Дисертация на тема

Разработване на виртуална среда за пенсионни актюерски изчисления в Python

Георги Веселинов Бурнаски
Катедра Приложна Математика
Факултет по Математика и Информатика
Софийски Университет "Св. Климент Охридски"

2025-09-19

1 Увод

Актюерската математика заема все по-значимо място в съвременното общество, където управлението на финансовите рискове и дългосрочната устойчивост на социалните системи са от първостепенно значение.

В условията на застаряващо население, динамични финансови пазари и нарастващи изисквания към пенсионните системи, надеждните и прецизни актюерски изчисления се превръщат в ключов инструмент за вземане на информирани решения. Те подпомагат както държавните институции при моделиране и реформиране на пенсионните схеми, така и частните компании при разработване на застрахователни и инвестиционни продукти [dalriada2023].

Актюерските методи намират широко приложение в различни сфери: пенсионното осигуряване, животозастраховането здравното застраховане, управлението на фондове и финансовото планиране. Общото между всички тези области е необходимостта от точни прогнози, основани на статистически модели и вероятностни методи, които да оценяват бъдещи парични потоци, продължителност на живота и свързаните с тях рискове [milliman2022].

В наши дни пенсионните системи се изправят пред редица сериозни предизвикателства. Демографските промени, и по-специално увеличаването на средната продължителност на живота и намаляването на раждаемостта, водят до нарастващо съотношение между пенсионери и активно работещо население [bostonfed1999]. Това поставя под натиск публичните пенсионни фондове и създава необходимост от по-прецизни модели за оценка на бъдещите задължения. Паралелно с това колебанията на финансовите пазари и инфлационните процеси влияят върху доходността на пенсионните активи и изискват по-гъвкави подходи при управлението на риска [ncpers2023].

Съвременните технологии и програмни езици като Python предлагат нови възможности за прилагане на актюерската математика в практиката. Разработването на специализирана библиотека за пенсионни актюерски изчисления има двойна значимост: от една страна, улеснява изследователите и практиците при прилагането на сложни модели, а от друга – допринася за повишаване на прозрачността, възпроизводимостта и достъпността на тези изчисления [lifelib2024, hyperexponential2023].

Въпреки наличието на различни софтуерни решения за актюерски изчисления,

много от тях са или твърде специализирани, или не предлагат необходимата гъвкавост и интеграция с други аналитични инструменти. Python, с богатия си екосистем от библиотеки за научни изчисления и машинно обучение, се явява като идеална платформа за разработване на такава библиотека. Тя би могла да обедини теоретичните основи на актюерската математика с практическите нужди на потребителите, предоставяйки лесен за използване и разширяем инструмент [NumPy2024, pandas2024].

Разработването на библиотека за пенсионни актюерски изчисления в Python би могло да включва функции за изчисляване на настояща стойност на бъдещи парични потоци, моделиране на демографски и финансови рискове, симулации на сценарии и оптимизация на пенсионни стратегии. Освен това, интеграцията с други библиотеки за визуализация и анализ би позволила по-добро представяне и интерпретация на резултатите [Matplotlib2024, Seaborn2024].

В заключение, създаването на специализирана библиотека за пенсионни актюерски изчисления в Python представлява важна стъпка към модернизацията и усъвършенстването на актюерската практика. Тя би могла да подпомогне както академичните изследвания, така и практическите приложения, като предостави мощен и достъпен инструмент за анализ и управление на пенсионните системи в съвременния свят.

Пенсионната система в България следва модела на много развити страни и се състои от три компонента, известни като "трите стълба":

- Първи стълб: Държавно задължително пенсионно осигуряване (солидарност между поколенията). Това е държавната пенсия, която се финансира от текущите осигурителни вноски на работещите. Тя е задължителна за всички работещи.

- Втори стълб: Допълнително задължително пенсионно осигуряване (ДЗПО). Това е индивидуално натрупване на средства в избрани от самия осигурено лице пенсионни фондове. Той също е задължителен за хората, родени след 31.12.1959 г., които не са избрали да останат само в Първи стълб.

- Трети стълб: Допълнително доброволно пенсионно осигуряване (ДДПО). Това е напълно доброволно допълнително пенсионно осигуряване, при което хората сами решават да спестяват допълнително за своята пенсия.

