Computational Essay – Darstellung und Ermittlung des Energieeintrags beim Autoklavieren

I. Beschreibung des realen Problems

Autoklaven sind Geräte, die zum Sterilisieren von Materialien genutzt werden. Sie werden zum Beispiel in der Medizin oder der Lebensmittelindustrie verwendet. Hier ist es wichtig, dass alle Organismen, wie Bakterien, Sporen oder Pilze, abgetötet werden. Es ist wichtig, dass die Temperatur einen Mindestwert, dies wird auch Temperaturprofil genannt, überschreitet und über einen gewissen Zeitraum dort bleibt, damit die Sterilisation erfolgreich abgeschlossen wird.

Um das Temperaturprofil bewerten zu können, wird ein Verfahren genutzt, bei dem die Fläche unter der Kurve betrachtet wird. Hiermit kann man den Prozess quantitativ bewerten, der Temperaturübertragung auf das Material. Das Ermitteln am Ende, das Autoklaviersprosses kann sich als schwierig erweisen.

Ziel unseres Projektes war es, eine Idee zu entwickeln, um den Sterilisationsprozess mit Autoklaven zu automatisieren und zu vereinfachen.

Forschungsaufträge

Die beiden zentralen Forschungsaufträge, die wir im Rahmen unseres Projektes erforschen sollen, lauten:

- Ein Verfahren zur Reduktion der Datenmenge zu entwickeln, bei dem keine relevanten Daten verloren gehen dürfen.
- Einen Algorithmus zu entwickeln, welcher vollautomatisch den Anfang und das Ende des Sterilisationsintervalles findet.

II. Die Daten

Die Daten lagen zum einen als Graph, als auch in Tabellen vor. Im Graphen sind mehrere verschieden Funktionsgraphen eingezeichnet. Bei allen Datensätzen ist die Produkttemperatur gegeben, teilweise auch der Druck und die Kammertemperatur der Autoklave. In der Tabelle sind die wichtigsten Zeitpunkte der Sterilisation eingetragen. Das ist sowohl das Vorbereiten der Kammer, wie auch der Sterilisationsprozess und das Abkühlen im Nachhinein. Zu jedem Zeitpunkt sind die Werte von allen Funktionsgraphen aufgeführt.

Ein Auszug aus der Tabelle und dem dazugehörigen Graphen könnte wie folgt aussehen:

Der oben gezeigte Datensatz besteht aus:

- drei Funktionsgraphen -> Temperatur des Produktes, Temperatur der Kammer und Druck
- Tabelle mit den verschiedenen Daten

Imports

Um die Daten auswerten zu können, haben wir verschiedene Bibliotheken genutzt, um die gegebenen Datensätze zunächst zu importieren.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as p
import copy
#from Tkinter import *
```

Laden der Daten

Hier werden die Daten in das Jupyter Notebook geladen. Wir lesen die Daten als sogenannte **npy-Dateien** ein. Es liegen zum einen Daten vor, die zur Entwicklung des Empfehlungssystems eingesetzt werden können. Diese Daten werden **Datensätze** genannt. Zudem liegen Daten vor, die zum Testen des Empfehlungssystems eingesetzt werden können. Diese bezeichnen wir als **Datensätze**.

In [3]:
```
def str_to_fl(inp):
    return float(inp.replace(",", "."))

def str_to_num(inp):
    return p.dates.datestr2num(inp.split(" ")[1])

data1 = pd.read_csv("Zirbus - Datensatz 1.csv", sep=';', skiprows=0, converters=
data2 = pd.read_csv("Zirbus - Datensatz 2.csv", sep=';', skiprows=0)
```

Um zu überprüfen, ob das Einlesen der Daten funktioniert hat, lassen wir uns den Datensatz ausgeben einmal als Tabelle und als Graph:

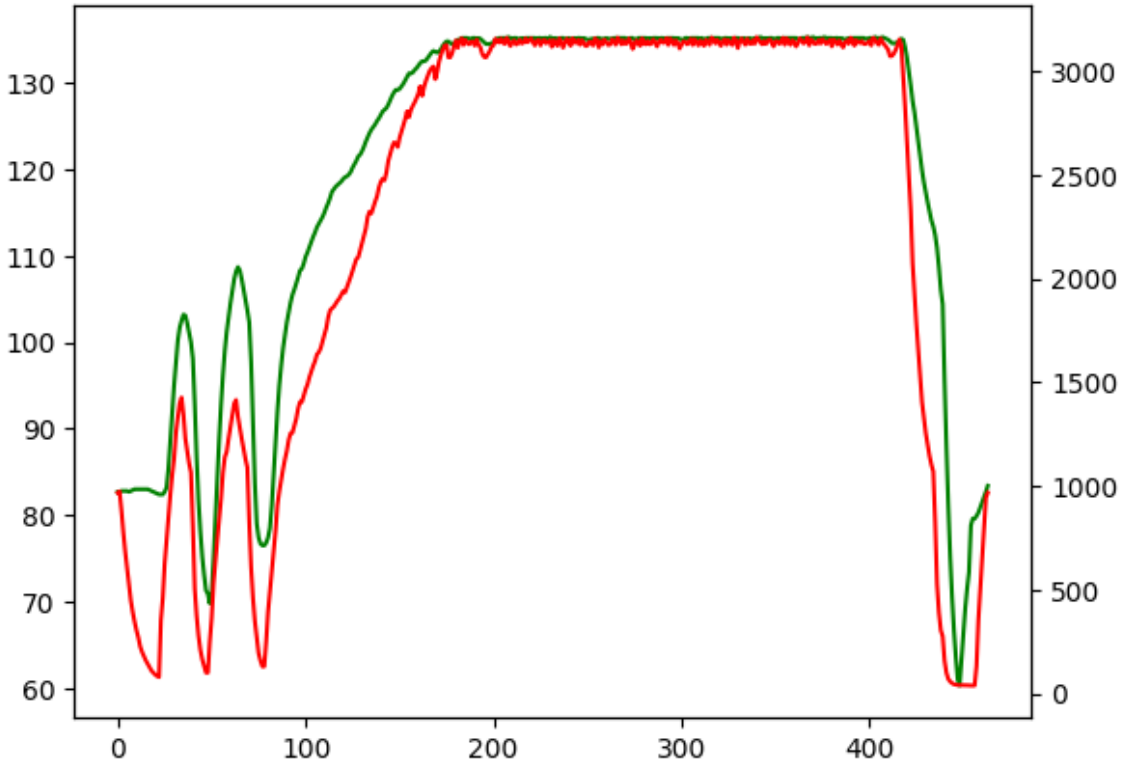
In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as p
import copy
#from Tkinter import *
```

