中国人口超老龄化进程预测研究

——基于队列要素与 Logistic 模型

参赛选手：龙川县第一中学　余鸿龙、钟龙杰、马天翔
指导老师：龙川县第一中学　魏海燕、邓清霞

摘 要

如今我国人口老龄化程度较深，60 岁及以上人口超 2.9 亿、占比超 21%，且进程快于多数发达国家，呈现“未富先老”特征；农村和中西部地区老龄化问题更突出，城乡与区域差异明显，这与生育率低、预期寿命延长及家庭养老功能弱化等因素密切相关。本文主要研究注重研究中国老龄化趋势，并预测我国何时进入超老龄化社会（65 岁人口占比超过 20%）。

针对问题，本文考虑了人口增长率、出生率、单岁年龄别死亡率以及每年平均死亡率，我们首先基于 logistic 模型计算每年的出生人口，其次采用队列要素模型对每年的人口年龄结构变化预测，并借助 Python 程序进行模型计算与推演，预测较为准确的我国超老龄化的时间。

优化结果及总结：在现有模型的基础上，可以考虑引入动态死亡率和人口迁移因素，以提高模型的预测精度。本模型预测 2032-2033 年进入超老龄化社会，为应对政策提供更可靠参考。

关键词：中国老龄化；趋势分析；队列要素模型；Logistic 模型；Python 程序

目 录

一、问题的提出 1
二、 问题分析 1
三、 模型假设 2
四、 定义与符号说明 2
五、 模型的建立与求解 3-9
5.1 模型的建立 5-6
5.2 参数的确定与模型的求解 6-9
5.3 结论 9
六、 模型的评价及优化 9-10
6.1 误差分析 9
6.2 模型的优点 9-10
6.3 模型的缺点 10
6.4 模型的推广 10
参考文献 11
附 录 12-16

一、问题的提出

当今社会，中国老龄化已经成为一个热门话题，2021 中国已进入中度老龄化社会，老龄化规模大、速度快、未富先老特征明显。老龄化进程不断加快，从 2000 年进入老龄化社会，到 2021 年迈入中度老龄化社会，仅仅用了 21 年。截至 2024 年末，我国 65 岁及以上人口 2.2 亿人，占全国人口的 15.64% ，且这一比例还在持续攀升(2021 年-2024 年我国人口年龄结构如下图所示)。[1]按照目前的趋势，究竟我国何时会正式踏入超老龄化社会？这一关键节点不仅关系到国家养老、医疗、社会保障等政策的制定与调整，也与每一个人的生活息息相关。未来，我们将如何应对超老龄化社会带来的种种挑战，又该如何挖掘其中的发展机遇，成为全社会亟待思考与探索的重要课题。

