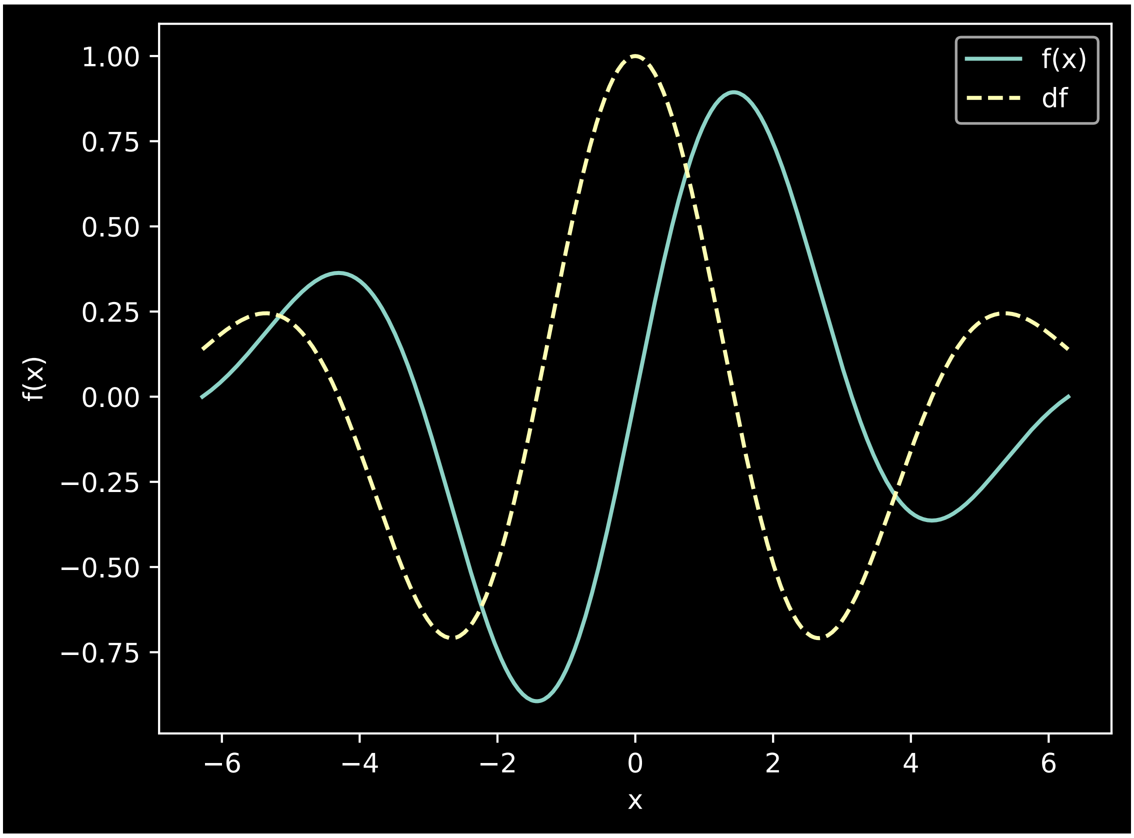
In [17]:
```
# import all necessary modules
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym # sympy to compute the partial derivatives

from IPython import display
display.set_matplotlib_formats('svg')


plt.style.use('dark_background')
```

```
/tmp/ipykernel_4969/4102998519.py:7: DeprecationWarning: `set_matplotlib_f
ormats` is deprecated since IPython 7.23, directly use `matplotlib_inlin
e.backend_inline.set_matplotlib_formats()`
  display.set_matplotlib_formats('svg')
```

In [ ]:

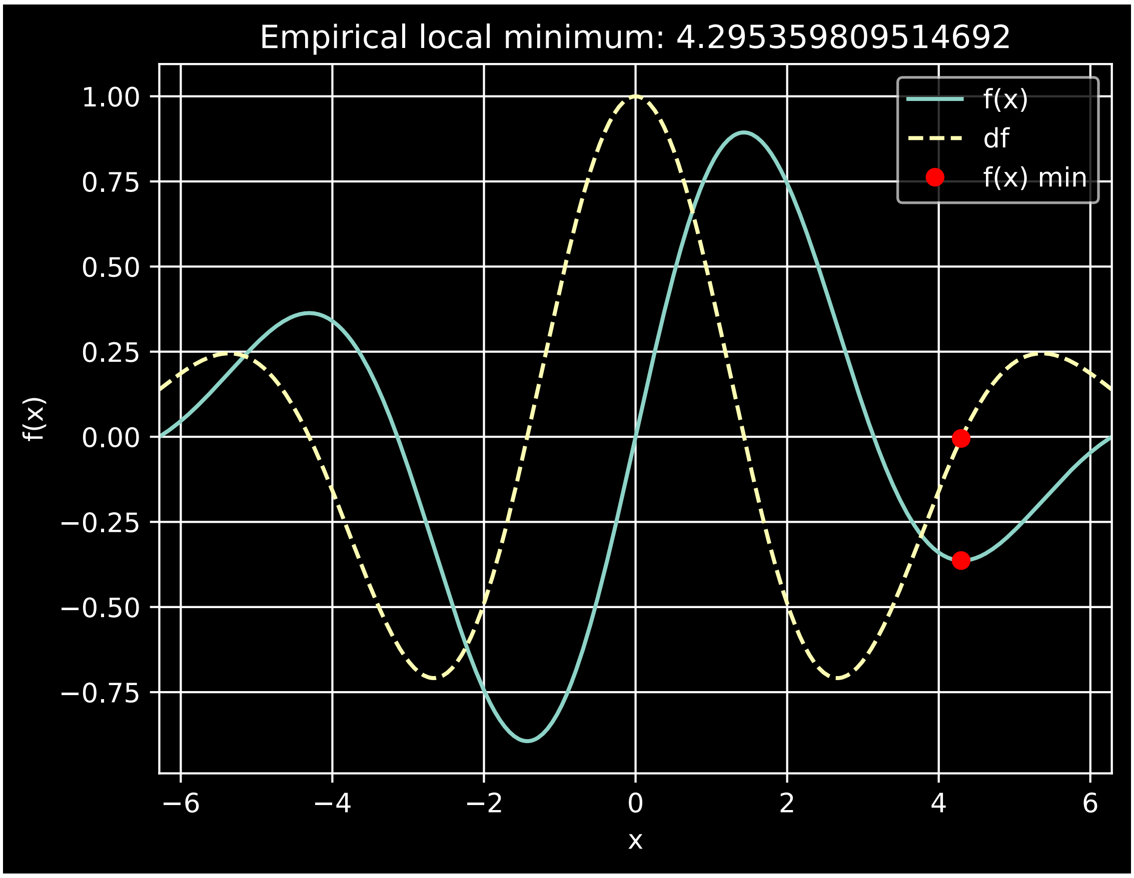
In [18]:
```
# The function (as a function)
x_range = np.linspace(-2 * np.pi, 2 * np.pi, 401)
custom_function = np.sin(x_range) * np.exp(-x_range**2 * 0.05)

# And its derivative
custom_derivative = np.cos(x_range) * np.exp(-x_range**2 * 0.05) - np.sin