In [12]:
```
print(data1)
```

```
     DatumUhrzeit  TempProdukt  TempFilter  TempKammer  Tempfrei  DruckKammer
\
0    19548.477778         82.7           0           0       NaN          968
1    19548.477778         82.7           0           0       NaN          974
2    19548.477778         82.8           0           0       NaN          883
3    19548.477778         82.8           0           0       NaN          777
4    19548.477778         82.8           0           0       NaN          684
..            ...          ...         ...         ...       ...          ...
459  19548.506250         80.9           0           0       NaN          502
460  19548.506250         81.5           0           0       NaN          650
461  19548.506944         82.1           0           0       NaN          798
462  19548.506944         82.7           0           0       NaN          944
463  19548.506944         83.4           0           0       NaN          968

     Schritt
0         20
1         20
2         20
3         20
4         20
..       ...
459       65
460       65
461       65
462       79
463       79

[464 rows x 7 columns]
```

In [6]:
```
fig, ax1 = plt.subplots()

ax1.plot(data1["TempProdukt"], color="g")

ax2 = ax1.twinx()
ax2.plot(data1["DruckKammer"], color="r")

#plt.xticks(data1["DatumUhrzeit"])
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x22853ff3150>]
```



Im Graphen zeigt der grüne Funktionsgraph die Temperatur des Produktes und der rote Funktionsgraph den Druck in der Kammer.

Erste Erkundung der Daten und Datenvorbereitung

Im nächsten Schritt haben wir den vorher entwickelten Algorithmus angewendet, um Datenpunkte zu reduzieren. Anschließend haben wir die beiden Funktionsgraphen übereinander gelegt, um die Abweichung sichtbar zu machen. In dem darunter gezeigten Beispiel haben wir das ganze nur mit der Temperaturkurve ausgeführt.

Erkenntnisse aus der Datenerkundung

III. Erster Modellansatz (zu Forschungsauftrag 1)

Vereinfachungen und Modellannahmen

Wir treffen zunächst folgende Annahmen:

1. Wir nehmen an, dass zwischen allen Punkten dieselbe Zeit vergeht. (Dies kann aber rückgängig gemacht werden)
2. Wir nehmen an, dass kleinere Extremstellen in einem monoton wachsendem Abschnitt nicht relevant ist.

Entwicklung eines ersten mathematischen Modells

Um Datenpunkte zu reduzieren haben wir uns drei aufeinanderfolgende Punkte angeschaut, anschließend haben wir die Steigung zwischen dem ersten und zweiten Punkt und zwischen dem zweiten und dritten Punkt berechnet. Nachdem die Steigung berechnet ist, haben wir die Steigungen miteinander verglichen, wenn die beiden Steigungen ähnlich zueinander sind, haben wir den Punkt dazwischen herausgeschmissen. Anschließend wird eine neue Gerade vom ersten zum zweiten Punkt. Wenn die Steigung zu unterschiedlich ist oder es sich um einen Extrempunkt handelt, bleibt der zweite Punkt bestehen und es werden die nächsten Steigungen angeschaut bis die ganzen Punkte auf dem Graphen so angeschaut wurden.

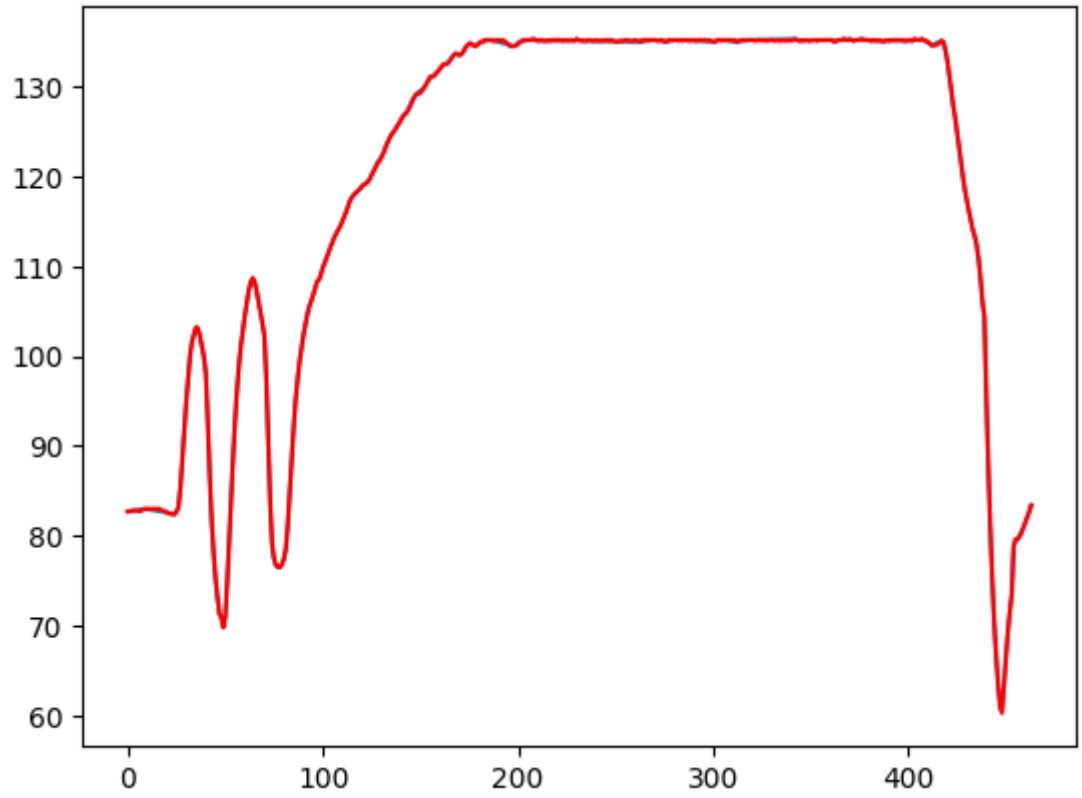
Computergestützte Berechnungen

Mithilfe von Python haben wir unser obengenanntes Modell in einen Algorithmus umgewandelt und in mehreren Anläufen alle Fehler entfernt. In der folgenden Codezelle ist die aufgeführt.

In [7]:
```
curve = data1["TempProdukt"].to_list()
list1 = []
for i in range(len(curve)):
    list2 = [i] + [curve[i]]
    list1.append(list2)

def is_similar(a, b, tolerance):
    if abs(b - a) < tolerance:
        return True
    else:
        return False

def downsize(array, index = 0):
    if(index >= len(array) - 3):
            return array

    gradient1 = array[index][1] - array[index + 1][1]
    gradient2 = array[index + 1][1] - array[index + 2][1]


    if(is_similar(gradient1, gradient2, 0.2)):
        array.pop(index + 1)
    else:
        index += 1

    return downsize(array, index)


downsized_curve = downsize(list1)

x = []
y = []

for i in downsized_curve:
```

```
    x.append(i[0])
    y.append(i[1])

np_x = np.array(x)
np_y = np.array(y)

plt.plot(np_x, np_y)
plt.plot(data1["TempProdukt"], color="r")

print(len(y))
print(len(data1["TempProdukt"]))
```

```
179
464
```



Im Graphen kann man den originalen Funktionsgraph und den Funktionsgraphen mit weniger Datenpunkten erkennen. Nach der Reduzierung sind noch 2,5-mal so wenige Datenpunkte vorhanden.

