpos': 0,    # Ignore times based on swpos=0␣
 ↪timings.
```

```
        },
        'ambient': {'temperature': 296.0},  # Set the assumed temperature of␣
↪the loads.
        'hot_load': {"temperature": 399.0}, # edges-cal by default uses the␣
↪thermistor
        'open': {'temperature': 296.0},     # to read the actual temperature,␣
↪instead of
        'short': {'temperature': 296.0}     # assuming it. This makes a␣
↪reasonably large difference.
    },
    s11_kwargs={
        "default": {
            'model_type': mdl.Fourier,       # Use Fourier model for all loads.
            'complex_model_type': mdl.ComplexRealImagModel,  # Fit on real/imag␣
↪instead of abs/phase
            'model_transform': mdl.ZerotooneTransform(range=(50, 100)),  # Alan␣
↪uses (0, 1) range of freq.
            'model_kwargs': {'period': 1.5}, # Alan uses 2pi/1.5 in his cos/sin␣
↪terms
            'n_terms': 27,                   # corresponds to nfit2 in alans␣
↪pipeline
        },
        'hot_load': {'model_delay': 2*np.pi*5e-10*u.s},  # Alan fits for a␣
↪delay. We just use his output value.
        'ambient': {'model_delay': 2*np.pi*6e-10*u.s},   # this value is␣
↪printed out in his updated pipeline.
        'open': {'model_delay': 2*np.pi*6.09e-8*u.s},
        'short': {'model_delay': 2*np.pi*6.09e-8*u.s},
    },
    receiver_kwargs = {
        'n_terms': 11,           # Use 11 terms
        'model_type': 'fourier', # Alan used Fourier series on receiver
        'model_transform': mdl.ZerotooneTransform(range=(50,100)),  # See above␣
↪for what these represent.
        'model_kwargs': {'period': 1.5},
        'complex_model_type': mdl.ComplexRealImagModel,
        'model_delay': -2*np.pi*1e-9*u.s
    },
    hot_load_loss_kwargs = {
        'model': mdl.Fourier(n_terms=27, period=1.5, transform=mdl.
↪ZerotooneTransform(range=(50,100))),
        'complex_model': mdl.ComplexRealImagModel,
    }
)

# The 'labcal' object additionally gets the antenna S11 so it can apply
```

```
# the lab calibration to field data. We are not particularly interested
# in whether we used the exact same raw input here, so we look at the
# calibrated antenna S11 Alan used.
def get_labcal(calobs,raw=False, use_spline=False):
    if use_spline:
        return LabCalibration(
            calobs=calobs,
            antenna_s11_model=AntennaS11(
                freq=calobs.freq, raw_s11=alan_ant_s11, use_spline=True,
                complex_model_type=mdl.ComplexRealImagModel,
                internal_switch=calobs.internal_switch
            )
        )

    if raw:
        s11_files = sorted(Path('/data5/edges/data/S11_antenna/low_band/
↪20151207').glob('run_0001_*20151208_0314*.s1p'))
    else:
        s11_files = '/data4/smurray/code_from_alan/
↪S11_blade_low_band_2015_342_03_14.txt.csv'

    return LabCalibration.from_s11_files(
        calobs=calobs, s11_files=s11_files
    )

labcal = get_labcal(calobs)
f = calobs.freq.freq
```

Now, `calobs` and `labcal` are our best-guess from what we have on disk on `enterprise`, but they do not line up with the S11's that are used in Alan's code, which he received from Raul. We make a new object here that exactly uses Alan's calibrated (but unmodeled) S11s, and then another object that uses his exact modeled S11's.

[13]:
```
hlc_alan = HotLoadCorrection(
    freq=FrequencyRange(alan_s11s['freq']*u.MHz),
    raw_s11=alan_s11s['semi_rigid_s11'],
    raw_s12s21=alan_s11s['semi_rigid_s12'],
    raw_s22=alan_s11s['semi_rigid_s22'],
    model = mdl.Fourier(n_terms=27, period=1.5, transform=mdl.
↪ZerotooneTransform(range=(50,100))),
    complex_model=mdl.ComplexRealImagModel,

)

hlc_alan_spline = HotLoadCorrection(
    freq=FrequencyRange(modeled_s11s['freq']*u.MHz),
    raw_s11=modeled_s11s['sr_s11_re']+ 1j*modeled_s11s['sr_s11_im'],
```

```
    raw_s12s21=modeled_s11s['sr_s12_re']+ 1j*modeled_s11s['sr_s12_im'],
    raw_s22=modeled_s11s['sr_s22_re']+ 1j*modeled_s11s['sr_s22_im'],
    use_spline=True,
    complex_model=mdl.ComplexRealImagModel
)
```

[14]:
```
# Let's fix all the S11's to be like Alans
afreq = FrequencyRange(alan_s11s['freq']*u.MHz)

calobs_alans11 = calobs.clone(
    receiver=attr.evolve(
        calobs.receiver,
        raw_s11=alan_s11s['lna'],
        freq=afreq,
    ),
    loads={
        name: attr.evolve(
            load,
            reflections=attr.evolve(
                load.reflections,
                raw_s11=alan_s11s[name],
                freq=afreq,
            ),
            loss_model=hlc_alan if name=='hot_load' else None
        )
        for name, load in calobs.loads.items()
    }
)

labcal_alans11 = get_labcal(calobs_alans11, use_spline=True)
```

[15]:
```
mfreq=FrequencyRange(modeled_s11s['freq']*u.MHz)
calobs_alans11_spline = calobs_alans11.clone(
    receiver=attr.evolve(
        calobs.receiver,
        freq = mfreq,
        raw_s11=modeled_s11s['lna_re'] + 1j*modeled_s11s['lna_im'],
        use_spline=True
    ),
    loads={
        name: attr.evolve(
            load,
            reflections=attr.evolve(
                load.reflections,
                use_spline=True,
                freq=mfreq,
                raw_s11=modeled_s11s[name+"_re"] + 1j*modeled_s11s[name+"_im"]
```

```
            ),
            loss_model=hlc_alan_spline if name=='hot_load' else None
        )
        for name, load in calobs_alans11.loads.items()
    }
)

