

#<window 480 on *Async Shell Command*>

I Capacité numérique

Simuler, à l'aide d'un langage de programmation, l'action d'un filtre sur un signal périodique dont le spectre est fourni. Mettre en évidence l'influence des caractéristiques du filtre sur l'opération de filtrage.

II Modules

Le module `signal` de `scipy` (documentation) donne accès entre autres :

- à des fonctions représentant des signaux usuels (créneau avec `square`, triangle et dents de scie avec `sawtooth`)
- à des fonctions définissant des fonctions de transfert de filtres par les coefficients de leurs fractions rationnelles (fonction `freqs`)

Le module `fft` de `scipy` (documentation) permet entre autres de réaliser les transformées de Fourier et transformées de Fourier inverse (en utilisant l'algorithme de Fast Fourier Transform) pour :

- à partir d'un signal $f(t)$ numérisé, calculer une approximation numérique de son spectre de Fourier discret
- à partir d'un spectre de Fourier discret, calculer la valeur d'une fonction correspondante en un ensemble d'instant.

Dans ces deux cas, l'ensemble des instants où est évaluée la fonction et l'ensemble des fréquences où est évaluée sa transformée de Fourier sont tous les deux discrets.

Le module `ma` de `numpy` (documentation) offre ici la facilité de « masquer » certains éléments d'un tableau qu'on souhaite éliminer : ici les parties du spectre de poids trop faible.

```
1 %matplotlib notebook
```

La ligne précédente ne doit apparaître que dans les notebooks Jupyter, pas dans un fichier python.

```
1 %matplotlib inline
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

III Étude d'un filtre du premier ordre

III.1 Définition

On définit un filtre passe-bas du premier ordre par son gain en bande passante et sa fréquence de coupure.

```
1 pi = np.pi
2 # Définition du filtre
3 f_c = 2e2 # fréquence de coupure (en Hz)
4 omega_c = 2.0*pi*f_c # pulsation de coupure (en rad/s)
5 H_0 = 2 # gain en bande passante
6 # Coefficients du dénominateur rangés par degrés décroissants
7 a = np.array( [1/omega_c, 1.0] )
8 # Coefficients du numérateur
9 b = np.array( [H_0] )
10 H_coefs = (b,a)
```

III.2 Diagramme de Bode

On fait calculer le module et l'argument de la fonction de transfert pour un ensemble de fréquences déterminé, choisi en échelle logarithmique : ici 400 fréquences équiréparties en échelle log entre 10^1 et 10^5 Hz. La fonction `signal.freqs` renvoie :

- w la liste des pulsations utilisées
- H le tableau des valeurs (complexes) de la fonction de transfert pour ces pulsations.

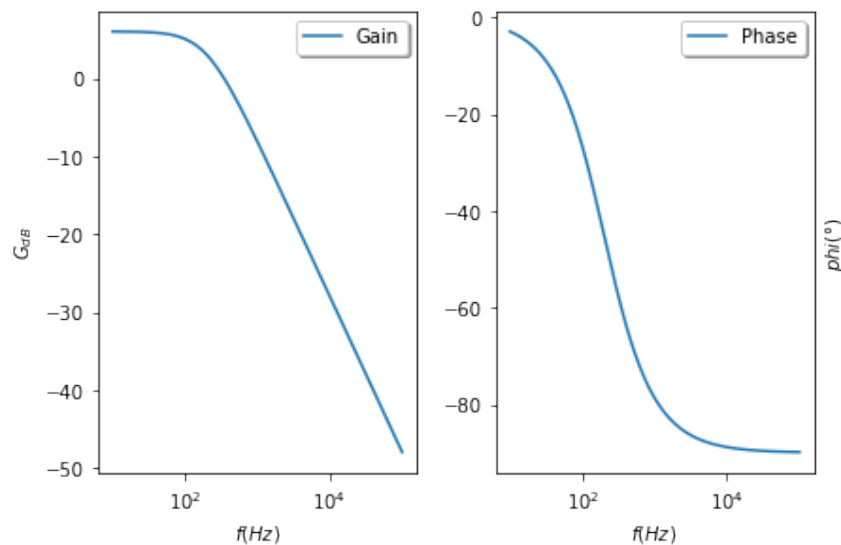
```
1 # Plage de fréquence à étudier
2 frequences = np.logspace(1, 5, 400)
3 [ w, H ] = signal.freqs(b,a,2*pi*frequences)
```

On peut ensuite tracer le diagramme de Bode correspondant.