Validierung der mathematischen Lösung

Um zu bestimmen, wie gut unsere Datenreduktion ist, berechnen wir folgendes Bewertungsmaß. Für jeden Messpunkt berechnen wir den Abstand zu unserer Kurve mit den reduzierten Messdaten. Diese Abstände werden daraufhin addiert und durch die Summe der y Koordinaten der Punkte, der nicht reduzierten Liste geteilt, damit bekommen wir eine relative Abweichung zur der nicht reduzierten Kurve.

Interpretation der Lösung

Das Ergebnis mit den reduzierten Datenpunkten ist relativ gut, man kann nur ein paar Abweichungen sehen. Jedoch ist es eher schwierig, die beiden Kurven voneinander zu unterscheiden.

IV. Zweiter Modellansatz (zu Forschungsauftrag 1)

Modifiziertes Modell

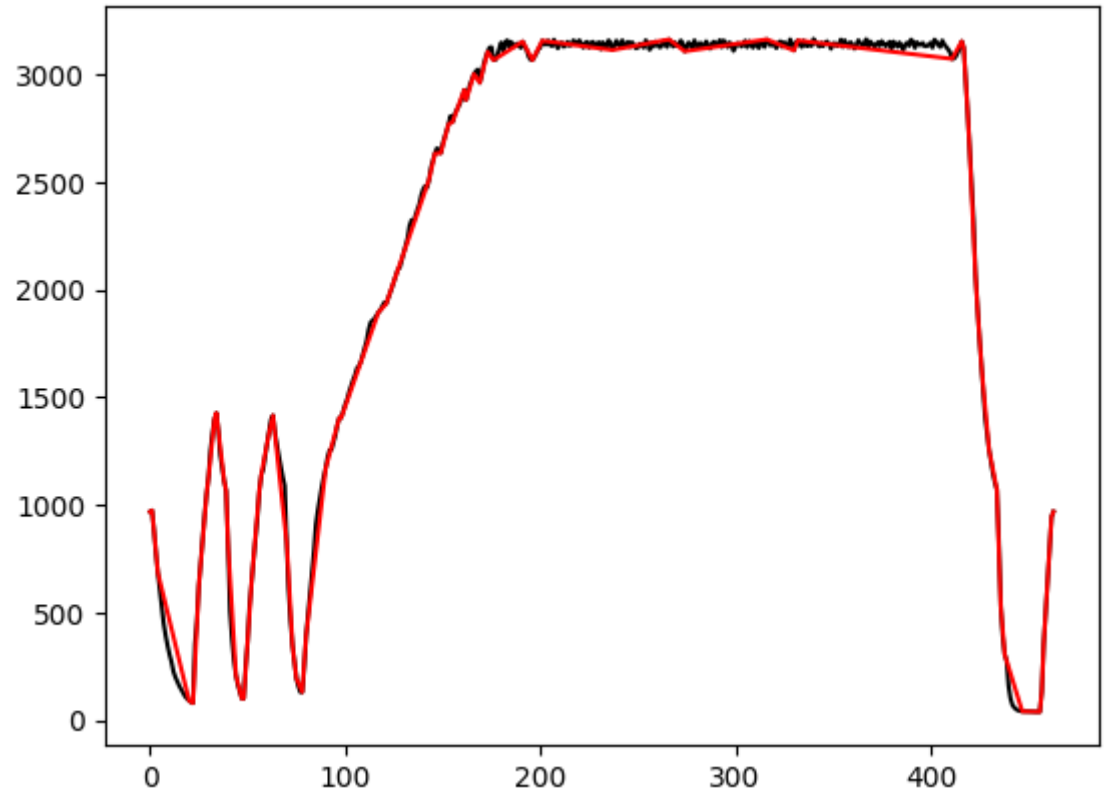
Wir haben an sich kaum etwas verändert. Das Einzige, was wir verändert haben, ist, dass wir versucht haben, die Kurve sichtbarer zu machen. Des Weiteren haben wir die Kurven weiter reduziert und jede Kurve in einen einzelnen Graphen eingetragen, um sie gegenüberzustellen. Das modifizierte Verfahren wird in der folgenden Codezelle umgesetzt.

In [8]:
```
print("Go!")

curve_to_reduce1 = "DruckKammer"

#curve1 = sum(data1.values.tolist(), [])
curve1 = data1[curve_to_reduce1].tolist()

def is_similar(a, b, tolerance):
    if abs(b - a) < tolerance:
        return True
    else:
        return False

strength = 0.1

def anti_interpolate(array, index = 0):
    global strength
    if(index >= len(array) - 3):
            return array

    gradient1 = array[index][1] - array[index + 1][1]
    gradient2 = array[index + 1][1] - array[index + 2][1]


    if(is_similar(gradient1, gradient2, strength)):
        array.pop(index + 1)
    else:
        index += 1

    return anti_interpolate(array, index)

def downsize_curve(curve, max_strength = 1000.0, max_comp = 8.0, snapshots = 5):
    global strength
    snapshots += 1
    ready_curve = []
    for i in range(len(curve)):
        ready_curve.append([i, curve[i]])
```

