Case1_oppdatert

October 17, 2022

1 Mappeoppgave 1

1.1 Beskrivelse

Les oppgaveteksten nøye. Se hvordan man leverer oppgaven

her

og

her. Husk at den skal leveres både som jupyter-fil og som PDF. Kommenter kodene du skriver i alle oppgaver og vær nøye på å definere aksene mm i figurer. I noen av oppgavetekstene står det hint, men det betyr ikke at de ikke kan løses på andre måter

For å hente denne filen til Jupyter gjør du slik: Åpne et "terminalvindu"

Gå til hjemmeområdet ditt

```
[user@jupty02 ~]$ cd
```

Lag en ny mappe på ditt hjemmeområde ved å skrive inn i terminalvinduet

```
[user@jupty02 ~]$ mkdir SOK-1003-eksamen-2022-mappe1
```

Gå så inn i den mappen du har laget ved å skrive

```
[user@jupty02 ~]$ cd SOK-1003-eksamen-2022-mappe1
```

Last ned kursmateriellet ved å kopiere inn følgende kommando i kommandovinduet:

```
[user@jupty02 sok-1003]$ git clone https://github.com/uit-sok-1003-h22/mappe/
```

Oppgi gruppenavn m/ medlemmer på epost o.k.aars@uit.no innen 7/10, så blir dere satt opp til tidspunkt for presentasjon 19/10. Bruk så denne filen til å gjøre besvarelsen din. Ved behov; legg til flere celler ved å trykke "b"

1.2 Oppgavene

1.2.1 Oppgave 1 (5 poeng)

a) Lag en kort fortelling i en python kode som inkluderer alle de fire typer variabler vi har lært om i kurset. Koden skal kunne kjøres med print(). Koden burde inneholde utregninger av elementer du har definert

[2]:
```
hens_lay_eggs=True

#På en uke legger de 7 egg
Eggs_per_week = 7

Price_per_egg = 5.5
big_eggs_per_week = 4

#Pris for 7 egg
total = Eggs_per_week * Price_per_egg

#Pris for store egg
price_big_eggs = 10

total = Eggs_per_week * Price_per_egg + price_big_eggs * big_eggs_per_week

s=f"""Det er {hens_lay_eggs} at hønene legger egg.
Hønene legger {Eggs_per_week} egg per uke.
Jeg vil selge eggene for {Price_per_egg} kr
Før jeg selger eggene legger defire store egg, de får jeg solgt for␣
 ↪{price_big_eggs} kr per egg.
Dersom jeg selger syv egg vil jeg tjene {Eggs_per_week*Price_per_egg} kr.
Jeg ender opp med å tjene {Eggs_per_week*Price_per_egg + big_eggs_per_week *␣
 ↪price_big_eggs} kr per uke på salg av egg.
Min bror som hjalp meg å høste eggen får en tredjedel av fortjenesten, dermed␣
 ↪får han {total/3}."""

print(s)
```

```
Det er True at hønene legger egg.
Hønene legger 7 egg per uke.
Jeg vil selge eggene for 5.5 kr
Før jeg selger eggene legger defire store egg, de får jeg solgt for 10 kr per
egg.
Dersom jeg selger syv egg vil jeg tjene 38.5 kr.
Jeg ender opp med å tjene 78.5 kr per uke på salg av egg.
Min bror som hjalp meg å høste eggen får en tredjedel av fortjenesten, dermed
får han 26.166666666666668.
```

1.2.2 Oppgave 2 (10 poeng)

Leieprisene i landet har steget de siste månedene. Ved å bruke realistiske tall a) Lag tilbuds og etterspørselsfunksjoner for leie av bolig (Bruk av ikke-lineære funksjoner belønnes). Definer funksjonene slik at det er mulig å finne en likevekt

