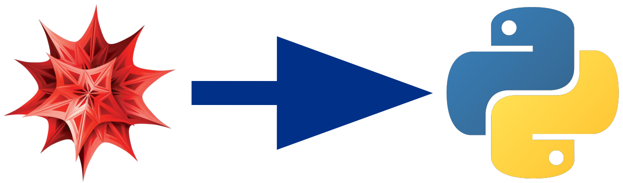
# Case Study -- Python Implementation

Dylan J Temples

MAGIS-100 Simulation & Analysis Workshop

26 January 2022

# Case Study -- Python Implementation



**Goal:** Port all functionality of the Mathematica case study notebook to Python 3

Motivation:

- Mathematica is proprietary software, licenses are expensive for non-Universities
- Python is open source, widely used, well documented, and powerful
- Utilizing high-performance computing resources for simulations (e.g. FNAL grid)

All methods available in the Mathematica notebook are duplicated in Python, mostly with identical names, arguments, and returned quantities.

# Git Repository and Installation

Currently available on personal github (private), will migrate to the MAGIS Redmine

https://github.com/dtemps123/MAGIS_analysis

**Clone with git**

**Install with pip**

**Fully documented in the git Readme**

**Usage examples**
- script-based
- interactive (Jupyter)

README.md

## MAGIS-100 Case Study Python Tools

### Dylan Temples

October 22, 2021

This notebook details the usage of the backend MAGIS Case Study Python Tools, and should serve as some level of documentation. This goes through step-by-step usage of the main functions an average user would use to do an analysis. At some critical points, it compares the calculations in this notebook to the same calculations done in the "MAGIS-100 Camera Example Parameters R21.nb" Mathematica notebook available on the DocDB: https://nd-docdb.fnal.gov/cgi-bin/sso/ShowDocument?docid=615.

## Installation

To download this repository: `git clone git@github.com:dtemps123/MAGIS_analysis.git` (this uses SSH so you'll need to add your public key to your github account).

Once downloaded, move to that directory: `cd MAGIS_analysis`

From here, the Python package can be installed which allows you to use this framework in code that lives outside this directory: `pip install .`

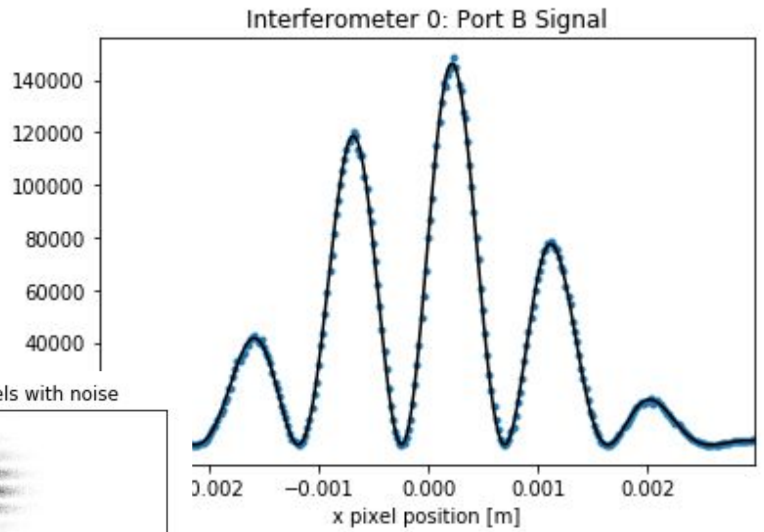
# Current Functionality

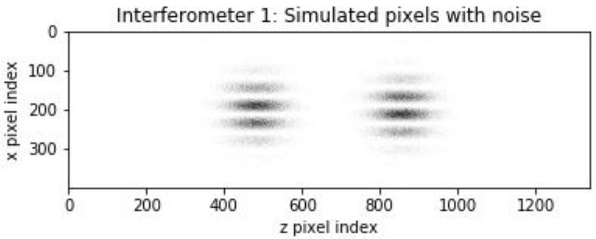
What does it do?