В този труд ще се фокусираме върху изчисленията, свързани със втори и трети стълб от пенсионносигурителната система, тъй като те са поети от частните пенсионноосигурителни дружества. В България 20% от доходите на работещите се отделят за пенсионно осигуряване, като 12.8% отиват за първи стълб, а 5% - за втори стълб. Важно е да се отбележи, че тези проценти могат да варират в зависимост от законодателството и икономическите условия. Останалите 2.2% са за здравно осигуряване и трудови злополуки.[ZDZPO2004, ZDOO2000]

Условията за пенсионните изчисления са регламентирани от различни нормативни актове, като основните от тях са:

- Кодекс за социално осигуряване (КСО)
- Закон за допълнително задължително пенсионно осигуряване (ЗДЗПО)
- Закон за допълнително доброволно пенсионно осигуряване (ЗДОО)
- Наредба № Н-8 от 2004 г. за определяне на методиката за изчисляване на пенсиите
- Наредба № 3 от 2003 г. за условията и реда за изплащане на пенсиите от частните пенсионни фондове

Като изчисленията трябва да отговарят на изискванията, заложени наредба 69 от 2003 г. за определяне на методиката за изчисляване на пенсиите от частните пенсионни фондове. [ZDZPO2004, ZDOO2000, DKFNPensions, NOIOfficial]

Целта на настоящата дисертация е именно изграждането на такава библиотека, която да обедини теоретичните основи на пенсионната математика с предимствата на съвременните програмни среди. Чрез нея се цели създаването на гъвкав инструмент, който да подкрепя както научната работа, така и практическите решения в областта на пенсионното осигуряване.

2 Математика

2.1 Вероятност за смърт

Вероятността за смърт q_x е вероятността човек на възраст x да умре преди да достигне възраст $x+1$. Тази вероятност се взима от статистическата статистика на държавната осигурителна институция в случая Националният осигурителен институт(НОИ)

2.2 Вероятност за преживяване

Вероятността за преживяване $p_x = 1-q_x$ е вероятността човек на възраст x да оцелее до следващата година, т.е. до възраст $x+1$.

2.3 Среден брой живи хора

Средният брой живи хора на възраст x се обозначава с l_x и представлява броя на хората, които са живи на тази възраст в дадена популация. Тази стойност се използва за изчисляване на вероятността за смърт и други актюерски изчисления. Изчислява се по формулата:

$$l_x = l_0 \cdot \prod_{i=0}^{x-1} p_i$$

където l_0 е началният брой живи хора (на възраст 0), а p_i е вероятността за преживяване на възраст i и l_x е броят на живите хора на възраст x.

2.4 Дисконтиращ фактор

Дисконтиращият фактор v се използва за изчисляване на настоящата стойност на бъдещи парични потоци. Той се изчислява по формулата:

$$v = \frac{1}{1+i}$$

където i е годишният лихвен процент зададен от пенсионноосигурителната компания. В нашият случай ще използваме техническия лихвен процент на пенсионна компания "Съгласие" който е 0.15% [DKFNPensions]

2.5 Настояща стойност

Настоящата стойност (D_x) на бъдещ паричен поток се изчислява чрез дисконтиране на този поток към настоящия момент. Формулата за изчисляване

на настоящата стойност е:

$$D_x = l_x \cdot v^x$$

2.6 Кумолативна настояща стойност

Кумулативната настояща стойност (N_x) представлява сумата от настоящите стойности на всички бъдещи плащания за период от n години. Тя се изчислява по формулата:

$$N_x = \sum_{i=x}^{max} D_x$$

2.7 Актюерски фактори и изчисления на пенсии

Актюерските фактори се използват за изчисляване на различни видове пенсии и други финансови продукти. Те включват:

- **Проста пожизнена пенсия (фактор k_1)** - изчислява се като:

$$k_1 = 12 \cdot \left(\frac{N_x}{D_x - \frac{11}{24}} \right)$$

  където N_x е кумулативната настояща стойност на бъдещите плащания, а D_x е настоящата стойност на бъдещия паричен поток. Това представлява фактор, който се използва за изчисляване на месечната сума на Пожизнена пенсия. Която се изчислява като:

$$P = \frac{S}{k_1}$$

  където P е месечната сума на пенсията, а S е натрупаната сума в пенсионния фонд към момента на пенсиониране.

