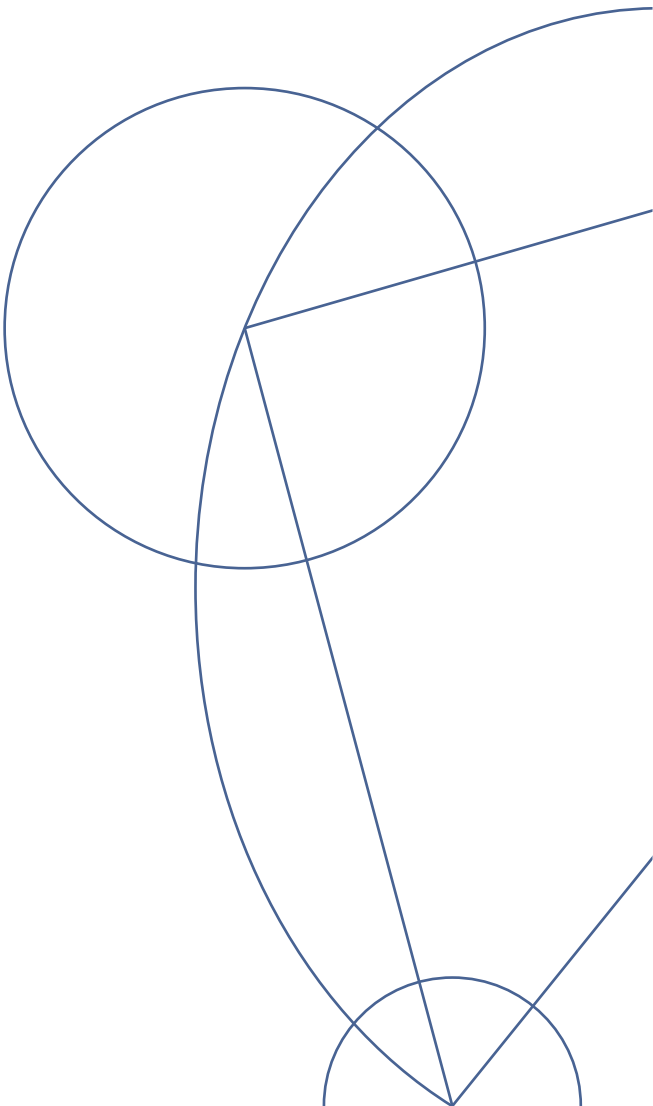
National Technical University of Athens
School of Electrical and Computer Engineering
Inter-Faculty Postgraduate Studies Programme
Data Science and Machine Learning

Numerical Solution of the Poisson Equation and Monte Carlo Simulations

The third assignment written in partial fulfillment of the requirements for the completion of the Data Driven Models in Engineering elective course of the Data Science & Machine Learning post-graduate studies programme.

Instructor
Prof. V. Papadopoulos

Written by
[redacted]



April 29, 2022

1 Derivation of the Equations

The purpose of the present assignment is the solution of Poisson's 2D equation,

$$\nabla^2 T(x,y) = -f(x,y), \tag{1.1}$$

where $T(x,y)$ is a temperature field and

$$f(x,y) = 100 \cdot \exp\left[-\frac{(x-0.55)^2 + (y-0.45)^2}{r}\right] \tag{1.2}$$

corresponds to an external heat source term. In Eq. (1.2), $r \sim \mathcal{N}\left(0.05, 0.005^2\right)$ is a normal random variable, the standard deviation of which is assumed to be $\sigma = 0.005$, instead of $\sqrt{0.005}$ as implied by the assignment's instructions[1]. The physical reason for this is that r should only assume positive values, since it corresponds to the heat source's exponential decay constant; negative values of r would imply that the heat provided by the source becomes more intense as the distance from the source increases. That said, if the standard deviation of r is $\sqrt{0.005} \approx 0.07$, then the probability of a value of r generated at the negative axis corresponds to less than 1σ. On the other hand, the probability of generating a negative value of r for $\sigma = 0.005$ corresponds to more than 10σ, which is considered acceptable[2].

