Estimation de la latence et de la capacité d'une connexion à partir de mesures asymétriques (sujet 4)

April 3, 2020

1 Estimation de la latence et de la capacité d'une connexion à partir de mesures asymétriques

Nous cherchons dans cette étude à décrire la performance d'une connexion de réseau.

Nous commençons par importer les packages basiques.

[51]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

1.1 Modification du jeu de données

Le jeu de données examine une connexion courte à l'intérieur d'un campus. Il est disponible avec l'URL suivante :

[52]:
```
data_url = "http://mescal.imag.fr/membres/arnaud.legrand/teaching/2014/
 ↪RICM4_EP_ping/liglab2.log.gz"
```

Une ligne est de la forme suivante : [1421761682.052172] 665 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=22.5 ms

La date de la mesure exprimée en seconde à compté du 1er janvier 1970 se trouve entre crochet. Puis, nous trouvons la taille du message en octets, suivie par le nom de la machine cible et son adresse IP entre parenthèse. La machine cible est normalement identique pour toutes les lignes à l'intérieur d'un jeu de données. Deux indications, icmp_seq et ttl, sont ensuite fournies mais elles ne nous seront pas utiles. Le temps d'envoi en milliseconde pour un aller-retour se trouve à la fin de la ligne.

Nous pouvons importer ces données à l'aide de pandas. Nous utilisons l'espace pour délimiter les colonnes. Les colonnes n'ont pas de titre pour le moment.

[53]:
```
raw_data = pd.read_table(data_url,sep=' ',header=None)
raw_data
```

[53]:

