

# EIOPA RISK-FREE CURVE JUNE-23 RECALCULATION

The risk-free curve is one of the principal inputs into an economic scenario generator. This notebook recalculates the risk-free curve using the parameters that are claimed to be used. The European Insurance and Occupational Pensions Authority (EIOPA) publishes their own yield curve prediction. To do this they use the Smith & Wilson algorithm.

## Summary

The goal of this test is to replicate the EIOPA yield curve. This test will use the methodology that EIOPA claims it is using and the calibration vector that they publish. If the test is passed, the user can be more confident, that EIOPA risk free rate (RFR) curve was generated using the described methodology/calibration and that the process was implemented correctly.

## Table of Contents

1. [Note on Smith & Wilson algorithm](#)
2. [Data requirements](#)
3. [Success criteria](#)
4. [External dependencies](#)
5. [Calibration parameters and calibration vector provided by EIOPA](#)
6. [Smith & Wilson calculation functions](#)
7. [Generation of the risk-free curve](#)
8. [Test 1; Comparison test](#)
9. [Test 1; Success criteria](#)
10. [Test 1; Comparison test](#)
11. [Conclusion](#)

## Note on Smith & Wilson algorithm

To replicate the calculations, this example uses a modified Smith&Wilson implementation (The original implementation is available on [GitHub](#):

- [Python](#)
- [Matlab](#)
- [JavaScript](#)

## Limitations of the implementation

Current implementation only looks at a single currency and with/without Volatility Adjustment (VA). The day count convention assumes that each year has the same number of days.

## Data requirements

This script contains the EIOPA risk-free rate publication for June 2023. The publication can be found on the [EIOPA RFR website](#).

The observed maturities `M_Obs` and the calibrated vector `Qb` can be found in the Excel sheet `EIOPA_RFR_20230630_Qb_SW.xlsx`.

The target maturities (`T_Obs`), the additional parameters (`UFR` and `alpha`), and the given curve can be found in the Excel `EIOPA_RFR_20230630_Term_Structures.xlsx`, sheet `RFR_spot_no_VA` if the test looks at the curve without the Volatility Adjustment and the sheet `RFR_spot_with_VA` if the test looks at the curve with the Volatility Adjustment.

[Back to the top](#)

## Success criteria

The following success criteria is defined:

- Maximum difference between the calculated curve and the one provided by EIOPA is less than 0.1 bps
- Average difference between the calculated curve and the one provided by EIOPA is less than 0.05 bps

In [126...]

```
test_statistics_max_diff_in_bps = 0.1
test_statistics_average_diff_in_bps = 0.05
```

The success function is called at the end of the test to confirm if the success criteria have been met.

In [127...]

```
def SuccessTest(TestStatistics, threshold_max, threshold_mean):
    out1 = False
    out2 = False
    if max(TestStatistics) < threshold_max:
        print("Test passed")
        out1 = True
    else:
        print("Test failed")

    if np.mean(TestStatistics) < threshold_mean:
        print("Test passed")
        out2 = True
    else:
        print("Test failed")
    return [out1, out2]
```

[Back to the top](#)

## External dependencies

This implementation uses three well established Python packages widely used in the financial industry. Pandas (<https://pandas.pydata.org/docs/>), Numpy (<https://numpy.org/doc/>), and Matplotlib (<https://matplotlib.org/stable/index.html>)

In [128...]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
%matplotlib notebook
pd.options.display.max_rows = 150
```

## Importing data

In [129...]

```
#selected_param_file = 'Param_VA.csv'
#selected_curves_file = 'Curves_VA.csv'

selected_param_file = 'Param_no_VA.csv'
selected_curves_file = 'Curves_no_VA.csv'
```

In [130...]

```
param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

## Parameter input

Parameters sheet

In [131...]

```
param_raw.head()
```

Out[131...]

