Sterbetafeln und "Data Science"
Generalversammlung der AVÖ

Markus Fulmek

Universität Wien
Markus.Fulmek@univie.ac.at

17. Mai 2018

Inhaltsverzeichnis

Sterbetafeln vor der AVÖ = Eulen nach Athen tragen

Die Organisatoren der Generalversammlung wünschten ein Vortragsthema "im Themenbereich Sterbetafeln":
Vor der Generalversammlung der österreichischen Aktuarvereinigung einen Vortrag über Sterbetafeln zu halten ist vergleichbar mit dem sprichwörtlichen "*Eulen nach Athen tragen*": Natürlich wissen alle hier im Raum, was Sterbetafeln *sind*, wie sie *erstellt* und in der versicherungstechnischen Praxis *verwendet* werden.
Die Gefahr ist also *riesengroß*, daß Sie bei meinem Vortrag einschlafen.
(Diese Gefahr wird ein bißchen gemildert durch den Umstand, daß der Vortrag nicht länger als 40 Minuten dauern soll.;-)

Uminterpretation des Vortragsthemas in Richtung "Data Science"

Um also einerseits dem Wunsch der Organisatoren nach einem Vortragsthema "im Themenbereich Sterbetafeln" zu entsprechen, andererseits aber die dabei drohende Langeweile möglichst zu verhindern, werde ich das Thema "Sterbetafeln" mit einem Thema in Beziehung setzen, das seit einigen Jahren "in aller Munde" ist: *Data Science*.

“Data Science”, zunächst recht allgemein

Unter *Data Science* versteht man recht allgemein

- das Ableiten von relevanten, nützlichen und klaren Informationen
- aus einem undurchsichtigen, riesigen Haufen von Daten (das erklärt den Begriffsbestandteil *DATA*), aus denen die gesuchten Informationen *nicht ohne weiteres* erkennbar sind.

Weil die gesuchten Informationen in der Regel nicht offen zutage liegen, müssen sie mit Methoden der Statistik, mathematischen Optimierung und Informatik *systematisch gesucht* werden (das erklärt den Begriffsbestandteil *SCIENCE*).

"Data Science": Konkrete Problemstellung (Beispiel)

Beispiel: Nützliche und relevante Information

Für ein Versicherungsunternehmen, das eine kostenintensive Werbekampagne für ein neues Produkt starten möchte, ist beispielsweise die Information *nützlich und relevant*, bei welchen Personen die teure Werbemaßnahme am ehesten zu dem angestrebten Verkaufserfolg führen wird.

Beispiel: Undurchsichtige und zahlreiche Daten

Aus verschiedenen Datenquellen (bei Bestandskunden z.B. aus den eigenen IT–Systemen) kann man für potentielle Adressaten verschiedene Merkmale (Alter, Geschlecht, Wohnort, Einkommensklasse, bestehende Versicherungen, Kreditwürdigkeit, etc.) sammeln, aber diese Rohdaten sind in der Regel *zahlreich, hochdimensional und undurchsichtig*.

“Data Science”: Konkrete Ansätze

Um die gewünschten Hinweise aus einer großen Datenmenge (*Big Data*) zu extrahieren, kann man folgende Ansätze wählen, die alle unter den Oberbegriff Data Science fallen:

- Klassische Datenanalyse: Visualisierung, Dimensionsreduktion (principal component analysis), Regressionsrechnung, Satz von Bayes,
- (Klassische) *Artificial Intelligence* (AI): regelbasierte Programmierung, Expertensystem,
- *Machine Learning*: Automatisches Clustering, logistische Regression, Entscheidungsbäume (Random forests), Support Vector Machines,
- *Deep Learning*: Beeindruckende Erfolge (besonders bei *image recognition*) haben in den letzten Jahren zu einem Revival der *Neuralen Netze* geführt, die schon im vorigen Jahrtausend untersucht wurden.

(Damit sind zugleich ein paar der wichtigsten *Buzzwords* genannt, die momentan in der Diskussion vorherrschen.)

“Data Science”: In meinem Vortrag sehr reduziert

Bitte erwarten Sie nun aber keinen Crash–Course in fortgeschrittenen Data–Science Themen wie Support Vector Machines oder Deep Learning: Das würde den Rahmen meines Vortrags völlig sprengen.

Ich möchte ihnen lediglich an ein paar Beispielen, die (mehr oder weniger) mit dem Generalthema “Sterbetafeln” verbunden sind, *illustrieren*, wie einfach man heutzutage Data Science–Ansätze und Methoden anwenden kann, wenn man die sehr reichhaltigen mathematisch–statistischen Programmbibliotheken zu nutzen weiß, die (großteils) frei zur Verfügung stehen.

“Data Science”: Einfache Anwendung!

Mathematische Grundlagen

Grundlage für die verschiedenen Data–Science Ansätze sind immer mathematische Konzepte und Methoden.

Informatische Umsetzung

Die technische Umsetzung der Methoden erfolgt in ausgeklügelten Computerprogrammen, in denen die hard– und softwaretechnischen Entwicklungen der letzten Jahrzehnte eingearbeitet sind.

Praktischer Einsatz

Für den praktischen Einsatz dieser Methoden kann man aber auf eine Fülle von vorgefertigten Softwarebausteinen zurückgreifen, die außerdem *kostenlos* zur Verfügung steht!

Prodesse et delectare

Ich möchte im folgenden frei zur Verfügung stehende Software–Tools auf frei zur Verfügung stehende Todesfalldaten anwenden und Ihnen damit (auf hoffentlich unterhaltsame Weise) einige *sehr vereinfachte* Anwendungen von Data Science–Ansätzen präsentieren, die zum gewünschten Thema "Sterbetafeln" passen.

Disclaimer

Mit meinem Vortrag will ich Ansätze und Vorgangsweisen lediglich *illustrieren*, ich erhebe *nicht den geringsten Anspruch* auf mathematische Exaktheit oder versicherungstechnische Eignung der vorgestellten Konzepte, Methoden und Tools.
Sehen Sie meinen Vortrag einfach als "spielerisch–experimentelle Aufwärmrunde" für die folgenden Fachvorträge (denen ich auch keineswegs vorgreifen will!) an.

Rohdaten: Deutsches Bundesamt für Statistik

Es ist gar nicht so leicht, frei zugängliche Rohdaten zur Sterblichkeit zu finden: Nach längerer Internet-Recherche scheint das deutsche Bundesamt für Statistik eine gute Quelle zu sein; konkret findet man unter dem Link

`https://www-genesis.destatis.de/genesis/online/logon`

eine Excel–Tabelle der Todesfälle in Deutschland für die Jahre 2011–2015, gegliedert nach Geschlecht und Lebensalter.
Ich habe dieses Excel–File heruntergeladen und werde es in der Folge zur Illustration verwenden.

Software und Libraries: Python, pandas, numpy, scipy, matplotlib

Ich verwende für meine illustrativen Data Science Experimente die Scriptsprache *Python* samt Software–Libraries, die für UNIX–Betriebssysteme (aber auch für Windows) *kostenlos* zur Verfügung stehen.

(Auch andre Software–Tools wie *R*, *Matlab* oder *Mathematica* bieten eine vergleichbare Funktionalität.)

Nach dem Einlesen der notwendigen Programmbibliotheken (konkret: *pandas* für das einfache Arbeiten mit Excel–Files, *numpy* für numerische Routinen, *matplotlib* für Graphiken und *scipy* für einfache Machine–Learning–Algorithmen) und ein paar "administrativen" Festlegungen (Pfadnamen) benötigt das Laden der Rohdaten eine einzige Zeile Python–Code.

Illustration: Python

1 Python-Illustration 1

```
In [ ]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd

        from scipy.stats import uniform, norm
        from pandas_datareader import data
        from scipy.fftpack import rfft, irfft, fft, rfft, dct, idct
        from scipy.cluster.vq import kmeans2 as kmeans
        from mpl_toolkits.mplot3d import Axes3D
        from sklearn.preprocessing import scale as scale_this
        %matplotlib inline
```

```
In [ ]: # Daten vom statistischen Bundesamt Deutschland:
        DATAPATH = '/Users/mfulmek/Work/Brotberuf/Vortraege/AVOE2018/destatis/'
        # Graphiken:
        GRAPHPATH = '/Users/mfulmek/Work/Brotberuf/Vortraege/AVOE2018/graphics/'
```

```
In [ ]: # Einlesen der Daten: 1 Zeile
        GD = pd.read_excel(DATAPATH+'12613-0003.xlsx')
```

Erster Schritt: Visualisierung!

Auch wenn man zum Schluß hochkomplizierte mathematische Methoden und ausgefeilte Computerprogramme einsetzt: Datenanalyse beginnt fast immer *optisch* — man schaut sich die Daten einmal an!

Leistungsfähige Bildverarbeitung

Das kommt Ihnen allzu schlicht vor? — Auch moderne Data-Science-Workstations setzen viel mehr auf die *Graphikkarte* (GPU) als auf den Hauptprozessor (CPU): Warum sollten wir also auf die enorme Leistungsfähigkeit der "optischen Signalverarbeitung" des menschlichen Gehirns verzichten!

Jedenfalls kann man aus den Daten *mit wenigen Zeilen* Python–Code einfache Plots herstellen.

Illustration: Python

2 Python-Illustration 2

```
In [ ]: # Hilfsfunktion zum Plotten der Rohdaten:
        def rohdaten_plot(gender):
            """Hilfsfunktion: Plot der Todesfall-Rohdaten für Männer ('m') oder Frauen (
            myplot = DATEN[gender]['roh'].plot(
                title=f'Todesfälle: {DATEN[gender]["bez"]} in Deutschland, 2011-2015',
                figsize=(12,8)
            )
            myplot.set_xlabel('Alter')
            myplot.set_ylabel('Anzahl Todesfälle')
            myplot.grid(b=True)
            plt.xticks(np.arange(0, 101, 5))
            plt.savefig(GRAPHPATH+f'roh_{gender}.eps')
            return myplot
```

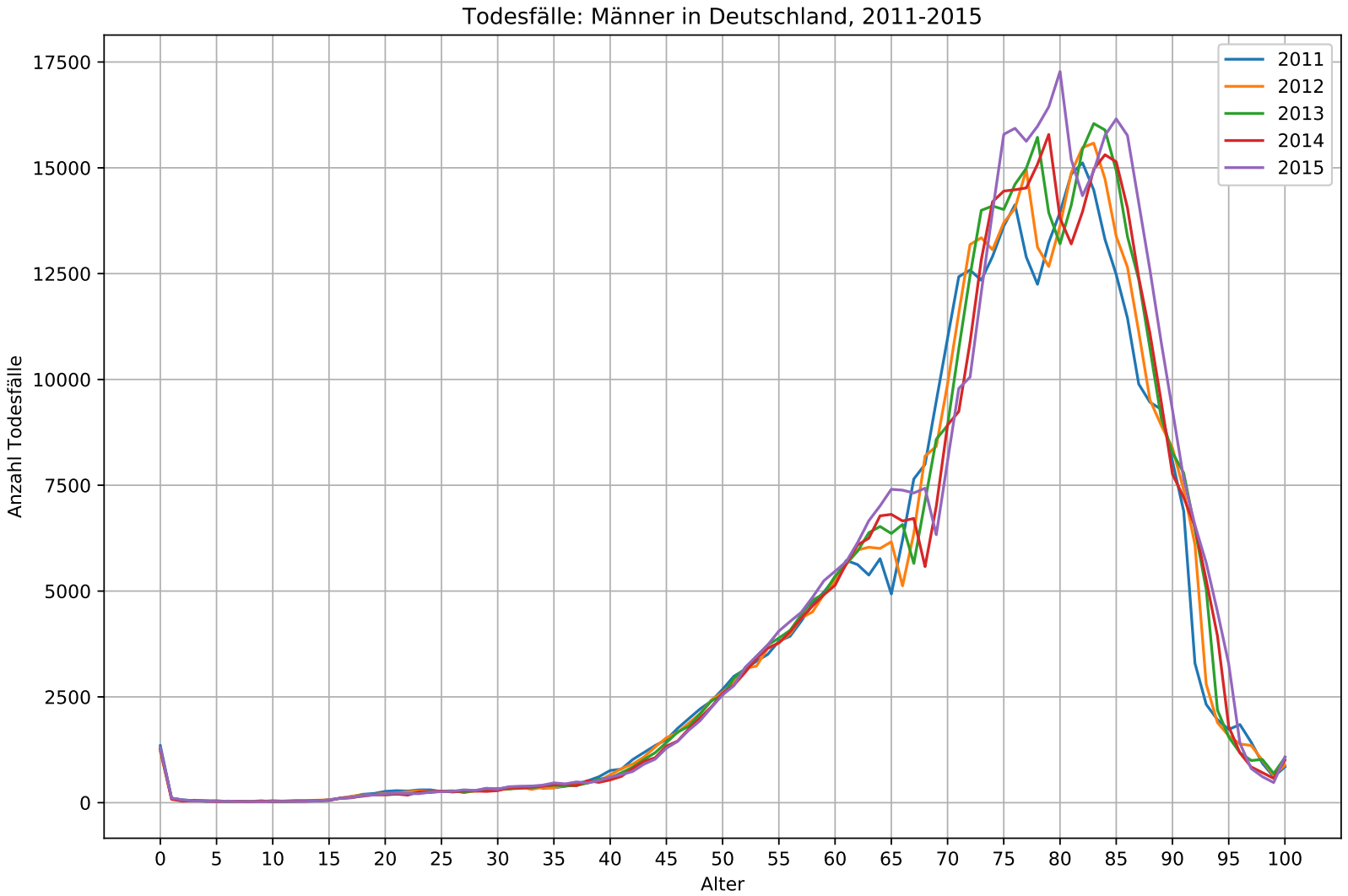
Übrigens: Slides und Python–Notebook im Internet

