Python scripting
&
A study on eigenvalues spacings

Alberto Salvador
Physics of Data, University of Padua
Course: Quantum information and computing
Assignment 3

11 November 2024

Outline of the presentation

- **Interfacing Fortran with Python**
  - Updates on the Fortran code from Assignment 1
  - Python scripting for execution

- **Eigenvalues spacings:** a study for random matrices
  - Random matrix theory (Hermitian matrices)
  - Eigenvalue spacings distributions
  - Code implementation and results

Interfacing Fortran with Python

FORTRAN

- Compiled ⟶ faster
- Optimized numerical computations
- Numerical precision
- Complex and less intuitive

PYTHON

- Powerful data analysis tools
- Easy of use
- ML & parallel coding extensibility
- Interpreted ⟶ slower

PYTHON SCRIPTING

Interfacing Fortran with Python

Assigment_1's code update

BUT FIRST.....

Some **updates** for the matrix multiplication Fortran code from 'Assignment 1':

- Dedicated output file for each method
- Log-log plot and linear fit of the curve N vs. CPU_time

$$\underbrace{y(x) = ax^b}_{\text{linear scale, polynomial fit}} \longrightarrow \underbrace{\log(y) = \log(a) + b\log(x)}_{\text{log-log scale, linear fit}}$$

Preferable!

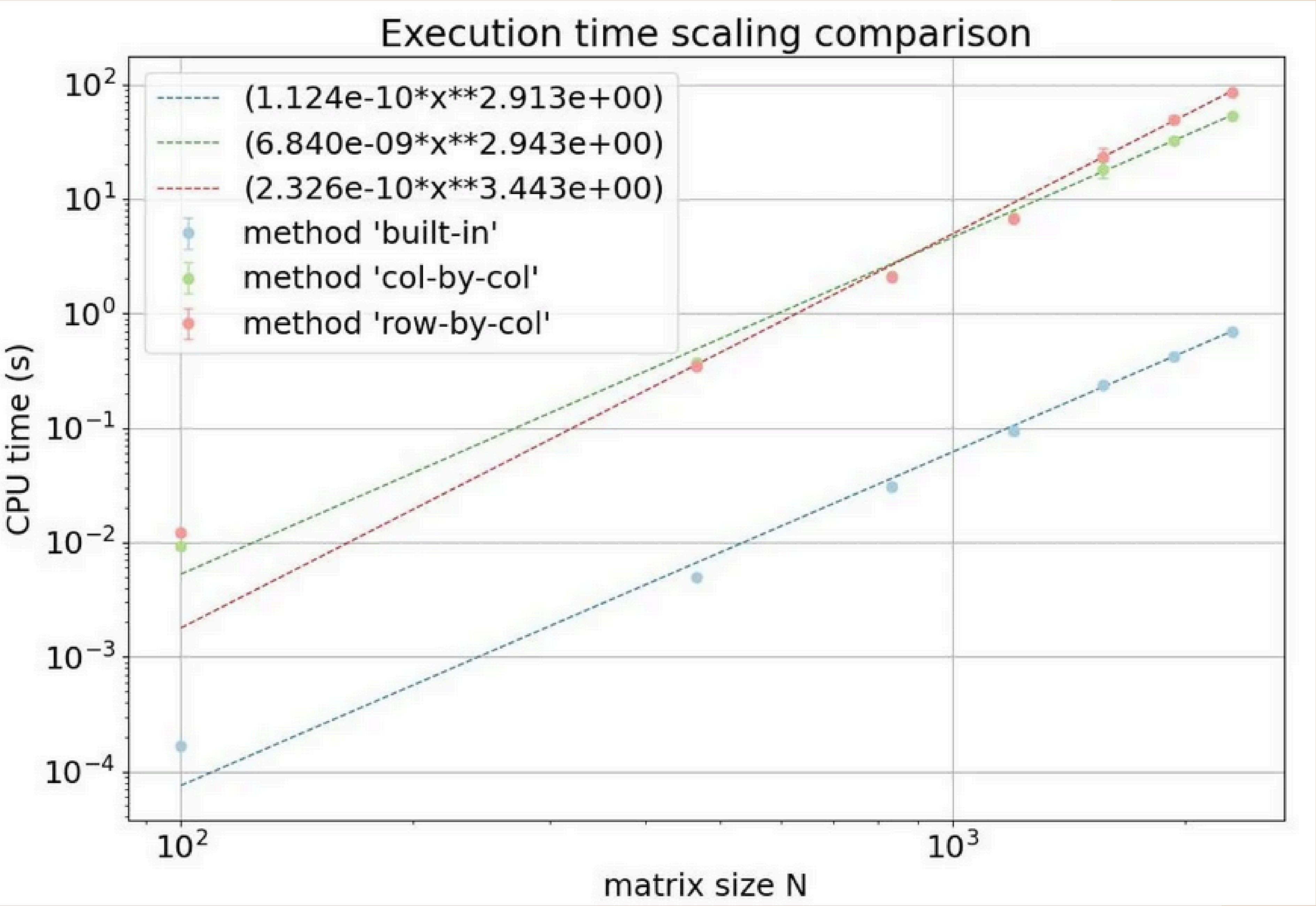
- easy curves comparison
- pre-factor VS exponent