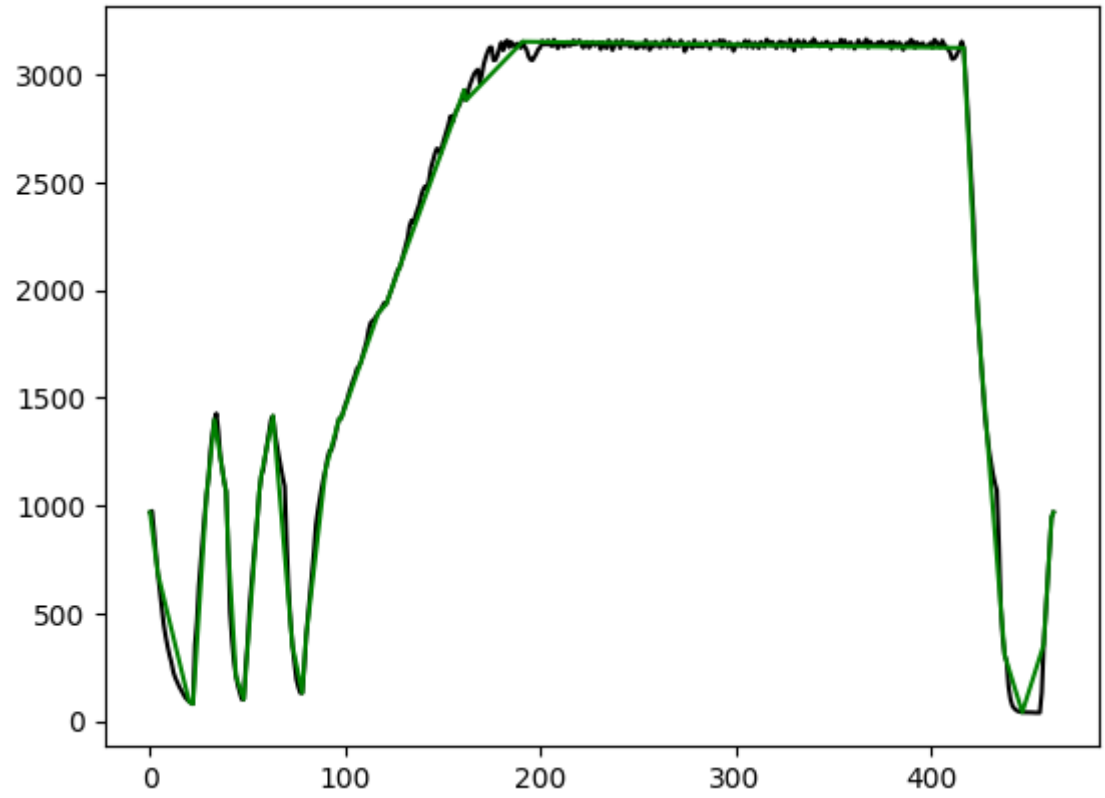
```
    rdcrv_copy = copy.deepcopy(ready_curve)
    downsize_len = len(ready_curve)
    checker = True

    results_x = []
    results_y = []

    index = 1

    while (len(rdcrv_copy) / downsize_len) < max_comp:
        checker = False

        downsized_curve = anti_interpolate(ready_curve)
        downsize_len = len(downsized_curve)
        if(len(rdcrv_copy) / downsize_len) > max_comp * (index/snapshots):
            index += 1
            x = []
            y = []
            for i in downsized_curve:
                x.append(i[0])
                y.append(i[1])
            results_x.append(np.array(x))
            results_y.append(np.array(y))

        if strength < max_strength:
            strength += 0.1
        else:
            break

    return results_x, results_y


    if max_comp > 16:
        print("WARNING: Compression Rates above 16 may provide severely inaccura

    return np.array(x), np.array(y)

def calc_accuracy(curve_x1,curve_y1,original_curve):
    sum_of_deviation = 0
    interpolated_curve_y1  = curve_y1.tolist()
    b = -1
    for i in range(len(curve_x1) - 1):
        difference_x = curve_x1[i+1]-(curve_x1[i]+1)
        for q in range(difference_x):
            b += 1
            difference_y = (curve_y1[i+1]-(curve_y1[i]))/(difference_x+1)
            to_insert = curve_y1[i] + difference_y*(q + 1)
            interpolated_curve_y1.insert(i+b+1,to_insert)

    acc = 0
    for i in range(len(interpolated_curve_y1)):
        acc += abs(original_curve[i] - interpolated_curve_y1[i])**2


    #deviance /= len(interpolated_curve_y1)
    mittelwert = 0

    for i in original_curve:
        mittelwert += i
    mittelwert /= len(original_curve)
```

```
    mittelwert1 = 0
    for i in original_curve:
        mittelwert1 += abs(i-mittelwert)**2


    acc /=  mittelwert1

    return (1-acc)

#curve_x1, curve_y1 = downsize_curve(curve1, max_comp = 2)
#print("Curve1 done!")
#score1 = calc_score(curve_x1,curve_y1,curve1)
#score2 = calc_score(curve_x2,curve_y2,curve2)

curve_x1, curve_y1 = downsize_curve(curve1, max_strength = 1000, max_comp = 100,

colors = ["r", "g", "b"]

#print(len(curve_x2))

#for i in range(len(curve_x1)):
    #plt.plot(curve_x1[i], curve_y1[i], color=colors[i%3])
    #plt.show()
#print(len(data1[curve_to_reduce1]) / len(curve_y1))
#print(len(data1[curve_to_reduce1]), " ", len(curve_y1))
#print(score1)

print("")

for q in range(len(curve_y1)):
    plt.plot(curve1, color="k")
    plt.plot(curve_x1[q], curve_y1[q], color=colors[q%3])

    print("")
    print("Coefficient of determination: {:.3f}".format(calc_accuracy(curve_x1[c
    print("Compression Rate: {:.3f}".format(len(curve1) / len(curve_y1[q])), " (

    #file_name = "Graph_" + str(q)
    #plt.savefig(file_name, dpi=5000)
    plt.show()


#print(len(data1[curve_to_reduce2]), " ", len(curve_y2))
#print(score2)
```

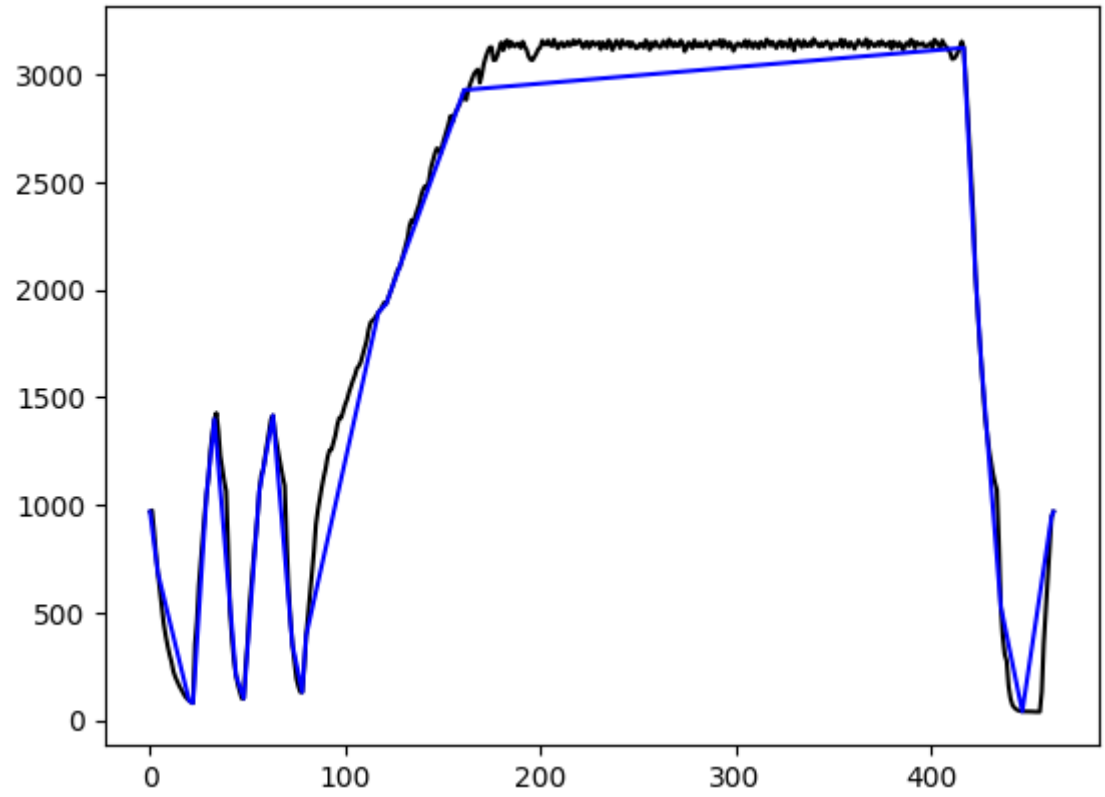
```
Go!


Coefficient of determination: 0.999
Compression Rate: 6.356  ( 464 / 73 )
```