图 1-1 2021 年中国人口年龄结构

图 1-2 2022 年中国人口年龄结构

图 1-3 2023 年中国人口年龄结构

图 1-4 2024 年中国人口年龄结构

二、问题分析

此问题属于预测类问题，通过查询相关数据，小组讨论，我们认为需要考虑以下几个因素：

第一，65 岁及以上人口占比超过 20%，就算步入超老龄化社会每隔一年每个没有死亡的人年龄加一，出生人口算在 0 岁。

第二，第 2 年 65 岁及以上人口占比与第 2 年出生率、死亡率有关，出生率、总死亡率决定第 2 年的总人口，单岁年龄别死亡率影响第 2 年人口削减情况，进

而导致人口年龄结构的变化。

第三，在社会相对安全稳定的情况下，人口的增长近似符合 logistic 模型。

由于以上原因，我们初步建立以下研究思路：

1.仪式队列要素法，可以通过每一岁的人口数以及死亡率的初始数据对接下来每一年的单岁人口情况进行预测；

2.Logistic 模型，可以通过人口增长率、出生率、死亡率之间的关系，并结合 logistic 模型公式，对第二年出生人口情况进行预测；

3.Python 程序，可以通过 Python 算法，代入各项初始数据，方便精准地运算，实现对未来 65 岁及以上人口数量占比的预测。

三、模型假设

通过对问题的合理探究分析，我们进行如下假设：

1.假设国家统计局、联合国人口司所给数据能够较好吻合当时人口状况；

2.在社会稳定的情况下，单岁年龄别死亡率不随时间变化；

3.不考虑国家人口迁入迁出等外部因素；

4.以 2023 年人口数据作为零点，假设 2024 年的人口变化为未来人口趋势。

四、定义与符号说明

符号定义	符号说明
x	第一年年末人口总数
C_n	n 岁年龄别人口（n 为非负整数）
S_n	n 岁年龄别死亡率（n 为非负整数）
C	第二年出生率，等于出生人口除于出生前总人口
S	第二年总死亡率，等于死亡人口除于死亡前总人口
r（x）	表示随总人口数变化的人口增长率
r	固有增长率（常数），即在没有外部因素影响下，每年人口自然增长的速度。
N	我国人口容量
C_i	$C_i=\{C_1，C_2，C_3，……，C_n\}$
S_i	$S_i=\{S_1，S_2,……,S_n\}$

五、模型的建立与求解

模型建立前首先进行数据预处理，检查数据完整性，发现国家统计局的数据不完整，而联合国人口司的预测数据完整但与国家统计局的数据存在差异，因此，我们通过比例放缩与分组调整，对联合国人口司的数据进行合理的预测与调整，使其尽量与国家统计局的数据吻合，从而得到更加完善的数据，以下是经处理的、来自国家统计局与联合国人口司的2023年人口数以及2024年死亡率的各个数据（通过微调数据操作，控制2023年人口数为1409670000人，出生率约为0.64%，65岁及以上人口占比约为15.4%）：

表5-1 2023年人口年龄与2024年死亡率统计表

年龄（岁）	人口（人）	死亡率	年龄（岁）	人口数（人）	死亡率
0	9021083	0.006415	51	23657693	0.003785
1	9225085	0.000545	52	24526563	0.004182
2	10321754	0.000426	53	24705752	0.004513
3	12233284	0.000349	54	24445911	0.004776
4	13845586	0.000289	55	23428155	0.00503
5	16132850	0.000262	56	22142851	0.00533
6	17548171	0.000264	57	22848059	0.005732
7	17172929	0.000284	58	22848719	0.006308
8	17361209	0.000308	59	23779923	0.007111
9	17714086	0.000324	60	23698553	0.008112
10	18618866	0.000324	61	16594014	0.009276
11	18359236	0.000322	62	12633396	0.010549
12	17668564	0.000329	63	12982278	0.011815
13	17740919	0.000334	64	14450861	0.01301

14	17602737	0.000339	65	15955384	0.014118
15	17052652	0.000347	66	16046745	0.015279
16	16587779	0.000363	67	15454878	0.016526
17	16247852	0.000386	68	15458550	0.017633
18	15998634	0.00042	69	14595772	0.019575
19	15754656	0.000462	70	14641069	0.022799
20	13698870	0.000512	71	13641300	0.025194
21	13797842	0.000565	72	12305561	0.027252
22	14235469	0.000615	73	11130237	0.029994
23	14295927	0.000657	74	9670213	0.033242
24	14085481	0.000688	75	9238786	0.036669
25	15129374	0.000708	76	8303390	0.040592
26	15553451	0.000729	77	7417023	0.045136
27	16065366	0.000748	78	6597512	0.050407
28	16528111	0.000762	79	5921159	0.056785
29	16981000	0.000775	80	5383055	0.064363
30	18257267	0.000794	81	5000078	0.07291
31	19283382	0.000811	82	4545204	0.082352
32	22344414	0.000825	83	3955081	0.092549
33	24326720	0.000837	84	3490126	0.103203
34	23540670	0.000852	85	3385147	0.114593
35	24511588	0.000877	86	2953996	0.126927

