KNOT USER GUIDE

Last updated: 03/09/2021

Contents

[Abbreviations**:** 3](#_Toc66212796)

[Part 0: Installing Python and relevant packages 4](#_Toc66212797)

[Part 1: Simulating data using KNOT 8](#_Toc66212798)

[Part 2: Applying KNOT to real data 9](#_Toc66212799)

[Part 3: Visualization options 10](#_Toc66212800)

[Part 4: Post-Processing 11](#_Toc66212801)

[Appendix A: User parameters 12](#_Toc66212802)

[Instrument parameters 12](#_Toc66212803)

[Appendix B: Troubleshooting common errors 14](#_Toc66212804)

[Appendix C: Code overview 15](#_Toc66212805)

[\_\_ENUM.py 15](#_Toc66212806)

[\_\_FUNCTION.py 15](#_Toc66212807)

[\_\_OPERATION.py 17](#_Toc66212808)

[\_\_VISUALS.py 18](#_Toc66212809)

[\_CREATE.py 18](#_Toc66212810)

[\_INITIALIZE.py 18](#_Toc66212811)

[\_PREPARE.py 18](#_Toc66212812)

[\_RECOVER.py 18](#_Toc66212813)

[\_SEGMENT.py 18](#_Toc66212814)

[\_TRACK.py 18](#_Toc66212815)

[CROP.py 18](#_Toc66212816)

[main.py 18](#_Toc66212817)

[USER.py 19](#_Toc66212818)

[Appendix D: Contact information 20](#_Toc66212819)

# Abbreviations**:**

DH: Double Helix

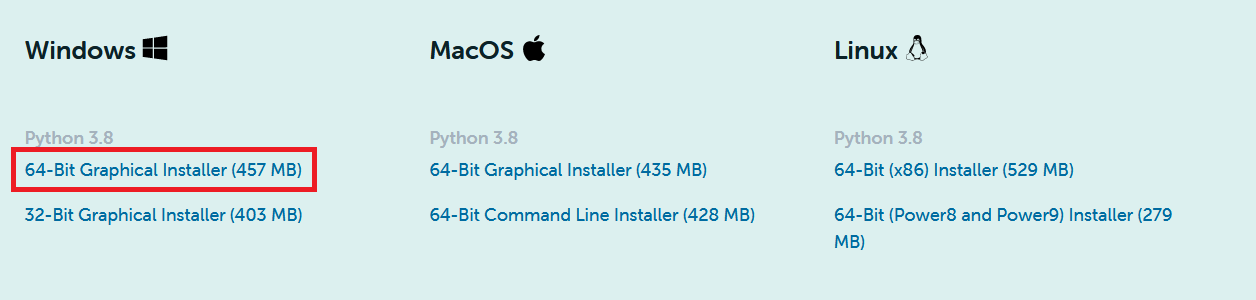
PSF: Point Spread Function

# Part 0: Installing Python and relevant packages

*This segment of the guide assumes the following:*

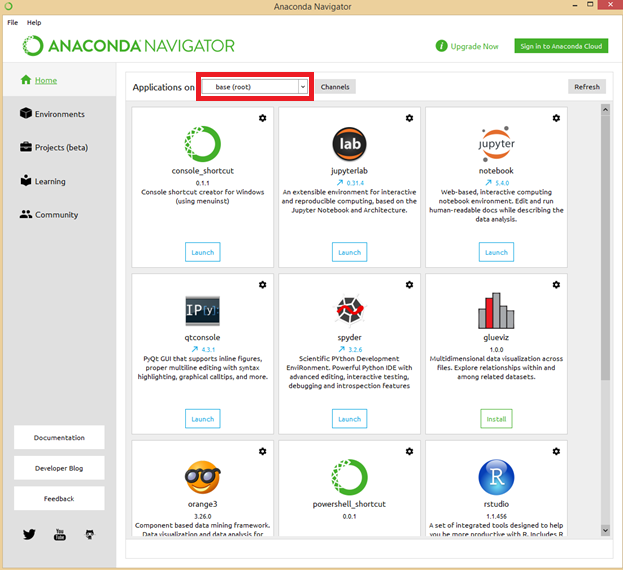
* *You do not have python installed on your computer*
* *You do not have anaconda for python installed on your computer*
* *You have a stable internet connection*

For its ease of use, we will be installing Anaconda for python (A recent release can be found here: <https://www.anaconda.com/products/individual> ). Download the appropriate installer to install Anaconda.