	Euro_Maturities	Euro_Values	Austria_Maturities	Austria_Values	Belgium_Maturities	Be
Country						
<b>Coupon_freq</b>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
<b>LLP</b>	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
<b>Convergence</b>	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000
<b>UFR</b>	3.450000	3.450000	3.450000	3.450000	3.450000	3.450000
<b>alpha</b>	0.116339	0.116339	0.116339	0.116339	0.116339	0.116339

5 rows × 106 columns

The country selected is:

In [132...]

```
country = "Turkey"
```

In [133...]

```
maturities_country_raw = param_raw.loc[:,country+"_Maturities"].iloc[6:]
param_country_raw = param_raw.loc[:,country + "_Values"].iloc[6:]
extra_param = param_raw.loc[:,country + "_Values"].iloc[:6]
```

## Extra parameters

Smith-Wilson calibration parameters

In [134...]

```
extra_param
```

Out[134...]

Country	
Coupon_freq	0.000000
LLP	9.000000
Convergence	51.000000
UFR	5.500000
alpha	0.165319
CRA	10.000000
Name:	Turkey_Values, dtype: float64

In [135...]

```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [136...]

```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

## Maturity vector

Vector of maturities used in the calibration

In [137...]

```
maturities_country.head(15)
```

Out[137...]

```
Country
1    1.0
2    2.0
3    3.0
4    4.0
5    5.0
6    6.0
7    8.0
8    9.0
Name: Turkey_Maturities, dtype: float64
```

In [138...]

```
Qb = param_country_raw.iloc[relevant_positions]
```

## Calibration vector

Vector **Qb** provided as input

In [139...]

```
Qb
```

Out[139...]

```
Country
1    2.763976
2    1.871045
3   -0.692405
4   -0.523172
5   -0.844859
6   -0.179319
7   -0.136517
8   -0.156069
Name: Turkey_Values, dtype: float64
```

In [140...]

```
curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

In [141...]

```
curve_country = curve_raw.loc[:,country]
```

[Back to the top](#)

# Calibration parameters and calibration vector provided

## EIOPA

In [142...]

```
# Maturity of observations:  
M_Obs = np.transpose(np.array(maturities_country.values))  
  
# Ultimate forward rate ufr represents the rate to which the rate curve will converge  
ufr = extra_param.iloc[3]/100  
  
# Convergence speed parameter alpha controls the speed at which the curve converges to the target  
alpha = extra_param.iloc[4]  
  
# For which maturities do we want the SW algorithm to calculate the rates. In this case all  
M_Target = np.transpose(np.arange(1,151))  
  
# Qb calibration vector published by EIOPA for the curve calibration:  
Qb = np.transpose(np.array(Qb.values))
```

[Back to the top](#)

## Smith & Wilson calculation functions

In this step, the independent version of the Smith&Wilson algorithm is implemented. To do this, two functions are taken from the publicly available repository and modified to accept the product of  $Q^*b$  instead of the calibration vector  $b$ .

In [143...]

```

def SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha):
    # SWEXTRAPOLATE Interpolate or/and extrapolate rates for targeted maturities using a
    # out = SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha) calculates the rates for maturities
    #
    # Arguments:
    #     M_Target = k x 1 ndarray. Each element represents a bond maturity of interest. Ex. M_Target = [1; 3]
    #     M_Obs = n x 1 ndarray. Observed bond maturities used for calibrating the calibration vector.
    #     Qb = n x 1 ndarray. Calibration vector calculated on observed bonds.
    #     ufr = 1 x 1 floating number. Representing the ultimate forward rate.
    #         Ex. ufr = 0.042
    #     alpha = 1 x 1 floating number. Representing the convergence speed parameter alpha.
    #
    #
    # Returns:
    #     k x 1 ndarray. Represents the targeted rates for a zero-coupon bond. Each rate is
    #
    # For more information see https://www.eiopa.europa.eu/sites/default/files/risk_free_curve.ipynb