36	24277066	0.000915	87	2546775	0.140083
37	22660067	0.000971	88	2140003	0.154497
38	21175714	0.001052	89	1759641	0.171939
39	20001988	0.001156	90	1527947	0.191852
40	20231555	0.001284	91	1145319	0.214176
41	20518534	0.001428	92	871473	0.243782
42	19242791	0.001579	93	645186	0.277476
43	18554050	0.001729	94	456174	0.310034
44	17820247	0.001879	95	344513	0.34109
45	17487289	0.002034	96	220637	0.372534
46	18113873	0.0022	97	141828	0.408289
47	19147401	0.002388	98	89798	0.44531
48	20485968	0.002631	99	51816	0.479667
49	22188690	0.002958	100 及以上	57805	0.514785
50	23089994	0.003355	-	-	-

5.1 模型的建立

推导预测第二年出生人口，Logistic 阻滞增长模型是荷兰数学家 Verhulst 在 19 世纪中叶提出的，它考虑到自然资源、环境条件等因素对人口增长的阻滞作用，并且随着人口的增加，阻滞作用越来越大，所谓阻滞增长模型就是对指数增长模型的基本假设进行修改后得到的。阻滞作用体现在对人口增长率的影响上，使得 r(x) 随着人口数量 N 的增长而下降。[3]有:

$$x'(t)=r(x)x^{[3]}$$

其中，第二年人口增长率 r(x)有：

$$r(x)=C-S=r(1-\frac{x}{N})^{[3]}$$

可得 $C=r(1-\frac{x}{N})+S$ ①

由出生率死亡率的定义可得 $C_0=C\,x$ ②

$$S=\frac{\sum_{i=0}^{n} S_i C_i}{\sum_{i=0}^{n} C_i} \quad ③$$

将③代入①中,得： $C=r(1-\frac{x}{N})+\frac{C_0 S_0+\sum_{i=1}^{n} C_i S_i}{C_0+x}$,

注：此时年份增加，岁数增加，原来的零岁人口变为了一岁人口，一岁人口变成了两岁人口以此类推，因此C_0变成了C_1，C_1变成了C_2……，式子中的$\sum_{i=1}^{n} C_i=x$

将②代入上式得： $\frac{S_0 C_0+\sum_{i=1}^{n} S_i C_i}{C_0+x}+r(1-\frac{x}{N})=\frac{C_0}{x}$ ④

进一步整理：

$$\frac{C_0^2}{x}+[1-S_0-r(1-\frac{x}{N})]C_0-[\sum_{i=1}^{n} C_i S_i+rx(1-\frac{x}{N})]=0 \quad ⑤$$

设 $b=1-S_0-r(1-\frac{x}{N})$,

$$c=-[\sum_{i=1}^{n} S_i C_i+rx(1-\frac{x}{N})]$$

则$\frac{C_0^2}{x}+bC_0+c=0$ ⑥

解得 $C_0=\frac{x(\sqrt{b^2-\frac{4c}{x}}-b)}{2}$⑦

注：由于出生人口为正数，$C_0=\frac{-x(\sqrt{b^2-\frac{4c}{x}}+b)}{2}$为负数，明显不符合实际，故舍去

故第二年出生人口经计算得约为$\frac{x(\sqrt{b^2-\frac{4c}{x}}-b)}{2}$

5.2 参数的确定与模型的求解

首先确定参数：

人口容量 N：袁建华等的估算表明[4]，在充分有效利用科技进步来增加粮食产量的条件下，在 21 世纪我国粮食最大可能的生产量是 80000 万吨，如果分别按每人需要粮食 500 公斤我国的最大人口容量为 16 亿。[5]

1986 年中国科学院自然资源综合考察委员会研究认为,中国粮食最大承载力为 16.6 亿和 15.1 亿人口[6]

