SBI

Group 2412

Beccarelli Cesare Tuscano Alessio

Tramarin Chiara Bettio Vittoria

UNIVERSITÀ DEGLI STUDI DI PADOVA

What is SBI?

Simulation-based inference, rather than directly computing the likelihood function, **uses simulations to generate data** under various parameter settings.

By comparing these simulated datasets to the actual observed data, they can infer the parameters of interest.

We worked with the SNPE method for SBI

Parameter Bayesian Inference

$$\overset{\text{Posterior}}{\boxed{p(\theta|x)}} = \frac{\overset{\text{Likelihood}}{\boxed{p(x|\theta)}}\,\overset{\text{Prior}}{p(\theta)}}{\underset{\text{Evidence}}{p(x)}}$$

Problem:
Intractable Likelihood

In some complex or high-dimensional models the **likelihood function cannot be evaluated**.

SBI Workflow

Creation of the dataset

$$\begin{tikzcd}
& D = \{(\overrightarrow{\theta_i}, \overrightarrow{x_i})\}_{i=1,\dots,n} \arrow[r, "\text{Summary}", "\text{statistics}"'] & D' = \{(\overrightarrow{\theta_i}, S(\overrightarrow{x_i}))\}_{i=1,\dots,n} \\
\text{SIMULATOR} \arrow[ur, "\text{Training set}"] \arrow[dr, "\text{Test set}"'] & & \\
& Q = \{\overrightarrow{x}\} \arrow[r, "\text{Summary}", "\text{statistics}"'] & Q' = \{S(\overrightarrow{x})\}
\end{tikzcd}$$

$$\begin{tikzcd}
& \text{Trained on } D' & \\
\text{SBI (N.N.)} \arrow[ur, "\text{Train}"] \arrow[dr, "\text{Test}"'] & & \\
& \text{Tested on } Q' \arrow[r, "\text{Output}"] & \text{Posterior Distribution } P(\theta|x)
\end{tikzcd}$$

What is the SNPE method?

Sequential Neural Posterior Estimation (SNPE) is a method within the framework of SBI.

Neural networks are employed to learn the relationship between parameters and the resulting data.

SNPE operates iteratively, refining the posterior estimate with each iteration.

SNPE

The Core idea of SNPE is to model directly the posterior probability p(θ|x)

SNPE iteratively refines the posterior distribution, improving the accuracy of the estimation with each iteration.

Different version of SNPE use progressively complex neural network methods

Base version of SNPE uses Mixture Density Network (MDN)

In Our Case, we used the most advanced version, **SNPE-C**.

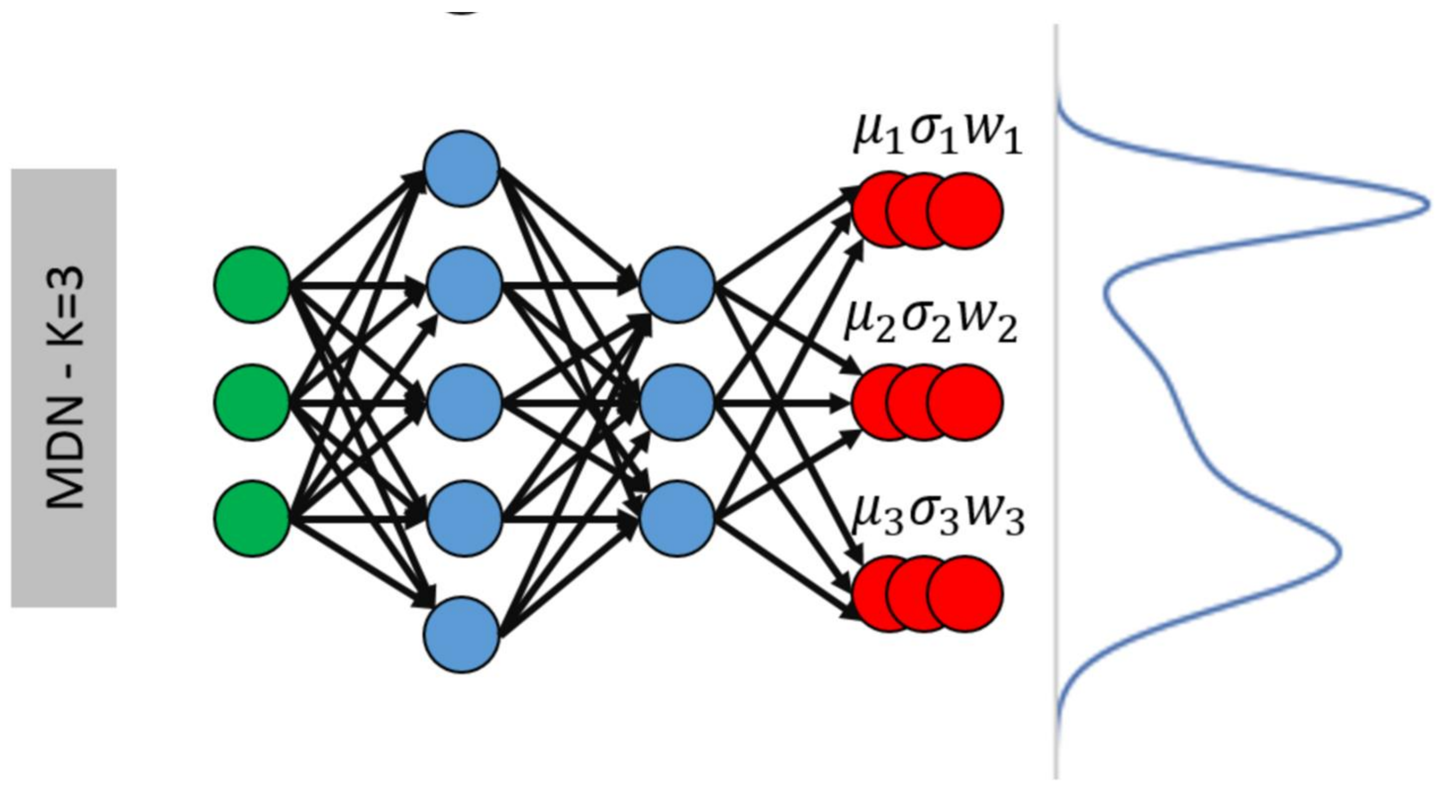
SNPE-C uses neural networks like Masked Autoregressive Flows (MAF) to model the conditional density p(θ|x)

SNPE - A

The objective is to model the posterior probability using MDN

The neural network outputs parameter of a gaussian mixture model

Trained minimizing negative log-likelihood



SNPE-C (MAF)

$$p(\boldsymbol{\theta} \mid \mathbf{x}) = \prod_i p(\theta_i \mid \theta_{1:i-1}, \mathbf{x})$$

$$\begin{aligned}
\theta_1^{(1)} &= \mu_1^{(1)}(x) + \sigma_1^{(1)}(x) \cdot z_1 \\
\theta_2^{(1)} &= \mu_2^{(1)}(\theta_1^{(1)}, x) + \sigma_2^{(1)}(\theta_1^{(1)}, x) \cdot z_2 \\
\theta_3^{(1)} &= \mu_3^{(1)}(\theta_{1:2}^{(1)}, x) + \sigma_3^{(1)}(\theta_{1:2}^{(1)}, x) \cdot z_3
\end{aligned}$$

$$p_\phi(\boldsymbol{\theta} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_0 \mid \mathbf{0}, \mathbf{I}) \left| \det\left(\frac{\partial f^{-1}}{\partial \boldsymbol{\theta}} \right) \right|$$

MAF uses an autoregressive model to parameterize the transformation. Each variable is modeled conditionally on the previous variables.

f_k are a series of autoregressive functions, essentially gaussian conditional, parametrized by a neural network that transforms a sample from a *trivial* distribution into a sample from a target probability

Integration of SDEs

When analytical solutions are intractable, numerical methods are employed to solve Stochastic Differential Equations.

Langevin Equation:

$$\cancel{m\ddot{x}_t} = -\gamma \dot{x}_t + F_{ext}(x_t) + \sqrt{2D}\xi$$

Overdamped, we can neglect the inertial force

Einstein Relation

$$D = \frac{k_B T}{\gamma}$$

$$dx = \frac{F_t}{\gamma} dt + \sqrt{2\tilde{D}} \underbrace{\xi_t^x dt}_{\sim \mathcal{N}(0, \sqrt{dt})}$$

Euler-Maruyama method:

$$\Rightarrow x_{t+dt} = x_t + \frac{F_t}{\gamma} dt + \sqrt{2Ddt}\underbrace{g}_{\sim \mathcal{N}(0,1)}$$

We focused on two stochastic models:

1. Quartic Potential Model

1. Red Blood Cells Model

1.Quartic potential model

$$\dot{x}_t = \mu(\overbrace{-U'(x_t) + f_t}^{F}) + \sqrt{2D}\xi_t^x$$

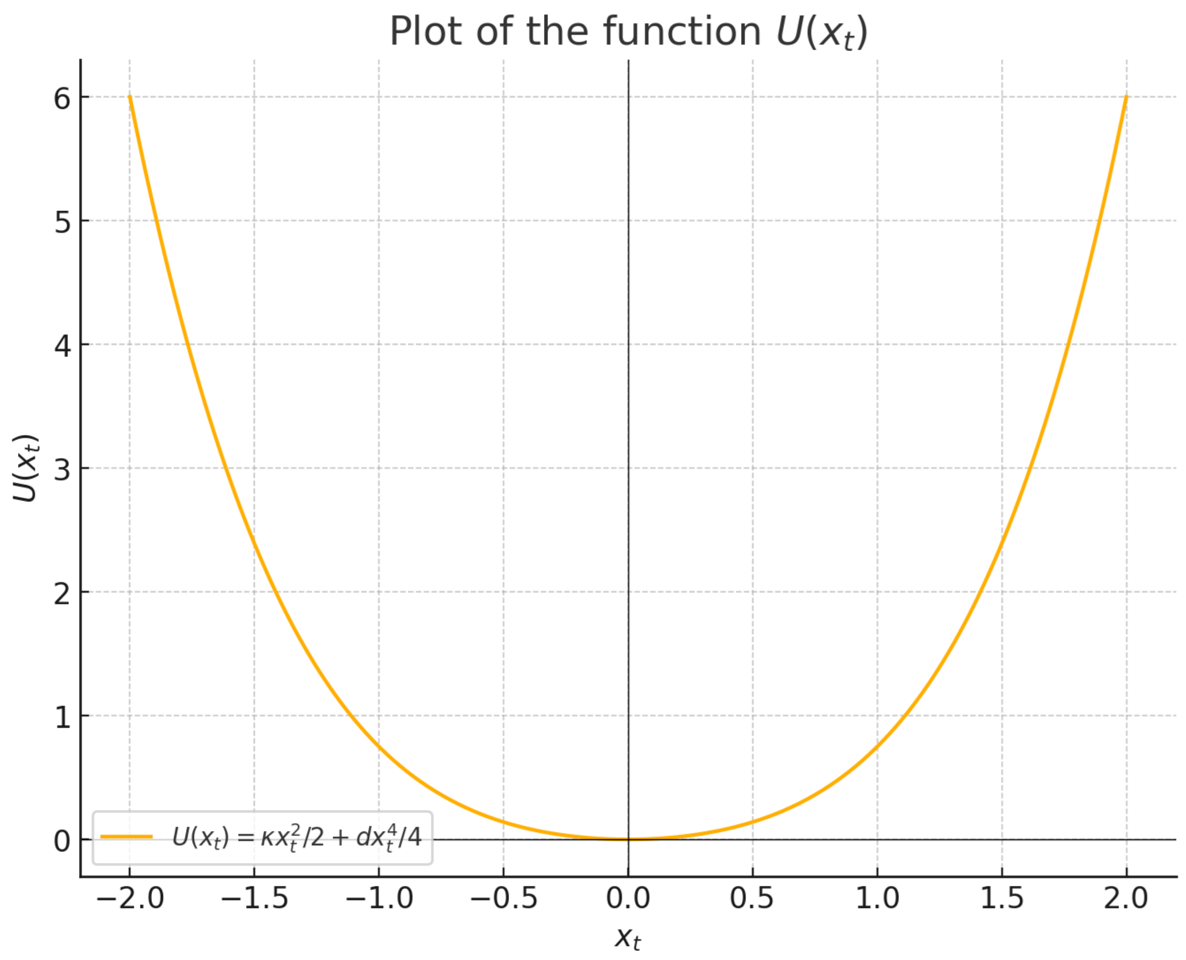
$$\dot{f}_t = f_t/\tau + \sqrt{2\varepsilon^2/\tau}\,\xi_t^x$$

Stochastic terms

The parameters we want to infer are:

θ = {d, ε, τ}

Fixing D= μ=k=1



$$U(x_t) = \kappa x_t^2/2 + dx_t^4/4$$

Entropy 1d

$$F = -kx - dx^3 + f_t$$

Average force for consecutive time steps:

$$Fs_t = \frac{F_t + F_{t-1}}{2}$$

Change in position of the particle:

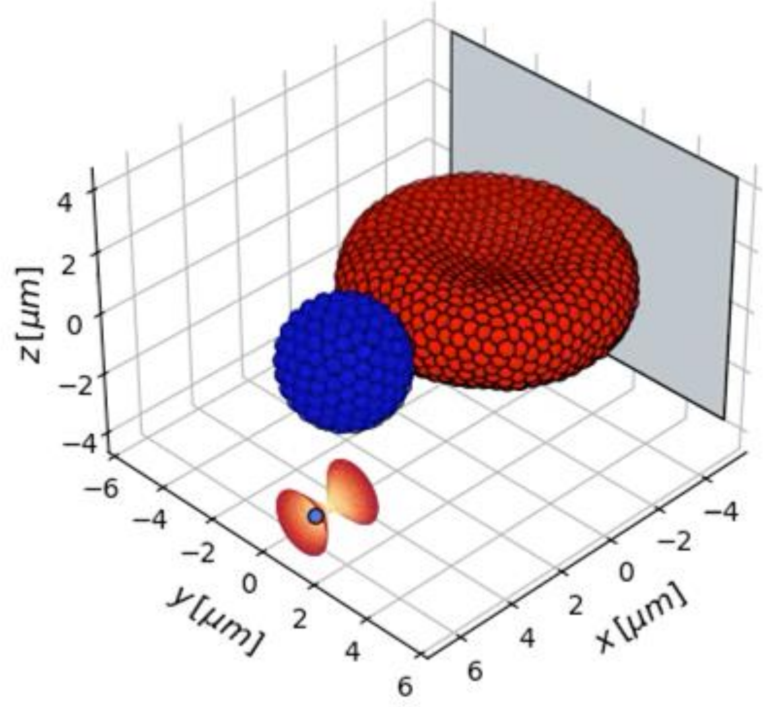
$$\delta x_t = x_t - x_{t-1}$$

We define the **entropy production rate** as:

$$\sigma = \frac{\sum_{t=1}^{T_{tot}} Fs_t \cdot \delta x_t}{D(T_{tot} - 1)}$$

2.Red Blood Cells Model

$$\dot{x}_t = \mu_x(-\kappa_x x_t + \kappa_{\text{int}}\, y_t) + \sqrt{2D_x}\,\xi_t^x$$
$$\dot{y}_t = \mu_y(-\kappa_y y_t + \kappa_{\text{int}}\, x_t) + \mu_y f_t + \sqrt{2D_y}\,\xi_t^y$$
$$\dot{f}_t = -f_t/\tau + \sqrt{2\epsilon^2/\tau}\,\xi_t^f$$



The parameters we want to infer are:

$\theta = \{k_{int}, \varepsilon, \tau\}$

Fixing $D_x=D_y=\mu_x=\mu_y=k_x=k_y=1$

Mathematical model for the movement of a red blood cells membrane.