```

1 figBode1, (axBodeGain1, axBodePhase1) = plt.subplots(1,2) #pour avoir deux figures côte à
  ↳ côte
2 figBode1.tight_layout()
3 # gain en décibel, np.absolute calcule le module
4 G = 20.0 * np.log10(np.absolute(H))
5 # phase en degrés, np.angle calcule l'argument en radian, converti par rad2deg
6 phase = np.rad2deg(np.angle(H))
7 axBodeGain1.semilogx()
8 axBodePhase1.semilogx()
9 axBodeGain1.plot(frequences, G, label='Gain')
10 axBodeGain1.set_xlabel(r"$f$ (Hz)")
11 axBodeGain1.set_ylabel(r"$G_{dB}$")
12 axBodePhase1.plot(frequences, phase, label='Phase')
13 axBodePhase1.set_xlabel(r"$f$ (Hz)")
14 axBodePhase1.set_ylabel(r"$\phi$ (°)")
15 axBodePhase1.yaxis.set_label_position("right")
16 axBodePhase1.legend(loc='best', shadow=True)
17 axBodeGain1.legend(loc='best', shadow=True)
18 figBode1.show()

```



IV Simulation du filtrage

IV.1 Échantillonnage de la fonction

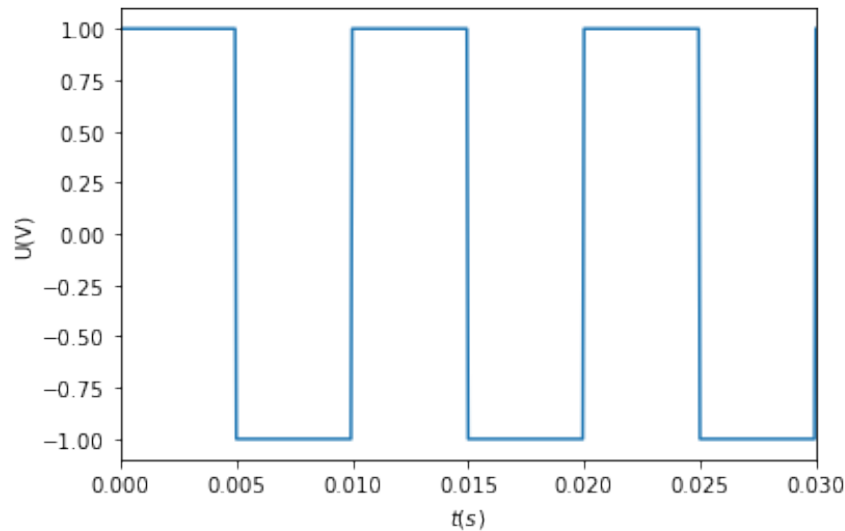
On calcule les valeurs de la fonction sur laquelle sera appliquée le filtre sur un échantillon d'instants. La durée entre deux instants Δt doit être suffisamment courte pour que deux fonctions différentes n'aient pas le même échantillonnage. On utilisera le « critère de Shannon » selon lequel $\Delta t = 1/(2 f_{\max})$ avec f_{\max} la fréquence maximale du spectre du signal.

On utilise ici une fonction créneau de fréquence f_{Cre} . Pour que son échantillonnage reproduise fidèlement ses harmoniques jusqu'au rang 9, on doit donc échantillonner avec $\Delta t = 1/(18 f_{\text{Cre}})$. On rajoute un facteur 10 pour augmenter la précision.

