Flumph 3 toestanden

Voor de Flumph geldt het volgende:

	Morgen Hongerig	Morgen Tevreden	Morgen Opgejaagd
Vandaag Hongerig	80%	10%	10%
Vandaag Tevreden	40%	50%	10%
Vandaag Opgejaagd	60%	20%	20%

Beginkansen:

	Kans
Hongerig	10%
Tevreden	70%
Opgejaagd	20%

Om dit om te zetten naar datatypes in python is de volgende code geschreven:

```
toestanden1 = {
    0 : 'Hongerig',
    1 : 'Tevreden',
    2 : 'Opgejaagd'
}
chances1 = [
    [0.8, 0.1, 0.1],
    [0.4, 0.5, 0.1],
    [0.6, 0.2, 0.2]
]
chanceVector1 = [0.1, 0.7, 0.2]
```

Deze data wordt vervolgens gebruikt om een Flumph object aan te maken:

```
flumph3 = Flumph(toestanden1, chances1, chanceVector1)
```

Deze klasse Flumph bevat een dictionary met welke staat welk nummer betekent, de kansmatrix van kansen na elke dag, en een vector van kansen bij geboorte:

```
def __init__(self, states, chances, startChances):
        self.states = states
        self.chances = chances
        self.startChances = startChances
```

Om de kansen door te rekenen wordt de volgende functie gebruikt:

```
def generateChances(self):
        currVector = self.startChances
        data = {}
        for i in range(len(self.states)):
            data[i] = []

        count = 0
        while(True):
            prevcV = currVector
            currVector = np.dot(currVector, self.chances)
            for i in range(len(currVector)):
                data[i].append(currVector[i])
            count+=1
            for i in range(len(currVector)):
                if(currVector[i] > 0.99):
                    return self.states, data
            if(np.array_equal(prevcV,currVector)):
                return self.states, data
```

In deze functie worden de kansen net zo lang doorgerekend totdat bij 1 staat de kans groter is dan 99% of er 2 keer achter elkaar eenzelfde vector uitkomt, wat betekent dat de vector constant hetzelfde gaat blijven.

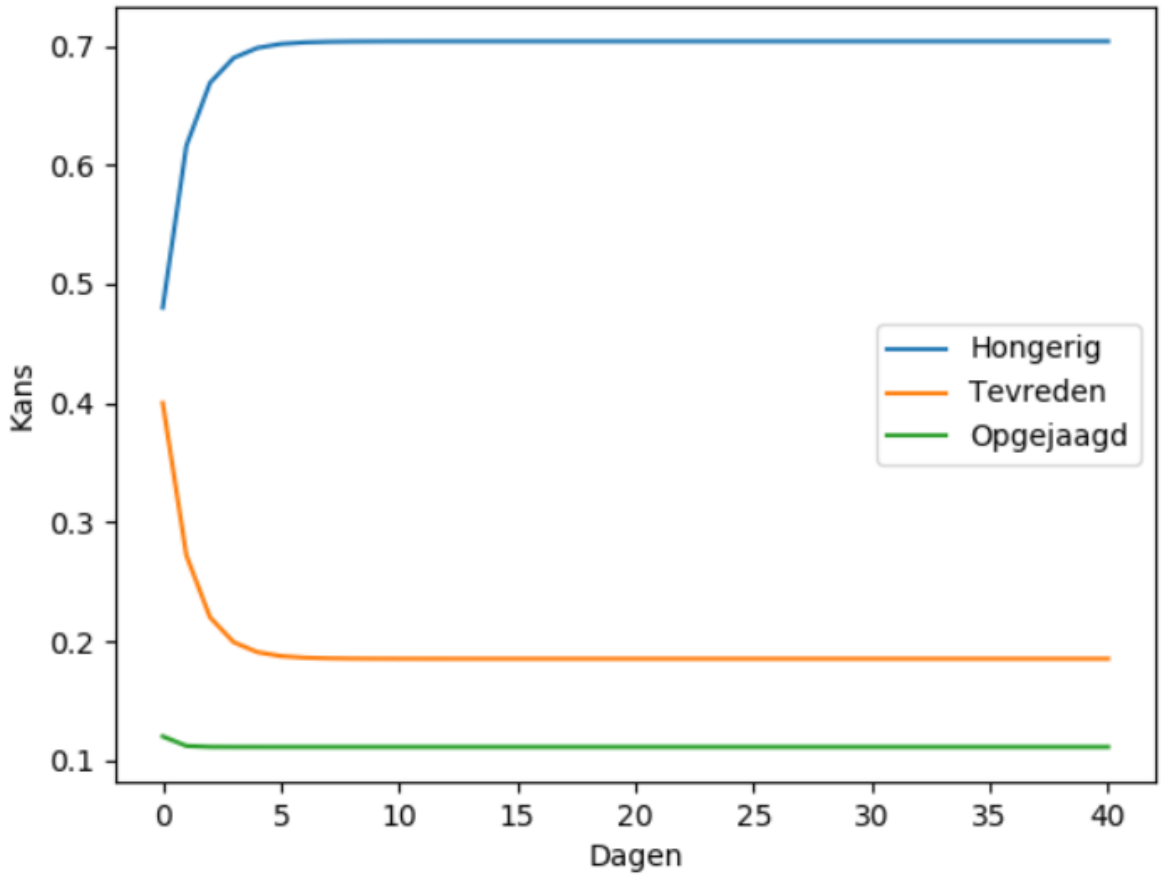
Om de kansen te plotten is de volgende functie geschreven:

```
def plotChances(self):
        toestanden, data = self.generateChances()

        x = list(range(len(data[0])))
        for i in range(len(data)):
            plt.plot(x, data[i])

        plt.gca().legend(toestanden.values())
        plt.ylabel('Kans')
        plt.xlabel('Dagen')
        plt.show()
```