labcal_alans11_spline = get_labcal(calobs_alans11_spline, use_spline=True)
```

1.5 Compare S11s

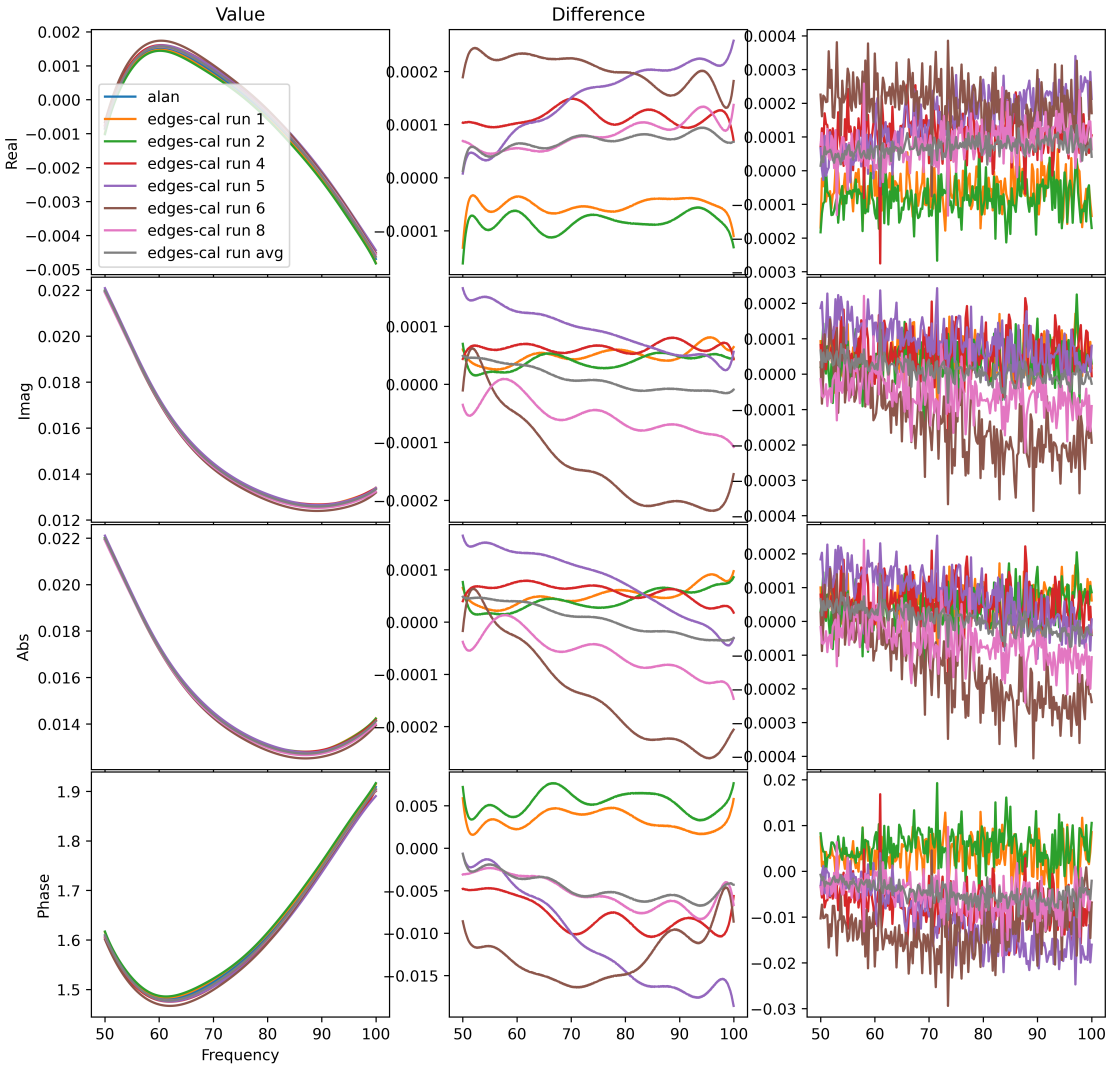
In this section, we go through and compare all the different S11's between the data on `enterprise` and the input to Alan's code. This is just for ease of comparison for later studies that try to get this exactly right.

1.5.1 Receiver S11

Here we plot each individual run number that is available on disk, as well as their average.

[16]:
```
fig, ax = plt.subplots(4, 3, sharex=True, figsize=(12,12),␣
↪gridspec_kw={'hspace': 0.01})

alan_lna = alan_cal['lna_s11']

ax[0,0].plot(f, np.real(alan_lna), label='alan')
ax[1,0].plot(f, np.imag(alan_lna))
ax[2,0].plot(f, np.abs(alan_lna))
ax[3,0].plot(f, np.angle(alan_lna))

ax[1,0].set_ylabel("Imag")
ax[3,0].set_xlabel("Frequency")
ax[0,0].set_ylabel('Real')
ax[2,0].set_ylabel("Abs")
ax[3,0].set_ylabel("Phase")

rcvs = []
for i, run_num in enumerate([1,2,4,5,6,8, 'avg']):
    if run_num != 'avg':
        rcvio = eio.io.ReceiverReading(calio.s11.path  /␣
↪f"ReceiverReading{run_num:02}", repeat_num=1)

        rcv = s11.Receiver.from_io(
            rcvio,
            n_terms=11,
            model_type='polynomial',
```

```
            calkit=rc.get_calkit(calobs.receiver.metadata['calkit'],␣
↪match={"offset_delay": 30*u.ps})
        )
        rcvs.append(rcvio)
    else:
        rcv = s11.Receiver.from_io(
            rcvs,
            n_terms=11,
            model_type='polynomial',
            calkit=rc.get_calkit(calobs.receiver.metadata['calkit'],␣
↪match={"offset_delay": 30*u.ps})
        )
    clb = calobs.clone(receiver=rcv)

    ax[0,0].plot(f, np.real(clb.receiver.s11_model(f)), label='edges-cal run␣
↪%s'%run_num)
    ax[1,0].plot(f, np.imag(clb.receiver.s11_model(f)))
    ax[2,0].plot(f, np.abs(clb.receiver.s11_model(f)))
    ax[3,0].plot(f, np.angle(clb.receiver.s11_model(f)))

    ax[0,1].plot(f, np.real(clb.receiver.s11_model(f)) - np.real(alan_lna),␣
↪color=f'C{i+1}')
    ax[1,1].plot(f, np.imag(clb.receiver.s11_model(f)) - np.imag(alan_lna),␣
↪color=f'C{i+1}')
    ax[2,1].plot(f, np.abs(clb.receiver.s11_model(f)) - np.abs(alan_lna),␣
↪color=f'C{i+1}')
    ax[3,1].plot(f, np.angle(clb.receiver.s11_model(f)) - np.angle(alan_lna),␣
↪color=f'C{i+1}')

    ax[0,2].plot(calobs.receiver.freq.freq.to_value("MHz"), np.real(clb.
↪receiver.raw_s11) - np.real(alan_s11s['lna']), color=f'C{i+1}')
    ax[1,2].plot(calobs.receiver.freq.freq.to_value("MHz"), np.imag(clb.
↪receiver.raw_s11) - np.imag(alan_s11s['lna']), color=f'C{i+1}')
    ax[2,2].plot(calobs.receiver.freq.freq.to_value("MHz"), np.abs(clb.receiver.
↪raw_s11) - np.abs(alan_s11s['lna']), color=f'C{i+1}')
    ax[3,2].plot(calobs.receiver.freq.freq.to_value("MHz"), np.angle(clb.
↪receiver.raw_s11) - np.angle(alan_s11s['lna']), color=f'C{i+1}')

fig.suptitle("LNA S11 Model")
ax[0,0].set_title("Value")
ax[0,1].set_title("Difference")

ax[0, 0].legend();
```

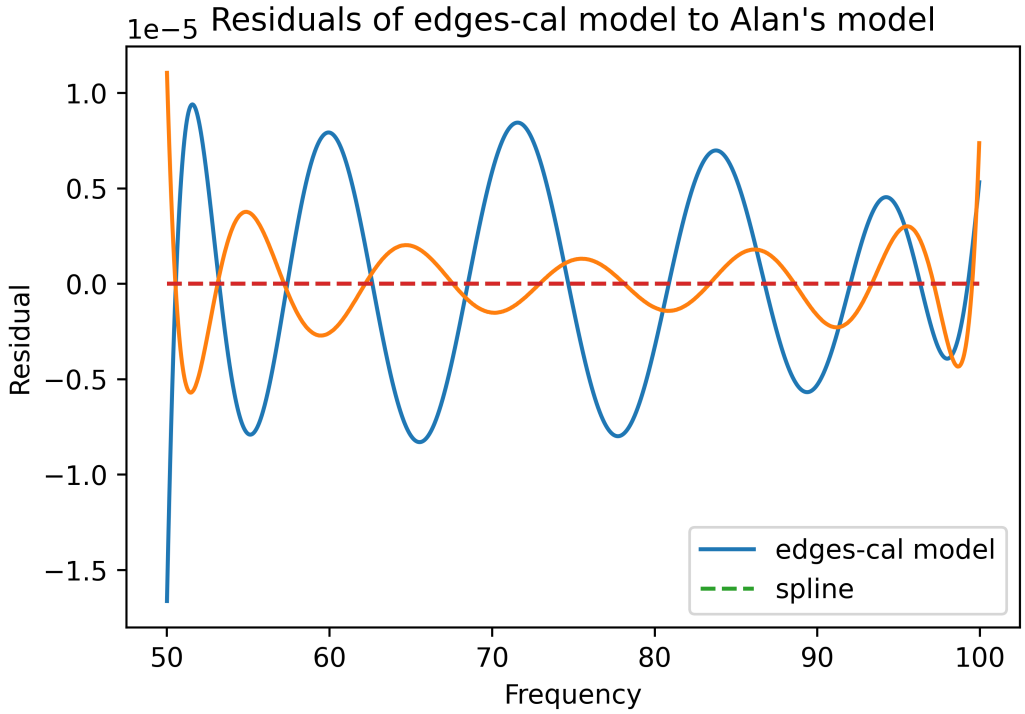


There are definitely differences in the Receiver S11 here, and it is difficult to se any particular average of any subset that would be exact. Nevertheless, the average is pretty close to perfect. Lastly, we just ensure that the calibration object with inject S11 lines up with Alan's:

[17]:
```
diff = np.real(calobs_alans11.receiver.s11_model(calobs_alans11.freq.freq)) -␣
↪modeled_s11s['lna_re']
rms = np.sqrt(np.mean(diff**2))
frac = 100 * rms / np.max(np.abs(calobs_alans11.receiver.
↪s11_model(calobs_alans11.freq.freq)))
```