Entropy 2d

For the second model, we can derive an analytical equation for the entropy, which we will then compare with the corresponding numerical estimate presented earlier.

$$\boxed{\sigma = \frac{\mu_y \epsilon^2}{(1+\kappa_y \mu_y \tau) - \kappa_{\text{int}}^2 \mu_x \mu_y \tau^2 / (1+\kappa_x \mu_x \tau)}}$$

Numerically:

$$\sigma = \frac{1}{T_{tot}-1} \left(\frac{\sum_{t=1}^{T_{tot}} Fs_t^x \cdot \delta x_t}{D_x} + \frac{\sum_{t=1}^{T_{tot}} Fs_t^y \cdot \delta y_t}{D_y} \right)$$

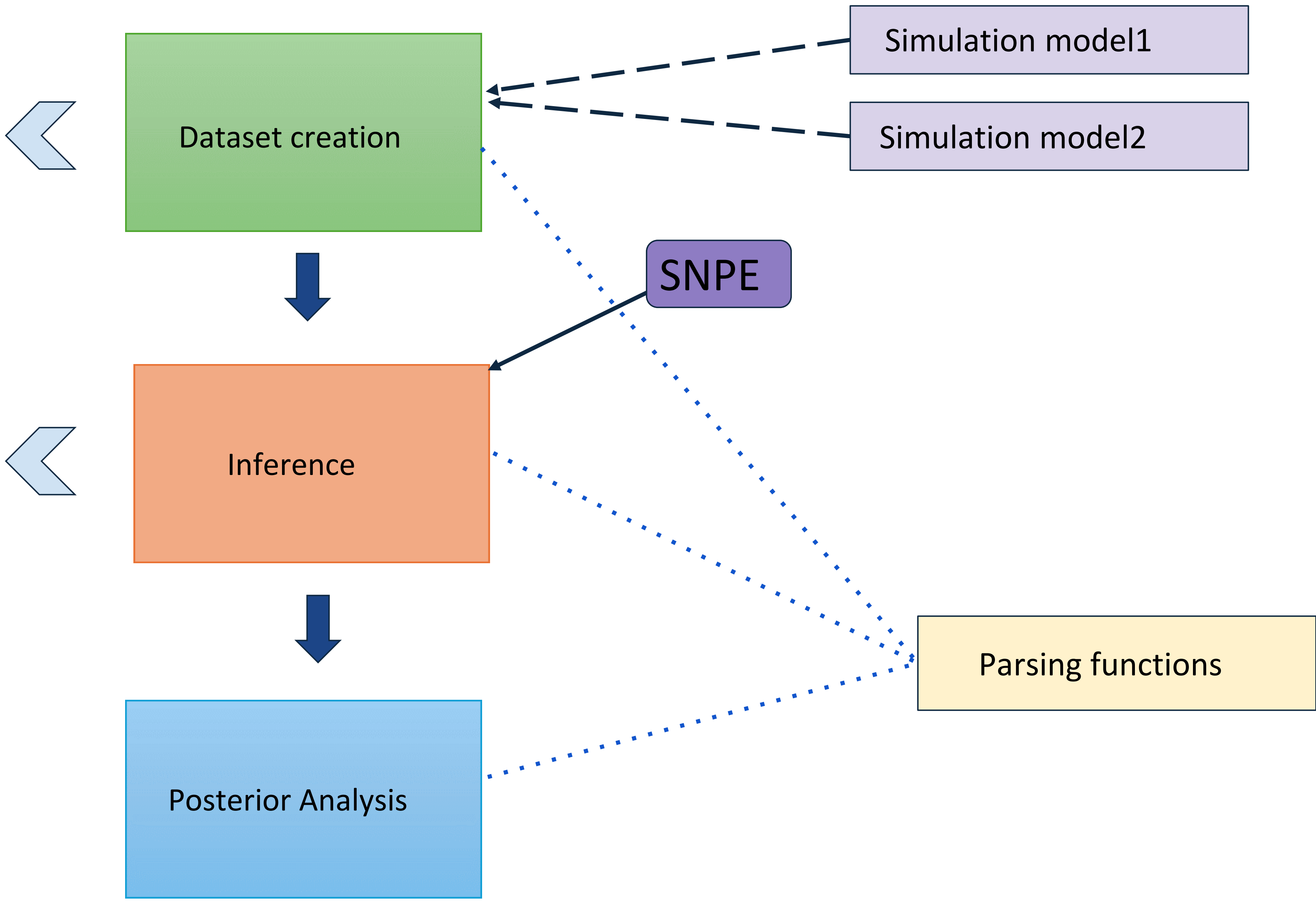
Importance Sampling for Entropy errors

- Single simulation that substitutes our observations
- Using the posterior to sample from that single observation (100)
- Calculating the log likelihood of the samples
- Filtering only above our threshold (0.65 of max logL)
- Simulating all the samples relative to the accepted logL
- Calculating Mean and Variance of the Entropy based on logL values.

Parallel processing

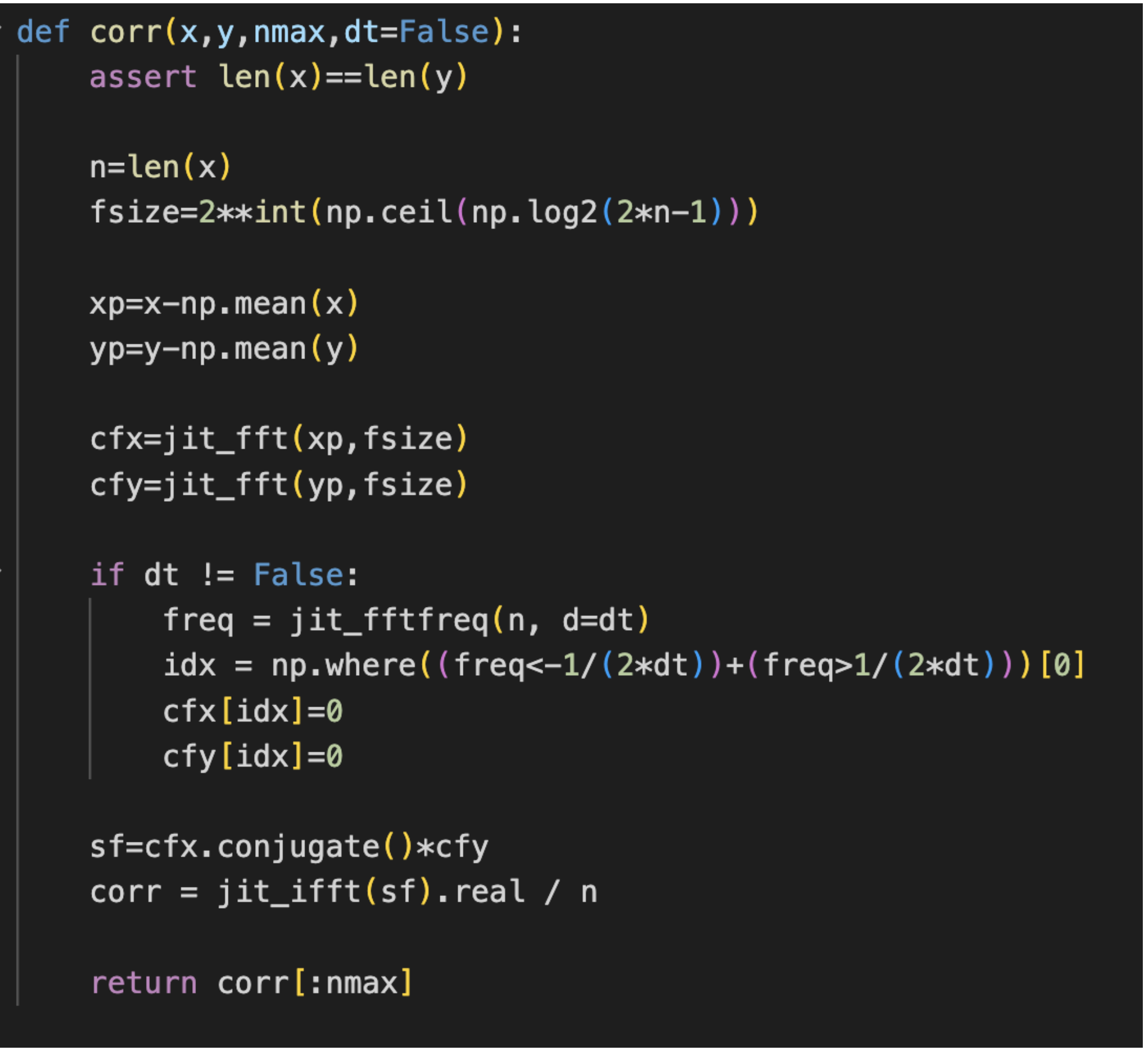
Saved simulations datasets ready for inference

Saved posterior datasets ready for analysis



Bonuses:
-hermite fit_model1
-posterior_heatmap_model2

Summary statistic



```python
def corr(x,y,nmax,dt=False):
    assert len(x)==len(y)

    n=len(x)
    fsize=2**int(np.ceil(np.log2(2*n-1)))

    xp=x-np.mean(x)
    yp=y-np.mean(y)

    cfx=jit_fft(xp,fsize)
    cfy=jit_fft(yp,fsize)

    if dt != False:
        freq = jit_fftfreq(n, d=dt)
        idx = np.where((freq<-1/(2*dt))+(freq>1/(2*dt)))[0]
        cfx[idx]=0
        cfy[idx]=0

    sf=cfx.conjugate()*cfy
    corr = jit_ifft(sf).real / n

    return corr[:nmax]
```

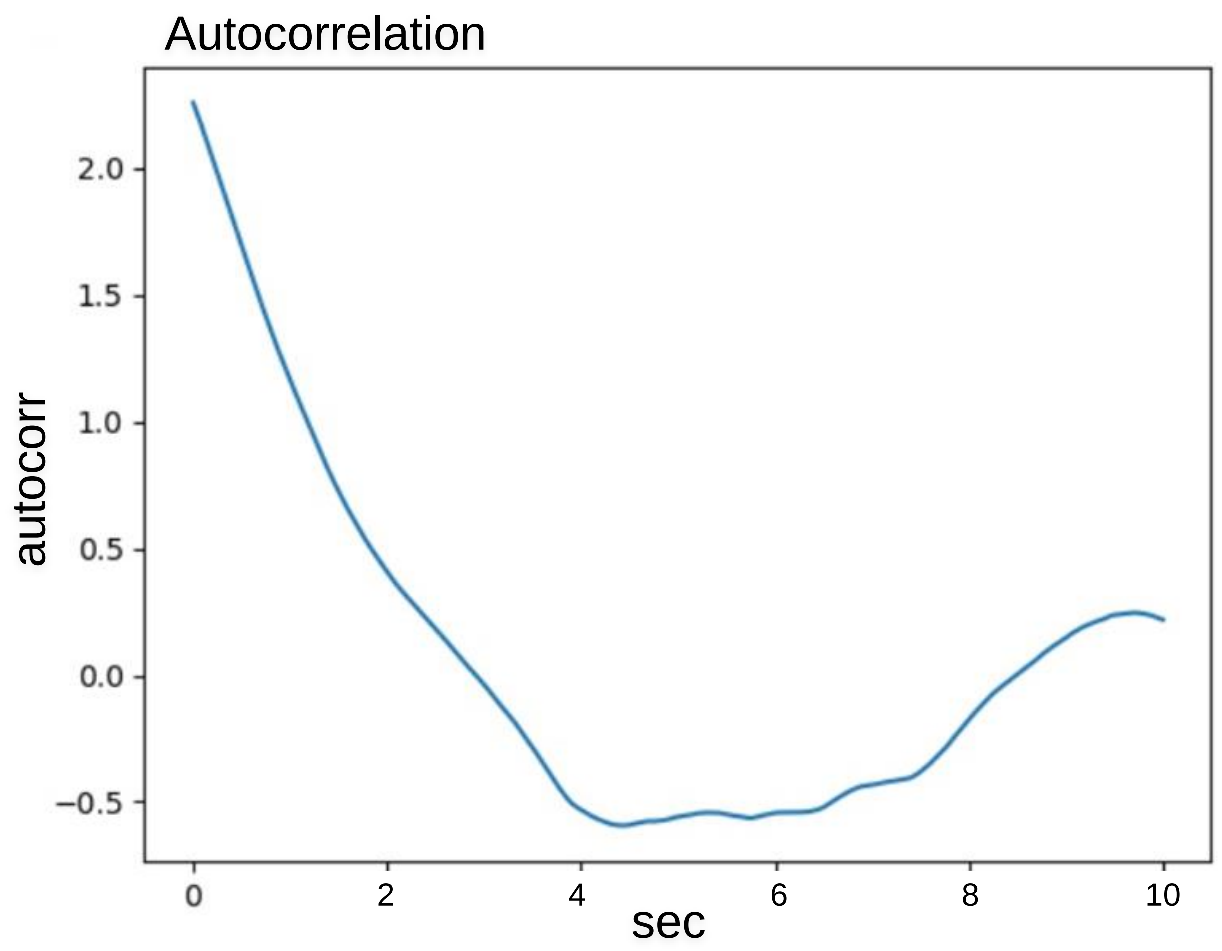
We want to obtain a suitable vector of feature for the SBI neural network that should be informative about the physical phenomenon

