projet.py

```
001| from random import randint
002| from hammingCode import HammingCode,ExtendedHammingCode,HammingCodeXOR
003| import numpy as np
004| import matplotlib.pyplot as plt
005| import time
006| import csv
007| import datetime
008|
009| def hammingDistance(l1,l2):
010|     if len(l1) != len(l2):
011|         raise "Arguments are not the same size"
012|     counter = 0
013|     for i in range(0,len(l1)):
014|         if l1[i] != l2[i] :
015|             counter += 1
016|     return counter
017|
018| def divideLists(l1,l2):
019|     if len(l1) != len(l2):
020|         raise "Arguments are not the same size"
021|     return [l1[i]/l2[i] for i in range(0,len(l1))]
022|
023| def generateRandomBits(n):
024|     return [ randint(0,1) for i in range(0,n)]
025|
026| def changeOneRandomBit(l):
027|     """chance = randint(0,1000)
028|     if chance < 10:"""
029|     pos = randint(0,len(l)-1)
030|     l[pos] = 1 - l[pos]
031|     return (l,pos)
032|
033| def calculateBitRate(times):
034|     l = [0]*len(times)
035|     for i in range(0,len(times)):
036|         k = 2+i
037|         l[i] = (1/times[i])*(2**k-k-1)
038|     return l
039|
040| def saveTimesToCSV(T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor):
041|     t = datetime.datetime.now()
042|     file_name = "times/{0}_{1}_{2}_T{3}
_{4}.csv".format(t.year,t.month,t.day,t.hour,t.minute)
043|     puissances = [3+i for i in range(0,len(T_enc_norm_gen))]
044|     tailles_données = [2**i - i -1 for i in puissances]
045|     tailles_paquets = [2**i -1for i in puissances]
046|     with open(file_name, 'w', newline='',encoding='utf-8') as csvfile:
047|         writer = csv.writer(csvfile, delimiter=';')
048|
049|         writer.writerow(["Facteur"] + puissances)
050|         writer.writerow(["Taille des données (en bits)"] + tailles_données)
051|         writer.writerow(["Taille des paquets (en bits)"] + tailles_paquets)
052|         writer.writerow(["Encodage normal (Matrice Génératrice)"] + T_enc_norm_gen)
053|         writer.writerow(["Encodage normal (Méthode Manuelle)"] + T_enc_norm_manual)
054|         writer.writerow(["Décodage normal"] + T_dec_norm)
055|         writer.writerow(["Encodage étendu (Matrice Génératrice)"] + T_enc_ext_gen)
056|         writer.writerow(["Encodage étendu (Méthode Manuelle)"] + T_enc_ext_manual)
057|         writer.writerow(["Décodage étendu"] + T_dec_ext)
058|         writer.writerow(["Encodage (Méthode XOR)"] + T_enc_xor)
059|         writer.writerow(["Décodage (Méthode XOR)"] + T_dec_xor)
060|
061| def measureManualMethod(code,nb_test):
062|     t_enc = 0
063|     t_dec = 0
064|     for i in range(0,nb_test):
065|         if i % 500 == 0:
066|             print(i)
067|         data = generateRandomBits(code.data_chunk_size*10)
068|         chunks = code.cutDataInChunks(data)
069|         t1 = time.time()
070|         blocks = code.encodeData(data)
071|         t2 = time.time()
072|         for block in blocks:
073|             changeOneRandomBit(block)
```

```
074|         t3 = time.time()
075|         corr_data = code.decodeBlocks(blocks)
076|         t4 = time.time()
077|
078|         if corr_data != chunks:
079|             print(corr_data)
080|             print(chunks)
081|             raise Exception("Erreur de décryption")
082|
083|         t_enc += t2-t1
084|         t_dec += t4-t3
085|     return (t_enc/(nb_test*10),t_dec/(nb_test*10))
086|
087| def measureGeneratorMethod(code,nb_test):
088|     t_enc = 0
089|     t_dec = 0
090|     for i in range(0,nb_test):
091|         if i % 500 == 0:
092|             print(i)
093|         data = generateRandomBits(code.data_chunk_size*10)
094|         chunks = code.cutDataInChunks(data)
095|         t1 = time.time()
096|         blocks = code.encodeDataWithGeneratorMatrix(data)
097|         t2 = time.time()
098|         for block in blocks:
099|             changeOneRandomBit(block)
