Fundamentals of Long-Term Actuarial Modeling

ExaminationAid® Series

§1: Introduction

§1.1: Abstract

This research proposal, titled "Fundamentals of Long-Term Actuarial Modeling," aims to integrate traditional actuarial practices with modern data science techniques through the application of object-oriented programming (OOP) principles in R. Inspired by insights from the Long-Term Actuarial Mathematics (LTAM) course and extensive professional experience in data science, this project presents a practical application of OOP in actuarial science. The proposal will illustrate how leveraging OOP concepts such as classes, objects, and inheritance can transform actuarial modeling by enhancing the modularity, reusability, and scalability of models.

Focusing on practical implementations in R, the proposal will use actuarial concepts from FAM-L to explain key programming principles like functions, algorithms, and plotting. The research demonstrates how to develop sophisticated data products that efficiently handle complex data sets, providing an alternative to traditional methods such as Excel tables. This approach serves as a foundation for developing innovative educational and professional tools, contributing to the Actuarial Science CRAN community, and leading to the eventual creation of comprehensive exam aid dashboards tailor made for the actuarial science industry and participants.

The findings and methodologies will be detailed in the succeeding sections of the proposal, showcasing real-world examples, case studies, and the practical application of OOP techniques in actuarial projects. This research ultimately aims to enhance actuarial workflows and foster greater collaboration within actuarial teams.

§1.2: Content Overview

§1.2.1 FAM-L Content

This proposal delves into the essential relationships between survival models, life tables, insurance benefits, annuities, and profitability, following a progression that mirrors the structure of "Actuarial Mathematics for Life Contingent Risks."

- **Chapter 2**: Survival Models
  - Survival models form the foundational basis for actuarial science, describing the distribution of time until events such as death. These models employ functions like the survival function and hazard function, which are crucial for understanding mortality trends. This understanding is essential for predicting lifespans and informing the design of life-contingent products.
- **Chapter 3**: Life Tables

  - Life tables, derived from survival models, provide empirical data on mortality rates and survival probabilities at various ages. They are indispensable tools for actuaries, offering critical insights that inform the creation and pricing of insurance policies and annuities by illustrating the likelihood of survival across different age groups.

- **Chapter 4**: Insurance Benefits

  - Insurance benefits are calculated by integrating survival probabilities to ensure adequate coverage. These benefits are designed based on the empirical data from life tables, ensuring that payouts are financially viable and meet the needs of policyholders. Accurate determination of present values for future payouts is crucial for the sustainability of insurance products.

- **Chapter 5**: Annuities

  - Annuities provide regular payments over the lifetime of the annuitant and rely heavily on survival models and life tables for valuation. Different types of annuities (immediate, deferred, fixed, and variable) each require precise calculation of expected present values to ensure they provide sufficient income streams while being financially sustainable.

- **Chapter 6**: Profitability (Policy Values)

  - Profitability in insurance products is assessed through policy values, which are the present values of future cash flows, including premiums and benefits. Actuarial models use survival probabilities, discount rates, and expense assumptions to determine these values. A profitable policy ensures that the present value of premiums surpasses the present value of benefits and expenses, thus ensuring the insurer's financial health.

The progression from survival models to profitability illustrates the integrated nature of actuarial science. Each chapter builds on the previous, from understanding mortality trends and using this knowledge to design viable insurance products, to ultimately ensuring these products are profitable. This comprehensive approach ensures the development of robust insurance and financial products that meet individual needs and support the financial stability of insurance companies.

§1.2.2 R-Programming Content

To achieve the goals outlined in the abstract and the overview, this proposal employs a variety of R programming techniques, catering to users of all skill levels—from novices to experts. These techniques demonstrate how R can be effectively utilized in professional environments to handle a range of actuarial tasks and situations.

§1.2.2.1 Novice and Intermediate Users

- **Plotting with ggplot2**: This powerful visualization package allows users to create complex and informative plots. Novices can start with basic plots, while intermediates can explore advanced customization and data visualization techniques, making it easier to communicate insights from actuarial data effectively.

- **Data-Frame Manipulation with dplyr and tidyverse**: These packages streamline data manipulation, enabling users to filter, arrange, and summarize data efficiently. Novices will learn the basics of data manipulation, and intermediates can leverage these tools for more sophisticated data wrangling tasks, essential for preparing actuarial data sets for analysis.

- **Algorithms with if-else Statements**: Basic conditional logic using if-else statements allows users to implement simple decision-making processes in their code. This foundational programming concept helps in automating repetitive tasks and is crucial for actuarial calculations and data analysis.

§1.2.2.2 Intermediate and Advanced Users

- **Algorithms with for-loop Statements**: For-loop statements enable the execution of repetitive tasks over a sequence of elements. Intermediate users will learn to iterate over data sets, performing complex calculations, and advanced users can optimize these loops for efficiency, crucial for handling large actuarial data sets.
- **Functional Programming with sapply**: The sapply function applies a function to each element of a list or vector, simplifying code and making it more readable. This concept introduces functional programming principles, which are beneficial for performing repetitive actuarial computations and data transformations.
- **Object-Oriented Programming with S4 Class Objects**: S4 class objects represent a robust OOP system in R, allowing for the creation of complex data structures. Advanced users will learn to design and implement reusable and scalable actuarial models, enhancing the modularity and maintainability of their code.

§1.2.2.3 Application in Professional Environments

- **Plotting**: Actuaries can create detailed visualizations of survival models and life tables, aiding in the presentation and interpretation of mortality trends and insurance benefits.
- **Data-Frame Manipulation**: Efficient data manipulation is crucial for preparing data for actuarial analysis, including calculating annuity values and policy profitability.
- **Algorithms**: Conditional logic and loop constructs automate the process of evaluating insurance policies and annuities, ensuring accuracy and efficiency.
- **Functional Programming**: Simplifies complex actuarial computations, making code easier to maintain and extend.
- **Object-Oriented Programming**: Facilitates the development of sophisticated actuarial models that are modular and reusable, supporting long-term maintenance and scalability.

By integrating these R techniques, this proposal not only achieves the goals set out in the abstract and overview but also equips users with the necessary skills to apply these concepts in diverse professional settings, enhancing their ability to develop robust and efficient actuarial solutions.

§1.2.3: Mortality parameters

Unless explicitly stated otherwise, the mortality parameters for both the ultimate and select periods used in this proposal follow the definitions provided in Appendix D of the textbook "Actuarial Mathematics for Life Contingent Risks" (Third Edition) by Dickson, Hardy, and Waters.

- **Law**:
  - *Name*: Makeham
  - *Formula*:

$$\mu_x = A + Bc^x$$

where

$$\begin{aligned} \mu_x &= \text{Standard Ultimate Survival Model}, \\ A &= \text{constant representing age-independent mortality}, \\ B &= \text{constant representing age-dependent mortality factors}, \\ c &= \text{constant representing age-dependent mortality factors}. \end{aligned}$$

- **Parameters**:
  - A: 0.00022
  - B: 0.0000027
  - c: 1.124
- **Select Period**:
  - *Duration*: 2 years ($0 \leq s \leq 2$)
  - *Formula*:

$$\mu_{[x]+s} = 0.9^{2-s}\mu_{x+s}$$

where

$$\mu_{[x]+s} = \text{Standard Select Survival Model},$$
$$\mu_{x+s} = \text{force of mortality at age } (x+s).$$

- **Interest Rate**:
  - i: 5%

§2: Body

The body of this proposal examines the key relationships among survival models, life tables, insurance benefits, annuities, and profitability, structured similarly to "Actuarial Mathematics for Life Contingent Risks." It begins with survival models, which are fundamental for predicting mortality trends and designing life-contingent products. Next, life tables provide empirical data on mortality rates, essential for creating and pricing insurance policies and annuities. Insurance benefits are then calculated using survival probabilities to ensure financial viability. Annuities are valued based on these models and tables to provide sustainable income streams. Finally, the profitability of insurance products is assessed through policy values, ensuring that premiums exceed benefits and expenses. Each section builds on the previous one, illustrating the integrated nature of actuarial science and its role in developing robust financial products.