We mainly used a combination of **correlation function** and **sum variance rule**

Correlation function is computed through FFT

Nmax bounds the length of correlation vector

Autocorrelation Threshold



Threshold at 10s

Variance sum rule (VSR)

$$\mathcal{V}_{\Delta \boldsymbol{x}}(t) + \mathcal{V}_{\Sigma \boldsymbol{\mu F}}(t) = 2\boldsymbol{D}t + \boldsymbol{\mathcal{S}}(t)$$

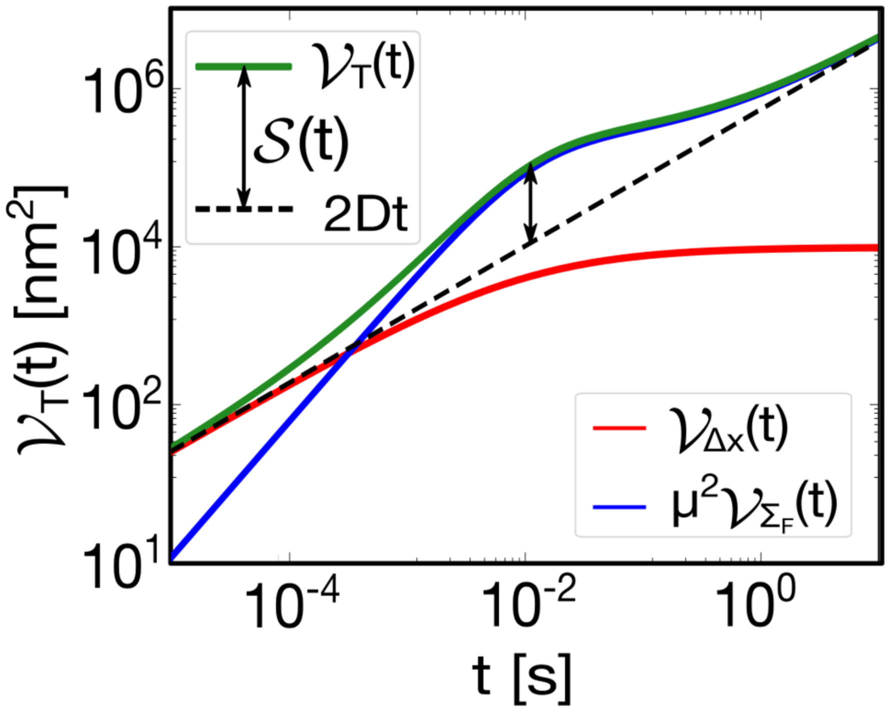
$$\mathrm{Cov}(\boldsymbol{x}_t - \boldsymbol{x}_0, \boldsymbol{x}_t - \boldsymbol{x}_0)$$

$$\mathrm{Cov}\left(\int_0^t \mathrm{d}\,t' \boldsymbol{\mu F}_{t'}, \int_0^t \mathrm{d}\,t' \boldsymbol{\mu F}_{t'}\right)$$

Excess variance:

$$4\int_0^t \mathrm{d}\,t' \int_0^{t'} \mathrm{d}\,t'' \left(\mathrm{Cov}(\dot{\boldsymbol{x}}_{t''}, \boldsymbol{\nu}_0)\right)_S$$

$$\boldsymbol{\nu}_t = \boldsymbol{\mu F}_t + \boldsymbol{D}\boldsymbol{\nabla}\ln(p(\boldsymbol{x}_t))$$



Variance Sum Rule for Entropy Production, I. Di Terlizzi et al. 2024 Science 383 971

HERMITE approximation up to the 12th order

1. The standard deviation σ_x of $x(t)$;
2. The averages $\langle \phi_i(x/\sigma_x)/\sqrt{\sigma_x} \rangle$ of Hermite functions $\phi_i(x)$, defined below, for $i = 0, 2, 4$;
3. The average, the standard deviation, and the mode of the normalized power-spectrum of $x(t)$;
4. The Hermite-function modes a_i of the autocorrelation function $\langle x(0)x(t) \rangle = \sum_i a_i \sqrt{\bar{f}}\, \phi_i\left(\bar{f}t\right)$, in which $\bar{f}$ is the average of the normalized power-spectrum of $x(t)$, for $i = 0, 2, 4, ..., 12$.

$$\phi_i(z) = \mathrm{e}^{-z^2/2} H_i(z) \left(2^i i! \sqrt{\pi}\right)^{-1/2}$$

Choosing of prior and bounds

Bounds are chosen in respect of the dataset size plugged into the inference:

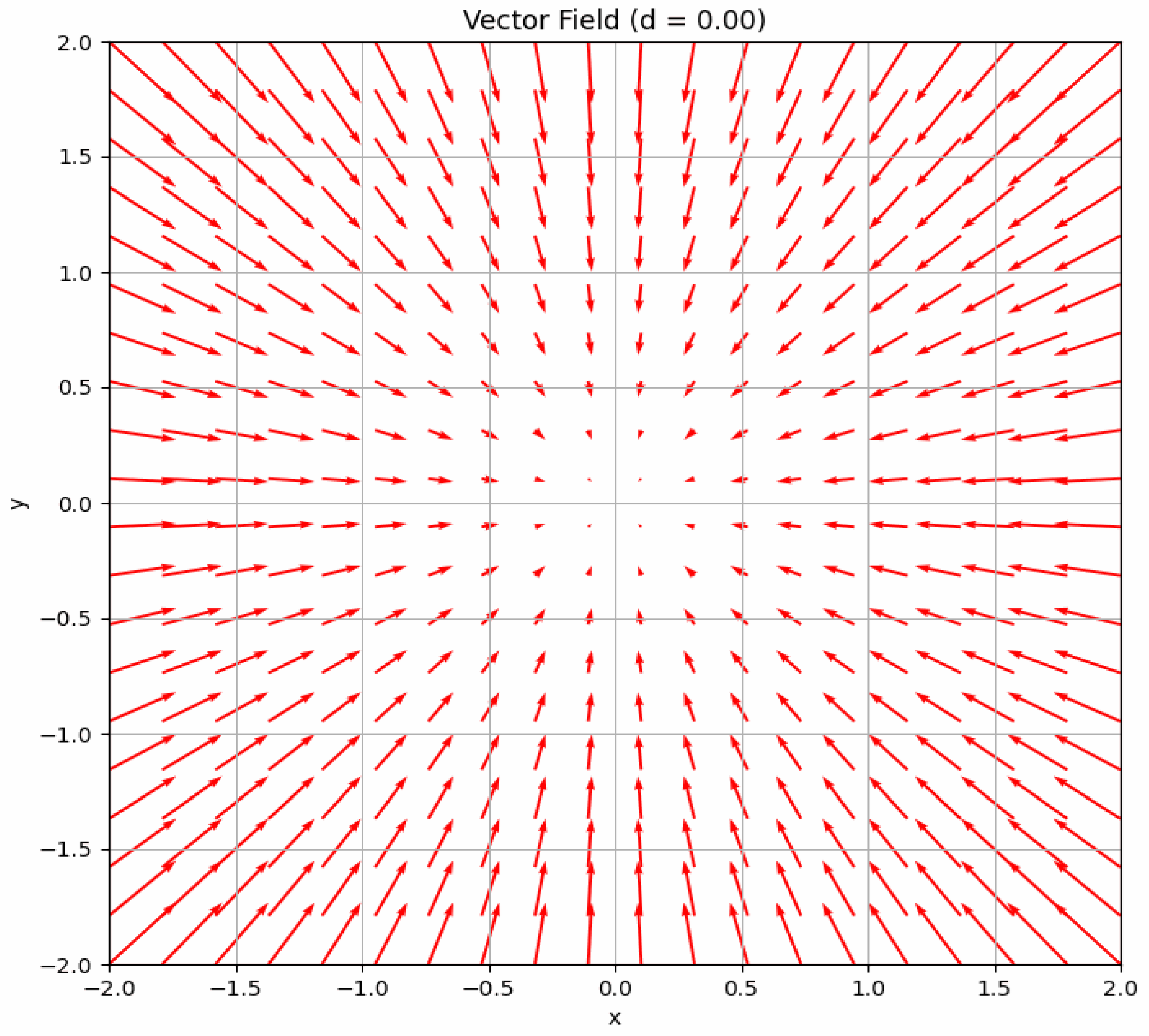
- range of d parameter extended as much as possible (linear dependence to entropy)
- range of ε,τ with similar bounds to have a factor of 1 as it appears as $\frac{\epsilon^2}{\tau} = D2$ simulation model

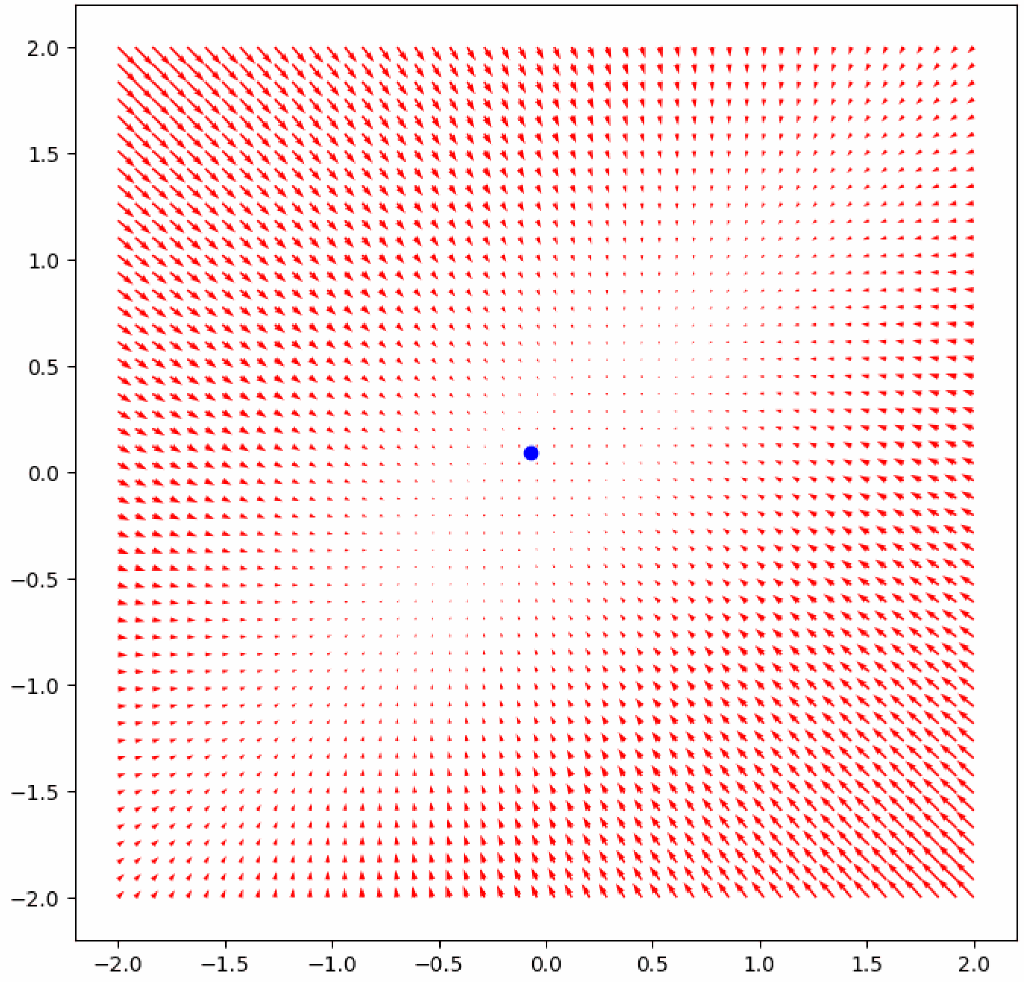
Model 1 $\longrightarrow$ ε ∈ [0,4] τ ∈ [0.1,3] d ∈ [0,15]

Model 2 $\longrightarrow$ ε ∈ [0,2] τ ∈ [0.1,2] d ∈ [0,1]

Note: we had to restrict d interval to have convergence in our potential and have a not divergent time series to work with.