- **Пожизнена пенсия с период на гарантирано изплащане (фактор $_2$)** - изчислява се като:

$$k_2 = 12 \cdot \left(\frac{N_{x+d}}{D_x} - \frac{11}{24} \cdot \frac{D_{x+d}}{D_x} \right) + \frac{1 - v^n}{1 - \sqrt[12]{v}}$$

  където N_x е кумулативната настояща стойност на бъдещите плащания, N_{x+n-1} е кумулативната настояща стойност на бъдещите плащания след изтичане на гарантирания период от n години, а D_x е настоящата стойност на бъдещия паричен поток. Това представлява фактор,

който се използва за изчисляване на месечната сума на Пожизнена пенсия с гарантиран период. Която се изчислява като:

$$P = \frac{S}{k_2}$$

- **Допълнителна пожизнена пенсия полагаща се след период на разсрочено изплащане (фактор k_3)** - изчислява се като:

$$k_3 = 12 \cdot \left(\frac{N_{x+d}}{D_x} - \frac{11}{24} \cdot \frac{D_{x+d}}{D_x} \right)$$

където N_x е кумулативната настояща стойност на бъдещите плащания, N_{x+d} е кумулативната настояща стойност на бъдещите плащания след изтичане на гарантирания период от n години, а D_x е настоящата стойност на бъдещия паричен поток. Това представлява фактор, който се използва за изчисляване на месечната сума на Допълнителна пожизнена пенсия с разсрочено изплащане. Която се изчислява като:

$$P = \frac{S - \sum_{i=1}^{m} T_i}{k_3}$$

където P е месечната сума на пенсията, S е натрупаната сума в пенсионния фонд към момента на пенсиониране, а T_m са месечните вноски по време на периода на разсрочено изплащане. Те се изчисляват като:

$$T_i = H_i \cdot v^{(\frac{b_i}{12})}$$

където H_i е размера на месечното плащане повреме на разсроченият петиод, а b_i е броят на месеците от началото на разсроченото изплащане до момента на съответното плащане.

3 Устройство

Имплементирани са следните класове:

- **Person** - представлява човек с атрибути като дата на раждане, пол и други.
- **PensionFund** - представлява пенсионен фонд с атрибути като име, тип (задължителен или доброволен), и други.
- **Company** - представлява компания, която предлага пенсионни продукти.
- **Calculator** - съдържа методи за изчисляване на пенсията въз основа на натрупаните вноски, лихвени проценти и други фактори.
- **Finances** - съдържа методи за финансови изчисления, като изчисляване на настояща стойност, бъдеща стойност и други.
- **main** - основен скрипт, който демонстрира използването на библиотеката.

И се използват следните примерни данни:

- **NSI_mortality_table.csv** - таблица на смъртността от Националната осигурителна институция (НОИ).
- **Saglasie_fund_actives.csv** - данни за активите на пенсионен фонд "Съгласие".

Също така са използвани следните външни библиотеки:

- **NumPy** - за числени изчисления и работа с масиви [NumPy2024].
- **Pandas** - за обработка и анализ на данни [pandas2024].
- **Matplotlib** - за визуализация на данни [Matplotlib2024].
- **Seaborn** - за статистическа визуализация на данни [Seaborn2024].
- **Datetime** - за работа с дати и времена.
- **Math** - за математически функции и операции.
- **Random** - за генериране на случайни числа.
- **CSV** - за работа с CSV файлове.

Нека зарзгледаме основните класове и техните методи по-подробно:

3.1 Company

Клас, който създава обекти - компания със съответните параметри. Те включват:

- Име на компанията
- Технически лихвен процент
- Риск при разсрочено изплащане
- База данни със стойностите на активите на компанията във различните фондове, като Универсален, Доброволен и Професионален Пенсионен фонд, подредени по дата.
- Хора - масив свъв който се добавят участниците във пенсионният фонд (обекти от класа Person)
- Общо хора (l_0) за актюерски изчисления, константа която по принцип се задава като равна на 100 000 за актюерските изчисления в България.

