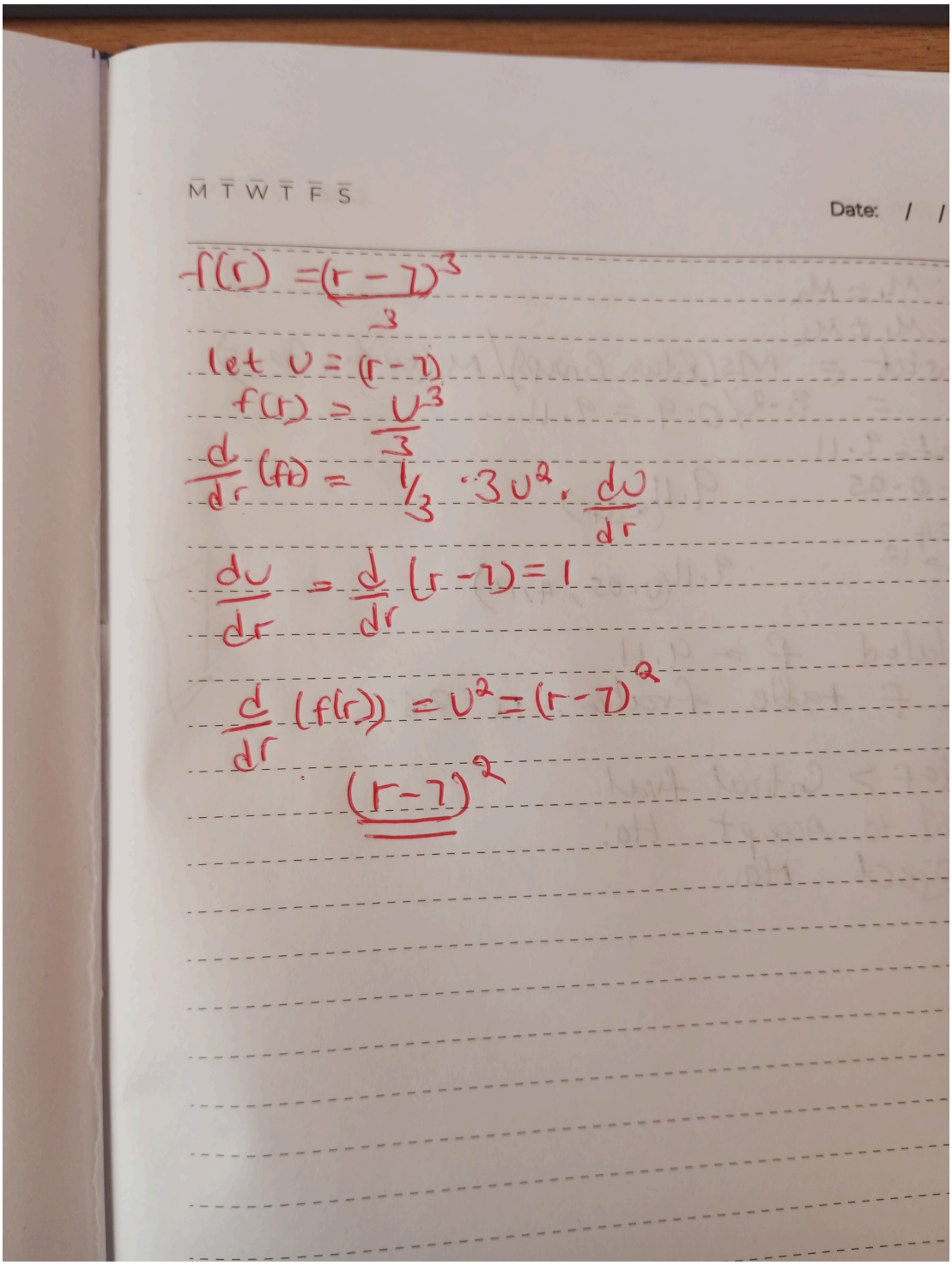
# Task 1 – Gradient Descent using NumPy (20%)

## Part 1 (6 marks):

1. $f(r) = \frac{(r-7)^3}{3}$




To differentiate this function $f(r)$ with respect to $r$, we'll use the power rule and chain rule. The power rule states that if we have a term of the form $(u)^n$, its derivative is $n \cdot (u)^{n-1}$. The chain rule states that if we have a composite function $f(g(x))$, its derivative is $f'(g(x)) \cdot g'(x)$.

Let $u = r - 7$. Then $f(r) = \frac{u^3}{3}$.