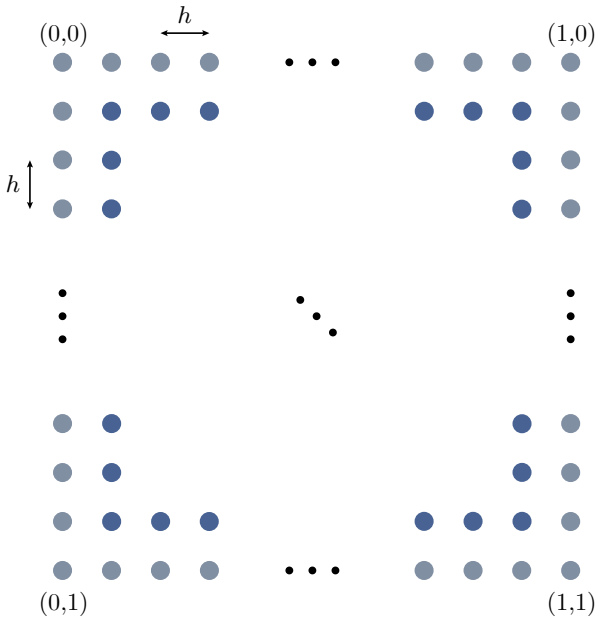


Figure 1: The grid considered for the solution of Eq. (1.1).

The space corresponds to a rectangular plate of unit dimensions and $T(x,y) = 0$ holds at the plate's edges. In order to solve Eq. (1.1), the plate is discretized using a grid consisting of 41×41 points in such a way that the grid's edges (light blue points in Fig. 1) are aligned on the $x = 0$, $y = 0$, $x = 1$, $y = 1$ lines. The points across the grid's axes are equidistant, with $\Delta x = \Delta y = h = 1/40$. As a result, the derivatives of Eq. (1.1) can be written as

[1] This was later confirmed via e-mail correspondence with Dr. I. Kalogeris.

[2] Technically speaking, the probability should be strictly zero, so any non-zero value is still theoretically pathological. However, the number of expected experiments required in order to get a single negative value of r is much higher than the number of experiments performed for the purposes of this assignment.

$$\nabla^2 T(x,y) \to \frac{1}{h^2}\left[T(x+h,y)+T(x-h,y)+T(x,y+h)+T(x,y-h)-4T(x,y)\right] \tag{1.3}$$

using a second-order central difference approximation. Therefore, the finite difference approximation to Eq. (1.1) is

$$(1.1) \to -T(x+h,y)-T(x-h,y)-T(x,y+h)-T(x,y-h)+4T(x,y)=h^2 f(x,y). \tag{1.4}$$

In Eq. (1.4), x and y are discrete variables that can assume values in the domain $\{0,h,2h,\ldots,39h,1\}$, thus the target is the calculation of the 41×41 values for $T(x,y)$ - one on each grid point. In fact, since the boundary conditions are

$$T(x=0,y)=T(x,y=0)=0=T(x=1,y)=T(x,y=1), \tag{1.5}$$

Eq. (1.4) needs to be solved for the 39×39 points that do not lie on the plate's edges (i.e. the dark blue points in Fig. 1). It is stressed at this point that "removing" the edge points is neither equivalent to "ignoring" them, nor a simplification, but rather the consistent thing to do from a physical standpoint: the present problem can be seen as a system of interacting points, where each point has a self interaction with relative strength 4 and a first-neighbor interaction with relative strength -1, as shown on the left of Fig. 2. By removing the edge points, essentially one (or more) interaction(s) is (are) removed for every non-edge point that has one (or more) edge points as first neighbor(s) (see the right of Fig. 2).

Figure 2: The interactions of a grid point with itself and its first neighbors in the case where (left) all neighbors are non-edge points and (right) one neighbor is an edge point.