```

1 HarmoniqueMax = 9
2 fCre = 100 #Hz
3 Delta_t = 1/(20*HarmoniqueMax*fCre)
4 echantillon_t = np.arange(0, 20/fCre, Delta_t) # pour disposer de 20 périodes pour
  ↳ l'échantillonnage
5 EntreeCreneau = signal.square(echantillon_t*2*pi*fCre) # signal.square a pour période
  ↳ 2 pi
6 figCreneau, axCreneau = plt.subplots()
7 axCreneau.set_xlim(0, 3/fCre)
8 axCreneau.plot(echantillon_t, EntreeCreneau)
9 axCreneau.set_xlabel(r"$t$ (s)")
10 axCreneau.set_ylabel(r"$U$ (V)")
11 figCreneau.show()

```



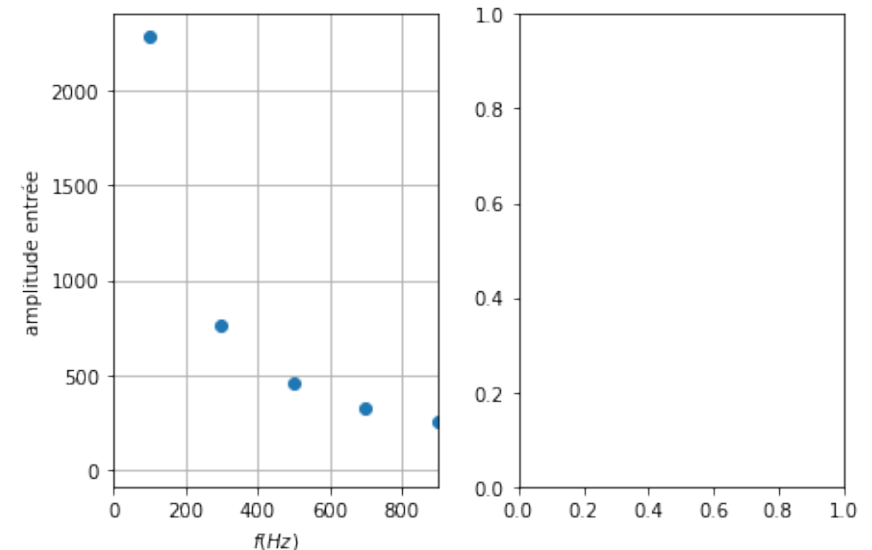
IV.2 Transformée de Fourier rapide

On calcule les fréquences `FrequencesFFTCreneau` adaptées à l'échantillonnage avec `fft.freq`. On applique la transformée de Fourier rapide au signal échantillonné avec `fft` pour obtenir `FFTEntreeCreneau`, un tableau des amplitudes complexes des composantes de Fourier de `EntreeCreneau`. Le masque `mask` permet de ne garder à l'affichage que les composantes de Fourier dont l'amplitude est supérieure à un seuil.

On trace ensuite les modules de ces amplitudes complexes.

```
1 FrequencesFFTCreneau = fftpack.fftfreq(len(EntreeCreneau), Delta_t)
2 FFTEntreeCreneau = fftpack.fft(EntreeCreneau)
3 CutoffGdB=40
4 FFTEntreeCreneauGdB=20*np.log10(np.absolute(FFTEntreeCreneau))
5 mask=ma.masked_less(FFTEntreeCreneauGdB, np.max(FFTEntreeCreneauGdB)-CutoffGdB).mask
6 FrequencesFFTCreneauMasked=FrequencesFFTCreneau[~mask]
7 FFTEntreeCreneauMasked=FFTEntreeCreneau[~mask]
8 FFTEntreeCreneauGdBMasked=FFTEntreeCreneauGdB[~mask]
9 figFFTCreneau, (axFFTEntreeCreneau, axFFTSortieCreneau) = plt.subplots(1,2) #pour avoir
10   ↪ deux figures côte à côte
11 figFFTCreneau.tight_layout()
12 axFFTEntreeCreneau.plot(FrequencesFFTCreneauMasked,
13   ↪ np.absolute(FFTEntreeCreneauMasked), 'o', label='Entree')
14 axFFTEntreeCreneau.set_xlabel(r"$f$ (Hz)")
15 axFFTEntreeCreneau.set_ylabel(r"amplitude entrée")
```

