

EIOPA RISK-FREE CURVE AUGUST-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 August 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_20230831_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_20230831_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 [94]:

```
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 [95]:  
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 [96]:  
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 [97]:  
#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 [98]:  
param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

Parameter input

Parameters sheet

In [99]: `param_raw.head()`

Out[99]:

	Euro_Maturities	Euro_Values	Austria_Maturities	Austria_Values	Belgium_Maturities	Be
Country						
Coupon_freq	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
LLP	20.00000	20.00000	20.00000	20.00000	20.00000	20.00000
Convergence	40.00000	40.00000	40.00000	40.00000	40.00000	40.00000
UFR	3.45000	3.45000	3.45000	3.45000	3.45000	3.45000
alpha	0.11312	0.11312	0.11312	0.11312	0.11312	0.11312

5 rows × 106 columns

The country selected is:

In [100...]: `country = "Turkey"`

In [101...]: `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 [102...]: `extra_param`

Out[102...]:

Country	
Coupon_freq	0.000000
LLP	9.000000
Convergence	51.000000
UFR	5.500000
alpha	0.164348
CRA	10.000000

Name: Turkey_Values, dtype: float64

In [103...]: `relevant_positions = pd.notna(maturities_country_raw.values)`

In [104...]

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

Maturity vector

Vector of maturities used in the calibration

In [105...]

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

Out[105...]

```
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 [106...]

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

Calibration vector

Vector **Qb** provided as input

In [107...]

```
Qb
```

Out[107...]

```
Country
1   -12.849459
2    7.480230
3   -2.619455
4    0.824166
5   -0.093618
6   -0.340462
7    0.745429
8   -0.724825
Name: Turkey_Values, dtype: float64
```

In [108...]

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

In [109...]

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

[Back to the top](#)

Calibration parameters and calibration vector provided

bv, EIOPA

In [110...]

```
# 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 [111...]

```

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.
    #     M_Obs = n x 1 ndarray. Observed bond maturities used for calibrating the curve.
    #     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 [112...]

```

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 [113...]

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

Out[113...]

Recalculated rates

0	0.197479
1	0.192255
2	0.191074
3	0.190492
4	0.190274
5	0.190172
6	0.189970
7	0.189868
8	0.189912
9	0.189477
10	0.188500
11	0.187084
12	0.185307
13	0.183230
14	0.180905

[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_20230831_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 [114...]

```
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 [115...]

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

EIOPA curve provided

Yield curve provided by EIOPA

In [116...]

```
target_curve.head()
```

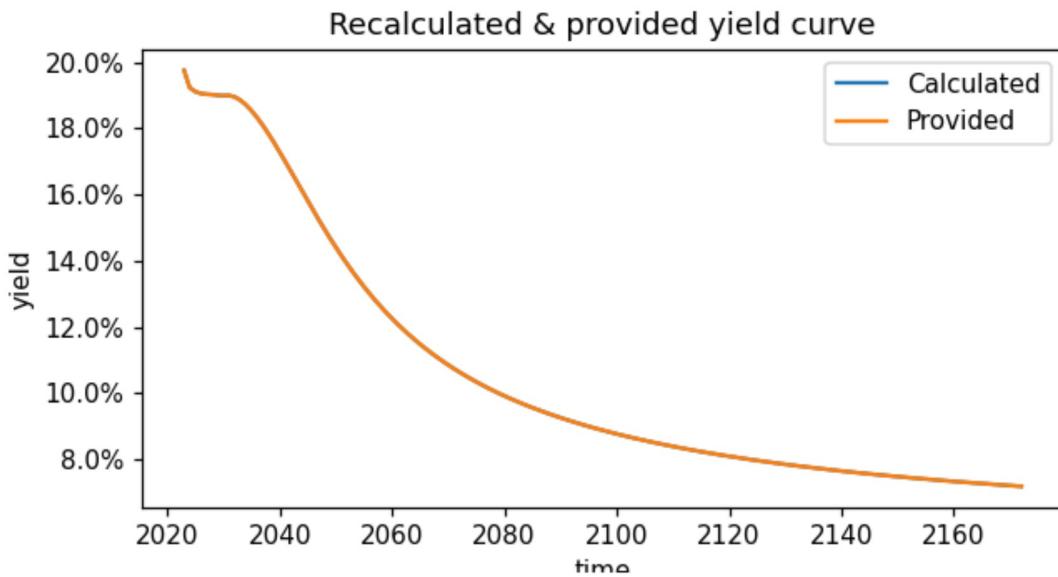
Out[116...]