Hierbij wordt de bovenstaande functie gebruikt die de kansen genereert. Deze worden vervolgens met matplotlib geplot. Met de kansen van de casus levert dat het volgende resultaat op:



Hierin is duidelijk te zien dat de kansen de eerste dagen nog relatief dichtbij elkaar zitten, na verloop van tijd is de kans op hongerig zijn +-70%, tevreden +-20% en opgejaagd 10%. De grafiek gaat niet verder omdat hierna de code is afgesloten omdat de vectors / kansen gelijk bleven. De groeiende curve van hongerig in de eerste dagen is te verklaren door de kleine kans van hongerig zijn op de eerste dag in vergelijking met de rest van de dagen. Zodra een Flumph hongerig is, is er 80% kans dat hij de dag daarop weer hongerig is, daarnaast is de kans bij opgejaagd ook het grootst dat de Flumph daarna hongerig is, dit verklaart de snelle groei. Het gegeven dat de lijn uiteindelijk niet meer veranderd komt doordat de input uiteindelijk dezelfde waarde teruggeeft dan de output, bijvoorbeeld voor de staat ‘hongerig’:

0,70 * 0,80 + 0,70 * 0,10 + 0,70 * 0,10 = 0,70.

Flumph 4 toestanden

Als vierde staat is de staat ‘Overleden’ toegevoegd, want ook Flumphs gaan helaas op een gegeven moment dood.

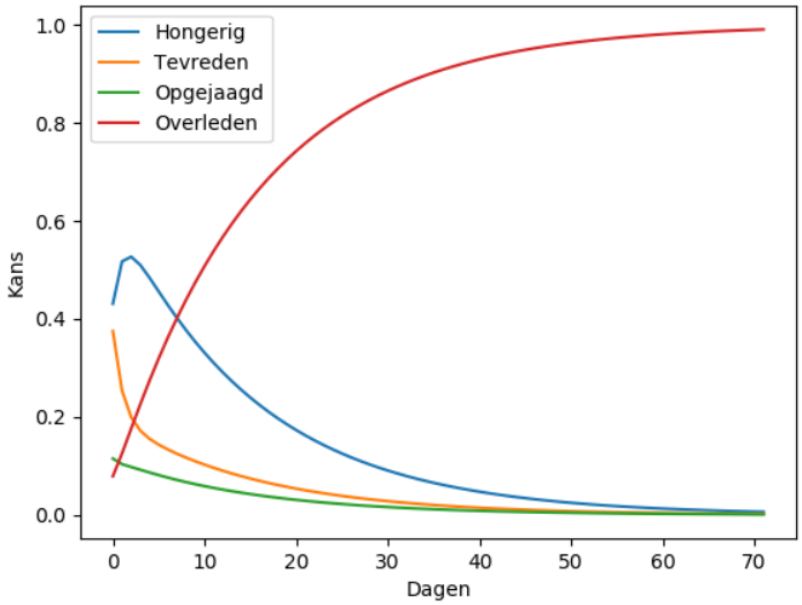
Hiervoor zijn de volgende kansen opgesteld:

```
toestanden2 = {
    0 : 'Hongerig',
    1 : 'Tevreden',
    2 : 'Opgejaagd',
    3 : 'Overleden'
}
chances2 = [
    [0.72, 0.1, 0.1, 0.08],
    [0.38, 0.5, 0.1, 0.02],
    [0.56, 0.2, 0.2, 0.04],
    [ 0.0, 0.0, 0.0,  1.0]
]
chanceVector2 = [0.1, 0.65, 0.2, 0.05]
```