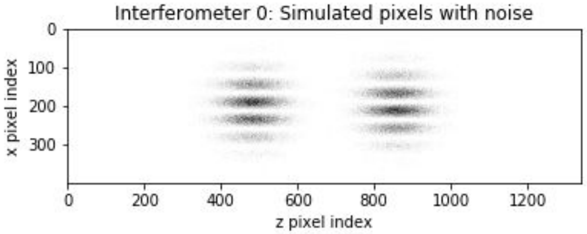
- Statistically generate atom positions from wavefunction
- Discretizes photon counts w/ noise across pixels to generate image using parameterized optics settings
- Simultaneous fits of up to 4x interference patterns to determine phase shift
- Automated reporting of simulation

What doesn't it do?

- Optical simulations (ray tracing)
- Quadratic wavefunction/fitting






Fermilab

# Performance vs Mathematica

Example code to generate one image per interferometer and extract the phase →
→

| Metric | Mathematica | Python |
|---|---|---|
| Time to simulate & fit images 100 times [sec] | 5550 | 1487* |
| Mean Δφ [rad] | -0.000204 | -0.003312 |
| Mean Δφ error [rad] | 0.000358 | 0.001577 |
| Mean fit $\chi^2$/Ndof | 16.6462 / 14 | 16.4153 / 14 |

* Most time is spent creating marginalized position distributions (only needs to be done once). Image generation and fitting is very fast.

```
import matplotlib.pyplot as plt

from MAGIS_analysis import Parameters as sp
from MAGIS_analysis import Interferometer as ic
from MAGIS_analysis import Wavefunctions as wf
from MAGIS_analysis import SimTools as st
from MAGIS_analysis import FitTools as ft

## Show the fundamental constants
print(sp.hbar)
print(sp.m)
print(sp.in_to_m)
print(sp.Gamma)
print(sp.pi)

## Show the user options
sp.PrintUsrOpts()
sp.PrintImgParams()
sp.PrintSnsrParams()

## Load up the wavefunctions
wf.PrintParams()

## Create the interferometer objects
int0 = ic.Interferometer()
int0.PrintParams()

int1 = ic.Interferometer()
int1.SetParam("atomsPerCloud", 1.1e6)
int1.SetParam("cloudRadius"  , 0.0018)
int1.PrintParams()

## Calculate the marginal distributions of atom positions
## in both interferometers
st.CalculateMarginalDistributions(int0)
st.CalculateMarginalDistributions(int1)

## Generate a single image from each interferometer
img0, x_marg0, z_marg0 = st.GenerateImage(int0, recalculate_pdfs=False)
img1, x_marg1, z_marg1 = st.GenerateImage(int1, recalculate_pdfs=False)

## Extract the images from each port on both interferometers
xvalsbox0, portAarray0, portBarray0, nzsummedA0, nzsummedB0 = ft.Extract2Ports(img0, x_marg0, z_marg0, 3, 3, int0)
xvalsbox1, portAarray1, portBarray1, nzsummedA1, nzsummedB1 = ft.Extract2Ports(img1, x_marg1, z_marg1, 3, 3, int1)

## Fit the image distributions
fr = ft.SimFits_2Port2Int(xvalsbox0, portAarray0, portBarray0, nzsummedA0, nzsummedB0,
                          xvalsbox1, portAarray1, portBarray1, nzsummedA1, nzsummedB1,
                          int0 , int1)
fr.show()

plt.show()
```

# Next Steps & Interfaces

To do:

- Improve fitting to reduce uncertainty
- Implement quadratic wavefunction and fitting
- Implement single interferometer and single port fits
- Improve automated reporting
- Test all combinations of user options
- Optimization and parallelization

Potential interfaces:

- Optical simulation tool: use simulated atom positions to generate images
- Integration into data pipeline for quick online analysis / performance monitoring
- Integration with signal model choice: phase shift vs time

# Thank You

Case Study -- Python Implementation

https://github.com/dtemps123/MAGIS_analysis

Dylan J Temples
dtemples@fnal.gov

# Package Contents

- MAGIS_analysis
- docs
- examples
- .gitignore
- LICENSE.txt
- README.md
- setup.py

| | |
|---|---|
| FitTools.py | Fit result class & fitting algs. |
| Interferometer.py | Interferometer class: parameterized |
| Parameters.py | Constants & sim params |
| Reporting.py | Saving plots and values (pdf/html) |
| SimTools.py | General image generation methods |
| Wavefunctions.py | Linear + test wavefunctions |
| __init__.py | Startup configuration |