Given rates

0	0.19748
1	0.19225
2	0.19107
3	0.19049
4	0.19027

```
In [117...]:  
x_data_label = range(2023, 2023+r_Target.shape[0], 1)
```

```
In [118...]:  
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 [119...]:  
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

```
In [120...]:  
test_statistics_bdp.head()
```

```
Out[120...]:  
Abs diff in bps
```

0	0.005979
1	0.047016
2	0.037848
3	0.024414

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.

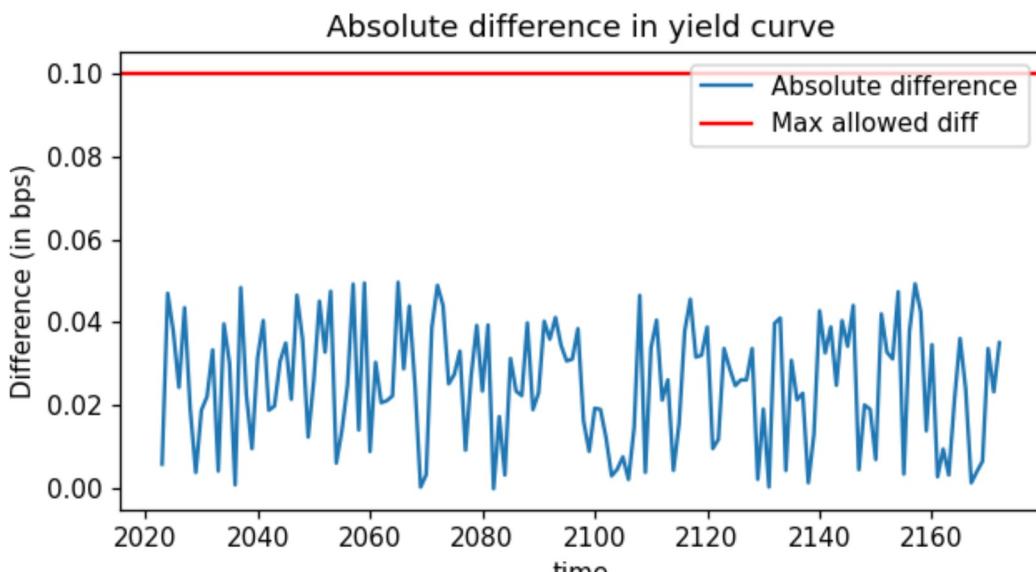
In [121...]

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

```
Test passed  
Test passed
```

In [122...]

```
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 August 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_20230831_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230831_Term_Structures.xlsx*.

In [123...]

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

Out[123...]

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

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

Final yield curve

Full yield curve provided by EIOPA in %

In [124...]

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

Out[124...]

Country	
1	19.748
2	19.225
3	19.107
4	19.049
5	19.027
6	19.017
7	18.997
8	18.987
9	18.991
10	18.948
11	18.850
12	18.708
13	18.531
14	18.323
15	18.091
16	17.838
17	17.569
18	17.288
19	16.998
20	16.702
21	16.403
22	16.104
23	15.807
24	15.514
25	15.226
26	14.945
27	14.672
28	14.407

29	14.150
30	13.903
31	13.666
32	13.437
33	13.218
34	13.008
35	12.807
36	12.614
37	12.429
38	12.253
39	12.084
40	11.923
41	11.768
42	11.620
43	11.478
44	11.343
45	11.213
46	11.088
47	10.968
48	10.853
49	10.743
50	10.636
51	10.534
52	10.436
53	10.342
54	10.251
55	10.163
56	10.078
57	9.997
58	9.918
59	9.842
60	9.768
61	9.697
62	9.628
63	9.561
64	9.497
65	9.434
66	9.374
67	9.315
68	9.258
69	9.202
70	9.149
71	9.096
72	9.046
73	8.996
74	8.948
75	8.902
76	8.856
77	8.812
78	8.769
79	8.727
80	8.686
81	8.646
82	8.607
83	8.569
84	8.532
85	8.496
86	8.461
87	8.426