100|         t3 = time.time()
101|         corr_data = code.decodeBlocks(blocks)
102|         t4 = time.time()
103|
104|         if corr_data != chunks:
105|             print(corr_data)
106|             print(chunks)
107|             raise Exception("Erreur de décryption")
108|
109|         t_enc += t2-t1
110|         t_dec += t4-t3
111|     return (t_enc/(nb_test*10),t_dec/(nb_test*10))
112|
113| def measureXORMethod(code,nb_test):
114|     t_enc = 0
115|     t_dec = 0
116|     for i in range(0,nb_test):
117|         if i % 500 == 0:
118|             print(i)
119|         data = generateRandomBits(code.data_chunk_size*10)
120|         chunks = code.cutDataInChunks(data)
121|         t1 = time.time()
122|         blocks = code.encodeData(data)
123|         t2 = time.time()
124|         for block in blocks:
125|             changeOneRandomBit(block)
126|         t3 = time.time()
127|         corr_data = code.decodeBlocks(blocks)
128|         t4 = time.time()
129|
130|         if corr_data != chunks:
131|             print(corr_data)
132|             print(chunks)
133|             raise Exception("Erreur de décryption")
134|
135|         t_enc += t2-t1
136|         t_dec += t4-t3
137|     return (t_enc/(nb_test*10),t_dec/(nb_test*10))
138|
139| def rawTimesGraphs(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor):
140|
141|     plt.figure(1)
142|     plt.title("Temps en fonction de la taille des paquets (2**k)")
143|     plt.plot(Y,T_enc_norm_manual,label="Encodage normal",color="red",ls=':')
144|     plt.plot(Y,T_dec_norm,label="Décodage normal",ls='--',color="red")
145|     plt.legend(loc='upper left')
146|     plt.show()
147|
148|     plt.figure(2)
149|     plt.title("Temps en fonction de la taille des paquets (2**k)")
```

```
150|     plt.plot(Y,T_enc_norm_manual,label="Encodage normal",color="red",ls=':')
151|     plt.plot(Y,T_dec_norm,label="Décodage normal",ls='--',color="red")
152|     plt.plot(Y,T_enc_ext_manual,label="Encodage étendu",color="green",ls=':')
153|     plt.plot(Y,T_dec_ext,label="Décodage étendu",ls='--',color="green")
154|     plt.legend(loc='upper left')
155|     plt.show()
156|
157|     plt.figure(3)
158|     plt.title("Temps en fonction de la taille des paquets (2**k)")
159|     plt.plot(Y,T_enc_norm_gen,label="Encodage normal (Matrice Génératrice)",color="red")
160|     plt.plot(Y,T_enc_norm_manual,label="Encodage normal (Méthode
Manuelle)",color="red",ls=':')
161|     plt.plot(Y,T_dec_norm,label="Décodage normal",ls='--',color="red")
162|     plt.plot(Y,T_enc_ext_gen,label="Encodage étendu (Matrice Génératrice)",color="green")
163|     plt.plot(Y,T_enc_ext_manual,label="Encodage étendu (Méthode
Manuelle)",color="green",ls=':')
164|     plt.plot(Y,T_dec_ext,label="Décodage étendu",ls='--',color="green")
165|     plt.legend(loc='upper left')
166|     plt.show()
167|
168|     plt.figure(4)
169|     plt.title("Temps en fonction de la taille des paquets (2**k)")
170|     plt.plot(Y,T_enc_norm_gen,label="Encodage normal (Matrice Génératrice)",color="red")
171|     plt.plot(Y,T_enc_norm_manual,label="Encodage normal (Méthode
Manuelle)",color="red",ls=':')
172|     plt.plot(Y,T_dec_norm,label="Décodage normal",ls='--',color="red")
173|     plt.plot(Y,T_enc_ext_gen,label="Encodage étendu (Matrice Génératrice)",color="green")
174|     plt.plot(Y,T_enc_ext_manual,label="Encodage étendu (Méthode
Manuelle)",color="green",ls=':')
175|     plt.plot(Y,T_dec_ext,label="Décodage étendu",ls='--',color="green")
176|     plt.plot(Y,T_enc_xor,label="Encodage (Méthode XOR)",color="blue",ls=':')
177|     plt.plot(Y,T_dec_xor,label="Décodage (Méthode XOR)",color="blue",ls='--')
178|     plt.legend(loc='upper left')
179|     plt.show()
180|
181| def bitRatesGraphs(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor):
182|     br_enc_norm_gen = calculateBitRate(T_enc_norm_gen)