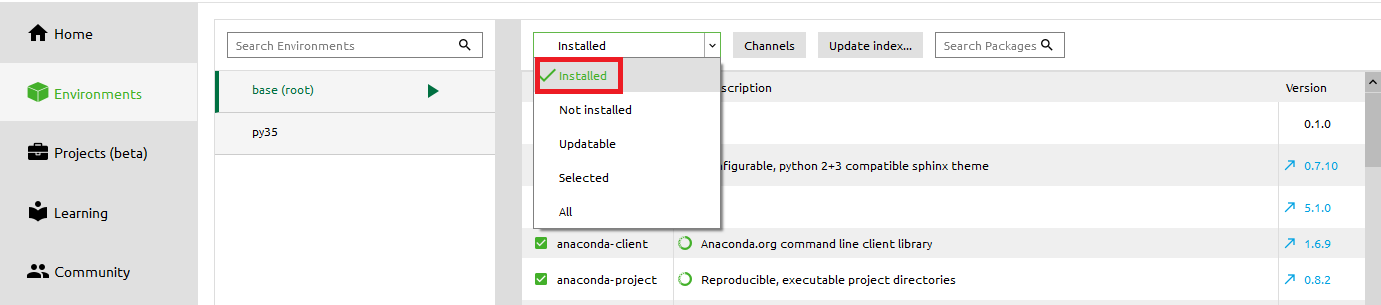
When installing, use all the default settings. Python can get finicky with the PATH variable and we do not want to cause an issue with python.

Once Anaconda is installed, open the **Anaconda Navigator** program shown below. You will see several different applications available to run or download that allow you to run and develop python code.



The base environment should already be selected. **IF YOU HAVE AN EXISTING PYTHON INSTALLATION, CREATE A NEW PYTHON ENVIRONMENT AT THIS POINT.**

Open the **Environments** tab and set the dropdown menu to Installed.

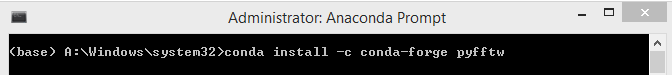


Proceed to use the search bar to ensure that the following packages and their dependencies are installed:

* numpy (*For array manipulations. Absolutely mandatory*)
* imageio (*For loading and saving images. Absolutely mandatory*)
* scipy (*For Delaunay triangulation. Mandatory*)
* matplotlib (*For visualization. Optional but helpful*)
* pyfftw (*For the FFTW. Optional but not installing requires workarounds*)
* scikit-image (*I don’t remember. Optional now?*)
* scikit-learn (*For K-means clustering. Obsolete now?)*