```
plt.plot(calobs_alans11.freq.freq.to_value("MHz"), diff, label='edges-cal␣
 ↪model')
plt.plot(calobs_alans11.freq.freq.to_value("MHz"), np.imag(calobs_alans11.
 ↪receiver.s11_model(calobs_alans11.freq.freq)) - modeled_s11s['lna_im'])


plt.plot(calobs_alans11.freq.freq.to_value("MHz"), np.
 ↪real(calobs_alans11_spline.receiver.s11_model(calobs_alans11.freq.freq)) -␣
 ↪modeled_s11s['lna_re'], ls='--', label='spline')
plt.plot(calobs_alans11.freq.freq.to_value("MHz"), np.
 ↪imag(calobs_alans11_spline.receiver.s11_model(calobs_alans11.freq.freq)) -␣
 ↪modeled_s11s['lna_im'], ls='--')

plt.title("Residuals of edges-cal model to Alan's model")
plt.legend()
plt.xlabel("Frequency")
plt.ylabel("Residual")
print("% RMS Diff: ", frac)
```

```
% RMS Diff:  0.023983592914129537
```



Here, we see that while we do use the exact calibrated S11, our model differs from Alan's slightly. This could be due to numerical noise adding a little to one of the fitted Fourier terms. However, it is at a very low level -- 10^{-5} which is about 1/10th of the noise.

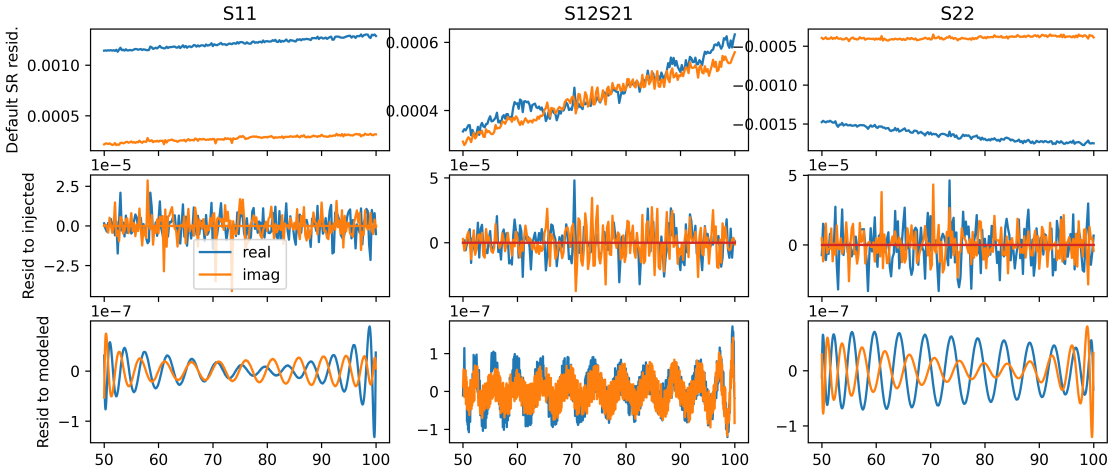
1.5.2 Semi-Rigid Cable

[18]:
```
hlc = HotLoadCorrection.from_file(f_low=50*u.MHz, f_high=100*u.MHz, model=mdl.
↪Fourier(n_terms=27, period=1.5, transform=mdl.
↪ZerotooneTransform(range=(50,100))), complex_model=mdl.ComplexRealImagModel)
```

[19]:
```
fig, ax = plt.subplots(3,3, sharex=True, figsize=(12,5))

ax[0, 0].set_ylabel("Default SR resid.")
ax[1, 0].set_ylabel("Resid to injected")
ax[2, 0].set_ylabel("Resid to modeled")

mfreq=modeled_s11s['freq']
ax[0, 0].set_title("S11")
ax[1, 0].plot(alan_s11s['freq'], np.real(hlc_alan.s11_model(alan_s11s['freq'])␣
↪- alan_s11s['semi_rigid_s11']), color='C0', label='real')
ax[1, 0].plot(alan_s11s['freq'], np.imag(hlc_alan.s11_model(alan_s11s['freq'])␣
↪- alan_s11s['semi_rigid_s11']), color='C1', label='imag')
ax[1, 0].legend()

ax[1, 0].plot(alan_s11s['freq'], np.real(hlc_alan.raw_s11 -␣
↪alan_s11s['semi_rigid_s11']), color='C0')
ax[1, 0].plot(alan_s11s['freq'], np.imag(hlc_alan.raw_s11 -␣
↪alan_s11s['semi_rigid_s11']), color='C1')
ax[0, 0].plot(alan_s11s['freq'], np.real(hlc.s11_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s11']), color='C0')
ax[0, 0].plot(alan_s11s['freq'], np.imag(hlc.s11_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s11']), color='C1')
ax[2, 0].plot(mfreq, np.real(hlc_alan.s11_model(mfreq)) -␣
↪modeled_s11s['sr_s11_re'], color='C0')
ax[2, 0].plot(mfreq, np.imag(hlc_alan.s11_model(mfreq)) -␣
↪modeled_s11s['sr_s11_im'], color='C1')



ax[0, 1].set_title("S12S21")
ax[1, 1].plot(alan_s11s['freq'], np.real(hlc_alan.
↪s12s21_model(alan_s11s['freq']) - alan_s11s['semi_rigid_s12']))
ax[1, 1].plot(alan_s11s['freq'], np.imag(hlc_alan.
↪s12s21_model(alan_s11s['freq']) - alan_s11s['semi_rigid_s12']))
ax[1, 1].plot(alan_s11s['freq'], np.real(hlc_alan.raw_s12s21 -␣
↪alan_s11s['semi_rigid_s12']))
ax[1, 1].plot(alan_s11s['freq'], np.imag(hlc_alan.raw_s12s21 -␣
↪alan_s11s['semi_rigid_s12']))
ax[0, 1].plot(alan_s11s['freq'], np.real(hlc.s12s21_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s12']), color='C0')
```