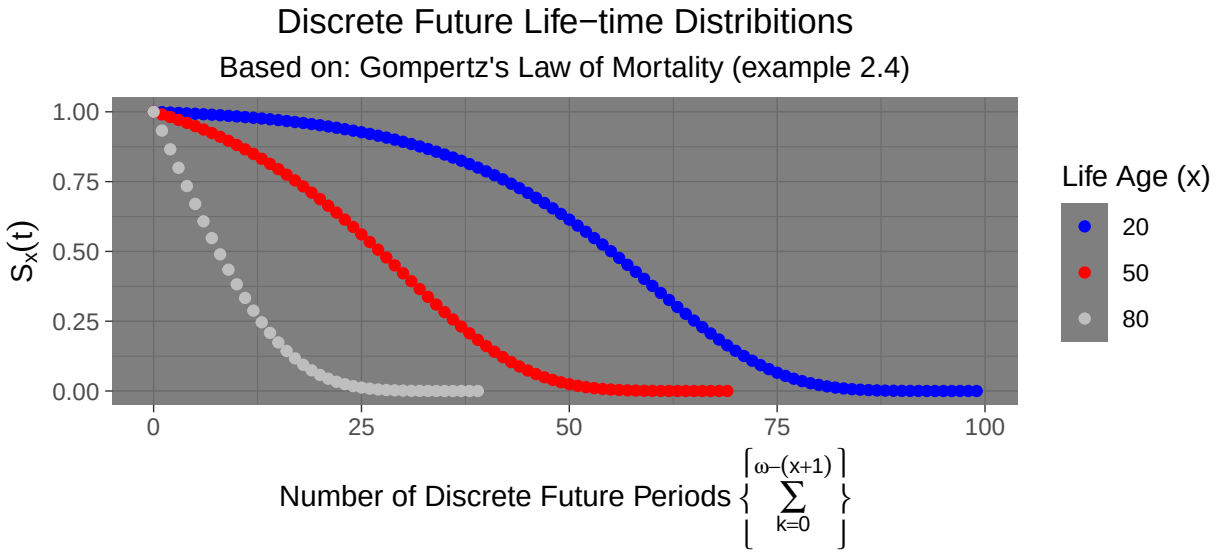
§2.1: Survival Models

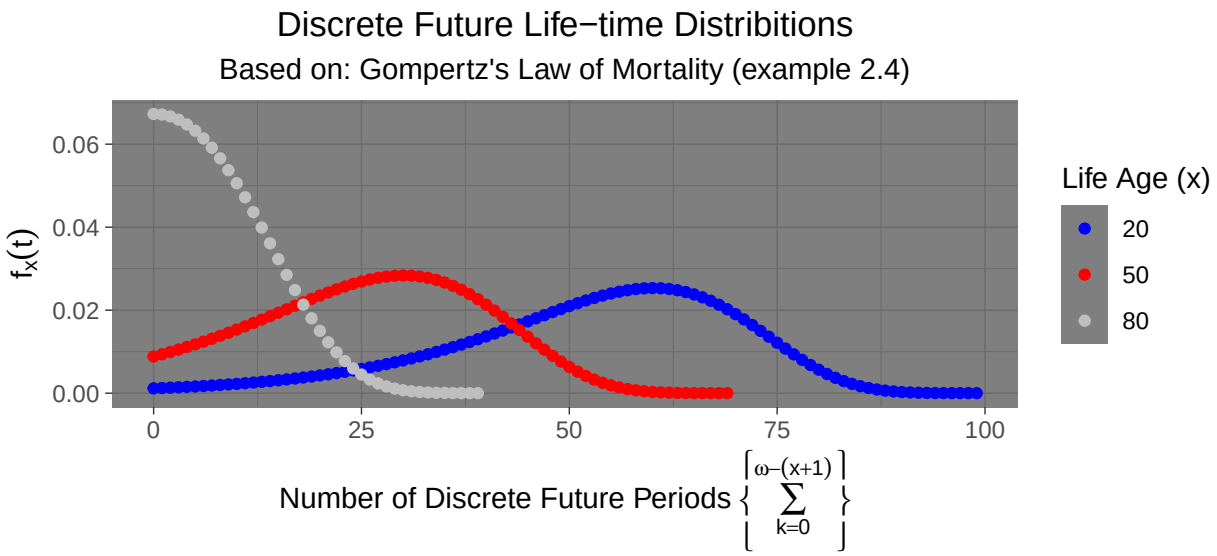
Survival models are the cornerstone of actuarial science, describing the distribution of time until events such as death. These models use functions like the survival function and hazard function, which are critical for understanding mortality trends. This knowledge is essential for predicting lifespans and designing life-contingent products.

§2.1.1: Gompertz Survival Model

The R code snippet for the Gompertz survival model demonstrates several key R concepts. Using ggplot2, it creates complex and informative plots to visualize survival probabilities and future lifetime probability density functions for ages 20, 50, and 80. The dplyr and tidyverse packages streamline data manipulation, allowing for efficient filtering, arranging, and summarizing of data, which is essential for actuarial analysis. Basic if-else statements implement conditional logic for simple decision-making processes. For-loop statements execute repetitive tasks over data sequences, with intermediate users learning iteration for complex calculations and advanced users optimizing for efficiency. The sapply function introduces functional programming principles, applying functions to list or vector elements to simplify code. Additionally, S4 class objects enable the creation of complex data structures, allowing advanced users to design reusable and scalable actuarial models, enhancing code modularity and maintainability.

Code: `example 2.4: Gompertz survival model appendix`

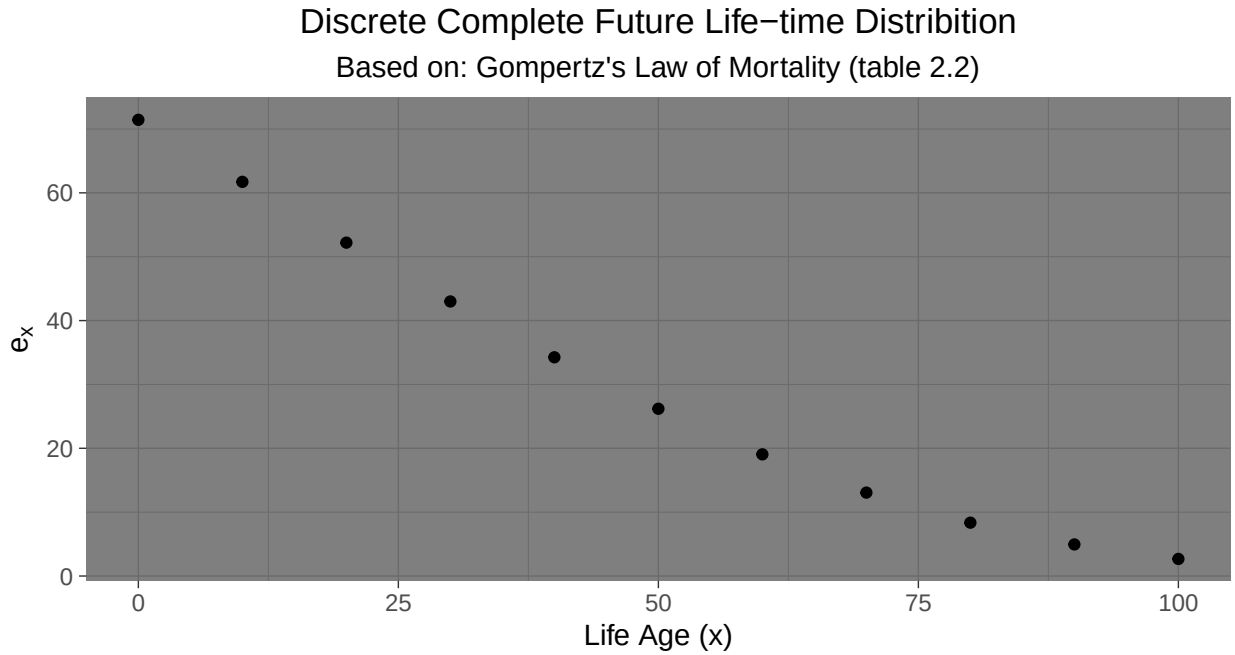




§2.1.2: Complete Future Life-Time

This R code snippet calculates and visualizes the complete future lifetime distribution based on Gompertz's Law of Mortality for ages ranging from 0 to 100 in increments of 10. Using the `discreteProbs` class, the code initializes mortality parameters (B) and (c) and sets the maximum age (ω) to 130. It iterates over the specified age list, computing the survival probabilities ${}_tp_x$ for each age, and accumulates these probabilities to calculate the complete future lifetime e_x and the adjusted complete future lifetime e_x^o. The results are then compiled into a data frame and plotted using `ggplot2` to display the discrete complete future lifetime distribution. The plot is customized with titles, subtitles, and axis labels, and the final results are displayed both as a plot and a formatted table using `knitr::kable`.