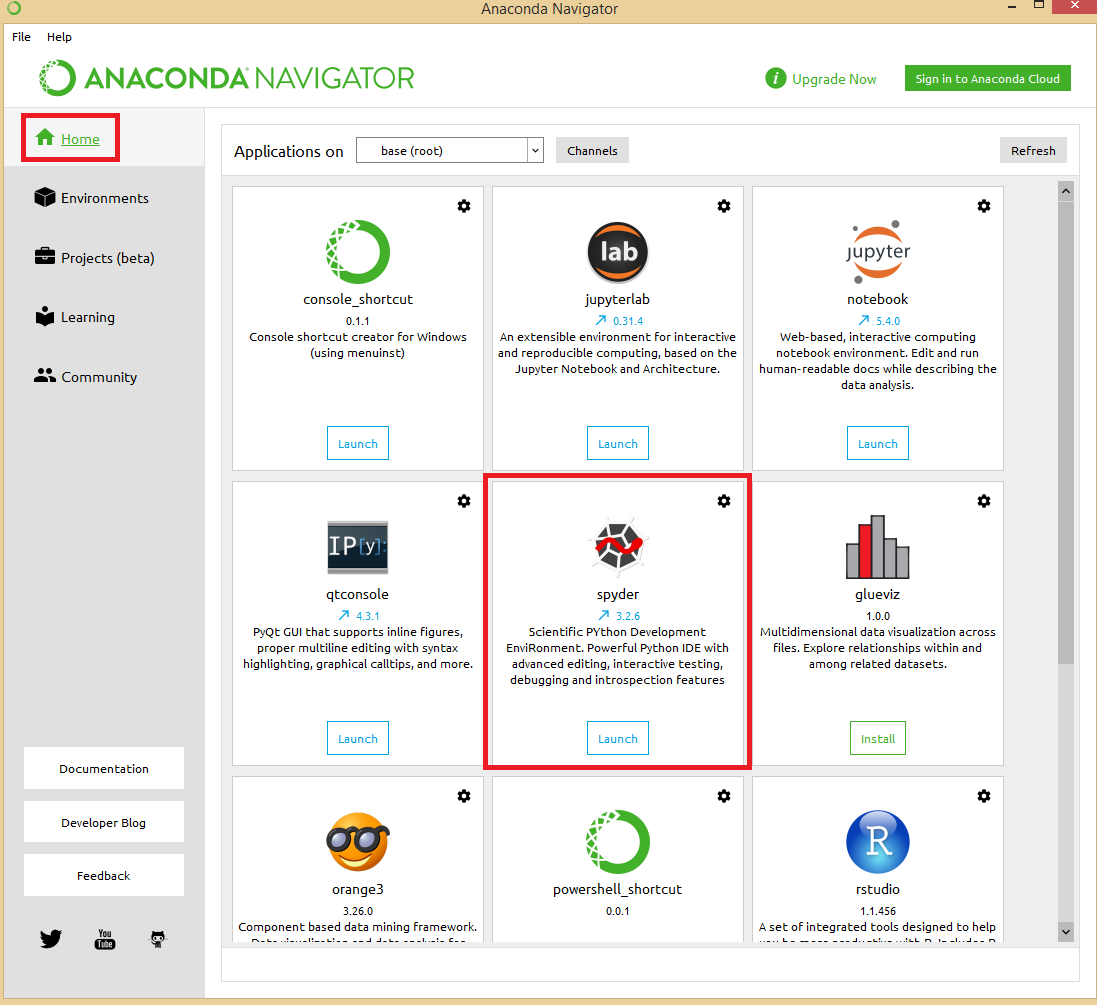
If one of these packages is missing, you can change the dropdown menu to Not Installed and use the search bar to locate these packages and install them. However, the one package you will be unable to install is the pyfftw package. For this package, you will need to open **Anaconda Prompt** likely using administrator privileges. In the command prompt, input the following command and install pyfftw:

conda install -c conda-forge pyfftw



**IF YOU ARE USING A NEW PYTHON ENVIRONMENT, MAKE SURE TO CHANGE ENVIRONMENTS IN ANACONDA PROMPT FIRST.**

Once you are finished installing pyfftw, you may close **Anaconda Prompt** and return to the Home tab of **Anaconda Navigator**.



If it is not already installed, install **Spyder**. Then proceed to the next section.

# Part 1: Simulating data using KNOT

Here we talk about all those goodies

# Part 2: Applying KNOT to real data

Simulating data is not an effective way to learn more about the real world, so KNOT needs to be able to access your experimental data to process. For best results, it is recommended to do the following:

* Crop to regions of interest (speeds up processing time greatly)
* Ensure simulated PSF is the same as experimental PSF (needs to only be done once per instrument. N.B. if you load an experimental PSF, it may be best to always use the same dimensions as your image.)
* Ensure preprocessing variables are suitable to your data (this should only need to be done once.)
* Ensure the linking radius is appropriately set (this may be data dependent)
* Run KNOT

For this example, we will be using the following 400x400 movie with 100 frames:

## Cropping to a suitable ROI

# Part 3: Visualization options

Pretty pictures

# Part 4: Post-Processing

For if you want to do things with it outside

# Appendix A: User parameters

Here’s a detailed list of everything we use and what it means

Apart from the list of files to run KNOT on and the dictionaries for updating or visualizing segments of the code, all user-defined parameters are present in USER.py, and are sorted by their appearance in the workflow. Detailed descriptions of each parameter follow, organized as follows:

**Parameter**: <units> Description | Default Value

Please note that the value of units also describes the organization of the variable. For example, (#, …, #) describes a tuple, [#, …, #] describes a list, [[#, …, #]] describes a 2D (or higher if there are more brackets) array, and a simple # denotes a singular value.