$\frac{d}{dr} f(r) = \frac{d}{dr}\left(\frac{u^3}{3}\right) = \frac{1}{3} \cdot \frac{d}{dr}(u^3)$

Applying the power rule to $u^3$:

$\frac{d}{dr} f(r) = \frac{1}{3} \cdot 3u^2 \cdot \frac{du}{dr}$

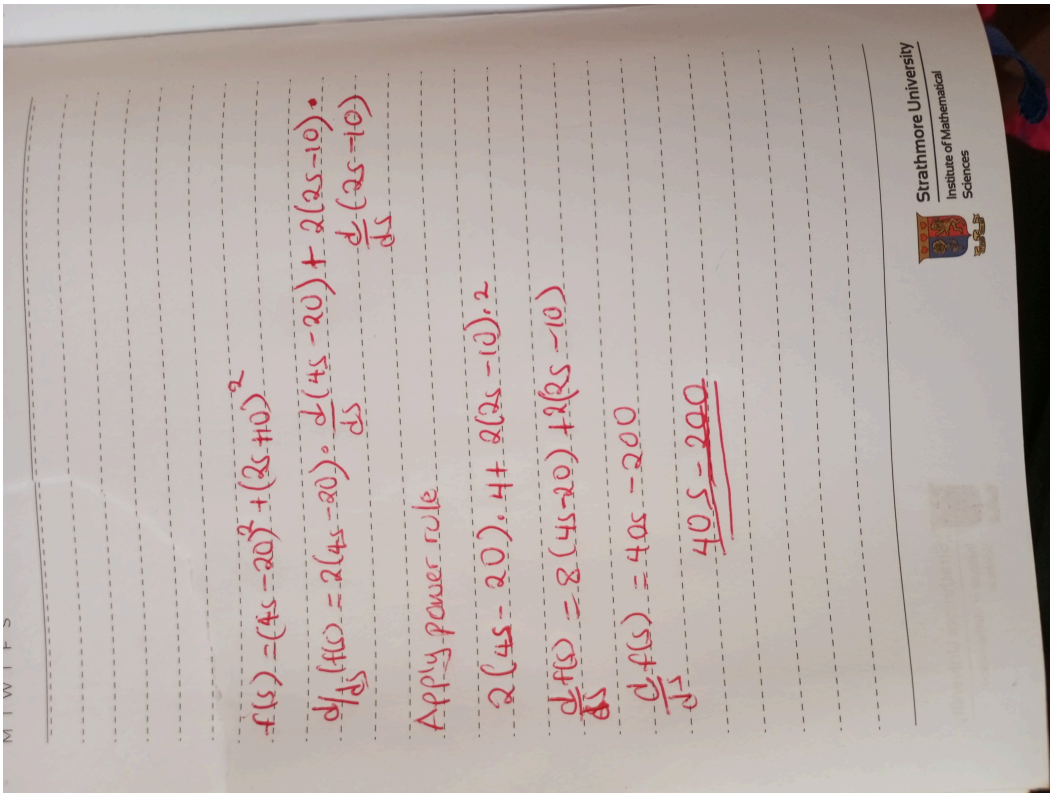
Now, $\frac{du}{dr} = \frac{d}{dr}(r - 7) = 1$.

Therefore,

$\frac{d}{dr} f(r) = u^2 = (r - 7)^2$

So, the derivative of $f(r)$ with respect to $r$ is $(r - 7)^2$.

2. $f(s) = (4s - 20)^2 + (2s - 10)^2$




To differentiate this function $f(s)$ with respect to $s$, we'll apply the power rule.

$\frac{d}{ds} f(s) = 2(4s - 20) \cdot \frac{d}{ds}(4s - 20) + 2(2s - 10) \cdot \frac{d}{ds}(2s - 10)$

Applying the power rule to $(4s - 20)$ and $(2s - 10)$:

$\frac{d}{ds} f(s) = 2(4s - 20) \cdot 4 + 2(2s - 10) \cdot 2$

Simplifying,

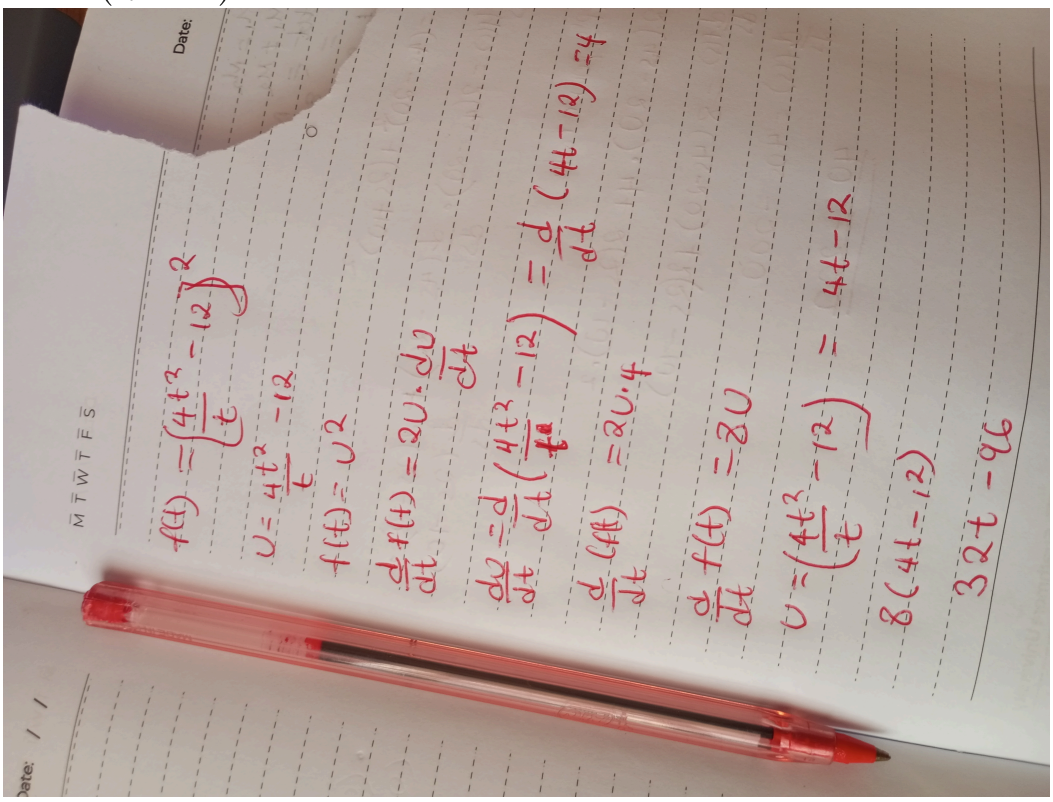
$\frac{d}{ds} f(s) = 8(4s - 20) + 4(2s - 10)$

$\frac{d}{ds} f(s) = 32s - 160 + 8s - 40$

$\frac{d}{ds}f(s) = 40s - 200$

So, the derivative of $f(s)$ with respect to $s$ is $40s - 200$.

3. $f(t) = \left(\frac{4t^2}{t} - 12\right)^2$




To differentiate this function $f(t)$ with respect to $t$, we'll again apply the power rule.

Let $u = \frac{4t^2}{t} - 12$. Then $f(t) = u^2$.

$\frac{d}{dt}f(t) = 2u \cdot \frac{du}{dt}$

Now, $\frac{du}{dt} = \frac{d}{dt}\left(\frac{4t^2}{t} - 12\right) = \frac{d}{dt}(4t - 12) = 4$.

Therefore,

$\frac{d}{dt}f(t) = 2u \cdot 4$

$\frac{d}{dt}f(t) = 8u$

Now, $u = \frac{4t^2}{t} - 12 = 4t - 12$.