3.2 Person

Клас, който създава обекти хора, които ще бъдат участници във фонда. Параметрите включват:

- Име
- ЕГН
- Пол
- Възраст
- Доход
- Спестявания
- Осигурителен стаж
- Пенсионна възраст
- Предполагаеми спестявания при пенсиониране

Използва редица методи за преобразуването на ЕГН в дата на раждане и изчисляване на възрастта, както и намиране на пола на лицето. Както и оставащото време за работа до пенсиониране, което служи за намирането на предполагаемите спестявания при пенсиониране.

3.3 Calculator

Основния клас във работната среда. Съдържа методи за изчисляване на пенсията, базирани на входните данни от класовете Person и Company. Съдържа класове за изчисляване на актюерските константи като: q_x, p_x, l_x, v_x, D_x и N_x по формулите описани в раздел Математика. Съдържа и клас от методи за извличане на информацията от базата данни от актюерски константи на НОИ. Освен това има и клас от функции - Pension с три подкласа изчисляващи факторите за трите пенсии и месечните плащания за всяка една от тях при всякакви условия.

3.4 Finances

Финансов клас, който управлява всички финансови операции и изчисления. Той включва методи за:

- Извличане на информация от базата данни на компанията за стойността на активи във различните фондове.
- Пресмятане на плаващо средно за стойността на активите, което служи за анализ на тенденциите във времето и прогнозиране на бъдещи стойности.
- Изчисляване на средното и стандартното отклонение на движенията на активите. (Предполага се че следват аналогия на бял шум (White Noise))
- Симулация на бъдещи стойности на активите чрез Монте Карло метод.
- Изчисляване на бъдещата стойност на спестяванията при различни сценарии.
- Изобразяване на данните чрез графики.

3.5 main

Това е основният скрипт, който демонстрира използването на библиотеката. Той включва:

- Създаване на обекти от класовете Company и Person с примерни данни.

- Извикване на методите от класовете Calculator и Finances за изчисляване на пенсията и анализ на финансовите данни.
- Визуализация на резултатите чрез графики.

4 Резултати

5 Заключение

References

[bostonfed1999] Munnell, A. H. (1999). *Demographic Changes and Funding for Pension Plans*. Boston Fed. Retrieved from: `https://www.bostonfed.org/-/media/Documents/conference/16/conf16b.pdf` Accessed: 2025-09-17.

[dalriada2023] Dalriada Trustees. (2023). *The Importance of Mathematics in Pensions*. Retrieved from: `https://www.dalriadatrustees.co.uk/the-importance-of-mathematics-in-pensions/` Accessed: 2025-09-17.

[milliman2022] Milliman. (2022). *Using Python as an Actuarial Modelling Platform*. Retrieved from: `https://www.milliman.com/en/insight/using-python-actuarial-modelling-platform` Accessed: 2025-09-17.

[ncpers2023] National Conference on Public Employee Retirement Systems (NCPERS). (2023). *The Impact of Demographic Shifts on Public Pensions (And What They Can Do About It)*. Retrieved from: `https://www.ncpers.org/blog_home.asp?display=377` Accessed: 2025-09-17.

[lifelib2024] Binette, O. (2024). *lifelib: Actuarial Models in Python*. Retrieved from: `https://lifelib.io/` Accessed: 2025-09-17.

[hyperexponential2023] Hyperexponential. (2023). *Python for Insurers and Actuaries*. Retrieved from: `https://www.hyperexponential.com/blog/python-for-insurance/` Accessed: 2025-09-17.

[ZDZPO2004] Народно събрание на Република България. (2004). *Закон за допълнителното задължително пенсионно осигуряване*. Доставен от: `https://www.parliament.bg/bg/laws` Последно посетен на: 18.09.2023 г.

[ZDOO2000] Народно събрание на Република България. (2000). *Закон за държавното обществено осигуряване*. Доставен от: `https://www.parliament.bg/bg/laws` Последно посетен на: 18.09.2023 г.

[DKFNPensions] Държавна комисия по финансов надзор (ДКФН). (2023). *Раздел „Пенсионни фондове“*. Доставен от: `https://www.dkn.bg` Последно посетен на: 18.09.2023 г.

[NOIOfficial] Национална осигурителна институция (НОИ). (2023). *Официален уебсайт*. Доставен от: `https://www.noi.bg` Последно посетен на: 18.09.2023 г.