综合考虑，最大人口数在 16 亿上下浮动，故我们取 N=1600000000。

固有增长率 r：考虑到目前人口未到 N,却出现负增长得到情况，判断当前人口情况出现较大变化，故在排除新冠疫情年份（2020,2021,2022）后，选择 2023~2024 年的人口变化，对固有增长率 r 进行估测，方法如下：

对 2023 年末的每岁人口进行岁数增长，得到新的每一岁的数据（不含 0 岁）：再对每一岁人口数按对应死亡率削减，并求和，得到总数为 1395911973 人，与 2024 年总人数相减得 2024 年 C_0 =12368027, 将其代入④中，并代入 x=1409670000,N=1600000000 解得：

$$r\approx -0.80446\%$$

在确定完参数后，我们开始对 Python 程序进行编写，我们希望此代码符合以下要求：

1.要每一岁的人口数编制成条形统计图，横轴为年龄（岁），刻度间隔为 1，纵轴为人口（人），刻度间隔为 500 万，同时计算年龄大于等于 65 岁的人数占总人数的比例，与今年的总人数和年份一起标记在条形统计图的旁边。

2.设计按钮功能：点击“下一年”，则开始进行计算，先将每组人口的岁数加一，接着按照⑦式，对第二年出生人口进行预测，然后将新的每岁人口数按照各自的死亡率进行削减，进行向下取整，若某一组人口小于一人，则删除那组数据，随后更新数据，按新的数据生成条形统计图，方法同上要求 1，历史数据保留，方便通过“上一年”按钮回溯。