Taking everything into consideration, the coupled equations of Eq. (1.5) can be expressed in matrix form as

$$\mathbf{K}\cdot\mathbf{t}=\mathbf{b}. \tag{1.6}$$

In the equation above, $\mathbf{t}$ is a 39×39-dimensional vector which hosts the values of $T(x,y)$ using the following convention: if (x_i,y_j) is a grid point, with $x_i=y_i=(i+1)h$, where $i,j\in\{0,\ldots,38\}$, then

$$t_{39i+j}=T(x_i,y_j). \tag{1.7}$$

Note that the notation is "pythonic", in the sense that indexing begins from 0 instead of 1, in order for the mathematical equations to be consistent with the code presented in what follows. Essentially, the solution of (1.4) is equivalent to the calculation of the $\mathbf{t}$ vector. The $\mathbf{K}$ matrix is the $(39\times 39, 39\times 39)$-dimensional "coupling matrix", which couples (x_i,y_j) to its first neighbors, namely (x_{i+1},y_j), (x_{i-1},y_j), (x_i,y_{j+1}) and (x_i,y_{j-1}), with a relative strength of -1, as long as these neighbors are not edge-points. In the latter case, the corresponding coupling strength is set equal to zero. Furthermore, $\mathbf{K}$ couples

(x_i, y_j) to itself with a relative strength of 4. Finally, **b** is the 39×39-dimensional vector corresponding to the value of $f(x_i, y_j)$ multiplied by a factor of h^2. The code developed to construct and solve these equations is given in Snippet 1.

```python
import numpy as np

def K_matrix(ppdim = 39):
    """
    Args:
        ppdim (int): The number of points that constitute the grid
        along each dimension, without counting the boundary.
    """
    K = np.empty((ppdim**2,ppdim**2))
    for i in range(ppdim**2):
        for j in range(ppdim**2):
            if j == i:
                K[i][j] = 4.0 # on-site interaction
            elif (j == i+1) or (j == i-1) or (j == i+ppdim) or (j == i-ppdim):
                K[i][j] = -1.0 # 1st neighbor interaction
            else:
                K[i][j] = 0.0 # no interaction
    return K

def b_vector(ppdim = 39, r = 0.05):
    """
    Args:
        r (float): The decay length of the candle
    """
    b, idx = np.empty(ppdim**2), 0
    h = 1/(ppdim+1)
    for i in range(0,ppdim):
        for j in range(0,ppdim):
            x, y = (i+1)*h, (j+1)*h # lattice points
            b[idx] = (h**2)*100.0*np.exp(-((x-0.55)**2+(y-0.45)**2)/r)
            idx += 1
    return b

def solve_system(invK, ppdim = 39, r = 0.05, full=True):
    """
    Args:
        invK (np.array): the inverse of the coupling matrix
        full (bool): Boolean switch to return the result as a vector
        or as a matrix, including the edge points.
    """
    b = b_vector(ppdim, r)
    t = np.matmul(invK,b) # Kt = b
    if full:
        t = t.reshape(ppdim,ppdim) # bring into grid form
        t = np.pad(t, 1, mode='constant') # add border
        t = t.T
    return t
```

Python Code Snippet 1: Code for the construction and solution of Eq. (1.6)

2 Monte Carlo Simulation

Before proceeding with the required Monte Carlo simulation, it is important to depict some solutions for $\mathbf{t}$ as a heatmap, in order to illustrate how the decay length, r, affects the system's solution. Using the `solve_system()` function defined in Snippet 1 for $r = \mu - 2\sigma$, $r = \mu$ and $r = \mu + 2\sigma$, where $(\mu, \sigma) = (0.05, 0.005)$, the following heatmaps (Fig. 3) are obtained. The x and y axes correspond to the grid, while the color intensity corresponds to the heat on every grid point. Note that the edge-points, the temperature of which is equal to zero, have also been included.