[NumPy2024] Harris, C. R., Millman, K. J., van der Walt, S. J. et al. (2024). *Array programming with NumPy*. Nature, 585(7825), 357–362. Retrieved from: `https://www.numpy.org` Accessed: 2025-09-17.

[pandas2024] The pandas development team. (2024). *pandas-dev/pandas: Pandas*. Zenodo. Retrieved from: `https://pandas.pydata.org` Accessed: 2025-09-17.

[Matplotlib2024] Hunter, J. D., and the Matplotlib development team. (2024). *Matplotlib: Visualization with Python*. Retrieved from: `https://matplotlib.org` Accessed: 2025-09-17.

[Seaborn2024] Waskom, M. L., and the seaborn development team. (2024). *Seaborn: statistical data visualization*. Journal of Open Source Software, 6(60), 3021. Retrieved from: `https://seaborn.pydata.org` Accessed: 2025-09-17.

6 Приложения

```python
from Company import Company
from Person import Person
from Calculator import Simple_pension, Guaranteed_pension,
    Instalment_pension

my_company = Company(name="Sagalsie", interest=0.075,
    risk_level=0.015, fund_data="Code/Data/
    Saglasie_fund_actives.csv")

me = Person(name="Georgi", egn=9606067062, income=2000,
    savings=10000)

my_company.add_person(me)

georgi = next(person for person in my_company.people if
    person.name == 'Georgi')

print(Simple_pension(
    q_csv="Code/Data/NSI_q_values.csv",
    age=65,
    saldo=georgi.funds_at_retirement,
    company=my_company
).pension)

print(
    Guaranteed_pension(
        q_csv="Code/Data/NSI_q_values.csv",
        age=65,
        saldo=georgi.funds_at_retirement,
        company=my_company,
        guaranteed_period_years=10,
    ).pension
)

print(
    Instalment_pension(
        q_csv="Code/Data/NSI_q_values.csv",
        age=65,
        saldo=georgi.funds_at_retirement,
        instalment_ammount=400,
        instalment_period_months=15*12,
        company=my_company,
    ).pension
)
```

Listing 1: main.py

```python
from dataclasses import dataclass
from Person import Person

@dataclass
class Company:
    name: str
    interest: float  # Annual interest rate as a decimal (e.g., 0.05 for 5%)
    risk_level: float  # Risk level as a decimal
    fund_data: str  # Path to fund data CSV file
    people: list = None  # List of Person objects associated with the company

    total_people: int = 100000  # Total number of people for actuarial calculations

    def add_person(self, person):
        if self.people is None:
            self.people = []
        self.people.append(person)
```

Listing 2: company.py

```python
from dataclasses import dataclass,field
from datetime import datetime, timedelta

@dataclass
class Person:
    name: str
    egn: int  # Unique identifier
    income: float  # Monthly income
    savings: float  # Current savings

    sex: str  =field(init=False) # 'M' or 'F'
    age: datetime = field(init=False)  # Age as a datetime object
    years: int = field(init=False)  # Age in years
    months: int = field(init=False)  # Additional months beyond full years

    retirement_age: timedelta = timedelta(days=65*365.25)  # Default retirement age
    months_to_retirement: int = field(init=False)  # Months until retirement

    funds_at_retirement: float = field(init=False)  # Estimated funds at retirement

    today = datetime.today()

