

EIOPA RISK-FREE CURVE JULY-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 July 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_20230731_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_20230731_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 [218...]

```
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 [219...]

```
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 [220...]

```
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 [221...]

```
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 [222...]

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

Parameter input

Parameters sheet

In [223...]

```
param_raw.head()
```

Out[223...]

Country	Euro_Maturities	Euro_Values	Austria_Maturities	Austria_Values	Belgium_Maturities	Be
Coupon_freq	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
LLP	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
Convergence	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000
UFR	3.450000	3.450000	3.450000	3.450000	3.450000	3.450000
alpha	0.108242	0.108242	0.108242	0.108242	0.108242	0.108242

5 rows × 106 columns

The country selected is:

In [224...]

```
country = "Turkey"
```

In [225...]

```
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 [226...]

```
extra_param
```

Out[226...]

```
Country
Coupon_freq      0.000000
LLP              9.000000
Convergence      51.000000
UFR              5.500000
alpha             0.165581
CRA               10.000000
Name: Turkey_Values, dtype: float64
```

In [227...]

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

In [228...]

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

Maturity vector

Vector of maturities used in the calibration

In [229...]

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

Out[229...]

```
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 [230...]

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

Calibration vector

Vector **Qb** provided as input

In [231...]

```
Qb
```

Out[231...]

```
Country
1    -2.594118
2     10.888107
3    -4.804714
4     0.023312
5    -2.015106
6    -0.304049
7    -0.981962
8     1.036681
Name: Turkey_Values, dtype: float64
```

In [232...]

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

In [233...]

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

[Back to the top](#)

Calibration parameters and calibration vector provided

EIOPA

In [234...]

```
# 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 [235...]

```

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 of the
    #
    #
    # 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 of the
    #
    # Returns:
    #     n_1 x n_2 matrix representing the Heart of the Wilson function for selected
    #
    # 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 [236...]

```

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 [237...]

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

Out[237...]

Recalculated rates

0	0.139454
1	0.149675
2	0.166894
3	0.182276
4	0.194183
5	0.201892
6	0.205980
7	0.207159
8	0.206124
9	0.204380
10	0.202331
11	0.200018
12	0.197477
13	0.194739
14	0.191832

[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_20230731_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 [238...]

```
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 [239...]

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

EIOPA curve provided

Yield curve provided by EIOPA

In [240...]

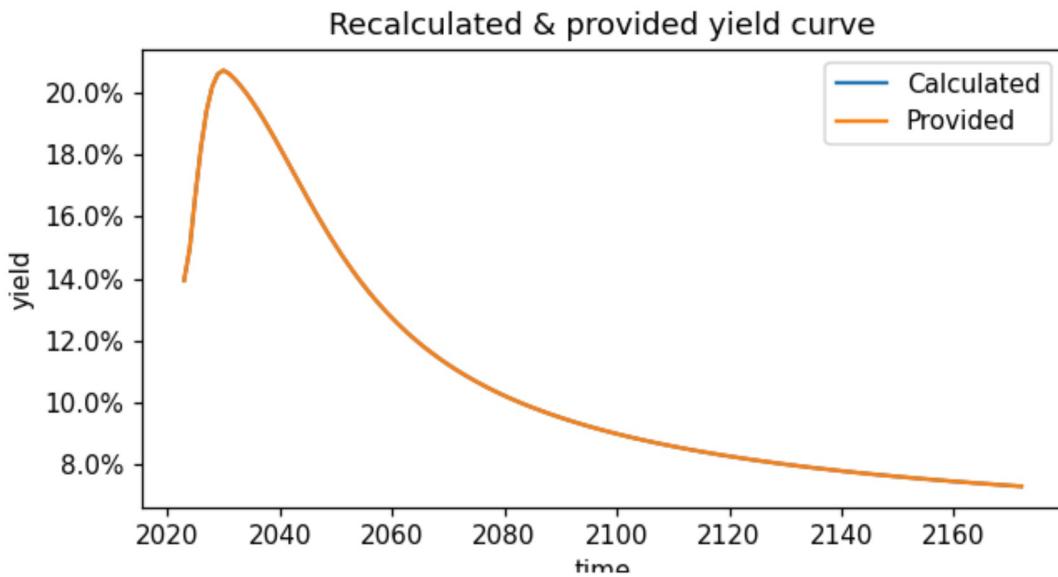
```
target_curve.head()
```

Out[240...]