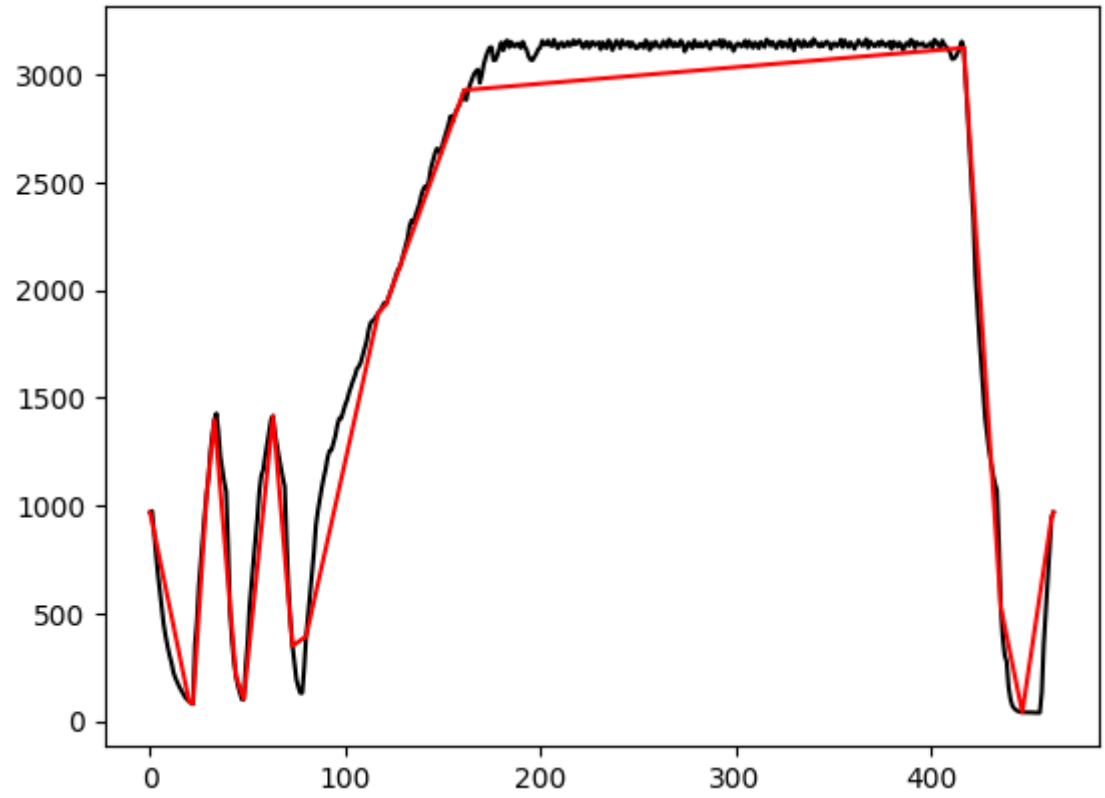

```
Coefficient of determination: 0.997
Compression Rate: 12.541  ( 464 / 37 )
```




```
Coefficient of determination: 0.985
Compression Rate: 19.333  ( 464 / 24 )
```




```
Coefficient of determination: 0.983
Compression Rate: 25.778  ( 464 / 18 )
```



Nun kann man die Graphen klar gegenüberstellen, und den für sie geeigneten Komprimierungsgrad wählen.

Um die Güte der erstellten Kurven vergleichen zu können, verwenden wir das Bestimmtheitsmaß R^2. Dabei ist $\hat{y} $ ein Funktionswert des reduzierten

Graphen und $ \overline{y} $ der Mittelwert der y-Werte der Messdaten. Das Bestimmtheitsmaß errechnet sich dann wie folgt:

$$R^2 = 1-\dfrac{\sum_{i=0}^N (y_i-\hat{y}_i)^2}{\sum_{i=0}^N (y_i-\overline{y})^2}$$

Umso näher der Wert R^2 an 1 ist, desto besser entspricht die reduzierte Kurve den ursprünglichen Messdaten.

III. Erster Modellansatz (zu Forschungsauftrag 2, Teil 1)

Vereinfachungen und Modellannahmen

Wir treffen zunächst folgende Annahmen:

1. Wir nehmen an, dass der Druck im Sterilisationsintervall "flattert".
2. Wir nehmen an, dass die Temperatur im Sterilisationsintervall annähernd gleich bleibt.
3. Wir nehmen an, dass der letzte Hochpunkt auch gleich das Ende der Sterilisation.
4. Wir nehmen an, dass der Mindestdruck und die Mindesttemperatur, damit der Sterilisationsvorgang stattfinden kann, bei jeder Sterilisation eine bekannte Größe ist, da dies Parameter sind, die vor der Sterilisation eingegeben werden müssen.

Entwicklung eines ersten mathematischen Modells

Für unsern Ansatz ist es wichtig, dass wir eine Gerade auf der Höhe der Mindesttemperatur einzeichnen (y=Mindesttemperatur). Hierbei ist klar, dass wir nur die Werte über der Gerade betrachten müssen, da sonst keine Sterilisation stattfinden kann. In den meisten Graphen ist ein Plateau zu erkennen. Der Algorithmus ist dafür ausgelegt, dieses Plateau zu erkennen. In den gegebenen Messdaten existierte immer ein kurzes Intervall, wo die Temperatur über der Mindestschwelle ist, aber nicht mehr Teil des Sterilisationintervalls ist. Die Kürze dieses Intervalls lässt sich dadurch erklären, dass, um das Produkt vor der Rekontamination zu schützen, stark vakuumiert wird, sobald der Sterilisationsvorgang beendet ist. Nach Analyse unserer Daten unterscheidet sich diese Intervalldauer kaum. Wir fassen diese Dauer unter einem konstanten Wert K zusammen, der etwas größer ist, als der größte Intervallsdauer, der von uns bestimmt wurde. Vor dem Sterilisationsintervall können verschiedene Fälle vorkommen:

1. Die Kurve der Temperatur des Produkts steigt über die Mindesttemperatur und die Sterilisation beginnt direkt.
2. Die Kurve überschreitet die Mindesttemperatur und erst kurze Zeit später beginnt die Sterilisation, dies geschieht aufgrund der Abwehrmechanismen in den Mikroorganismen, die dafür sorgen, dass sie eine kurze Zeit lang überleben, trotz Überschreiten der Mindesttemperatur.