[3]:
```
import matplotlib.pyplot as plt
import numpy as np
```

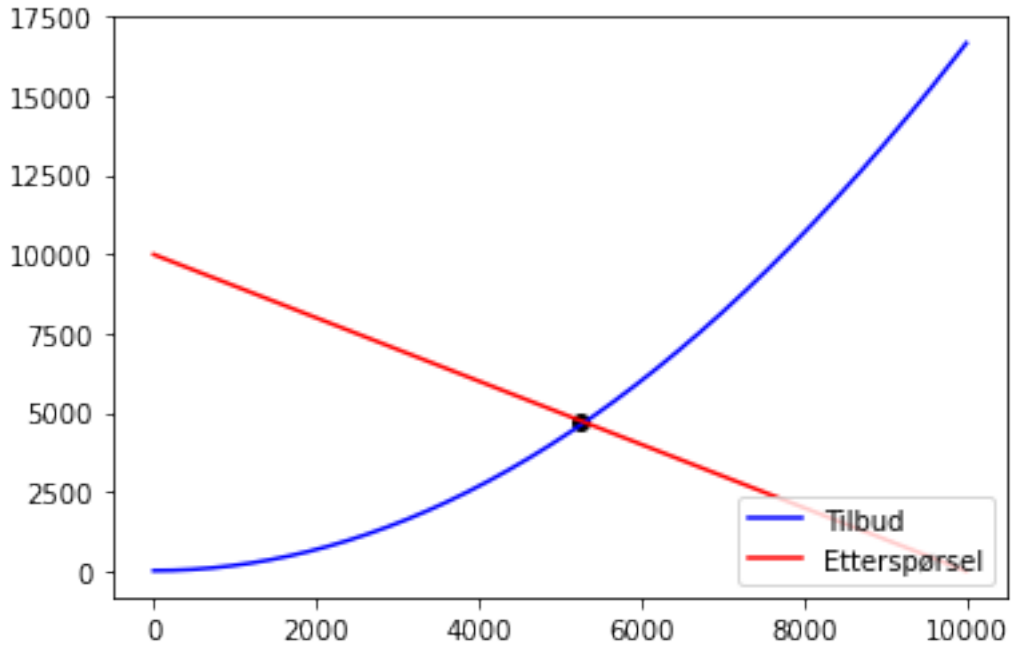
```
x = np.linspace(1,10000,10000)
def supply(x):
    return (x**2)*(12/72000)

def demand(x):
    return 10000-(10+x)

plt.plot(x,supply(x),color= "blue" ,label='Tilbud')

plt.plot(x,demand(x),color='red',label='Etterspørsel')
plt.legend(loc='lower right')
plt.scatter(5250, 4700, color="black")
```

[3]: <matplotlib.collections.PathCollection at 0x7fa81c928040>



b) Vis at disse er henholdvis fallende og stigende, ved bruk av

- Regning
- figurativt (matplotlib) Husk å markere aksene tydelig og at funskjonene er definert slik at linjene krysser

[69]:
```
def f(x):
    return (x**2)*(12/72000)

print(f(10000))
```

```
def g(x):
    return 10000-(10+x)
print(g(1000))
```

```
16666.666666666668
8990
```

c) Kommenter funksjonene og likevekten. Vis gjerne figurativt hvor likevekten er ved bruk av scatter

[ ]:
```
#Vi ser at grafene går litt om seg selv litt vanskelig å se på denne grafen.
# Det vi ser er om boliger er gratis da tilbud er på 0 har mange lyst på.
# Etterhvert som tilbudet går opp går etterspørselen ned fordi mindre har lyst
 ↪på leilighet til en viss pris.
# Da tilbud når høyt nok vil denne falle igjen å etterspørselen gå opp.
# Der vi har lagt in scatter plot er der linjene møtes, dette er cirka det
 ↪prisen faktisk ligger på der man får grei pris for leiligheten samtidig som
 ↪man har folk som har lyst på leilighet.
```

1.2.3 Oppgave 3 (15 poeng)

SSB har omfattende data på befolkningsutvikling (https://www.ssb.no/statbank/table/05803/tableViewLayout1/) Disse dataene skal du bruke i de neste deloppgavene.

a) lag lister av følgende variabler: "Befolkning 1. januar", "Døde i alt", "Innflyttinger" og "Utflyttinger". Velg selv variabelnavn når du definerer dem i python. Første element i hver liste skal være variabelnavnet. Bruk tall for perioden 2012-2021. Lag så en liste av disse listene. Du kan kalle den "ssb". Hint: når du skal velge variabler på SSB sin nettside må du holde inne ctrl for å velge flere variabler.