```
14 axFFTEntreeCreneau.set_xlim([0, HarmoniqueMax*fCre])
15 axFFTEntreeCreneau.grid(which='both')
```



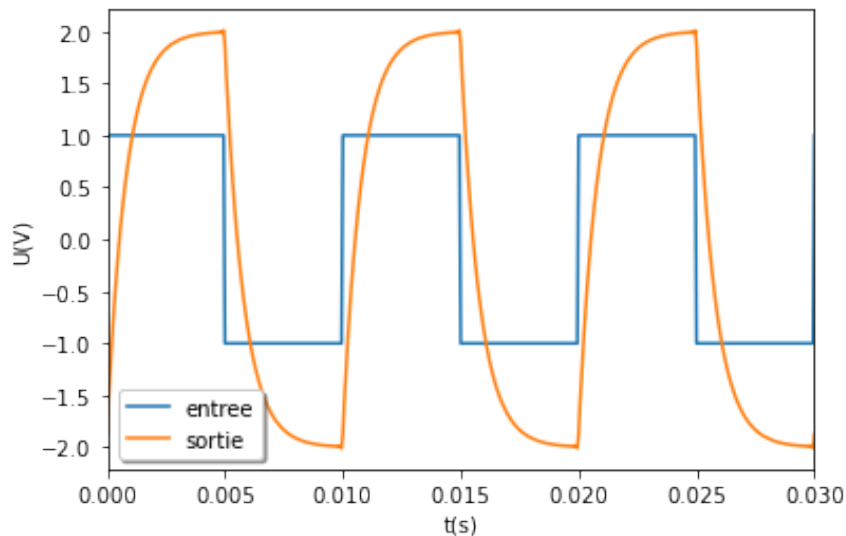
On évalue ensuite les valeurs du gain complexe `HFFTCreneau` calculé pour ces fréquences puis on multiplie une à une les amplitudes de `FFTEntreeCreneau` par la valeur du gain `HFFTCreneau` pour obtenir la transformée de Fourier discrète du signal de sortie `FFTSortieCreneau`.

```
1 HFFTCreneau = signal.freqs(b,a,2*pi*FrequencesFFTCreneau)[1]
2 FFTSortieCreneau = HFFTCreneau * FFTEntreeCreneau
3 FFTSortieCreneauMasked = FFTSortieCreneau[~mask]
4 axFFTSortieCreneau.plot(FrequencesFFTCreneauMasked,
5   ↪ np.absolute(FFTSortieCreneauMasked), '+', label='Sortie')
6 axFFTSortieCreneau.set_xlabel(r"$f$ (Hz)")
7 axFFTSortieCreneau.set_ylabel(r"amplitude sortie")
8 axFFTSortieCreneau.set_xlim([0, HarmoniqueMax*fCre])
9 axFFTSortieCreneau.grid(which='both')
10 axFFTSortieCreneau.yaxis.set_label_position("right")
11 axFFTSortieCreneau.legend(loc='best', shadow=True)
12 axFFTEntreeCreneau.legend(loc='best', shadow=True)
13 figFFTCreneau.show()
```

Il suffit ensuite d'utiliser la transformée de Fourier Inverse `ifft` pour retrouver la fonction en sortie de filtre, qu'on superpose à la fonction en entrée pour les comparer.

L'algorithme d'aller-et-retour entre fft et ifft peut faire apparaître des parties imaginaires dans le signal de sortie. On vérifie qu'elles sont négligeables avec `max(np.imag)` et on ne trace que la partie réelle