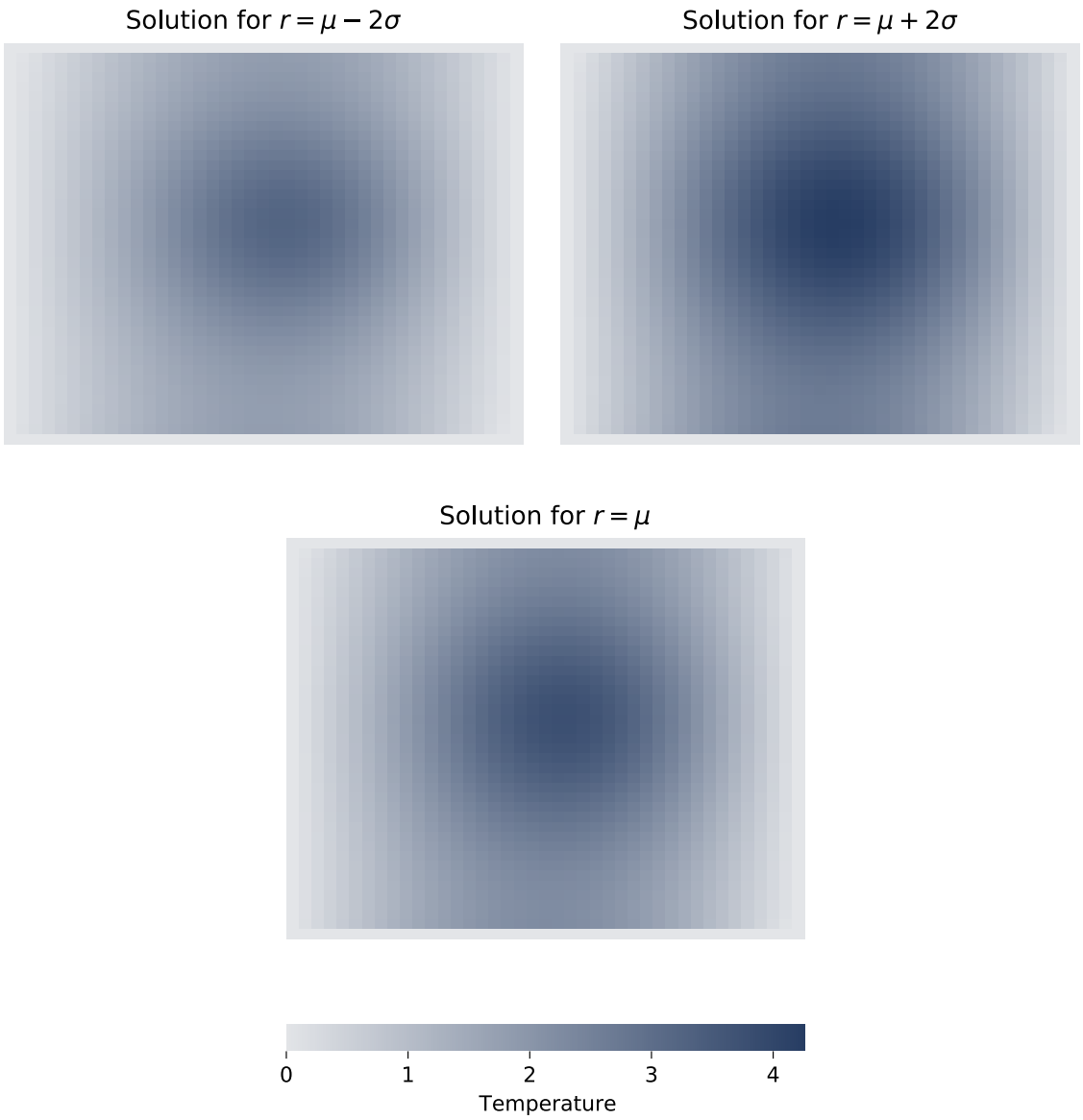


Figure 3: Heatmaps of the equations' solution for three different values of r.

The colorbar is calibrated in such a way that all heatmaps are portrayed on the same color scale, allowing the identification of the effect that r has on the final solution. It becomes evident that as the decay length increases, so does the grid's overall temperature, since the perturbation caused by the heat source is able to propagate further. In addition, an interesting observation is that the heatmaps are not symmetric with respect to the grid's midpoint, which is to be expected since the heat source is not located on it.