	0	1	2	3	4	\
0	[1421761682.052172]	665	bytes	from	lig-publig.imag.fr	
1	[1421761682.277315]	1373	bytes	from	lig-publig.imag.fr	
2	[1421761682.502054]	262	bytes	from	lig-publig.imag.fr	
3	[1421761682.729257]	1107	bytes	from	lig-publig.imag.fr	
4	[1421761682.934648]	1128	bytes	from	lig-publig.imag.fr	
5	[1421761683.160397]	489	bytes	from	lig-publig.imag.fr	
6	[1421761683.443055]	1759	bytes	from	lig-publig.imag.fr	
7	[1421761683.672157]	1146	bytes	from	lig-publig.imag.fr	
8	[1421761683.899933]	884	bytes	from	lig-publig.imag.fr	
9	[1421761684.122687]	1422	bytes	from	lig-publig.imag.fr	
10	[1421761684.344135]	1180	bytes	from	lig-publig.imag.fr	
11	[1421761684.566271]	999	bytes	from	lig-publig.imag.fr	
12	[1421761684.770828]	21	bytes	from	lig-publig.imag.fr	
13	[1421761684.998504]	1020	bytes	from	lig-publig.imag.fr	
14	[1421761685.205172]	71	bytes	from	lig-publig.imag.fr	
15	[1421761685.414106]	34	bytes	from	lig-publig.imag.fr	
16	[1421761685.620117]	1843	bytes	from	lig-publig.imag.fr	
17	[1421761685.824949]	407	bytes	from	lig-publig.imag.fr	
18	[1421761686.029177]	356	bytes	from	lig-publig.imag.fr	
19	[1421761686.234464]	1511	bytes	from	lig-publig.imag.fr	
20	[1421761686.438772]	587	bytes	from	lig-publig.imag.fr	
21	[1421761686.643208]	809	bytes	from	lig-publig.imag.fr	
22	[1421761686.848323]	1364	bytes	from	lig-publig.imag.fr	
23	[1421761687.053400]	1153	bytes	from	lig-publig.imag.fr	
24	[1421761687.257704]	853	bytes	from	lig-publig.imag.fr	
25	[1421761687.463275]	1510	bytes	from	lig-publig.imag.fr	
26	[1421761687.668423]	123	bytes	from	lig-publig.imag.fr	
27	[1421761687.874230]	1966	bytes	from	lig-publig.imag.fr	
28	[1421761688.078667]	933	bytes	from	lig-publig.imag.fr	
29	[1421761688.283655]	922	bytes	from	lig-publig.imag.fr	
…	…	…	…	…	…	
44383	[1421771180.743715]	1772	bytes	from	lig-publig.imag.fr	
44384	[1421771180.949053]	41	bytes	from	lig-publig.imag.fr	
44385	[1421771181.155685]	1944	bytes	from	lig-publig.imag.fr	
44386	[1421771181.362095]	400	bytes	from	lig-publig.imag.fr	
44387	[1421771181.569409]	226	bytes	from	lig-publig.imag.fr	
44388	[1421771181.780805]	466	bytes	from	lig-publig.imag.fr	
44389	[1421771181.998869]	350	bytes	from	lig-publig.imag.fr	
44390	[1421771182.248969]	1829	bytes	from	lig-publig.imag.fr	
44391	[1421771182.512386]	1954	bytes	from	lig-publig.imag.fr	
44392	[1421771182.717961]	1074	bytes	from	lig-publig.imag.fr	
44393	[1421771182.923292]	46	bytes	from	lig-publig.imag.fr	
44394	[1421771183.129965]	1844	bytes	from	lig-publig.imag.fr	
44395	[1421771183.335449]	645	bytes	from	lig-publig.imag.fr	
44396	[1421771183.540901]	444	bytes	from	lig-publig.imag.fr	
44397	[1421771183.747983]	1940	bytes	from	lig-publig.imag.fr	

```
44398  [1421771183.954099]  1411  bytes  from  lig-publig.imag.fr
44399  [1421771184.159879]    49  bytes  from  lig-publig.imag.fr
44400  [1421771184.365815]   420  bytes  from  lig-publig.imag.fr
44401  [1421771184.571516]   227  bytes  from  lig-publig.imag.fr
44402  [1421771184.777325]   947  bytes  from  lig-publig.imag.fr
44403  [1421771184.983905]  1960  bytes  from  lig-publig.imag.fr
44404  [1421771185.188976]   531  bytes  from  lig-publig.imag.fr
44405  [1421771185.394275]   374  bytes  from  lig-publig.imag.fr
44406  [1421771185.600745]  1503  bytes  from  lig-publig.imag.fr
44407  [1421771185.805877]   572  bytes  from  lig-publig.imag.fr
44408  [1421771186.011910]  1338  bytes  from  lig-publig.imag.fr
44409  [1421771186.222729]  1515  bytes  from  lig-publig.imag.fr
44410  [1421771186.429007]  1875  bytes  from  lig-publig.imag.fr
44411  [1421771186.634747]  1006  bytes  from  lig-publig.imag.fr
44412  [1421771186.840222]  1273  bytes  from  lig-publig.imag.fr
```