Unterlagen im Internet

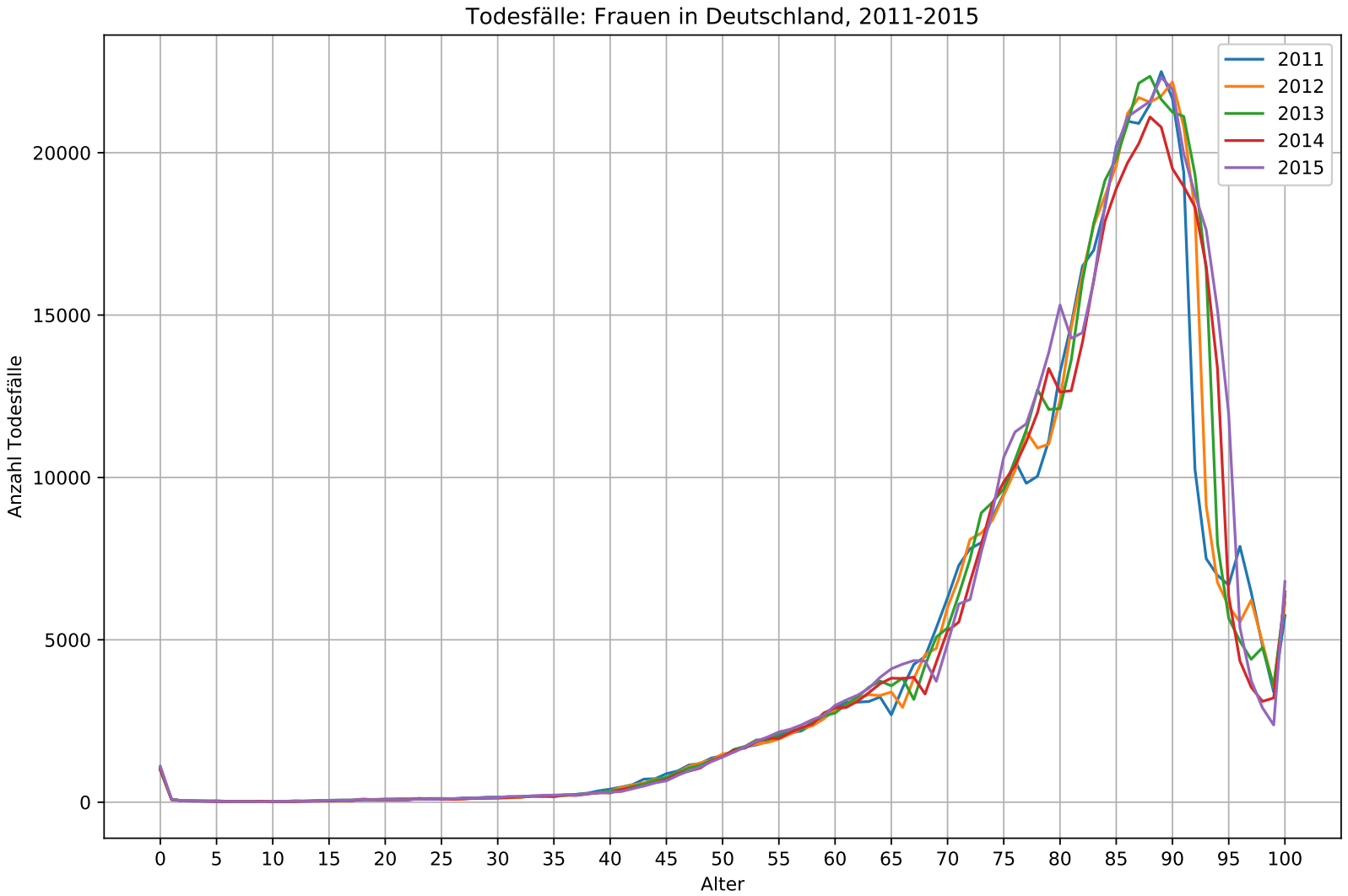
Bei Interesse finden Sie die Details (Vortragsfolien und IPython–Notebook) zu meinem Vortrag auf meiner Homepage

`http://www.mat.univie.ac.at/~mfulmek/avoe2018.shtml`.

Todesfälle, Männer



Todesfälle, Frauen



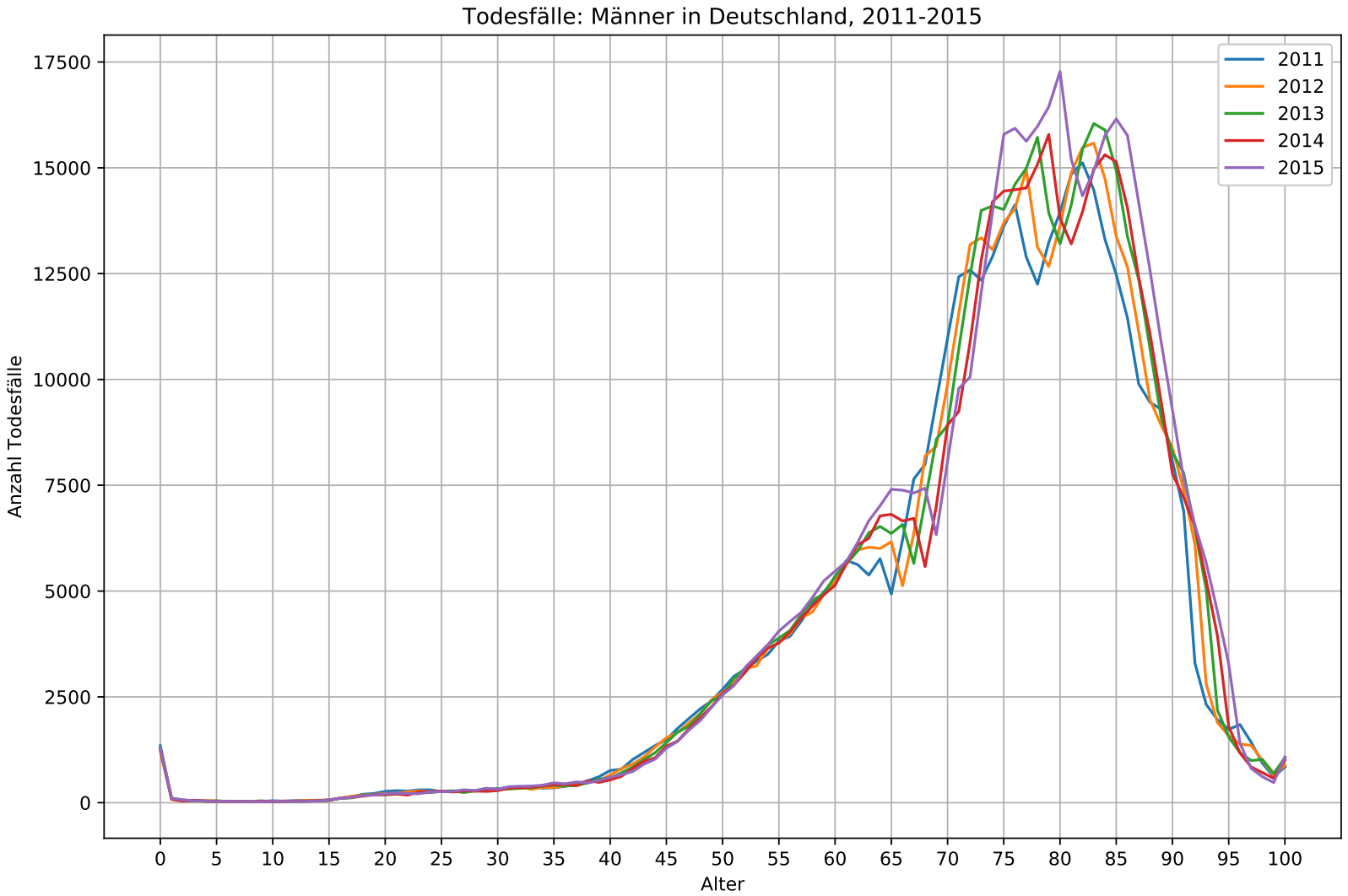
Erste Einsichten: Verschiebungen im Zeitablauf

Tatsächlich erkennt man — nur durch *Betrachtung* — der Plots unter anderem, daß die Todesfallzahlen (als Funktion des Lebensalters)

- nach Erreichen des ersten Lebensjahres stark zurückgehen,
- und danach monoton ansteigen, bis ein erstes lokales Maximum (etwa bei Alter 63) erreicht wird, das von einem kurzzeitigen Rückgang (etwa bis Alter 67) gefolgt wird.

Die erste Beobachtung entspricht natürlich der noch immer relativ hohen Säuglingssterblichkeit, die zweite Beobachtung könnte man vielleicht durch den belebenden Effekt der Pensionierung erklären — auffällig ist aber, daß die Kurven für die verschiedenen Jahre 2011 bis 2015 bis zum Pensionsalter sehr ähnlich sind, daß sich danach aber das lokale Maximum *von Jahr zu Jahr verschiebt*, und zwar "nach rechts oben".

Nochmals: Todesfälle, Männer



Veränderungen bei der Sterblichkeit im Zeitablauf?

Es ist ein vieldiskutiertes Phänomen, daß die Sterblichkeiten im Zeitablauf nicht konstant bleiben; vereinfacht gesagt: Dank der verbesserten medizinischen Versorgung steigt die Lebenserwartung.

Bekanntlich versucht die Versicherungswirtschaft, dies durch die Verwendung von *Generationensterbetafeln* adäquat abzubilden, bei denen die Sterblichkeit nicht nur von Alter und Geschlecht, sondern auch vom Geburtsjahrgang abhängt. Ich will hier aber nicht versuchen, die beobachteten Verschiebungen bei den Todesfallzahlen in diesem Sinne zu deuten — dafür ist mein Datenmaterial auch viel zu mager.

Stattdessen werde ich weiter versuchen, Sie davon zu überzeugen, daß man "mächtige mathematische Werkzeuge" mit sehr geringem Aufwand zum Einsatz bringen kann, wenn man sich der vorhandenen Software–Tools bedient.

