```python
import numpy as np
from scipy.stats import norm


def learning_function_MLE (mean_hat, var_hat, learning_function_type):
    '''
    Returns learning function value.

    Parameters
    ----------
    mean_hat : mean of metamodel predictions
        element type: array ; float

    var_hat : variance of metamodel predictions
        element type: array ; float

    learning_function_type : type of learning function employed (Options: U, var)
        element type: string

    Returns
    -------
    LFV : learning function values at population points
        element type: array ; float
    '''

    N = len(mean_hat)
    LFV = np.zeros(N)

    sigma_hat = np.sqrt(var_hat)

    if learning_function_type == 'U':
        for i in range(N):
            U = np.abs(mean_hat[i])/sigma_hat[i]
            LFV[i] = -U

    elif learning_function_type == 'var':
        LFV = sigma_hat

    return LFV

def learning_function_bayesian(p2_5, p50, p97_5, learning_function_type):
    '''
    Returns learning function value, Bayesian approach.

    Parameters
    ----------
    p2_5 : 2.5th percentile of predictions in each location
        element type: array ; float

    p50 : 50th percentile of predictions in each location
        element type: array ; float

    p97_5 : 97.5th percentile of predictions in each location
        element type: array ; float

    learning_function_type : type of learning function employed (Options: U, var)
        element type: string

    Returns
    -------
    LFV : learning function values at population points
        element type: array ; float
    '''
    N = len(p50)

    LFV = np.zeros(N)
```

```
    if learning_function_type == 'U':
        for i in range(N):
            U = np.abs(p50[i])/max(np.abs(p50[i] - p2_5[i]), np.abs(p97_5[i] - p50[i]))
            LFV[i] = -U

    elif learning_function_type == 'var':
        for i in range(N):
            LFV[i] = max(np.abs(p50[i] - p2_5[i]), np.abs(p97_5[i] - p50[i]))

    return LFV

def MSE_solver(median_preds, true_values):
    '''
    Solves mean squared error between model and metamodel.

    Parameters
    ----------
    median_preds : predictive medians of metamodel
        element type: array ; float

    true_values : actual values from true model
        element type: array ; float

    Returns
    -------
    mean squared error betweel model and metamodel
        element type: float
    '''
    return np.mean((mean_preds - true_values) ** 2)

def calculate_Pf_MLE(mean_hat, var_hat):
    '''
    Calculates probability of failure (including bounds) and convergence (for reliability
problems).

    Parameters
    ----------
    mean_hat : mean of metamodel predictions
        element type: array ; float

    var_hat : variance of metamodel predictions
        element type: array ; float

    Returns
    -------
    Pf : predicted probability of failure
        element type: float
    Pf_plus : upper bound of predicted probability of failure
        element type: float
    Pf_minus : lower bound of predicted probability of failure
        element type: float
    conv_criterion : convergence criteriion value of predicted probability of failure
        element type: float
    '''

    N = len(mean_hat)
    sigma_hat = np.sqrt(var_hat)

    Pf_plus = np.sum(mean_hat-1.96*sigma_hat <= 0)/N
    Pf = np.sum(mean_hat <= 0)/N
    Pf_minus = np.sum(mean_hat+1.96*sigma_hat <= 0)/N

    conv_criterion = (Pf_plus - Pf_minus)/Pf

    return Pf, Pf_plus, Pf_minus, conv_criterion

def calculate_Pf_bayesian(p2_5, p50, p97_5):
```

```
    '''
    Calculates probability of failure (including bounds) and convergence (for reliability
problems), Bayesian approach.

    Parameters
    ----------
    p2_5 : 2.5th percentile of predictions in each location
        element type: array ; float

    p50 : 50th percentile of predictions in each location
        element type: array ; float

    p97_5 : 97.5th percentile of predictions in each location
        element type: array ; float

    Returns
    -------
    Pf : predicted probability of failure
        element type: float
    Pf_plus : upper bound of predicted probability of failure
        element type: float
    Pf_minus : lower bound of predicted probability of failure
        element type: float
    conv_criterion : convergence criteriion value of predicted probability of failure
        element type: float
    '''

    N = len(p50)

    Pf_plus = np.sum(p2_5 <= 0)/N
    Pf = np.sum(p50 <= 0)/N
    Pf_minus = np.sum(p97_5 <= 0)/N

    conv_criterion = (Pf_plus - Pf_minus)/Pf

    return Pf, Pf_plus, Pf_minus, conv_criterion
```