Interfacing Fortran with Python

Python scripting

```
root@LAPTOP-9GNANRQ5:/home/albertos/quantumInfo/ex3/3.1/data# ls -1 *.csv
execution_times_M1.csv
execution_times_M2.csv
execution_times_M3.csv
```

```python
import subprocess
import numpy as np

#the path to the executable
path_exe = "./matmatmul.exe"

#defining the range for the size of the matrix
N_range = [100, 2300]
N = np.linspace(start=N_range[0],
                stop=N_range[1], num=7).astype(int)

# Initializingthe input variables
debug = ".FALSE."
verbosity = 0 #integer in {0,1,2}
rep_meas = 2

#taking the measures
for n in N:
    # input string for terminal
    input_from_terminal = str(n) + "\n" + debug + \
                          "\n" + str(verbosity) + \
                          "\n" + str(rep_meas)


    #executing bash command to run the fortran program with the specified input
    subprocess.run(path_exe, input=input_from_terminal,
                        text=True)
```



Eigenvalue spacings

Random (Hermitian) matrix theory

- $\boxed{\text{Random matrix}}$ ≡ entries generated from a p.d.f $\boxed{m_{ij} \sim p(x)}$
- $\boxed{\text{Hermitian matrix}}$ ≡ complex square, equal to its conjugate transpose $\boxed{m_{ij} = \overline{m_{ji}}}$

- Special properties:
  - A model for heavy nuclei (Wigner)
  - A model for the behaviour of large disordered systems
  - (...and more...)

- An important result: **eigenvalues spacing**

Given a random matrix one can compute the distance between its eigenvalues.

Different scenarios:

- hermitian → Wigner - Dyson
- diagonal → Poisson

$$s_{i,norm} = \frac{s_i}{\sum_{i=1}^{N} s_i}$$

pdfs of the normalized spacings

USEFUL →

spacings distribution as a *signature* for chaotic/exactly-solvable dynamics

Eigenvalue spacings

Code implementation

```python
def gen_hermitian(N, mean_gaussian=0, var_gaussian=1):

    # Creating an empty matrix
    M = np.empty((N, N), dtype=float)

    # Generating random entries for the upper triangular
    upper_triangle_size = int(N * (N + 1) / 2)
    upper_triangle_flat = np.random.normal(loc=mean_gaussian,
                                           scale=np.sqrt(var_gaussian), size=upper_triangle_size)

    # Filling the matrix upper triangular part + diago
    M[np.triu_indices(N)] = upper_triangle_flat

    # Filling the lower triangular part exploiting herm
    lower_triangle = M.T - np.diag(M.diagonal())
    M = M + lower_triangle

    return M
```

1

```python
def compute_eigenspaces(matrix):
    #computing the eigenvalues
    eigs, _ = np.linalg.eig(matrix)

    #filtering ans sorting
    eigs = np.unique(eigs)
    eigs = np.sort(eigs)

    #computing the spacings
    eigs_shifted = np.roll(eigs,shift=-1)
    spacings= np.abs(eigs_shifted-eigs)
    spacings= spacings[:-1]

    #computing the average spacing
    avg_space = spacings.mean()

    #Compute the normalized spacing
    normalized_spacings = spacings/avg_space

    return normalized_spacings
```