	Given rates
0	0.13945
1	0.14968
2	0.16689
3	0.18228
4	0.19418

```
In [241...]:  
x_data_label = range(2023, 2023+r_Target.shape[0], 1)
```

```
In [242...]:  
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 [243...]:  
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

```
In [244...]:  
test_statistics_bdp.head()
```

```
Out[244...]:  
Abs diff in bps
```

0	0.040392
1	0.049192
2	0.044102
3	0.037470

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.

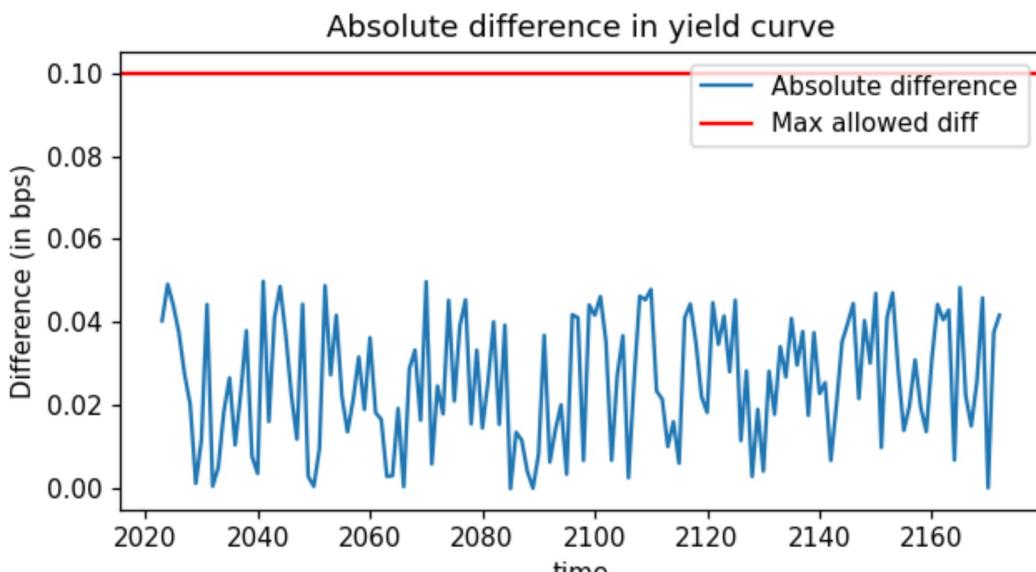
In [245...]

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

```
Test passed  
Test passed
```

In [246...]

```
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 July 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_20230731_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230731_Term_Structures.xlsx*.

In [247...]

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

Out[247...]

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [248...]

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

Out[248...]

Country	
1	13.945
2	14.968
3	16.689
4	18.228
5	19.418
6	20.189
7	20.598
8	20.716
9	20.612
10	20.438
11	20.233
12	20.002
13	19.748
14	19.474
15	19.183
16	18.879
17	18.563
18	18.239
19	17.909
20	17.577
21	17.244
22	16.913
23	16.586
24	16.265
25	15.950
26	15.644
27	15.346
28	15.058

29	14.780
30	14.512
31	14.255
32	14.008
33	13.772
34	13.545
35	13.328
36	13.121
37	12.922
38	12.732
39	12.551
40	12.377
41	12.211
42	12.052
43	11.900
44	11.754
45	11.614
46	11.480
47	11.352
48	11.229
49	11.110
50	10.996
51	10.887
52	10.782
53	10.680
54	10.583
55	10.489
56	10.398
57	10.310
58	10.226
59	10.144
60	10.065
61	9.989
62	9.915
63	9.844
64	9.775
65	9.708
66	9.643
67	9.580
68	9.519
69	9.460
70	9.402
71	9.346
72	9.292
73	9.239
74	9.188
75	9.138
76	9.089
77	9.042
78	8.995
79	8.950
80	8.907
81	8.864
82	8.822
83	8.782
84	8.742
85	8.703
86	8.665
87	8.629