Having witnessed the effect of r on the final solution, one may proceed with a Monte Carlo simulation in order to obtain the probability density function of the plate's temperature, the probabilistic nature of which is a result of r being a random variable. Building up on the previously presented code, the following block (Snippet 2) performs the required simulation for 15000 experiments and returns the results only for the plate's midpoint.

```
K = K_matrix(39)
invK = np.linalg.inv(K)

n_samples = 15000
results = []
for i in range(n_samples):
    r = np.random.normal(0.05,0.005,1)
    t = solve_system(invK = invK, ppdim = 39, r = r)
    results.append(t[20,20])
```

Python Code Snippet 2: Code for the Monte Carlo simulation

Using the simulation's results, one finds that

$$\mathbb{E}\left[T\left(0.5, 0.5\right)\right] = 3.693, \qquad \mathrm{Var}\left[T\left(0.5, 0.5\right)\right] = 0.065. \tag{2.1}$$

The full density histogram corresponding to the results is shown in Fig. 4, where an estimated distribution (Seaborn's kernel density estimate) is also depicted as a solid line.

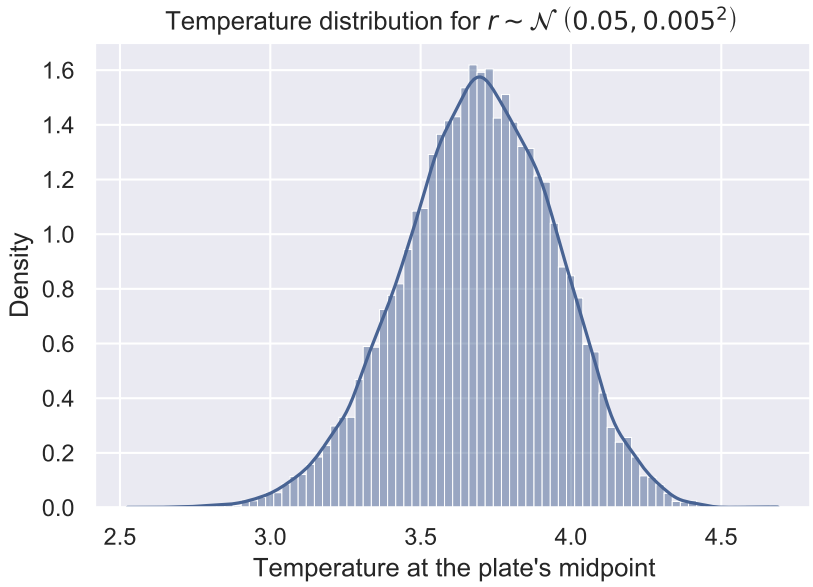


Figure 4: Density histogram for the plate's midpoint temperature and estimated distribution.

Even though visually it appears as if the distribution that $T\left(0.5, 0.5\right)$ follows is Gaussian, more information on this can only be acquired via analytical approaches. There are various options, such as the Kolmogorov-Smirnov test, however the more frequently used Kullback-Leibler (KL) divergence is utilized here. For this purpose, 15000 samples are drawn from $\mathcal{N}\left(3.693, 0.065\right)$ and the KL divergence of the distribution shown in Fig. 4 from the distribution $\mathcal{N}\left(3.693, 0.065\right)$ is calculated using SciPy's `stats.entropy()` function. The result is $D_{\mathrm{KL}} = 0.005 = \mathcal{O}\left(10^{-3}\right)$, implying that the distribution shown in Fig. 4 indeed approaches $\mathcal{N}\left(3.693, 0.065\right)$. Of course, the fact that $T\left(x, y\right)$ follows a normal distrbution is to be expected, since r also does.