Differenzieren verstärkt Schwankungen, Integrieren glättet

Einfache Faustregel

Differenzieren einer Funktion (Differenzenbildung einer Zahlenfolge) verstärkt Schwankungen, Integrieren einer Funktion (Summieren einer Zahlenfolge) glättet Schwankungen aus.

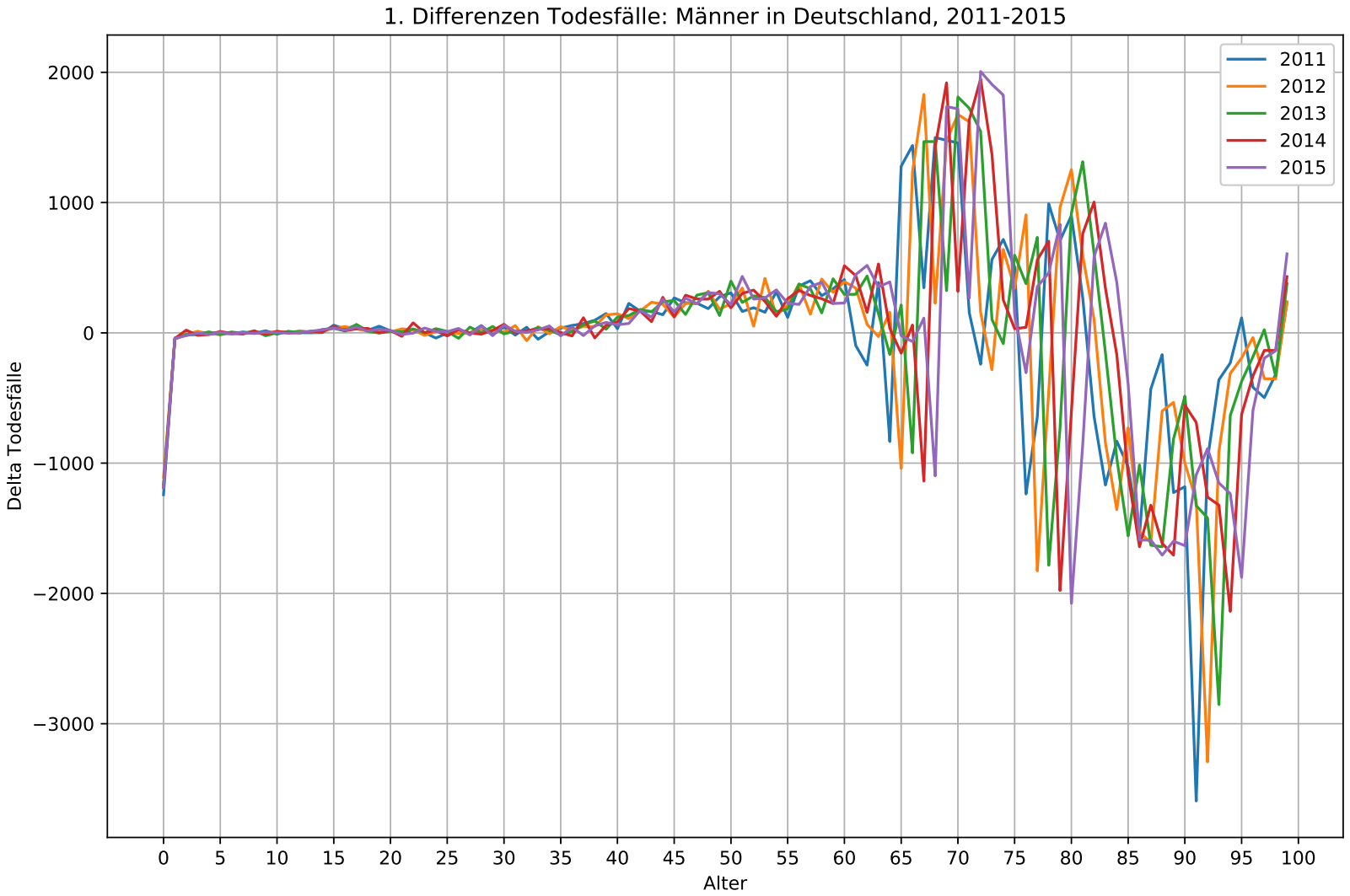
Gemäß obiger "Faustregel" bilden wir die ersten Differenzen (in Python: *eine Zeile* Programmcode) der Todesfallzahlen und schauen uns wieder das Ergebnis an:

Illustration: Python

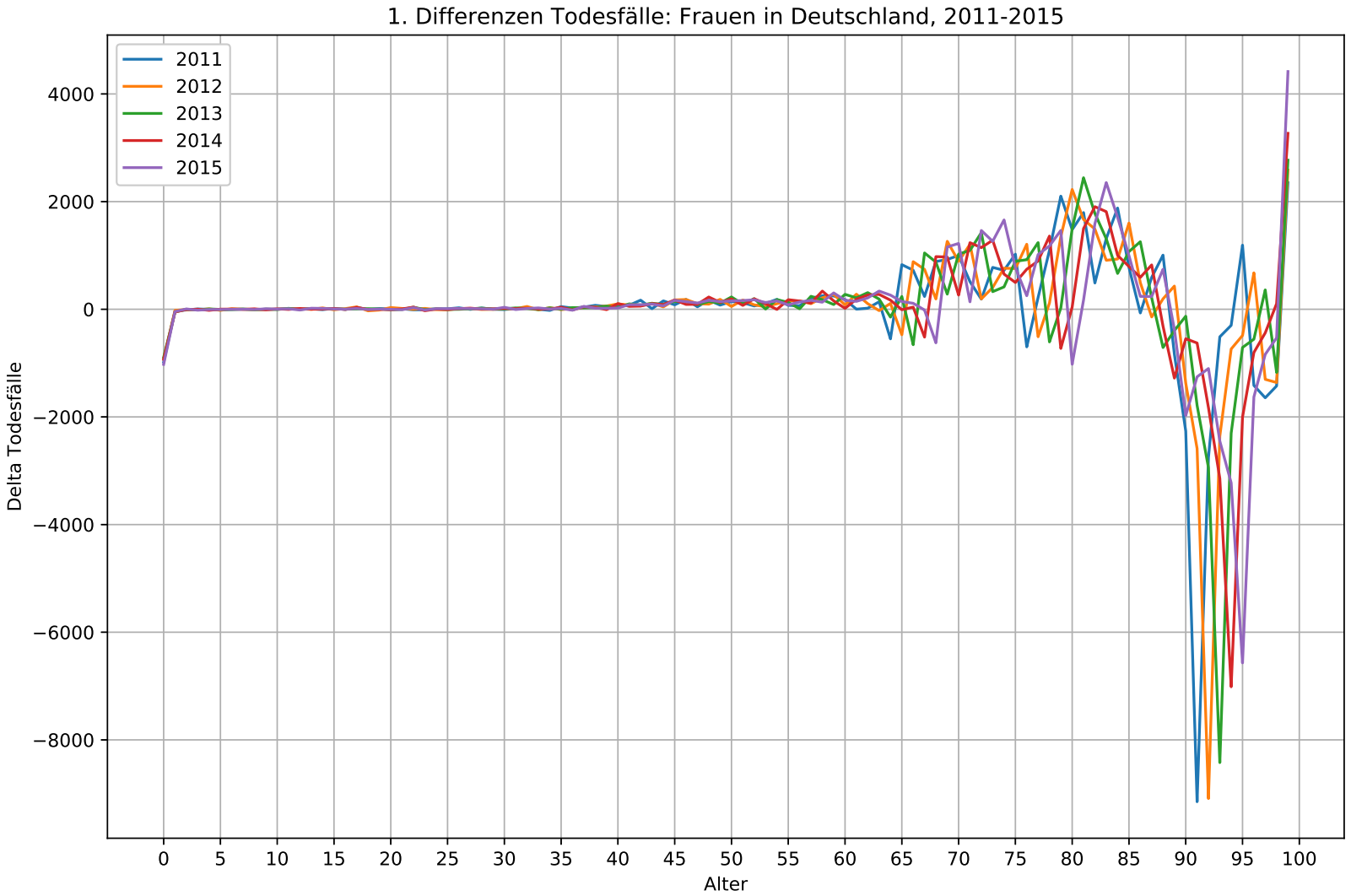
3 Python-Illustration 3

```
In [ ]: # Wir bilden die ersten Differenzen
        GMD_DIFF = pd.DataFrame(GMD[1:].values - GMD[:-1].values, columns=COLUMNS)
        GFD_DIFF = pd.DataFrame(GFD[1:].values - GFD[:-1].values, columns=COLUMNS)
        # Eine Hilfsfunktion zum Plotten der Differenzen:
        def diff_rohdaten_plot(gender):
            """Hilfsfunktion: Plot der ersten Differenzen der Todesfall-Rohdaten für Mä
            myplot = DATEN[gender]['delta'].plot(
                title=f'1. Differenzen Todesfälle: {DATEN[gender]["bez"]} in Deutschland
                figsize=(12,8)
            )
            myplot.set_xlabel('Alter')
            myplot.set_ylabel('Delta Todesfälle')
            plt.xticks(np.arange(0, 101, 5))
            myplot.grid(b=True)
            plt.savefig(GRAPHPATH+f'delta_roh_{gender}.eps')
            return myplot
```

Erste Differenzen der Todesfälle, Männer



Erste Differenzen der Todesfälle, Frauen

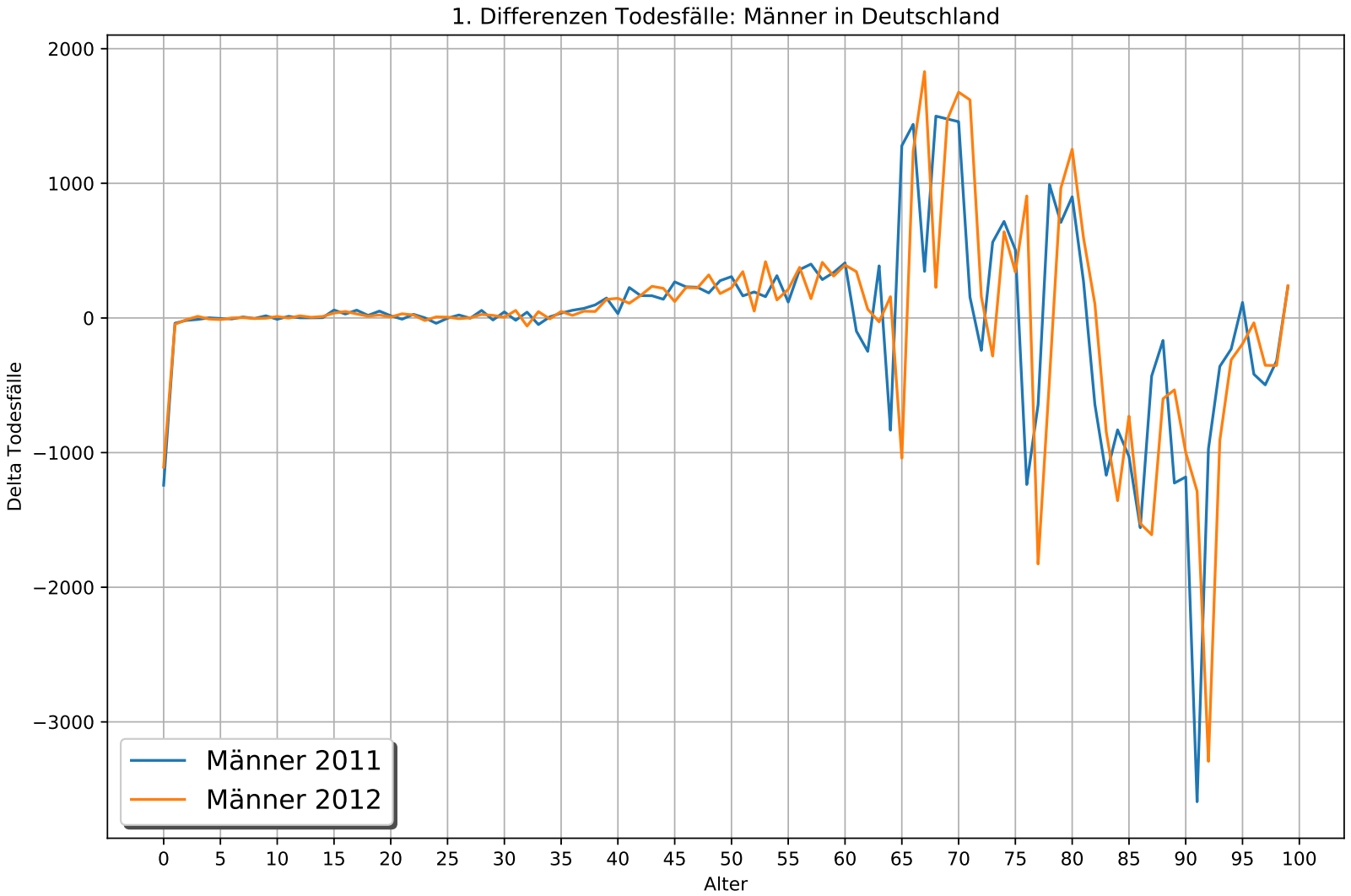


Mit etwas Phantasie: "Schwingung" erkennbar

Noch deutlicher als zuvor ist nun die "Wanderung der Zacken des Funktionsgraphen" im Zeitablauf 2011 bis 2015 erkennbar.