```
ax[0, 1].plot(alan_s11s['freq'], np.imag(hlc.s12s21_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s12']), color='C1')
ax[2, 1].plot(mfreq, np.real(hlc_alan.s12s21_model(mfreq)) -␣
↪modeled_s11s['sr_s12_re'], color='C0')
ax[2, 1].plot(mfreq, np.imag(hlc_alan.s12s21_model(mfreq)) -␣
↪modeled_s11s['sr_s12_im'], color='C1')

ax[0, 2].set_title("S22")
ax[1, 2].plot(alan_s11s['freq'], np.real(hlc_alan.s22_model(alan_s11s['freq'])␣
↪- alan_s11s['semi_rigid_s22']))
ax[1, 2].plot(alan_s11s['freq'], np.imag(hlc_alan.s22_model(alan_s11s['freq'])␣
↪- alan_s11s['semi_rigid_s22']))
ax[1, 2].plot(alan_s11s['freq'], np.real(hlc_alan.raw_s22 -␣
↪alan_s11s['semi_rigid_s22']))
ax[1, 2].plot(alan_s11s['freq'], np.imag(hlc_alan.raw_s22 -␣
↪alan_s11s['semi_rigid_s22']))
ax[0, 2].plot(alan_s11s['freq'], np.real(hlc.s22_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s22']), color='C0')
ax[0, 2].plot(alan_s11s['freq'], np.imag(hlc.s22_model(alan_s11s['freq']) -␣
↪alan_s11s['semi_rigid_s22']), color='C1')
ax[2, 2].plot(mfreq, np.real(hlc_alan.s22_model(mfreq)) -␣
↪modeled_s11s['sr_s22_re'], color='C0')
ax[2, 2].plot(mfreq, np.imag(hlc_alan.s22_model(mfreq)) -␣
↪modeled_s11s['sr_s22_im'], color='C1')
```

[19]: [<matplotlib.lines.Line2D at 0x7f56683019a0>]



Here we see that the semi-rigid cable measurements on disk are systematically offset from what Alan uses by a number about an order of magnitude more than the noise level (top row). The middle row shows edges-cal model residuals against the raw injected S11 values. Finally, in the

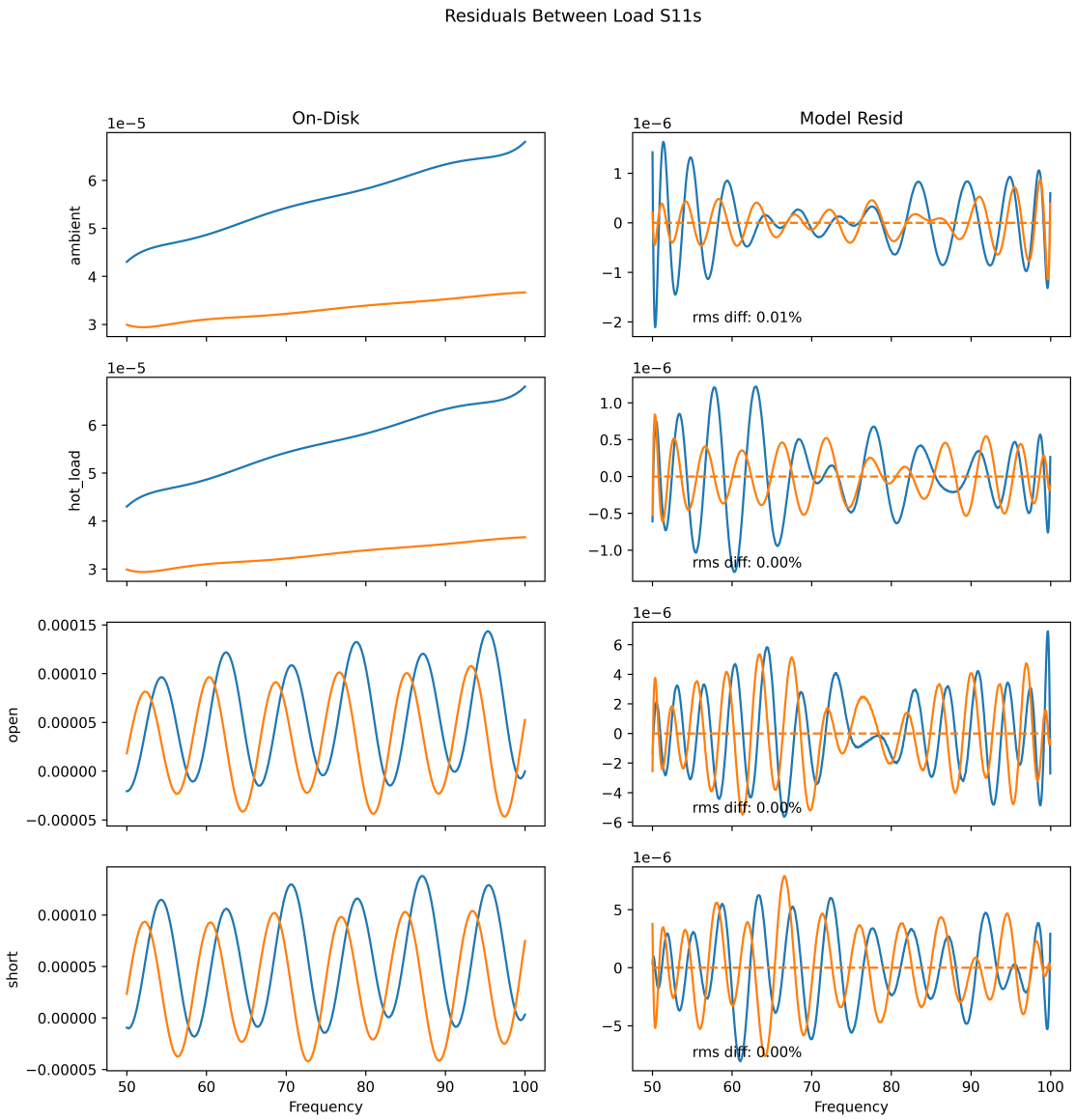
bottom row we show the residuals of `edges-cal` models versus Alan's models. the differences are very small.

1.5.3 Load S11s

[20]:
```
fig, ax = plt.subplots(4, 2, sharex=True, figsize=(12, 12))

for i, (name, load) in enumerate(calobs.loads.items()):
    gload = calobs_alans11.loads[name]
    ax[i, 0].plot(alan_s11s['freq'], np.real(load.reflections.raw_s11 -␣
 ↪alan_s11s[name]), color='C0')
    ax[i, 0].plot(alan_s11s['freq'], np.imag(load.reflections.raw_s11 -␣
 ↪alan_s11s[name]), color='C1')

#     ax[i, 1].plot(alan_s11s['freq'], np.real(gload.reflections.raw_s11 -␣
 ↪alan_s11s[name]), color='C0')
#     ax[i, 1].plot(alan_s11s['freq'], np.imag(gload.reflections.raw_s11 -␣
 ↪alan_s11s[name]), color='C1')

    diff = np.real(gload.reflections.s11_model(modeled_s11s['freq'])) -␣
 ↪modeled_s11s[name+'_re']
    rms = np.sqrt(np.mean(diff**2))
    frac = rms / np.max(np.abs(gload.reflections.
 ↪s11_model(modeled_s11s['freq'])))

    ax[i, 1].plot(modeled_s11s['freq'], diff, color='C0')
    ax[i, 1].plot(modeled_s11s['freq'], np.imag(gload.reflections.
 ↪s11_model(modeled_s11s['freq'])) - modeled_s11s[name+'_im'], color='C1')

    sload = calobs_alans11_spline.loads[name]
    ax[i, 1].plot(modeled_s11s['freq'], np.real(sload.reflections.
 ↪s11_model(modeled_s11s['freq'])) - modeled_s11s[name+'_re'], ls='--',␣
 ↪color='C0')
    ax[i, 1].plot(modeled_s11s['freq'], np.imag(sload.reflections.
 ↪s11_model(modeled_s11s['freq'])) - modeled_s11s[name+'_im'], ls='--',␣
 ↪color='C1')

    ax[i, 1].text(55, 0.95* diff.min(), f"rms diff: {frac*100:.2f}%")
    ax[i, 0].set_ylabel(name)

ax[0, 0].set_title("On-Disk")
ax[0, 1].set_title("Model Resid")