88	8.592
89	8.557
90	8.523
91	8.489
92	8.456
93	8.424
94	8.392
95	8.362
96	8.331
97	8.302
98	8.273
99	8.244
100	8.217
101	8.189
102	8.163
103	8.137
104	8.111
105	8.086
106	8.061
107	8.037
108	8.013
109	7.990
110	7.967
111	7.944
112	7.922
113	7.901
114	7.879
115	7.858
116	7.838
117	7.818
118	7.798
119	7.778
120	7.759
121	7.740
122	7.722
123	7.703
124	7.685
125	7.668
126	7.650
127	7.633
128	7.616
129	7.600
130	7.584
131	7.568
132	7.552
133	7.536
134	7.521
135	7.506
136	7.491
137	7.476
138	7.462
139	7.447
140	7.433
141	7.420
142	7.406
143	7.392
144	7.379
145	7.366
146	7.353

147	7.341
148	7.328
149	7.316
150	7.303
151	7.291
152	7.279
153	7.267
154	7.254
155	7.241
156	7.228
157	7.215
158	7.202
159	7.189
160	7.176
161	7.163
162	7.150
163	7.137
164	7.124
165	7.111
166	7.098
167	7.085
168	7.072
169	7.059
170	7.046
171	7.033
172	7.020
173	7.007
174	6.994
175	6.981
176	6.968
177	6.955
178	6.942
179	6.929
180	6.916
181	6.903
182	6.890
183	6.877
184	6.864
185	6.851
186	6.838
187	6.825
188	6.812
189	6.800
190	6.787
191	6.774
192	6.761
193	6.748
194	6.735
195	6.722
196	6.709
197	6.696
198	6.683
199	6.670
200	6.657
201	6.644
202	6.631
203	6.618
204	6.605
205	6.592
206	6.579
207	6.566
208	6.553
209	6.540
210	6.527
211	6.514
212	6.501
213	6.488
214	6.475
215	6.462
216	6.449
217	6.436
218	6.423
219	6.410
220	6.397
221	6.384
222	6.371
223	6.358
224	6.345
225	6.332
226	6.319
227	6.306
228	6.293
229	6.280
230	6.267
231	6.254
232	6.241
233	6.228
234	6.215
235	6.202
236	6.189
237	6.176
238	6.163
239	6.150
240	6.137
241	6.124
242	6.111
243	6.098
244	6.085
245	6.072
246	6.059
247	6.046
248	6.033
249	6.020
250	6.007
251	5.994
252	5.981
253	5.968
254	5.955
255	5.942
256	5.929
257	5.916
258	5.903
259	5.890
260	5.877
261	5.864
262	5.851
263	5.838
264	5.825
265	5.812
266	5.800
267	5.787
268	5.774
269	5.761
270	5.748
271	5.735
272	5.722
273	5.709
274	5.696
275	5.683
276	5.670
277	5.657
278	5.644
279	5.631
280	5.618
281	5.605
282	5.592
283	5.579
284	5.566
285	5.553
286	5.540
287	5.527
288	5.514
289	5.501
290	5.488
291	5.475
292	5.462
293	5.449
294	5.436
295	5.423
296	5.410
297	5.397
298	5.384
299	5.371
300	5.358
301	5.345
302	5.332
303	5.319
304	5.306
305	5.293
306	5.280
307	5.267
308	5.254
309	5.241
310	5.228
311	5.215
312	5.202
313	5.189
314	5.176
315	5.163
316	5.150
317	5.137
318	5.124
319	5.111
320	5.098
321	5.085
322	5.072
323	5.059
324	5.046
325	5.033
326	5.020
327	5.007
328	4.994
329	4.981
330	4.968
331	4.955
332	4.942
333	4.929
334	4.916
335	4.903
336	4.890
337	4.877
338	4.864
339	4.851
340	4.838
341	4.825
342	4.812
343	4.800
344	4.787
345	4.774
346	4.761
347	4.748
348	4.735
349	4.722
350	4.709
351	4.696
352	4.683
353	4.670
354	4.657
355	4.644
356	4.631
357	4.618
358	4.605
359	4.592
360	4.579
361	4.566
362	4.553
363	4.540
364	4.527
365	4.514
366	4.501
367	4.488
368	4.475
369	4.462
370	4.449
371	4.436