[17]:
```
SSB=[
['Befolkning januar', 'BJ', 4985870, 5051275, 5109056, 5165802, 5213985,
 ↪5258317, 5295619, 5328212, 5367580, 5391369],
['Døde i alt', 'DIA', 41992, 41282, 40394, 40727, 40726, 40774, 40840, 40684,
 ↪40611, 42002],
['Innflyttninger', 'INN', 78570, 75789, 70030, 67276, 66800, 58192, 52485,
 ↪52153, 38071, 53947],
['Utflyttninger', 'UT', 31227, 35716, 31875, 37474, 40724, 36843, 34382, 26826,
 ↪26744, 34297],
]
print(SSB)
```

```
[['Befolkning januar', 'BJ', 4985870, 5051275, 5109056, 5165802, 5213985,
5258317, 5295619, 5328212, 5367580, 5391369], ['Døde i alt', 'DIA', 41992,
41282, 40394, 40727, 40726, 40774, 40840, 40684, 40611, 42002],
['Innflyttninger', 'INN', 78570, 75789, 70030, 67276, 66800, 58192, 52485,
52153, 38071, 53947], ['Utflyttninger', 'UT', 31227, 35716, 31875, 37474, 40724,
36843, 34382, 26826, 26744, 34297]]
```

b) konverter "ssb" til en numpy matrise og gi den et nytt navn

[25]:
```
import numpy as np
SSB_np= np.array(SSB)
print(type(SSB_np))
```

```
<class 'numpy.ndarray'>
```

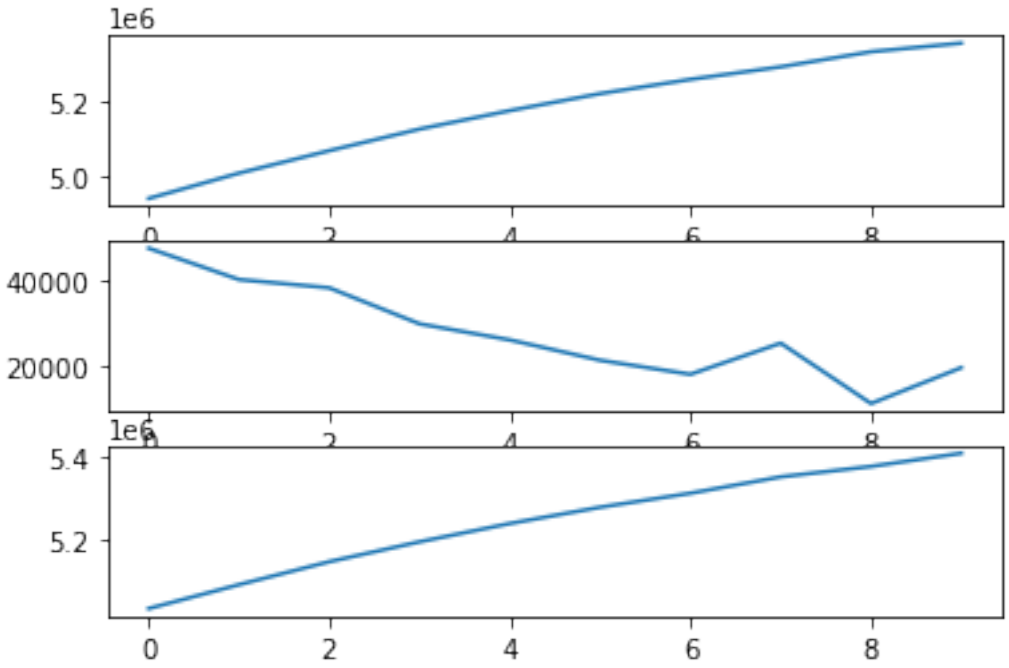
c) Putt alle tallene inn i en egen matrise og konverter disse til int