ax[-1, 0].set_xlabel("Frequency")
ax[-1, 1].set_xlabel("Frequency")
```

```
fig.suptitle("Residuals Between Load S11s")
```

[20]: `Text(0.5, 0.98, 'Residuals Between Load S11s')`



So, the load S11s are off by something like 10^{-5} (left column). In the right column, we show that `edges-cal` modelling starting from the same input calibrated S11 as Alan gets differences of 10^-6 compared to Alan's model. The dashed line is a check on whether using a splined model started directly from Alan's modeled S11s gives zero residual (which it does).

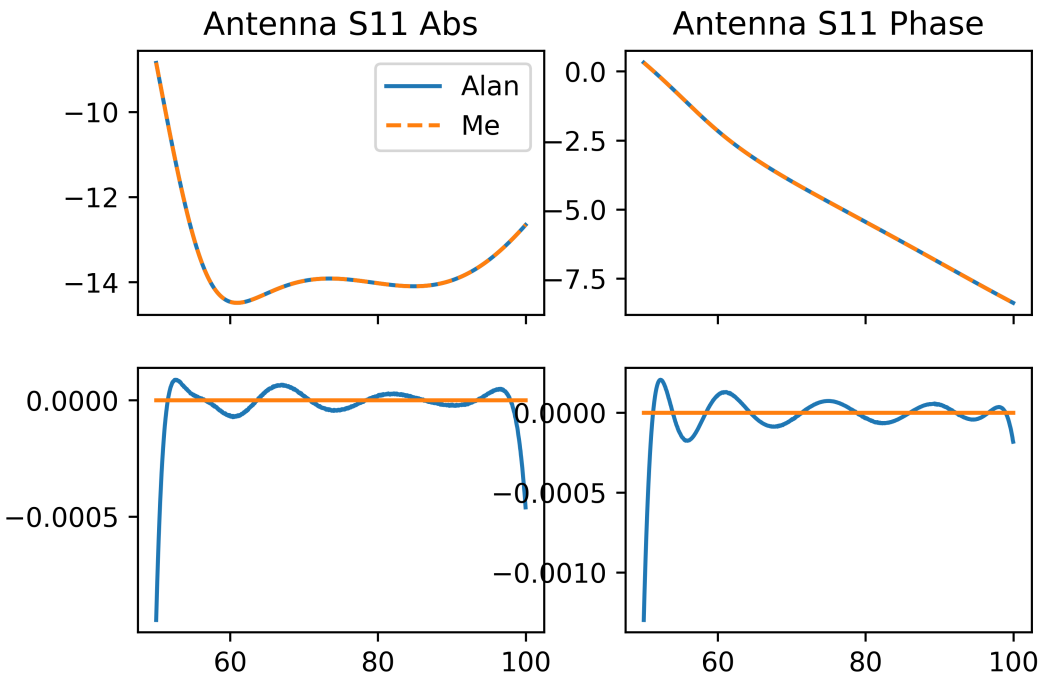
1.5.4 Antenna S11

We also compare the Antenna S11 here, not because it affects the receiver calibration, but because need it to check the effect of the receiver calibration on the final sky data.

[21]:
```
alan_ant_s11_rl = spline(calobs.freq.freq, np.real(alan_ant_s11))
alan_ant_s11_im = spline(calobs.freq.freq, np.imag(alan_ant_s11))

def alan_ant_s11_func(freq):
    return alan_ant_s11_rl(freq) + 1j*alan_ant_s11_im(freq)
```

[22]:
```
fig, ax = plt.subplots(2, 2, sharex=True)

ax[0, 0].plot(f, 20*np.log10(np.abs(alan_ant_s11)),  label='Alan')
ax[0, 1].plot(f, np.unwrap(np.angle(alan_ant_s11)), )

ax[1, 0].plot(f, np.abs(alan_ant_s11)/np.abs(labcal.antenna_s11_model(f)) - 1)
ax[1, 0].plot(f, np.abs(alan_ant_s11)/np.abs(labcal_alans11.
 ↪antenna_s11_model(f)) - 1)

ax[0, 0].plot(f, 20*np.log10(np.abs(labcal.antenna_s11_model(f))), ls='--', ␣
 ↪label='Me')
ax[0, 1].plot(f, np.unwrap(np.angle(labcal.antenna_s11_model(f))), ls='--', )
ax[1, 1].plot(f, np.unwrap(np.angle(alan_ant_s11)) - np.unwrap(np.angle(labcal.
 ↪antenna_s11_model(f))))

ax[1, 1].plot(f, np.unwrap(np.angle(alan_ant_s11)) - np.unwrap(np.
 ↪angle(labcal_alans11.antenna_s11_model(f))))


ax[0, 0].legend()
ax[0, 0].set_title("Antenna S11 Abs")
ax[0, 1].set_title("Antenna S11 Phase")
```

[22]:
```
Text(0.5, 1.0, 'Antenna S11 Phase')
```

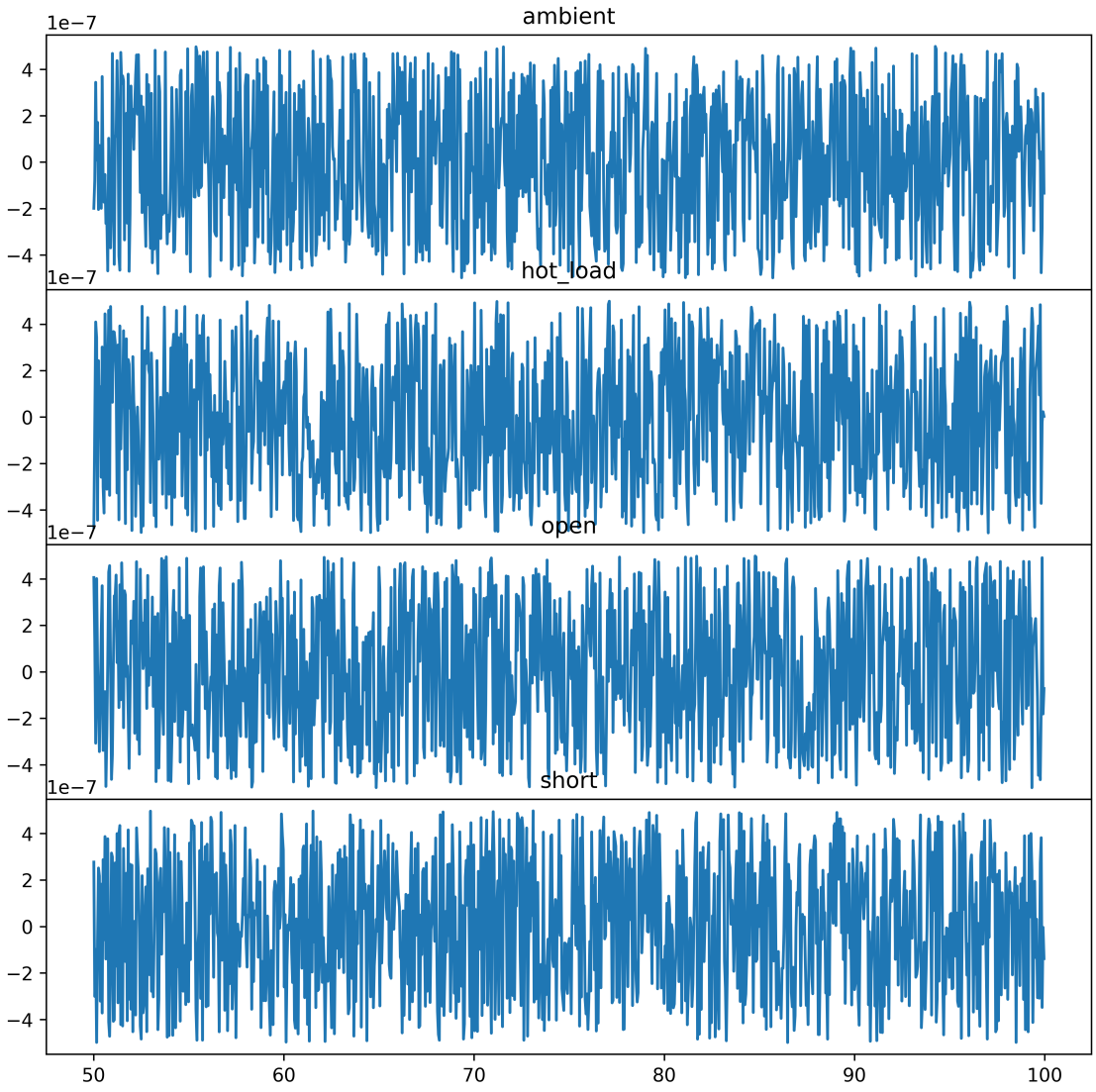


We don't care too much about antenna S11 here (that's a job for another time), but we see that using a spline (orange in bottom plot) gets it exactly right with Alan's modeled S11, so it's not a problem there.

1.6 Compare Averaged Spectra

We now come to comparing spectra, which is something we *should* be able to get very close to exactly right, since we know Alan's raw input.

[23]:
```
fig, ax = plt.subplots(4, 1, figsize=(10, 10), sharex=True,␣
↪gridspec_kw={'hspace': 0})