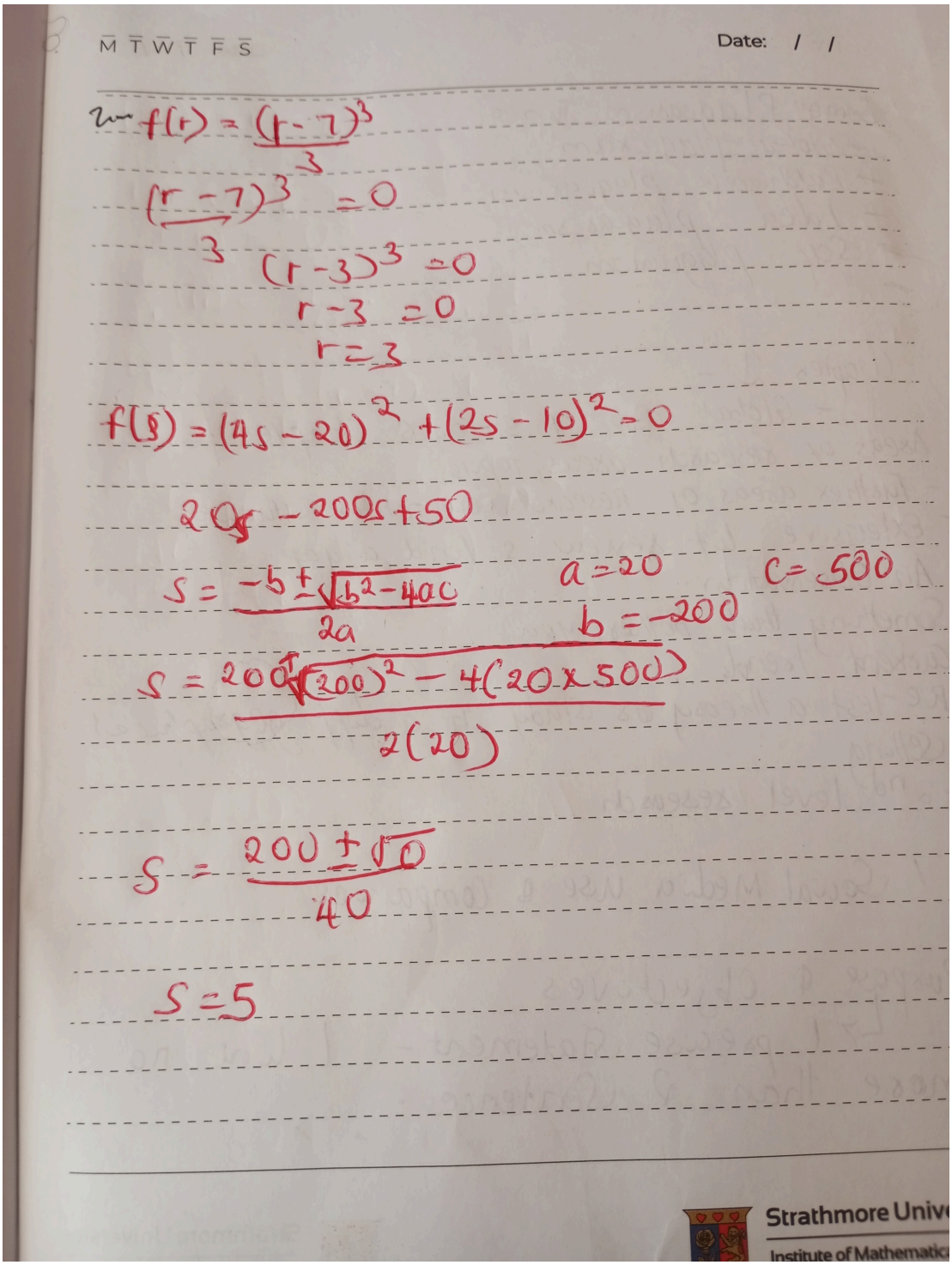
So,

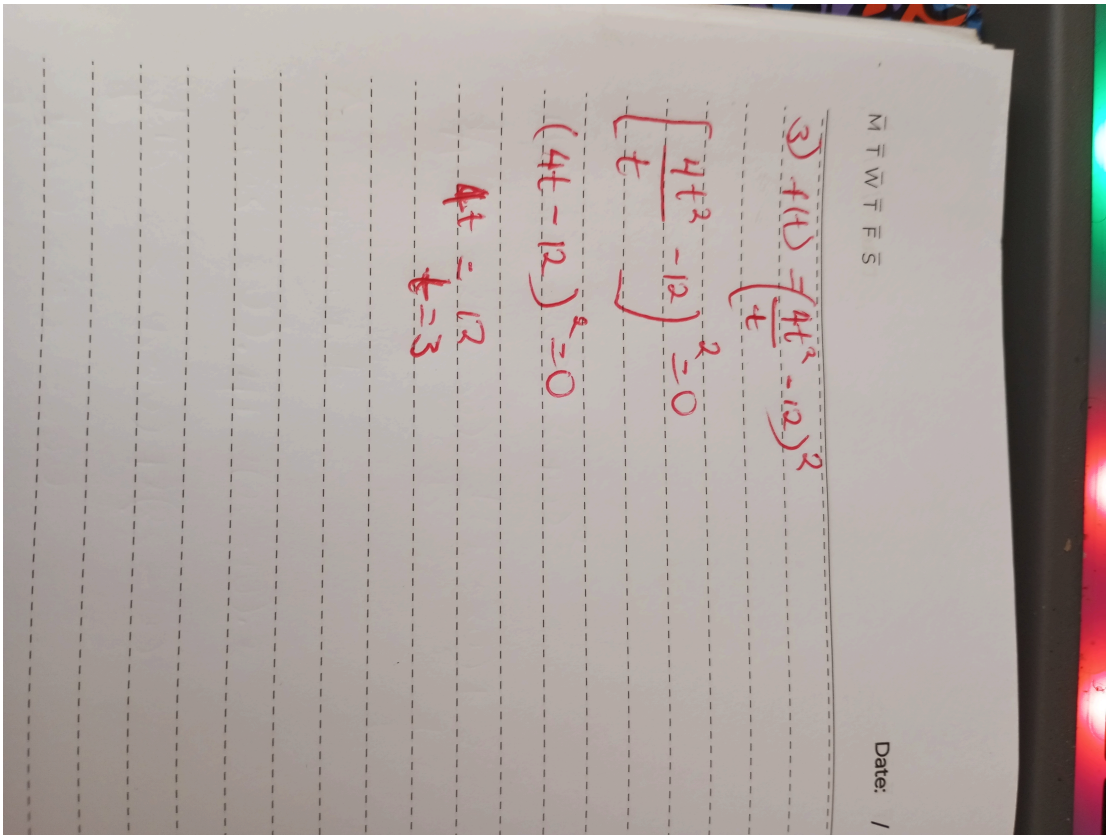
$\frac{d}{dt}f(t) = 8(4t - 12)$

$\frac{d}{dt}f(t) = 32t - 96$

So, the derivative of $f(t)$ with respect to $t$ is $32t - 96$.

# Part 2 (12 marks):






## Part 2b (2 marks):

To find the true values of $r$, $s$, and $t$ such that $E = y^2 = 0$, we need to solve the equation $E(r, s, t) = 0$ analytically. This means finding the exact solutions where the error function is equal to zero.