3 PCA Implementation

One of the biggest obstacles encountered during the implementation described in the previous section is often the computational time required for the inversion of the coupling matrix[3]. In the present case, its dimensions are $(39 \times 39, 39 \times 39)$, however there are problems where the number of effective features (the components of $\mathbf{t}$ in this case) is much larger. In these cases, the application of dimensionality reduction methods is a common practice, with POD/PCA being the most widely used. To apply the POD/PCA method in the present case, one first performs N_{red} full Monte Carlo simulations, however now $N_{\text{red}} \ll 15000$, where 15000 is the number of experiments performed in the previous section ($N_{\text{red}} = 100$ is a good choice). Using the results of these simulations, a dataset is constructed which can be presented as a $(39 \times 39, N_{\text{red}})$-dimensional matrix, $\mathbf{V}$, each column of which corresponds to the solution of one simulation. Then, the application of PCA in the dataset becomes equivalent to the identification of the m eigenvectors $\phi_1, \dots, \phi_m$ that correspond to the m highest eigenvalues of $\mathbf{V} \cdot \mathbf{V}^\top$, where $m \leq N_{\text{red}}$. Denoting $\Phi = [\phi_1, \dots, \phi_m]^\top$, one may write

$$\mathbf{t} \simeq \Phi^\top \tilde{\mathbf{t}}, \tag{3.1}$$

where $\tilde{\mathbf{t}}$ is the m-dimensional solution vector and the approximation error of Eq. (3.1) is equal to the sum of the $39 \times 39 - m$ remaining eigenvalues (supposing normalization). The purpose of this approximation is that Eq. (1.6) now assumes the form

$$\mathbf{K} \cdot \Phi^\top \cdot \tilde{\mathbf{t}} = \mathbf{b} \Leftrightarrow \mathbf{K}_{\text{red}} \cdot \tilde{\mathbf{t}} = \mathbf{b}_{\text{red}}, \tag{3.2}$$

where $\mathbf{K}_{\text{red}}$ is the reduced (m, m)-dimensional coupling matrix satisfying

$$\mathbf{K}_{\text{red}} = \Phi \cdot \mathbf{K} \cdot \Phi^\top \tag{3.3}$$

and $\mathbf{b}_{\text{red}} = \Phi \cdot \mathbf{b}$. In this way, the problem's solution now requires the inversion of a matrix with considerably lower dimensions. This leads to a significant reduction of computational time, at the expense of a configurable loss in accuracy, in the sense that the percentage of the explained variance can be calculated for any choice of m. The code developed to perform the sampling and the subsequent application of PCA can be seen in Snippet 3.

```
N_red, ppdim = 100, 39
pcadata = np.zeros((N_red,ppdim**2))
for i in range(N_red):
    r = np.random.normal(0.05,0.005,1)
    t = solve_system(ppdim = ppdim, r = r, full=False)
    pcadata[i] = t

from sklearn.decomposition import PCA

depth = min(pcadata.shape[1],N_red)
pca_vars = np.zeros(depth+1)
for i in range(1,depth+1):
    pca = PCA(n_components=i).fit(pcadata)
    pca_vars[i] = 100*(pca.explained_variance_ratio_.sum())
```

Python Code Snippet 3: Code for the sampling and application of PCA

[3] Note that, while Python is not the optimal coding language when it comes to linear algebra, the `NumPy` package used herein includes precompiled C libraries which are optimized for such purposes.

The graph of Fig. 5 depicts the percentage of the dataset's explained variance as a function of the number of principal components, m, chosen for the PCA.