结果：考虑到工程量、复杂度与实现效果，我们通过合理地借助 AI 工具，并经过多次调整与优化，最终得出较为完善 Python 代码，并绘制出以下条形统计图：

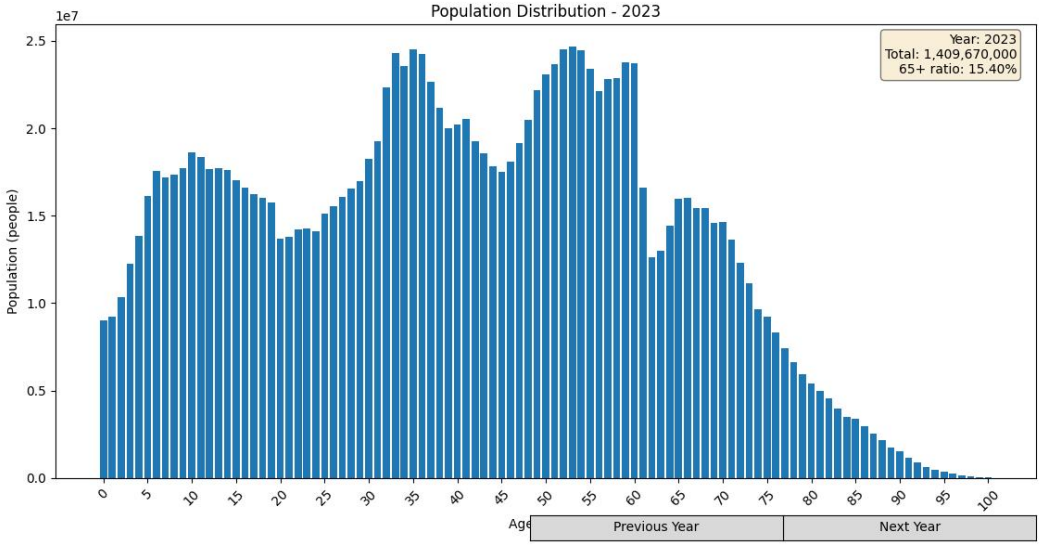

图 5-1 2023 年人口年龄结构

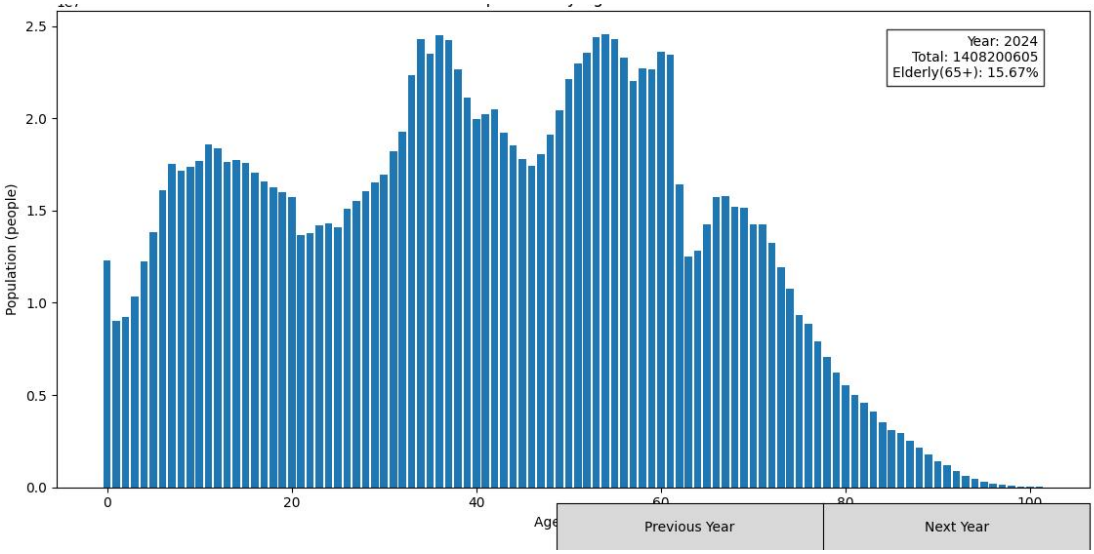

图 5-2 2024 年人口结构（模拟得出）

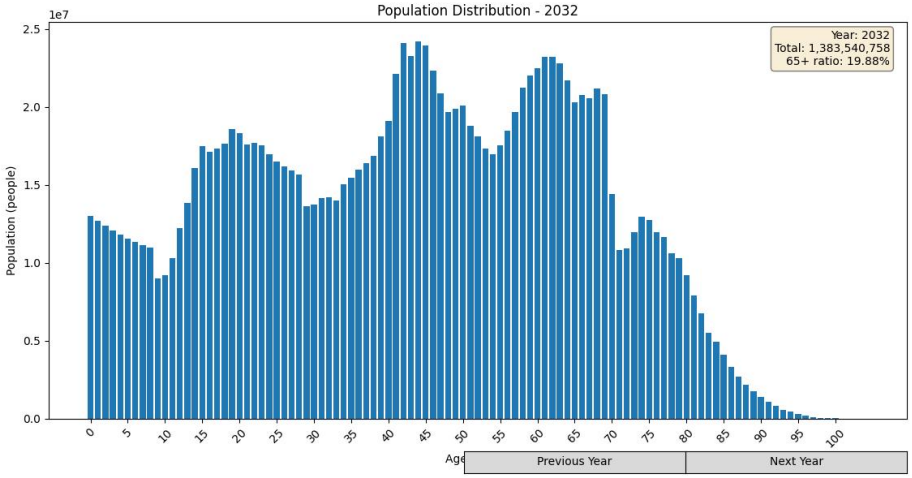

图 5-3 2032 年人口年龄结构（模拟得出）

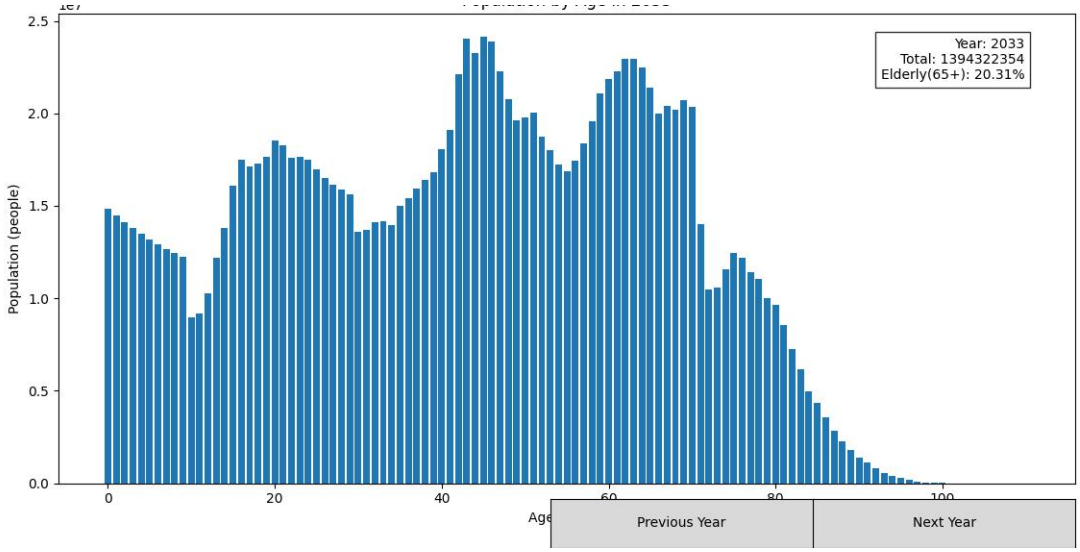

图 5-4 2033 年人口年龄结构（模拟得出）

5. 3 结论

由图 5-3 和图 5-4 预测结果可知，若按目前趋势，我国转变成超老龄化社会的时间在 2032~2033 年之间。

六、模型的评价及优化

将模型进行数值计算，并与附件中的真实采样值进行列表或图示比较。对误差进行数据分析，给出误差分析的理论估计。

6.1 误差分析

不难发现，2024 年的预测值相比实际值总人口仅仅相差 8 万人左右，误差约为$\frac{2}{35207}$,65 岁及以上人口占比相差约 0.03%，出现误差的原因，可能是在数据预处理时，准确性不够高，并且 2023 年的 65 岁及以上人口取 15.4%导致（真实值约 15.37664%）。而且本文中假设单岁年龄死亡率保持不变，与现实情况存在差别。

6.2 模型的优点

采用 Logistic 阻滞增长模型，考虑了自然资源、环境条件等对人口增长的阻滞作用，符合人口增长的实际情况；结合队列要素法和 Python 程序，提高了预测的便捷性和精准性。在社会稳定、单岁年龄别死亡率不变、不考虑人口迁移等