```
1 SortieCreneau = fftpack.ifft(FTSortieCreneau)
2 figCreneauES, axCreneauES = plt.subplots()
3 axCreneauES.plot(echantillon_t, EntreeCreneau, label='entree')
4 print(f'max des parties imaginaires {max(np.imag(SortieCreneau))}')
5 axCreneauES.plot(echantillon_t, np.real(SortieCreneau), label='sortie')
6 axCreneauES.set_xlim([0, 3/fCre])
7 axCreneauES.set_xlabel(r't (s)')
8 axCreneauES.set_ylabel(r'U (V)')
9 axCreneauES.legend(loc='best', shadow=True)
10 figCreneauES.show()
```

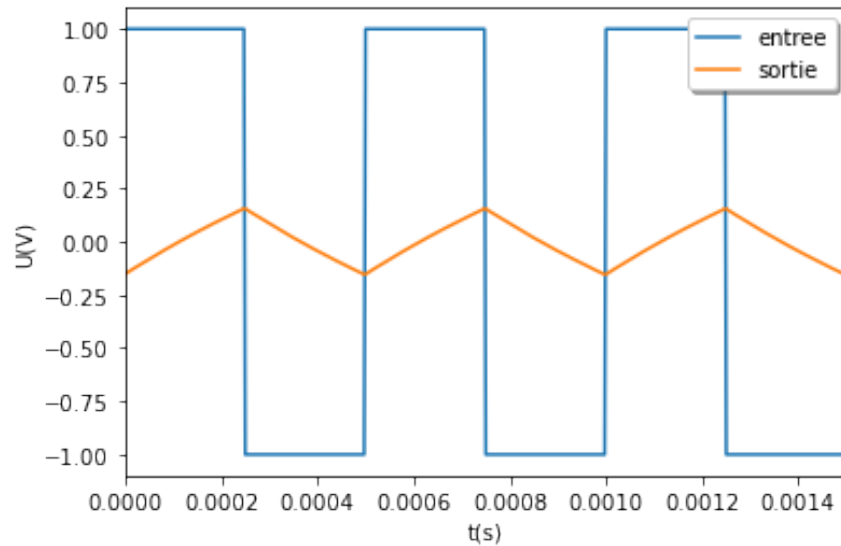


V Questions du DM05

V.1 II4a

```
1 # Définition du filtre
2 DMf_c = 2e2 # fréquence de coupure (en Hz)
3 DMomega_c = 2.0*pi*DMf_c # pulsation de coupure (en rad/s)
```