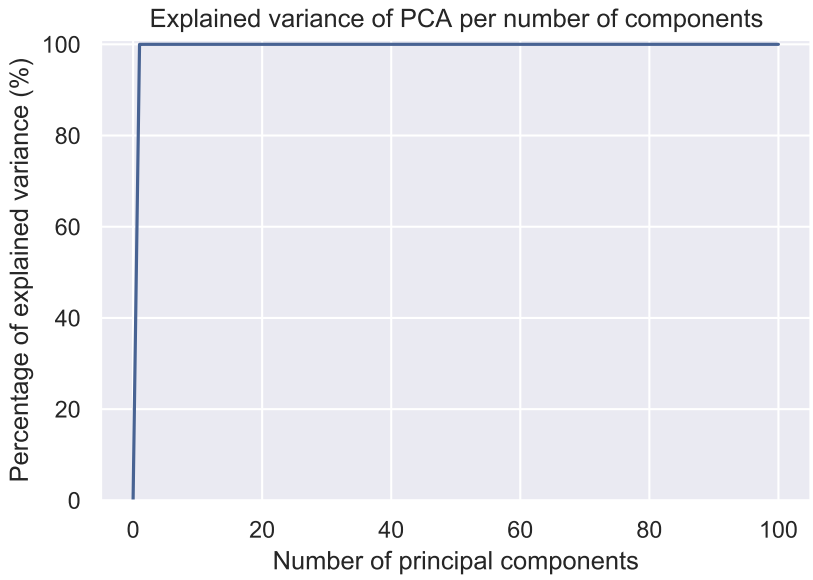


Figure 5: Percentage of explained variance as a function of m after the application of PCA.

Evidently, 99.993% of the dataset's variance can be explained by a single principal component. For this reason, the procedure explained in Eqs. (3.1) - (3.3) is followed for $m = 1$, in order to reduce the dimensions of the coupling matrix. In fact, since $m = 1$, the coupling matrix is reduced to a single scalar, therefore its inversion is trivial. The code developed to apply Eqs. (3.1) - (3.3) and perform the Monte Carlo simulation for 15000 samples is shown in Snippet 4.

```
pca = PCA(n_components=1).fit(pcadata)
Phi = pca.components_

K = K_matrix(ppdim)
K_red = np.matmul(Phi,np.matmul(K,Phi.T))
K_red_inv = 1./K_red[0][0] # K_red is a scalar

pca_results = []

for i in range(n_samples):
    r = np.random.normal(0.05,0.005,1)
    b = b_vector(ppdim, r)
    b_red = np.matmul(Phi,b)
    t = np.matmul(Phi.T,K_red_inv*b_red)
    pca_results.append(t[39*19+19]) # t is 1D here
```

Python Code Snippet 4: Code for the Monte Carlo simulation after applying PCA

The results of this simulation indicate that

$$\mathbb{E}\left[T'\left(0.5, 0.5\right)\right] = 2.99, \qquad \mathrm{Var}\left[T'\left(0.5, 0.5\right)\right] = 0.064, \tag{3.4}$$

where $T'\left(0.5, 0.5\right)$ is the solution for the temperature at the plate's midpoint using the POD/PCA method. It turns out that, while the variance of the new results is practically the same with the variance corresponding to the previous section's method, the same can't be told about their mean value. This

difference is a result owing to the fact that `scikit-learn`'s implementation of the PCA algorithm includes an additional step of subtracting the dataset's mean vector from each sample before performing the decomposition (see *Different variations of POD/PCA*, Section 5.4 of [1]). Nonetheless, the KL divergence of the distribution obtained in this section from the distribution obtained in the previous section is calculated to be equal to 0.006, thus indicating an expectedly high percentage of similarity. The full density histogram corresponding to the results using the POD/PCA method can be seen in light blue color in Fig. 6 (labeled as 'PCA'), alongside the original density histogram (labeled as 'Full').

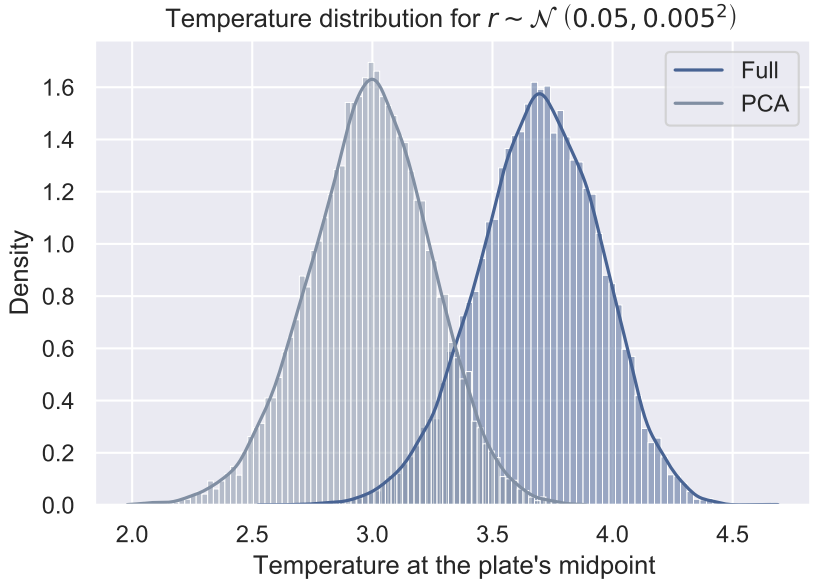


Figure 6: Temperature distribution for the plate's midpoint using the POD/PCA method (light blue).