	5	6	7	8	9
0	(129.88.11.7):	icmp_seq=1	ttl=60	time=22.5	ms
1	(129.88.11.7):	icmp_seq=1	ttl=60	time=21.2	ms
2	(129.88.11.7):	icmp_seq=1	ttl=60	time=21.2	ms
3	(129.88.11.7):	icmp_seq=1	ttl=60	time=23.3	ms
4	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.41	ms
5	(129.88.11.7):	icmp_seq=1	ttl=60	time=21.9	ms
6	(129.88.11.7):	icmp_seq=1	ttl=60	time=78.7	ms
7	(129.88.11.7):	icmp_seq=1	ttl=60	time=25.1	ms
8	(129.88.11.7):	icmp_seq=1	ttl=60	time=24.0	ms
9	(129.88.11.7):	icmp_seq=1	ttl=60	time=19.5	ms
10	(129.88.11.7):	icmp_seq=1	ttl=60	time=18.0	ms
11	(129.88.11.7):	icmp_seq=1	ttl=60	time=18.8	ms
12	(129.88.11.7):	icmp_seq=1	ttl=60	NaN	NaN
13	(129.88.11.7):	icmp_seq=1	ttl=60	time=24.3	ms
14	(129.88.11.7):	icmp_seq=1	ttl=60	time=3.45	ms
15	(129.88.11.7):	icmp_seq=1	ttl=60	time=5.85	ms
16	(129.88.11.7):	icmp_seq=1	ttl=60	time=2.31	ms
17	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.14	ms
18	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.10	ms
19	(129.88.11.7):	icmp_seq=1	ttl=60	time=2.18	ms
20	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.27	ms
21	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.33	ms
22	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.51	ms
23	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.44	ms
24	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.30	ms
25	(129.88.11.7):	icmp_seq=1	ttl=60	time=2.17	ms
26	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.21	ms
27	(129.88.11.7):	icmp_seq=1	ttl=60	time=2.20	ms
28	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.34	ms
29	(129.88.11.7):	icmp_seq=1	ttl=60	time=1.42	ms

```
…                …          …      …          …   …
44383  (129.88.11.7):  icmp_seq=1  ttl=60  time=28.8   ms
44384  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.14   ms
44385  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.32   ms
44386  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.98   ms
44387  (129.88.11.7):  icmp_seq=1  ttl=60  time=3.01   ms
44388  (129.88.11.7):  icmp_seq=1  ttl=60  time=7.45   ms
44389  (129.88.11.7):  icmp_seq=1  ttl=60  time=13.5   ms
44390  (129.88.11.7):  icmp_seq=1  ttl=60  time=45.9   ms
44391  (129.88.11.7):  icmp_seq=1  ttl=60  time=58.5   ms
44392  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.45   ms
44393  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.11   ms
44394  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.26   ms
44395  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.24   ms
44396  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.25   ms
44397  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.46   ms
44398  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.47   ms
44399  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.21   ms
44400  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.55   ms
44401  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.22   ms
44402  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.34   ms
44403  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.43   ms
44404  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.19   ms
44405  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.14   ms
44406  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.19   ms
44407  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.29   ms
44408  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.47   ms
44409  (129.88.11.7):  icmp_seq=1  ttl=60  time=7.02   ms
44410  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.33   ms
44411  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.61   ms
44412  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.35   ms

[44413 rows x 10 columns]
```

Comme nous pouvons le voir, les colonnes 2, 3, 4, 5, 6, 7 et 9 ne sont pas utiles à notre études. Nous pouvons les laisser car elle n'impacteront pas note étude. Par contre, la ligne 12 nous montre que des données sont manquantes. Nous allons voir combien de lignes a des données manquantes.

[54]:
```
np.shape(raw_data[raw_data.isnull().any(axis=1)])[0]
```

[54]: 377

377 lignes parmis les 44413 n'ont pas de données. Nous pouvons les supprimer sans qu'elles n'aient une grande insidence sur l'étude.

[55]:
```
data = raw_data.dropna().copy()
data
```

[55]:

	0	1	2	3	4 \
0	[1421761682.052172]	665	bytes	from	lig-publig.imag.fr
1	[1421761682.277315]	1373	bytes	from	lig-publig.imag.fr
2	[1421761682.502054]	262	bytes	from	lig-publig.imag.fr
3	[1421761682.729257]	1107	bytes	from	lig-publig.imag.fr
4	[1421761682.934648]	1128	bytes	from	lig-publig.imag.fr
5	[1421761683.160397]	489	bytes	from	lig-publig.imag.fr
6	[1421761683.443055]	1759	bytes	from	lig-publig.imag.fr
7	[1421761683.672157]	1146	bytes	from	lig-publig.imag.fr
8	[1421761683.899933]	884	bytes	from	lig-publig.imag.fr
9	[1421761684.122687]	1422	bytes	from	lig-publig.imag.fr
10	[1421761684.344135]	1180	bytes	from	lig-publig.imag.fr
11	[1421761684.566271]	999	bytes	from	lig-publig.imag.fr
13	[1421761684.998504]	1020	bytes	from	lig-publig.imag.fr
14	[1421761685.205172]	71	bytes	from	lig-publig.imag.fr
15	[1421761685.414106]	34	bytes	from	lig-publig.imag.fr
16	[1421761685.620117]	1843	bytes	from	lig-publig.imag.fr
17	[1421761685.824949]	407	bytes	from	lig-publig.imag.fr
18	[1421761686.029177]	356	bytes	from	lig-publig.imag.fr
19	[1421761686.234464]	1511	bytes	from	lig-publig.imag.fr
20	[1421761686.438772]	587	bytes	from	lig-publig.imag.fr
21	[1421761686.643208]	809	bytes	from	lig-publig.imag.fr
22	[1421761686.848323]	1364	bytes	from	lig-publig.imag.fr
23	[1421761687.053400]	1153	bytes	from	lig-publig.imag.fr
24	[1421761687.257704]	853	bytes	from	lig-publig.imag.fr
25	[1421761687.463275]	1510	bytes	from	lig-publig.imag.fr
26	[1421761687.668423]	123	bytes	from	lig-publig.imag.fr
27	[1421761687.874230]	1966	bytes	from	lig-publig.imag.fr
28	[1421761688.078667]	933	bytes	from	lig-publig.imag.fr
29	[1421761688.283655]	922	bytes	from	lig-publig.imag.fr
30	[1421761688.488688]	24	bytes	from	lig-publig.imag.fr
…	…	…	…	…	…
44383	[1421771180.743715]	1772	bytes	from	lig-publig.imag.fr
44384	[1421771180.949053]	41	bytes	from	lig-publig.imag.fr
44385	[1421771181.155685]	1944	bytes	from	lig-publig.imag.fr
44386	[1421771181.362095]	400	bytes	from	lig-publig.imag.fr
44387	[1421771181.569409]	226	bytes	from	lig-publig.imag.fr
44388	[1421771181.780805]	466	bytes	from	lig-publig.imag.fr
44389	[1421771181.998869]	350	bytes	from	lig-publig.imag.fr
44390	[1421771182.248969]	1829	bytes	from	lig-publig.imag.fr
44391	[1421771182.512386]	1954	bytes	from	lig-publig.imag.fr
44392	[1421771182.717961]	1074	bytes	from	lig-publig.imag.fr
44393	[1421771182.923292]	46	bytes	from	lig-publig.imag.fr
44394	[1421771183.129965]	1844	bytes	from	lig-publig.imag.fr
44395	[1421771183.335449]	645	bytes	from	lig-publig.imag.fr
44396	[1421771183.540901]	444	bytes	from	lig-publig.imag.fr
44397	[1421771183.747983]	1940	bytes	from	lig-publig.imag.fr

```
44398  [1421771183.954099]  1411  bytes  from  lig-publig.imag.fr
44399  [1421771184.159879]    49  bytes  from  lig-publig.imag.fr
44400  [1421771184.365815]   420  bytes  from  lig-publig.imag.fr
44401  [1421771184.571516]   227  bytes  from  lig-publig.imag.fr
44402  [1421771184.777325]   947  bytes  from  lig-publig.imag.fr
44403  [1421771184.983905]  1960  bytes  from  lig-publig.imag.fr
44404  [1421771185.188976]   531  bytes  from  lig-publig.imag.fr
44405  [1421771185.394275]   374  bytes  from  lig-publig.imag.fr
44406  [1421771185.600745]  1503  bytes  from  lig-publig.imag.fr
44407  [1421771185.805877]   572  bytes  from  lig-publig.imag.fr
44408  [1421771186.011910]  1338  bytes  from  lig-publig.imag.fr
44409  [1421771186.222729]  1515  bytes  from  lig-publig.imag.fr
44410  [1421771186.429007]  1875  bytes  from  lig-publig.imag.fr
44411  [1421771186.634747]  1006  bytes  from  lig-publig.imag.fr
44412  [1421771186.840222]  1273  bytes  from  lig-publig.imag.fr
```

```
                  5           6       7          8   9
0      (129.88.11.7):  icmp_seq=1  ttl=60  time=22.5  ms