2

3

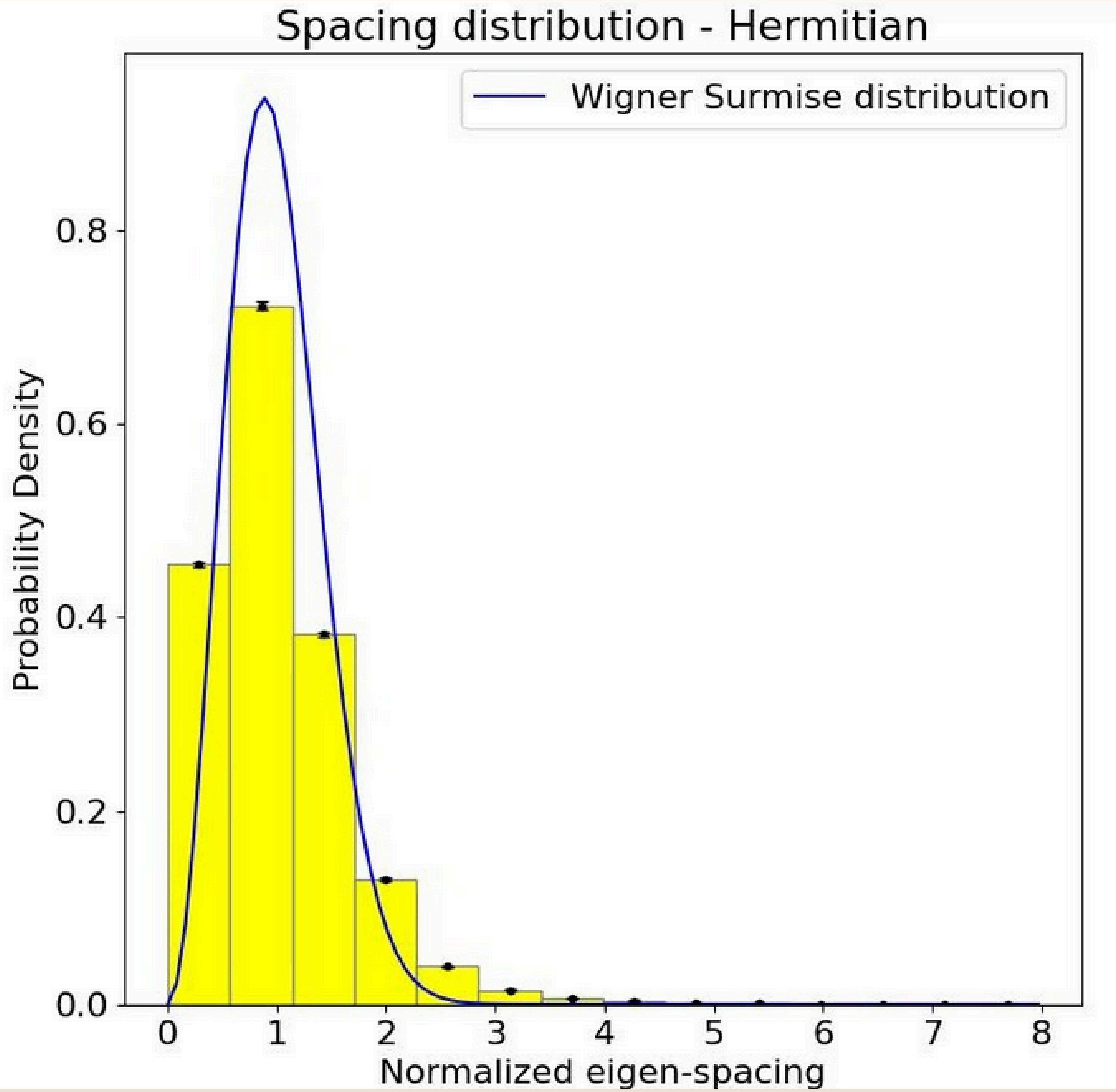
```python
def wigner_surmise(s):
    return (32 / np.pi**2) * s**2 * np.exp(-4 * s**2 / np.pi)

def poisson_distr(x, lam):
    return poisson.pmf(x, lam)
```