[23]:
```
SSB_numbers = [4985870, 5051275, 5109056, 5165802, 5213985, 5258317, 5295619,␣
 ↪5328212, 5367580, 5391369,41992, 41282, 40394, 40727, 40726, 40774, 40840,␣
 ↪40684, 40611, 42002,
               78570, 75789, 70030, 67276, 66800, 58192, 52485, 52153, 38071,␣
 ↪53947, 31227, 35716, 31875, 37474, 40724, 36843, 34382, 26826, 26744, 34297]
SSB_numbers1 = ''.join(map(str,SSB_numbers))
SSB_numbers2 = int(SSB_numbers1)
print(type(SSB_numbers2))
```

```
<class 'int'>
```

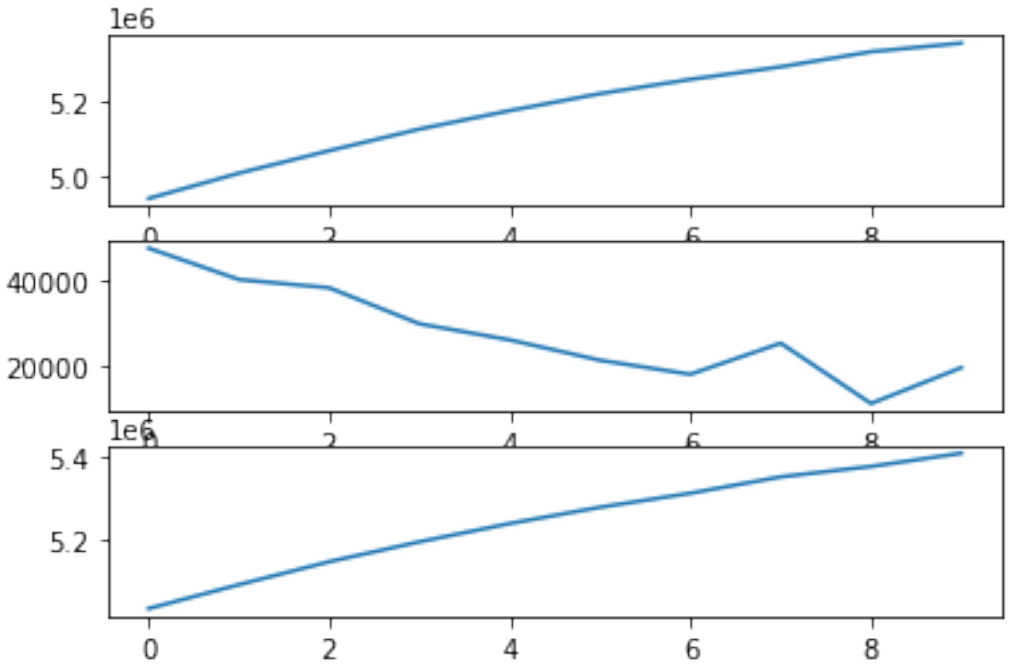
d) vis befolkningsutviklingen grafisk for de gjeldene årene ved bruk av matplotlib, og mer spesifikt "fig, ax = plt.subplots()". Vis befolkning på y-aksen i millioner

[8]:
```
from matplotlib import pyplot as plt
BJ = np.array([4985870, 5051275, 5109056, 5165802, 5213985, 5258317, 5295619,␣
 ↪5328212, 5367580, 5391369])
DIA = np.array([41992, 41282, 40394, 40727, 40726, 40774, 40840, 40684, 40611,␣
 ↪42002])
INN = np.array([78570, 75789, 70030, 67276, 66800, 58192, 52485, 52153, 38071,␣
 ↪53947])
UT = np.array([31227, 35716, 31875, 37474, 40724, 36843, 34382, 26826, 26744,␣
 ↪34297])
BJDIA = BJ - DIA
INNUT = INN -UT
BJINNUT = BJ + INNUT
fig,ax=plt.subplots(3)
ax[0].plot(BJDIA)
ax[1].plot(INNUT)
ax[2].plot(BJINNUT)
plt.show()
```



e) Lag det samme plottet ved bruk av oppslag. Hva er fordelen med dette?