1      (129.88.11.7):  icmp_seq=1  ttl=60  time=21.2  ms
2      (129.88.11.7):  icmp_seq=1  ttl=60  time=21.2  ms
3      (129.88.11.7):  icmp_seq=1  ttl=60  time=23.3  ms
4      (129.88.11.7):  icmp_seq=1  ttl=60  time=1.41  ms
5      (129.88.11.7):  icmp_seq=1  ttl=60  time=21.9  ms
6      (129.88.11.7):  icmp_seq=1  ttl=60  time=78.7  ms
7      (129.88.11.7):  icmp_seq=1  ttl=60  time=25.1  ms
8      (129.88.11.7):  icmp_seq=1  ttl=60  time=24.0  ms
9      (129.88.11.7):  icmp_seq=1  ttl=60  time=19.5  ms
10     (129.88.11.7):  icmp_seq=1  ttl=60  time=18.0  ms
11     (129.88.11.7):  icmp_seq=1  ttl=60  time=18.8  ms
13     (129.88.11.7):  icmp_seq=1  ttl=60  time=24.3  ms
14     (129.88.11.7):  icmp_seq=1  ttl=60  time=3.45  ms
15     (129.88.11.7):  icmp_seq=1  ttl=60  time=5.85  ms
16     (129.88.11.7):  icmp_seq=1  ttl=60  time=2.31  ms
17     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.14  ms
18     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.10  ms
19     (129.88.11.7):  icmp_seq=1  ttl=60  time=2.18  ms
20     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.27  ms
21     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.33  ms
22     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.51  ms
23     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.44  ms
24     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.30  ms
25     (129.88.11.7):  icmp_seq=1  ttl=60  time=2.17  ms
26     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.21  ms
27     (129.88.11.7):  icmp_seq=1  ttl=60  time=2.20  ms
28     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.34  ms
29     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.42  ms
30     (129.88.11.7):  icmp_seq=1  ttl=60  time=1.12  ms
```

```
…                …          …      …          …  ..
44383  (129.88.11.7):  icmp_seq=1  ttl=60  time=28.8  ms
44384  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.14  ms
44385  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.32  ms
44386  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.98  ms
44387  (129.88.11.7):  icmp_seq=1  ttl=60  time=3.01  ms
44388  (129.88.11.7):  icmp_seq=1  ttl=60  time=7.45  ms
44389  (129.88.11.7):  icmp_seq=1  ttl=60  time=13.5  ms
44390  (129.88.11.7):  icmp_seq=1  ttl=60  time=45.9  ms
44391  (129.88.11.7):  icmp_seq=1  ttl=60  time=58.5  ms
44392  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.45  ms
44393  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.11  ms
44394  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.26  ms
44395  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.24  ms
44396  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.25  ms
44397  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.46  ms
44398  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.47  ms
44399  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.21  ms
44400  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.55  ms
44401  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.22  ms
44402  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.34  ms
44403  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.43  ms
44404  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.19  ms
44405  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.14  ms
44406  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.19  ms
44407  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.29  ms
44408  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.47  ms
44409  (129.88.11.7):  icmp_seq=1  ttl=60  time=7.02  ms
44410  (129.88.11.7):  icmp_seq=1  ttl=60  time=2.33  ms
44411  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.61  ms
44412  (129.88.11.7):  icmp_seq=1  ttl=60  time=1.35  ms

[44036 rows x 10 columns]
```