# Package Contents

FitTools.py

```python
class FitResult():

    def __init__(self, n_param):

    def show(self):

    def show_cov_mat(self):

    def show_cor_mat(self):

    def params_to_dict(self):

    def p_errs_to_dict(self):

def GetFitQuality(data_x_values,

def Z2Ports1Int_FitFunc(z, ampA,

def FitZ2Ports1Int(z_dist_hpts, z

def X2PortsCombined_FitFunc(x, am

def FitX2PortsCombined(x_dist_hpt

def Extract2Ports(pxl_img, x_marg

def SimFits_2Port2Int(int0_x, int
```

Interferometer.py

```python
## Create an instantiable class for In
class Interferometer:
    id_n            =  0

    ## Define Default Interferometer Par
    phiphi          =  sp.pi / 2.
    atomsPerCloud  =  1e6
    cloudRadius     =  0.002   ## [ m ]
    xCenter         =  0.000   ## [ m ]
    zCenter         = -0.0075  ## [ m ]

    ## Derived Parameters
    acceptedPhotons  = atomsPerCloud * sp
    signalPerCloud   = acceptedPhotons *
    pixelAcceptance  = (sp.snsr_params["d
    signalPerPixel   = pixelAcceptance*si

    ## Initialization Method
    def __init__(self, id_n):

    def CalculateDerivedParams(self):

    def SetParam(self, key, val):

    def GetParam(self, key):

    def PrintParams(self):

    def GetName(self):
```

Wavefunctions.py

```python
## Wavefunction Parameters
tFinalBS   = 3.0  ## [ s ]
tExtra     = 0.1  ## [ s ] time be
portBvz    = 0.15 ## [ m/s ] diffe
zsep       = portBvz * tExtra ## [
kFringe    = 1/0.0003
yFringe    = 2.0 * sp.pi / kFringe
aQuad      = 1e-12

## Dictionary for Sensor Paramet
params = {
}

def CalculateDerivedParams():

def SetParam(key, val):

def GetParam(key):

def PrintParams():

## Define the actual wavefuncti
def testfunction(x, z, xA, zA, p

def wavefunction(x, z, xA, zA, p

def EvaluateWavefunction(wf_obj,

def Evaluate(intfr, x_array, z_a
```