方面做出假设，使研究在可控范围内进行，便于模型的建立和推演。通过 Python 程序模拟得出 2032-2033 年我国将进入超老龄化社会的结论，并有相应的图表支撑，直观易懂。

6.3 模型的缺点

2023 年 65 岁及以上人口占比取值与真实值存在细微差异，会对结果产生影响。假设单岁年龄别死亡率不随时间变化，而现实中随着医疗水平的进步，死亡率可能会下降，这会导致预测结果不够准确；不考虑人口迁入迁出等外部因素，也与实际人口变化情况有出入。固有增长率 r 仅基于 2023-2024 年的人口变化（排除新冠疫情年份）进行估测，可能无法完全反映长期的人口增长趋势。尽管如此，该模型对短期时间的预测，还是较为准确的，通过不断优化和引入更多实际数据，这些缺点可以逐步得到改善。

6.4 模型的推广

该模型可推广到对我国不同省份、城市的人口老龄化趋势进行预测。只需将模型中的全国人口数据替换为相应区域的人口数据，同时调整人口容量等参数，就能为各区域制定针对性的老龄化应对政策提供参考。除了预测超老龄化社会的时间，该模型还可用于预测未来出生人口数量、各年龄段人口占比变化等。通过调整模型中的相关参数和计算指标，可实现对不同人口问题的研究。

参考文献

[1] 国家统计局. 年度数据[DB/OL].[2022 - 2024]. https://data.stats.gov.cn/。

[2] 联合国人口司. World Population Prospects: The 2024 Revision[DB/OL]. 2024[2024-07-20]. https://population.un.org/wpp/