[9]:
```
SSB = dict()
fig,ax=plt.subplots(3)
ax[0].plot(BJDIA)
ax[1].plot(INNUT)
ax[2].plot(BJINNUT)
plt.show()
print(type(SSB))
```




```
<class 'dict'>
```

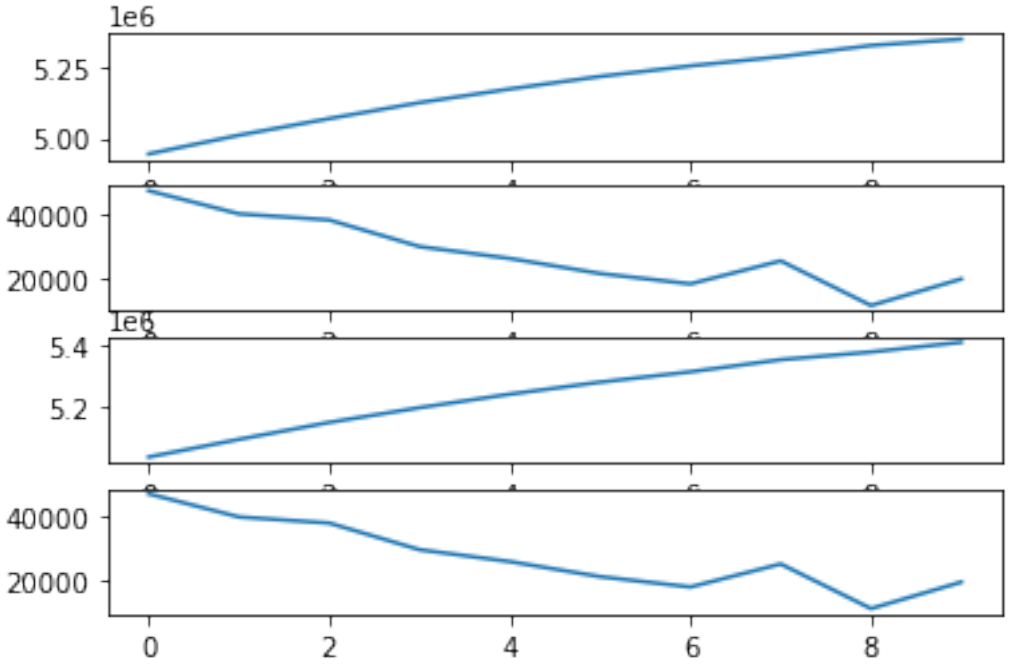
[ ]:
```
#Fordelen med å konvertere listen til et oppslag er at det blir enkelt å legge␣
↪til flere elementer i den alerede definerte lista.
```

f) Hva er den relative befolkningstilveksten utenom fødsler (dvs. innvandring/utvandring)? Definer en ny array og legg den til i oppslaget du laget i oppgaven tidligere. Kall den "rel_immigration". Plot denne sammen med grafen du laget i (d).

[10]:
```
rel_immigration = INN - UT
#SSB1 = SSB + ["rel_immigration", 47343, 40073, 38155, 29802, 26076, 21349,␣
↪18103, 25327, 11327, 19650]
#print(SSB1)
#print(rel_immigration)
SSB = {
        "rel_immigration" : INN - UT
         }
print(SSB)
from matplotlib import pyplot as plt
BJ = np.array([4985870, 5051275, 5109056, 5165802, 5213985, 5258317, 5295619,␣
↪5328212, 5367580, 5391369])
DIA = np.array([41992, 41282, 40394, 40727, 40726, 40774, 40840, 40684, 40611,␣
↪42002])
INN = np.array([78570, 75789, 70030, 67276, 66800, 58192, 52485, 52153, 38071,␣
↪53947])
```

```
UT = np.array([31227, 35716, 31875, 37474, 40724, 36843, 34382, 26826, 26744,␣
 ↪34297])
BJDIA = BJ - DIA
INNUT = INN -UT
BJINNUT = BJ + INNUT
fig,ax=plt.subplots(4)
ax[0].plot(BJDIA)
ax[1].plot(INNUT)
ax[2].plot(BJINNUT)
ax[3].plot(rel_immigration)
plt.show()
```