# Quick plot for inspection
plt.plot(x_range, custom_function, x_range, custom_derivative, '--')
plt.legend(['f(x)', 'df'])
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



In [19]:
```
# Function (note: over-writing variable names!)
```

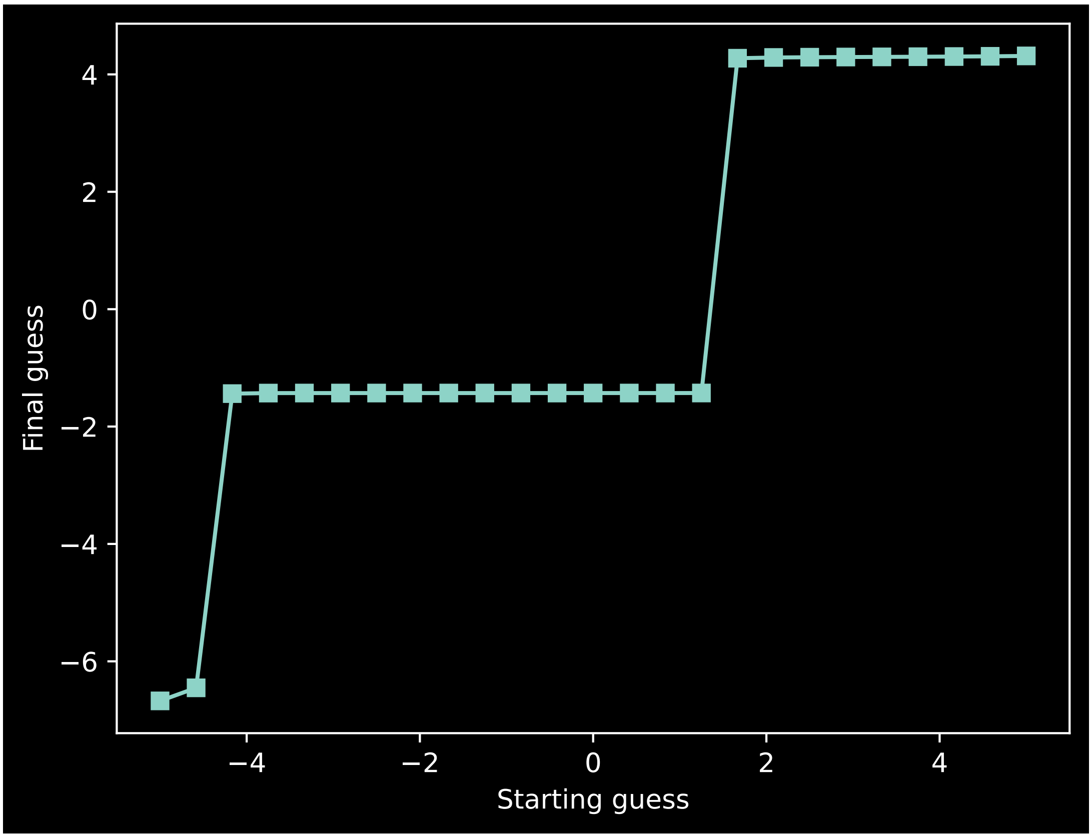
```
def custom_function(x):
    return np.sin(x) * np.exp(-x**2 * 0.05)