Code: `table 2.2: complete future life-time appendix`



age_x	e_x	e_o_x
0	71.437538	71.937538
10	61.722842	62.222842
20	52.202974	52.702974
30	42.992149	43.492149
40	34.251927	34.751927
50	26.191880	26.691880
60	19.051899	19.551899
70	13.057704	13.557704
80	8.354054	8.854054
90	4.943593	5.443593
100	2.673255	3.173255

§2.2: Life-Tables

Life tables, derived from survival models, provide empirical data on mortality rates and survival probabilities at various ages. They are vital tools for actuaries, offering crucial insights that inform the creation and pricing of insurance policies and annuities by illustrating survival likelihoods across different age groups.

§2.2.1: Select & Ultimate Life-Tables

This R code snippet extends the concepts from earlier sections on survival models to calculate and visualize select and ultimate life-tables for ages 20 to 110 using Gompertz's Law of Mortality. Building on survival probabilities, it initializes mortality parameters and sets an interest rate, then employs the discreteProbs class to generate life-table values. The code iterates over select periods, computing survival probabilities and life-table values for each period, demonstrating the dependency on previously established survival models. It calculates pure endowment values for terms of 5, 10, and 20 years, adjusting based on whether the term falls within the select or ultimate period. The results, which showcase survival probabilities and life-table

values, are compiled into a data frame and formatted for presentation, filtered for specific age ranges, and displayed using `knitr::kable`.

Code: `table 3.1: ultimate life-table appendix`

age+2	l_x+2	age_x	l_[x]+1	l_[x]	5_E_[x]	10_E_[x]	20_E_[x]
20	100000.00	18	NA	NA	NA	NA	NA
21	99975.04	19	NA	NA	NA	NA	NA
22	99949.71	20	99973.75	99995.08	0.7825546	0.6122692	0.3744139
23	99923.98	21	99948.40	99970.04	0.7825367	0.6122293	0.3743102
24	99897.79	22	99922.65	99944.63	0.7825167	0.6121845	0.3741937
25	99871.08	23	99896.43	99918.81	0.7824941	0.6121342	0.3740628
26	99843.80	24	99869.70	99892.52	0.7824688	0.6120776	0.3739157
27	99815.86	25	99842.38	99865.69	0.7824403	0.6120140	0.3737505
28	99787.20	26	99814.41	99838.28	0.7824082	0.6119426	0.3735648
29	99757.71	27	99785.70	99810.20	0.7823722	0.6118623	0.3733562
30	99727.29	28	99756.17	99781.36	0.7823318	0.6117720	0.3731219
31	99695.83	29	99725.70	99751.69	0.7822863	0.6116706	0.3728588
32	99663.20	30	99694.18	99721.06	0.7822352	0.6115566	0.3725632
33	99629.26	31	99661.48	99689.36	0.7821777	0.6114285	0.3722313
34	99593.83	32	99627.47	99656.47	0.7821131	0.6112846	0.3718585
35	99556.75	33	99591.96	99622.23	0.7820406	0.6111228	0.3714400
36	99517.80	34	99554.78	99586.47	0.7819590	0.6109410	0.3709701
37	99476.75	35	99515.73	99549.01	0.7818673	0.6107368	0.3704427
67	93398.05	65	93989.16	94467.11	0.7554530	0.5537115	0.2441055
68	92706.06	66	93363.38	93895.00	0.7521413	0.5468165	0.2314308
69	91936.88	67	92667.50	93258.63	0.7484363	0.5391689	0.2179684
70	91082.43	68	91894.03	92551.02	0.7442937	0.5307006	0.2037691
71	90133.96	69	91034.84	91764.58	0.7396647	0.5213410	0.1889103
72	89082.09	70	90081.15	90891.07	0.7344961	0.5110176	0.1734984
73	87916.84	71	89023.56	89921.62	0.7287297	0.4996579	0.1576711
74	86627.64	72	87852.03	88846.72	0.7223023	0.4871908	0.1415981
75	85203.46	73	86555.99	87656.25	0.7151455	0.4735487	0.1254795
76	83632.89	74	85124.37	86339.55	0.7071859	0.4586704	0.1095418
77	81904.34	75	83545.75	84885.49	0.6983450	0.4425045	0.0940311
78	80006.23	76	81808.54	83282.61	0.6885397	0.4250131	0.0792030
79	77927.35	77	79901.17	81519.30	0.6776828	0.4061768	0.0653085
80	75657.16	78	77812.44	79584.04	0.6656839	0.3859994	0.0525787
81	73186.31	79	75531.88	77465.70	0.6524505	0.3645140	0.0412073
82	70507.19	80	73050.22	75153.97	0.6378899	0.3417887	0.0313340

§2.3: Insurance Benefits

Insurance benefits are calculated by integrating survival probabilities to ensure adequate coverage. These benefits are based on empirical data from life tables, ensuring that payouts are financially viable and meet policyholders' needs. Accurately determining the present values of future payouts is crucial for the sustainability of insurance products.

§2.3.1: Discrete Insurance Benefits

This R code snippet continues the exploration of survival models by focusing on discrete insurance benefits. Building on earlier sections, it calculates and compares insurance benefits for various ages (30 to 32, 50 to

52, and 98 to 100) and monthly schedules (1, 4, 12, and 365 months). Using the `discreteProducts` class and an interest rate of 5%, it initializes product objects and computes whole life insurance benefit values for each age and monthly schedule. The results, showcasing insurance benefits for different age groups and schedules, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This demonstrates the application of survival models to determine insurance benefits, highlighting their role in actuarial calculations and product development.

Code: `table 4.1: insurance benefits comparison appendix`

age_x	A_x_(1)	A_x_(4)	A_x_(12)	A_x_(365)
30	0.0769828	0.0783982	0.0787172	0.0788721
31	0.0805419	0.0820227	0.0823565	0.0825185
32	0.0842693	0.0858186	0.0861678	0.0863374
50	0.1893079	0.1927898	0.1935745	0.1939553
51	0.1978038	0.2014422	0.2022621	0.2026601
52	0.2066380	0.2104392	0.2112957	0.2117114
98	0.8517747	0.8683882	0.8719789	0.8737015
99	0.8615266	0.8784388	0.8820775	0.8838209
100	0.8706841	0.8878989	0.8915840	0.8933472

§2.3.2: M-thly Insurance Benefits

Expanding upon the previous sections on insurance benefits, this R code snippet focuses on m-thly insurance benefits. It calculates and compares insurance benefits for ages spanning from 20 to 21, 50 to 51, and 127 to 128, using a monthly schedule of 12 months, an interest rate of 5%, and an annuity schedule of 'Due'. Utilizing the `discreteProducts` class, the code initializes product objects and computes probabilities ${}_tp_x$ and ${}_tq_x$ as well as whole-life insurance benefit values A_x for each age. The results, illustrating insurance benefits for different age groups and monthly schedules, are organized into a data frame and presented as a comparison table using `knitr::kable`. This underscores the application of survival models in determining insurance benefits and underscores their significance in actuarial calculations and product evaluation.