```

```
23      def calculate_age(self):
24
25          birth_year = str(self.egn)[:2]
26          if int(birth_year) <= int(str(self.today.year)[2:]):
27              birth_year = 2000 + int(birth_year)
28          else:
29              birth_year = 1900 + int(birth_year)
30
31          birth_month = str(self.egn)[2:4]
32          birth_month = int(birth_month)
33          if birth_month > 40:
34              birth_month -= 40
35          birth_day = str(self.egn)[4:6]
36          birth_day = int(birth_day)
37          birth_date = datetime(birth_year, birth_month,
    birth_day)
38
39          if self.today.month < birth_month or (self.today.
    month == birth_month and self.today.day < birth_day):
40              years = self.today.year - birth_date.year - 1
41          else:
42              years = self.today.year - birth_date.year
43
44          if self.today.day < birth_day:
45              months = (self.today.month - birth_month - 1) %
    12
46          else:
47              months = (self.today.month - birth_month) % 12
48
49          age = self.today - birth_date
50
51          return age, years, months
52
53
54      def determine_sex(self):
55          if int(str(self.egn)[-2]) % 2 == 0:
56              return 'M'
57          else:
58              return 'F'
59
60      def calculate_months_to_retirement(self):
61          if self.age>=self.retirement_age:
62              return 0
63          else:
64              return int((self.retirement_age - self.age).days
    // 30.44)  # Approximate month length
65
66      def calculate_funds_at_retirement(self):
67          if self.months_to_retirement <= 0:
```

```
68              return self.savings
69          else:
70              return self.savings + self.income * self.
    months_to_retirement * 0.05  # Simplified calculation
71
72      def __post_init__(self):
73          self.age, self.years, self.months = self.
    calculate_age()
74          self.sex = self.determine_sex()
75          self.months_to_retirement = self.
    calculate_months_to_retirement()
76          self.funds_at_retirement = self.
    calculate_funds_at_retirement()
```

