ZADANIE 1

```
import math

def fun(x):
  return math.sin(0.4 * math.pow(x, 2) - 2.1) / math.sqrt(1.1 * x + 0.3)

def trapezy():
  a = float(input("poczatek przedzialu: "))

  b = float(input("koniec przedzialu: "))
  n = int(input("n: "))
  dx = (b - a) / n

  w = 0
  for i in range(1, n):
    w += (fun(a + (i * dx ))) * dx
  w += (fun(a)/2 * dx) + (fun(b)/2 * dx)

  return w

def simpson():
  a = float(input("poczatek przedzialu: "))
  b = float(input("koniec przedzialu: "))
  n = int(input("n: "))
  tmp1 = (b - a) / n

  dx = (b - (b - tmp1)) / 2
  t = 0
  w = 0
  a1 = a
  a2 = a
  for i in range(0, n):

    tmp = a2 + dx
    t += 4 * fun(tmp)
    a2 += tmp1
  for i in range(1, n):
    tmp2 = a1 + tmp1
    w += 2 * fun(tmp2)
    a1 += tmp1
  w = (dx / 3) * (fun(a) + fun(b) + t + w)
  return w
def prostokaty():
  a = float(input("poczatek przedzialu: "))
  b = float(input("koniec przedzialu: "))
  n = int(input("n: "))
  dx = (b - a) / n
  w = 0

  for i in range(1, n+1):
    w += fun(a + i * dx)
  w *= dx
  return w
```

```
M_trapezow()
```

```
poczatek przedzialu: 1
koniec przedzialu: 500
n: 20000
-0.375638897691601
```

```
M_prostokatow()
```

```
poczatek przedzialu: 1
koniec przedzialu: 19
n: 1000000
-0.38849181905544694
```

```
M_simpsona()
```

```
poczatek przedzialu: 1
koniec przedzialu: 10
 n: 45
-0.43375448419940327
```

Przy wiekszych wartosciach n, program liczy szybciej

ZADANIE 2

f(x)=(x−2)(x+1)(x−4)

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
```

```
model = tf.keras.Sequential()
model.add(keras.layers.Dense(units=200, input_shape=[1]))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dense(units=160))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dense(units=45))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dense(units=1))
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-1), loss='mean_squared_error', metrics=['mean_squared_error'])
```

```
xs = np.array([-3.0, -2.0, -1.0, 0.0, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0, 7.0,], dtype=float)
ys = np.array([-70.0, -24.0, 0.0, 8.0, 6.0, 3.725, 0.0, -2.625, -4.0, 0.0, 18.0, 56.0, 120.0], dtype=float)
```

```
model.fit(xs, ys, epochs=1400)
```

```
1/1 [==============================] - 0s 11ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1373/1400
1/1 [==============================] - 0s 10ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1374/1400
1/1 [==============================] - 0s 11ms/step - loss: 0.0270 - mean_squared_error: 0.0270
```

```
Epoch 1386/1400
1/1 [==============================] - 0s 11ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1387/1400
1/1 [==============================] - 0s 9ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1388/1400
1/1 [==============================] - 0s 21ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1389/1400
1/1 [==============================] - 0s 15ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1390/1400
1/1 [==============================] - 0s 9ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1391/1400
1/1 [==============================] - 0s 11ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1392/1400
1/1 [==============================] - 0s 8ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1393/1400
1/1 [==============================] - 0s 11ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1394/1400
1/1 [==============================] - 0s 8ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1395/1400
1/1 [==============================] - 0s 7ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1396/1400
1/1 [==============================] - 0s 10ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1397/1400
1/1 [==============================] - 0s 15ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1398/1400
1/1 [==============================] - 0s 8ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1399/1400
1/1 [==============================] - 0s 5ms/step - loss: 0.0270 - mean_squared_error: 0.0270
Epoch 1400/1400
1/1 [==============================] - 0s 6ms/step - loss: 0.0270 - mean_squared_error: 0.0270
<tensorflow.python.keras.callbacks.History at 0x7f6ee3add630>
```

```
print(model.predict([7.0]))
```

```
[[119.99983]]
```

liczba epochow na 1400

wynik dla 7 to w przyblizeniu 120 wiec jest okej, dla wiekszych liczb wynik jest mniej doladny

```
print(model.predict([9.0]))
```

```
[[248.43904]]
```

podczas uzycia klastrow gpu liczenie jest kilka sekund szybsze

ZADANIE 3

```python
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train, y_train), (X_val, y_val) = fashion_mnist.load_data()

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, add, Dense

model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train,
                    y_train,
                    epochs=10,
                    verbose=1,
                    batch_size = 256,
                    validation_split = 0.2
                   )

import numpy as np

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


predicted = model.predict(np.expand_dims(X_val[0],0))
class_names[np.argmax(predicted)]
```

środowisko keras pozwolilo stworzyc siec neuronowa, ktora potrafi rozpoznac czesc ubioru i podac jej nazwe