Code: `table 4.2: m-thly insurance benefits appendix`

age_x	p_x_(12)	q_x_(12)	A_x_(12)
20.00000	0.9999793	0.0000207	0.0503284
20.08333	0.9999793	0.0000207	0.0505138
20.16667	0.9999793	0.0000207	0.0507000
20.25000	0.9999793	0.0000207	0.0508869
20.33333	0.9999792	0.0000208	0.0510745
20.41667	0.9999792	0.0000208	0.0512629
20.50000	0.9999792	0.0000208	0.0514520
20.58333	0.9999792	0.0000208	0.0516419
20.66667	0.9999791	0.0000209	0.0518325
20.75000	0.9999791	0.0000209	0.0520239
20.83333	0.9999791	0.0000209	0.0522161
20.91667	0.9999791	0.0000209	0.0524090
21.00000	0.9999790	0.0000210	0.0526027
50.00000	0.9999036	0.0000964	0.1935745
50.08333	0.9999028	0.0000972	0.1942854
50.16667	0.9999020	0.0000980	0.1949987
50.25000	0.9999013	0.0000987	0.1957144
50.33333	0.9999005	0.0000995	0.1964324

age_x	p_x_(12)	q_x_(12)	A_x_(12)
50.41667	0.9998997	0.0001003	0.1971528
50.50000	0.9998989	0.0001011	0.1978756
50.58333	0.9998981	0.0001019	0.1986007
50.66667	0.9998972	0.0001028	0.1993282
50.75000	0.9998964	0.0001036	0.2000581
50.83333	0.9998956	0.0001044	0.2007904
50.91667	0.9998947	0.0001053	0.2015250
51.00000	0.9998939	0.0001061	0.2022621
127.00000	0.5308185	0.4691815	0.9914573
127.08333	0.5275379	0.4724621	0.9915161
127.16667	0.5242458	0.4757542	0.9915754
127.25000	0.5209423	0.4790577	0.9916360
127.33333	0.5176276	0.4823724	0.9916997
127.41667	0.5143019	0.4856981	0.9917701
127.50000	0.5109653	0.4890347	0.9918544
127.58333	0.5076180	0.4923820	0.9919669
127.66667	0.5042601	0.4957399	0.9921365
127.75000	0.5008920	0.4991080	0.9924217
127.83333	0.4975137	0.5024863	0.9929425
127.91667	0.4941254	0.5058746	0.9939456
128.00000	0.4907274	0.5092726	0.9959424

§2.3.3: Mean & Variance of Insurance Benefit EPVs

Building upon the previous sections on insurance benefits, this R code snippet focuses on calculating the mean and standard deviation of insurance benefit EPVs (Equivalent Present Values). It considers m-thly schedules (1, 4, and 12), an interest rate of 5%, and an annuity schedule of 'Due'. Age ranges from 20 to 100, with a benefit value S set at 10^5. The code initializes product objects for each m-thly variation using the `discreteProducts` class and computes the first and second moments of insurance benefit EPVs. It then calculates the mean and standard deviation using these moments. The results, demonstrating the mean and standard deviation for each age and m-thly schedule, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This underscores the application of survival models in assessing the variability of insurance benefit EPVs and their significance in actuarial calculations and risk analysis.

Code: `table 4.3: mean and standard deviation of insurance benefit EPV appendix`

age_x	mean_A_x_(1)	mean_A_x_(4)	mean_A_x_(12)	std_x_(1)	std_x_(4)	std_x_(12)
20	4921.934	5012.444	5032.843	5810.199	5917.485	5941.594
40	12105.921	12328.500	12378.671	9389.205	9561.076	9599.940
60	29028.218	29562.687	29683.038	15516.790	15800.616	15864.829
80	59293.307	60394.389	60640.815	17254.507	17576.873	17648.706
100	87068.415	88789.887	89158.402	7860.275	8073.515	8110.447

§2.3.4: Term Insurance Benefits

Expanding on the analysis of insurance benefits, this R code snippet focuses specifically on term insurance benefits. It calculates term insurance benefits for ages ranging from 20 to 100 and for term periods of 10 years. Utilizing m-thly schedules of 1, 4, and 12, an interest rate of 5%, and an annuity schedule of 'Due', the code initializes product objects for each m-thly variation using the `discreteProducts` class. It then computes term insurance benefits for each age and m-thly schedule. The results, illustrating term insurance

benefits for different age groups and m-thly schedules, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This demonstrates the application of survival models in determining term insurance benefits and underscores their importance in actuarial calculations and risk management.

Code: `table 4.4: term insurance benefits appendix`

age_x	A_x´*10*(1)	A_x´*10*(4)	A_x´*10*(12)
20	0.0020875	0.0021260	0.0021347
40	0.0057320	0.0058365	0.0058602
60	0.0425209	0.0432926	0.0434682
80	0.3372216	0.3434060	0.3448026
100	0.8694208	0.8866070	0.8902866

§2.3.5: Endowment Insurance Benefits

This R code snippet delves into the analysis of endowment insurance benefits. It calculates endowment insurance benefits for ages ranging from 20 to 100 and for a term period of 10 years. Employing m-thly schedules of 1, 4, and 12, an interest rate of 5%, and an annuity schedule of 'Due', the code initializes product objects for each m-thly variation using the `discreteProducts` class. It then computes endowment insurance benefits for each age and m-thly schedule. The resulting benefits, representing different age groups and m-thly schedules, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This highlights the application of survival models in determining endowment insurance benefits and underscores their significance in actuarial calculations and financial planning.

Code: `table 4.5: endowment insurance benefits appendix`

age_x	A_x_10_(1)	A_x_10_(4)	A_x_10_(12)
20	0.6143265	0.6143651	0.6143737
40	0.6149367	0.6150413	0.6150650
60	0.6211644	0.6219360	0.6221116
80	0.6767371	0.6829215	0.6843180
100	0.8707765	0.8879628	0.8916423

§2.3.6: Ratios of Insurance Benefits

This R code snippet focuses on calculating the ratios of insurance benefits, providing insight into comparative benefit values. It computes these ratios for ages ranging from 20 to 120, considering different pairs of m-thly schedules. Using m-thly ratio schedules of (4, 1) and (12, 1), an interest rate of 5%, and an annuity schedule of 'Due', the code initializes product objects for both numerator and denominator distributions using the `discreteProducts` class. It then computes the insurance benefit ratios for each age and ratio schedule pair. The resulting ratios, indicating the relationship between benefit values for different m-thly schedules, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This highlights the application of survival models in assessing and comparing insurance benefit values, offering valuable insights for actuarial analysis and product evaluation.

Code: `table 4.6: ratios of insurance benefits appendix`

age_x	A_x_(4) / A_x_(1)	A_x_(12) / A_x_(1)
20	1.018389	1.022534
40	1.018386	1.022530
60	1.018412	1.022558

age_x	A_x_(4) / A_x_(1)	A_x_(12) / A_x_(1)
80	1.018570	1.022726
100	1.019772	1.024004
120	1.029636	1.034607

§2.4: Annuities

Annuities provide regular payments over the annuitant's lifetime and rely heavily on survival models and life tables for valuation. Various types of annuities (immediate, deferred, fixed, and variable) require precise calculations of expected present values to ensure they provide sufficient income streams while remaining financially sustainable.

§2.4.1: Whole-Life Annuities

This R code snippet focuses on calculating whole-life annuities, providing insights into annuity values across different m-thly schedules and annuity payment timings. It computes whole-life annuities for ages ranging from 20 to 100, considering m-thly schedules of 1, 4, and 12, and both 'Due' and 'Immediate' annuity schedules. The code initializes product objects for each combination of m-thly schedule and annuity schedule using the `discreteProducts` class. It then computes the whole-life annuity values for each age and schedule combination. The resulting annuity values, representing different age groups and payment timings, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This highlights the application of survival models in assessing and comparing annuity values, offering valuable insights for actuarial analysis and financial planning.