Schaut man sich die Kurven für einzelne Jahre (z.B. 2011 und 2012 in der folgenden Graphik) an, so kann man (ab Alter etwa 65) eine "Überlagerung von Schwingungen" erkennen (mit etwas Phantasie, oder einem Hintergrund in Signalverarbeitung;-).

Erste Differenzen der Todesfälle, Männer, 2011 und 2012



Zerlegung in Schwingungen und Ausfiltern der hohen Frequenzen: Glätten der Sterbewahrscheinlichkeiten

Sie kennen sicherlich die lehrbuchmäßige Darstellung: Aus den "roh geschätzten Sterbewahrscheinlichkeiten", die naturgemäß statistische Schwankungen aufweisen, werden durch geeignete Verfahren (z.B. Ausgleichungsrechnung mit Polynomfunktionen) die "geglätteten" Wahrscheinlichkeiten bestimmt.

Eine solche Glättung können wir für unsere Daten nun wie folgt durchführen:

Fast Fourier Transform & Inverse Fast Fourier Transform

- Wir bestimmen die "Schwingungen, die in den Kurven stecken" mit *Fast Fourier Transform*,
- Wir lassen alle hochfrequenten Schwingungen (die wir als "white noise" — also als statistisches Rauschen — verdächtigen) einfach weg, setzen also die entsprechenden Fourier–Koeffizienten auf Null,
- Wir setzen die Kurve ohne die weggelassenen Hochfrequenzen wieder zusammen, mit *Inverse Fast Fourier Transform*.

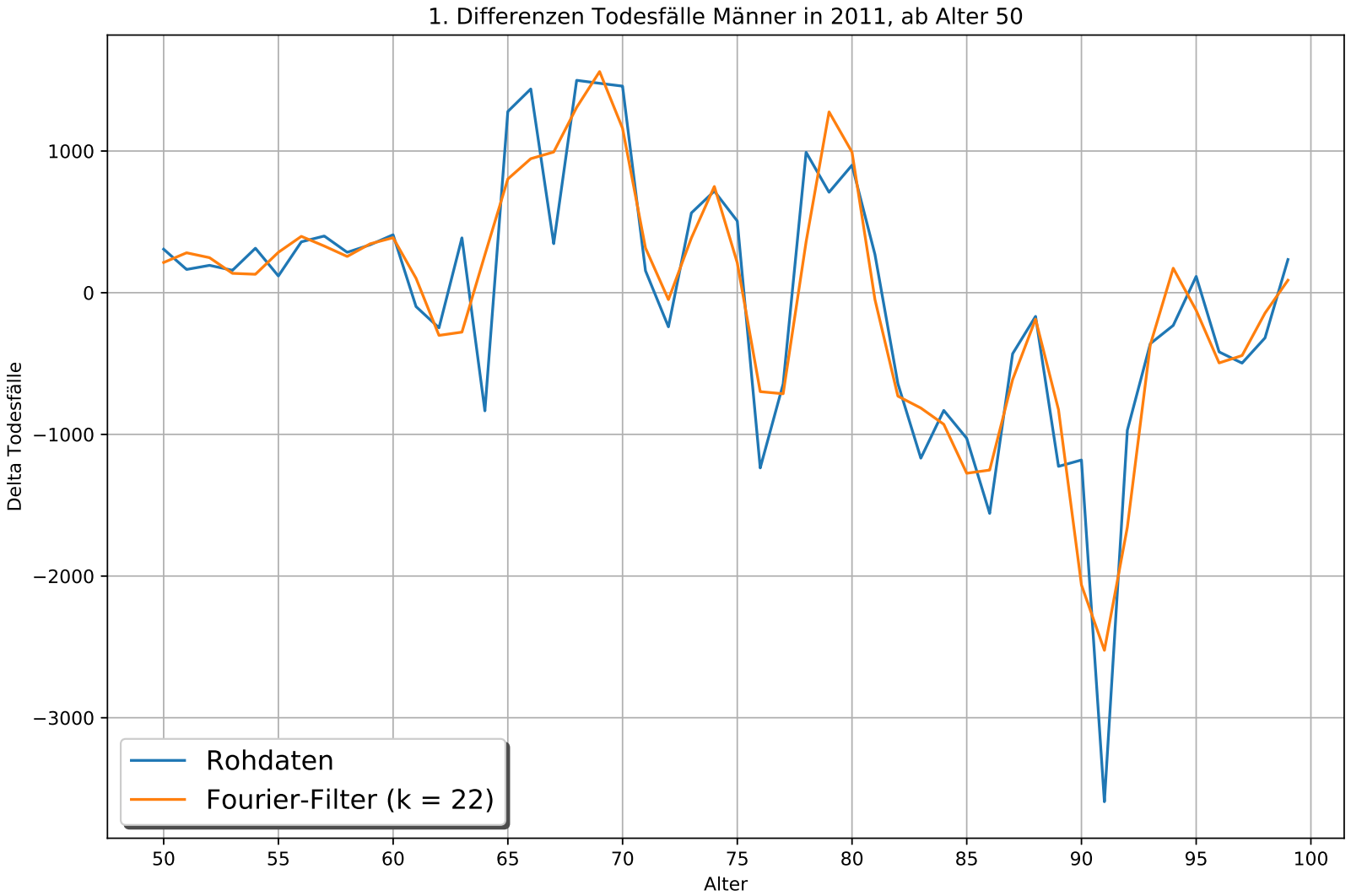
Diese drei Schritte entsprechen tatsächlich *drei Zeilen* Python–Code!

Illustration: Python

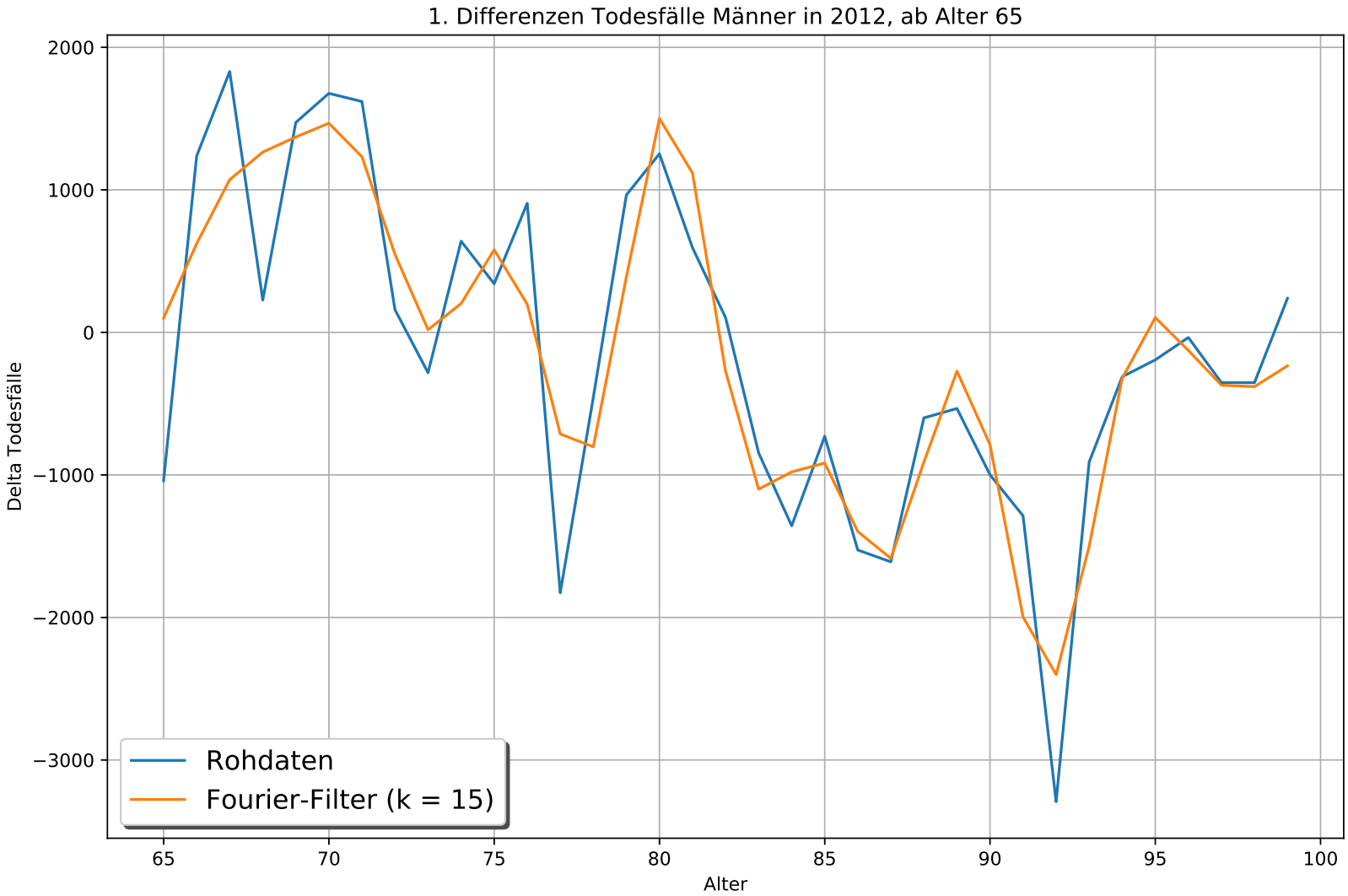
4 Python-Illustration 4

```
In [ ]: # Eine Hilfsfunktion zum Plotten der Fourier-Anpassung:
        def diff_fourier_plot(gender, year, start, k):
            """Hilfsfunktion: Plot der ersten Differenzen der Todesfall-Rohdaten für Männer ('m
            data = DATEN[gender]['delta'][year][start:]

            # Glätten der Daten mit Fourier-Transformation in 3 Zeilen:
            fcoeffs = rfft(data) # Fourier-Transformation
            fcoeffs[k+1:] = 0.0 # Entfernen der hohen Frequenzen
            fitted_data = irfft(fcoeffs) # Inverse Fourier-Transformation

            # Formatierte Graphik
            fig, ax = plt.subplots(figsize=(12,8))
            ax.plot(data.values, label='Rohdaten')
            ax.set_title(f'1. Differenzen Todesfälle {DATEN[gender]["bez"]} in {year}, ab Alter
            ax.plot(fitted_data, label=f'Fourier-Filter (k = {k})')
            ax.set_xlabel('Alter')
            ax.set_ylabel('Delta Todesfälle')
            legend = ax.legend(loc='lower left', shadow=True, fontsize='x-large')
            xticks_range = np.arange(0, 100 - start + 1, 5)
            labels = plt.xticks(xticks_range, xticks_range + start)
            ax.grid(b=True)
            plt.savefig(GRAPHPATH+f'ff_{gender}_{year}_{start}_{k}.eps')
            plt.show()
```

Schnelle Fouriertransformation ab 50, Ausfiltern $k > 22$



Schnelle Fouriertransformation ab 65, Ausfiltern $k > 15$



Fouriertransformation

Mathematik

Die *Mathematik*, die der Fouriertransformation (Entwicklung nach einem vollständigen Orthonormalsystem in einem Hilbertraum) zugrundeliegt, ist *nicht elementar*.