88	8.392
89	8.359
90	8.327
91	8.296
92	8.265
93	8.235
94	8.205
95	8.177
96	8.148
97	8.121
98	8.094
99	8.067
100	8.041
101	8.016
102	7.991
103	7.966
104	7.942
105	7.919
106	7.896
107	7.873
108	7.851
109	7.829
110	7.808
111	7.787
112	7.766
113	7.746
114	7.726
115	7.706
116	7.687
117	7.668
118	7.650
119	7.631
120	7.613
121	7.596
122	7.578
123	7.561
124	7.545
125	7.528
126	7.512
127	7.496
128	7.480
129	7.465
130	7.449
131	7.434
132	7.420
133	7.405
134	7.391
135	7.376
136	7.363
137	7.349
138	7.335
139	7.322
140	7.309
141	7.296
142	7.283
143	7.271
144	7.258
145	7.246
146	7.234

147	7.222
148	7.210
149	7.199
150	7.187
151	7.176
152	7.165
153	7.154
154	7.143
155	7.132
156	7.121
157	7.110
158	7.099
159	7.088
160	7.077
161	7.066
162	7.055
163	7.044
164	7.033
165	7.022
166	7.011
167	7.000
168	6.989
169	6.978
170	6.967
171	6.956
172	6.945
173	6.934
174	6.923
175	6.912
176	6.901
177	6.890
178	6.879
179	6.868
180	6.857
181	6.846
182	6.835
183	6.824
184	6.813
185	6.802
186	6.791
187	6.780
188	6.769
189	6.758
190	6.747
191	6.736
192	6.725
193	6.714
194	6.703
195	6.692
196	6.681
197	6.670
198	6.659
199	6.648
200	6.637
201	6.626
202	6.615
203	6.604
204	6.593
205	6.582
206	6.571
207	6.560
208	6.549
209	6.538
210	6.527
211	6.516
212	6.505
213	6.494
214	6.483
215	6.472
216	6.461
217	6.450
218	6.439
219	6.428
220	6.417
221	6.406
222	6.395
223	6.384
224	6.373
225	6.362
226	6.351
227	6.340
228	6.329
229	6.318
230	6.307
231	6.296
232	6.285
233	6.274
234	6.263
235	6.252
236	6.241
237	6.230
238	6.219
239	6.208
240	6.197
241	6.186
242	6.175
243	6.164
244	6.153
245	6.142
246	6.131
247	6.120
248	6.109
249	6.098
250	6.087
251	6.076
252	6.065
253	6.054
254	6.043
255	6.032
256	6.021
257	6.010
258	6.000
259	5.989
260	5.978
261	5.967
262	5.956
263	5.945
264	5.934
265	5.923
266	5.912
267	5.901
268	5.890
269	5.879
270	5.868
271	5.857
272	5.846
273	5.835
274	5.824
275	5.813
276	5.802
277	5.791
278	5.780
279	5.769
280	5.758
281	5.747
282	5.736
283	5.725
284	5.714
285	5.703
286	5.692
287	5.681
288	5.670
289	5.659
290	5.648
291	5.637
292	5.626
293	5.615
294	5.604
295	5.593
296	5.582
297	5.571
298	5.560
299	5.549
300	5.538
301	5.527
302	5.516
303	5.505
304	5.494
305	5.483
306	5.472
307	5.461
308	5.450
309	5.439
310	5.428
311	5.417
312	5.406
313	5.395
314	5.384
315	5.373
316	5.362
317	5.351
318	5.340
319	5.329
320	5.318
321	5.307
322	5.296
323	5.285
324	5.274
325	5.263
326	5.252
327	5.241
328	5.230
329	5.219
330	5.208
331	5.197
332	5.186
333	5.175
334	5.164
335	5.153
336	5.142
337	5.131
338	5.120
339	5.109
340	5.098
341	5.087
342	5.076
343	5.065
344	5.054
345	5.043
346	5.032
347	5.021
348	5.010
349	5.000
350	4.989
351	4.978
352	4.967
353	4.956
354	4.945
355	4.934
356	4.923
357	4.912
358	4.901
359	4.890
360	4.879
361	4.868
362	4.857
363	4.846
364	4.835
365	4.824
366	4.813
367	4.802
368	4.791
369	4.780
370	4.769
371	4.758