372	4.423
373	4.410
374	4.397
375	4.384
376	4.371
377	4.358
378	4.345
379	4.332
380	4.319
381	4.306
382	4.293
383	4.280
384	4.267
385	4.254
386	4.241
387	4.228
388	4.215
389	4.202
390	4.189
391	4.176
392	4.163
393	4.150
394	4.137
395	4.124
396	4.111
397	4.098
398	4.085
399	4.072
400	4.059
401	4.046
402	4.033
403	4.020
404	4.007
405	3.994
406	3.981
407	3.968
408	3.955
409	3.942
410	3.929
411	3.916
412	3.903
413	3.890
414	3.877
415	3.864
416	3.851
417	3.838
418	3.825
419	3.812
420	3.800
421	3.787
422	3.774
423	3.761
424	3.748
425	3.735
426	3.722
427	3.709
428	3.696
429	3.683
430	3.670
431	3.657
432	3.644
433	3.631
434	3.618
435	3.605
436	3.592
437	3.579
438	3.566
439	3.553
440	3.540
441	3.527
442	3.514
443	3.501
444	3.488
445	3.475
446	3.462
447	3.449
448	3.436
449	3.423
450	3.410
451	3.397
452	3.384
453	3.371
454	3.358
455	3.345
456	3.332
457	3.319
458	3.306
459	3.293
460	3.280
461	3.267
462	3.254
463	3.241
464	3.228
465	3.215
466	3.202
467	3.189
468	3.176
469	3.163
470	3.150
471	3.137
472	3.124
473	3.111
474	3.098
475	3.085
476	3.072
477	3.059
478	3.046
479	3.033
480	3.020
481	3.007
482	2.994
483	2.981
484	2.968
485	2.955
486	2.942
487	2.929
488	2.916
489	2.903
490	2.890
491	2.877
492	2.864
493	2.851
494	2.838
495	2.825
496	2.812
497	2.800
498	2.787
499	2.774
500	2.761
501	2.748
502	2.735
503	2.722
504	2.709
505	2.696
506	2.683
507	2.670
508	2.657
509	2.644
510	2.631
511	2.618
512	2.605
513	2.592
514	2.579
515	2.566
516	2.553
517	2.540
518	2.527
519	2.514
520	2.501
521	2.488
522	2.475
523	2.462
524	2.449
525	2.436
526	2.423
527	2.410
528	2.397
529	2.384
530	2.371
531	2.358
532	2.345
533	2.332
534	2.319
535	2.306
536	2.293
537	2.280
538	2.267
539	2.254
540	2.241
541	2.228
542	2.215
543	2.202
544	2.189
545	2.176
546	2.163
547	2.150
548	2.137
549	2.124
550	2.111
551	2.098
552	2.085
553	2.072
554	2.059
555	2.046
556	2.033
557	2.020
558	2.007
559	1.994
560	1.981
561	1.968
562	1.955
563	1.942
564	1.929
565	1.916
566	1.903
567	1.890
568	1.877
569	1.864
570	1.851
571	1.838
572	1.825
573	1.812
574	1.800
575	1.787
576	1.774
577	1.761
578	1.748
579	1.735
580	1.722
581	1.709
582	1.696
583	1.683
584	1.670
585	1.657
586	1.644
587	1.631
588	1.618
589	1.605
590	1.592
591	1.579
592	1.566
593	1.553
594	1.540
595	1.527
596	1.514
597	1.501
598	1.488
599	1.475
600	1.462
601	1.449
602	1.436
603	1.423
604	1.410
605	1.397
606	1.384
607	1.371
608	1.358
609	1.345
610	1.332
611	1.319
612	1.306
613	1.293
614	1.280
615	1.267
616	1.254
617	1.241
618	1.228
619	1.215
620	1.202
621	1.189
622	1.176
623	1.163
624	1.150
625	1.137
626	1.124
627	1.111
628	1.098
629	1.085
630	1.072
631	1.059
632	1.046
633	1.033
634	1.020
635	1.007
636	9.94
637	9.81
638	9.68
639	9.55
640	9.42
641	9.29
642	9.16
643	9.03
644	8.90
645	8.77
646	8.64
647	8.51
648	8.38
649	8.25
650	8.12
651	8.00
652	7.87
653	7.74
654	7.61
655	7.48
656</td	