Informatik

Die *Programmierung* der schnellen (diskreten) Fouriertransformation ist *trickreich*.

Praktische Anwendung

Die *Verwendung* der schnellen Fouriertransformation ist aber *kinderleicht*!

Disclaimer

Die vorigen Beispiele dienten *nur der Illustration*: Keineswegs soll damit behauptet werden, daß Fouriertransformation ein geeignet(er)es Glättungsverfahren für die Erstellung von Sterbetafeln liefert!

Sterbetafeln

Bei unseren *Todesfallzahlen* fehlt die Informationen, *wieviele* x–Jährige durch die Todesfälle im x-ten Lebensjahr dezimiert wurden: Es ist also *nicht möglich*, die (rohen) Sterbewahrscheinlichkeiten

$$q_x \simeq \frac{\text{im } x\text{—ten Jahr Verstorbene}}{x\text{-Jährige}}$$

direkt aus den Daten zu schätzen.

Um dennoch weiterzumachen (es geht hier ja nur um eine *Illustration* von modernen Software–Tools), fasse ich die vorliegenden Zahlen als erste Differenzen einer Ausscheideordnung auf (mit Startwert gleich der Summe über alle Todesfälle).

Nur zur Illustration!

Vorgangsweise sachlich unrichtig, dient nur zur Illustration!

Bei der sachlich richtigen Vorgangsweise für *Kohortensterbetafeln* müßten u.a. die Wanderungssalden berücksichtig werden: Diese Daten stehen mir hier nicht zur Verfügung.

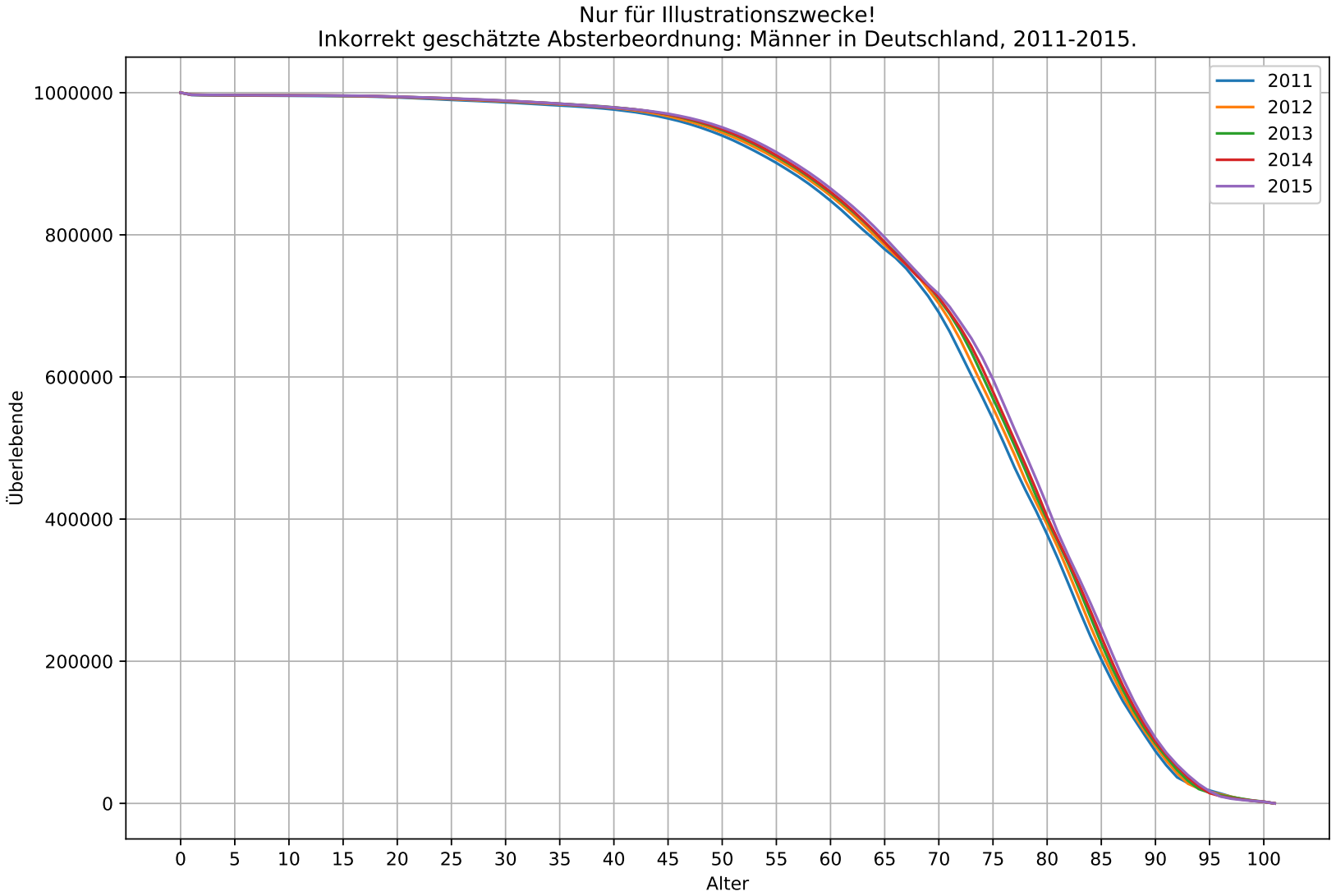
Die (falsch geschätzten) Ausscheideordnungen erhält man jedenfalls durch *wenige* Zeilen Python–Code:

Illustration: Python

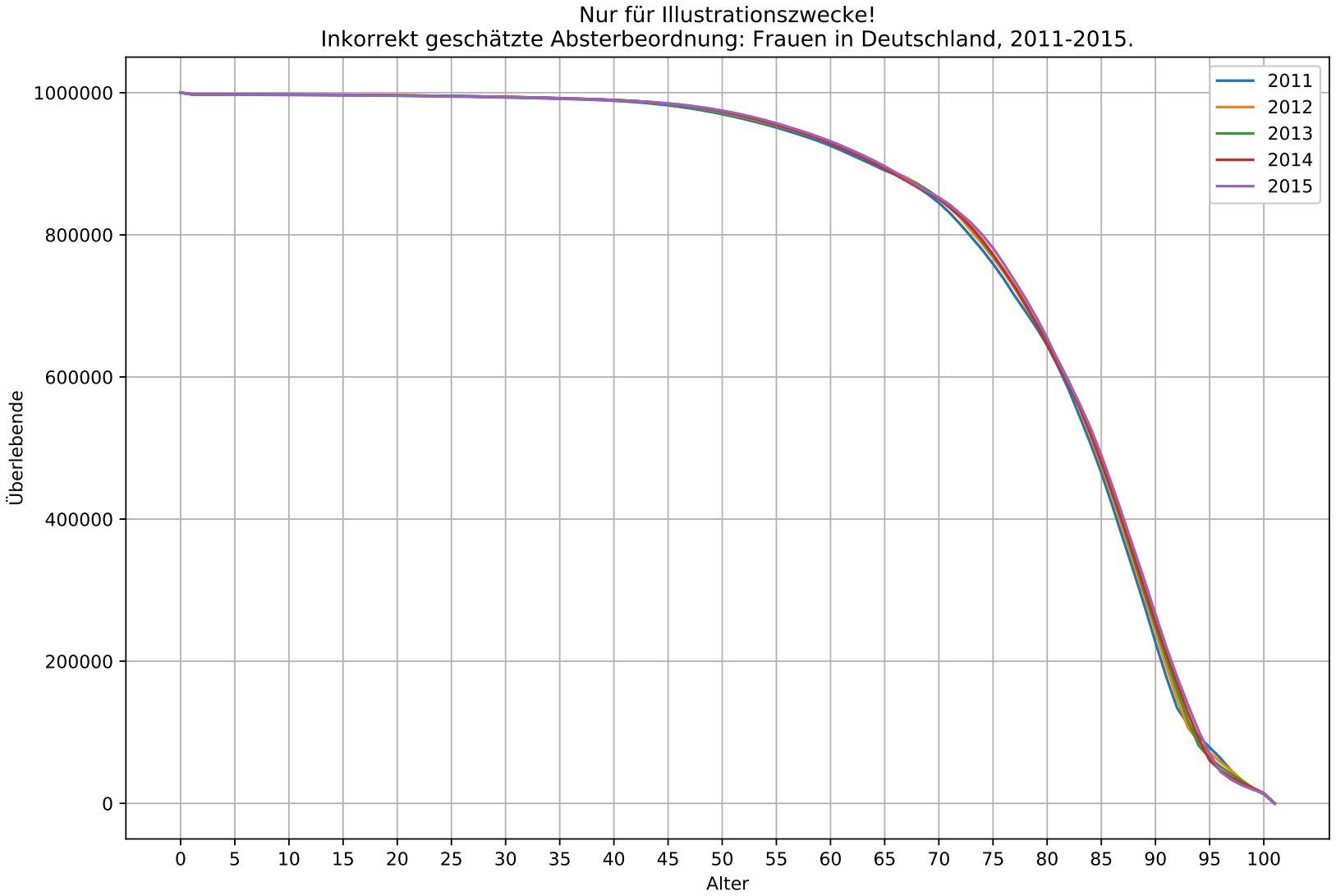
5 Python-Illustration 5

```
In [ ]: # "Ausscheideordnung" aus den Todesfalldaten (sachlich inkorrekt, nur zur Illustration!!
        GMD_SUMS = dict(zip(GMD.columns, [GMD[column].sum() for column in GMD.columns]))
        GFD_SUMS = dict(zip(GFD.columns, [GFD[column].sum() for column in GFD.columns]))
        GMD_AO=pd.DataFrame(
            np.array([np.r_[GMD[column].values[::-1].cumsum()[::-1]/GMD_SUMS[column]*(10**6), np
            columns = GMD.columns
        )
        GFD_AO=pd.DataFrame(
            np.array([np.r_[GFD[column].values[::-1].cumsum()[::-1]/GFD_SUMS[column]*(10**6), np
            columns = GFD.columns
        )
```

Ausscheideordnung (nur zur Illustration) für Männer



Ausscheideordnung (nur zur Illustration) für Frauen



Sensitivitätsanalysen

Eine naheliegende Fragestellung ist sicherlich: "Welche Unterschiede in den versicherungsmathematischen Barwerten ergeben sich, wenn man die 5 verschiedenen (nur für Illustrationszwecke geschätzten!) Sterbetafeln der Jahre 2011 bis 2015 betrachtet?"

Wir wollen diese *Sensitivitätsanalyse* folgendermaßen konkretisieren: Welche relativen Änderungen ergeben sich im Vergleich zur Ausscheideordnung 2011

- für den Barwert einer nachschüssigen Leibrente ab Alter x
- bei Rechnungszins 1.5%
- bei Verwendung der "Ausscheideordnungen 2012 bis 2015"?