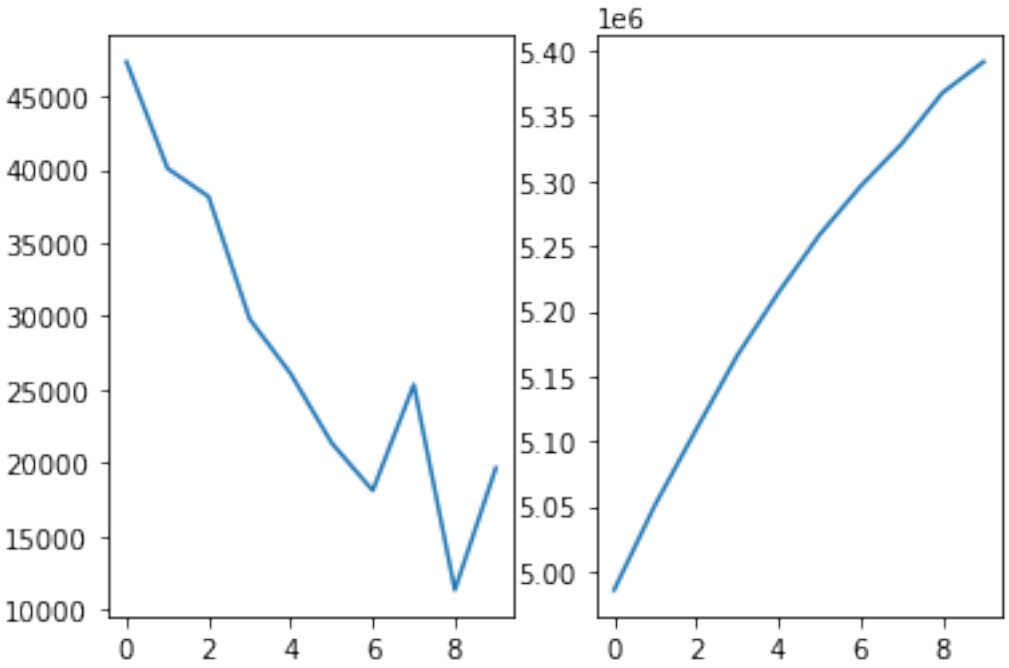
```
{'rel_immigration': array([47343, 40073, 38155, 29802, 26076, 21349, 18103,
25327, 11327,
       19650])}
```



g) ekstrapoeng. Kan plotte de samme tallene (dvs "rel_immigration" og "befolkning" sammen med år) i to figurer ved siden av hverandre ved bruk av "fig, (ax1, ax2) = plt.subplots(1, 2)". Gi grafene ulik farge

[11]:
```
from matplotlib import pyplot as plt
BJ = np.array([4985870, 5051275, 5109056, 5165802, 5213985, 5258317, 5295619,␣
 ↪5328212, 5367580, 5391369])
DIA = np.array([41992, 41282, 40394, 40727, 40726, 40774, 40840, 40684, 40611,␣
 ↪42002])
```

```
INN = np.array([78570, 75789, 70030, 67276, 66800, 58192, 52485, 52153, 38071,␣
 ↪53947])
UT = np.array([31227, 35716, 31875, 37474, 40724, 36843, 34382, 26826, 26744,␣
 ↪34297])
BJDIA = BJ - DIA
INNUT = INN -UT
BJINNUT = BJ + INNUT
fig,ax=plt.subplots(1, 2)
ax[0].plot(rel_immigration)
ax[1].plot(BJ)
plt.show()
```



