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

current_minibatch = None

def generate_trainingdata(m=25):
    return np.array([0,0]) + 0.25 * np.random.randn(m,2)


def f(x, minibatch):
    # loss function sum_{w in training data} f(x,w)
    y = 0
    count = 0
    for w in minibatch:
        z = x - w - 1
        left = 10 * (z[0]**2+z[1]**2)
        right = (z[0]+2)**2+(z[1]+4)**2
        y = y + min(left, right)
        count = count + 1
    return y/count


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 sympy_loss(minibatch):
    x1, x2 = sp.symbols('x1 x2', real=True)
    function = 0
    for w in minibatch:
        z1 = x1 - w[0] - 1
        z2 = x2 - w[1] - 1
        left = 10 * (z1**2 + z2**2)
        right = (z1 + 2)**2 + (z2 + 4)**2
        function = sp.Min(left, right) + function
    function = function / len(minibatch)
    return function

def gradient_function(minibatch):
    function = sympy_loss(minibatch)
    def gradient(x):
        dydx1 = function.diff(x1)
        dydx2 = function.diff(x2)
        return np.array([
            dydx1.subs(x1, x[0]).subs(x2, x[1]),
            dydx2.subs(x1, x[0]).subs(x2, x[1]),
        ])

    return gradient


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,):
    global current_minibatch
    if shuffle:
        T = T.copy()
        if seed:
            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 = None
        if finite_difference:
            gradf = gradient_function_fd(minibatch)
        else:
            gradf = gradient_function(minibatch)
        yield (optim_func, gradf)
    yield "finished"


if __name__ == "__main__":
    import os
    os.makedirs("data", exist_ok=True)
    T = generate_trainingdata()
    import pandas as pd
    df = pd.DataFrame(T)
    df.to_csv("data/T.csv", index=False)

    x = np.array([3, 3])
```