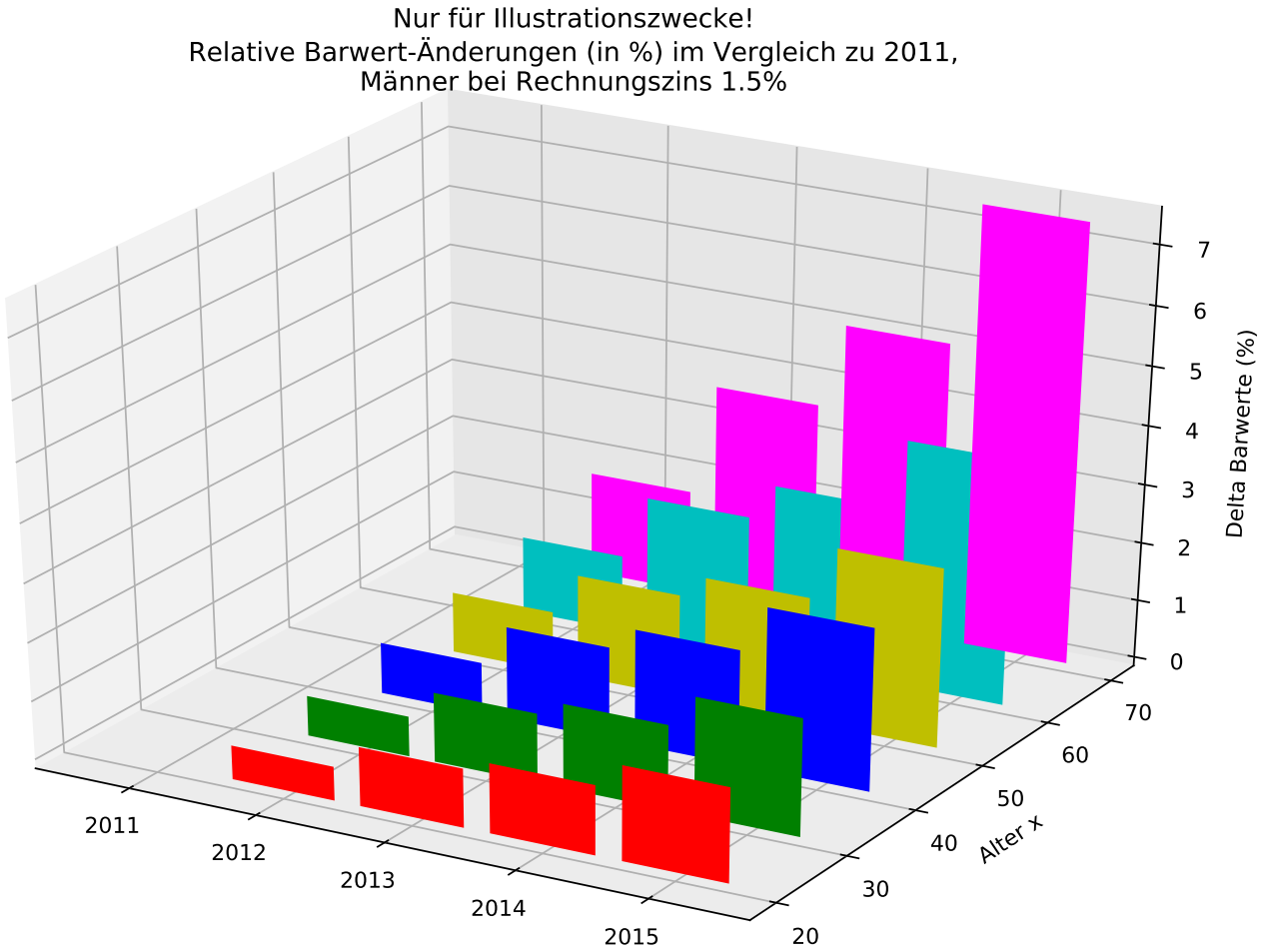
Die Berechung des gesuchten Barwerts entspricht einer *dreizeiligen Funktion* in Python, das Erzeugen einer hübschen Graphik ist in etwa 20 Zeilen erledigt:

Illustration: Python

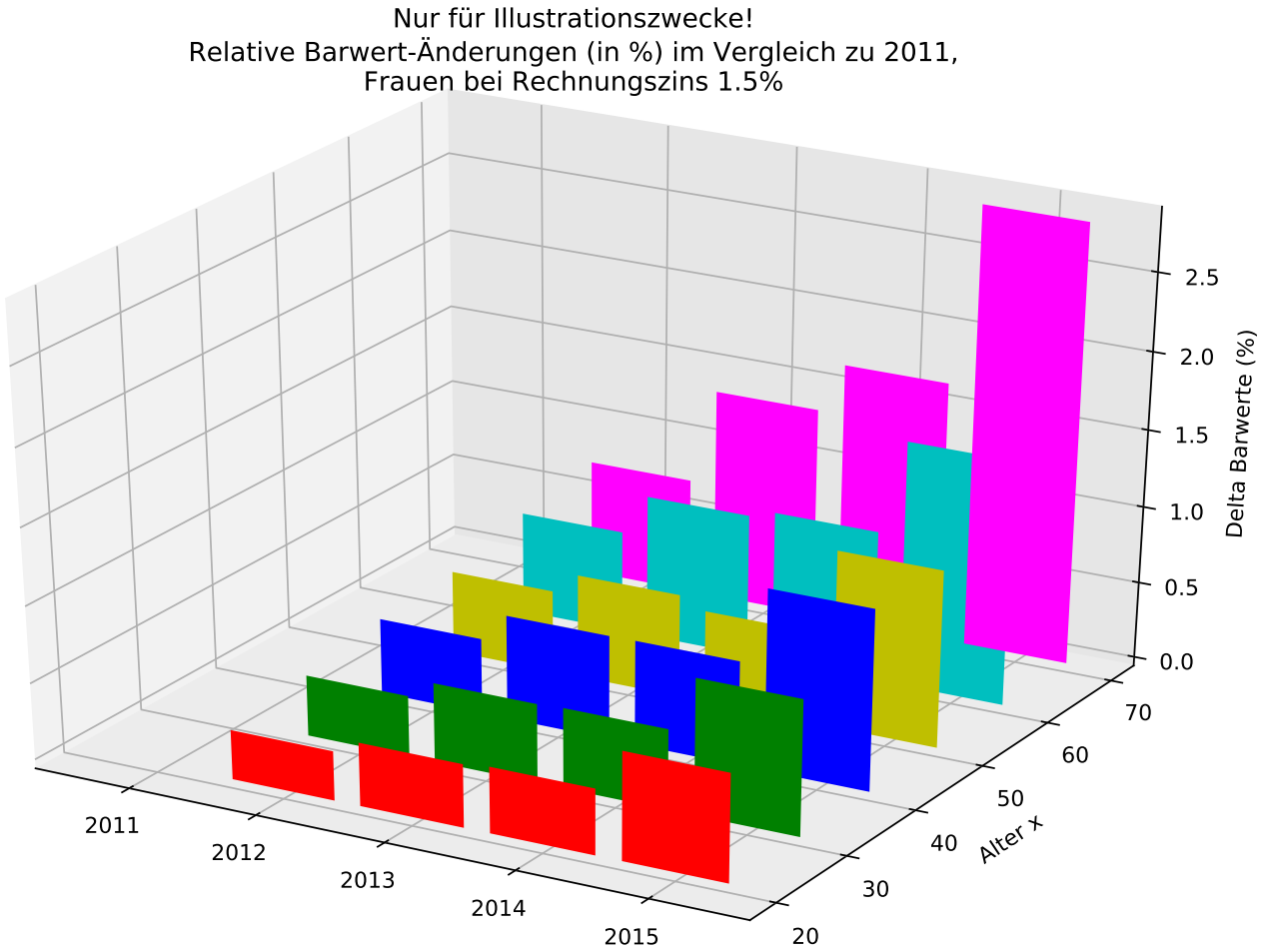
6 Python-Illustration 6

```
In [ ]: def actuarial_pv(x, ao, rz):
            """Barwert einer lebenslänglichen nachschüssigen Leibrente ab Alter x"""
            survival_probabilities = ao[x+1:]/ao[x]
            discount_factors = np.array([(1+rz/100) ** (-t) for t in range(1, len(ao) - x)])
            return np.inner(survival_probabilities, discount_factors)
        def sensitivity_plot(gender, rz):
            """Sehr einfache Sensitivitätsanalyse"""
            ao = DATEN[gender]['ao']
            fig = plt.figure(figsize=(12,8))
            ax = fig.add_subplot(111, projection='3d')
            for colour, x in zip(['r', 'g', 'b', 'y', 'c', 'magenta'], [20, 30, 40, 50, 60, 70]):
                xs = range(5)
                ys = np.array([actuarial_pv(x, ao[year], rz) for year in COLUMNS])
                ref_value = ys[0]
                ys = (ys - ref_value)/ref_value * 100
                cs = [colour] * len(xs)
                ax.bar(xs, ys, zs=x, zdir='y', color=cs, alpha=0.8)
            # ax.set_xlabel('X')
            ax.set_title(f'Nur für Illustrationszwecke!\nRelative Barwert-Änderungen (in %) im V
            ax.set_ylabel('Alter x')
            ax.set_zlabel('Delta Barwerte (%)')
            plt.xticks(np.arange(5), COLUMNS)
            plt.savefig(GRAPHPATH+f'sens_{gender}_{rz}.pdf')
```

Sensitivitätsanalyse Männer, Rechnungszins 1.5



Sensitivitätsanalyse Frauen, Rechnungszins 1.5



Biometrische Risiken, klassisch

Die klassische versicherungsmathematische Modellierung von biometrischen Risiken (Passivseite) ist ja recht einfach:

- Bestimme den (versicherungsmathematischen) *Barwert* der Leistungen,
- (und verlasse Dich auf das *Gesetz der großen Zahlen*;-).

Tatsächlich kann man aber mit wenig Aufwand mehr Information gewinnen: Zum Beispiel kann man den aus dem Risikomanagement von Banken bekannten Begriff *Value at Risk* ganz leicht auf ein Portfolio von Rentenversicherungen übertragen.

Value at Risk (VaR)

Definition (Value at Risk)

Der Value at Risk (VaR) für ein gegebenen Portfolio zu einem vorgegebenen Konfidenzniveau α ist einfach das $(1-\alpha)$–Quantil der dem Portfolio entsprechenden Gewinn– und Verlustverteilung.
Also:

$$P\left(GuV \leq VaR\right) = 1 - \alpha.$$

Das Problem in der Praxis liegt darin, daß die *Verteilung von Gewinnen und Verlusten* (GuV) geeignet *geschätzt* werden muß: Das kann für große Portfolios mit komplexen Finanzinstrumenten schon recht mühsam werden.

Value at Risk (VaR) mit Monte–Carlo–Simulation

Mit *Monte–Carlo–Simulation* läßt sich aber ein Value at Risk für ein Portfolio leicht schätzen (jedenfalls im Prinzip): Wenn man z.B. für 100 zufällig erzeugte Szenarien die entsprechenden Gewinne/Verluste für das Portfolio bestimmt und die so erhaltene Liste von 100 Zahlen aufsteigend sortiert, dann ist die beste Schätzung für die Value at Risks zum Konfidenzniveau

- $\alpha = 99\%$ einfach der erste (also der kleinste) Wert in dieser Liste $(1 - \alpha = \frac{1}{100})$,
- $\alpha = 95\%$ einfach der fünfte Wert in dieser Liste $(1 - \alpha = \frac{5}{100})$.

Monte–Carlo–Simulation für ein Portfolio von Rentenversicherungen

Für Illustrationszwecke betrachten wir ein Portfolio aus N Leibrenten (Rente jeweils $1/N$) für 20–jährige Männer: Aus der Ausscheideordnung ergeben sich sofort Wahrscheinlichkeiten, wie lange die Renten bezahlt werden müssen, und eine darauf basierende Monte–Carlo–Simulation (mit M Szenarien) ist in einer *vierzeiligen* Python–Funktion erledigt.

Wenn wir von den Mittelwerten der so simulierten Szenarien den Erwartungswert der Leistungen (also den versicherungsmathematischen Barwert einer Leibrente der Höhe 1 abziehen), dann erhalten wir eine (empirische) Verteilung von (einer ersten Näherung an die) versicherungstechnischen Gewinne/Verluste.