Code: `table 5.1: whole-life annuities appendix`

age_x	aD_x_(1)	aI_x_(1)	aD_x_(4)	aI_x_(4)	aD_x_(12)	aI_x_(12)
20	19.966394	18.966394	19.587563	19.337563	19.504002	19.420669
40	18.457757	17.457757	18.078905	17.828905	17.995344	17.912010
60	14.904074	13.904074	14.525011	14.275011	14.441437	14.358104
80	8.548406	7.548406	8.167148	7.917148	8.083443	8.000110
100	2.715633	1.715633	2.311659	2.061659	2.226607	2.143274

§2.4.2: Term Annuities

This R code snippet focuses on calculating term annuities, offering insights into annuity values for specified term periods across different m-thly schedules and annuity payment timings. It computes term annuities for ages ranging from 20 to 100 and for a term period of 10 years. Considering m-thly schedules of 1 and 4, as well as both 'Due' and 'Immediate' annuity schedules, the code initializes product objects for each combination of m-thly schedule and annuity schedule using the `discreteProducts` class. It then computes the term annuity values for each age and schedule combination. The resulting annuity values, representing different age groups, term periods, and payment timings, are compiled into a data frame and presented as a comparison table using `knitr::kable`. This highlights the application of survival models in assessing and comparing annuity values over specific term periods, offering valuable insights for actuarial analysis and financial planning.

Code: `table 5.2: term annuities appendix`

age_x	aD_x´10(1)	aI_x´10(1)	aD_x´10(4)	aI_x´10(4)
20	8.099144	7.711383	7.952251	7.855311
40	8.086329	7.695533	7.938306	7.840607
60	7.955548	7.534192	7.796128	7.690789
80	6.788521	6.128036	6.538536	6.373415
100	2.713694	1.715049	2.310341	2.060680

§2.4.3: Select & Ultimate Insurance Benefits and Annuities

This section delves into computing insurance benefits and annuity values, bridging select and ultimate periods while considering various advanced concepts in R programming. Initially, it establishes parameters such as m-thly schedules, interest rates, and annuity schedules, alongside age lists and terms. Employing the `discreteProducts` and `discreteProbs` classes, the code calculates insurance benefits and annuities, encompassing select and ultimate periods. Key advanced R concepts utilized include dynamic variable naming, conditional computation based on select periods, and recursive calculations to handle multiple select periods. By iteratively computing survival probabilities and benefit values, the snippet demonstrates sophisticated data manipulation techniques and showcases the integration of survival models into actuarial analysis. The resulting table presents a comprehensive overview of insurance benefits and annuities, offering valuable insights for actuarial assessments and financial planning strategies.

Code: `table 5.3: insurance benefits and annuity values appendix`

age+2	A_x+2	a_x+2	age_x	2A_[x]	A_[x]	a_[x]
20	0.0492193	19.966394	18	0.0051492	0.0450338	20.054290
21	0.0514435	19.919686	19	0.0054387	0.0470553	20.011840
22	0.0537760	19.870704	20	0.0057551	0.0491755	19.967315
23	0.0562218	19.819342	21	0.0061007	0.0513991	19.920619
24	0.0587862	19.765489	22	0.0064781	0.0537309	19.871650
25	0.0614746	19.709033	23	0.0068903	0.0561761	19.820303
26	0.0642927	19.649852	24	0.0073401	0.0587397	19.766467
27	0.0672464	19.587826	25	0.0078310	0.0614272	19.710029
28	0.0703417	19.522824	26	0.0083665	0.0642443	19.650870
29	0.0735850	19.454715	27	0.0089505	0.0671968	19.588867
30	0.0769828	19.383361	28	0.0095871	0.0702909	19.523892
31	0.0805419	19.308620	29	0.0102810	0.0735327	19.455813
32	0.0842693	19.230344	30	0.0110369	0.0769289	19.384493
33	0.0881722	19.148383	31	0.0118600	0.0804862	19.309790
34	0.0922581	19.062580	32	0.0127561	0.0842115	19.231558
35	0.0965346	18.972774	33	0.0137312	0.0881121	19.149645
36	0.1010096	18.878799	34	0.0147918	0.0921954	19.063896
37	0.1056912	18.780485	35	0.0159450	0.0964690	18.974151
67	0.3831331	12.954204	65	0.1533320	0.3540744	13.564437
68	0.3978282	12.645607	66	0.1641099	0.3680148	13.271688
69	0.4128481	12.330190	67	0.1754813	0.3823009	12.971682
70	0.4281760	12.008304	68	0.1874577	0.3969203	12.664674
71	0.4437929	11.680350	69	0.2000480	0.4118586	12.350970
72	0.4596769	11.346785	70	0.2132581	0.4270987	12.030927
73	0.4758039	11.008118	71	0.2270907	0.4426214	11.704950
74	0.4921470	10.664912	72	0.2415447	0.4584047	11.373501
75	0.5086769	10.317785	73	0.2566152	0.4744242	11.037091
76	0.5253617	9.967405	74	0.2722929	0.4906529	10.696288
77	0.5421671	9.614491	75	0.2885639	0.5070614	10.351710

age+2	A_x+2	a_x+2	age_x	2A_[x]	A_[x]	a_[x]
78	0.5590568	9.259808	76	0.3054092	0.5236179	10.004025
79	0.5759922	8.904164	77	0.3228049	0.5402880	9.653951
80	0.5929331	8.548406	78	0.3407217	0.5570357	9.302251
81	0.6098376	8.193410	79	0.3591249	0.5738225	8.949728
82	0.6266628	7.840081	80	0.3779744	0.5906085	8.597221

§3: Remarks

§3.1: Next Steps

The next steps of the research are geared towards achieving short and long-term goals outlined in the context of the proposal. In the short term, the focus is on interacting with potential user bases and collaborators to gather insights and feedback on the proposed methodologies and techniques. This interaction serves as a vital step in validating the relevance and applicability of the research to the actuarial community. Sharing initial findings with collaborators and stakeholders allows for iterative improvements and adjustments to align the work with the actual needs and challenges faced by actuarial practitioners.

In the long term, the research aims to translate its findings into tangible outcomes that benefit both the actuarial community and broader stakeholders. One key goal is the development of ExaminationAid® dashboards, which will serve as comprehensive tools for exam preparation, educational purposes, and professional development within the actuarial science field. Additionally, the research aims to disseminate its findings through publications, contributing valuable insights to the Actuarial Science CRAN community and academic discourse. By sharing work with the CRAN community, the research seeks to foster collaboration and innovation in actuarial modeling and data science applications. Moreover, the research aims to explore other avenues where the concepts and techniques developed can be applied, potentially expanding into related fields and industries beyond actuarial science. Through these long-term goals, the research aspires to make a lasting impact on actuarial education, practice, and research, driving advancements in the field and enhancing the capabilities of actuarial professionals worldwide.