Listing 3: person.py

```
1  from dataclasses import dataclass, field
2  from math import prod, floor
3  import csv
4  from scipy.stats import norm
5
6  ## Creates a new age object with all the actuarial factors
       calculated
7  @dataclass
8  class new_Age_Data:
9      company: object
10     q: list[float]
11
12     age: list[int] = field(default_factory=lambda: [i for i
       in range(0, 101)])
13
14     p: list[float] = field(init=False)
15
16     l: list[int] = field(init=False)
17
18     v: list[float] = field(init=False)
19
20     d: list[float] = field(init=False)
21     n: list[float] = field(init=False)
22
23     ## Calculates l, v, d, n based on the age and q values
24
25     def calculate_p(self):
26         p = []
27         for value in self.q:
28             p.append(1 - value)
29         return p
30
31     def calculate_l(self):
32         l = []
```

```
33          for age in self.age:
34              if age == 0:
35                  l.append(self.company.total_people)
36              else:
37                  l.append(round(self.company.total_people *
    prod(self.p[:age])))
38          return l
39
40      def calculate_v(self):
41          v = []
42          for age in self.age:
43              v.append((1 / (1 + self.company.interest))**age)
44          return v
45
46      def calculate_d(self):
47          d = []
48          for age in self.age:
49              d.append(self.l[age] * self.v[age])
50          return d
51
52      def calculate_n(self):
53          n = []
54          for age in self.age:
55              n.append(sum(self.d[age :]))
56          return n
57
58      def __post_init__(self):
59          self.p = self.calculate_p()
60          self.l = self.calculate_l()
61          self.v = self.calculate_v()
62          self.d = self.calculate_d()
63          self.n = self.calculate_n()
64
65
66  #
    ------------------------------------------------------------------------

67  ## Analyzes the q values from NSI and finds the mean and
    standard diviation of age of death
68  ## Returns the q and p lists for further calculations
69  @dataclass
70  class Analyze_q_list:
71
72      csv_file: csv
73      company: object
74      q_list: list = field(init=False)
75      p_list: list = field(init=False)
76
77      l_difference: list = field(init=False)
```

```
78
79      age_of_death: list = field(init=False)
80      smoothing_factor: int = 0
81
82      mean: float = field(init=False)
83      standard_diviation: float = field(init=False)
84
85
86      ## Chains the data from NSI to a list
87      def convert_csv_to_list(self):
88          q_list = []
89          p_list = []
90          with open(self.csv_file, "r") as q_csv:
91              example_q_list = csv.reader(q_csv)
92              for row in example_q_list:
93                  q_list.append(float(row[0]))
94                  p_list.append(1 - float(row[0]))
95          return [q_list, p_list]
96
97      ## Model Deaths at Age on average, given the data
98      def calculate_l_list(self):
99          age = 0
100         l_list = []
101         l_difference = []
102         p_product = 1
103         while age < len(self.p_list):
104             l_list.append(round(self.company.total_people *
    p_product))
105             p_product *= self.p_list[age]
106             age += 1
107         i = 1
108         while i < len(l_list):
109             l_difference.append(l_list[i - 1] - l_list[i])
110             i += 1
111         return l_difference
112
113     def find_parameters(self):
114         ages = []
115         age_of_death = []
116         if self.smoothing_factor != 0:
117             for i in range(len(self.l_difference) - self.
    smoothing_factor + 1):
118                 ages.append(
119                     sum(self.l_difference[i : i + self.
    smoothing_factor])
120                     / self.smoothing_factor
121                 )
122         else:
123             ages = self.l_difference
```

```
        j = 0
        for l in ages:
            i = 0
            while i < l:
                age_of_death.append(j)
                i += 1
            j += 1

        return age_of_death

        # https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.fit.html#scipy.stats.rv_continuous.fit
        # https://www.probabilitycourse.com/chapter8/8_2_3_max_likelihood_estimation.php

    def __post_init__(self):
        self.q_list = self.convert_csv_to_list()[0]
        self.p_list = self.convert_csv_to_list()[1]
        self.l_difference = self.calculate_l_list()
        self.age_of_death = self.find_parameters()
        self.mean, self.standard_diviation = norm.fit(self.age_of_death)


#-------------------------------------------------------------------------------

### Base Pension class
@dataclass
class Pension:
    q_csv: str
    age: int
    saldo: float
    company: object
    data: list[object] = field(init=False)


    def create_age_data(self):
        return new_Age_Data(q=Analyze_q_list(csv_file=self.q_csv, company=self.company).q_list, company=self.company)

    def __post_init__(self):
        self.data = self.create_age_data()


## Simple Pension Calculator
@dataclass
class Simple_pension(Pension):
```

```
166     k: float = field(init=False)
167     pension: float = field(init=False)
168 
169     def get_k(self):
170         k = 12 * ((self.data.n[self.age] / self.data.d[self.
    age]) - (11 / 24))
171         return k
172 
173     def get_pension(self):
174         return round(self.saldo / self.k, 2)
175 
176     def __post_init__(self):
177         super().__post_init__()
178         self.k = self.get_k()
179         self.pension = self.get_pension()
180 
181 
182 ## Guaranteed Pension Calculator
183 @dataclass
184 class Guaranteed_pension(Pension):
185     guaranteed_period_years: int
186     k: float = field(init=False)
187     pension: float = field(init=False)
188 
189     def get_k(self):
190         v=self.data.v[1]
191         k = 12 * ((self.data.n[self.age + self.
    guaranteed_period_years]/ self.data.d[self.age])- ((11 /
    24)* (self.data.d[self.age +self.guaranteed_period_years]/
     self.data.d[self.age]))) + ((1-v**self.
    guaranteed_period_years)/(1-v**(1/12)))
192         return k
193 
194     def get_pension(self):
195         return round(self.saldo / self.k, 2)
196 
197     def __post_init__(self):
198         super().__post_init__()
199         self.k = self.get_k()
200         self.pension = self.get_pension()
201 
202 
203 ## Instalment Pension Calculator
204 @dataclass
205 class Instalment_pension(Pension):
206     instalment_ammount: float
207     instalment_period_months: int
208     k: float = field(init=False)
209     pension: float = field(init=False)
```

```
210
211     def get_k(self):
212         k = 12 * (
213             self.data.n[floor((12*self.age + self.
    instalment_period_months) / 12)]
214             / self.data.d[self.age]
215             - (11 / 24)
216             * self.data.d[floor((12*self.age + self.
    instalment_period_months) / 12)]
217             / self.data.d[self.age]
218         )
219         return k
220
221     def find_saldo_after_instalments(self):
222         summ=0
223         h=self.instalment_ammount
224         v=self.data.v[1]
225         i=0
226         while i < self.instalment_period_months:
227             summ+=h*v**(i/12)
228             i+=1
229         return self.saldo - summ
230
231     def get_pension(self):
232         return round(
233             ((self.find_saldo_after_instalments()*(1-self.
    company.risk_level))
234             / self.k),
235             2)
236
237     def __post_init__(self):
238         super().__post_init__()
239         self.k = self.get_k()
240         self.pension = self.get_pension()
```