# Derivative function
def custom_derivative(x):
    return np.cos(x) * np.exp(-x**2 * 0.05) - np.sin(x) * 0.1 * x * np.ex
```

In [20]:
```
# Random starting point
localmin = np.random.choice(x_range, 1)  # np.array([6])#

# Learning parameters
learning_rate = 0.01
training_epochs = 1000

# Run through training
for i in range(training_epochs):
    grad = custom_derivative(localmin)
    localmin = localmin - learning_rate * grad

# Plot the results
plt.plot(x_range, custom_function(x_range), x_range, custom_derivative(x_
plt.plot(localmin, custom_derivative(localmin), 'ro')
plt.plot(localmin, custom_function(localmin), 'ro')

plt.xlim(x_range[[0, -1]])
plt.grid()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend(['f(x)', 'df', 'f(x) min'])
plt.title('Empirical local minimum: %s' % localmin[0])
plt.show()
```



In [21]:
```
start_locs = np.linspace(-5, 5, 25)
```

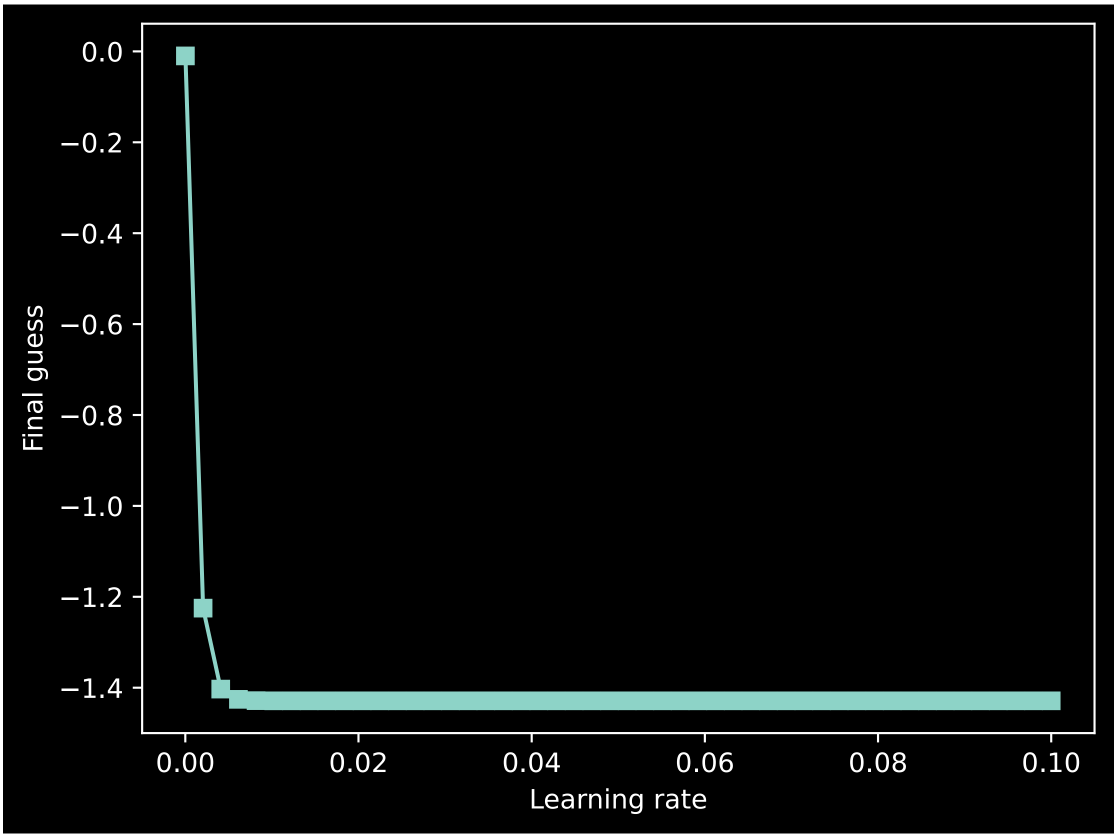
```
final_res = np.zeros(len(start_locs))

# Loop over starting points
for idx, localmin in enumerate(start_locs):

    # Run through training
    for i in range(training_epochs):
        grad = custom_derivative(localmin)
        localmin = localmin - learning_rate * grad

    # Store the final guess
    final_res[idx] = localmin

# Plot the results
plt.plot(start_locs, final_res, 's-')
plt.xlabel('Starting guess')
plt.ylabel('Final guess')
plt.show()
```



In [25]:

```
learning_rates = np.linspace(1e-5, 1e-1, 50)
final_res = np.zeros(len(learning_rates))

# Loop over learning rates
for idx, learning_rate in enumerate(learning_rates):

    # Force starting guess to 0
    localmin = 0

    # Run through training
    for i in range(training_epochs):
        grad = custom_derivative(localmin)
        localmin = localmin - learning_rate * grad