For completeness, before closing this presentation, additional code is provided in Snippet 5 where a "custom" (i.e. without using `scikit-learn`'s implementation) PCA method is developed, without including the mean vector subtraction step. The purpose of this is to prove that the two distributions' mean values' divergence is indeed due to this step.

```
V_mat = np.matmul(pcadata.T,pcadata)
w, v = np.linalg.eig(V_mat)

n_components = 1 # <- change this to other values
Phi_new = np.real(v[:,:n_components]).T

K = K_matrix(ppdim)
K_red = np.matmul(Phi_new,np.matmul(K,Phi_new.T))
K_red_inv = np.linalg.inv(K_red)

pca_results_new = []

for i in range(n_samples):
    r = np.random.normal(0.05,0.005,1)
    b = b_vector(ppdim, r)
    b_red = np.matmul(Phi_new,b)
    t = np.matmul(Phi_new.T,np.matmul(K_red_inv,b_red))
    pca_results_new.append(t[39*19+19])
```

Python Code Snippet 5: Code for the Monte Carlo simulation after applying a customly written PCA

The results of this final simulation indicate that

$$\mathbb{E}\left[T''\left(0.5, 0.5\right)\right] = 3.694, \qquad \mathrm{Var}\left[T''\left(0.5, 0.5\right)\right] = 0.065, \tag{3.5}$$

where $T''\left(0.5, 0.5\right)$ is the solution for the temperature at the plate's midpoint using the "custom" PCA algorithm. Indeed, in this case both the mean value and the variance are practically identical to the ones obtained when the full coupling matrix was used. The corresponding density histogram can be seen in Fig. 7 (light blue color) alongside the original density histogram of Fig. 4 (dark blue color).

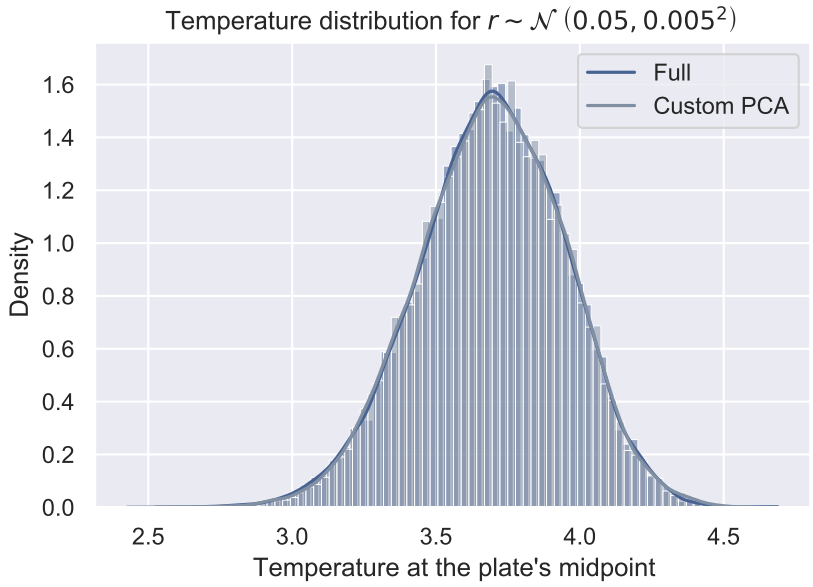


Figure 7: Results using the "custom" PCA method (light blue).

References

[1] I. Kalogeris, *Lecture Notes on Stochastic Finite Element Methods & Data-driven models in Engineering Applications*. 8.04.2021.