```
4 DMH_0 = 1 # gain en bande passante
5 # Coefficients du dénominateur rangés par degrés décroissants
6 DMA = np.array([1/DMomega_c, 1.0])
7 # Coefficients du numérateur
8 DMb = np.array([DMH_0])
9 DMH_coeffs = (DMb, DMA)
10 DMHarmoniqueMax = 9
11 DMfCre = 2e3 # Hz
12 DMCreneauAmplitude = 1
13 DMDelta_t = 1/(20*DMHarmoniqueMax*DMfCre)
14 DMechantillon_t = np.arange(0, 20/DMfCre, DMDelta_t) # pour disposer de 20 périodes
15 # pour l'échantillonnage
16 DMEntreeCreneau = DMCreneauAmplitude * signal.square(DMechantillon_t*2*pi*DMfCre) #
17 # signal.square a pour période 2 pi
18 DMFFTFrequenceCreneau = fftpack.fftfreq(len(DMEntreeCreneau), DMDelta_t)
19 DMFFTEntreeCreneau = fftpack.fft(DMEntreeCreneau)
20 DMCutoffGdB=40
21 DMFFTEntreeCreneauGdB=20*np.log10(np.absolute(DMFFTEntreeCreneau))
22 DMmask=ma.masked_less(DMFFTEntreeCreneauGdB, np.max(DMFFTEntreeCreneauGdB)-DMCutoffGdB).mask
23 DMFFTEntreeCreneauMasked=DMFFTEntreeCreneau[~DMmask]
24 DMFFTEntreeCreneauMaskedGdB=DMFFTEntreeCreneauMasked[~DMmask]
25 DMHFFTCreneau = signal.freqs(DMb, DMA, 2*pi*DMFFTEntreeCreneauMaskedGdB)[1]
26 DMFFTSortieCreneau = DMHFFTCreneau * DMFFTEntreeCreneauMaskedGdB
27 DMSortieCreneau = fftpack.ifft(DMFFTSortieCreneau)
28 figDMCreneauES, axDMCreneauES = plt.subplots()
29 axDMCreneauES.plot(DMechantillon_t, DMSortieCreneau, label='entree')
30 print(f'max des parties imaginaires {max(np.imag(DMSortieCreneau))}')
31 axDMCreneauES.plot(DMechantillon_t, np.real(DMSortieCreneau), label='sortie')
32 axDMCreneauES.set_xlim([0, 3/DMfCre])
33 axDMCreneauES.set_xlabel(r't (s)')
34 axDMCreneauES.set_ylabel(r'U (V)')
35 axDMCreneauES.legend(loc='best', shadow=True)
36 figDMCreneauES.show()
```



On calcule ensuite l'amplitude du signal triangulaire obtenu :

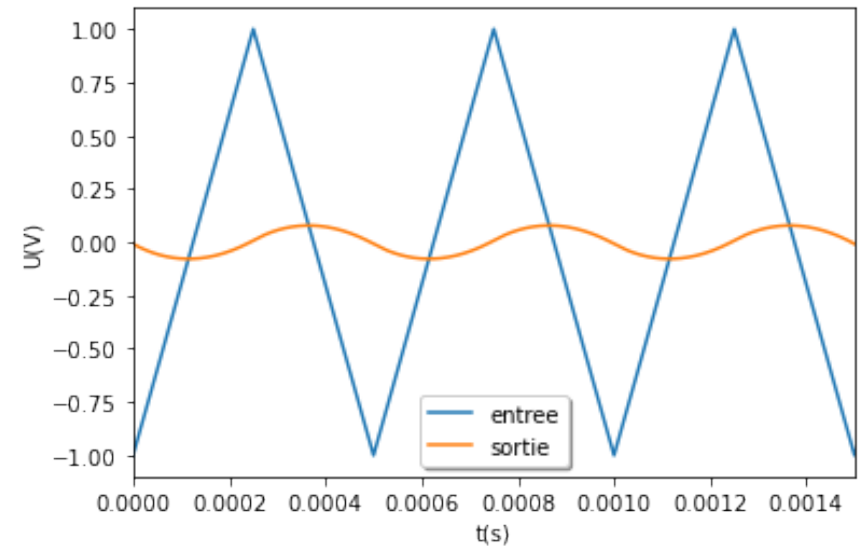
```
1 print(f'amplitude: {(max(np.real(DMSortieCreneau)) -
  ↳ min(np.real(DMSortieCreneau)))/2} V/')
```

V.2 II4c

On change simplement le signal d'entrée :

```
1 DMTriAmplitude = 1
2 DMEntreeTri = DMTriAmplitude * signal.sawtooth(DMechantillon_t*2*pi*DMfCre, width=.5)
  ↳ # signal.sawtooth a pour période 2 pi
3 DMFFrequencesFFTTri = fftpack.fftfreq(len(DMEntreeTri), DMDelta_t)
4 DMFFTEntreeTri = fftpack.fft(DMEntreeTri)
5 DMFFTEntreeTriGdB = 20*np.log10(np.absolute(DMFFTEntreeTri))
6 DMmask = ma.masked_less(DMFFTEntreeTriGdB, np.max(DMFFTEntreeTriGdB) - DMCutoffGdB).mask
7 DMFFrequencesFFTTriMasked = DMFFrequencesFFTTri[~DMmask]
8 DMFFTEntreeTriMasked = DMFFTEntreeTri[~DMmask]
9 DMFFTEntreeTriGdBMasked = DMFFTEntreeTriGdB[~DMmask]
10 DMHFFFTTri = signal.freqs(DMb, DMA, 2*pi*DMFFrequencesFFTTri)[1]
11 DMFFTSortieTri = DMHFFFTTri * DMFFTEntreeTri
12 DMFFTSortieTriMasked = DMFFTSortieTri[~DMmask]
13 DMSortieTri = fftpack.ifft(DMFFTSortieTri)
```