Vector Field





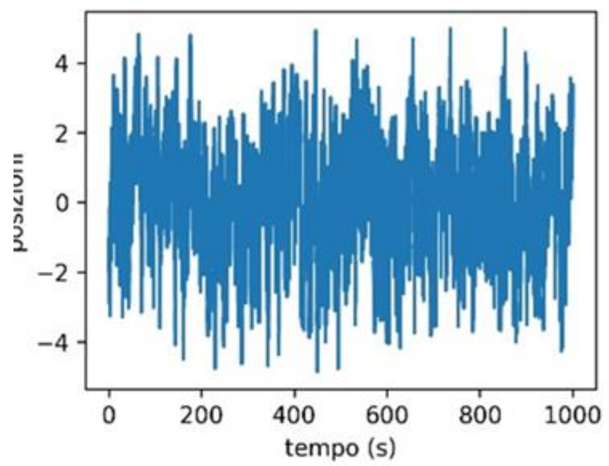
Choosing the simulation parameters

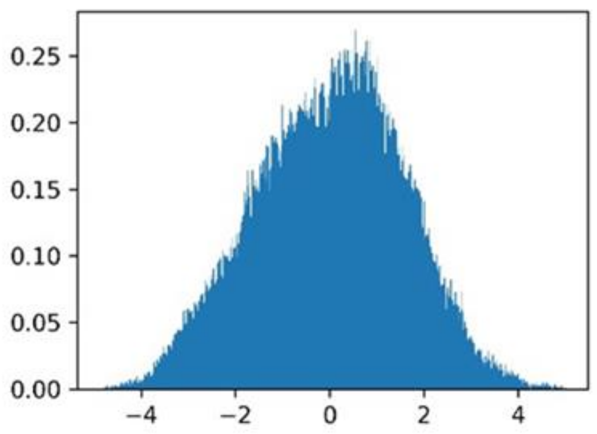
For our parameters we settled on these values after many considerations:

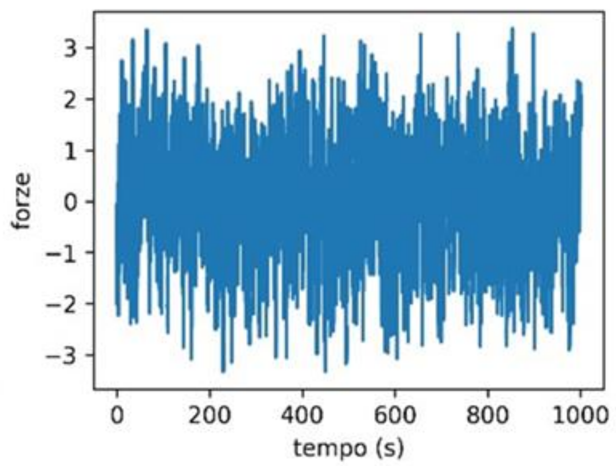
- dt = 1e-2
- oversampling = 5
- prerun = 1e3
- Npts = 5e4
- Num_simulations = 10000

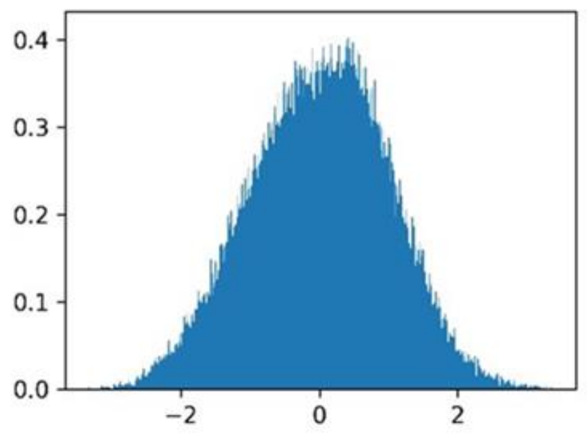
We chose these values to have a good number of peaks in both models through the time series, while maintaining a reasonable precision and overall dataset size

Trace



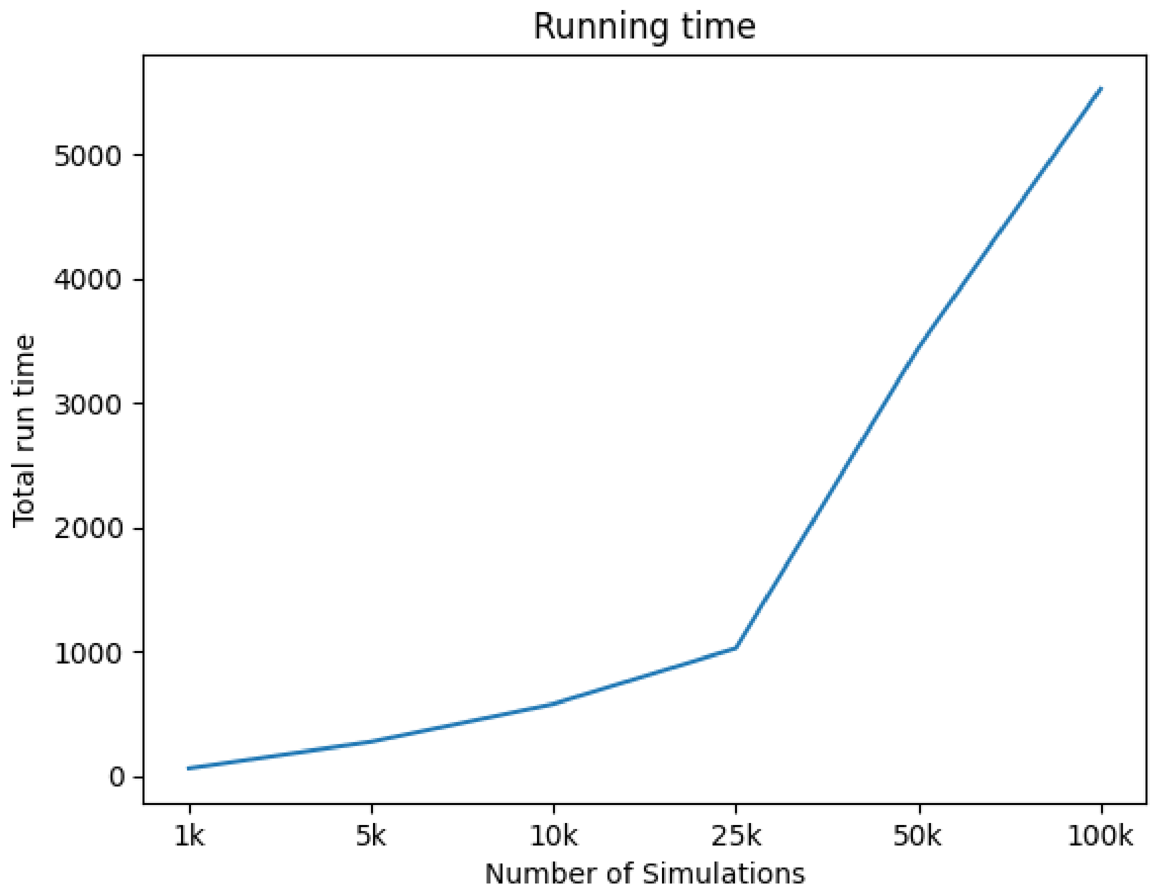






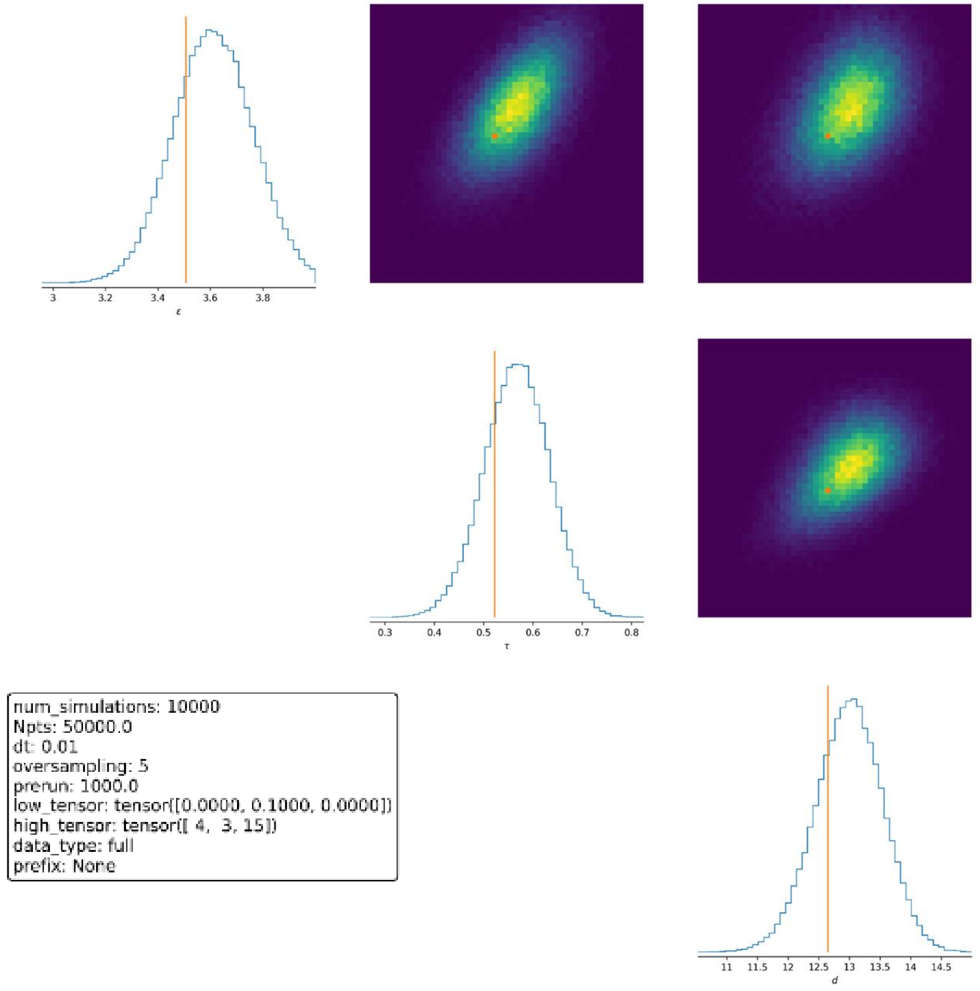
Optimization and Running Time

- Numba, rocket-fft, icc_rt
- Parallelization (dataset_creation and compute_entropies)
- numpy.empty
- Run times

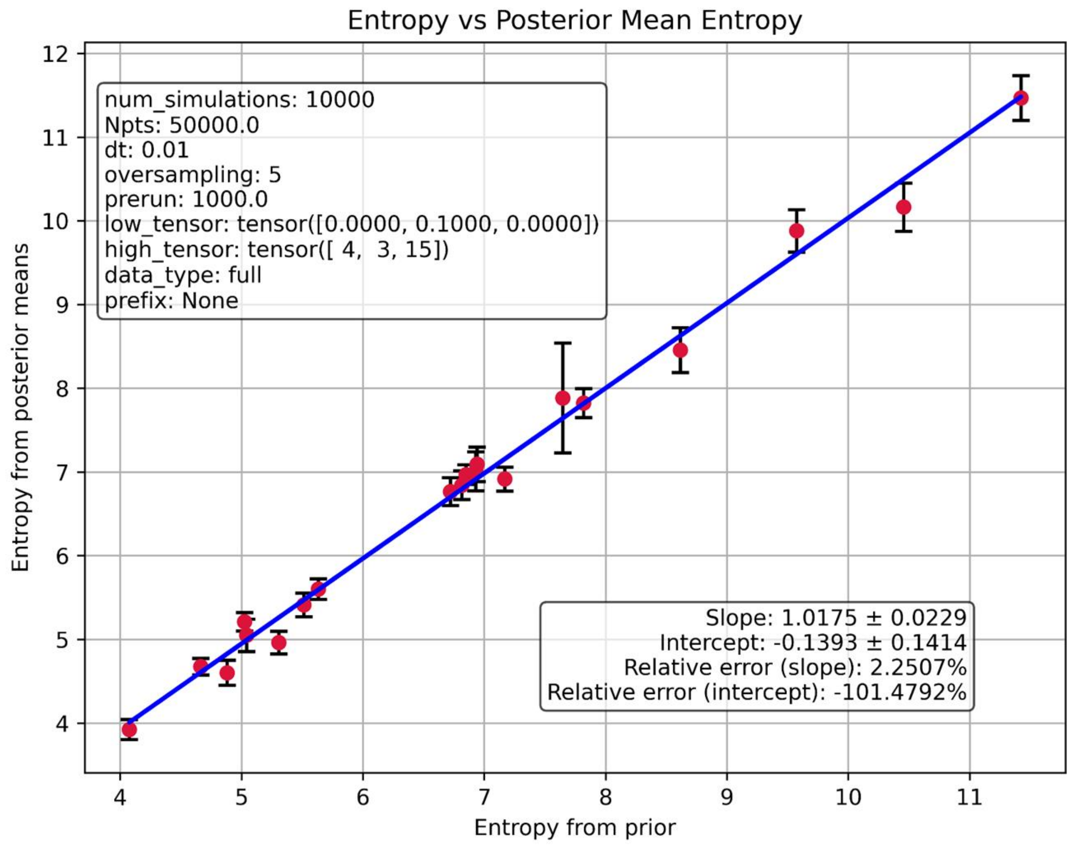


RESULTS

Quartic potential model



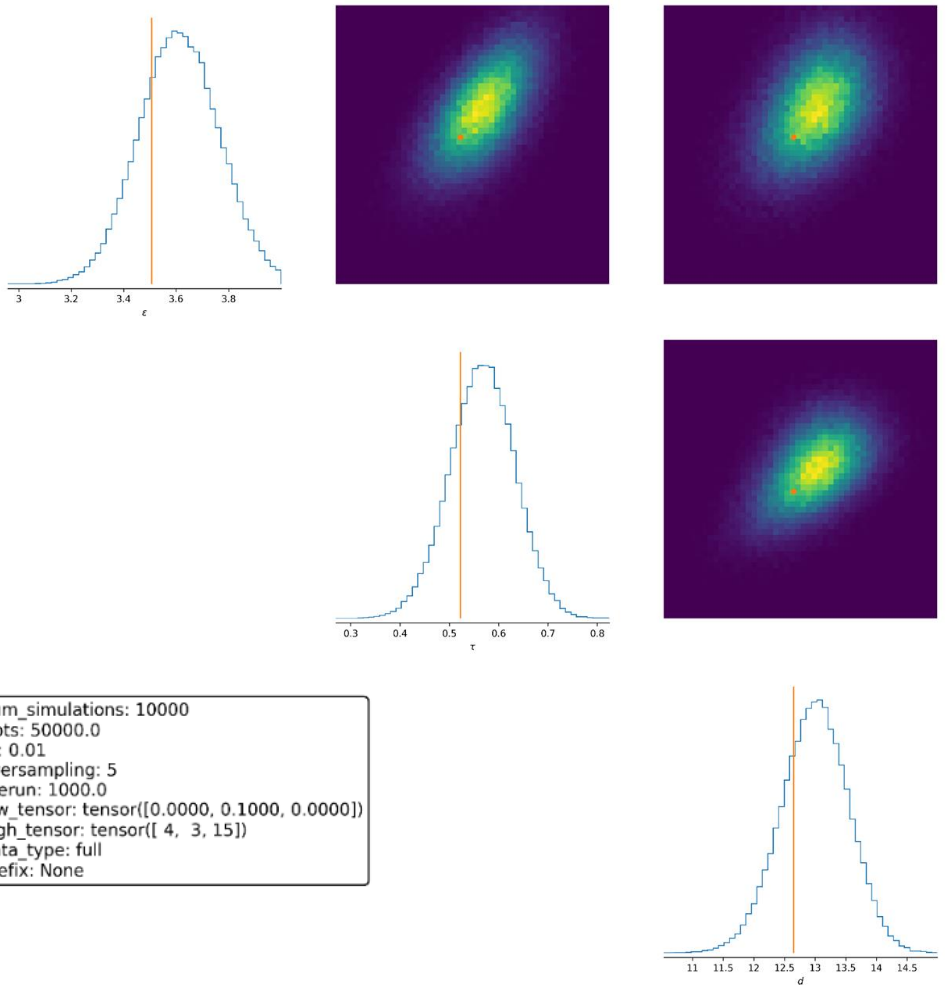
```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.1000, 0.0000])
high_tensor: tensor([ 4,  3, 15])
data_type: full
prefix: None
```