# Package Contents

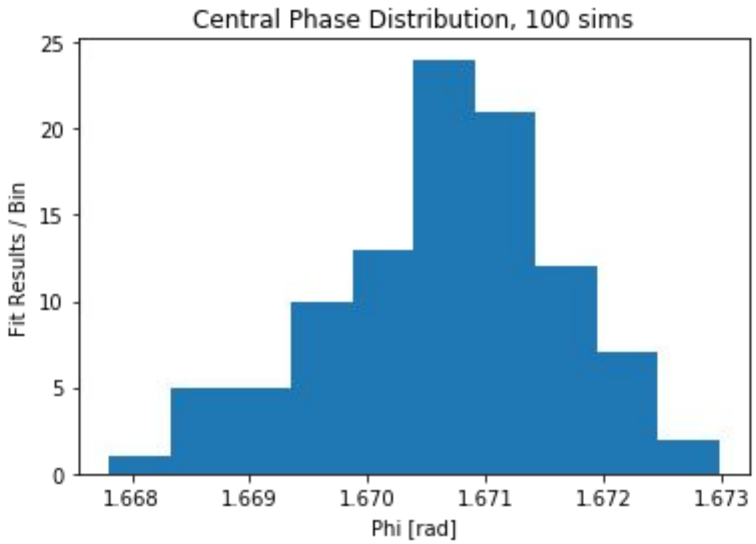
SimTools.py

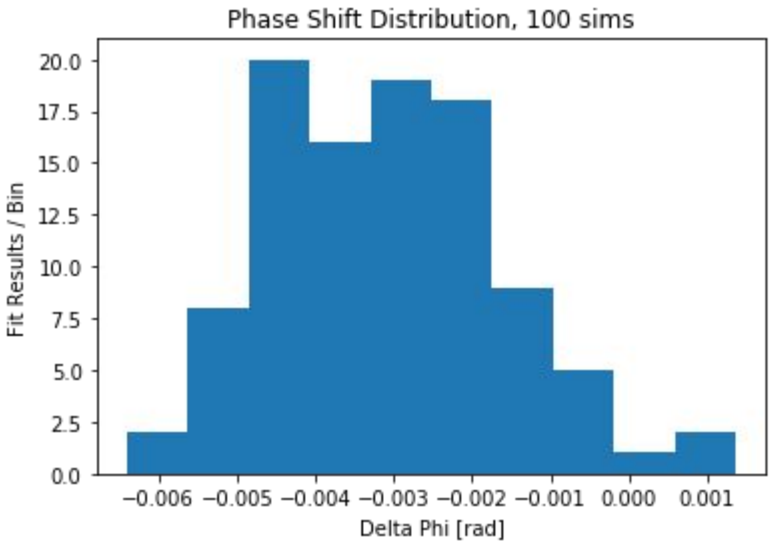
```
def PrintDerivedParameters(intfr0,intfr1):...

def CalculateMarginalDistributions(intfr, n_samps=2000):...

def SampleFromInvertedCDF(rvars, pvals, n_samps):...

def WaveFunctionPixelation(intfr):...

def PhotonPixelation(phtn_array):...

def DiffusedAtomPositions(N_atoms):...

def RandomSampleAtomPositions(intfr, port="A"):...

def ProjectToPixels(portA_atoms_x, portA_atoms_z, portB_ato

def GenerateImage(intfr, recalculate_pdfs=False):...
```

Reporting.py

```
data_path = "./"            ## Path where this
series_id = "2222-9999_0000"    ## The uni

smry_prefix = "log"         ## Appended to t
smry_suffix = ".md"         ## Appended to t

enable = False                 ## Do you 

options = {...
}

def SetOpt(key, val):...

def GetOpt(key):...

def PrintOpts():...

def Initialize(path_str="./",series=None):

def LogTextLine(*message):...

def LogOutputLine(*output_msg):...

def LogCodeBlock(*code):...

def LogImage(img_file_name, fig_obj):...

def LogMarkdownDump(mkdn_str):...
```

# Phase Shift Statistics



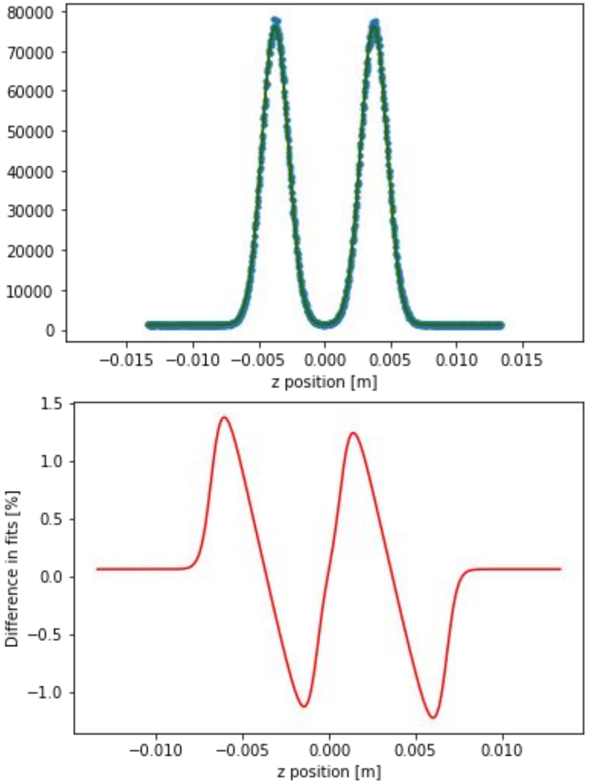





Fermilab

# 1D Fit Result: Python vs Mathematica

Z-position distribution (1 interferometer, integrated over both ports)




X-position distribution (1 interferometer, integrated over both ports)