183|     br_enc_norm_manual = calculateBitRate(T_enc_norm_manual)
184|     br_dec_norm = calculateBitRate(T_dec_norm)
185|     br_enc_ext_gen = calculateBitRate(T_enc_ext_gen)
186|     br_enc_ext_manual = calculateBitRate(T_enc_ext_manual)
187|     br_dec_ext = calculateBitRate(T_dec_ext)
188|     br_enc_xor = calculateBitRate(T_enc_xor)
189|     br_dec_xor = calculateBitRate(T_dec_xor)
190|
191|     plt.figure(11)
192|     plt.title("Débit en fonction de la taille des paquets (2**k)")
193|     plt.plot(Y,br_enc_norm_manual,label="Encodage normal",color="red",ls=':')
194|     plt.plot(Y,br_dec_norm,label="Décodage normal",ls='--',color="red")
195|     plt.legend(loc='upper left')
196|     plt.show()
197|
198|     plt.figure(12)
199|     plt.title("Débit en fonction de la taille des paquets (2**k)")
200|     plt.plot(Y,br_enc_norm_manual,label="Encodage normal",color="red",ls=':')
201|     plt.plot(Y,br_dec_norm,label="Décodage normal",ls='--',color="red")
202|     plt.plot(Y,br_enc_ext_manual,label="Encodage étendu",color="green",ls=':')
203|     plt.plot(Y,br_dec_ext,label="Décodage étendu",ls='--',color="green")
204|     plt.legend(loc='upper left')
205|     plt.show()
206|
207|     plt.figure(13)
208|     plt.title("Débit en fonction de la taille des paquets (2**k)")
209|     plt.plot(Y,br_enc_norm_gen,label="Encodage normal (Matrice Génératrice)",color="red")
210|     plt.plot(Y,br_enc_norm_manual,label="Encodage normal (Méthode
Manuelle)",color="red",ls=':')
211|     plt.plot(Y,br_dec_norm,label="Décodage normal",ls='--',color="red")
212|     plt.plot(Y,br_enc_ext_gen,label="Encodage étendu (Matrice Génératrice)",color="green")
213|     plt.plot(Y,br_enc_ext_manual,label="Encodage étendu (Méthode
Manuelle)",color="green",ls=':')
214|     plt.plot(Y,br_dec_ext,label="Décodage étendu",ls='--',color="green")
215|     plt.legend(loc='upper left')
216|     plt.show()
217|
218|     plt.figure(14)
219|     plt.title("Débit en fonction de la taille des paquets (2**k)")
```

```
220|     plt.plot(Y,br_enc_norm_gen,label="Encodage normal (Matrice Génératrice)",color="red")
221|     plt.plot(Y,br_enc_norm_manual,label="Encodage normal (Méthode
Manuelle)",color="red",ls=':')
222|     plt.plot(Y,br_dec_norm,label="Décodage normal",ls='--',color="red")
223|     plt.plot(Y,br_enc_ext_gen,label="Encodage étendu (Matrice Génératrice)",color="green")
224|     plt.plot(Y,br_enc_ext_manual,label="Encodage étendu (Méthode
Manuelle)",color="green",ls=':')
225|     plt.plot(Y,br_dec_ext,label="Décodage étendu",ls='--',color="green")
226|     plt.plot(Y,br_enc_xor,label="Encodage (Méthode XOR)",color="blue",ls=':')
227|     plt.plot(Y,br_dec_xor,label="Décodage (Méthode XOR)",color="blue",ls='--')
228|     plt.legend(loc='upper left')
229|     plt.show()
230|
231| def methodRatesGraph(Y,T_enc_norm_gen,T_enc_norm_manual,T_enc_ext_gen,T_enc_ext_manual):
232|     r_norm = divideLists(T_enc_norm_gen,T_enc_norm_manual)
233|     r_ext = divideLists(T_enc_ext_gen,T_enc_ext_manual)
234|     plt.figure(21)
235|     plt.title("Rapport temporel entre la méthode génératrice et la méthode manuelle")
236|     plt.plot(Y,r_norm,label="Encodage normal", color="red")
237|     plt.plot(Y,r_ext,label="Encodage étendu", color="green")
238|     plt.legend()
239|     plt.show()
240|
241| def extendedRatesGraph(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
T_enc_ext_manual,T_dec_ext):
242|     r_enc_gen = divideLists(T_enc_ext_gen,T_enc_norm_gen)
243|     r_enc_man = divideLists(T_enc_ext_manual,T_enc_norm_manual)
244|     r_dec = divideLists(T_dec_ext,T_dec_norm)
245|     plt.figure(22)
246|     plt.title("Rapport temporel entre le code étendu et le code classique")
247|     plt.plot(Y,r_enc_gen,label="Encodage (Matrice Génératrice)", color="red")
248|     plt.plot(Y,r_enc_man,label="Encodage (Méthode Manuelle)", color="green")