§4: Appendix

```
## #example 2.4: Gompertz survival model
## #-------------------------------------------------------------------------------code
## #discrete life ages
## age_list <- list(20,50,80)
## #mortality parameters
## B <- .0003; c <- 1.07
## #initialize discreteProbs object
## probFunc1 <- discreteProbs$new(
##   #age distribution range
##   age_range=list(0,120),
##   #mortality law
##   mort_law='Gompertz',
##   #mortality parameters
##   other_params=list(B,c))
## #initialize t_p_x plots
## tempPlot1 <- ggplot()
## #initialize t_f_x plots
## tempPlot2 <- ggplot()
```

```
## #create unique plots per life age (x)
## for(x in age_list){
##   #create data frame with desired probabilities...initialize future ages...max age of 120
##   tempData <- data.frame(age=c(0:(120-(x+1)))) %>%
##     #compute survival probability...t_p_x
##     mutate(p=sapply(age, probFunc1$p_x_t, x=x)) %>%
##     #compute future life-time pdf...t_f_x
##     mutate(f=sapply(age, probFunc1$f_x_t, x=x)) %>%
##     #assign age (x) indicator from age_list
##     mutate(age_x=factor(x))
##   #add computed values to plots
##   #plot t_p_x for age (x)
##   tempPlot1 <- tempPlot1 + geom_point(data=tempData, aes(age, p, colour=age_x))
##   #plot t_f_x for age (x)
##   tempPlot2 <- tempPlot2 + geom_point(data=tempData, aes(age, f, colour=age_x))
## }
## #customize plots
## #define color scheme for plot legend
## cols <- c("20" = "blue", "50" = "red", "80" = "grey")
## tempPlot1 <- tempPlot1 +
##              labs(
##                title='Discrete Future Life-time Distribitions',
##                x = TeX(r'(Number of Discrete Future Periods $\left{\sum_{k=0}^{\omega-(x+1)}\right}$)
##                y = TeX(r'($S_{x}(t)$)'),
##                subtitle = "Based on: Gompertz's Law of Mortality (example 2.4)",
##                color='Life Age (x)'
##              ) +
##              theme_dark() +
##              theme(plot.title = element_text(hjust = 0.5)) +
##              theme(plot.subtitle = element_text(hjust = 0.5)) +
##              scale_colour_manual(values = cols)
## tempPlot2 <- tempPlot2 +
##              labs(
##                title='Discrete Future Life-time Distribitions',
##                x = TeX(r'(Number of Discrete Future Periods $\left{\sum_{k=0}^{\omega-(x+1)}\right}$)
##                y = TeX(r'($f_{x}(t)$)'),
##                subtitle = "Based on: Gompertz's Law of Mortality (example 2.4)",
##                color='Life Age (x)'
##              ) +
##              theme_dark() +
##              theme(plot.title = element_text(hjust = 0.5)) +
##              theme(plot.subtitle = element_text(hjust = 0.5)) +
##              scale_colour_manual(values = cols)
## #display plots
## print(tempPlot1); print(tempPlot2)

## #table 2.2: complete future life-time
## #-------------------------------------------------------------------------------code
## #desired ages
## age_list <- seq(0,100,10)
## #mortality parameters
## B <- .0003; c <- 1.07; omega <- 130
## #initialize discreteProbs object
## probFunc2 <- discreteProbs$new(
```

```
##   #age distribution range
##   age_range=list(0,omega),
##   #mortality law
##   mort_law='Gompertz',
##   #mortality parameters
##   other_params=list(B,c))
## #create base data frame with desired figures
## #initialize discrete list
## e_x_list <- list()
## #initialize continuous list
## e_o_x_list <- list()
## #perform iterations per age in input list
## for (x in age_list){
##   #setup distribution age range
##   table2_2_1 <- data.frame(age=c(1:(omega-(x+1)))) %>%
##     #compute t_p_x
##     mutate(p=sapply(age, probFunc2$p_x_t, x=x))
##   #compute and append e_x to e_x_list per x
##   e_x_list <- append(e_x_list,sum(table2_2_1$p))
##   #compute and append e_o_x to e_o_x_list per x
##   e_o_x_list <- append(e_o_x_list,(sum(table2_2_1$p)+0.5))
## }
## #plot results
## table2_2_2 <- data.frame(
##   #define table output...age_x || e_x || e_o_x
##   age_x = age_list,
##   e_x = unlist(e_x_list),
##   e_o_x = unlist(e_o_x_list)
## )
## plot2_2 <- ggplot(data=table2_2_2) +
##   geom_point(aes(x=age_x,y=e_x))+
##   labs(
##        title='Discrete Complete Future Life-time Distribition',
##        x = 'Life Age (x)',
##        y = TeX(r'($e_{x}$)'),
##        subtitle = "Based on: Gompertz's Law of Mortality (table 2.2)"
##      ) +
##    theme_dark() +
##    theme(plot.title = element_text(hjust = 0.5)) +
##    theme(plot.subtitle = element_text(hjust = 0.5)) +
##    scale_colour_manual(values = cols)
## print(plot2_2); knitr::kable(table2_2_2, align = "c")

## #table 3.1: ultimate life-table
## #-------------------------------------------------------------------------------code
## d <- 2; age_list <- list(20,110); i <- .05; v <- 1/(1+i)
## radix <- 10^5; terms <- c(5,10,20)
## probFunc3 <- discreteProbs$new()
## #create data frame with desired life-table values
## #define age range
## table_3_7_1 <- data.frame(age_x=c((age_list[[1]]-d):age_list[[2]])) %>%
##   #define ultimate period age
##   mutate(!!paste('age+',d,sep=''):=age_x+d) %>%
##   #define ultimate period life-table value
```

```
##   mutate(!!paste('l_x+',d,sep=''):=sapply(!!sym(paste('age+',d,sep='')),
##                                           probFunc3$lt_x_t,t=0,
##                                           radix=radix,
##                                           x0=age_list[[1]]))
##   for(select in c(1:d)){
##     #loop through periods
##     temp_name <- paste("p_[x]_",select,sep='')
##     #initialize survival feature name
##     table_3_7_1 <- table_3_7_1 %>%
##       #compute select period survival probabilities
##       mutate(!!temp_name:=sapply(age_x,probFunc3$select_p_x_t,t=select,d=d))
##     if (select > 1){ #setup period survival probabilities
##       #initialize survival feature name...select-1
##       previous_temp_name <- paste("p_[x]_",select-1,sep='')
##       #initialize survival feature name...select
##       current_temp_name <- paste("p_[x]_",select,sep='')
##       #initialize survival feature name...select-1
##       new_p_name <- paste("p_[x]+",select-1,"_1",sep='')
##       #initialize life-table feature name...select
##       new_l_name <- paste("l_[x]+",select-1,sep='')
##       table_3_7_1 <- table_3_7_1 %>%
##         #compute survival probabilities
##         mutate(!!new_p_name:= !!sym(current_temp_name) / !!sym(previous_temp_name)) %>%
##         #compute life-table values
##         mutate(!!new_l_name:= !!sym(paste('l_x+',d,sep=''))/!!sym(new_p_name))
##       table_3_7_1[1:d,new_l_name] <- NA
##     }
##     if (select == d){
##       #initialize life-table feature name
##       l_name <- 'l_[x]'
##       table_3_7_1 <- table_3_7_1 %>%
##         #compute life-table values
##         mutate(!!l_name := !!sym(paste('l_x+',d,sep=''))/!!sym(paste("p_[x]_",select,sep='')))
##       table_3_7_1[1:d,l_name] <- NA
##     }
##   }
## #compute varying select pure endowment
## for(term in terms){
##   if(term < d){ #within select period
##     temp_numerator <- 'l_[x]'
##     lag_level <- term
##   }else if(term >= d){ #within ultimate period
##     temp_numerator <- paste('l_x+',d,sep='')
##     lag_level <- term - d
##   }
##   temp_denominator <- 'l_[x]'
##   temp_list <- list()
##   table_3_7_1 <- table_3_7_1 %>%
##     mutate(!!paste(term,'_E_[x]',sep=''):=(v^term)*lead(!!sym(temp_numerator),lag_level)/!!sym(temp_
##   }
## #query output
## table_3_7_2 <- table_3_7_1 %>%
##   #filter columns
##   select(c('age+2','l_x+2','age_x','l_[x]+1','l_[x]','5_E_[x]','10_E_[x]','20_E_[x]')) %>%
```

```