De kansen van de andere staten zijn iets bijgesteld, en er is steeds een kleine kans dat een Flumph overlijdt, ook bij geboorte. Als een Flumph overlijdt kan hij niet meer in een andere staat terecht komen.

Ook hiervoor wordt een Flumph object aangemaakt:

```
flumph4 = Flumph(toestanden2, chances2, chanceVector2)
```



Dit levert uiteindelijk deze grafiek op. Dit is een hele verandering ten opzichte van de Flumph die niet kan overlijden. Datgene wat hier het grote verschil maakt is dat een Flumph na zijn dood niet in een andere staat terecht kan komen. Deze ontwikkeling is ook te zien in de grafiek, de kans van de staat overlijden wordt steeds groter.

Een interessante curve is die van de staat hongerig in de eerste dagen van het leven van een Flumph. Deze ontwikkeling komt door de grote kans van hongerig zijn, in combinatie met het feit dat een Flumph niet uit de dood op kan staan. De kans wordt namelijk met de tijd steeds groter dat de Flumph wel een keer de staat overleden betreedt, wat ervoor zorgt dat deze hier niet meer uit kan komen. Dit zorgt ervoor dat de Flumph niet langer leeft dan ongeveer 60 dagen.

Extra: Tensorproduct

Als extra is er gekozen voor het tensorproduct. Hiervoor is de volgende functie geschreven in de klasse:

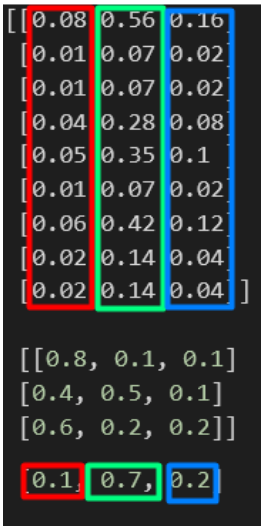
```
def getTensorProduct(self, steps = 1):
        value = self.startChances
        for i in range(steps):
            value = np.outer(self.chances, value)
        return value
```

Dit geeft voor de Flumph met 3 toestanden voor 1 stap het volgende resultaat:

```
[[0.08 0.56 0.16]
 [0.01 0.07 0.02]
 [0.01 0.07 0.02]
 [0.04 0.28 0.08]
 [0.05 0.35 0.1 ]
 [0.01 0.07 0.02]
 [0.06 0.42 0.12]
 [0.02 0.14 0.04]
 [0.02 0.14 0.04]]

 [[0.8, 0.1, 0.1]
 [0.4, 0.5, 0.1]
 [0.6, 0.2, 0.2]]

 [0.1, 0.7, 0.2]
```



Wat deze matrix betekent is het volgende.

Elk getal in de kansmatrix wordt gemultipliceerd met de getallen in de originele kansvector, zie rechts gevisualiseerd. Elke kolom hoort bij een getal uit de kansvector. De hele eerste horizontale rij van de output hoort dus bij (1,1) uit de kansmatrix, gemultipliceerd door elk getal uit de kansvector. De tweede rij hoort dan weer bij (1,2), en zo door.
Voor de kans is het tensorproduct niet heel erg interessant, omdat het eigenlijk te veel weergeeft. De enige mogelijke kansen zijn namelijk:

Kolom 1: Rij 1 tot 3
Kolom 2: Rij 4 tot 6
Kolom 3: Rij 7 tot 9
Deze 9 waardes vormen alle mogelijke paden voor de eerste stap.

GitHub

https://github.com/berryhijwegen/AC_HU/tree/master/statistiek/MarkovChain/code

Bronnen

https://nl.wikipedia.org/wiki/Tensorproduct