1.3 Oppgave 4 (20 poeng)

Et lån består som regel av et månedlig terminbeløp. Dette beløpet er summen av avdrag (nedbetalingen på lånet) og renter. Vi antar månedlig forrenting i alle oppgavene. Dvs. at det er 12 terminer i hvert år. a) Lag en funksjon som regner ut hvor mye lånet "x" koster deg i renteutgifter for "t" terminer med årlig rente "r" for et serielån. Siden dette er et serielån, så vil avdragene være like hver måned men renteutgiftene reduseres i takt med avdragene. Renteutgiftene for en gitt termin "t" vil derfor være den årlige renten "r" (delt på antall forrentinger "f") på gjenværende beløp på det tidspunktet. $renteutgifter_t = (x - a * (t-1)) * r/f$ Det vil si at renteutgiftene første termin er $renteutgifter_1 = (x - a * 0) * r/f$, og andre termin er $renteutgifter_2 = (x - a * 1) * r/f$ osv.. Siden vi er ute etter den totale kostnaden i svaret, må du summere renteutgiftene over alle terminer, det vil si $\sum_{t=1}^{N}(x - a * (t-1)) * r/f$. Dette betyr egentlig bare $renteutgifter_1 + renteutgifter_2, ..., + renteutgifter_t$

Hint: siden terminbeløpet varierer for hver måned (pga at rentene endres), må alle enkeltperioder summeres. Det kan være nyttige å bruke funksjonen np.arange() til dette. Mao, det er ikke nødvendig å bruke sigma ($\sum_{t=1}^{N}$) i formelen til dette

[99]:
```
import numpy as np

def serielan(x, a, r, t, f):
    return (x-a*np.arange(0, t))*r/f

print(serielan(1000000,7152, 0.03, 12, 12))
print(np.sum(serielan(1000000,7152, 1.03, 12, 12)))
```

```
[2500.   2482.12 2464.24 2446.36 2428.48 2410.6  2392.72 2374.84 2356.96
 2339.08 2321.2  2303.32]
989483.92
```

b) regn ut hvor mye lånet koster deg med henholdsvis 10, 20 og 30 års tilbakebetaling. Anta 1 000 000 kr lånebeløp med 3% rente

[13]:
```
loan_amt = float(input("enter the loan amout:"))
rate = float(input("enter the annual interest rate precentage:"))
years = int(input("how many years will it take to repay the loan?"))
period_rate = rate/12/100
num_payments = years * 12
payment_amt = period_rate * loan_amt / (1 - (1 + period_rate) ** -num_payments)
total_cost = num_payments * payment_amt - loan_amt
print("the payment will be NOK", payment_amt, "per month.")
print("the total cost of borrowing will be NOK", total_cost, ".")
```

```
enter the loan amout: 1000000
enter the annual interest rate precentage: 3
how many years will it take to repay the loan? 20

the payment will be NOK 5545.975978539206 per month.
the total cost of borrowing will be NOK 331034.23484940943 .
```

[ ]:
```
#https://www.youtube.com/watch?
↪v=qrTOhcRPBrM&ab_channel=Ben%27sComputerScienceVideos
```

c) Vis hva det samme lånet koster som annuitetslån, dvs differansen mellom alle terminbeløp og lånebeløp. Annuitetslån gir like terminbeløp hver måned, men renten utgjør en større del av dette beløpet i starten. Terminbeløpet for et annuitetslån er definert ved formelen: $T = x * \frac{r/f}{(1-(1+(r/f))^{-t})}$, hvor x=lånebeløp, r = årlig rente, t = terminer, f= antall forrentinger

