LOWFSSim

Brandon D. Dube
August 31, 2021
public version
Jet Propulsion Laboratory,
California Institute of Technology

1 Introduction

LOWFSSim is a software tool for simulating the Low Order Wavefront Sensing and Control (LOWFSC) system on CGI. It uses python 3.6 or newer. The model is faster than the real LOWFS system that will fly aboard CGI, and has been validated against CGI's performance testbed to better than 0.08%.

The library contains models for each of CGI's three configurations HLC/NFOV, SPC-SPEC, and SPC-WFOV. They are available to the user under a common interface. The model is sufficiently fast that the integrated model including multi-plane diffraction, radiometry, EMCCD noises, and wavefront sensing can be executed in less than 0.5 milli-seconds. At such speed, there is nearly no modeling question too large to be asked and answered.

2 Installation

Python 3.6 or newer is required. You should obtain a copy of the software along with this manual from github.com/nasa-jpl/lowfssim.

The conventional way to install the package is by invoking `pip install .` from the lowfssim directory, which contains the setup.py file. pip will take care of the dependencies (numpy, scipy, astropy, prysm) for you.

You may also "install" it by invoking:

```
import sys
sys.path.append(r"/path/to/lowfssim")
```

though this is not recommended usage for any python code.

The repository contains both the code and the necessary data files. git-lfs may be needed to download the data files.

3 Usage

Forward modeling and WFS are separate activities in lowfssim. We will cover the former first, as it is a pre-requesite for the latter.

3.1 Forward Modeling

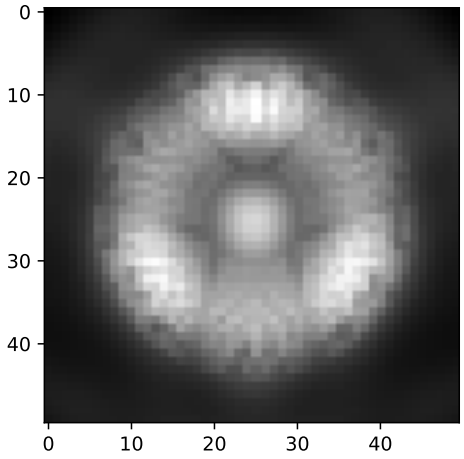
Before the model can be run, the software must be imported and dependencies must be loaded into memory:

```
from lowfsc.data import DesignData
import lowfsc.props as prop

from pathlib import Path

from matplotlib import pyplot as plt

data_fldr = Path('~/proj/wfirst/lowfs/data2').expanduser()
dd = DesignData.hlc_design(data_fldr)
dd.seed_zernikes(range(2,12))
```

The following snippets will assume these imports were made and additional imports will be added over the course of these snippets. The penultimate line loads the HLC design model with the Roman pupil, DM wavefront error solutions, and focal plane mask prescription. The text `hlc` can be replaced by `spec` or `wfov`. The final line pre-computes the Zernike polynomials used to create disturbances.

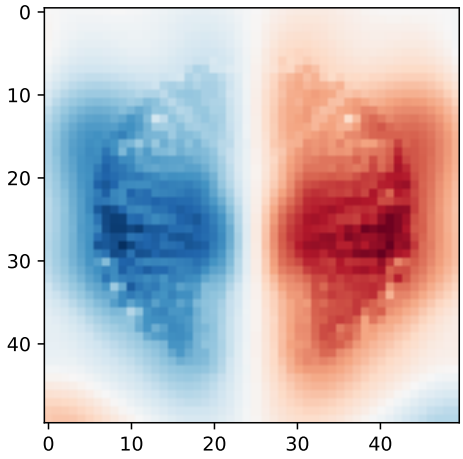
The most basic usage is to produce an image:

```
# Zernike coefficients for the Noll terms in seed_zernikes,
# zeros = no WFE ref frame
# .575 = 575 nm
wt = np.zeros(10)
ref = prop.forward_model(.575, dd, zernikes=wt)
plt.imshow(ref, cmap='gray')
```



The system can be perturbed by modifying `wt` ("weights"), for example applying 3 nanometers of tilt and making a differential image:

```
wt[0] = 3
im = prop.forward_model(.575, dd, zernikes=wt)
plt.imshow(im - ref, cmap='RdBu')
```



Polychromatic models replace single wavelengths with a pair of vectors, one for wavelengths and another for weights:

```
from lowfsc.spectral import (
        ThroughputDatabase,
        StellarDatabase,
        LOWFS_BANDPASS,
)
star_type = 'g0v'
wvl = LOWFS_BANDPASS

mode='hlc' # or spec or wfov
dd = DesignData.hlc_design(root)
sd = StellarDatabase.bijan_data(root)
td = ThroughputDatabase.bijan_data(root)

throughput = td(mode, wvl)

weights = sd(star_type1, wvl)
fudge = sd.sparsity_fudge_factor(star_type, wvl)


im = prop.polychromatic(wvl, weights, dd)
```

There is no other difference from polychromatic and monochromatic modeling. The star type may be any of b3v, a0v, a5v, f5v, g0v, f5v, k0v, k5v, m0v, m5v. mode must be one of hlc, spec, wfov.