def SWHeart(u, v, alpha):
    # SWHEART Calculate the heart of the Wilson function.
    # H = SWHeart(u, v, alpha) calculates the matrix H (Heart of the Wilson function) for maturities specified by vectors u and v. The formula is
    # taken from the EIOPA technical specifications paragraph 132.
    #
    # Arguments:
    #     u = n_1 x 1 vector of maturities. Ex. u = [1; 3]
    #     v = n_2 x 1 vector of maturities. Ex. v = [1; 2; 3; 5]
    #     alpha = 1 x 1 floating number representing the convergence speed parameter alpha.
    #
    # Returns:
    #     n_1 x n_2 matrix representing the Heart of the Wilson function for selected maturities.
    #
    # For more information see https://www.eiopa.europa.eu/sites/default/files/risk_free_curve.ipynb

    u_Mat = np.tile(u, [v.size, 1]).transpose()
    v_Mat = np.tile(v, [u.size, 1])
    return 0.5 * (alpha * (u_Mat + v_Mat) + np.exp(-alpha * (u_Mat + v_Mat)) - alpha)

H = SWHeart(M_Target, M_Obs, alpha) # Heart of the Wilson function from paragraph 132
p = np.exp(-np.log(1+ufr)* M_Target) + np.diag(np.exp(-np.log(1+ufr) * M_Target))
return p ** (-1/ M_Target) -1 # Convert obtained prices to rates and return price

```

[Back to the top](#)

## Generation of the risk-free curve

The observed maturities, target maturities, and the model parameters provided by EIOPA are used to generate the target curve.

In [144...]

```

r_Target = SWExtrapolate(M_Target,M_Obs, Qb, ufr, alpha)
r_Target = pd.DataFrame(r_Target,columns=['Recalculated rates'])

```

### Yield curve calculated

Yield curve calculated using the calibration vector **Qb**

In [145...]

```
r_Target.head(15)
```

Out[145...]

#### Recalculated rates

0	0.128955
1	0.137800
2	0.148035
3	0.157283
4	0.164767
5	0.170364
6	0.174457
7	0.177373
8	0.179220
9	0.180106
10	0.180217
11	0.179713
12	0.178709
13	0.177292
14	0.175536

---

[Back to the top](#)

## Test 1; Comparison test

Comparison of the calculated yield curve with the yield curve provided by EIOPA. The test is passed if the success criteria is reached.

The provided yield curve can be found in file *EIOPA\_RFR\_20230630\_Term\_Structures.xlsx*, sheet *RFR\_spot\_no\_VA* if the test looks at the curve without the Volatility Adjustment and the sheet *RFR\_spot\_with\_VA* if the test looks at the curve with the Volatility Adjustment.

In [146...]

```
target_curve = np.transpose(np.array(curve_country.values))
```

This implementation looks at two kinds of test statistics. The average deviation and the maximum deviation.

The average deviation is defined as:

$$S_{AVERAGE} = \frac{1}{T} \sum_{t=0}^T |r_{EIOPA}(t) - r_{EST}(t)|$$

The maximum deviation is defined as:

$$S_{MAX} = \max_t |r_{EIOPA}(t) - r_{EST}(t)|$$

Where  $T$  is the maximum maturity available.

The average difference test is successful if:

$$S_{AVERAGE} < 0.05bps$$

The maximum difference test is successful if:

$$S_{MAX} < 0.1bps$$

In [147...]

```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

### EIOPA curve provided

Yield curve provided by EIOPA

In [148...]