```
14 figDMTriES, axDMTriES = plt.subplots()
15 axDMTriES.plot(DMechantillon_t, DMEntreeTri, label='entree')
16 print(f'max des parties imaginaires {max(np.imag(DMSortieTri))}')
17 axDMTriES.plot(DMechantillon_t, np.real(DMSortieTri), label='sortie')
18 axDMTriES.set_xlim([0, 3/DMfCre])
19 axDMTriES.set_xlabel(r't(s)')
20 axDMTriES.set_ylabel(r'U(V)')
21 axDMTriES.legend(loc='best', shadow=True)
22 figDMTriES.show()
```



V.3 III4b

On change simplement la fonction de transfert, de coefficients DM2a. On trace la courbe du signal de sortie ainsi que les spectres de Fourier des signaux d'entrée et de sortie : on y vérifie bien que les amplitudes du fondamental et de la 2^e harmonique sont bien très proches¹ La différence, de l'ordre de 5%, est vraisemblablement une erreur d'arrondi qui existait déjà dans le spectre du signal d'entrée dont l'amplitude n'était pas exactement 20% de celle du fondamental.}

On trace également à titre de comparaison la somme :

$i_{\{$

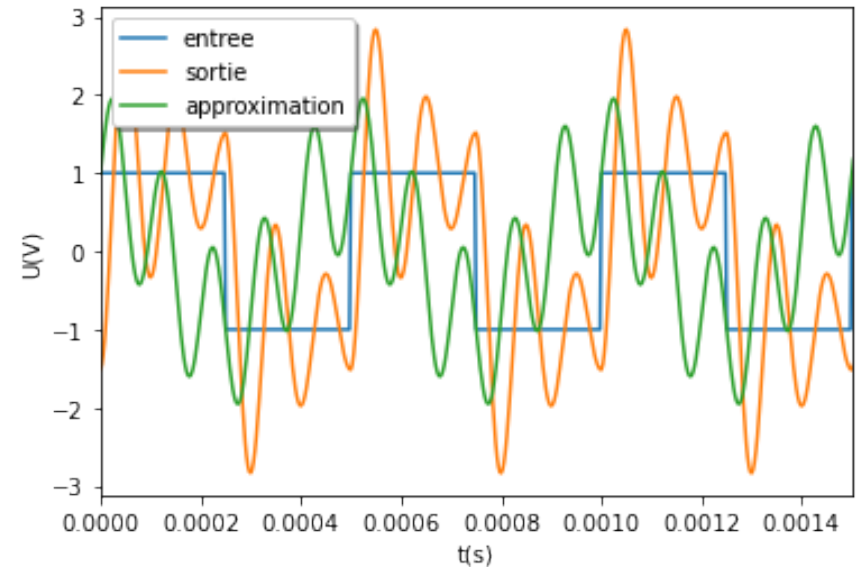
$$\cos(2\pi f_C r t) + \cos(10\pi f_C r t - \pi/2),$$

qui ressemble au signal produit par le filtre mais en diffère significativement car l'harmonique de rang 3 possède une amplitude non négligeable.