3. Die Kurve überschreitet relativ weit bevor Beginn der Sterilisation die Mindesttemperatur.

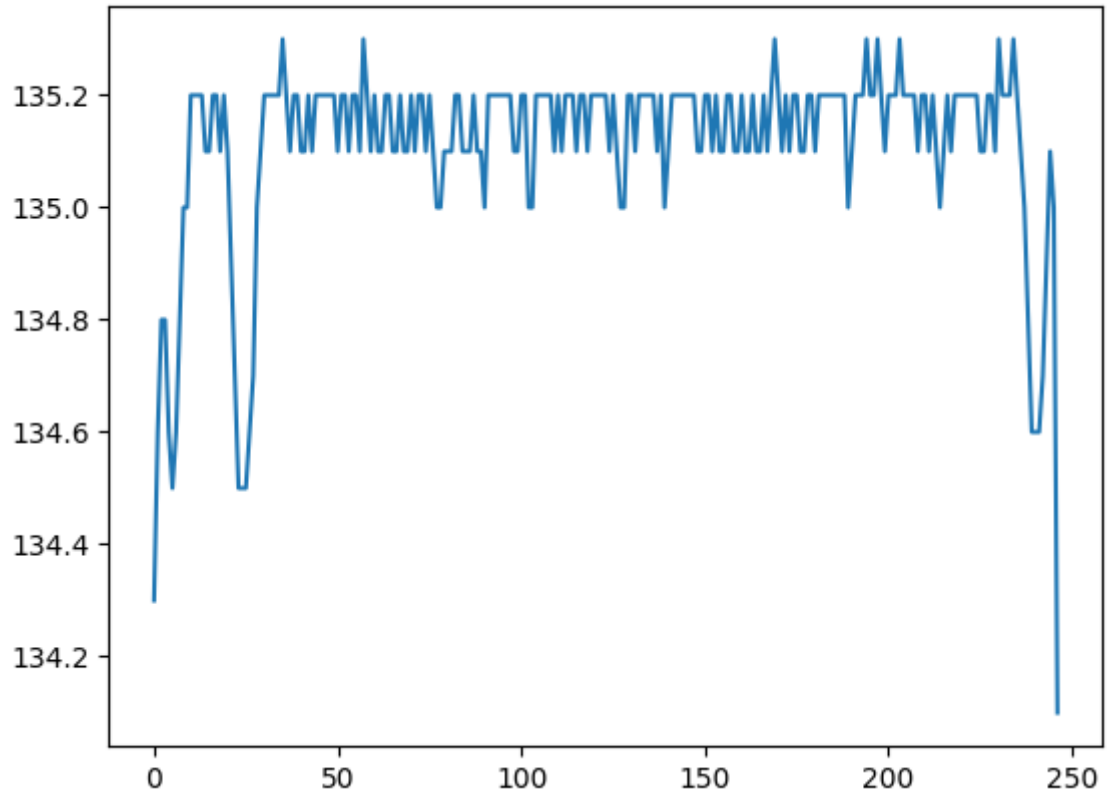
Computergestützte Berechnungen

Mithilfe von Phython haben wir ein Programm geschrieben, dass den höchsten Tiefpunkt und den tiefsten Hochpunkt über der Mindestschwelle findet und den Betrag der Differenz ihrer y-Koordinaten ausrechnet und diesen dann mit einem Faktor multipliziert, der von der Länge des Intervalls der Temperaturkurve über der Mindestschwelle abhängt und bekommen dadurch eine Toleranz. Wir überprüfen ab dem Überschreiten der Mindesttemperatur, ob aufeinanderfolgende Extrema diese Toleranz erfüllen und dann nimmt es den ersten Extrema, der zusammen mit seinen darauffolgenden Extrema die Toleranz erfüllt, als Anfang des Sterilisationsintervalls. Das oben genannte Modell wurde in der folgenden Codezelle aufgeführt.

In [9]:
```
print("Go!")

curve_to_reduce1 = "TempProdukt"
curve_to_reduce2 = "DruckKammer"


curve1 = data1[curve_to_reduce1].to_list()
curve2 = data1[curve_to_reduce2].to_list()
min_temperature = 134


def find_plateau(curve, min_temperature, tol = True, crv = None):

    if crv is None:
        curve_copy = copy.copy(curve)
        crv = []
        for i,q in enumerate(curve_copy):
            if q > min_temperature:
                crv.append(i)

        return crv[0], crv[-1]

def room_under_the_curve(curve,min_temperature):
    curve_copy = copy.copy(curve)
    sum1 = 0
    crv = []
    plot = []
    for n,i in enumerate(curve_copy):
        if i > min_temperature:
            crv.append([i,n])
    for i in crv:
        sum1 += i[0]
    sum1 /= len(crv)
    sum1 -= min_temperature
    sum1 *= crv[-1][1]-crv[0][1]
    print(crv[-1][1])
    print(len(crv))
    for i in crv:
        plot.append(i[0])
    plt.plot(plot)
```

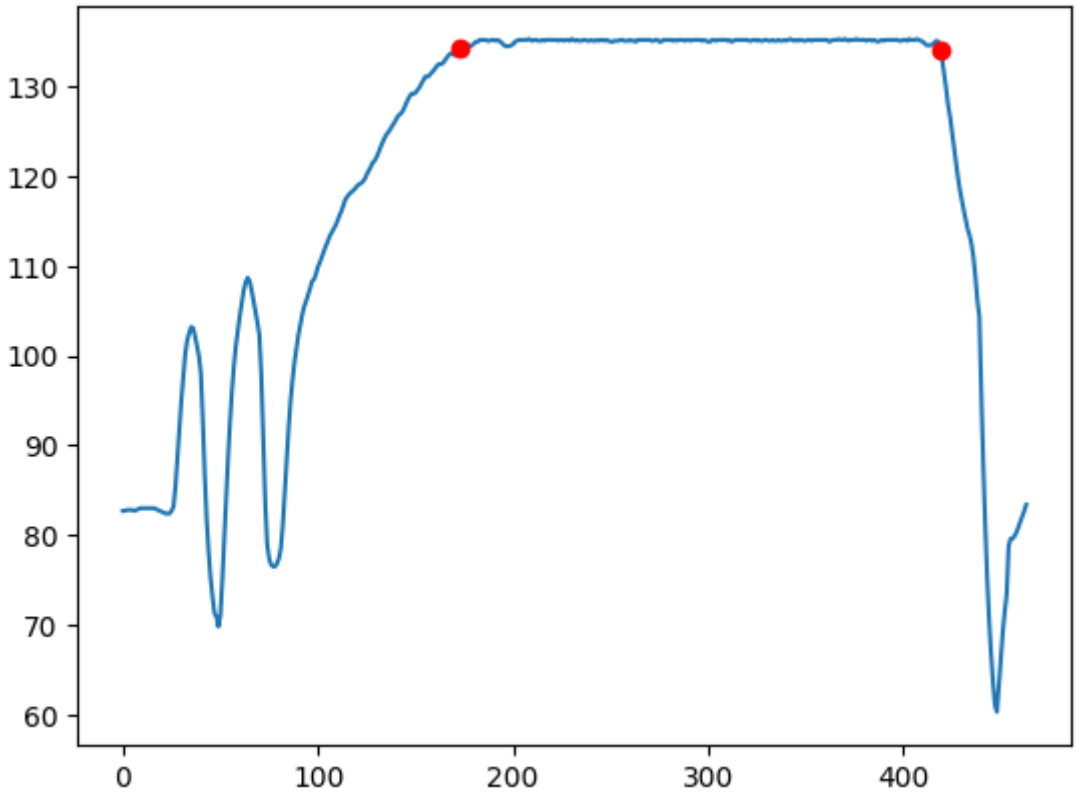
```
    plt.show()
    return sum1
print(room_under_the_curve(curve1,min_temperature))
print(2496/len(curve1))

start_index, end_index = find_plateau(curve1,min_temperature)

plt.plot(curve1)
plt.plot(start_index, curve1[start_index], "ro")
plt.plot(end_index, curve1[end_index], "ro")
```

```
Go!
419
247
```




```
273.3886639676323
5.379310344827586
```