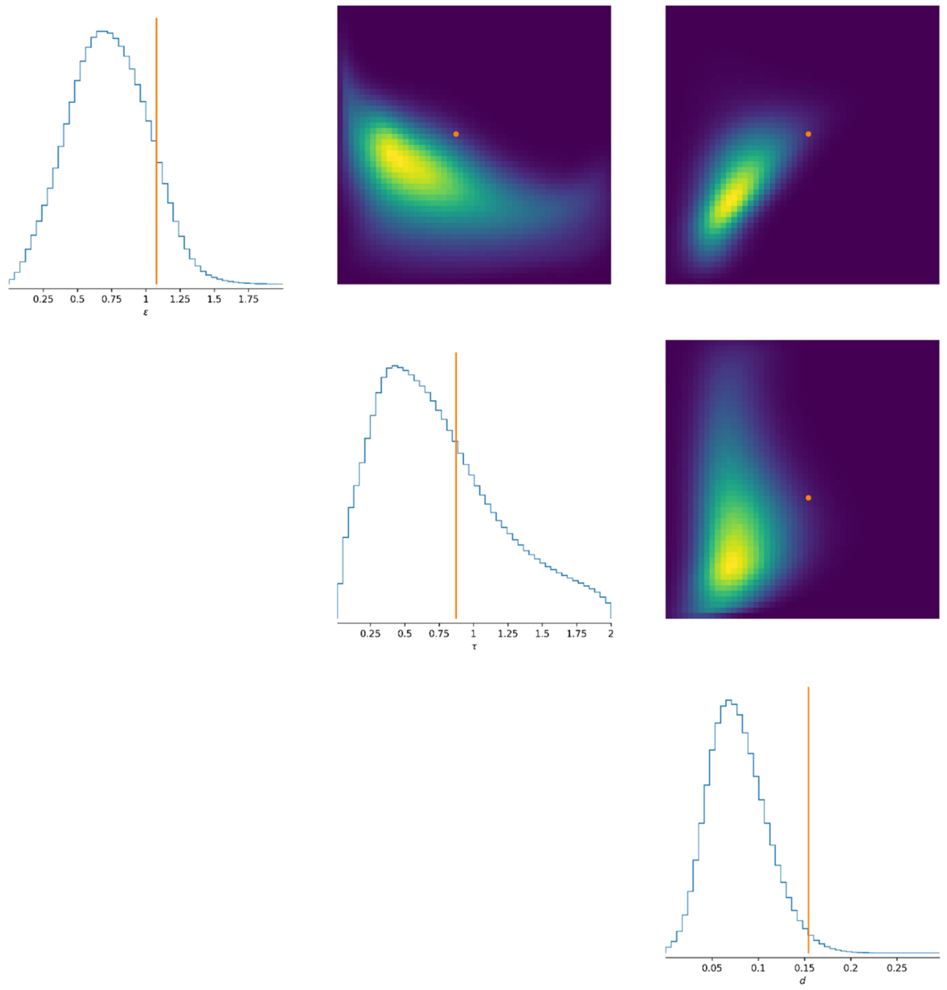
```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.1000, 0.0000])
high_tensor: tensor([ 4,  3, 15])
data_type: full
prefix: None
```

Slope: 1.0175 ± 0.0229
Intercept: -0.1393 ± 0.1414
Relative error (slope): 2.2507%
Relative error (intercept): -101.4792%

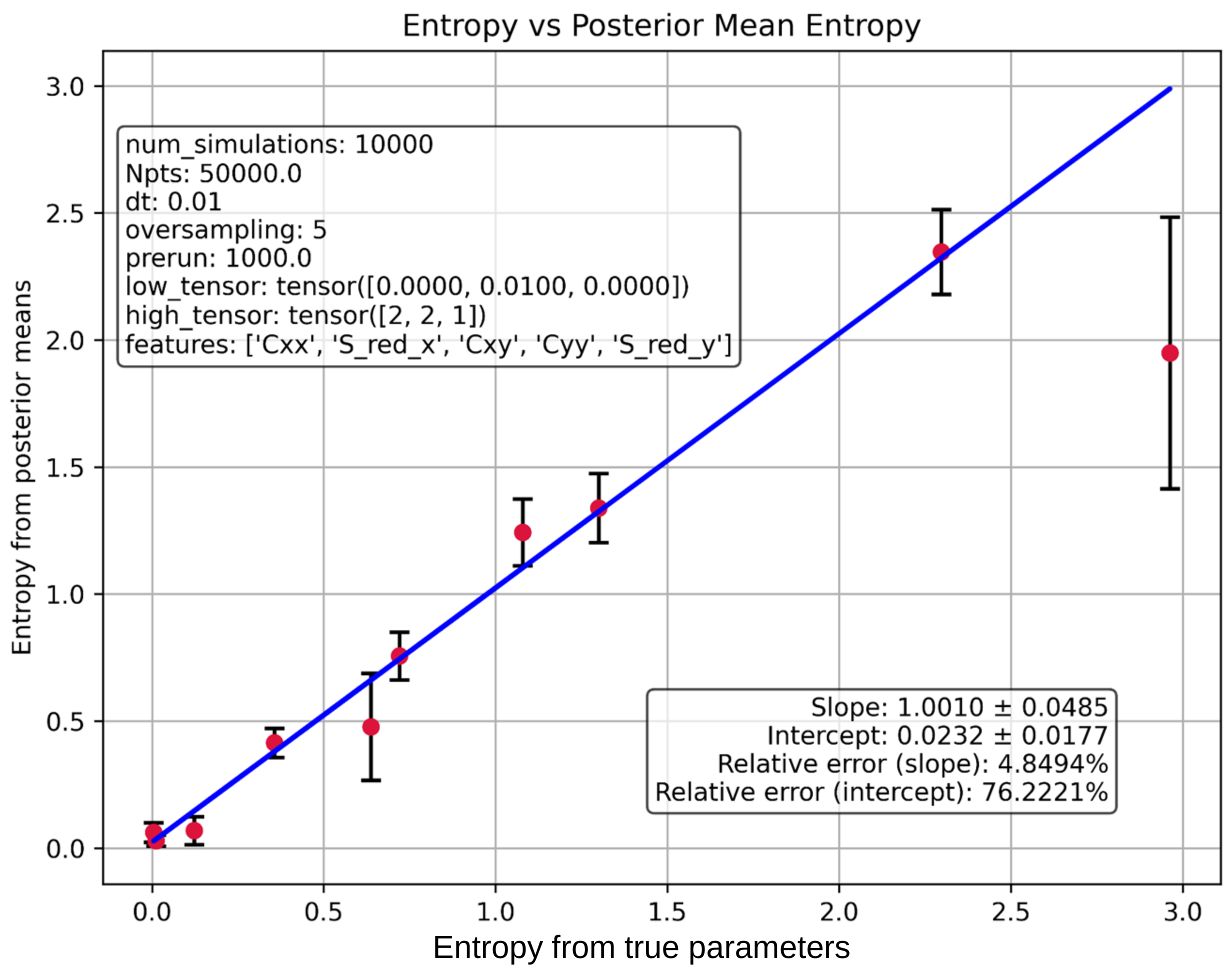
Quartic potential model using Hermite summary statistics



```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.1000, 0.0000])
high_tensor: tensor([ 4,  3, 15])
data_type: full
prefix: None
```

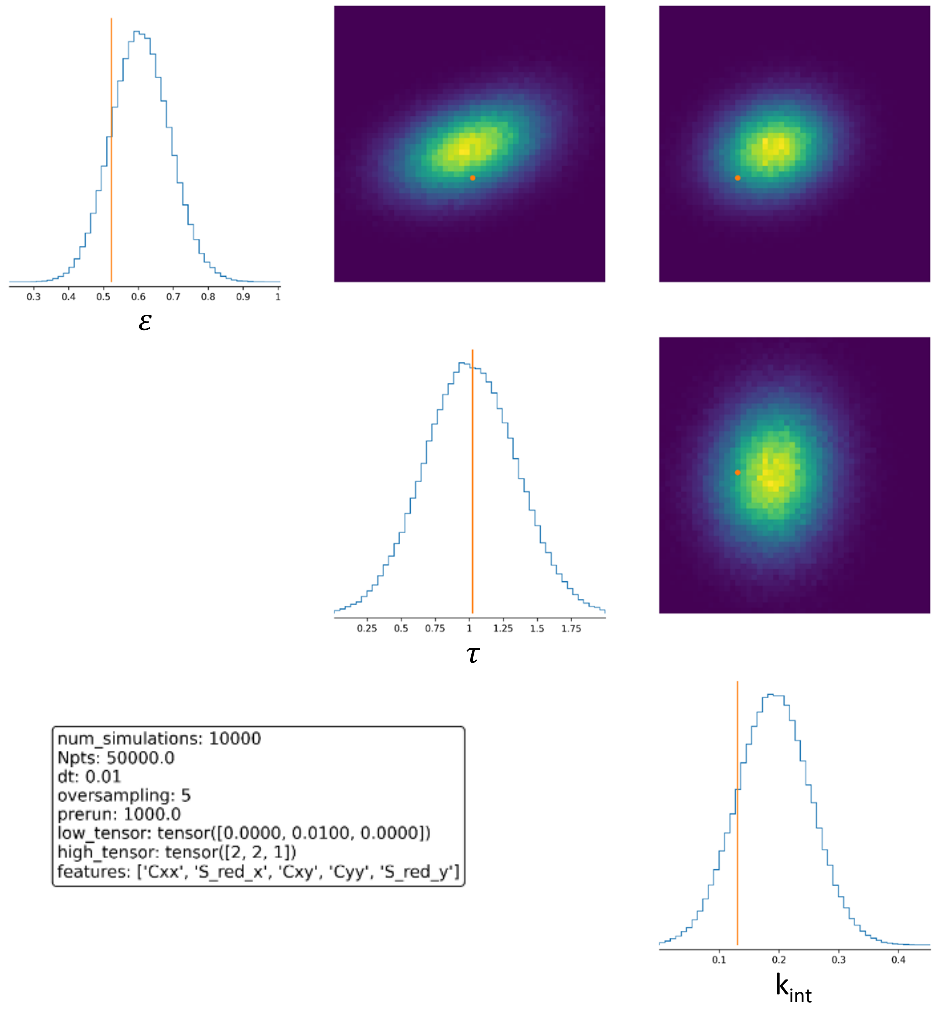


Red Blood Cells model using information on both x and y



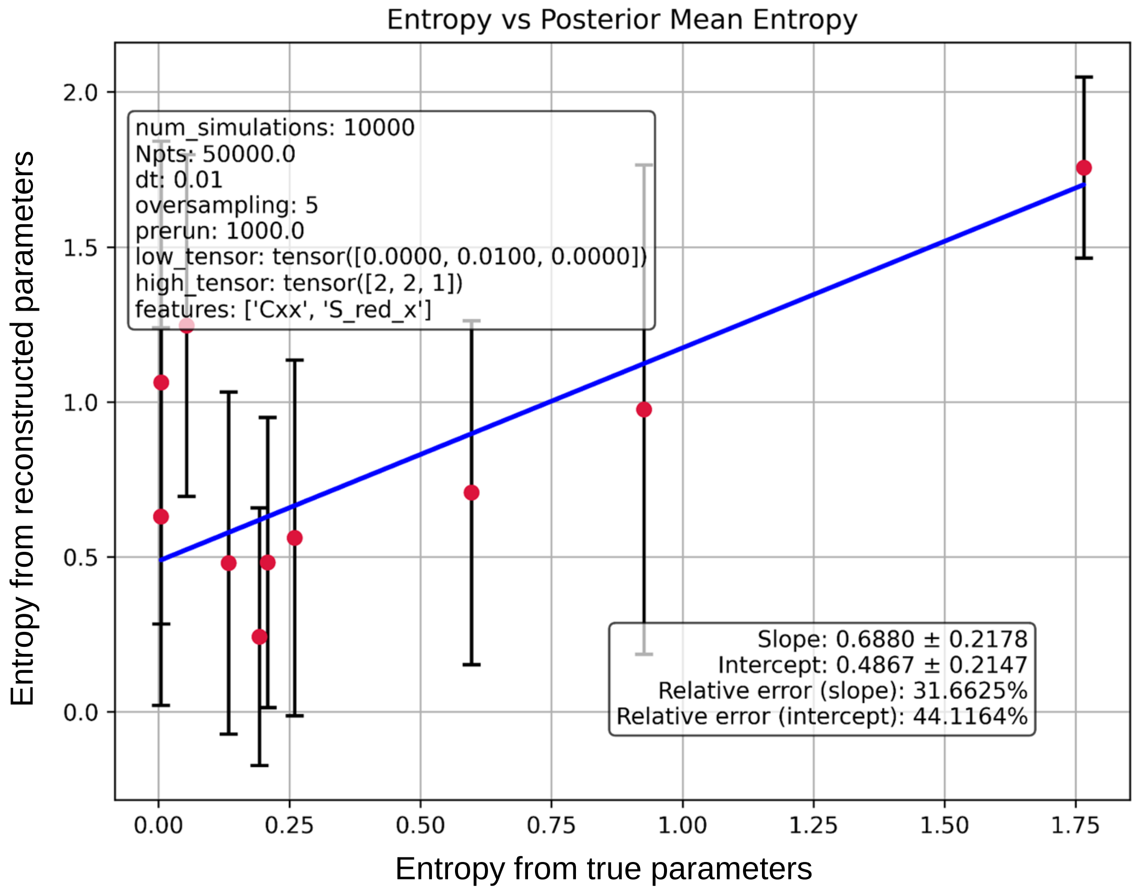
```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x', 'Cxy', 'Cyy', 'S_red_y']
```

