(a) $L(y,\hat{y}) = \frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y})^2$

$$\frac{\partial L(y,f_0)}{\partial f_0} = \frac{\partial \frac{1}{2}\sum_{i=1}^{n}(y_i-f_0)^2}{\partial f_0} = \sum_{i=1}^{n}(y_i-f_0)=0$$

$$\Rightarrow f_0 = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$r_{t,i} = -\frac{\partial}{\partial f(x_i)}\sum_{j=1}^{n} L(y_j, f(x_i))\Big|_{f(x_j)=f_{t-1}(x_j)}$$

$$= -\frac{\partial \frac{1}{2}\sum_{j=1}^{n}(y_j - f_{t-1}(x_j))^2}{\partial f_{t-1}(x_i)}\Big|_{f(x_j)=f_{t-1}(x_j)}$$

$$= -\,-2\times\frac{1}{2}(y_i - f_{t-1}(x_i))$$

$$= y_i - f_{t-1}(x_i)$$

$$h_t = \underset{f\in F}{\operatorname{argmin}}\sum_{i=1}^{n} l(r_{t,i}, f(x_i)) = \underset{f\in F}{\operatorname{argmin}}\sum_{i=1}^{n} l(y_i - f_{t-1}(x_i), f(x_i))$$

$$\Rightarrow h_t = \underset{f}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{2}(f_{t-1}(x_i)+c-y_i)^2 = \underset{f}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{2}[c-(y_i - f_0(x_i))]^2$$

$$= \underset{f}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{2}(c-r_{ti})^2$$

(b)

$$\alpha_t = \underset{\alpha}{\operatorname{argmin}}\sum_{i=1}^{n} l(y_i, f_{t-1}(x_i) + \alpha h_t(x_i))$$

$$= \underset{\alpha}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{2}(y_i - f_{t-1}(x_i) - \alpha h_t(x_i))^2$$

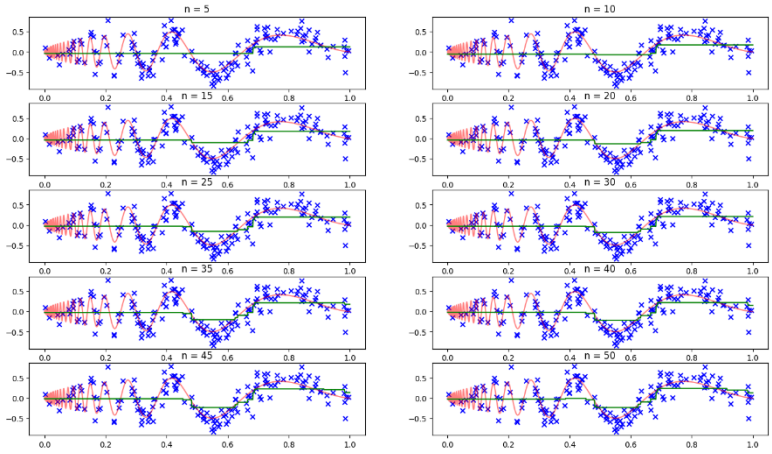
$$= \underset{\alpha}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{2}(r_{ti} - \alpha h_t(x_i))^2$$

$$\frac{\partial \frac{1}{2}\sum_{i=1}^{n}(r_{ti}-\alpha h_t(x_i))^2}{\partial \alpha} = 0 \Rightarrow \alpha\sum_{i=1}^{n} h_t^2(x_i) - 2\sum_{i=1}^{n} h_t(x_i)\cdot r_{ti} = 0$$

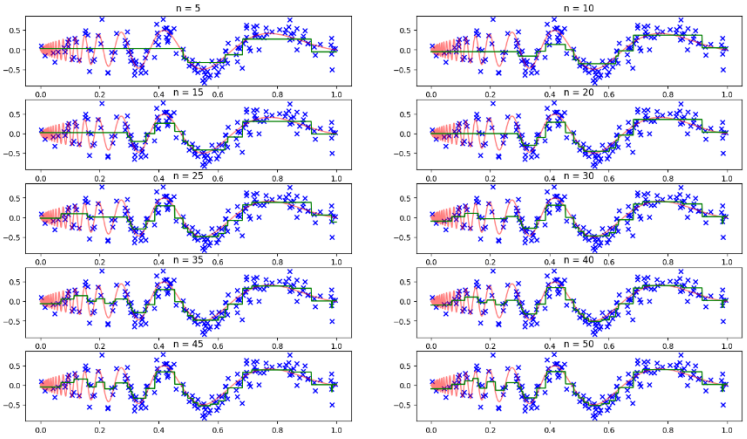
$$\Rightarrow \alpha = \frac{\sum_{i=1}^{n} h_t(x_i)\cdot r_{ti}}{\sum_{i=1}^{n} h_t^2(x_i)} = \frac{\sum_{i=1}^{n} h_t(x_i)(y_i - f_{t-1}(x_i))}{\sum_{i=1}^{n}(h_t(x_i))^2}$$

(c)

Learning rate = 0.1:



Learning rate =adaptive step size:



The model obtained from the adaptive step is a better fit to the data set.