for i, (name, load) in enumerate(calobs.loads.items()):
    ax[i].plot(alans_spec[name][:, 0], alans_spec[name][:, 1] - load.spectrum.
↪averaged_spectrum)
    ax[i].set_title(name)
```



Here we have float-level precision, as we should expect.

1.7 Compare Calibration Solutions on Sky Data

Here, we use our observations to find the calibration solutions, and then compare those solutions as well as their effect on the calibrated sky data. To do the latter, we decalibrate the sky using Alan's exact calibration solutions, then re-calibrate with a given solution.

[24]:
```
labcal_alans11_spline.antenna_s11_model(calobs.freq.freq) - alan_ant_s11
```

[24]:
```
array([ 5.55111512e-17+1.38777878e-17j,  0.00000000e+00-1.38777878e-17j,
        0.00000000e+00+0.00000000e+00j, …,
       -2.77555756e-17-5.55111512e-17j, -1.38777878e-17-2.77555756e-17j,
```

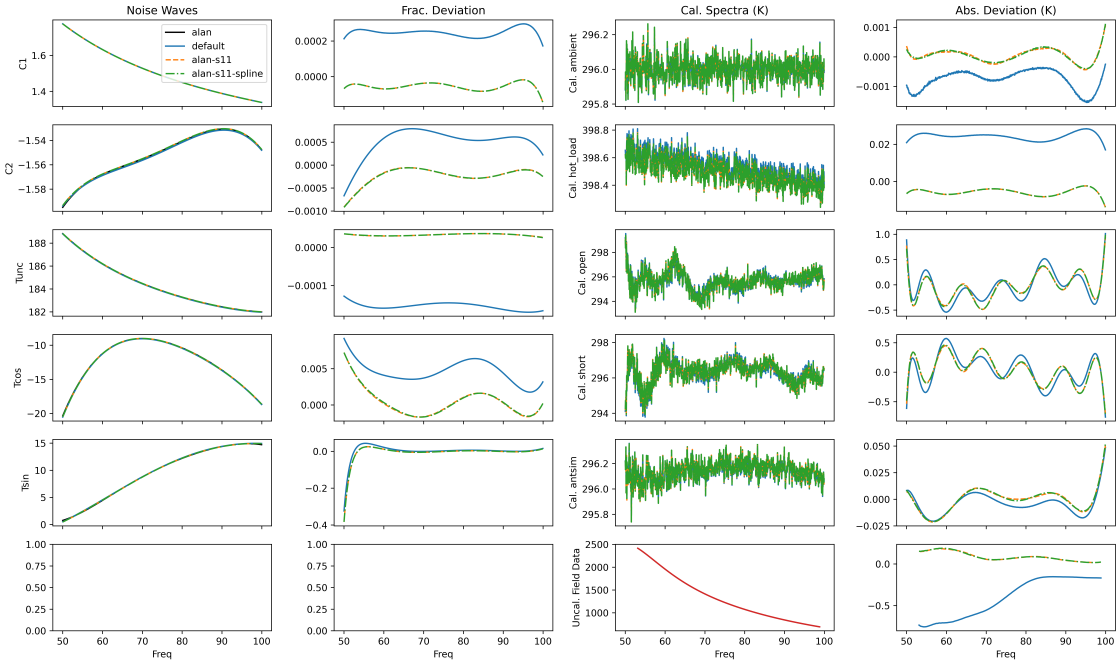
```
        0.00000000e+00+0.00000000e+00j])
```

[25]:
```
def decalibrate(t_sky, f_sky, labcal=None):
    if labcal is None:
        ra, rb = rcf.get_linear_coefficients(
            gamma_ant = alan_ant_s11,
            gamma_rec = alan_cal['lna_s11'],
            sca = alan_cal['scale'],
            off = alan_cal['offset'],
            t_unc = alan_cal['tunc'],
            t_cos = alan_cal['tcos'],
            t_sin = alan_cal['tsin'],
            t_load=300
        )
        tload = 300
        tloadns = 1000
    else:
        ra, rb = labcal.get_linear_coefficients()
        tload = labcal.calobs.t_load
        tloadns = labcal.calobs.t_load_ns

    ra = spline(f, ra)(f_sky)
    rb = spline(f, rb)(f_sky)

    return ((loss/bmcorr)*t_sky + loss_temp - rb - ra*tload)/(tloadns*ra), ra,␣
 ↪rb

def recalibrate(labcal, t_sky, f_sky, with_same=False):
    q, ra, rb = decalibrate(t_sky, f_sky, labcal=labcal if with_same else None)

    if with_same:
        a, b = ra, rb
    else:
        a, b = labcal.get_linear_coefficients()
        a = spline(f, a)(f_sky)
        b = spline(f, b)(f_sky)
    new_t = (labcal.calobs.t_load_ns*a*q + a*labcal.calobs.t_load + b -␣
 ↪loss_temp)*bmcorr/loss
    return new_t

antsim = calobs.new_load('AntSim1', calio,  reflection_kwargs={"n_terms": 105})

def plot_coefficient_comparison(labcals, with_total_effects=True, t_sky=None,␣
 ↪f_sky=None):

    fig, ax = plt.subplots(6 if with_total_effects else 5, 4 if␣
 ↪with_total_effects else 2, sharex=True, figsize=(single_width*5, 10))
```