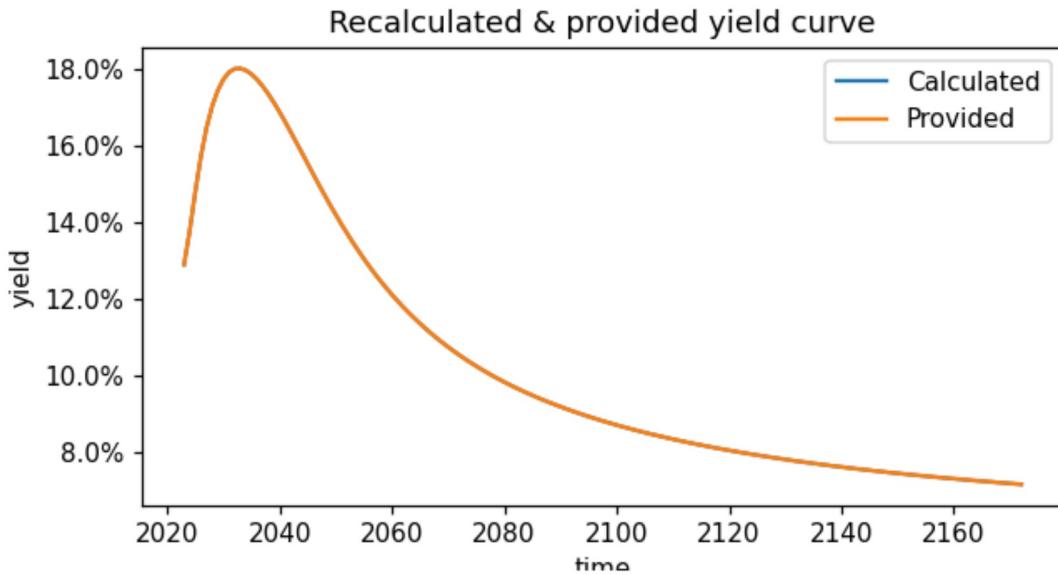
```
target_curve.head()
```

Out[148...]

	Given rates
0	0.12896
1	0.13780
2	0.14803
3	0.15728
4	0.16477

```
In [149...]:  
x_data_label = range(2023,2023+r_Target.shape[0],1)
```

```
In [150...]:  
fig, ax1 = plt.subplots(1,1)  
ax1.plot(x_data_label, r_Target.values*100, color='tab:blue',label="Calculated")  
ax1.plot(x_data_label, target_curve.values*100, color='tab:orange',label="Provided")  
  
ax1.set_ylabel("yield")  
ax1.set_title('Recalculated & provided yield curve')  
ax1.set_xlabel("time")  
ax1.legend()  
ax1.yaxis.set_major_formatter(mtick.PercentFormatter())  
fig.set_figwidth(6)  
fig.set_figheight(3)  
plt.show()
```



```
In [151...]:  
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

### EIOPA curve comparison

Absolute difference in bps

```
In [152...]:  
test_statistics_bdp.head()
```

```
Out[152...]:  
Abs diff in bps
```

	Abs diff in bps
0	0.047840
1	0.004590
2	0.048918
3	0.028604

Abs diff in bps

[Back to the top](#)

## Test 1; Success criteria

The successful application of the success criteria marks the completion of the test.

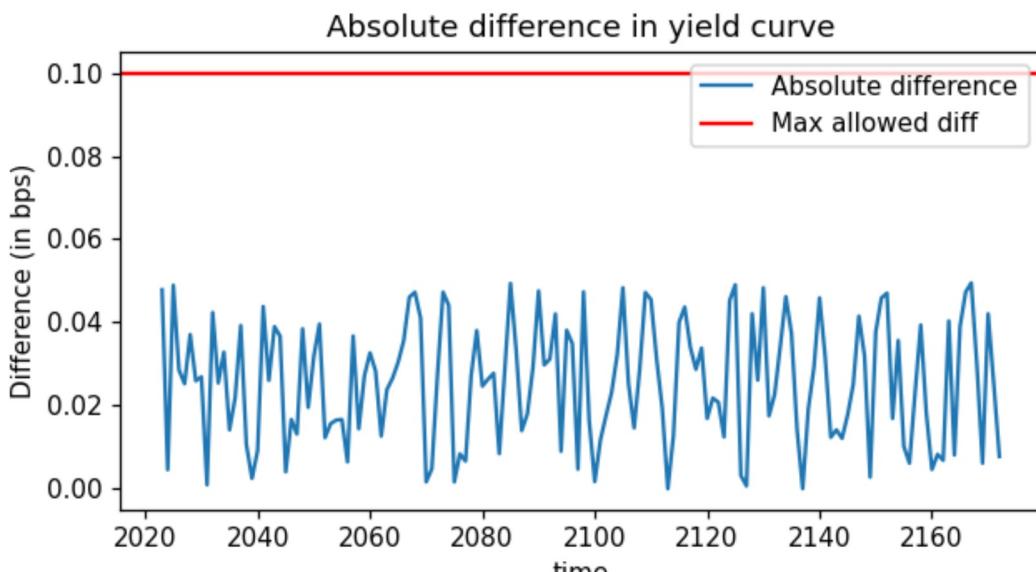
In [153...]