##   filter(age_x %in% c((age_list[[1]]-d):(age_list[[1]]+15),(80-15):80))
## knitr::kable(table_3_7_2, align = "c")

## #table 4.1: insurance benefits
## #-------------------------------------------------------------------------------code
## #define m_thly schedules
## m_list <- c(1,4,12,365)
## #define interest rate and annuity schedule
## i <- .05; annuity_schedule <- 'Due'
## #define age list
## age_list <- c(30:32,50:52,98:100)
## #initialize output table
## table_4_1_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){
##   #initialize product object
##   tempPB <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th)
##   table_4_1_1 <- table_4_1_1 %>%
##     #compute whole life insurance benefit values
##     mutate(!!paste('A_x_(',m_th,')',sep=''):=sapply(age_x,tempPB$whole_A_x,moment=1))
## }
## table_4_1_2 <- table_4_1_1 %>%
##   #filter columns
##   select(!contains('p'))
## knitr::kable(table_4_1_2, align = "c")

## #table 4.2: insurance benefits
## #-------------------------------------------------------------------------------code
## #define m_thly schedule, interest rate, and annuity schedule
## m_th <- 12; i <- .05; annuity_schedule <- 'Due'
## #define ages
## age_list <- c(seq(20,21,1/m_th),seq(50,51,1/m_th),seq(127,128,1/m_th))
## #define products class
## productFun1 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th)
## table_4_2_1 <- data.frame(age_x=age_list) %>%
##   #compute t_p_x
##   mutate('p_x_(12)'=sapply(age_x,productFun1$p_x_t,t=1)) %>%
##   #compute t_q_x
##   mutate('q_x_(12)'=sapply(age_x,productFun1$q_x_t,t=1)) %>%
##   #compute whole-life probability
##   mutate('A_x_(12)'=sapply(age_x,productFun1$whole_A_x,moment=1))
## knitr::kable(table_4_2_1, align = "c")

## #table 4.3: mean and standard deviation of insurance benefit EPV
## #-------------------------------------------------------------------------------code
## #define m_thly schedules, interest rate, and annuity schedule
## m_list <- c(1,4,12); i <- .05; annuity_schedule <- 'Due'
## #define ages and benefit value
## age_list <- seq(20,100,20); S <- 10^5
## #initialize output table
## table_4_3_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){
##   #define case product object...m_th variation
##   productFun2 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th)
```

```
##    table_4_3_1 <- table_4_3_1 %>%
##     #compute first moment
##     mutate(!!paste('A_x_(',m_th,')',sep=''):=sapply(age_x,productFun2$whole_A_x,moment=1)) %>%
##     #compute second moment
##     mutate(!!paste('2_A_x_(',m_th,')',sep=''):=sapply(age_x,productFun2$whole_A_x,moment=2)) %>%
##     #compute mean
##     mutate(!!paste('mean_A_x_(',m_th,')',sep=''):=S*!!sym(paste('A_x_(',m_th,')',sep=''))) %>%
##     #compute standard deviation
##     mutate(!!paste('std_x_(',m_th,')',sep=''):=S*sqrt(!!sym(paste('2_A_x_(',m_th,')',sep=''))-
##                                                         (!!sym(paste('A_x_(',m_th,')',sep=''))*
##                                                            !!sym(paste('A_x_(',m_th,')',sep='')))))
## }
## table_4_3_2 <- table_4_3_1 %>%
##   #collect age, mean, and std columns
##   select(contains(c('age','mean','std')))
## knitr::kable(table_4_3_2, align = "c")

## #table 4.4: term insurance benefits
## #-------------------------------------------------------------------------------code
## #define m_thly schedules, interest rate, and annuity schedule
## m_list <- c(1,4,12); i <- .05; annuity_schedule <- 'Due'
## #define ages and term period
## age_list <- seq(20,100,20); term <- 10
## #initialize output table
## table_4_4_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){
##   #define case product object...m_th variation
##   productFun3 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th)
##   table_4_4_1 <- table_4_4_1 %>%
##     #compute term insurance benefit
##     mutate(!!paste('A_x´_',term,'_(',m_th,')',sep=''):=sapply(age_x,productFun3$term_A_x,moment=1,n=
## }
## knitr::kable(table_4_4_1, align = "c")

## #table 4.5: endowment insurance benefits
## #-------------------------------------------------------------------------------code
## #define m_thly schedules, interest rate, and annuity schedule
## m_list <- c(1,4,12); i <- .05; annuity_schedule <- 'Due'
## #define ages and term period
## age_list <- seq(20,100,20); term <- 10
## #initialize output table
## table_4_5_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){
##   #define case product object...m_th variation
##   productFun4 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th)
##   table_4_5_1 <- table_4_5_1 %>%
##     #compute endowment insurance benefit
##     mutate(!!paste('A_x_',term,'_(',m_th,')',sep=''):=sapply(age_x,productFun4$endowment_A_x,moment=
## }
## knitr::kable(table_4_5_1, align = "c")

## #table 4.6: ratios of insurance benefits
## #-------------------------------------------------------------------------------code
```

```
## #define m_thly ratio schedules, interest rate, and annuity schedule
## ratio_mth <- list(c(4,1),c(12,1)); i <- .05; annuity_schedule <- 'Due'
## #define ages
## age_list <- seq(20,120,20)
## #initialize output table
## table_4_6_1 <- data.frame(age_x=age_list)
## #define case product object...m_th variation
## for(m_th in ratio_mth){
##   #initialize numerator product distribution
##   productFun5 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th[[1]])
##   #initialize denominator product distribution
##   productFun6 <- discreteProducts$new(i=i,annuity_schedule=annuity_schedule,m=m_th[[2]])
##   #append computations to initialized output table
##   table_4_6_1 <- table_4_6_1 %>%
##     #compute insurance benefit ratios
##     mutate(!!paste('A_x_(',m_th[[1]],') / A_x_(',m_th[[2]],')',sep='') := sapply(age_x,
##                                                                                  productFun5$whole_A
##                                                                                  moment=1)/
##                                                                               sapply(age_x,
##                                                                                      productFun6$whole
##                                                                                      moment=1))
## }
## knitr::kable(table_4_6_1, align = "c")

## #table 5.1: whole-life annuities
## #-------------------------------------------------------------------------------code
## #define m_thly schedules, interest rate, and annuity schedules
## m_list <- c(1,4,12); i <- .05; annuity_schedules <- c('Due','Immediate')
## #define ages
## age_list <- seq(20,100,20)
## #initialize output table
## table_5_1_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){ #m-thly variation
##   for(schedule in annuity_schedules){ #annuity schedule variation
##     #define case product object...m_th and schedule variation
##     productFun7 <- discreteProducts$new(i=i,annuity_schedule=schedule,m=m_th)
##     table_5_1_1 <- table_5_1_1 %>%
##       #compute whole-life annuity values
##       mutate(!!paste('a',substr(schedule,1,1),'_x_(',m_th,')',sep=''):=sapply(age_x,
##                                                                               productFun7$whole_a_x)
## }}
## knitr::kable(table_5_1_1, align = "c")

## #table 5.2: term annuities
## #-------------------------------------------------------------------------------code
## #define m_thly schedules, interest rate, and annuity schedules
## m_list <- c(1,4); i <- .05; annuity_schedules <- c('Due','Immediate')
## #define ages and term period
## age_list <- seq(20,100,20); term <- 10
## #initialize output table
## table_5_2_1 <- data.frame(age_x=age_list)
## for(m_th in m_list){ #m-thly variation
##   for(schedule in annuity_schedules){ #annuity schedule variation
##     #define case product object...m_th and schedule variation
```

```
