Practice with random Datasets

Julio Candanedo[*]

May 13, 2024

Abstract

In this work, we give a method to compute random datasets, typically encountered in Bragg-scattering experiments. This allows for the rapid prototyping and bench-marking of relevant algorithms.

Contents

1 Introduction

Bragg-spots encountered in Wide-Angle-Scattering, may be indexed by a series of three integers known as the *Miller-indices*, denoted respectively as h, k, and l. They appear as an ordered speckled pattern on the detector, such that every pixel has an associated $\{h, k, l\}$ index. However, most pixels are not illuminated, as subject to selection-rules of the relevant space-group, and have a numeric value of null, i.e. '0'. The few illuminated pixels have some nonzero value, due to the coherence of the illuminating beam, correlation of the sample, and displacement of the Ewald sphere from a 3d Bragg spot.

In this work we generate a 'random' dataset consisting of N snapshots (denoted by i), the illuminated (nonzero) pixels (enumerated/indexed by x, total number of pixels given by D), and their numerical value (denoted by d). This forms parts of a sparse-matrix d_{ix}, this may be visualized as a dense matrix/2d-array with many zero entries.

2 Algorithm & Inputs

We create an algorithm to compute a random dataset of N snapshots with an average of $\langle n\rangle$ Bragg-spots (per snapshot), with the following inputs:

- average number of Bragg-spots per snapshot $\langle n\rangle$,

[*] `jcandane@uwm.edu, jcandane@asu.edu`

- total number of snapshots N,
- total number of pixels, D.

With these we define two numbers: m a surrogate-number, and s the total-number of nonzero/sparse elements. These may be calculated by:

$$m = \frac{2D}{\langle n \rangle}$$
$$s = \langle n \rangle N \qquad .$$

Now we sample s number of random-integers from a uniform-distribution on $[1, m]$, this yields an array $\tilde{i}$. Next these are cumulatively-summed and modular-integer-divided by the highest-pixel-number D over all entries. This results in an array of many consistently-increasing parts. The consistently-increasing parts define the snapshot.

2.1 sampling for the snapshots

The snapshots are defined as the increasing-domains of i. Therefore, $\delta x = x[1:] - x[:-1]$ defines when this a given increasing-domain ends, i.e. when $\delta x < 0$, these locations are used to determine the start of a new snapshot.

⚠ Because the snapshots are defined for increasing-regions of the 1d-pixel-array, their actual number N is also defined by them. Therefore the desired N and the actual N can in general differ.

2.2 sampling for the data

The data must be strictly positive, as the detector cannot detect a negative signal. Therefore sample from a skewed log-normal distribution, defined on the positive-real-numbers, $\mathbb{R}^+$.

2.3 Initial Example

```python
import numpy as np
import matplotlib.pyplot as plt

######## INPUTS ##########
np.random.seed(seed=179)
D   = 4000 ### number of pixels
avg = 200
N   = 500
############################

m   = int(2*D/avg) ### surrogate counter
sND = int(avg*N) ### approx number of sparse entries

x = np.cumsum( np.random.randint(1, m, size=sND) ) % D
i = np.cumsum( np.diff( x, append=x[-1]) < 0 )
d = np.random.lognormal(mean=0.0, sigma=1.0, size=i.size)

##### analyze
snapshot_sizes = np.diff( np.where( np.diff(i, prepend=-1, append=i.size)!=0)[0])
counts, bins = np.histogram(snapshot_sizes)
plt.stairs(counts/snapshot_sizes.size, bins)
```

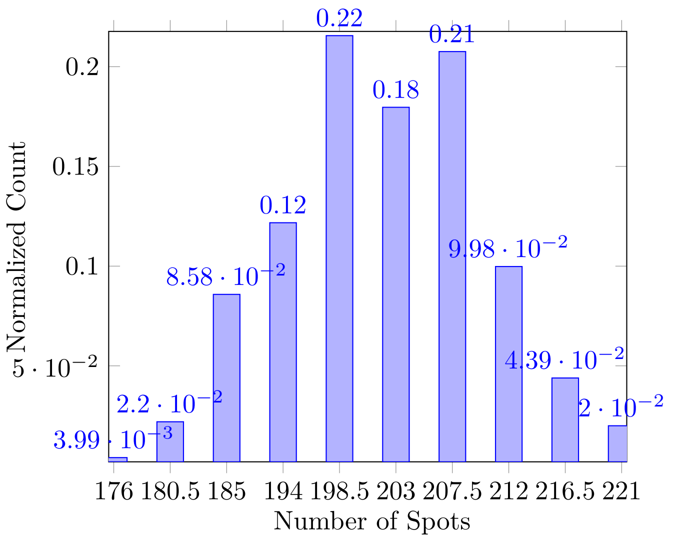

Figure 1: Result from the code above. Plots the distribution, given by the counts, of many snapshots.

```
plt.title("snapshot sizes")
plt.show()
```

Executing the code above yields fig. 1, with the resulting distribution being approximately normal.

2.4 Definition

In order to use this code, we package this into the following python-function/definition.

```
def randombragg(avg:float, D:int, N:int, seed:int=179):
    """
    Get a 'random' dataset of Bragg-diffraction
    GIVEN: avg:np.float64 (average number of activated-pixels per snapshot)
           D:int (number of total detector pixels)
           N:int (**number of snapshots, about!!)
           seed:int=17 (psuedorandom np seed)
    GET:   d:np.ndarray[1d, float] (data-value for each activated-pixel)
           i:np.ndarray[1d, int] (snapshot for each activated-pixel)
           x:np.ndarray[1d, int] (pixel for each activated-pixel )
    """

    np.random.seed(seed=seed)
    m   = int(2*D/avg) ### surrogate counter
    sND = int(avg*N) ### approx number of sparse entries

    x = np.cumsum( np.random.randint(1, m, size=sND) ) % D
    i = np.cumsum( np.diff( x, append=x[-1]) < 0 )
    d = np.random.lognormal(mean=0.0, sigma=1.0, size=i.size)

    return d, i, x
```

2.5 JAX Definition

```python
import jax
from functools import partial

#@partial(jax.jit, static_argnames=['avg','D','N'])
def randombragg(avg:float, D:int, N:int, seed:int=179):
    """
    Get a 'random' dataset of Bragg-diffraction
    GIVEN: avg:np.float64 (average number of activated-pixels per snapshot)
           D:int (number of total detector pixels)
           N:int (**number of snapshots, about!!)
           seed:int=17 (psuedorandom np seed)
    GET:   d:np.ndarray[1d, float] (data-value for each activated-pixel)
           i:np.ndarray[1d, int] (snapshot for each activated-pixel)
           x:np.ndarray[1d, int] (pixel for each activated-pixel )
    """

    key = jax.random.key(seed)
    m   = int(2*D/avg) ### surrogate counter
    sND = int(avg*N) ### approx number of sparse entries

    x = jax.numpy.cumsum( jax.random.randint(key, (sND,), 1, m) ) % D
    i = jax.numpy.cumsum( jax.numpy.diff( x, append=x[-1]) < 0 )
    d = jax.random.lognormal(key, sigma=1.0, shape=(i.size,))

    return d, i, x
```