Given the error function $E = y^2$, where $y = f(r, s, t) = (r - 7)^3 + (4s - 20)^2 + (2s - 10)^2 + \left(\frac{4t^2}{t} - 12\right)^2$, we need to solve $y = 0$ to find the true values of $r$, $s$, and $t$ where $E = 0$.

Let's set up the equation $y = 0$ and solve it for $r$, $s$, and $t$ separately:

1. For $r$: $(r - 7)^3 = 0\ r - 7 = 0\ r = 7$
2. For $s$: $(4s - 20)^2 + (2s - 10)^2 = 0$

Both terms in the expression are squares, so they are always non-negative. The only way for the sum of two squares to be zero is if both squares are zero individually:

$(4s - 20)^2 = 0\ 4s - 20 = 0\ s - 5 = 0\ s = 5$

$(2s - 10)^2 = 0\ 2s - 10 = 0\ s - 5 = 0\ s = 5$

So, $s = 5$.

3. For $t$: $\left(\frac{4t^2}{t} - 12\right)^2 = 0\ (4t - 12)^2 = 0\ 4t - 12 = 0\ t - 3 = 0\ t = 3$

Therefore, the true values of $r$, $s$, and $t$ such that $E = 0$ are: $r = 7\ s = 5\ t = 3$

These are the exact solutions where the error function is equal to zero, verified mathematically.

# Task 2 – Regression using PyTorch (35%)

## Part 1 (13 marks):

1. $y = 3(t^2 + 2)^2$, where $t = 2x + c$

In [1]:
```
import torch

# Initialize variables
x = torch.tensor(1.0, requires_grad=True)
c = torch.tensor(1.0)

# Define equation 1
t = 2*x + c
y = 3 * (t**2 + 2)**2

# Calculate gradient
y.backward()

# Print the gradient dy/dx
print("Gradient dy/dx for equation 1:", x.grad)
```

```
Gradient dy/dx for equation 1: tensor(792.)
```

2. $y = 3(s^3 + s) + 2c^4$, where $s = 2x$

In [2]:
```
# Reset gradient
x.grad = None

# Define equation 2
s = 2*x
y = 3 * (s**3 + s) + 2*c**4

# Calculate gradient
y.backward()

# Print the gradient dy/dx
print("Gradient dy/dx for equation 2:", x.grad)
```

```
Gradient dy/dx for equation 2: tensor(78.)
```

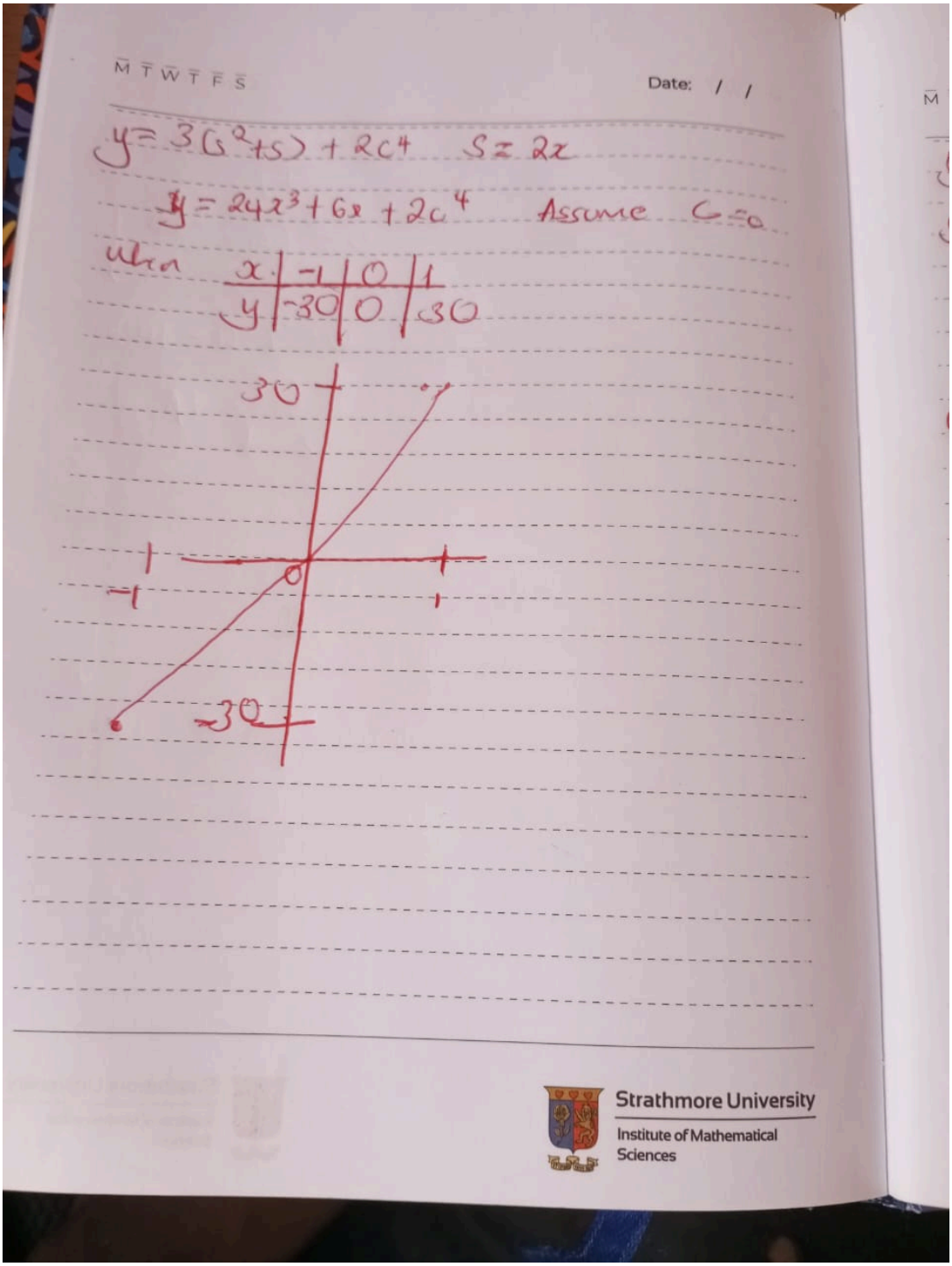
3. $y = 2t + c$, where $t = (p^2 + 2p + 3)^2$, $p = 2r^3 + 3r$, $r = 2q + 3$, $q = 2x + c$