The model struggles to achieve true function levels in the faster oscillating left-hand side of the data segment, but the fit is excellent in the slower oscillating right-hand side.

```python
from sklearn.tree import DecisionTreeRegressor
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    t1 = np.sqrt(x * (1-x))
    t2 = (2.1 * np.pi) / (x + 0.05)
    t3 = np.sin(t2)
    return t1*t3


def f_sampler(f, n=100, sigma=0.05):
    # sample points from function f with Gaussian noise (0,sigma**2)
    xvals = np.random.uniform(low=0, high=1, size=n)
    yvals = f(xvals) + sigma * np.random.normal(0,1,size=n)

    return xvals, yvals
```

```python
def predict(n_estimators, learning_rate, estimators, X):
    H = np.ones(X.shape[0]) * H0
    for k in range(n_estimators):
        estimator = estimators[k]
        y_predict = estimator.predict(X)
        H += learning_rate * y_predict
    return H


def predict2(n_estimators, learning_rate, estimators, X):
    H = np.ones(X.shape[0]) * H0
    for k in range(n_estimators):
        estimator = estimators[k]
        y_predict = estimator.predict(X)
        H += learning_rate[k] * y_predict
    return H
```

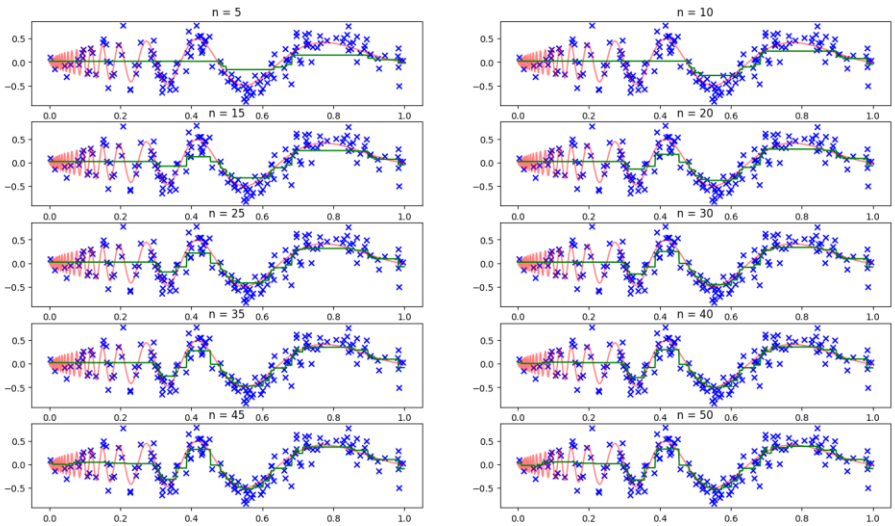
```
np.random.seed(123)
X, y = f_sampler(f, 160, sigma=0.2)
X = X.reshape(-1,1)

# (ii)
fig, ax = plt.subplots(5,2)
for i, ax in enumerate(ax.flat):
    n_estimators = 5*(i+1)
    learning_rate = 0.1
    H0 = np.average(y)
    H = np.ones(X.shape[0]) * H0
    estimators = []
    for k in range(n_estimators):
        y_hat = y - H
        dt = DecisionTreeRegressor(max_depth=1)
        dt.fit(X, y_hat)
        y_predict = dt.predict(X)
        H += learning_rate * y_predict
        estimators.append(dt)
    estimators = np.array(estimators)
    xx = np.linspace(0, 1, 1000)
    ax.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
    ax.scatter(X, y, marker='x', color='blue', label='observed')
    ax.plot(xx, predict(n_estimators, learning_rate, estimators, xx.reshape(-1, 1)),
    title = 'n = '+str(n_estimators)
    ax.set_title(title)
# plt.legend(loc='upper right')
plt.show()
```

```
fig, ax = plt.subplots(5,2)
for i, ax in enumerate(ax.flat):
    n_estimators = 5*(i+1)
    H0 = np.average(y)
    H = np.ones(X.shape[0]) * H0
    estimators = []
    learning_rates = []
    for k in range(n_estimators):
        y_hat = y - H
        dt = DecisionTreeRegressor(max_depth=1)
        dt.fit(X, y_hat)
        y_predict = dt.predict(X)
        learning_rate = np.dot(y_predict,y_hat)/np.dot(y_predict,y_predict.T)
        H += learning_rate * y_predict
        estimators.append(dt)
        learning_rates.append(learning_rate)
    estimators = np.array(estimators)
    xx = np.linspace(0, 1, 1000)
```