Illustration: Python

7 Python-Illustration 7

```
In [ ]: def mc_actuarial_pv(x, ao, rz, nof_persons, nof_scen):
            """Monte-Carlo-Simulation: Barwert einer lebenslänglichen nachschüssigen Leibrente d
            # Wahrscheinlichkeiten, das (x+k)-te Lebensjahr _nicht_ zu erreichen, für k = 1,2,3,
            probabilities = np.array(1.0 - ao[x+1:]/ao[x])
            # Rein finanzmathematische Barwerte von k jährlichen Zahlungen der Höhe 1, für k = 1
            pv = np.cumsum(np.array([0.0]+[(1+rz/100) ** (-t) for t in range(1, len(probabilitie
            # Erzeuge nof_scen gleichverteilte (Pseudo-)Zufallszahlen
            scenarios = uniform.rvs(size=nof_scen*nof_persons)
            # Simuliere nof_scen-mal eine Risikogemeinschaft von nof_persons x-Jährigen
            # und bestimme die mittlere Auszahlung einer Leibrente der Höhe 1 für jedes Szenario
            return np.sort(
                np.mean(
                    np.array([pv[np.searchsorted(probabilities, s)] for s in scenarios]).reshape
                    axis=(1,)
                )
            )
```

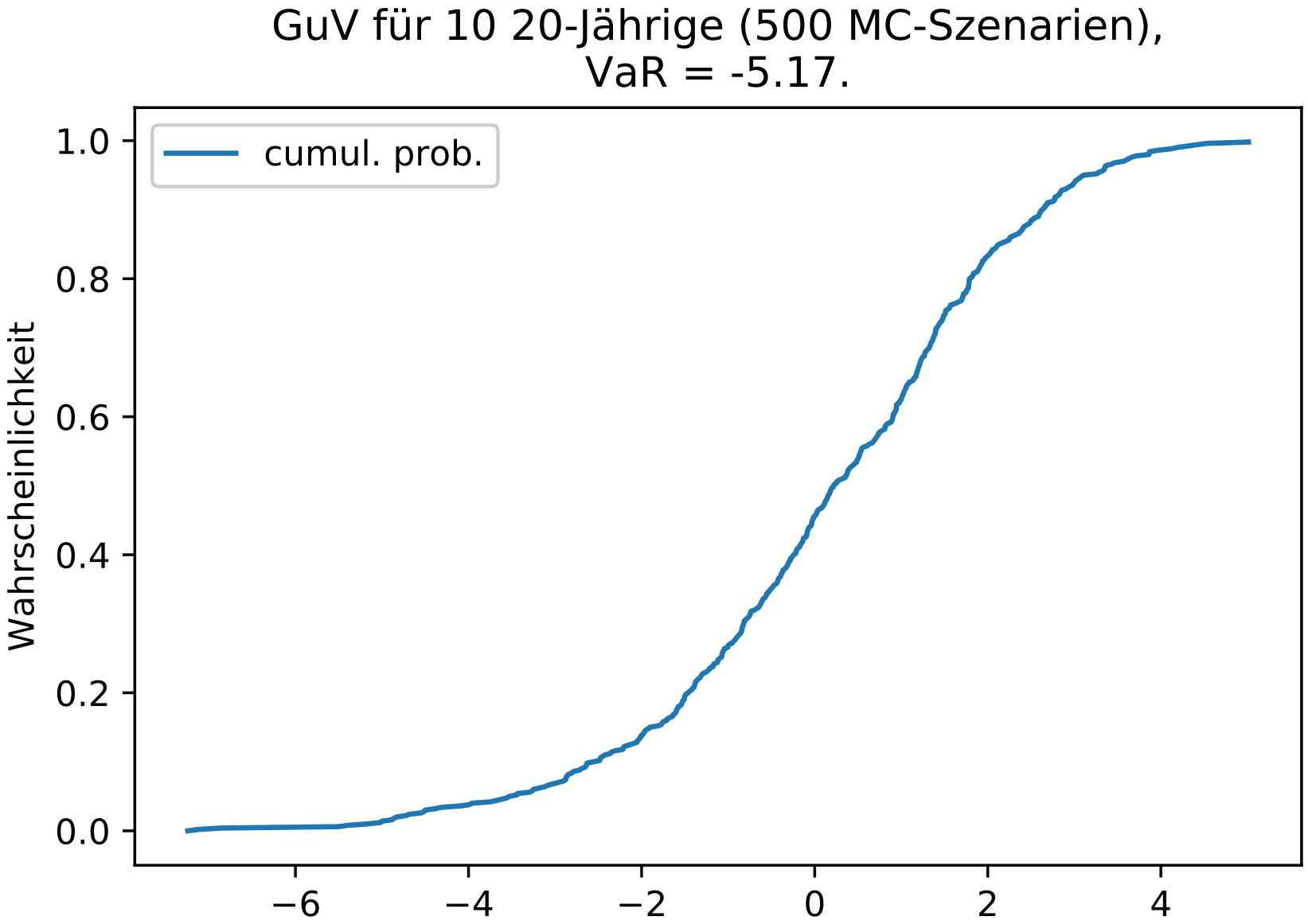
Das ist natürlich nur ein sehr einfaches Beispiel!

Beispiel ist leicht zu verallgemeinern

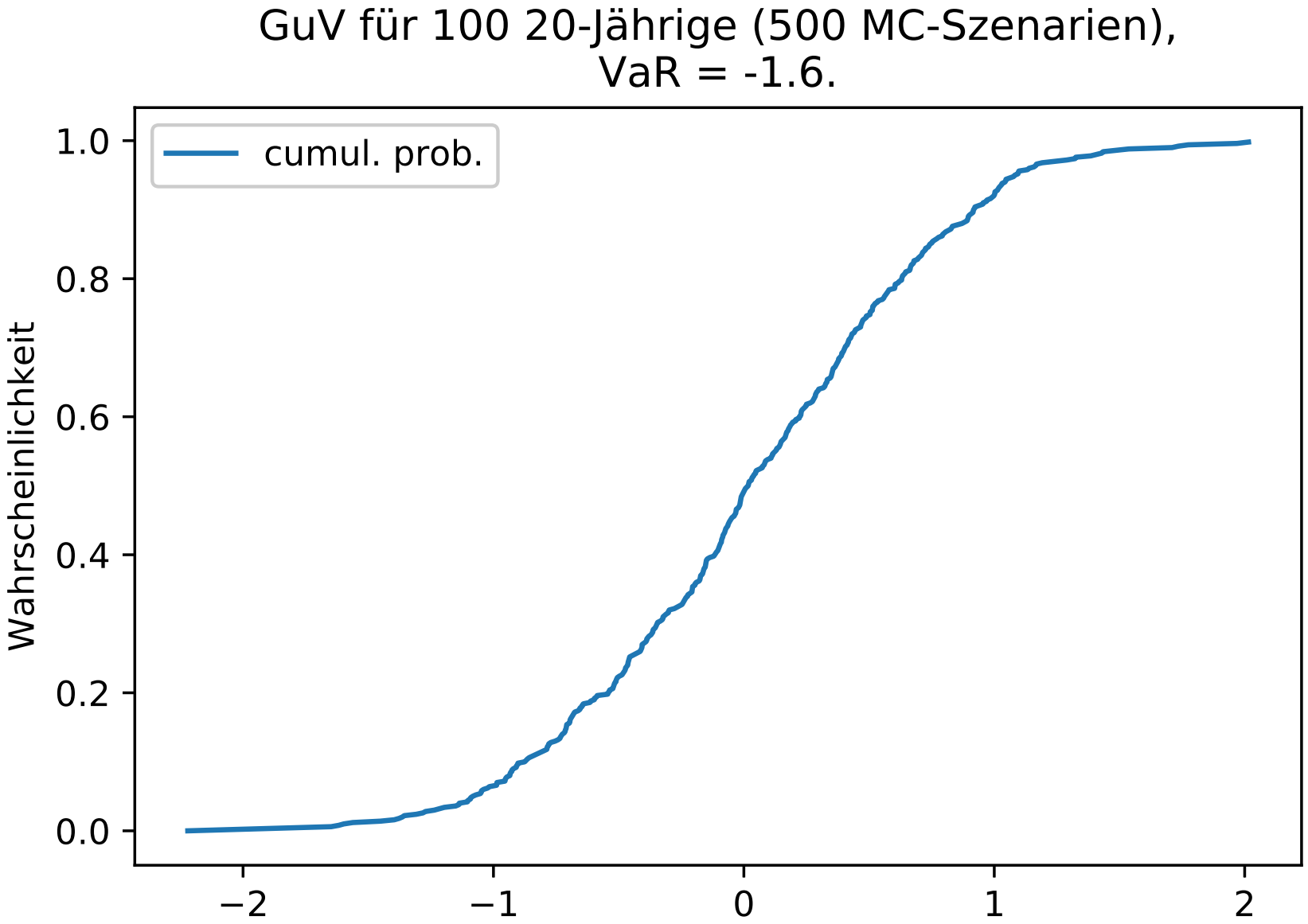
Für eine praktische Anwendung müßte man natürlich Portfolios mit *verschiedenen* Rentenhöhen und Lebensaltern betrachten, außerdem könnte man auch noch den Rechnungszins stochastisch modellieren: Diese Verallgemeinerungen wären auch gar nicht schwer zu implementieren, aber für meine Illustrationszwecke sollte das einfache Beispiel schon ausreichend sein.

Die folgenden Graphiken zeigen die Verteilung und den Value at Risk für $N = 10, 100$ und 1000; die Anzahl der Monte–Carlo–Szenarien ist immer $M = 500$.

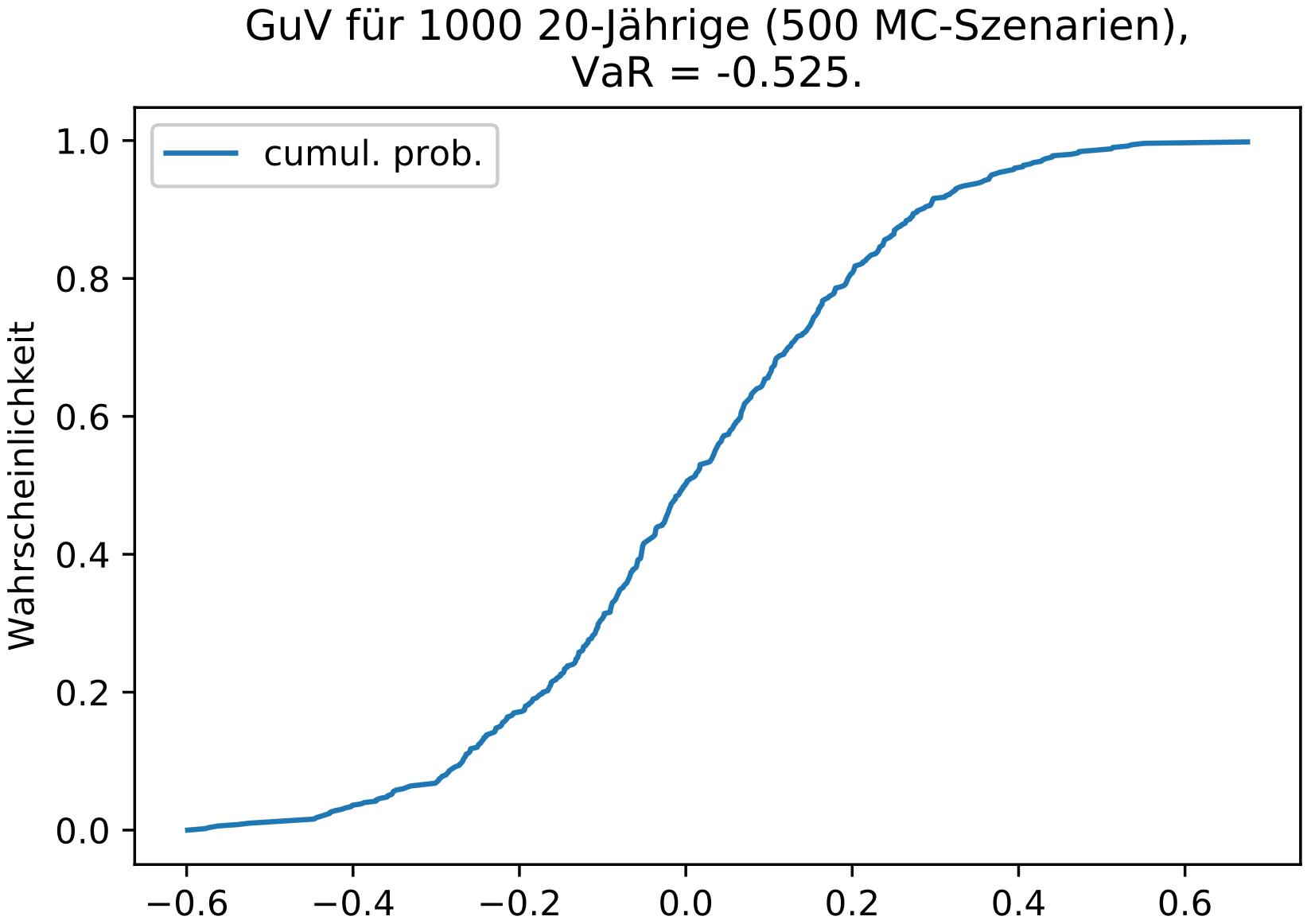
GuV und 99%–VaR für $N = 10$, $M = 500$.