372	4.747
373	4.736
374	4.725
375	4.714
376	4.703
377	4.692
378	4.681
379	4.670
380	4.659
381	4.648
382	4.637
383	4.626
384	4.615
385	4.604
386	4.593
387	4.582
388	4.571
389	4.560
390	4.549
391	4.538
392	4.527
393	4.516
394	4.505
395	4.494
396	4.483
397	4.472
398	4.461
399	4.450
400	4.439
401	4.428
402	4.417
403	4.406
404	4.395
405	4.384
406	4.373
407	4.362
408	4.351
409	4.340
410	4.329
411	4.318
412	4.307
413	4.296
414	4.285
415	4.274
416	4.263
417	4.252
418	4.241
419	4.230
420	4.219
421	4.208
422	4.197
423	4.186
424	4.175
425	4.164
426	4.153
427	4.142
428	4.131
429	4.120
430	4.109
431	4.098
432	4.087
433	4.076
434	4.065
435	4.054
436	4.043
437	4.032
438	4.021
439	4.010
440	4.000
441	3.989
442	3.978
443	3.967
444	3.956
445	3.945
446	3.934
447	3.923
448	3.912
449	3.901
450	3.890
451	3.879
452	3.868
453	3.857
454	3.846
455	3.835
456	3.824
457	3.813
458	3.802
459	3.791
460	3.780
461	3.769
462	3.758
463	3.747
464	3.736
465	3.725
466	3.714
467	3.703
468	3.692
469	3.681
470	3.670
471	3.659
472	3.648
473	3.637
474	3.626
475	3.615
476	3.604
477	3.593
478	3.582
479	3.571
480	3.560
481	3.549
482	3.538
483	3.527
484	3.516
485	3.505
486	3.494
487	3.483
488	3.472
489	3.461
490	3.450
491	3.439
492	3.428
493	3.417
494	3.406
495	3.395
496	3.384
497	3.373
498	3.362
499	3.351
500	3.340
501	3.329
502	3.318
503	3.307
504	3.296
505	3.285
506	3.274
507	3.263
508	3.252
509	3.241
510	3.230
511	3.219
512	3.208
513	3.197
514	3.186
515	3.175
516	3.164
517	3.153
518	3.142
519	3.131
520	3.120
521	3.109
522	3.098
523	3.087
524	3.076
525	3.065
526	3.054
527	3.043
528	3.032
529	3.021
530	3.010
531	3.000
532	2.989
533	2.978
534	2.967
535	2.956
536	2.945
537	2.934
538	2.923
539	2.912
540	2.901
541	2.890
542	2.879
543	2.868
544	2.857
545	2.846
546	2.835
547	2.824
548	2.813
549	2.802
550	2.791
551	2.780
552	2.769
553	2.758
554	2.747
555	2.736
556	2.725
557	2.714
558	2.703
559	2.692
560	2.681
561	2.670
562	2.659
563	2.648
564	2.637
565	2.626
566	2.615
567	2.604
568	2.593
569	2.582
570	2.571
571	2.560
572	2.549
573	2.538
574	2.527
575	2.516
576	2.505
577	2.494
578	2.483
579	2.472
580	2.461
581	2.450
582	2.439
583	2.428
584	2.417
585	2.406
586	2.395
587	2.384
588	2.373
589	2.362
590	2.351
591	2.340
592	2.329
593	2.318
594	2.307
595	2.296
596	2.285
597	2.274
598	2.263
599	2.252
600	2.241
601	2.230
602	2.219
603	2.208
604	2.197
605	2.186
606	2.175
607	2.164
608	2.153
609	2.142
610	2.131
611	2.120
612	2.109
613	2.098
614	2.087
615	2.076
616	2.065
617	2.054
618	2.043
619	2.032
620	2.021
621	2.010
622	2.000
623	1.989
624	1.978
625	1.967
626	1.956
627	1.945
628	1.934
629	1.923
630	1.912
631	1.901
632	1.890
633	1.879
634	1.868
635	1.857
636	1.846
637	1.835
638	1.824
639	1.813
640	1.802
641	1.791
642	1.780
643	1.769
644	1.758
645	1.747
646	1.736
647	1.725
648	1.714
649	1.703
650	1.692
651	1.681
652	1.670
653	1.659
654	1.648
655	