Slope: 1.0010 ± 0.0485
Intercept: 0.0232 ± 0.0177
Relative error (slope): 4.8494%
Relative error (intercept): 76.2221%



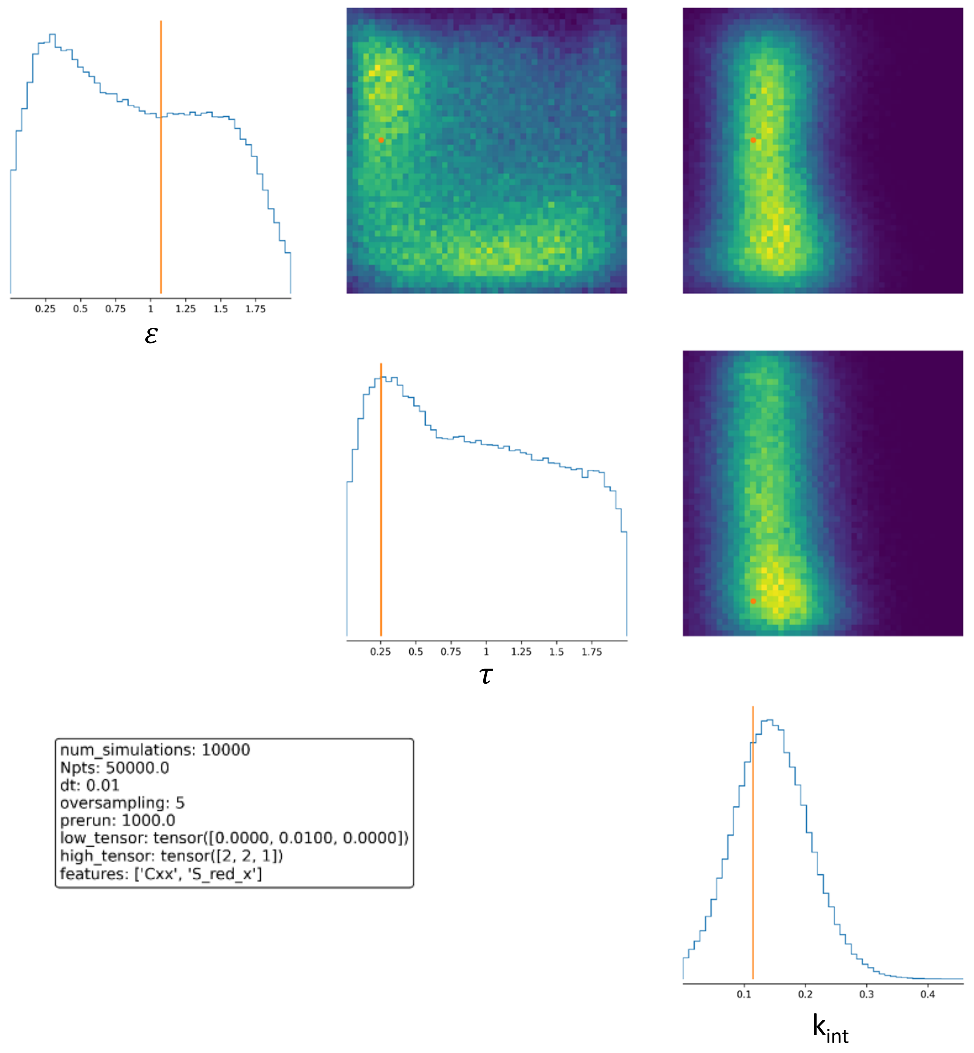
```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x', 'Cxy', 'Cyy', 'S_red_y']
```

Red Blood Cells model using information on x



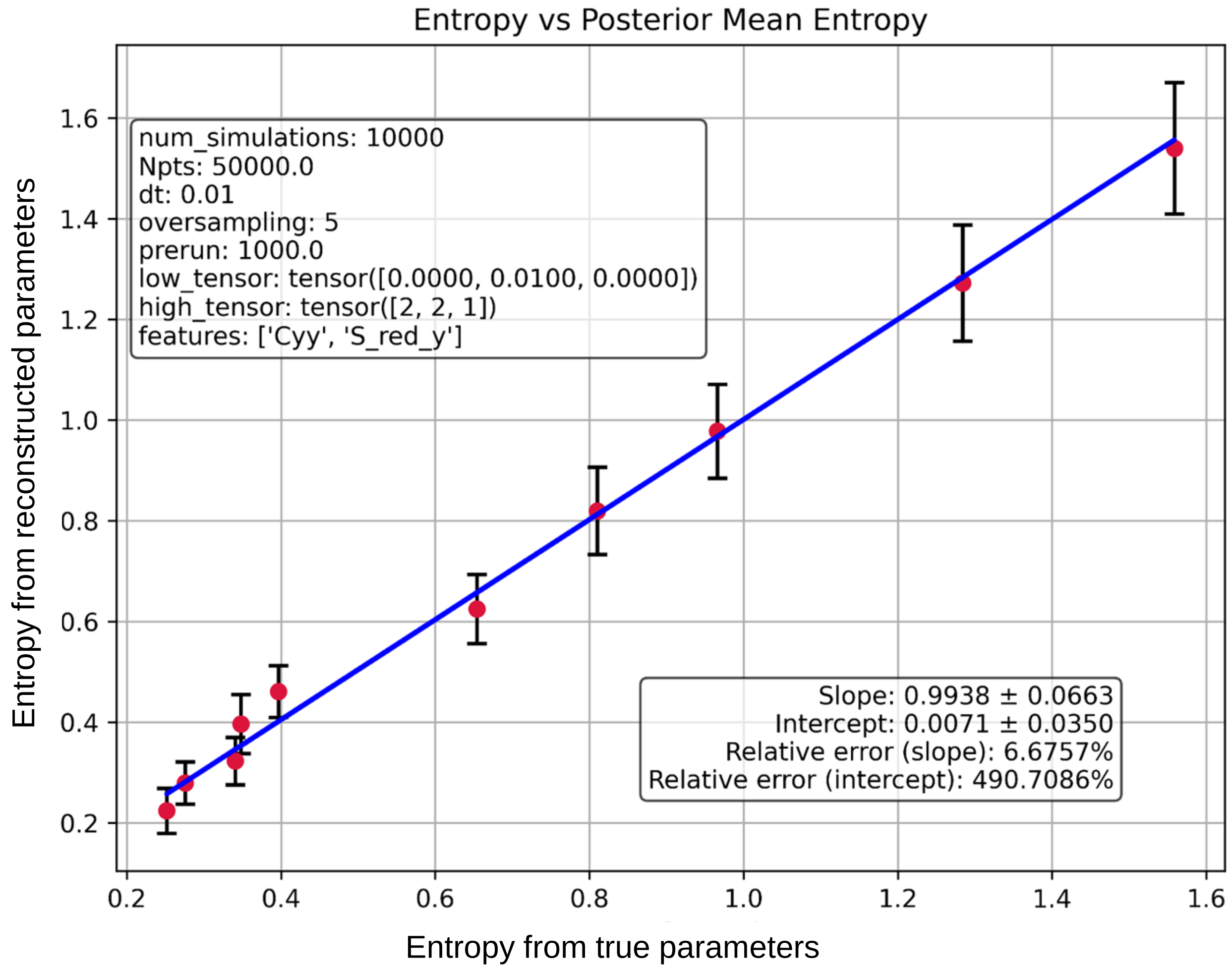
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x']

Slope: 0.6880 ± 0.2178
Intercept: 0.4867 ± 0.2147
Relative error (slope): 31.6625%
Relative error (intercept): 44.1164%



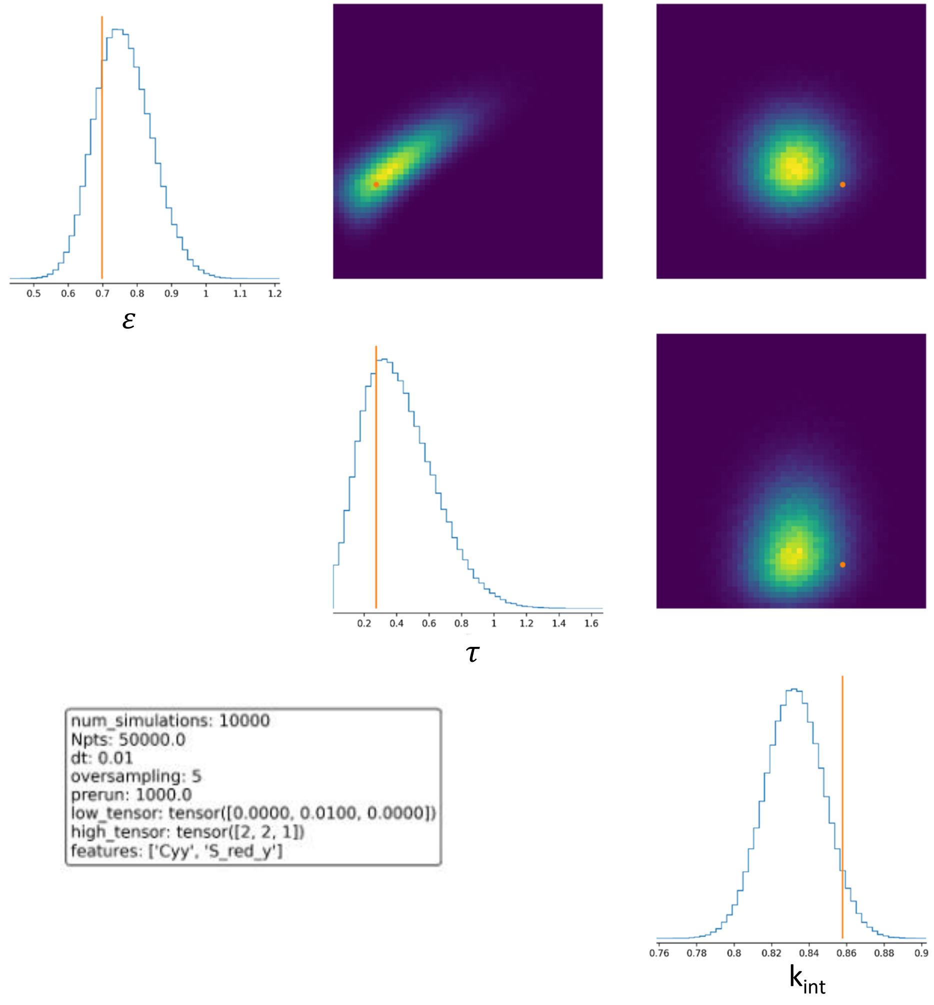
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x']

Red Blood Cells model using information on y



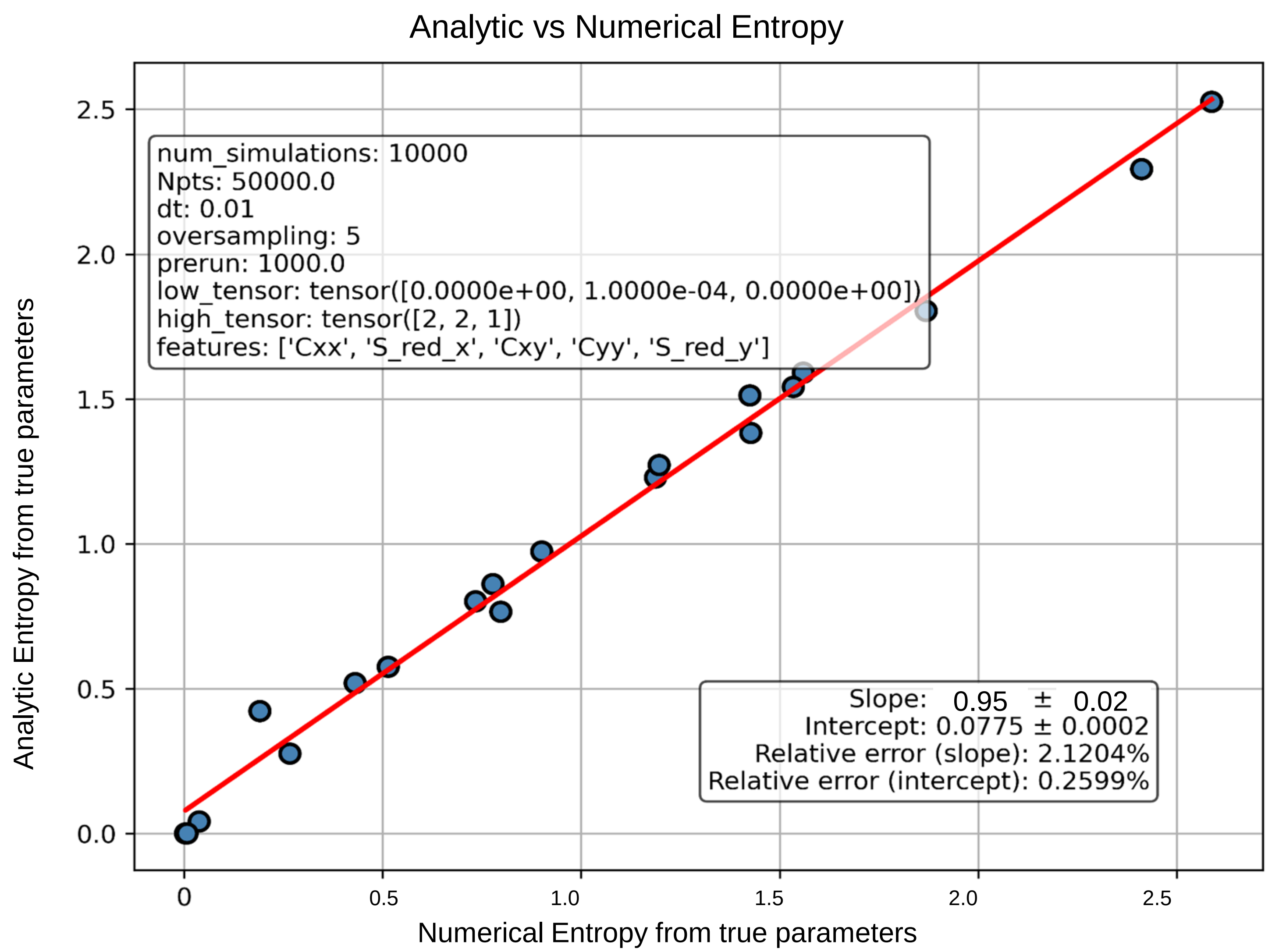
```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cyy', 'S_red_y']
```

```
Slope: 0.9938 ± 0.0663
Intercept: 0.0071 ± 0.0350
Relative error (slope): 6.6757%
Relative error (intercept): 490.7086%
```



```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.0100, 0.0000])
high_tensor: tensor([2, 2, 1])
features: ['Cyy', 'S_red_y']
```