[3] 姜启源，谢金星，叶俊.数学模型(第三版)[M].北京：高等教育出版社，2003.12-13

[4] 袁建华，许屹，姜涛，实施可持续发展战略：面向 21 世纪的我国人口控制对策研究［A］. 周生贤，主编.中国人口与环境（第四辑）［C］北京:中国环境科学出版社②1998

[5] 陈卫，孟向京. 中国人口容量与适度人口问题研究[J]. 市场与人口分析, 2000, 6(1):21-31.

[6] 陈百明.“中国土地资源生产能力及人口承载量”项目研究方法概论 [J]. 自然资源学报, 1991, (03): 197-205.

附 录

代码如下：

```
import matplotlib.pyplot as plt
from matplotlib.widgets import Button
import math
import numpy as np

data_a = dict([(0, 9021083), (1, 9225085), (2, 10321754), (3, 12233284), (4, 13845586), (5, 16132850),
               (6, 17548171), (7, 17172929), (8, 17361209), (9, 17714086), (10, 18618866), (11, 18359236),
               (12, 17668564), (13, 17740919), (14, 17602737), (15, 17052652), (16, 16587779), (17, 16247852),
               (18, 15998634), (19, 15754656), (20, 13698870), (21, 13797842), (22, 14235469), (23, 14295927),
               (24, 14085481), (25, 15129374), (26, 15553451), (27, 16065366), (28, 16528111), (29, 16981000),
               (30, 18257267), (31, 19283382), (32, 22344414), (33, 24326720), (34, 23540670), (35, 24511588),
               (36, 24277066), (37, 22660067), (38, 21175714), (39, 20001988), (40, 20231555), (41, 20518534),
               (42, 19242791), (43, 18554050), (44, 17820247), (45, 17487289), (46, 18113873), (47, 19147401),
               (48, 20485968), (49, 22188690), (50, 23089994), (51, 23657693), (52, 24526563), (53, 24705752),
               (54, 24445911), (55, 23428155), (56, 22142851), (57, 22848059), (58, 22848719), (59, 23779923),
               (60, 23698553), (61, 16594014), (62, 12633396), (63, 12982278), (64, 14450861), (65, 15955384),
               (66, 16046745), (67, 15454878), (68, 15458550), (69, 14595772), (70, 14641069), (71, 13641300),
               (72, 12305561), (73, 11130237), (74, 9670213), (75, 9238786), (76, 8303390), (77, 7417023),
               (78, 6597512), (79, 5921159), (80, 5383055), (81, 5000078), (82, 4545204), (83, 3955081),
               (84, 3490126), (85, 3385147), (86, 2953996), (87, 2546775), (88, 2140003), (89, 1759641),
               (90, 1527947), (91, 1145319), (92, 871473), (93, 645186), (94, 456174), (95, 344513),
               (96, 220637), (97, 141828), (98, 89798), (99, 51816), (100, 57805)])
```

```
data_b = dict([(0, 0.006415), (1, 0.000545), (2, 0.000426), (3, 0.000349), (4, 0.000289), (5, 0.000262),
               (6, 0.000264), (7, 0.000284), (8, 0.000308), (9, 0.000324), (10, 0.000324), (11, 0.000322),
               (12, 0.000329), (13, 0.000334), (14, 0.000339), (15, 0.000347), (16, 0.000363), (17, 0.000386),
               (18, 0.00042), (19, 0.000462), (20, 0.000512), (21, 0.000565), (22, 0.000615), (23, 0.000657),
               (24, 0.000688), (25, 0.000708), (26, 0.000729), (27, 0.000748), (28, 0.000762), (29, 0.000775),
               (30, 0.000794), (31, 0.000811), (32, 0.000825), (33, 0.000837), (34, 0.000852), (35, 0.000877),
               (36, 0.000915), (37, 0.000971), (38, 0.001052), (39, 0.001156), (40, 0.001284), (41, 0.001428),
               (42, 0.001579), (43, 0.001729), (44, 0.001879), (45, 0.002034), (46, 0.0022), (47, 0.002388),
               (48, 0.002631), (49, 0.002958), (50, 0.003355), (51, 0.003785), (52, 0.004182), (53, 0.004513),