3.2 Reverse Modeling

Reverse modeling is the umbrella under which "estimation" or "reconstruction" falls. The reverse modeling interface is only a single function once the estimator is built. However, we must begin by building the estimator. Without going into particular detail on why things are done this way – we will publish a journal paper with such information – we "chop" LOWFS by collecting images with positive and negative Zernike perturbations about an arbitrary zero point. This is a standard macro included with the LOWFS model, but we must prepare the inputs:

```
from lowfsc.automate import chop_bipolar

CHOPSIZE = 5
chops = []
for i in range(len(wt)):
        wt2 = np.zeros_like(wt)
        wt2[i] = CHOPSIZE
        chops.append(wt2)

ref_z = cp.zeros_like(wt)
diffs, ups, downs = chop_bipolar(wvl, weights, dd, props, ref_z, chops,
    nframes_avg=nframes_avg)
for diff in diffs:
        diff /= CHOPSIZE
```

Once the differential images have been constructed, we crank through the formalism of building a LOWFS estimator:

```
from lowfsc.reconstruction import Reconstructor, prepare_Zmm,
    synthesize_pupil_shear

ref = props.polychromatic(wvl, weights, dd, ref_z)
sy = synthesize_pupil_shear(ref, chop_shear_px, 0)
sx = synthesize_pupil_shear(ref, chop_shear_px, 1)
sy /= chop_shear_px
sx /= chop_shear_px
mask = np.ones_like(ref)
mask[0,:] = 0
mask[-1,:] = 0
mask[:,0] = 0
mask[:,-1] = 0

zmm = prepare_Zmm(diffs, fm, (sx,sy), mask)
R = Reconstructor(zmm, ref)
R.estimate(ups[0])
```

The pupil shear modes teach the estimator what it looks like when the image moves laterally on the camera and removes any linear coupling between that disturbance and the Zernike estimates. The mask excludes pixels at the periphery of the image due to artifacts in the shear mode synthesis. The variable `R` is a Reconstructor object with an `estimate` method that takes any absolute image and returns estimates of the wavefront coefficients. Assuming the `seed_zernikes` line was not changed earlier in this document, the ordering of the slots are

Array Index	Unit	Meaning
0	a.u.	Detector Bias
1	nm	Z2
2	nm	Z3
3	nm	Z4
4	nm	Z5
5	nm	Z6
6	nm	Z7
7	nm	Z8
8	nm	Z9
9	nm	Z10
10	nm	Z11
11	%	Relative Flux
12	px	X Pupil Shear
13	px	Y Pupil Shear
14	n/a	spare
15	n/a	spare
16	n/a	spare

the spare slots are an artifact of the FPGA implementation of this scheme for the flight Roman-CGI. The relative flux slot is 100% when there is a component of the input image that looks exactly like the reference image.

3.3 Examples

The lowfssim repository is distributed with an examples folder, containing jupyter notebooks that execute:

1. Sensing for a 2D scan of input Z2 and Z3
2. Computation of the sensing offsets between reference and target stars, including adjusting the EM gain between them

Both of these examples showcase using lowfssim with GPU computing, utilizing a server with an Nvidia Titan XP at JPL. GPU results are over 3x faster when using Nvidia A100 GPUs at the Texas Advanced Computing Center.

4 GPU Computing

LOWFSSim may be used on a GPU to accelerate computation. Before you run any other code, you should execute the following lines:

```
from prysm import config
from prysm.fttools import mdft

mdft.clear()

import cupy as cp
```

```
from cupyx.scipy import fft as cpfft

from prysm.mathops import np, fft
np._srcmodule = cp
fft._srcmodule = cpfft
config.precision = 32
```

The cache clear is not strictly required when this chance proceeds any user code. If it is in the middle of usage, the clear is needed. The cache is conceptually similar to FFT wisdom. No further changes are needed. As a teething issue, when using Cupy with matplotlib, you must plot `arr.get()` instead of `arr`, or you will receive an error.

5 Content of the Models

All of the models assume there is precisely zero amplitude or phase error upstream of the FSM in CGI, including no error in the Roman OTA. The models assumes the DMs are placed at their nominal locations, with DM1 exactly in a pupil plane conjugate and DM2 despaced 1m from the nearest pupil plane conjugate. Both Shaped Pupil models exclude DM solutions to refine the dark hole and are based only on the amplitude masks. The SPC model does not propagate to the DM planes at all, as they would be no-ops.

6 Acknowledgements

Brian Kern, Bijan Nemati, A.J. Riggs, Hanying Zhou, John Krist, Dan Wilson, Bala Balasubramanian, and Dwight Moody have contributed to this model.