    # Store the final guess
    final_res[idx] = localmin
```

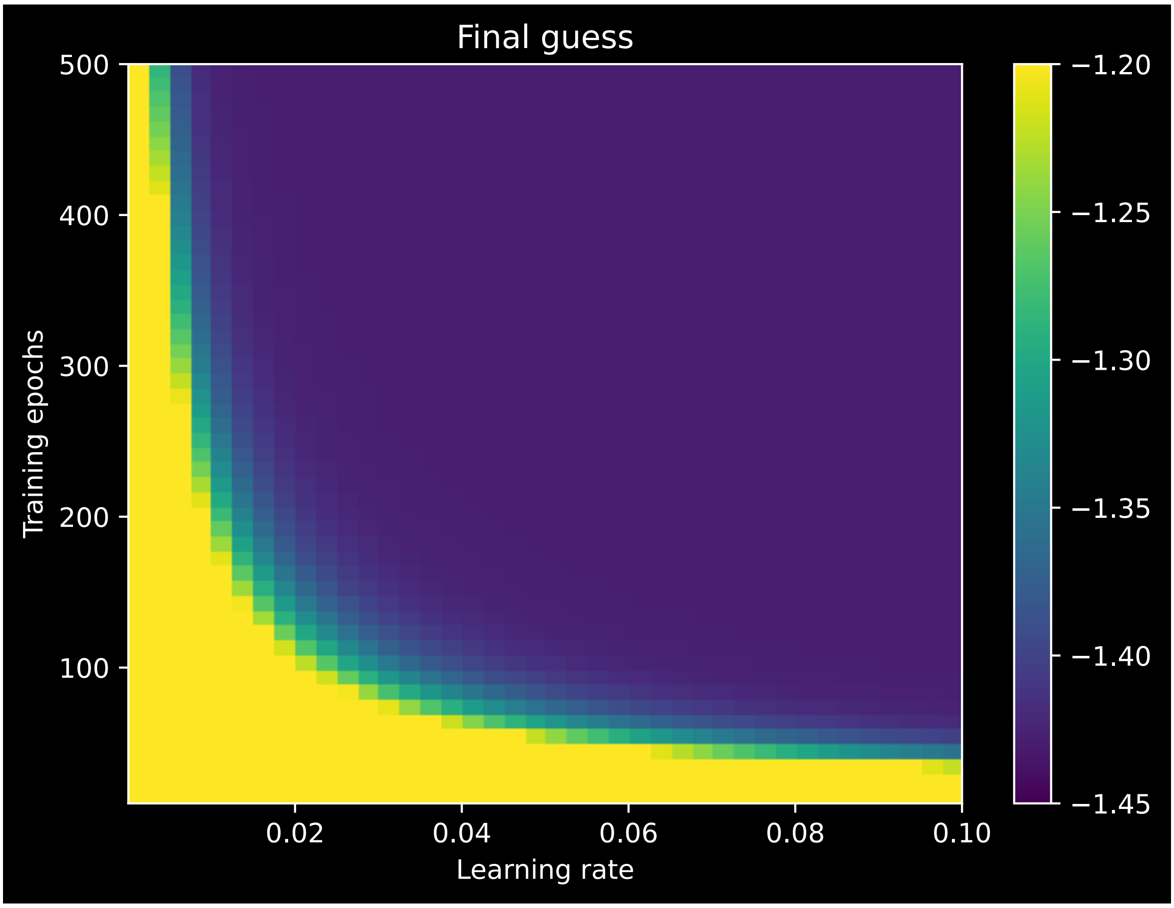
```
plt.plot(learning_rates, final_res, 's-')
plt.xlabel('Learning rate')
plt.ylabel('Final guess')