```

1  # Définition du filtre
2  DM2fCre = 2e3 #Hz
3  DM2CreneauAmplitude = 1
4  DM2f_c = 5*DM2fCre          # 5x la fréquence du créneau
5  DM2Q = 5
6  DM2omega_c = 2.0*pi*DM2f_c  # pulsation de coupure (en rad/s)
7  DM2H_0 = 1                  # gain en bande passante
8  # Coefficients du dénominateur rangés par degrés décroissants
9  DM2a = np.array( [1/(DM2omega_c)**2, 1/(DM2omega_c*DM2Q), 1.0] )
10 # Coefficients du numérateur
11 DM2b = np.array( [DM2H_0] )
12 DM2H_coeffs = (DM2b, DM2a)
13 DM2HarmoniqueMax = 9
14 DM2Delta_t = 1/(20*DM2HarmoniqueMax*DM2fCre)
15 DM2echantillon_t = np.arange(0, 20/DM2fCre, DM2Delta_t) # pour disposer de 20 périodes
16 # pour l'échantillonnage
17 DM2EntreeCreneau = DM2CreneauAmplitude * signal.square(DM2echantillon_t*2*pi*DM2fCre)
18 # signal.square a pour période 2 pi
19 DM2FrequenciesFFTCreneau = fftpack.fftfreq(len(DM2EntreeCreneau), DM2Delta_t)
20 DM2FFTEntreeCreneau = fftpack.fft(DM2EntreeCreneau)
21 DM2CutoffGdB=40
22 DM2FFTEntreeCreneauGdB=20*np.log10(np.absolute(DM2FFTEntreeCreneau))
23 DM2mask=ma.masked_less(DM2FFTEntreeCreneauGdB, np.max(DM2FFTEntreeCreneauGdB)-
24 # DM2CutoffGdB).mask
25 DM2FrequenciesFFTCreneauMasked=DM2FrequenciesFFTCreneau[~DM2mask]
26 DM2FFTEntreeCreneauMasked = DM2FFTEntreeCreneau[~DM2mask]
27 DM2FFTEntreeCreneauGdBMasked = DM2FFTEntreeCreneauGdB[~DM2mask]
28 DM2HFFTCreneau = signal.freqs(DM2b, DM2a, 2*pi*DM2FrequenciesFFTCreneau)[1]
29 DM2FFTSortieCreneau = DM2HFFTCreneau * DM2FFTEntreeCreneau
30 DM2FFTSortieCreneauMasked = DM2FFTSortieCreneau[~DM2mask]
31 DM2SortieCreneau = fftpack.ifft(DM2FFTSortieCreneau)
32
33 DM2Approx = np.cos(2*pi*DM2fCre*DM2echantillon_t) +
34 # np.cos(10*pi*DM2fCre*DM2echantillon_t - np.pi/2)
35
36 figDM2CreneauES, axDM2CreneauES = plt.subplots()
37 axDM2CreneauES.plot(DM2echantillon_t, DM2EntreeCreneau, label='entree')
38 print(f'max des parties imaginaires {max(np.imag(DM2SortieCreneau))}')
39 axDM2CreneauES.plot(DM2echantillon_t, np.real(DM2SortieCreneau), label='sortie')
40 axDM2CreneauES.plot(DM2echantillon_t, DM2Approx, label='approximation')
41 axDM2CreneauES.set_xlim([0, 3/DM2fCre])
42 axDM2CreneauES.set_xlabel(r't (s)')
43 axDM2CreneauES.set_ylabel(r'U (V)')
44 axDM2CreneauES.legend(loc='best', shadow=True)
45 figDM2CreneauES.show()

```



```

1  figDM2FFT, axDM2FFT = plt.subplots()
2  figDM2FFT.tight_layout()
3  axDM2FFT.plot(DM2FrequenciesFFTCreneauMasked, np.absolute(DM2FFTEntreeCreneauMasked),
4  # label='Entree')
5  axDM2FFT.plot(DM2FrequenciesFFTCreneauMasked, np.absolute(DM2FFTSortieCreneauMasked),
6  # label='Sortie')
7  axDM2FFT.set_xlabel(r"$f$ (Hz)")
8  axDM2FFT.set_ylabel(r"amplitude entrée")
9  axDM2FFT.set_xlim([0, HarmoniqueMax*DM2fCre])
10 axDM2FFT.grid(which='both')
11 figDM2FFT.show()

```