```
    ax[0, 0].plot(f, alan_cal['scale'], label='alan', color='k')
    ax[1, 0].plot(f, alan_cal['offset'], color='k')
    ax[2, 0].plot(f, alan_cal['tunc'], color='k')
    ax[3, 0].plot(f, alan_cal['tcos'], color='k')
    ax[4, 0].plot(f, alan_cal['tsin'], color='k')

    for j, (name, labcal) in enumerate(labcals.items()):
        clb=labcal.calobs
        ls=['-','--','-.',':'][j%4]

        ax[0, 0].plot(calobs.freq.freq, clb.C1(), label=name, ls=ls)
        ax[0, 1].plot(calobs.freq.freq, clb.C1()/alan_cal['scale'] -1, ls=ls)
        ax[0, 0].set_ylabel("C1")

        ax[1, 0].plot(calobs.freq.freq, clb.C2(), ls=ls)
        ax[1, 1].plot(calobs.freq.freq, clb.C2()/alan_cal['offset'] -1, ls=ls)
        ax[1, 0].set_ylabel("C2")

        ax[2, 0].plot(calobs.freq.freq, clb.Tunc(), ls=ls)
        ax[2, 1].plot(calobs.freq.freq, clb.Tunc()/alan_cal['tunc'] -1, ls=ls)
        ax[2, 0].set_ylabel("Tunc")

        ax[3, 0].plot(calobs.freq.freq, clb.Tcos(), ls=ls)
        ax[3, 1].plot(calobs.freq.freq, clb.Tcos()/alan_cal['tcos'] -1, ls=ls)
        ax[3, 0].set_ylabel("Tcos")

        ax[4, 0].plot(calobs.freq.freq, clb.Tsin(), ls=ls)
        ax[4, 1].plot(calobs.freq.freq, clb.Tsin()/alan_cal['tsin'] -1, ls=ls)
        ax[4, 0].set_ylabel("Tsin")

        if with_total_effects:
            for i, (src_name, source) in enumerate(list(calobs.loads.items()) +␣
↪[('antsim', antsim)]):
                a, b = clb._linear_coefficients(clb.freq.freq, source.
↪s11_model(clb.freq.freq.to_value("MHz")))

                ra, rb = rcf.get_linear_coefficients(
                    gamma_ant = source.s11_model(calobs.freq.freq),
                    gamma_rec = alan_cal['lna_s11'],
                    sca = alan_cal['scale'],
                    off = alan_cal['offset'],
                    t_unc = alan_cal['tunc'],
                    t_cos = alan_cal['tcos'],
                    t_sin = alan_cal['tsin'],
                    t_load=300
                )
```

```
                ax[i, 2].plot(calobs.freq.freq, source.spectrum.
↪averaged_spectrum*a + b, ls=ls)
                ax[i, 3].plot(calobs.freq.freq, source.spectrum.
↪averaged_spectrum*(a-ra) + (b-rb), ls=ls)
                ax[i, 2].set_ylabel(f"Cal. {src_name}")

            ra, rb = rcf.get_linear_coefficients(
                gamma_ant = alan_ant_s11,
                gamma_rec = alan_cal['lna_s11'],
                sca = alan_cal['scale'],
                off = alan_cal['offset'],
                t_unc = alan_cal['tunc'],
                t_cos = alan_cal['tcos'],
                t_sin = alan_cal['tsin'],
                t_load=300
            )
            ra = spline(calobs.freq.freq, ra)(f_sky)
            rb = spline(calobs.freq.freq, rb)(f_sky)

            a, b = labcal.get_linear_coefficients(freq=calobs.freq.freq)

            a = spline(calobs.freq.freq, a)(f_sky)
            b = spline(calobs.freq.freq, b)(f_sky)


            ax[5, 2].plot(f_sky, (t_sky-b)/a, ls=ls)
            ax[5, 3].plot(f_sky, t_sky- recalibrate(labcal,t_sky, f_sky), ls=ls)
            ax[5, 2].set_ylabel("Uncal. Field Data")



    ax[0,0].set_title("Noise Waves")
    ax[0,1].set_title("Frac. Deviation")
    ax[0,2].set_title("Cal. Spectra (K)")
    ax[0,3].set_title("Abs. Deviation (K)")



    if with_total_effects:
        ax[5, 2].plot(f_sky, (t_sky-rb)/ra)

    for i in range(4):
        ax[-1, i].set_xlabel("Freq")
    ax[0, 0].legend()
    plt.tight_layout()
```

[26]:
```
plot_coefficient_comparison(
    {
        "default": labcal,
        'alan-s11': labcal_alans11,
        'alan-s11-spline': labcal_alans11_spline
    },
    t_sky=alan_sky_data['t_ant'],
    f_sky=alan_sky_data['freq']
)
```



We find that the calibration parameters (second column) are very close to those obtained by Alan. The worst is $T_{\sin}$, which has a 20% deviation at the lowest frequencies, even when the S11 is *exactly* the same as Alan's. But most parameters are good to well less than 1%.

Note: getting the S11 modeling correct beyond the current precision seems to do essentially nothing to the results. The only remaining source of difference must be in the actual iterative fit for the NW parameters.

This ends up inducing <1K differences in the final calibrated field spectra, which also exhibit small wiggles over frequency (bottom right). While this is fairly good agreement, it is unclear if this directly affects parameter estimation much, so we go on to figure that out.

1.8 Summary Differences in Field Data

Let's try to summarize the gross differences induced by our calibration observation.

Our first idea is to inject Alan's exact calibration parameters one at a time, to see if any are dominant in their overall effects:

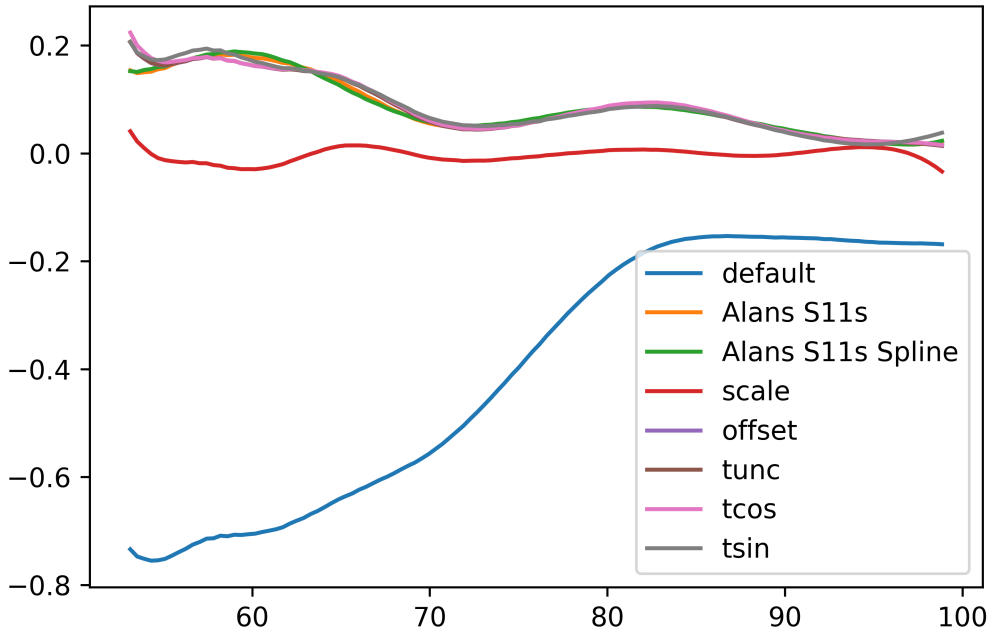
1.8.1 Differences in Calibrated Sky Temperature

[27]:
```
recal_tsky_default = recalibrate(labcal, alan_sky_data['t_ant'],␣
↪alan_sky_data['freq'])
recal_tsky_alans11 = recalibrate(labcal_alans11, alan_sky_data['t_ant'],␣
↪alan_sky_data['freq'])
recal_tsky_alans11_spline = recalibrate(labcal_alans11_spline,␣
↪alan_sky_data['t_ant'], alan_sky_data['freq'])
```

[28]:
```
recal_tsky_injcalpar = {}
fnorm = np.linspace(-1, 1, len(alan_cal['freq']))

for names in [('C1', 'scale'), ('C2', 'offset'), ('Tunc', 'tunc'), ('Tcos',␣
↪'tcos'), ('Tsin', 'tsin')]:
        inj = {names[0]: spline(fnorm, alan_cal[names[1]])}
        _c = calobs_alans11.to_calibrator().clone(
            **inj
        )
        _l = get_labcal(_c)
        recal_tsky_injcalpar[names[1]] = recalibrate(_l,␣
↪alan_sky_data['t_ant'], alan_sky_data['freq'])
```

[29]:
```
plt.plot(alan_sky_data['freq'], alan_sky_data['t_ant'] - recal_tsky_default,␣
↪label='default')
plt.plot(alan_sky_data['freq'], alan_sky_data['t_ant'] - recal_tsky_alans11,␣
↪label='Alans S11s')
plt.plot(alan_sky_data['freq'], alan_sky_data['t_ant'] -␣
↪recal_tsky_alans11_spline, label='Alans S11s Spline')