```
result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, tolerance)
```

```
Test passed  
Test passed
```

In [154...]

```
x_data_label = range(2023,2023+r_Target.shape[0],1)  
fig, ax1 = plt.subplots(1,1)  
ax1.plot(x_data_label, test_statistics_bdp, label= "Absolute difference")  
ax1.axhline(y = test_statistics_max_diff_in_bps, color = 'r', linestyle = '--',label='Max allowed diff')  
  
ax1.set_xlabel("time")  
ax1.set_ylabel("Difference (in bps)")  
ax1.set_title('Absolute difference in yield curve')  
ax1.legend()  
fig.set_figwidth(6)  
fig.set_figheight(3)  
  
plt.show()
```



[Back to the top](#)

## Conclusion

This test checks the success criteria on the EIOPA curve generated for June 2023. If the tests are passed, it is likely that the curve was generated using the Smith & Wilson algorithm with the calibration vector that was provided in the file *EIOPA\_RFR\_20230630\_Qb\_SW.xlsx* and the parameters displayed in the file *EIOPA\_RFR\_20230630\_Term\_Structures.xlsx*.

In [155...]

```
pd.DataFrame(data = [result1], columns = ["Mean test", "Max test"], \
              index= ["Provided vs calculated"])
```

Out[155...]

	Mean test	Max test
--	-----------	----------

Provided vs calculated	True	True
------------------------	------	------

### Final yield curve

Full yield curve provided by EIOPA in %

In [156...]