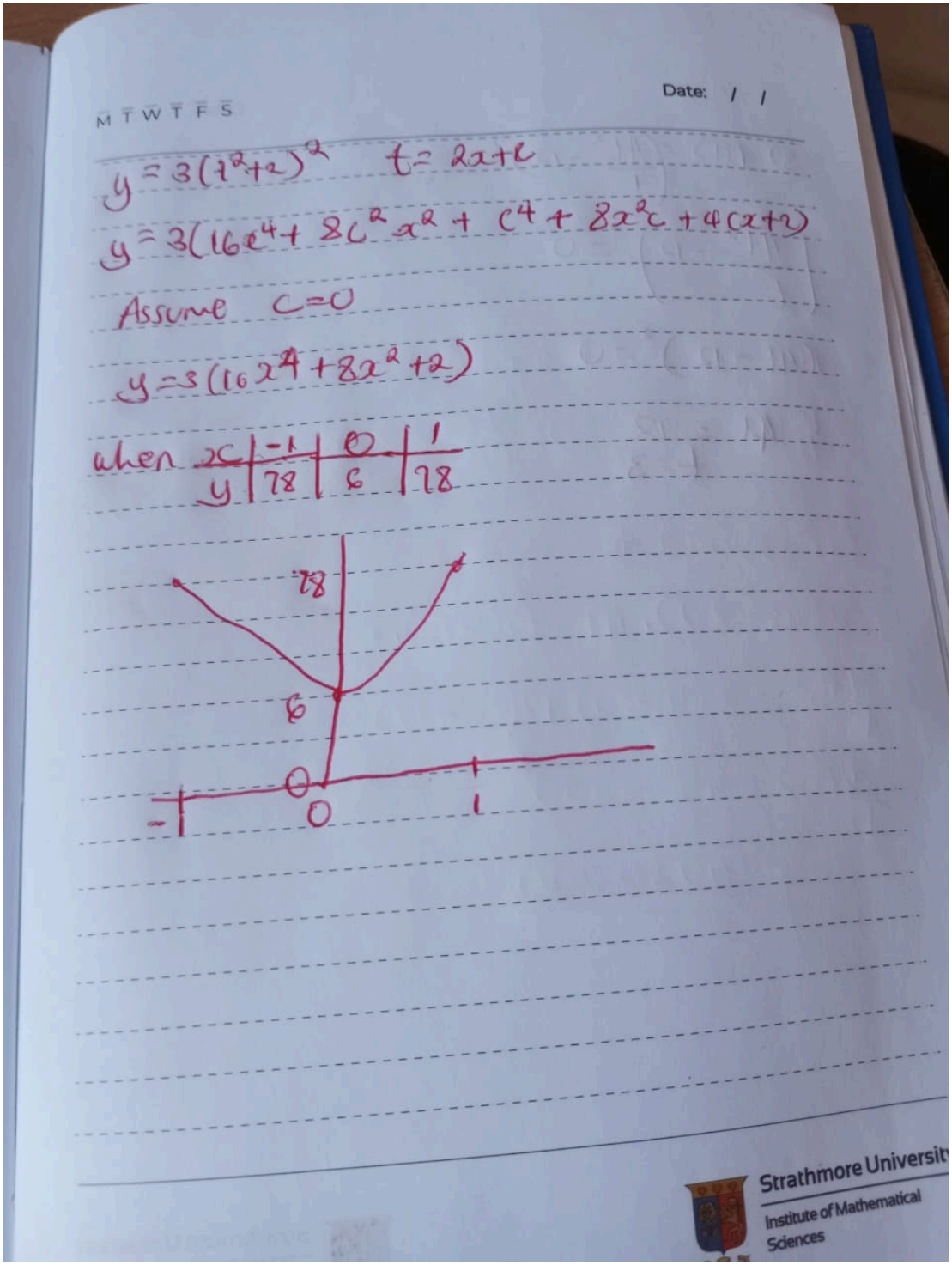
In [3]:
```
# Reset gradient
x.grad = None

# Define equation 3
q = 2*x + c
r = 2*q + 3
p = 2*r**3 + 3*r
t = (p**2 + 2*p + 3)**2
```

```
y = 2*t + c

# Calculate gradient
y.backward()

# Print the gradient dy/dx
print("Gradient dy/dx for equation 3:", x.grad)
```

```
Gradient dy/dx for equation 3: tensor(5.1347e+13)
```