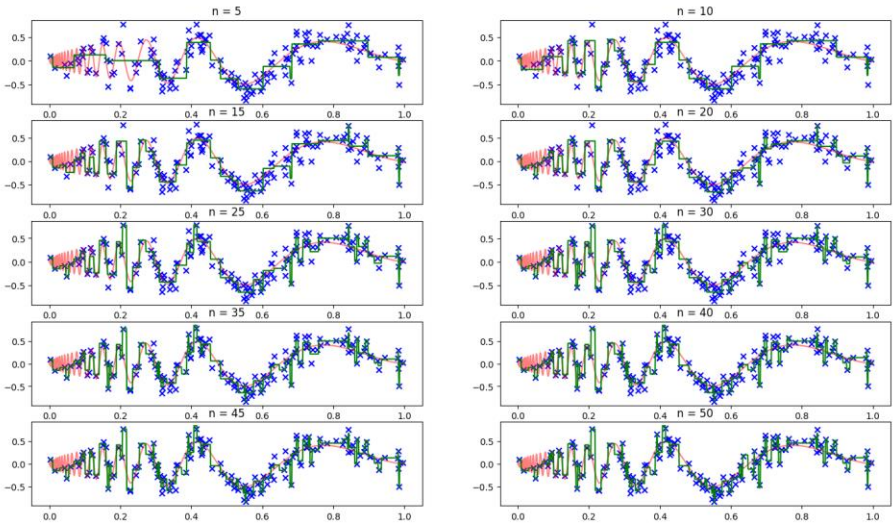
```
    ax.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
    ax.scatter(X, y, marker='x', color='blue', label='observed')
    ax.plot(xx, predict2(n_estimators, learning_rates, estimators, xx.reshape(-1, 1)), col
    title = 'n = '+str(n_estimators)
    ax.set_title(title)
# plt.legend(loc='upper right')
plt.show()
```

(d)

Learning rate = 0.1:



Learning rate =adaptive step size:

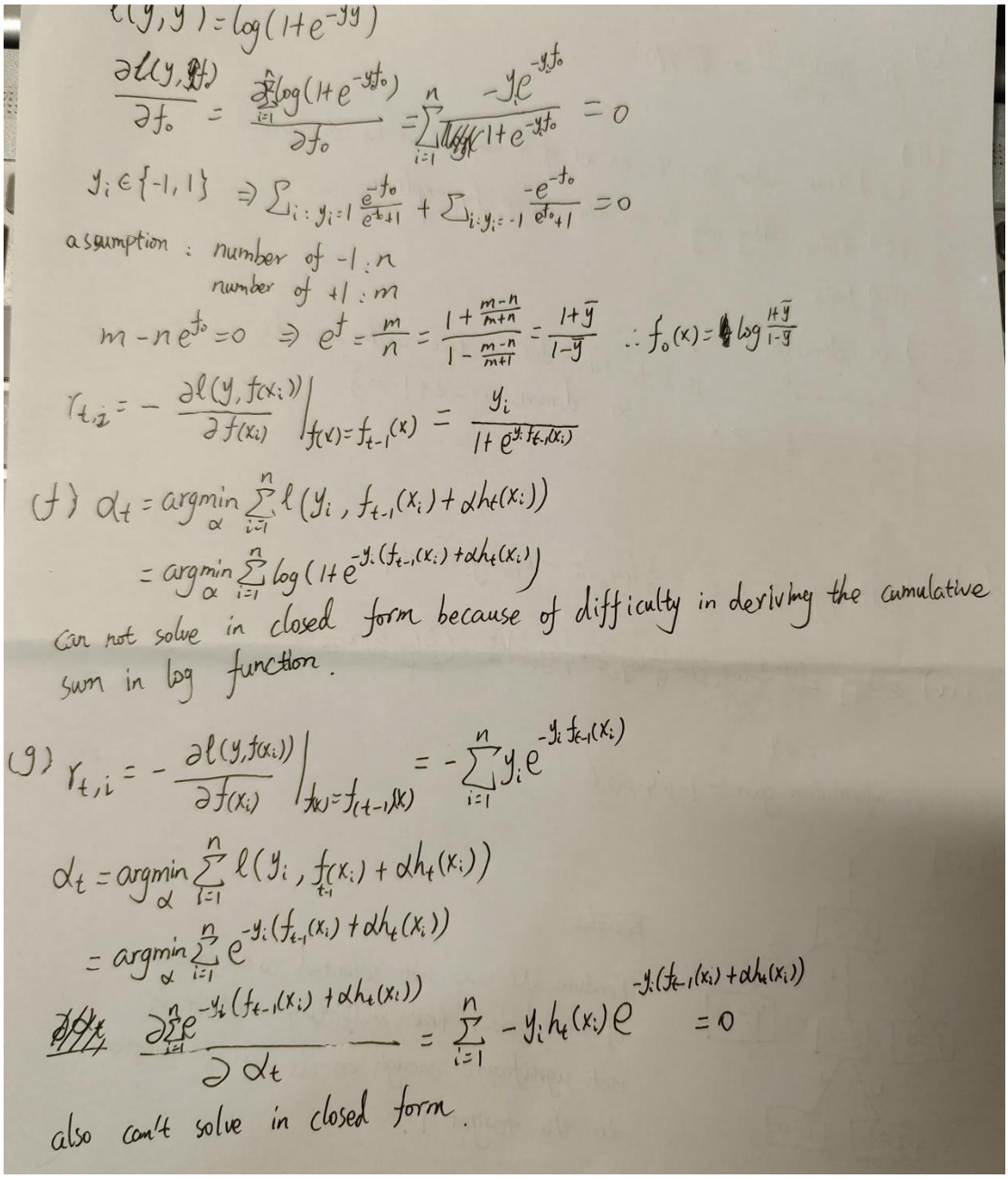


Overfitting occurs in models with adaptive step sizes, while models with fixed step sizes are relatively more consistent with the true function.

```
fig, ax = plt.subplots(5,2)
for i, ax in enumerate(ax.flat):
    n_estimators = 5*(i+1)
    learning_rate = 0.1
    H0 = np.average(y)
    H = np.ones(X.shape[0]) * H0
    estimators = []
    for k in range(n_estimators):
        y_hat = y - H
        dt = DecisionTreeRegressor(max_depth=2)
        dt.fit(X, y_hat)
        y_predict = dt.predict(X)
        H += learning_rate * y_predict
        estimators.append(dt)
    estimators = np.array(estimators)
    xx = np.linspace(0, 1, 1000)
    ax.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
    ax.scatter(X, y, marker='x', color='blue', label='observed')
    ax.plot(xx, predict(n_estimators, learning_rate, estimators, xx.reshape(-1, 1)), color=
```

```
    title = 'n = '+str(n_estimators)
    ax.set_title(title)
# plt.legend(loc='upper right')
plt.show()

fig, ax = plt.subplots(5,2)
for i, ax in enumerate(ax.flat):
    n_estimators = 5*(i+1)
    H0 = np.average(y)
    H = np.ones(X.shape[0]) * H0
    estimators = []
    learning_rates = []
    for k in range(n_estimators):
        y_hat = y - H
        dt = DecisionTreeRegressor(max_depth=2)
        dt.fit(X, y_hat)
        y_predict = dt.predict(X)
        learning_rate = np.dot(y_predict,y_hat)/np.dot(y_predict,y_predict.T)
```

```
        H += learning_rate * y_predict
        estimators.append(dt)
        learning_rates.append(learning_rate)
    estimators = np.array(estimators)
    xx = np.linspace(0, 1, 1000)
    ax.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
    ax.scatter(X, y, marker='x', color='blue', label='observed')
    ax.plot(xx, predict2(n_estimators, learning_rates, estimators, xx.reshape(-1, 1)), col
    title = 'n = '+str(n_estimators)
    ax.set_title(title)
# plt.legend(loc='upper right')
plt.show()
```



(h)

Set an initial step size and then calculate the loss function for each iteration, using the product of the step size and the difference between the loss functions between the two iterations as the new step size.

Additional calculations include the calculation of the loss function and the multiplicative iterations of the step size.