Out[9]: `[<matplotlib.lines.Line2D at 0x2285641ff50>]`



Interpretation der Lösung

Der Algorithmus kann uns sehr genau das Plateau und somit den Sterilisationszeitraum bestimmen.

III Erster Modellansatz (Forschungsauftrag 2, Teil 2)

Vereinfachungen und Modellannahmen

Wir treffen zunächst folgende Annahme:

1. Wir nehmen an, dass der letzte Tiefpunkt vor dem Sterilisationsintervall der Anfang und der erste Tiefpunkt nach dem Sterilisationsintervall das Ende des erweiterten Intervalls ist, wenn die Daten stark komprimiert vorliegen.

Entwicklung eines ersten mathematischen Modells

Wir nehmen an, dass der letzte Tiefpunkt vor dem Sterilisationsintervall der Anfang und der erste Tiefpunkt nach dem Sterilisationsintervall das Ende des erweiterten Intervalls ist, wenn die Daten stark komprimiert vorliegen.

Computergestützte Berechnung

Unser Programm begrenzt das erweiterte Intervall in mehreren Schritten:

1. Suche nach dem Punkt, der als Erstes die Mindesttemperatur erreicht bzw. überschreitet
2. Starke Komprimierung der Daten
3. Suche nach dem letzten Tiefpunkt vor dem Plateau
4. Durch Umkehren der Daten bestimmen wir das Ende des Intervalls mithilfe des gleichen Verfahrens (Schritt 1–3)

In der folgenden Codezelle ist das Programm aufgeführt:

In [10]:
```
def anfangSteri(punkte,mindTemp):
    punkteVorSteri=[]
    i=0
    while(punkte[i][1]<mindTemp):
        punkteVorSteri.append(punkte[i])
        i+=1
    punkteVorSteri.append(punkte[i])
    print(punkteVorSteri)
    return punkteVorSteri

def tiefpunktFinden(punkte):
    tiefpunkt=-1
    i=0
    while i<len(punkte)-2:
        punkt1=punkte[i]
        punkt2=punkte[i+1]
        punkt3=punkte[i+2]
        steigung1=steigung(punkt1,punkt2)
        steigung2=steigung(punkt2,punkt3)

        if (steigung1<0 and steigung2>0):
            tiefpunkt=punkt2[0]
        i+=1
    return tiefpunkt

def steigung(punkt1,punkt2):
    return (punkt2[1]-punkt1[1])/(punkt2[0]--punkt1[0])

def komprimieren(punkte):
    return punkte

def grenzenIntervallFinden(allePunkte,mindTemp):
    neuePunkte=anfangSteri(allePunkte,mindTemp)
    komprimierteListe=komprimieren(neuenPunkte)
    tiefpunkt=tiefpunktFinden(neuePunkte)
    return tiefpunkt

#Beispiel
punkte=[[3],[0],[3],[-1],[6],[6],[0],[1.5],[-2]]
i=0
for punkt in punkte:
    punkt.insert(0,i)
    i+=1
print(punkte)
```

```
anfangIntervall=grenzenIntervallFinden(punkte,4) #Queue
punkte.reverse()
endeIntervall=grenzenIntervallFinden(punkte,4) #Stack
print("anfang Intervall: ",anfangIntervall)
print("ende Intervall: " ,endeIntervall)
```

```
[[0, 3], [1, 0], [2, 3], [3, -1], [4, 6], [5, 6], [6, 0], [7, 1.5], [8, -2]]
[[0, 3], [1, 0], [2, 3], [3, -1], [4, 6]]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 46
     43     i+=1
     44 print(punkte)
---> 46 anfangIntervall=grenzenIntervallFinden(punkte,4) #Queue
     47 punkte.reverse()
     48 endeIntervall=grenzenIntervallFinden(punkte,4) #Stack

Cell In[10], line 34, in grenzenIntervallFinden(allePunkte, mindTemp)
     32 def grenzenIntervallFinden(allePunkte,mindTemp):
     33     neuePunkte=anfangSteri(allePunkte,mindTemp)
---> 34     komprimierteListe=komprimieren(neuenPunkte)
     35     tiefpunkt=tiefpunktFinden(neuePunkte)
     36     return tiefpunkt

NameError: name 'neuenPunkte' is not defined
```

IV weiter Modellansatz (Forschungsauftrag 2, Teil 2)

Vereinfachungen und Modellannahmen

Aufgrund neuer Informationen, die wir durch den Problemsteller erhalten, haben treffen zunächst folgende Annahme:

1. Wir nehmen an, dass der letzte Punkt vor und der erste nach dem Steriliationsintervall, welcher auf Raumtemperatur liegt, die jeweilige Grenze des Intervalls abbildet.