Additionally, the following abbreviations are used for convenience:

px: pixel

um: micron

s: second

f: frame

Enum: Enumerator (acceptable values are provided in the description)

## Instrument parameters

**CHIP**: <(px, px)> The number of pixels along each dimension (x, y) of the camera. Only applicable when simulating new images, as KNOT uses the dimensions of the input image otherwise. | (64, 64)

**RES**: <(um/px, um/px)> The lateral resolution of the camera along each dimension (x, y). | (0.0068.5, 0.00685)

**DOF**: <(um, um)> The Depth of Field used when performing 3D tracking. Currently, the first parameter is used when doing XYZ tracking using the DH-PSF, while the second parameter is used when doing XYZT tracking using the stretching lobe phase mask. Other phase mask types (such as Astigmatism and Tetrapod) may be included in the future, and will be included in this variable. | (4.000, 1.600)

**FRATE**: <s/f> The frame rate of the camera in seconds per frame. Used more for visualization than actual computation. | 0.030

**KER\_TYPE**: <Enum> [NONE, HELIX] The phase mask to simulate for deconvolution or simulation. The HELIX value also supports the Airy disk PSF provided that KER\_Z and KER\_T are both 1. | PM.HELIX

# Appendix B: Troubleshooting common errors

Disgusting

# Appendix C: Code overview

It should be noted that the code base is split up into three categories based on the name prefix:

* \_\_NAME.py indicates a python module that contains global constructs that both span and are independent of the KNOT process. It is advised to leave these files untouched.
* \_NAME.py indicates a python module that contains functions and classes necessary for KNOT to process data. The filenames are selected to illustrate the order of execution.
* NAME.py indicates a python module that acts as a controller for the user, be it through parameters or the main.py module which runs KNOT. These files can manually be run by the user and will complete a specific task.

If you are looking to extend or modify any part of the KNOT code, please make sure to read this section and any associated comments in these files.

## \_\_ENUM.py

This module contains enumerators used for various purposes such as determining the file format used, function codes for \_\_FUNCTION.py, and the phase mask or localization scheme being used.

## \_\_FUNCTION.py

This module contains the Function class, which allows for semi-analytic construction of motion using functions such as polynomials, sinusoids, or Gaussians as well as non-differentiable motion such as Wiener processes (also known as Brownian motion). The actual usage of the Function class is relatively limited and may be expanded in the future to include additional functionality. A brief tutorial is given in Part 1 and secondary information is given below. Importing this module is usually denoted as follows:

import \_\_FUNCTION as FXN

Function objects are constructed using the following template:

\_Handle(argument 1, … argument n, amp=amplitude, off=offset, seed=seed)

Which will construct a function fitting the handle provided with arguments 1 through n. The function will be scaled by the value provided in amp and offset from 0 by the value provided in off. For random processes, the seed to use is provided in seed. All arguments have default values (function arguments vary with the function being used, amp=1, off=0, seed=0) and the number of function arguments provided depends on the function used. When writing functions in equation form, amp will be written as and off as . Below is a list of functions, their arguments, and descriptions:

* Point() – A point in space; no arguments, amp has no effect, off shifts the point and is the only argument considered.
* Line() – A line in space; no arguments, amp determines the slope, off shifts the starting point.
* Poly(cn, cn-1, …, c2, c1) – Constructs an nth degree polynomial with coefficients cn­ through c1. amp determines the scale of the polynomial (and is recommended to be left at 1) while off affects c0. The number of arguments input determines the degree of the polynomial.
* Sine(f=1, phi=0) – Constructs a sine wave with frequency f and phase phi. Note that f is determined to be in oscillations per unit, so the default parameter will cycle once every unit. phi is evaluated in radians.
* Cosine(f=1, phi=0) – Constructs a cosine wave with frequency f and phase phi. Behaves like Sine(f, phi) but with the phase shift.
* Exp(mu=0, k=1) – Constructs an exponential with shift mu and rate constant k. Note that negative values of k correspond to a decaying exponential.
* Gauss(mu=0, sigma=1) – Constructs a non-normalized 1-D Gaussian distribution with mean mu and standard deviation sigma.
* Lorentz(mu=0, gamma=1) – Constructs a non-normalized 1-D Lorentzian distribution with mean mu and FWHM gamma.
* Wiener(mu=0, sigma=1) – Constructs a Wiener process with drift mu and standard deviation sigma. Note that these parameters define the underlying Gaussian distribution which the Wiener process accumulates and is dependent on the time steps provided in the domain. See [the Wikipedia page](https://en.wikipedia.org/wiki/Wiener_process) for more information about the Wiener process.

It should be noted that currently Function objects can be added, subtracted, multiplied, and divided using the normal python operators. (e.g. h = f + g) Additional function composition is also possible. For example, if you wanted to obtain a decaying sinusoid, you may construct it as follows:

Decay = FXN.\_Exp(0, -1)

Decay\_Sine = FXN.\_Sine(amp=Decay)

The type of stop-and-go motion observed in cells may be simulated using several Gaussian impulses in a Wiener process:

dom = np.linspace(0, 10, 1001) # The domain of evaluation #

imp = FXN.\_Point() # Initialization #

for i in range(10):

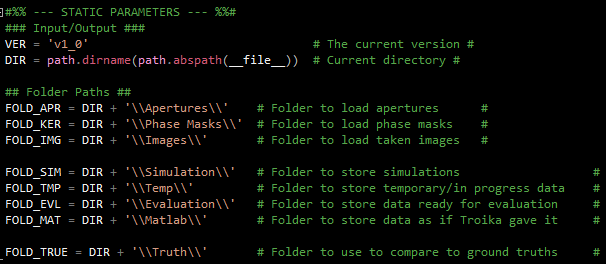
mu = 10 \* np.rand() # Generate a number between 0 and 10 #

imp = imp + FXN.\_Gauss(mu, 0.1) # Impulses are ~0.5 seconds long #

motion = FXN.\_Wiener(imp) # Brownian motion with impulses #

## \_\_OPERATION.py

This module handles various input/output methods for loading and saving data. The only parameters that may be of interest are the folder paths dictated at the top of the file:



Modification of DIR or any of the folder paths will cause KNOT to create new directories in those locations if not already present. As such, it is recommended to copy (preferably cropped) images into KNOT’s Images folder before running.

## \_\_VISUALS.py

This module contains helpful building blocks for visualization of data analysis or processing progress.

The \_ProgressBar function is used commonly for long operations to show the current progress in the console.

The methods listed under **STATIC METHODS** are useful for visualizing SFD distributions (\_DispSFDDist), displaying 2-D images on 3-D axes (\_Disp3Dimg), showing the time dependence of trajectories using a saturation and value gradient (\_DispLineGrad), and a helpful tool to draw images in figures at certain locations on the screen (\_VisImg).

For examples, the code that constructs various figures in the manuscript is shown in the section labeled **MANUSCRIPT FIGURES**.

If you wish to modularly create new visualizations, it may be best to place them as new functions in this module.

## \_CREATE.py

The module that contains information for particle simulation.

## \_INITIALIZE.py

The instrument simulation module.

## \_PREPARE.py

The preprocessing module.

## \_RECOVER.py

The deconvolution module.

## \_SEGMENT.py

The particle identification module.

## \_TRACK.py

The tracking module.

## crop.py

The module that crops images to form a specific region of interest in space and time.

## main.py

The main module used to run KNOT.

## simulate.py

The module that simulates particle motion using \_CREATE.py as a base.

## USER.py

The module that contains all the user parameters needed to run KNOT. See Appendix A for descriptions of each parameter.

# Appendix D: Contact information

You can reach the code designer at (Jorge Zepeda O: [email@domain.com](mailto:email@domain.com)) or the corresponding author (Christy F. Landes: [email@domain.com](mailto:email@domain.com)) for any further questions you may have.