```python
import numpy as np
import sympy as sp

def gradient_function_fd(minibatch, epsilon=10**(-15)):
    def gradient_fd(x):
        dydx1 = (f(x + np.array([epsilon, 0]), minibatch) - f(x, minibatch)) / epsilon
        dydx2 = (f(x + np.array([0, epsilon]), minibatch) - f(x, minibatch)) / epsilon
        return np.array([dydx1, dydx2])
    return gradient_fd

def loss(x, w):
    z = x - w - 1
    left = 10 * (z[0]**2+z[1]**2)
    right = (z[0]+2)**2+(z[1]+4)**2
    return min(left, right)

def f_clear(x, minibatch):
    return sum(loss(x, w) for w in minibatch) / len(minibatch)

def generate_minibatches(T, n=5, seed=42, shuffle=True):
    if shuffle:
        T = T.copy()
        np.random.seed(seed)
        np.random.shuffle(T)
    num_rows = T.shape[0]
    i = 0

    minibatch = np.zeros((n, T.shape[1]), T.dtype)
    while True:
        for j in range(n):
            minibatch[j] = T[i % num_rows]
            i += 1
            if shuffle and i >= num_rows:
                # begin next epoch
                np.random.shuffle(T)
                i = 0
        current_minibatch = minibatch
        yield minibatch

def generate_optimisation_functions(batch, minibatch_size=5, finite_difference=True, **kwargs):
    minibatch_generator = generate_minibatches(
        batch, n=minibatch_size, **kwargs)
    for minibatch in minibatch_generator:
        def optim_func(x):
            return f_clear(x, minibatch)
        gradf = gradient_function_fd(minibatch)
        yield (optim_func, gradf)
    yield "finished"
```