##     productFun8 <- discreteProducts$new(i=i,annuity_schedule=schedule,m=m_th)
##     table_5_2_1 <- table_5_2_1 %>%
##       #compute term annuity values
##       mutate(!!paste('a',substr(schedule,1,1),'_x´_',term,'_(',m_th,')',sep=''):=sapply(age_x,
##                                                                                         productFun8$
##                                                                                         n=term))
## }}
## knitr::kable(table_5_2_1, align = "c")

## #table 5.3: insurance benefits and annuity values
## #-------------------------------------------------------------------------------code
## d <- 2; age_list <- list(20,80); i <- .05; v <- 1/(1+i) #compute annual discount rate
## S <- 10^5; terms <- c(5,10,20)
## productFun9 <- discreteProducts$new()
## probFunc4 <- discreteProbs$new()
## #create data frame with desired life-table values
## #define age range
## table_5_3_1 <- data.frame(age_x=c((age_list[[1]]-d):(age_list[[1]]+15),
##                                   (age_list[[2]]-15):age_list[[2]])) %>%
##   #define ultimate period age
##   mutate(!!paste('age+',d,sep=''):=age_x+d)  %>%
##   #define ultimate period life-table value
##   #first-moment insurance benefit value
##   mutate(!!paste('A_x+',d,sep=''):=sapply(!!sym(paste('age+',d,sep='')),
##                                           productFun9$whole_A_x,
##                                           moment=1)) %>%
##   #second-moment insurance benefit value
##   mutate(!!paste('2A_x+',d,sep=''):=sapply(!!sym(paste('age+',d,sep='')),
##                                            productFun9$whole_A_x,
##                                            moment=2)) %>%
##   #annuity value
##   mutate(!!paste('a_x+',d,sep=''):=sapply(!!sym(paste('age+',d,sep='')),
##                                           productFun9$whole_a_x))
##   #compute helper columns and products value
##   for(select in c(1:d)){
##     #loop through periods
##     temp_name <- paste("p_[x]_",select,sep='')
##     #initialize survival feature name
##     table_5_3_1 <- table_5_3_1 %>%
##       #compute select period survival probabilities
##       mutate(!!temp_name:=sapply(age_x,
##                                  probFunc4$select_p_x_t,
##                                  t=select,
##                                  d=d))
##     if (select > 1){ #setup period survival probabilities
##       #initialize survival feature name...select-1
##       previous_temp_name <- paste("p_[x]_",select-1,sep='')
##       #initialize survival feature name...select
##       current_temp_name <- paste("p_[x]_",select,sep='')
##       #initialize survival feature name...select-1
##       new_p_name <- paste("p_[x]+",select-1,"_1",sep='')
##       table_5_3_1 <- table_5_3_1 %>%
##         #compute survival probabilities
##         mutate(!!new_p_name:= !!sym(current_temp_name) / !!sym(previous_temp_name))
```

```
##     }
##     if (select == d){
##       #assign previous and new variable names
##       if(d == 1){
##         #1 term select period
##         previous_A_name <- 'A_x+1'; current_A_name <- 'A_[x]'
##         previous_2A_name <- '2A_x+1'; current_2A_name <- '2A_[x]'
##         previous_a_name <- 'a_x+1'; current_a_name <-  'a_[x]'
##         #compute current variables
##         current_p_name <- "p_[x]_1"
##         table_5_3_1 <- table_5_3_1 %>%
##           mutate(!!current_A_name := (v*(1-!!sym(current_p_name)))+(v*!!sym(current_p_name)*
##                                                                     !!sym(previous_A_name))) %>%
##           mutate(!!current_a_name := 1+(v*!!sym(current_p_name)*!!sym(previous_a_name))) %>%
##           mutate(!!current_2A_name := ((v^2)*(1-!!sym(current_p_name)))+((v^2)*!!sym(current_p_name)*
##                                                                         !!sym(previous_2A_name)))
##       }else if(d == 2){
##         #2 term select period
##         previous_A_name <- 'A_x+2'; current_A_name <-  'A_[x]+1'; new_A_name <- 'A_[x]'
##         previous_2A_name <- '2A_x+2'; current_2A_name <-  '2A_[x]+1'; new_2A_name <- '2A_[x]'
##         previous_a_name <- 'a_x+2'; current_a_name <-  'a_[x]+1'; new_a_name <- 'a_[x]'
##         #compute current variables
##         current_p_name <- "p_[x]+1_1"
##         new_p_name <- "p_[x]_1"
##         table_5_3_1 <- table_5_3_1 %>%
##           #initialize first-moment benefit parameters
##           mutate(!!current_A_name := (v*(1-!!sym(current_p_name)))+(v*!!sym(current_p_name)*
##                                                                     !!sym(previous_A_name))) %>%
##           #initialize first-moment benefit parameters
##           mutate(!!new_A_name := (v*(1-!!sym(new_p_name)))+(v*!!sym(new_p_name)*!!sym(current_A_name)
##           #initialize first-moment annuity parameters
##           mutate(!!current_a_name := 1+(v*!!sym(current_p_name)*!!sym(previous_a_name))) %>%
##           #initialize first-moment annuity parameters
##           mutate(!!new_a_name := 1+(v*!!sym(new_p_name)*!!sym(current_a_name))) %>%
##           #initialize second-moment benefit parameters
##           mutate(!!current_2A_name := ((v^2)*(1-!!sym(current_p_name)))+((v^2)*!!sym(current_p_name)*
##                                                                         !!sym(previous_2A_name)))
##           #initialize second-moment benefit parameters
##           mutate(!!new_2A_name := ((v^2)*(1-!!sym(new_p_name)))+((v^2)*!!sym(new_p_name)*
##                                                                  !!sym(current_2A_name)))
##       }else if(d > 2){
##         #when select period exceeds 2
##         for(select2 in c((d-1):1)){
##           if(select2 == d-1){
##             #initialize first-moment benefit parameters
##             previous_A_name <- paste('A_x+',d,sep='')
##             current_A_name <- paste('A_[x]+',select2,sep='')
##             #initialize second-moment benefit parameters
##             previous_2A_name <- paste('2A_x+',d,sep='')
##             current_2A_name <- paste('2A_[x]+',select2,sep='')
##             #initialize first-moment annuity parameters
##             previous_a_name <- paste('a_x+',d,sep='')
##             current_a_name <- paste('a_[x]+',select2,sep='')
##           }else if(select2 > 1&&select < (d-1)){
```

```
##             #initialize first-moment benefit parameters
##             previous_A_name <- paste('A_[x]+',select2+1,sep='')
##             current_A_name <- paste('A_[x]+',select2,sep='')
##             #initialize second-moment benefit parameters
##             previous_2A_name <- paste('2A_[x]+',select2+1,sep='')
##             current_2A_name <- paste('2A_[x]+',select2,sep='')
##             #initialize first-moment annuity parameters
##             previous_a_name <- paste('a_[x]+',select2+1,sep='')
##             current_a_name <- paste('a_[x]+',select2,sep='')
##           }else if(select2 == 1){
##             #initialize first-moment benefit parameters
##             previous_A_name <- paste('A_[x]+',select2+1,sep='')
##             current_A_name <- 'A_[x]'
##             #initialize second-moment benefit parameters
##             previous_2A_name <- paste('2A_[x]+',select2+1,sep='')
##             current_2A_name <- '2A_[x]'
##             #initialize first-moment annuity parameters
##             previous_a_name <- paste('a_[x]+',select2+1,sep='')
##             current_a_name <- 'a_[x]'}
##         #compute current variables
##         current_p_name <- paste("p_[x]+",select2,"_1",sep='')
##         table_5_3_1 <- table_5_3_1 %>%
##           #compute first-moment benefit values
##           mutate(!!current_A_name := (v*(1-!!sym(current_p_name)))+(v*!!sym(current_p_name)*
##                                                                       !!sym(previous_A_name))) %>%
##           #compute first-moment annuity values
##           mutate(!!current_a_name := 1+(v*!!sym(current_p_name)*!!sym(previous_a_name))) %>%
##           #compute first-moment annuity values
##           mutate(!!new_a_name := 1+(v*!!sym(new_p_name)*!!sym(current_a_name))) %>%
##           #compute second-moment benefit values
##           mutate(!!current_2A_name := ((v^2)*(1-!!sym(current_p_name)))+((v^2)*!!sym(current_p_name)*
##                                                                           !!sym(previous_2A_name)))
##           #compute second-moment benefit values
##           mutate(!!new_2A_name := ((v^2)*(1-!!sym(new_p_name)))+((v^2)*!!sym(new_p_name)*
##                                                                  !!sym(current_2A_name)))}}}}
## #query output
## table_5_3_2 <- table_5_3_1 %>%
##   #filter columns
##   select(c('age+2','A_x+2','a_x+2','age_x','2A_[x]','A_[x]','a_[x]'))
## knitr::kable(table_5_3_2, align = "c")
```

§5: References


```
## Dutang C, Goulet V (2024). _ CRAN Task View: Actuarial Science_.
## <https://CRAN.R-project.org/view=ActuarialScience>.

## C.Dickson D, Hardy MR, Waters HR (2020). _Actuarial Mathematics for
## Life Contingent Risks_. Cambridge University Press, Cambridge. ISBN
## 9781108478083, <www.cambridge.org/isas>.

## Wickham H (2019). _Advanced R_. Chapman and Hall/CRC. ISBN
## 9780815384571, <https://adv-r.hadley.nz/#welcome>.
```