```
import numpy as np
import matplotlib.pyplot as plt
import sys
import os.path
from numba import njit
# explicitly add project root dir to path to fix import issue
sys.path.append(os.path.join(os.path.dirname(__file__), '..'))
from homework_6 import heateqn


# At first, we calculate the Clenshaw - Curtis nodes for the
interpolation
@njit
def ClenshawCurtisNodes(M):
    nodes = []
    for i in range(M):
        x = np.cos((i * np.pi) / M)
        nodes.append(x)
    return nodes


# Next, we construct the Lagrange basis functions
@njit
def lagrange_basis(x, data_points, j):
    """
    Calculate the Lagrange basis function for the j-th data point.

    Inputs:
    x: The point at which we evaluate the basis function.
    data_points (array-like): The list of data points (x values).
    j: The index of the data point for which we calculate the basis
function.

    Output:
    basis: The value of the j-th Lagrange basis function at point x.
    """
    n = len(data_points)
    basis = 1.0
    for i in range(n):
        if j != i:
            basis *= (x - data_points[i]) / (data_points[j] -
data_points[i])

    return basis

@njit
def to_nonstandard_uniform(val, a, b):
    """Map value on interval [-1, 1] to [a, b] where [a, b] is from
U[a, b]."""
    return ((b-a)/2)*val + (a + b)/2


def CollocationApproximation(Z, zetas, pde_eval_dict, x_vals):
    '''
    Calculates the approximation for a given Z
```

```
    Arguments:
        Z -- random variable
        zetas -- values of the nodes
        pde_eval_dict -- dictionary with the evaluation of the PDE at
the given nodes
        x_vals -- x values to be evaluated

    Returns:
        approximation, x values and the node values
    '''
    lagrange_basis_dict = {}
    # evaluate the PDE for all the zeta values
    for k, val in enumerate(zetas):
        # calculate the Lagrange basis functions
        basis = lagrange_basis(Z, zetas, k)
        lagrange_basis_dict[val] = basis

    u_approx = np.zeros(len(x_vals))
    for val in zetas:
        u_approx += lagrange_basis_dict[val]*pde_eval_dict[val]

    return u_approx, x_vals, zetas

def approx_M(M, zeta_vals):
    '''
    Generates a PDE approximation with M nodes for each value
zeta_value

    Arguments:
        M -- number of nodes
        zeta_vals -- array of zeta values

    Returns:
        the approximation and x values
    '''
    nodes = ClenshawCurtisNodes(M)
    # get the transformed set of nodes Z
    a = 2 # distribution parameters from problem statement
    b = 16
    zetas = [to_nonstandard_uniform(val, a, b) for val in nodes]

    #
    pde_eval_dict = {}
    for k, val in enumerate(zetas):
        x_vals, pde_sol = heateqn.heat_eq(val)
        pde_eval_dict[val] = pde_sol

    realisations = np.zeros((len(zeta_vals),len(x_vals)))
    for i, val in enumerate(zeta_vals):
        approx, _, _ = CollocationApproximation(val, np.array(zetas),
pde_eval_dict, x_vals)
        realisations[i] = approx

    return x_vals, realisations
```