[70]:
```
import numpy as np

def annuitetslan(x, r, t , f):
    return x*r/f/(1-(1+(r/f))**-t)

print(np.sum(annuitetslan(1000000,0.03, 120, 10)))
```

```
print(np.sum(annuitetslan(1000000,0.03, 240, 20)))
print(np.sum(annuitetslan(1000000,0.03, 360, 30)))
```

```
9935.499427302084
4964.6604858778055
3309.0869411921294
```

c) Vis hvordan utviklingen i rentekostnader og avdrag på terminer for serielån grafisk ved hjelp av stackplot funksjonen i matplotlib. Anta et bankinnskudd x = 1 000 000 kr, årlig rente r=3% og antall terminer t = 240 (det vil si 20 år). Siden vi må vise utviklingen per termin, husk at "t" også definerer hvilken måned vi er i. Dvs, hvis t=15, har det gått 1 år og 3 mnd med terminer. Se forøvrig relevante formler i oppgave (a)

Hint1: Siden avdragene er like for alle måneder, kan det være lurt å definere det månedlige avdraget som en liste og gange det med antall perioder. Hint2: Siden vi er ute etter både rentekostnader og avdrag hver for seg, kan det være lurt å definere en funksjon for hver av dem.

[98]:
```
from matplotlib import pyplot as plt

renter = serielan(1000000,7152,0.03,240,240)
print(renter)
terminer = range(0,240)
avdrag = [7152]*240

plt.stackplot(terminer, avdrag, renter, colors =["steelblue", "r"])
```

```
[125.    124.106 123.212 122.318 121.424 120.53  119.636 118.742 117.848
 116.954 116.06  115.166 114.272 113.378 112.484 111.59  110.696 109.802
 108.908 108.014 107.12  106.226 105.332 104.438 103.544 102.65  101.756
 100.862  99.968  99.074  98.18   97.286  96.392  95.498  94.604  93.71
  92.816  91.922  91.028  90.134  89.24   88.346  87.452  86.558  85.664
  84.77   83.876  82.982  82.088  81.194  80.3    79.406  78.512  77.618
  76.724  75.83   74.936  74.042  73.148  72.254  71.36   70.466  69.572
  68.678  67.784  66.89   65.996  65.102  64.208  63.314  62.42   61.526
  60.632  59.738  58.844  57.95   57.056  56.162  55.268  54.374  53.48
  52.586  51.692  50.798  49.904  49.01   48.116  47.222  46.328  45.434
  44.54   43.646  42.752  41.858  40.964  40.07   39.176  38.282  37.388
  36.494  35.6    34.706  33.812  32.918  32.024  31.13   30.236  29.342
  28.448  27.554  26.66   25.766  24.872  23.978  23.084  22.19   21.296
  20.402  19.508  18.614  17.72   16.826  15.932  15.038  14.144  13.25
  12.356  11.462  10.568   9.674   8.78    7.886   6.992   6.098   5.204
   4.31    3.416   2.522   1.628   0.734  -0.16   -1.054  -1.948  -2.842
  -3.736  -4.63   -5.524  -6.418  -7.312  -8.206  -9.1    -9.994 -10.888
 -11.782 -12.676 -13.57  -14.464 -15.358 -16.252 -17.146 -18.04  -18.934
 -19.828 -20.722 -21.616 -22.51  -23.404 -24.298 -25.192 -26.086 -26.98
 -27.874 -28.768 -29.662 -30.556 -31.45  -32.344 -33.238 -34.132 -35.026
 -35.92  -36.814 -37.708 -38.602 -39.496 -40.39  -41.284 -42.178 -43.072
 -43.966 -44.86  -45.754 -46.648 -47.542 -48.436 -49.33  -50.224 -51.118
 -52.012 -52.906 -53.8   -54.694 -55.588 -56.482 -57.376 -58.27  -59.164
```

```
-60.058 -60.952 -61.846 -62.74  -63.634 -64.528 -65.422 -66.316 -67.21
-68.104 -68.998 -69.892 -70.786 -71.68  -72.574 -73.468 -74.362 -75.256
-76.15  -77.044 -77.938 -78.832 -79.726 -80.62  -81.514 -82.408 -83.302
-84.196 -85.09  -85.984 -86.878 -87.772 -88.666]
```

[98]:
```
[<matplotlib.collections.PolyCollection at 0x7fa80f1d9370>,
 <matplotlib.collections.PolyCollection at 0x7fa80f1d9760>]
```