Entwicklung eines ersten mathematischen Modells

Computergestützte Berechnung

Unser Programm begrenzt das erweiterte Intervall in mehreren Schritten:

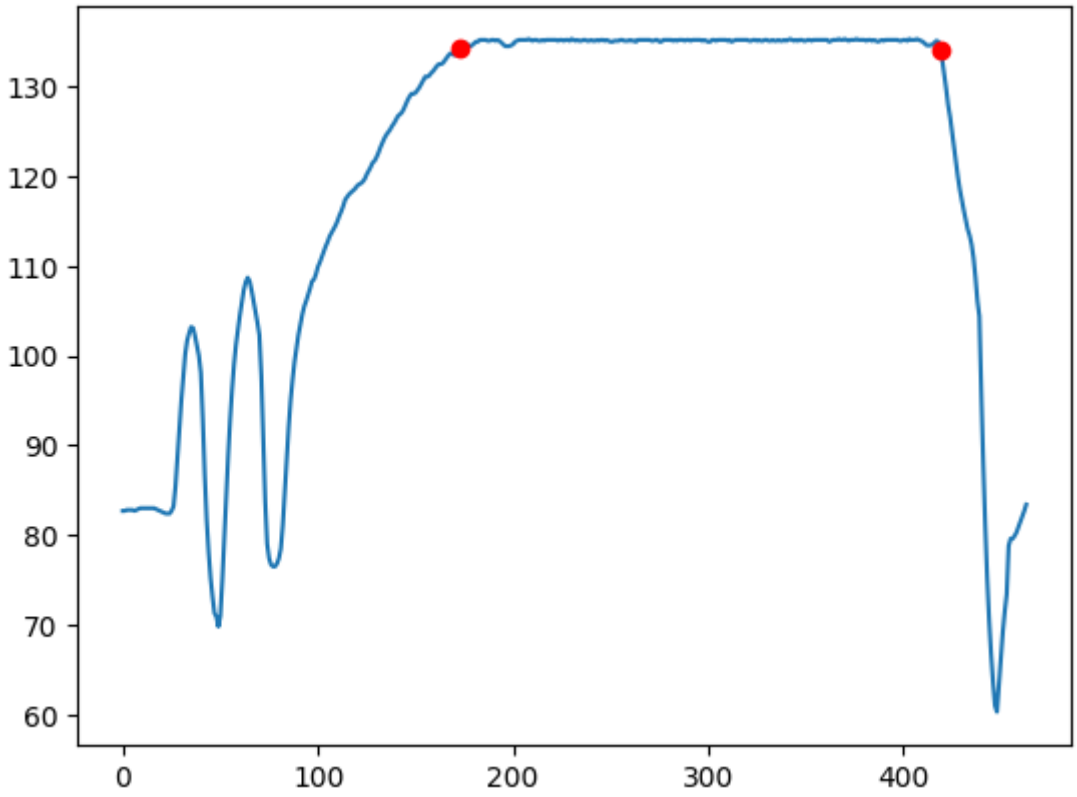
1. Suche nach dem Punkt, der als Erstes die Mindesttemperatur erreicht bzw. überschreitet
2. Starke Komprimierung der Daten um
3. Suche nach dem nächsten Punkt zum Anfang des Sterilisationsintervalls, der auf Raumtemperatur liegt

In der folgenden Codezelle ist das Programm aufgeführt:

In [11]:
```
print("Go!")

curve_to_reduce1 = "TempProdukt"
curve_to_reduce2 = "DruckKammer"


curve1 = data1[curve_to_reduce1].to_list()
curve2 = data1[curve_to_reduce2].to_list()
min_temperature = 134
def find_plateau(curve, min_temperature, tol = True, crv = None):

    if crv is None:
        curve_copy = copy.copy(curve)
        crv = []
        for i,q in enumerate(curve_copy):
            if q > min_temperature:
                crv.append(i)



    return crv[0], crv[-1]

def room_under_the_curve(curve,min_temperature):
    curve_copy = copy.copy(curve)
    sum1 = 0
    crv = []
    plot = []
    for n,i in enumerate(curve_copy):
        if float(i) > min_temperature:
            crv.append([i,n])
    for i in crv:
        sum1 += i[0]
    sum1 /= len(crv)
    sum1 -= min_temperature
    sum1 *= crv[-1][1]-crv[0][1]

    for i in crv:
        plot.append(i[0])

    return sum1
print(room_under_the_curve(curve1,min_temperature))


start_index, end_index = find_plateau(curve1,min_temperature)

plt.plot(curve1)
plt.plot(start_index, curve1[start_index], "ro")
plt.plot(end_index, curve1[end_index], "ro")
```

```
Go!
273.3886639676323
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x228578dd310>]
```



V. Schlussfolgerung / Empfehlungen

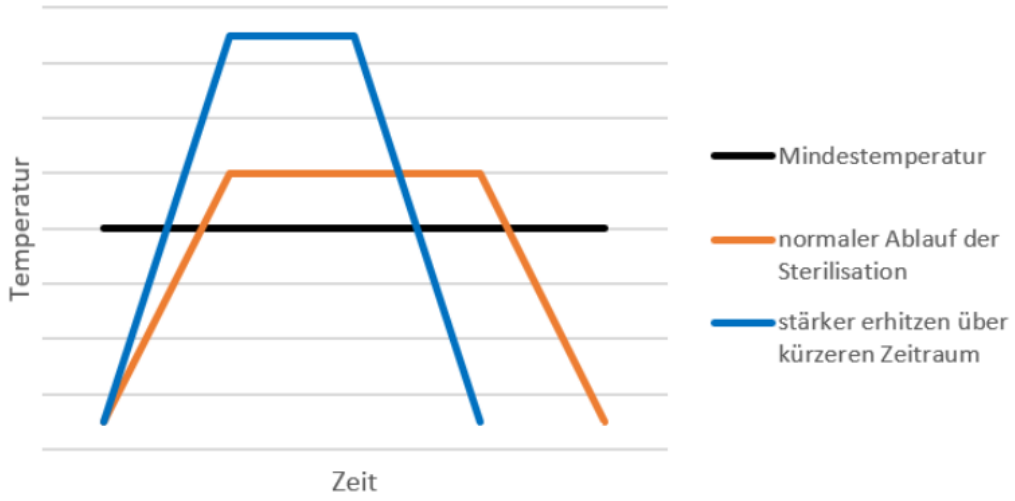
Mithilfe von uns entwickelter Algorithmen kann man den kompletten Prozess automatisieren und muss nicht auf alte Methoden, zurückgreifen, wie eine Schablone darüber legen oder ausdrücken und ausschneiden und wiegen. Zudem ist es auf jede Autoklave anpassbar. Aus unserem Forschungsauftrag 1 kann man außerdem bestimmen, wie sehr man es reduziert haben möchte.

Ausblick, um die Energieaufwand zu verringern

- Nutzen wir das Wissen, was wir aus diesem zweiten Intervall gewinnen, dann könnten wir Aussagen darüber machen, wie viele Mikroorganismen vor bzw. nach dem Sterilisationsintervall sterben, wodurch wir das Sterilisationsintervall selbst kürzen könnten.

- Es gibt zwei weitere Ansätze, welche wir uns nur theoretisch überlegt haben, um den Energieaufwand zu verringern

    1. In einem kürzen Intervall eine starke Erhitzung vornehmen, wobei die AUC gleich bleiben muss



2) Man könnte auch bis zu einem Wert erhitzen, sodass nur durch die Abkühlung alle Zellen sterben