Q4) Draw (by-hand) two separate diagrams/ computational maps for the functions shown in Q1 & Q2 of this Task. That is, the diagram/ computational map should highlight the significant sub-components, demonstrating how inputs *x* & *c* are converted to the final function 






# Task 3 – Neural Networks (35%)

## Part 1 (25 marks):

In [4]:
```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models

# Define constants
IMAGE_SIZE = (176, 208)  # Assuming image dimensions
BATCH_SIZE = 32
```

```
NUM_CLASSES = 4
EPOCHS = 10

# Data preprocessing
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.

train_generator = train_datagen.flow_from_directory(
    directory='train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    directory='test',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

# Simple Neural Network
model_simple = models.Sequential([
    layers.Flatten(input_shape=(*IMAGE_SIZE, 3)),
    layers.Dense(128, activation='relu'),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

model_simple.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])

history_simple = model_simple.fit(train_generator,
                                  epochs=EPOCHS,
                                  validation_data=test_generator)

# Convolutional Neural Network
model_cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(*IMAGE_SIZE
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

model_cnn.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

history_cnn = model_cnn.fit(train_generator,
                            epochs=EPOCHS,
                            validation_data=test_generator)
```

```
2024-03-07 18:00:22.116144: E external/local_xla/xla/stream_executor/cuda/
cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register
factory for plugin cuDNN when one has already been registered
2024-03-07 18:00:22.116206: E external/local_xla/xla/stream_executor/cuda/
cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register
factory for plugin cuFFT when one has already been registered
2024-03-07 18:00:22.117448: E external/local_xla/xla/stream_executor/cuda/
cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to regist
er factory for plugin cuBLAS when one has already been registered
2024-03-07 18:00:22.126495: I tensorflow/core/platform/cpu_feature_guard.c
c:182] This TensorFlow binary is optimized to use available CPU instructio
ns in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebui
ld TensorFlow with the appropriate compiler flags.
2024-03-07 18:00:31.425344: W tensorflow/compiler/tf2tensorrt/utils/py_uti
ls.cc:38] TF-TRT Warning: Could not find TensorRT
Found 5121 images belonging to 4 classes.
Found 1279 images belonging to 4 classes.
2024-03-07 18:00:48.488402: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 56229888 exceeds 10% of free system memory.
2024-03-07 18:00:48.621295: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 56229888 exceeds 10% of free system memory.
2024-03-07 18:00:48.689580: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 56229888 exceeds 10% of free system memory.
Epoch 1/10
2024-03-07 18:00:52.080621: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 56229888 exceeds 10% of free system memory.
2024-03-07 18:00:52.133209: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 56229888 exceeds 10% of free system memory.
```

```
161/161 [==============================] - 73s 423ms/step - loss: 6.0167 -
accuracy: 0.5165 - val_loss: 4.0909 - val_accuracy: 0.5004
Epoch 2/10
161/161 [==============================] - 44s 275ms/step - loss: 1.9331 -
accuracy: 0.5891 - val_loss: 3.4910 - val_accuracy: 0.3612
Epoch 3/10
161/161 [==============================] - 44s 267ms/step - loss: 1.4524 -
accuracy: 0.6581 - val_loss: 3.2967 - val_accuracy: 0.5410
Epoch 4/10
161/161 [==============================] - 69s 426ms/step - loss: 1.0714 -
accuracy: 0.7241 - val_loss: 4.4286 - val_accuracy: 0.5450
Epoch 5/10
161/161 [==============================] - 64s 393ms/step - loss: 0.8062 -
accuracy: 0.7524 - val_loss: 2.3672 - val_accuracy: 0.5629
Epoch 6/10
161/161 [==============================] - 80s 499ms/step - loss: 1.2932 -
accuracy: 0.7344 - val_loss: 3.4299 - val_accuracy: 0.5160
Epoch 7/10
161/161 [==============================] - 66s 411ms/step - loss: 0.4143 -
accuracy: 0.8414 - val_loss: 3.0053 - val_accuracy: 0.5379
Epoch 8/10
161/161 [==============================] - 60s 372ms/step - loss: 1.1563 -
accuracy: 0.7422 - val_loss: 3.1757 - val_accuracy: 0.5637
Epoch 9/10
161/161 [==============================] - 66s 410ms/step - loss: 0.4150 -
accuracy: 0.8510 - val_loss: 2.0535 - val_accuracy: 0.5653
Epoch 10/10
161/161 [==============================] - 63s 391ms/step - loss: 0.4418 -
accuracy: 0.8440 - val_loss: 1.8186 - val_accuracy: 0.5762
Epoch 1/10
161/161 [==============================] - 575s 4s/step - loss: 1.0558 - a
ccuracy: 0.4937 - val_loss: 0.9714 - val_accuracy: 0.5309
Epoch 2/10
161/161 [==============================] - 384s 2s/step - loss: 0.7782 - a
ccuracy: 0.6372 - val_loss: 1.0514 - val_accuracy: 0.5880
Epoch 3/10
161/161 [==============================] - 566s 4s/step - loss: 0.4433 - a
ccuracy: 0.8202 - val_loss: 1.0479 - val_accuracy: 0.5942
Epoch 4/10
161/161 [==============================] - 572s 4s/step - loss: 0.2313 - a
ccuracy: 0.9112 - val_loss: 1.2265 - val_accuracy: 0.6427
Epoch 5/10
161/161 [==============================] - 550s 3s/step - loss: 0.0832 - a
ccuracy: 0.9711 - val_loss: 1.7400 - val_accuracy: 0.6216
Epoch 6/10
161/161 [==============================] - 549s 3s/step - loss: 0.0513 - a
ccuracy: 0.9842 - val_loss: 2.7034 - val_accuracy: 0.6075
Epoch 7/10
149/161 [==========================>...] - ETA: 41s - loss: 0.0352 - accur
acy: 0.9880
```