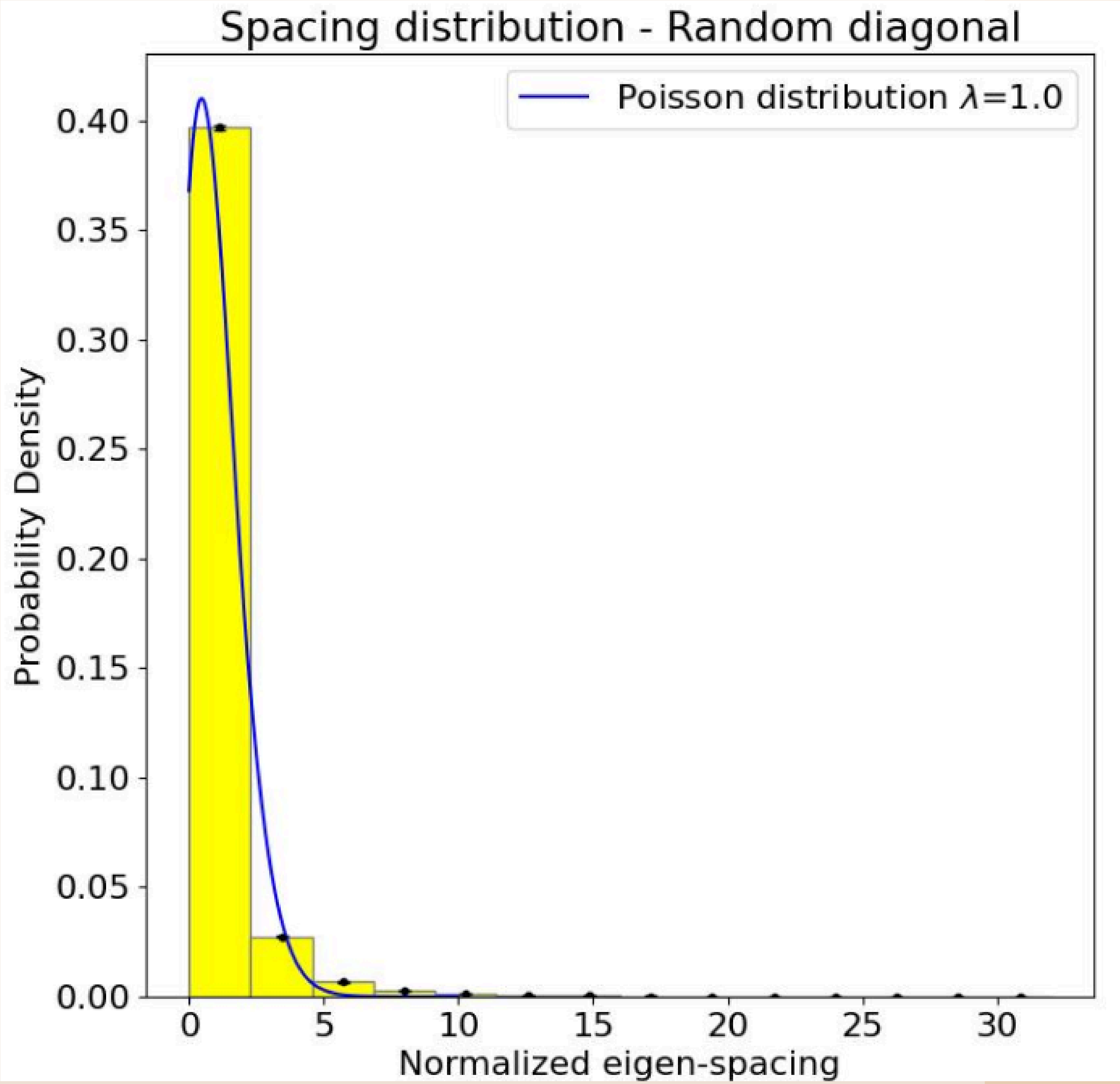
4

```python
#Setting parameters
N=100
N_meas=1000

## Making repeated measures
eigsp_herm = np.full((N_meas, N-1), np.nan)
eigsp_rand = np.full((N_meas, N-1), np.nan)
for i in range(N_meas):
    #Generating the matrices
    M_herm = gen_hermitian(N)
    M_rand = gen_random_diag(N)

    #Computing the spacings
    norm_herm = compute_eigenspaces(M_herm)
    norm_rand = compute_eigenspaces(M_rand)

    # Filling the array containing the measures
    eigsp_herm[i, :len(norm_herm)] = norm_herm
    eigsp_rand[i, :len(norm_rand)] = norm_rand
#flattening and removing eventualy NaN entries
eigsp_herm = eigsp_herm.flatten()
eigsp_herm = eigsp_herm[~np.isnan(eigsp_herm)]
eigsp_rand = eigsp_rand.flatten()
eigsp_rand = eigsp_rand[~np.isnan(eigsp_rand)]


# Plotting the distributions of spacings
plot_histogram_spacings(eigsp_herm, name='Hermitian')
plot_histogram_spacings(eigsp_rand, name='Random diagonal')
```

Eigenvalue spacings

Results





Conclusions

- Prefer Fortran (or specific optimized libraries) for heavy numerical computation and Python for generic purposes or data analysis tools
- Built-in function provides an advantage in changing the *prefactor* of the polynomial law regulating matrix multiplication
- Eigenvalues spacing of random hermitian matrices follows a *Wigner-Dyson* distribution while for a diagonal random matrix the distribution is *Poissonian*

Thanks for the attention 😊