249|     plt.plot(Y,r_dec,label="Décodage", color="blue")
250|     plt.legend()
251|     plt.show()
252|
253|
254| def Graphs(factor_offset = 1,nb_test = 2000):
255|     T_enc_norm_manual  =[]
256|     T_enc_norm_gen  =[]
257|     T_enc_ext_manual = []
258|     T_enc_ext_gen  =[]
259|     T_enc_xor  =[]
260|     T_dec_xor  =[]
261|     T_dec_norm  =[]
262|     T_dec_ext = []
263|     Y = np.arange(2,2+factor_offset)
264|
265|     t1 = time.time()
266|     for k in Y:
267|         print(k)
268|         code = HammingCode(k)
269|         ext_code = ExtendedHammingCode(k)
270|         xor_code = HammingCodeXOR(k)
271|
272|         t_manual = measureManualMethod(code,nb_test)
273|         t_gen = measureGeneratorMethod(code,nb_test)
274|         t_ext_manual = measureManualMethod(ext_code,nb_test)
275|         t_ext_gen = measureGeneratorMethod(ext_code,nb_test)
276|         t_xor = measureXORMethod(xor_code,nb_test)
277|
278|         T_enc_norm_gen.append(t_gen[0])
279|         T_enc_norm_manual.append(t_manual[0])
280|         T_enc_ext_gen.append(t_ext_gen[0])
281|         T_enc_ext_manual.append(t_ext_manual[0])
282|         T_enc_xor.append(t_xor[0])
283|         T_dec_xor.append(t_xor[1])
284|         T_dec_norm.append(t_manual[1])
285|         T_dec_ext.append(t_ext_manual[1])
286|
287|     t2 = time.time()
288|     print(t2-t1)
289|     saveTimesToCSV(T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
290|                    T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor)
291|
292|     rawTimesGraphs(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
293|                    T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor)
```

```
294|
295|     bitRatesGraphs(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,T_enc_ext_gen,
296|                    T_enc_ext_manual,T_dec_ext,T_enc_xor,T_dec_xor)
297|
298|     methodRatesGraph(Y,T_enc_norm_gen,T_enc_norm_manual,T_enc_ext_gen,
299|                    T_enc_ext_manual)
300|     extendedRatesGraph(Y,T_enc_norm_gen,T_enc_norm_manual,T_dec_norm,
301|                        T_enc_ext_gen, T_enc_ext_manual,T_dec_ext)
302|
303|
304| Graphs(8,1500)
305|
306| def CorrectionUneErreur():
307|     h = HammingCode(3)
308|     b = generateRandomBits(4)
309|     print("Données à encoder: ",b)
310|     d = h.getEncryptedDataChunk(b)
311|     print("Bloc encodé: ",d)
312|     print("Bits ajoutés: [X  X  -  X  -  -  -]")
313|     err_d,pos = changeOneRandomBit(d)
314|     print("Bloc encodé erroné: ",err_d)
315|     print("Erreur en position :",pos)
316|     corr_b = h.getDecryptedHammingBlock(err_d)
317|     print("Données récupérées du bloc erroné: ",corr_b)
318|
319| def DeuxErreursNormal():
320|     h = HammingCode(3)
321|     b = generateRandomBits(4)
322|     print("Données à encoder: ",b)
323|     d = h.getEncryptedDataChunk(b)
324|     print("Bloc encodé: ",d)
325|     print("Bits ajoutés: [X  X  -  X  -  -  -]")
326|     err_d = d[:]
327|     err_d[0] = 1 - err_d[0]
328|     err_d[5] = 1 - err_d[5]
329|     print("Bloc encodé erroné: ",err_d)
330|     print("Erreur en position : 0 et 5")
331|     corr_b = h.getDecryptedHammingBlock(err_d)
332|     print("Données récupérées du bloc erroné: ",corr_b)
333|
334| def DetectionDeuxErreursEtendu():
335|     h = ExtendedHammingCode(3)
336|     b = generateRandomBits(4)
337|     print("Données à encoder: ",b)
338|     d = h.getEncryptedDataChunk(b)
339|     print("Bloc encodé: ",d)
340|     print("Bits ajoutés: [X  X  X  -  X  -  -  -]")
341|     err_d = d[:]
342|     err_d[0] = 1 - err_d[0]
343|     err_d[5] = 1 - err_d[5]
344|     print("Bloc encodé erroné: ",err_d)
345|     print("Erreur en position : 0 et 5")
346|     corr_b = h.getDecryptedHammingBlock(err_d)
347|     print("Données récupérées du bloc erroné: ",corr_b)
348|
349| CorrectionUneErreur()
350| DeuxErreursNormal()
351| DetectionDeuxErreursEtendu()
```