for k, v in recal_tsky_injcalpar.items():
    plt.plot(alan_sky_data['freq'], alan_sky_data['t_ant'] - v, label=k)

plt.legend()
```

[29]:
```
<matplotlib.legend.Legend at 0x7f55c6f26ac0>
```



Here, it becomes obvious that the C_1 parameter is the key difference, and so that will provide a starting point for getting closer results.

1.8.2 Differences in Foreground/Feature Residuals

Even though there are differences, do they really affect final estimates of the absorption feature? Maybe they would be picked up by the LinLog?

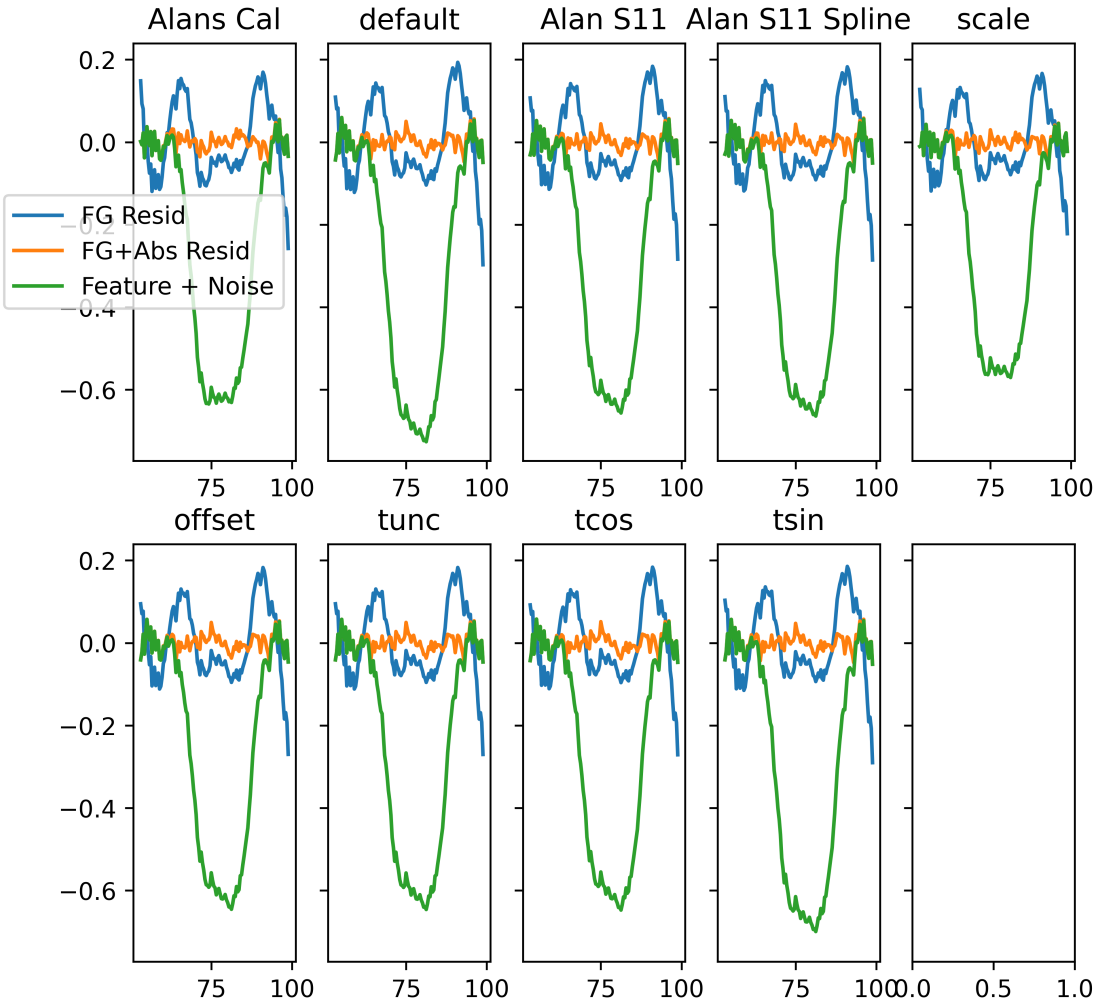
[30]:
```
def make_absorption(freq, fix=tuple()):
    params={
        'A': {'max':2, 'min': 0.05, 'fiducial': 0.5},
        'nu0': {'min': 60, 'max': 90, 'fiducial': 78.5},
        'tau': {'min': 1, 'max': 20, 'fiducial': 7},
        'w': {'min':1, 'max': 25, 'fiducial': 15}
    }

    fid = {}
    for p in fix:
        fid[p] = params.pop(p)['fiducial']

    return AbsorptionProfile(
        name='absorption',
        fiducial=fid,
        params=params,
        freqs=freq
```

```
    )
```

[31]:
```
linlog5 = mdl.LinLog(n_terms=5)
```

[32]:
```
ff = alan_sky_data['freq']

all_recals = {
    'Alans Cal': alan_sky_data['t_ant'],
    'default': recal_tsky_default,
    'Alan S11': recal_tsky_alans11,
    'Alan S11 Spline': recal_tsky_alans11_spline,
}
all_recals = {
    **all_recals,
    **recal_tsky_injcalpar
}


fig, ax = plt.subplots(2, len(all_recals)//2+1, sharey=True,␣
↪figsize=(double_width, double_width))

for i, (title, spec) in enumerate(all_recals.items()):
    slf = SemiLinearFit(
        fg=linlog5.at(x=alan_sky_data['freq']),
        eor=make_absorption(alan_sky_data['freq']), # fix=('tau',)
        spectrum=spec,
        sigma=0.03
    )
    res = slf()

    if i==0:
        res_fid = res

    ax.flatten()[i].plot(ff, slf.get_resid([0, 75, 20, 7]), label="FG Resid")
    ax.flatten()[i].plot(ff, slf.get_resid(res.x), label="FG+Abs Resid")
    ax.flatten()[i].plot(ff, slf.get_eor(res.x) + slf.get_resid(res.x),␣
↪label="Feature + Noise")

    ax[0,0].legend()

    ax.flatten()[i].set_title(title)

    if i==0:
        hf = Hessian(slf.neg_lk)
        print(f"{'STDEV':>15}", np.sqrt(np.diag(np.linalg.inv(hf(res.x)))))
    print(f"{title:>15}", res.x - res_fid.x)
```

```
          STDEV [0.04004209 0.13243109 0.83729586 0.25429138]
      Alans Cal [0. 0. 0. 0.]
        default [ 0.07687649  0.18845395 -1.25362144 -0.20392754]
       Alan S11 [ 0.00694519  0.2736431  -0.49456197 -0.3229493 ]
Alan S11 Spline [ 0.0163974   0.27383069 -0.54122798 -0.29509465]
          scale [-0.06663286 -0.1378361   0.17593066 -0.21900177]
         offset [-0.00822119  0.21858352 -0.58300245 -0.23244332]
           tunc [-0.00770781  0.21955915 -0.58830455 -0.23179744]
           tcos [-0.00836911  0.23885539 -0.53713487 -0.18805814]
           tsin [ 0.04912663  0.23109682 -0.87174593  0.08419442]
```



Here, we have some interesting results. It seems that

1. Getting the S11's right is very important.
2. Effects are very much entangled, so setting S11s AND fixing the scale parameter, we get *worse* agreement.
3. Differences between calibrations are roughly of order the expected standard deviation from the

hessian, except for the depth of the feature, which changes significantly less than a standard deviation for most of the calibrations.