GuV und 99%–VaR für $N = 100$, $M = 500$.



GuV und 99%–VaR für $N = 1000$, $M = 500$.



Interpretation der Graphiken

Die Graphiken schauen alle gleich (oder zumindest sehr ähnlich) aus? — Man beachte die *Skalierung* auf der x–Achse bzw. die immer kleiner werdenden Absolutbeträge der Values at Risk (Quantile):

Bankenwelt: Diversifikationseffekt

Das in unserem Beispiel klar zu beobachtende "Zusammenziehen" der GuV–Verteilung um den Erwartungswert, wenn die Anzahl der Instrumente im Portfolio größer wird, wird in der Bankenwelt als *Diversifikationseffekt* bezeichnet.

Versicherungswelt: Risikoausgleich im Kollektiv

Derselbe Effekt wird in der Versicherungswelt oft als *Risikoausgleich im Kollektiv* bezeichnet (in der klassischen Versicherungsmathematik aber meist nicht näher quantifiziert).

Clustering

Meine bisherigen illustrierenden Beispiele zu Data Science gehören klar in das Gebiet "Klassische Datenanalyse", und es ist nicht ganz leicht, von *Sterbetafeln* einen Bezug zu *Machine Learning* herzustellen — ich werde das jetzt trotzdem irgendwie versuchen;-):

Bei unserer allerersten Visualisierung der Todesfalldaten konnten wir "mit freiem Auge" zwei "Blöcke" identifizieren, die sich "qualitativ" voneinander unterschieden:

- Todesfallzahlen für Alter etwa 0 bis 60 zeigten geringe Schwankungen und sehr ähnliches Verhalten für alle Jahre 2011 bis 2015,
- Todesfallzahlen für Alter über 60 zeigten starke Schwankungen und eine "Verschiebung" im Ablauf der Jahre 2011 bis 2015.

Zweck von Clustering

Die allgemeine Fragestellung "In welche (sinnvollen) Teile (Cluster) zerfällt der Raum meiner Daten?" ist in vielen Situationen durchaus von konkretem Interesse; z.B. könnte bei Schadensfällen eine Zerlegung in 2 Cluster "problematisch/unproblematisch" nützlich sein.

Wenn aber ein Schadensfall durch (z.B.) 7 Parameter beschrieben wird, kommt man "mit freiem Auge" nicht weiter, weil die 7–dimensionalen Fall–Beschreibungen keine so einfache Visualisierung ermöglichen wie bei unserem Todesfallzahlen–Beispiel.

Ein verblüffend einfaches Verfahren liefert aber auch in hochdimensionalen Datenräumen solche Zerlegungen: Der *k–Means–Algorithmus* benötigt als Input nur die Daten und die Anzahl k der gewünschten Cluster!

k–Means Algorithmus

Die Idee ist, jeden Datenpunkt einem von k Mittelwerten (Means) zuzuordnen, und zwar jenem, dem er (in einer geeigneten Metrik) am nächsten ist. — Algorithmus: Wähle zu Beginn k *zufällige* Means

$$\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)} \text{ aus den Datenpunkten.}$$

Jeder Datenpunkt wird dem *nächstliegenden* Mean zugeordnet:

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\|^2 \leq \left\| \mathbf{x}_j - \mathbf{m}_n^{(t)} \right\|^2 \text{ für alle } n = 1, \ldots, k \right\}.$$

Dann werden die Means als *Schwerpunkte* der bisherigen Cluster $S_i^{(t)}$ neu berechnet:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j.$$

Das wird so lange wiederholt, bis sich die Cluster nicht mehr ändern.

Praktische Anwendbarkeit

Es ist vielleicht schwer zu glauben, daß ein so simples Verfahren etwa Vernünftiges liefert: Tatsächlich ist das aber in vielen praktisch relevanten Problemstellungen der Fall!

Außerdem ist es ganz einfach, die Sache auszuprobieren, denn man muß den Algorithmus nicht selbst programmieren: Nach dem Laden der entsprechenden Programmbibliothek genügen *zwei Zeilen* Python–Code!

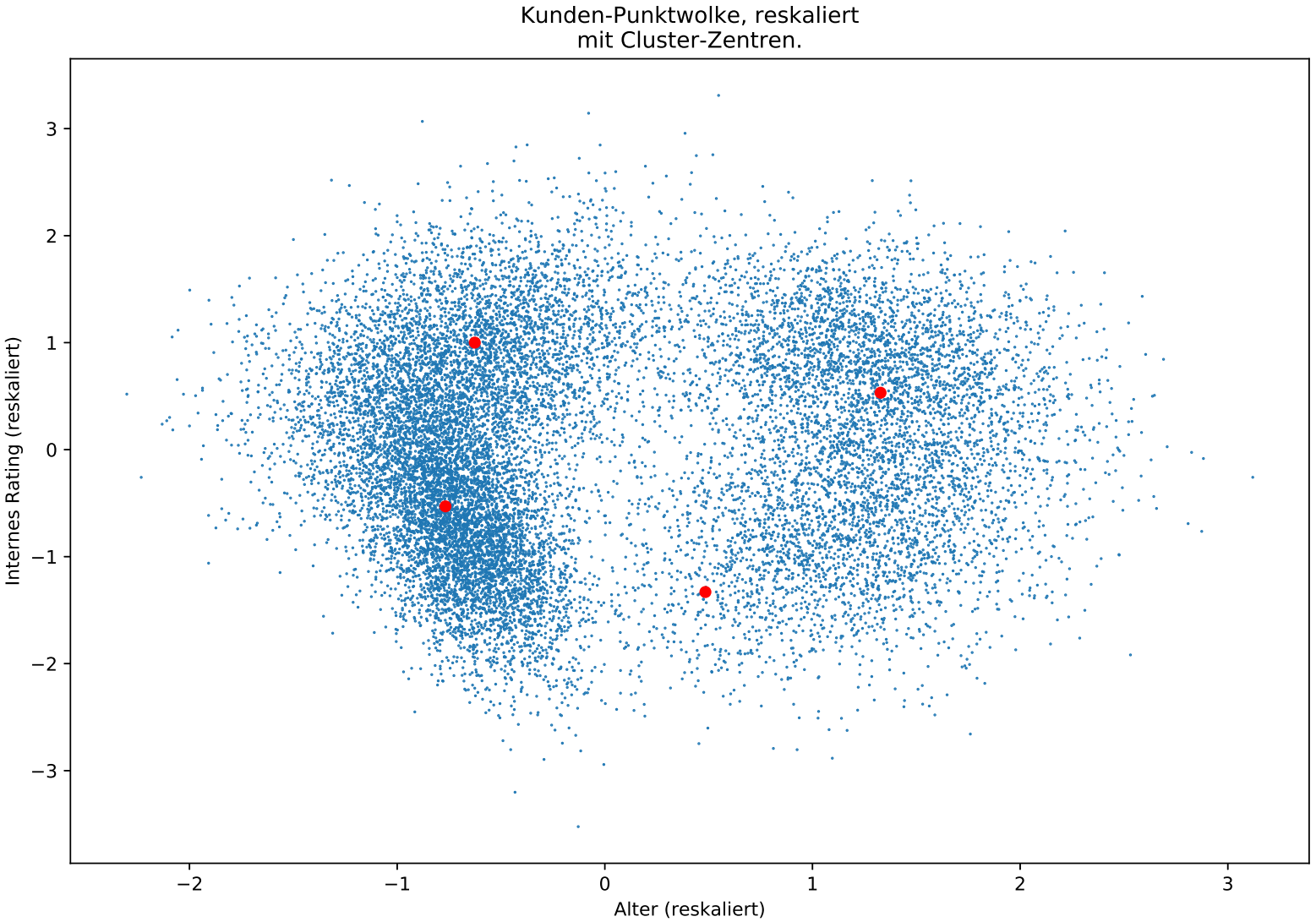
Zum Abschluß noch eine Illustration: Der k–Means–Algorithmus findet in *Sekundenbruchteilen* 4 Cluster–Zentren in einer 2–dimensionalen "Wolke" aus 16000 Punkten. (In Dimension 2 kann man die Cluster vielleicht noch mit freiem Auge erkennen, insofern ist das kein gutes Beispiel: Aber höherdimensionale Datenräume lassen sich nicht mehr gut visualisieren.)

Illustration: Python

8 Python-Illustration 8

```
In [ ]: # Skalieren der Daten:
        data = scale_this(data)
        # k-means-Algorithmus:
        centroids = kmeans(data, 3)[0]
```

Beispiel: k–Means mit $k = 4$



Big Data: Auch für Versicherungen interessant

Methoden aus Data Science werden heute nicht nur von Konzernen wie Google oder Facebook eingesetzt, sondern zunehmend auch von Banken, die auf Basis immer umfangreicherer Datenbestände zum Kundenverhalten versuchen, statistisch signifikante Vorhersagen für

- Ansprechbarkeit für Kredit– oder Veranlagungsangebote,
- Kreditwürdigkeit,
- Abwanderungsgefahr,
- Preissensitivität

zu machen und auf dieser Basis

- strategische (sollen wir unsere Kostenstruktur anpassen?)
- und taktische (sollen wir jetzt unseren Vertrieb auf diese Kundengruppe ansetzen?)

Geschäftsentscheidungen zu fällen.

Forschungsplattform "Data Science" an der Uni Wien

Brauchbare Vorhersagen dieser Art sind wahrscheinlich auch für Versicherungsunternehmen von Interesse: Auf diesem Gebiet ist heute vieles möglich, woran vor 20 Jahren noch nicht zu denken war.

Forschungsplattform Data Science

Wenn ich mit meinen Ausführungen Ihr Interesse am Thema "Data Science" geweckt habe, möchte ich Sie abschließend auf die unlängst neu eingerichtete Forschungsplattform "Data Science" an der Universität Wien hinweisen; siehe

`http://datascience.univie.ac.at`

Vielen Dank für Ihre Aufmerksamkeit!

Die Botschaft meines heutigen Vortrags:

- Sie können mit modernen Software–Werkzeugen sehr viele *Data Science*–Fragen selbst beantworten,
- und wenn Sie tiefergehende Fragen haben, finden Sie an der *Universität Wien* Experten aus Mathematik, Informatik und Statistik.

Die Folien zu meinem Vortrag und das IPython–Notebook, mit dem ich meine (rein illustrativen!) Beispiele ausgearbeitet habe, stehen unter

`http://www.mat.univie.ac.at/~mfulmek/avoe2018.shtml`

zur Verfügung.

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!