```
(curve_country*100).head(150)
```

Out[156...]

Country	
1	12.896
2	13.780
3	14.803
4	15.728
5	16.477
6	17.036
7	17.446
8	17.737
9	17.922
10	18.011
11	18.022
12	17.971
13	17.871
14	17.729
15	17.554
16	17.350
17	17.124
18	16.880
19	16.623
20	16.355
21	16.082
22	15.805
23	15.527
24	15.251
25	14.978
26	14.710
27	14.449
28	14.194

29	13.948
30	13.709
31	13.479
32	13.258
33	13.045
34	12.841
35	12.645
36	12.458
37	12.278
38	12.106
39	11.942
40	11.784
41	11.633
42	11.489
43	11.350
44	11.218
45	11.090
46	10.968
47	10.851
48	10.739
49	10.631
50	10.527
51	10.427
52	10.331
53	10.239
54	10.150
55	10.064
56	9.981
57	9.901
58	9.824
59	9.750
60	9.678
61	9.608
62	9.541
63	9.475
64	9.412
65	9.351
66	9.292
67	9.234
68	9.178
69	9.124
70	9.072
71	9.021
72	8.971
73	8.923
74	8.876
75	8.830
76	8.785
77	8.742
78	8.700
79	8.659
80	8.619
81	8.580
82	8.542
83	8.505
84	8.468
85	8.433
86	8.398
87	8.365

88	8.332
89	8.299
90	8.268
91	8.237
92	8.207
93	8.177
94	8.149
95	8.120
96	8.093
97	8.066
98	8.039
99	8.013
100	7.988
101	7.963
102	7.938
103	7.915
104	7.891
105	7.868
106	7.845
107	7.823
108	7.802
109	7.780
110	7.759
111	7.739
112	7.718
113	7.699
114	7.679
115	7.660
116	7.641
117	7.623
118	7.605
119	7.587
120	7.569
121	7.552
122	7.535
123	7.518
124	7.502
125	7.486
126	7.470
127	7.454
128	7.439
129	7.423
130	7.409
131	7.394
132	7.379
133	7.365
134	7.351
135	7.337
136	7.324
137	7.310
138	7.297
139	7.284
140	7.271
141	7.258
142	7.246
143	7.234
144	7.221
145	7.210
146	7.198

147	7.186
148	7.175
149	7.163
150	7.152
151	7.141
152	7.130
153	7.119
154	7.108
155	7.097
156	7.086
157	7.075
158	7.064
159	7.053
160	7.042
161	7.031
162	7.020
163	7.009
164	6.998
165	6.987
166	6.976
167	6.965
168	6.954
169	6.943
170	6.932
171	6.921
172	6.910
173	6.899
174	6.888
175	6.877
176	6.866
177	6.855
178	6.844
179	6.833
180	6.822
181	6.811
182	6.800
183	6.789
184	6.778
185	6.767
186	6.756
187	6.745
188	6.734
189	6.723
190	6.712
191	6.701
192	6.690
193	6.679
194	6.668
195	6.657
196	6.646
197	6.635
198	6.624
199	6.613
200	6.602
201	6.591
202	6.580
203	6.569
204	6.558
205	6.547
206	6.536
207	6.525
208	6.514
209	6.503
210	6.492
211	6.481
212	6.470
213	6.459
214	6.448
215	6.437
216	6.426
217	6.415
218	6.404
219	6.393
220	6.382
221	6.371
222	6.360
223	6.349
224	6.338
225	6.327
226	6.316
227	6.305
228	6.294
229	6.283
230	6.272
231	6.261
232	6.250
233	6.239
234	6.228
235	6.217
236	6.206
237	6.195
238	6.184
239	6.173
240	6.162
241	6.151
242	6.140
243	6.129
244	6.118
245	6.107
246	6.096
247	6.085
248	6.074
249	6.063
250	6.052
251	6.041
252	6.030
253	6.019
254	6.008
255	5.997
256	5.986
257	5.975
258	5.964
259	5.953
260	5.942
261	5.931
262	5.920
263	5.909
264	5.898
265	5.887
266	5.876
267	5.865
268	5.854
269	5.843
270	5.832
271	5.821
272	5.810
273	5.800
274	5.789
275	5.778
276	5.767
277	5.756
278	5.745
279	5.734
280	5.723
281	5.712
282	5.701
283	5.690
284	5.679
285	5.668
286	5.657
287	5.646
288	5.635
289	5.624
290	5.613
291	5.602
292	5.591
293	5.580
294	5.569
295	5.558
296	5.547
297	5.536
298	5.525
299	5.514
300	5.503
301	5.492
302	5.481
303	5.470
304	5.459
305	5.448
306	5.437
307	5.426
308	5.415
309	5.404
310	5.393
311	5.382
312	5.371
313	5.360
314	5.349
315	5.338
316	5.327
317	5.316
318	5.305
319	5.294
320	5.283
321	5.272
322	5.261
323	5.250
324	5.239
325	5.228
326	5.217
327	5.206