               (54, 0.004776), (55, 0.00503), (56, 0.00533), (57, 0.005732), (58, 0.006308), (59, 0.007111),
               (60, 0.008112), (61, 0.009276), (62, 0.010549), (63, 0.011815), (64, 0.01301), (65, 0.014118),
               (66, 0.015279), (67, 0.016526), (68, 0.017633), (69, 0.019575), (70, 0.022799), (71, 0.025194),
               (72, 0.027252), (73, 0.029994), (74, 0.033242), (75, 0.036669), (76, 0.040592), (77, 0.045136),
               (78, 0.050407), (79, 0.056785), (80, 0.064363), (81, 0.07291), (82, 0.082352), (83, 0.092549),
               (84, 0.103203), (85, 0.114593), (86, 0.126927), (87, 0.140083), (88, 0.154497), (89, 0.171939),
               (90, 0.191852), (91, 0.214176), (92, 0.243782), (93, 0.277476), (94, 0.310034), (95, 0.34109),
               (96, 0.372534), (97, 0.408289), (98, 0.44531), (99, 0.479667), (100, 0.514785)])

    current_year = 2023
    history = []
    N = 1600000000
    r = -0.0080446

    fig, ax = plt.subplots(figsize=(12, 8))
    plt.subplots_adjust(bottom=0.15)

    def plot_data():
```

```
        ax.clear()
        ages = sorted(data_a.keys())
        pops = [data_a[age] for age in ages]
        ax.bar(ages, pops, width=0.8)
        ax.set_xlabel('Age (years)')
        ax.set_ylabel('Population (people)')
        ax.set_title(f'Population by Age in {current_year}')
        total = sum(data_a.values())
        elderly = sum(pop for age, pop in data_a.items() if age >= 65)
        ratio = elderly / total * 100
        info_text = f'Year: {current_year}\nTotal: {total:.0f}\nElderly(65+): {ratio:.2f}%'
        ax.text(0.95, 0.95, info_text, transform=ax.transAxes, ha='right', va='top',
                bbox=dict(facecolor='white', alpha=0.8))
        plt.draw()

    def next_year(event):
        global current_year, data_a, history
        history.append((current_year, data_a.copy()))
        x = sum(data_a.values())

        new_a = {}
        z = 0
        for age, pop in data_a.items():
            new_age = age + 1
            new_a[new_age] = pop
            mort = data_b[100] if new_age > 100 else data_b.get(new_age, data_b[100])
            z += pop * mort

        d = 0.993585 - r * (1 - x/N)
        e_val = -(z + r * x * (1 - x/N))
        disc = d**2 - 4 * e_val / x
        c = max(0, x * (math.sqrt(max(0, disc)) - d) / 2) if disc >= 0 else 0
        new_a[0] = c

        next_a = {}
        for age, pop in new_a.items():
            mort = data_b[100] if age > 100 else data_b.get(age, data_b[100])
            survived = math.floor(pop * (1 - mort))
            if survived >= 1:
                next_a[age] = survived

        data_a = next_a
```

```
        current_year += 1
        plot_data()

def prev_year(event):
        global current_year, data_a, history
        if history:
                current_year, data_a = history.pop()
                plot_data()
```

```
plot_data()
ax_next = plt.axes([0.7, 0.05, 0.2, 0.075])
ax_prev = plt.axes([0.5, 0.05, 0.2, 0.075])
btn_next = Button(ax_next, 'Next Year')
btn_prev = Button(ax_prev, 'Previous Year')
btn_next.on_clicked(next_year)
btn_prev.on_clicked(prev_year)
plt.show()
```