plt.show()
```

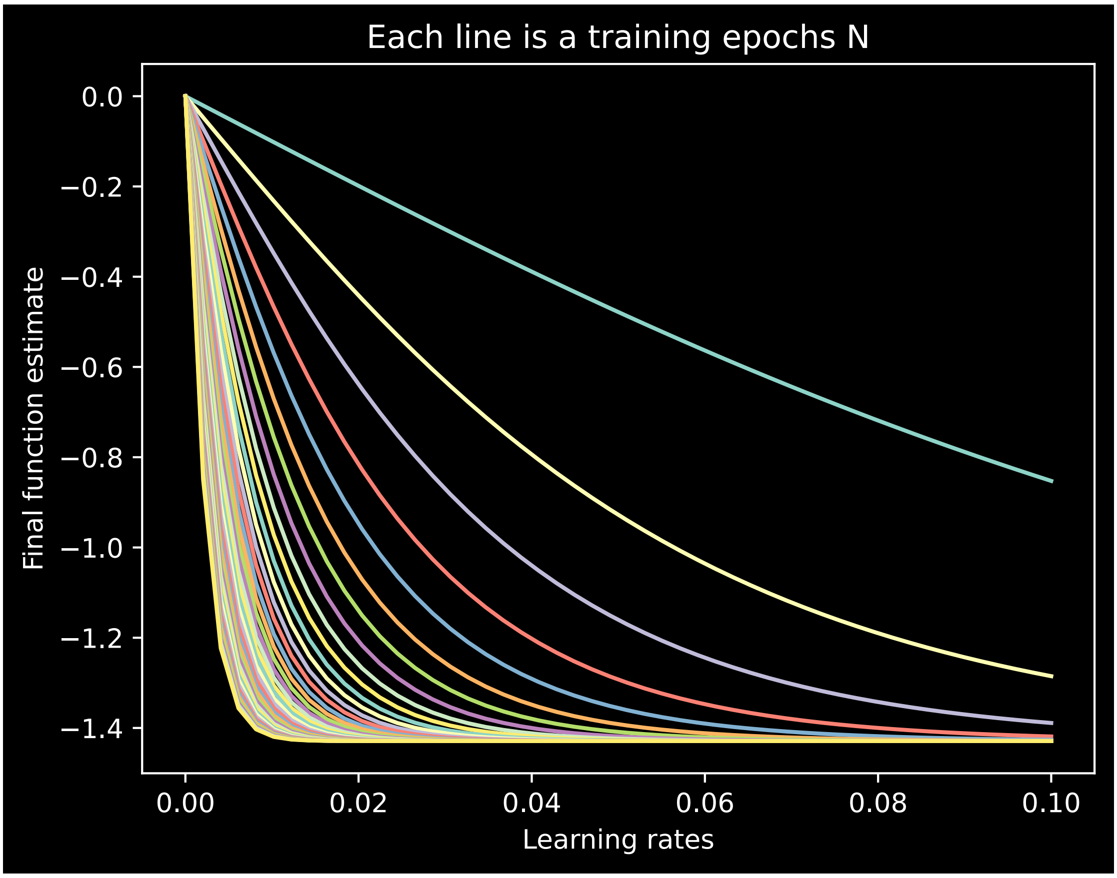


In [7]:
```
# Setup parameters
learning_rates = np.linspace(1e-10, 1e-1, 50)
training_epochs = np.round(np.linspace(10, 500, 40))