328	5.195
329	5.184
330	5.173
331	5.162
332	5.151
333	5.140
334	5.129
335	5.118
336	5.107
337	5.096
338	5.085
339	5.074
340	5.063
341	5.052
342	5.041
343	5.030
344	5.019
345	5.008
346	5.000
347	4.991
348	4.982
349	4.973
350	4.964
351	4.955
352	4.946
353	4.937
354	4.928
355	4.919
356	4.910
357	4.901
358	4.892
359	4.883
360	4.874
361	4.865
362	4.856
363	4.847
364	4.838
365	4.829
366	4.820
367	4.811
368	4.802
369	4.793
370	4.784
371	4.775
372	4.766
373	4.757
374	4.748
375	4.739
376	4.730
377	4.721
378	4.712
379	4.703
380	4.694
381	4.685
382	4.676
383	4.667
384	4.658
385	4.649
386	4.640
387	4.631
388	4.622
389	4.613
390	4.604
391	4.595
392	4.586
393	4.577
394	4.568
395	4.559
396	4.550
397	4.541
398	4.532
399	4.523
400	4.514
401	4.505
402	4.496
403	4.487
404	4.478
405	4.469
406	4.460
407	4.451
408	4.442
409	4.433
410	4.424
411	4.415
412	4.406
413	4.397
414	4.388
415	4.379
416	4.370
417	4.361
418	4.352
419	4.343
420	4.334
421	4.325
422	4.316
423	4.307
424	4.298
425	4.289
426	4.280
427	4.271
428	4.262
429	4.253
430	4.244
431	4.235
432	4.226
433	4.217
434	4.208
435	4.199
436	4.190
437	4.181
438	4.172
439	4.163
440	4.154
441	4.145
442	4.136
443	4.127
444	4.118
445	4.109
446	4.100
447	4.091
448	4.082
449	4.073
450	4.064
451	4.055
452	4.046
453	4.037
454	4.028
455	4.019
456	4.010
457	4.001
458	3.992
459	3.983
460	3.974
461	3.965
462	3.956
463	3.947
464	3.938
465	3.929
466	3.920
467	3.911
468	3.902
469	3.893
470	3.884
471	3.875
472	3.866
473	3.857
474	3.848
475	3.839
476	3.830
477	3.821
478	3.812
479	3.803
480	3.794
481	3.785
482	3.776
483	3.767
484	3.758
485	3.749
486	3.740
487	3.731
488	3.722
489	3.713
490	3.704
491	3.695
492	3.686
493	3.677
494	3.668
495	3.659
496	3.650
497	3.641
498	3.632
499	3.623
500	3.614
501	3.605
502	3.596
503	3.587
504	3.578
505	3.569
506	3.560
507	3.551
508	3.542
509	3.533
510	3.524
511	3.515
512	3.506
513	3.497
514	3.488
515	3.479
516	3.470
517	3.461
518	3.452
519	3.443
520	3.434
521	3.425
522	3.416
523	3.407
524	3.398
525	3.389
526	3.380
527	3.371
528	3.362
529	3.353
530	3.344
531	3.335
532	3.326
533	3.317
534	3.308
535	3.299
536	3.290
537	3.281
538	3.272
539	3.263
540	3.254
541	3.245
542	3.236
543	3.227
544	3.218
545	3.209
546	3.200
547	3.191
548	3.182
549	3.173
550	3.164
551	3.155
552	3.146
553	3.137
554	3.128
555	3.119
556	3.110
557	3.101
558	3.092
559	3.083
560	3.074
561	3.065
562	3.056
563	3.047
564	3.038
565	3.029
566	3.020
567	3.011
568	3.002
569	2.993
570	2.984
571	2.975
572	2.966
573	2.957
574	2.948
575	2.939
576	2.930
577	2.921
578	2.912
579	2.903
580	2.894
581	2.885
582	2.876
583	2.867
584	2.858
585	2.849
586	2.840
587	2.831
588	2.822
589	2.813
590	2.804
591	2.795
592	2.786
593	2.777
594	2.768
595	2.759
596	2.750
597	2.741
598	2.732
599	2.723
600	2.714
601	2.705
602	2.696
603	2.687
604	2.678
605	2.669
606	2.660
607	2.651
608	2.642
609	2.633
610	2.624
611	2.615
612	2.606
613	2.597
614	2.588
615	2.579
616	2.570
617	2.561
618	2.552
619	2.543
620	2.534
621	2.525
622	2.516
623	2.507
624	2.498
625	2.489
626	2.480
627	2.471
628	2.462
629	2.453
630	2.444
631	2.435
632	2.426
633	2.417
634	2.408
635	2.399
636	2.390
637	2.381
638	2.372
639	2.363
640	2.354
641	2.345
642	2.336
643	2.327
644	2.318
645	2.309
646	2.300
647	2.291
648	2.282
649	2.273
650	2.264
651	2.255
652	2.246
653	2.237
654	2.228
655</td	