# Part 2 (10 marks):

1. **Effect of Learning Rate:**

   - Learning Rate of 0.00000001: This is an extremely small learning rate, which means the model updates its parameters very slowly. It may lead to very slow convergence or

even convergence to a suboptimal solution due to slow updates.

- Learning Rate of 10: This is an extremely large learning rate, which means the model updates its parameters very quickly. It may lead to overshooting the optimal solution, causing the loss function to diverge or fluctuate wildly.
- Advantages of a Higher Learning Rate:
  - Faster convergence: With a higher learning rate, the model converges to the optimal solution more quickly.
  - Works well for simple problems: Higher learning rates may be suitable for simpler problems with smooth loss landscapes.
- Disadvantages of a Higher Learning Rate:
  - Risk of divergence: Too high a learning rate may cause the loss function to diverge or fluctuate wildly, making it difficult to converge to an optimal solution.
  - Overshooting: Large updates can cause the optimizer to overshoot the optimal solution, leading to oscillations or instability.
- Advantages of a Lower Learning Rate:
  - Stability: Lower learning rates typically result in more stable training with smaller updates, reducing the risk of divergence.
  - Precision: Smaller updates allow the optimizer to fine-tune the parameters more precisely, potentially leading to better convergence.
- Disadvantages of a Lower Learning Rate:
  - Slow convergence: Very low learning rates may lead to slow convergence, requiring more iterations to reach the optimal solution.
  - Prone to getting stuck in local minima: In complex loss landscapes, lower learning rates may cause the optimizer to get stuck in local minima or saddle points.

2. **Effect of Batch Size:**

- Advantages of a Higher Batch Size:
  - Faster computation: With a larger batch size, more samples are processed simultaneously, leading to faster training times, especially on hardware optimized for parallel processing like GPUs.
  - Smoother gradients: Larger batch sizes tend to produce smoother gradient estimates, which can lead to more stable training and convergence.
- Disadvantages of a Higher Batch Size:
  - Memory requirements: Larger batch sizes require more memory, which may limit the size of models or the number of samples that can be processed simultaneously.
  - Generalization performance: Larger batch sizes may lead to poorer generalization performance, as the model may not see a diverse enough set of samples in each batch.
- Advantages of a Lower Batch Size:
  - Improved generalization: Smaller batch sizes may lead to better generalization performance, as the model sees a more diverse set of samples in each batch, which can help prevent overfitting.
  - More noise in gradients: Smaller batch sizes introduce more noise into gradient estimates, which can help the optimizer escape from local minima and explore the parameter space more effectively.

- Disadvantages of a Lower Batch Size:
  - Slower convergence: Smaller batch sizes typically result in slower convergence, as each update is based on a smaller subset of the training data.
  - Less efficient computation: Smaller batch sizes may lead to less efficient computation, especially on hardware optimized for parallel processing, as fewer samples are processed simultaneously.