Comparison between Analytical and Numerical Entropy with x and y

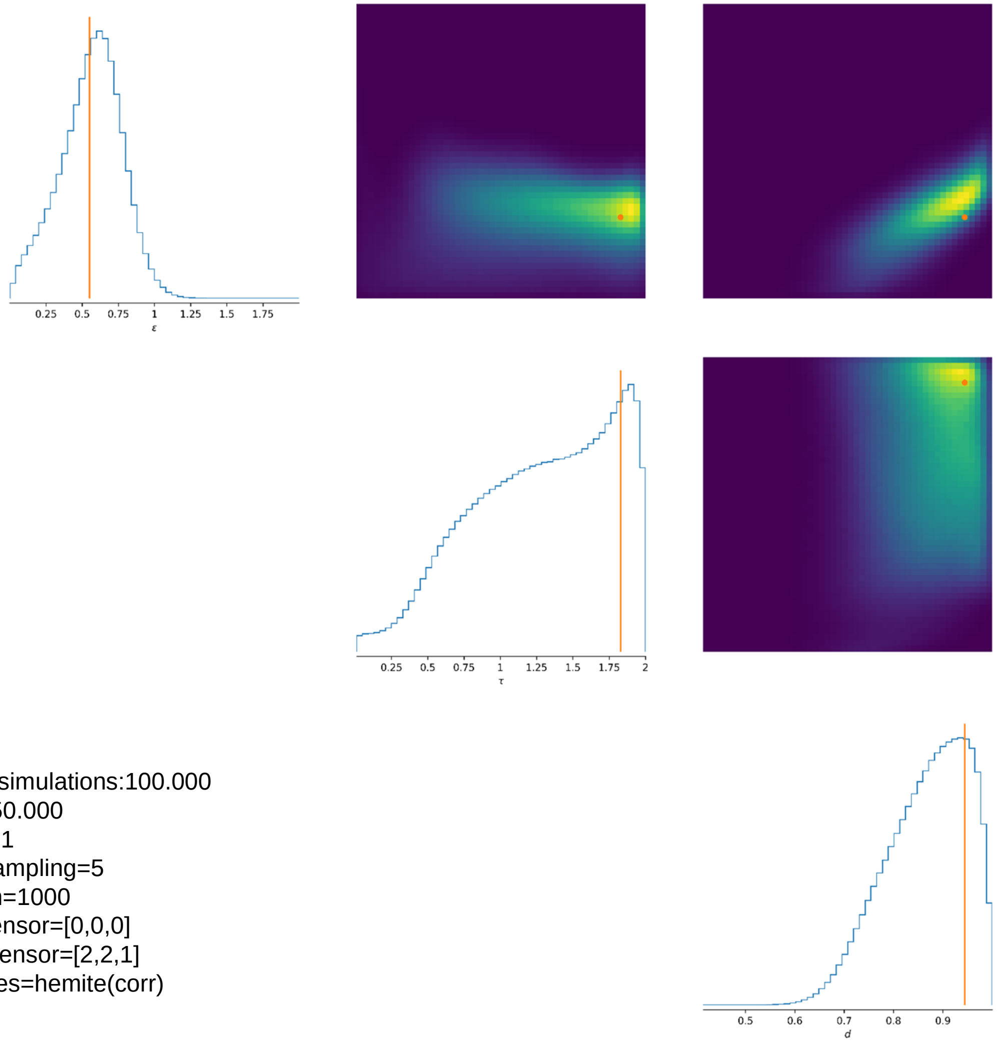


```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000e+00, 1.0000e-04, 0.0000e+00])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x', 'Cxy', 'Cyy', 'S_red_y']
```

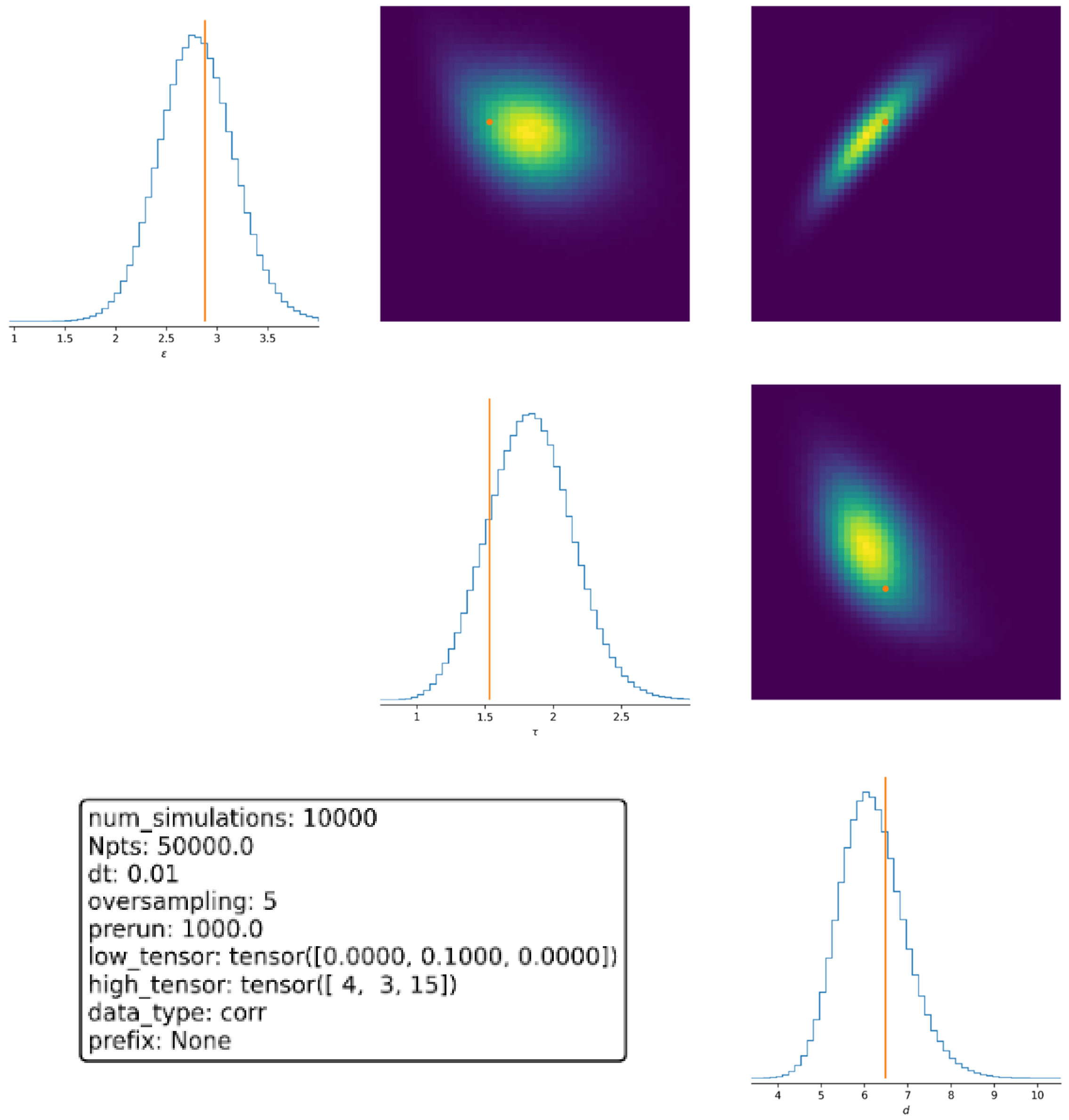
```
Slope:  0.95  ±  0.02
Intercept: 0.0775 ± 0.0002
Relative error (slope): 2.1204%
Relative error (intercept): 0.2599%
```

Backup slides

Hermite vs non hermite (just correlation)

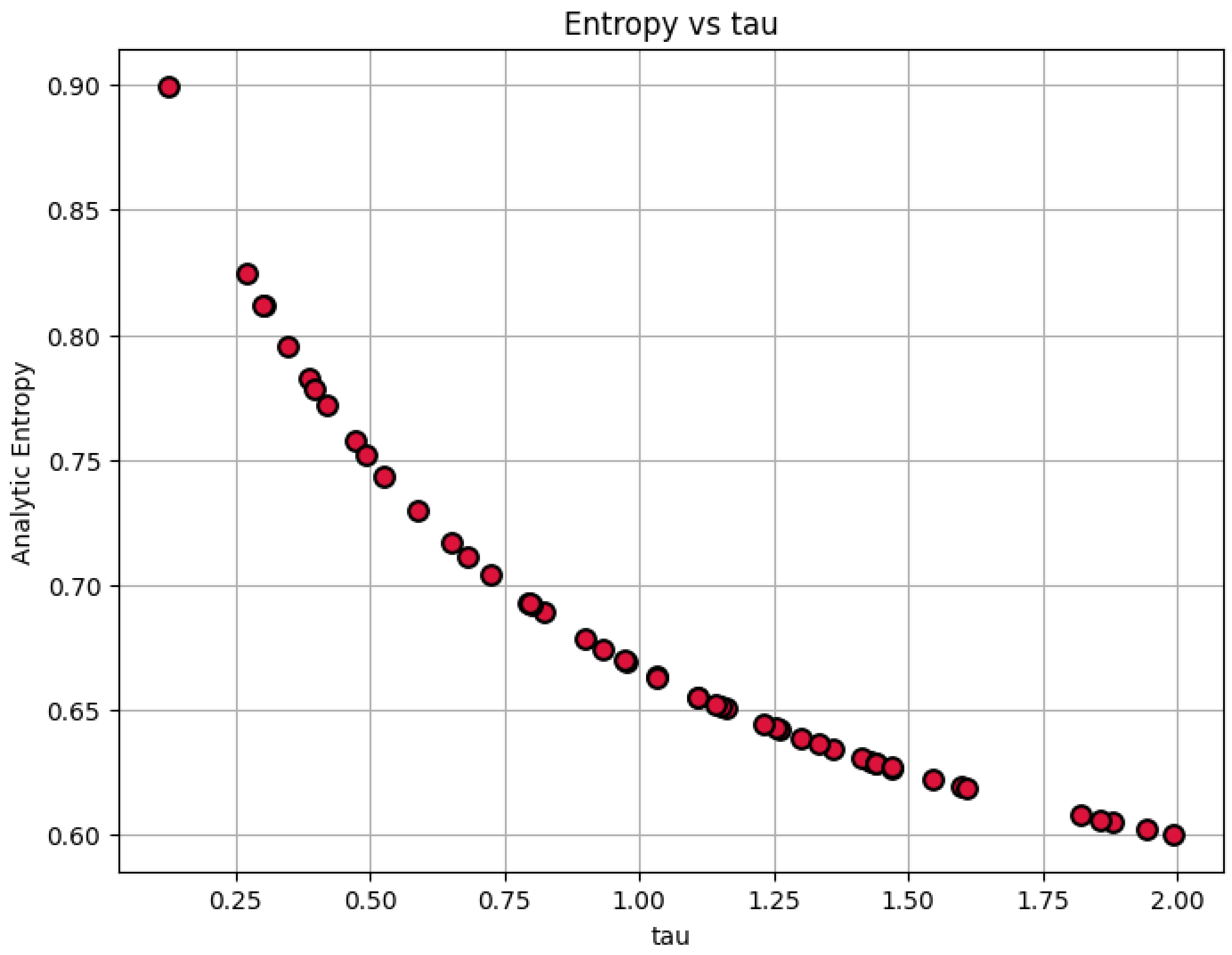


Num_simulations:100.000
Npts:50.000
dt=0.01
oversampling=5
prerun=1000
low_tensor=[0,0,0]
high_tensor=[2,2,1]
features=hemite(corr)