# Initialize matrix to store results
final_res = np.zeros((len(learning_rates), len(training_epochs)))

# Loop over learning rates
for Lidx, learning_rate in enumerate(learning_rates):

    # Loop over training epochs
    for Eidx, train_epochs in enumerate(training_epochs):

        # Run through training (again fixing starting location)
        localmin = 0
        for i in range(int(train_epochs)):
            grad = custom_derivative(localmin)
            localmin = localmin - learning_rate * grad

        # Store the final guess
        final_res[Lidx, Eidx] = localmin
```

In [8]:
```
# Cell 8: Plot the Results for Experiment 3

fig, ax = plt.subplots(figsize=(7, 5))

plt.imshow(final_res, extent=[learning_rates[0], learning_rates[-1], trai
           aspect='auto', origin='lower', vmin=-1.45, vmax=-1.2)
```

```
plt.xlabel('Learning rate')
plt.ylabel('Training epochs')
plt.title('Final guess')
plt.colorbar()
plt.show()

# Another visualization
plt.plot(learning_rates, final_res)
plt.xlabel('Learning rates')
plt.ylabel('Final function estimate')
plt.title('Each line is a training epochs N')
plt.show()
```



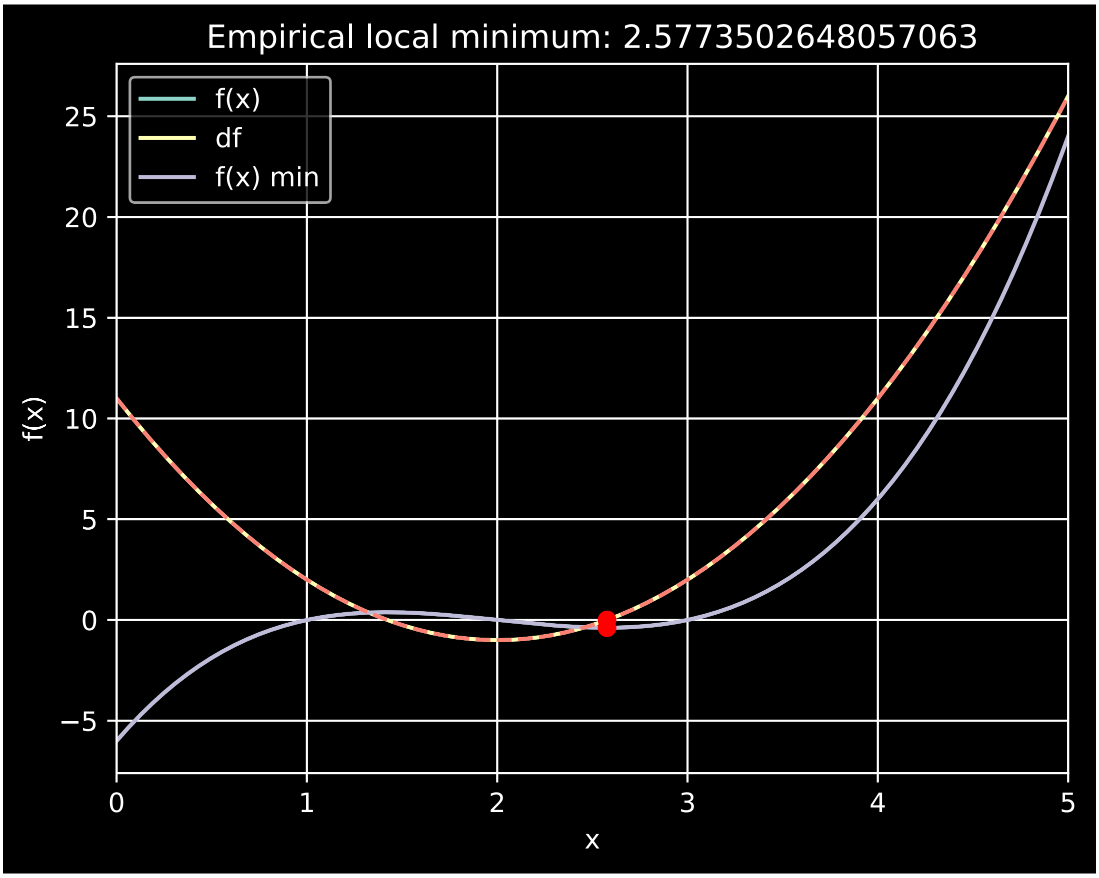


In [ ]:

In [30]:

```
import numpy as np
import matplotlib.pyplot as plt

# Custom function and its derivative
def custom_function(x_range):
    return x_range**3 - 6*x_range**2 + 11*x_range - 6

def custom_derivative(x_range):
    return 3*x_range**2 - 12*x_range + 11

# Cell 2: Define the range for x
x = np.linspace(0, 5, 2001)
fx = custom_function(x)
df = custom_derivative(x)

# Quick plot for inspection
plt.plot(x, fx, x, df)
plt.legend(['f(x)', 'df'])

# Cell 3: Random starting point
local_min = np.random.choice(x, 1)
init_val = local_min[:]  # Store the initial value

# Learning parameters
learning_rate = 0.1
training_epochs = 50

# Run through training
for i in range(training_epochs):
    grad = custom_derivative(local_min)
    local_min = local_min - learning_rate * grad
```

```python
# Plot the results
plt.plot(x, fx, x, df, '--')
plt.plot(local_min, custom_derivative(local_min), 'ro')
plt.plot(local_min, custom_function(local_min), 'ro')

plt.xlim(x[[0, -1]])
plt.grid()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend(['f(x)', 'df', 'f(x) min'])
plt.title('Empirical local minimum: %s' % local_min[0])
plt.show()
```



Summary of Observations

- Having a function with many local minima makes the gradient decent algorithm sensitive to its starting point regarding getting stuck in local minima. The first example shows that a random initial value will determine which minima the final result gets stuck in. The second plot exemplifies this by showing the minimum the function finds for different starting guesses. We see that the final guess varies by quite a bit depending on the starting location.
- The third example shows the effect of diminishing returns of choosing a too small of a learning rate. An extremely small value results in the algorithm never converging on the minimum while the there is no real difference between learning rates between 0.01 and 0.10. If anything a lower learning rate will in most cases slow down training as the delta step for the gradient is too small as shown in the heatmap of experiment 3. The lowest learning rates took 500 iterations to

converge while the higher rates took less than 100. Therefore it is highly important to pick a learning rate that will help you converge quickly and accurately. There is a possibility that if the loss landscape is extremely complex, the large learning rates might never converge with a small amount of epochs. It is all a balancing game given your specific model.