Listing 4: calculator.py

```
1 from dataclasses import dataclass, field
2 import csv
3 import matplotlib.pyplot as plt
4 import random
5
6 @dataclass
7 class Finances:
8     csv_path: str = "Code/Data/Saglasie_fund_actives.csv" #
    Path to the CSV file containing fund actives data
9
10     data: list[list[str],list[float],list[float],list[float]]
     = field(init=False)
11     whindow_size: int = 30  # For moving average
```

```
    days_to_predict: int = 365  # Number of days to predict into the future


    def convert_csv_to_list(self):
        data = [[],[],[],[]]
        with open(self.csv_path, "r") as csv_file:
            data_csv = csv.reader(csv_file)
            for row in data_csv:
                i=0
                while i < len(row):
                    data[i].append(row[i])
                    i+=1
        return data

    def moving_average(self, values, window=whindow_size):
        if len(values) < window:
            return [None] * len(values)
        ma = []
        for i in range(len(values)):
            if i < window - 1:
                ma.append(None)
            else:
                ma.append(sum(values[i-window+1:i+1]) / window)
        return ma

    def average_difference_per_date(self, column_index: int):
        values = [float(x) for x in self.data[column_index][1:]]  # Skip header
        if len(values) < 2:
            return 0, 0
        differences = [values[i] - values[i-1] for i in range(1, len(values))]
        positive_diffs = [diff for diff in differences if diff > 0]
        negative_diffs = [diff for diff in differences if diff < 0]
        avg_positive = sum(positive_diffs) / len(positive_diffs) if positive_diffs else 0
        avg_negative = sum(negative_diffs) / len(negative_diffs) if negative_diffs else 0
        return avg_positive, avg_negative

    def mean_and_std_of_differences(self, column_index: int):
     # column_index: 1 for DPF, 2 for PPF, 3 for UPF
        values = [float(x) for x in self.data[column_index][1:]]  # Skip header
        if len(values) < 2:
```

```
51              return 0, 0
52          differences = [values[i] - values[i-1] for i in range
    (1, len(values))]
53          mean_diff = sum(differences) / len(differences)
54          std_diff = (sum((d - mean_diff) ** 2 for d in
    differences) / len(differences)) ** 0.5 if len(differences
    ) > 1 else 0.0
55          return mean_diff, std_diff
56
57      def predict_next_n(self, column_index: int):
58          values = [float(x) for x in self.data[column_index
    ][1:]]  # Skip header
59          if len(values) < 2:
60              return []
61          mean_diff, std_diff = self.
    mean_and_std_of_differences(column_index)
62          predictions = []
63          last_value = values[-1]
64          for _ in range(self.days_to_predict):
65              next_diff = random.gauss(mean_diff, std_diff)
66              next_value = last_value + next_diff
67              predictions.append(next_value)
68              last_value = next_value  # Use the new value for
    the next prediction
69          return predictions
70
71      def plot_fund_actives(self):
72          dates = self.data[0][1:]  # Skip header
73          dpf = [float(x) for x in self.data[1][1:]]
74          ppf = [float(x) for x in self.data[2][1:]]
75          upf = [float(x) for x in self.data[3][1:]]
76
77          dpf_ma = self.moving_average(dpf, 30)
78          ppf_ma = self.moving_average(ppf, 30)
79          upf_ma = self.moving_average(upf, 30)
80
81          plt.figure(figsize=(12, 6))
82          plt.plot(dates, dpf, label='Fund Actives DPF')
83          plt.plot(dates, ppf, label='Fund Actives PPF')
84          plt.plot(dates, upf, label='Fund Actives UPF')
85          plt.plot(dates, dpf_ma, color='red', linestyle='--',
    label='DPF 30-day MA')
86          plt.plot(dates, ppf_ma, color='red', linestyle=':',
    label='PPF 30-day MA')
87          plt.plot(dates, upf_ma, color='red', linestyle='-.',
    label='UPF 30-day MA')
88          plt.xlabel('Date')
89          plt.ylabel('Fund Actives')
90          plt.title('Fund Actives Over Time')
```

```
91          plt.legend()
92          plt.tight_layout()
93          plt.show()
94
95
96      def __post_init__(self):
97          self.data = self.convert_csv_to_list()
98
99  # Example usage:
100 fin = Finances()
101 predicted_dpf = fin.predict_next_n(1)
102 print("Predicted next 365 DPF values:", predicted_dpf[:5], "
        ...")  # Print first 5 as a sample
```

Listing 5: finances.py