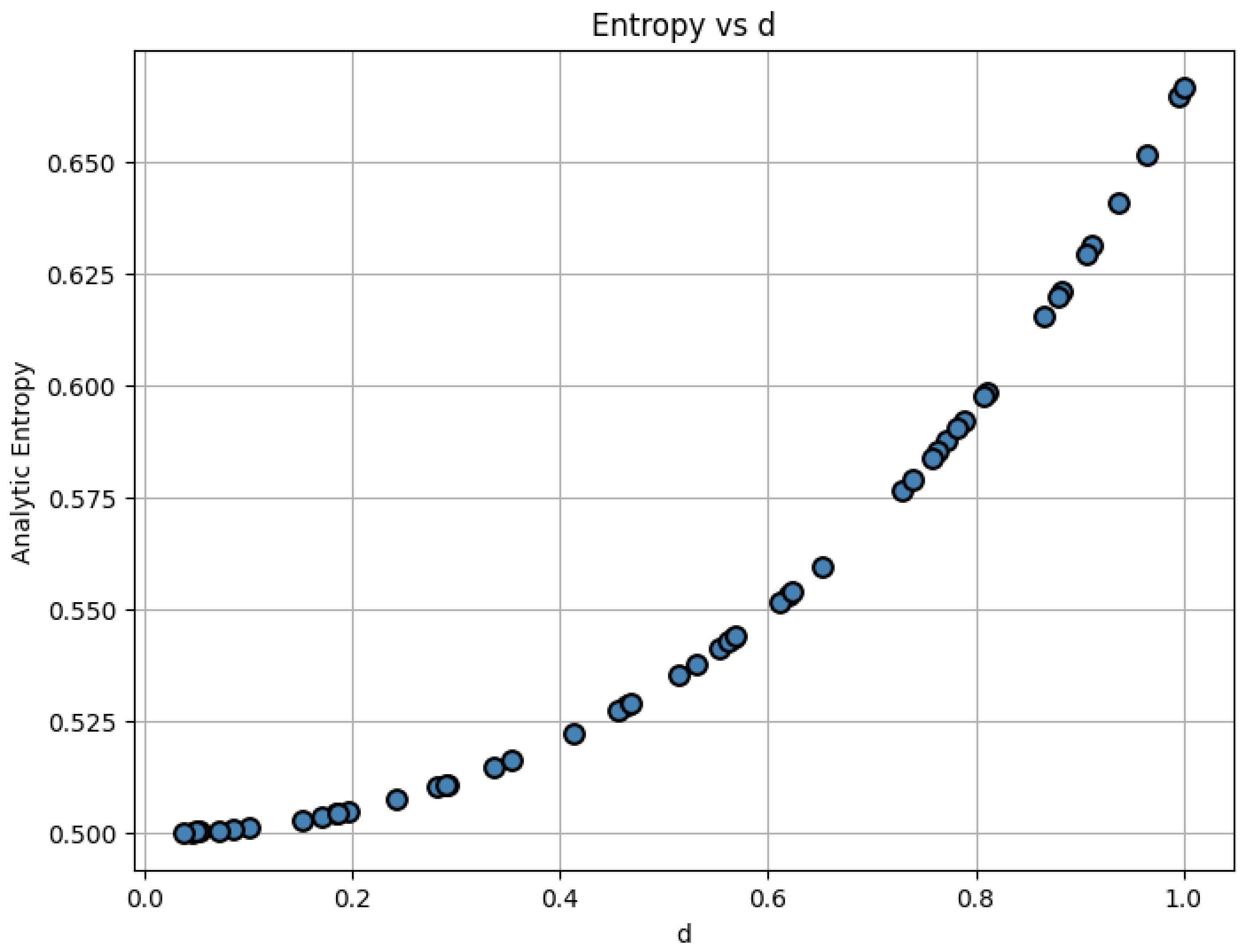


```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000, 0.1000, 0.0000])
high_tensor: tensor([ 4,  3, 15])
data_type: corr
prefix: None
```

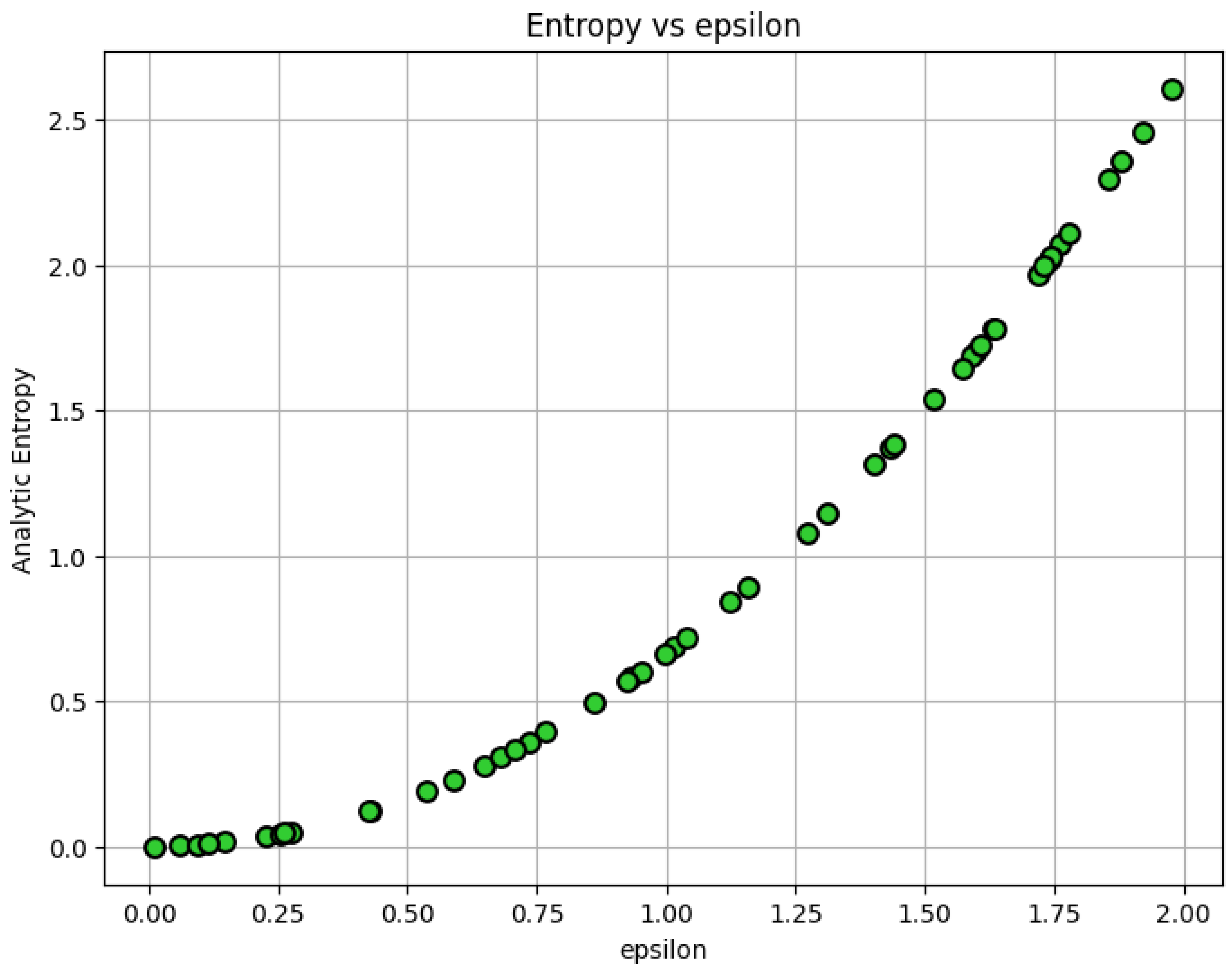
Analytic Entropy vs tau Model 2



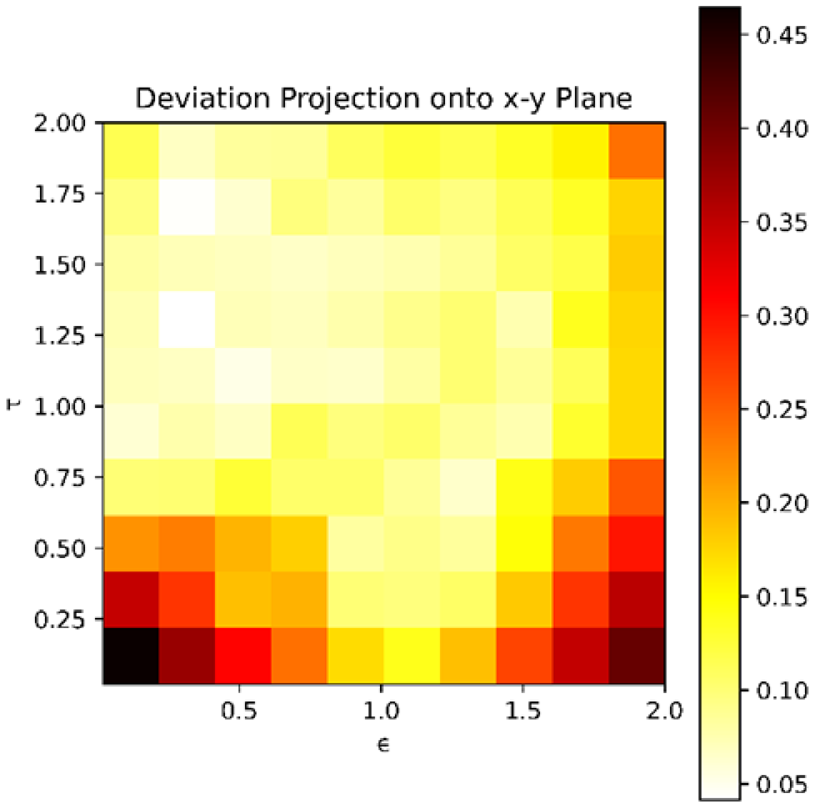
Analytic Entropy vs d Model 2

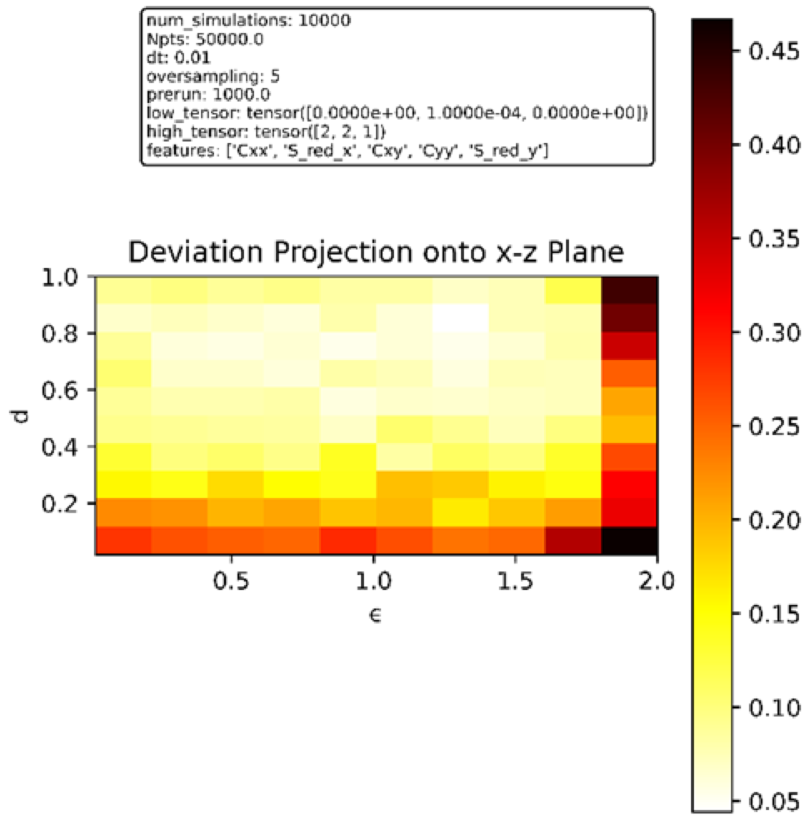


Analytic Entropy vs epsilon Model 2

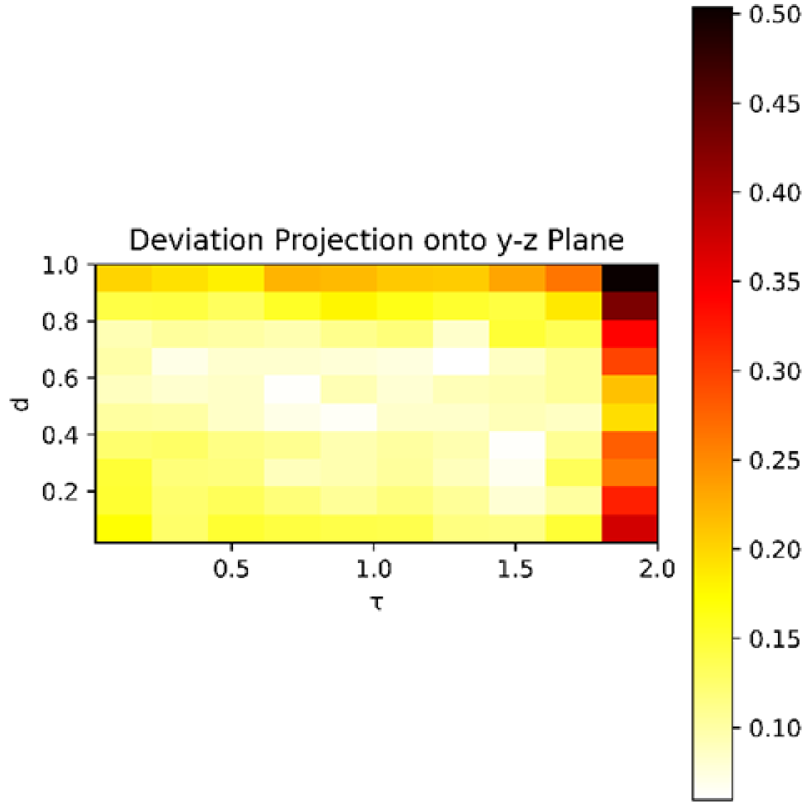


Heatmap della deviazione di entropia rispetto ai parametri





```
num_simulations: 10000
Npts: 50000.0
dt: 0.01
oversampling: 5
prerun: 1000.0
low_tensor: tensor([0.0000e+00, 1.0000e-04, 0.0000e+00])
high_tensor: tensor([2, 2, 1])
features: ['Cxx', 'S_red_x', 'Cxy', 'Cyy', 'S_red_y']
```



Plugging $\overline{E}(\tau)$ and $E(t+\tau)$ into the autocorrelation function therefore gives

$$\begin{aligned}
C(t) &= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}\overline{E}_{\nu}\, e^{2\pi i \nu \tau}\, d\nu\right]\left[\int_{-\infty}^{\infty} E_{\nu'}\, e^{-2\pi i \nu' (t+\tau)}\, d\nu'\right] d\tau \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\overline{E}_{\nu}\, E_{\nu'}\, e^{-2\pi i \tau(\nu'-\nu)}\, e^{-2\pi i \nu' t}\, d\tau\, d\nu\, d\nu' \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\overline{E}_{\nu}\, E_{\nu'}\, \delta(\nu'-\nu)\, e^{-2\pi i \nu' t}\, d\nu\, d\nu' \\
&= \int_{-\infty}^{\infty}\overline{E}_{\nu}\, E_{\nu}\, e^{-2\pi i \nu t}\, d\nu \\
&= \int_{-\infty}^{\infty}|E_{\nu}|^2\, e^{-2\pi i \nu t}\, d\nu \\
&= \mathcal{F}_{\nu}\left[|E_{\nu}|^2\right](t),
\end{aligned}$$

so, amazingly, the autocorrelation is simply given by the Fourier transform of the absolute square of E_{ν}.