Nous allons maintenant changer la forme des données. Commençons par les dates où les mesures ont été effectuées (colonne 0). Il faut supprimer les crochets et transformer en nombre décimale.

[56]:
```
def convert_time(time):
    return float(time[1:-1])

data['time_passed'] = [convert_time(time) for time in data[0]]
```

Nous allons maintenant transformer la colonne des tailles des messages en octets (colonne 1) en les changeant en entier.

[57]:
```
def convert_message(message):
    return int(message)
```

```
data['size_message'] = [convert_message(message) for message in data[1]]
```

Il nous reste à transformer la colonne des durées d'envoi (colonne 8) en supprimant le "time=" et en modifiant en nombre décimale.

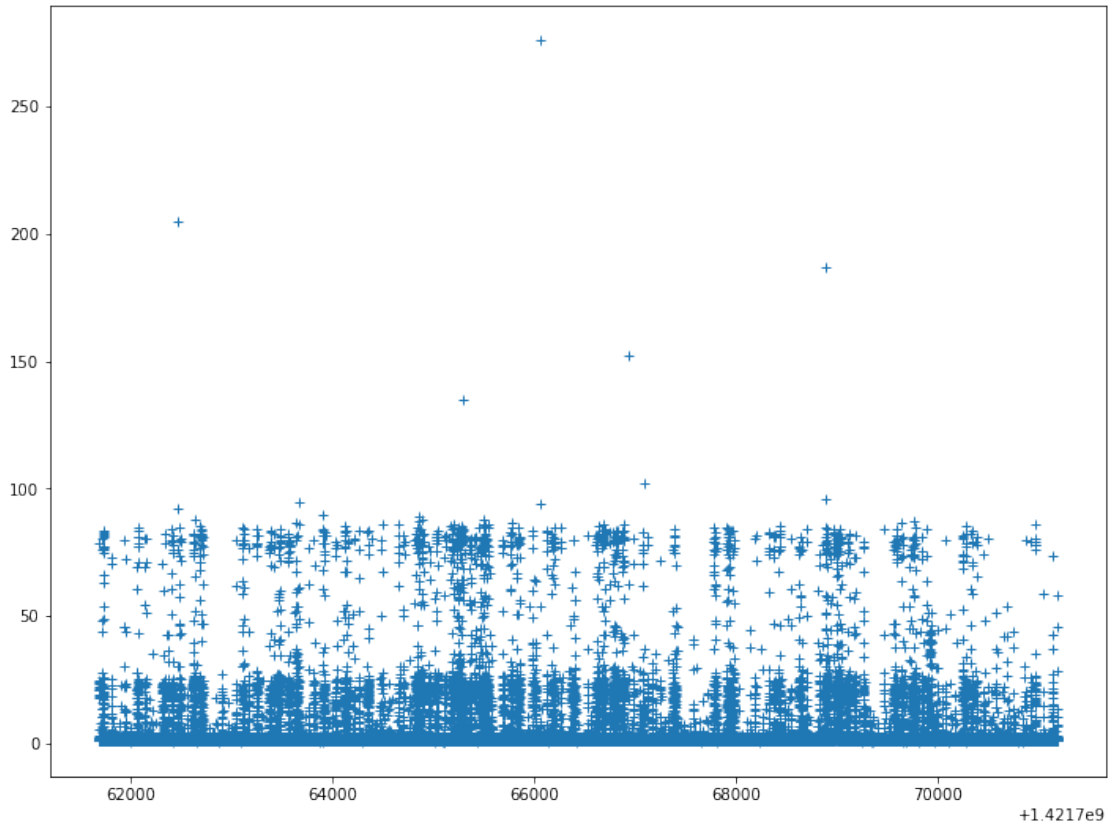
[58]:
```
def convert_duration(duration):
    return float(duration[5:])

data['duration'] = [convert_duration(duration) for duration in data[8]]
```

1.2 Evolution du temps d'envoi

Nous regardons maintenant comment évolue la durée d'envoi au cours du temps. En effet, la durée d'envoi pourrait évoluer avec le temps par exemple à cause de la dégradation du matériel.

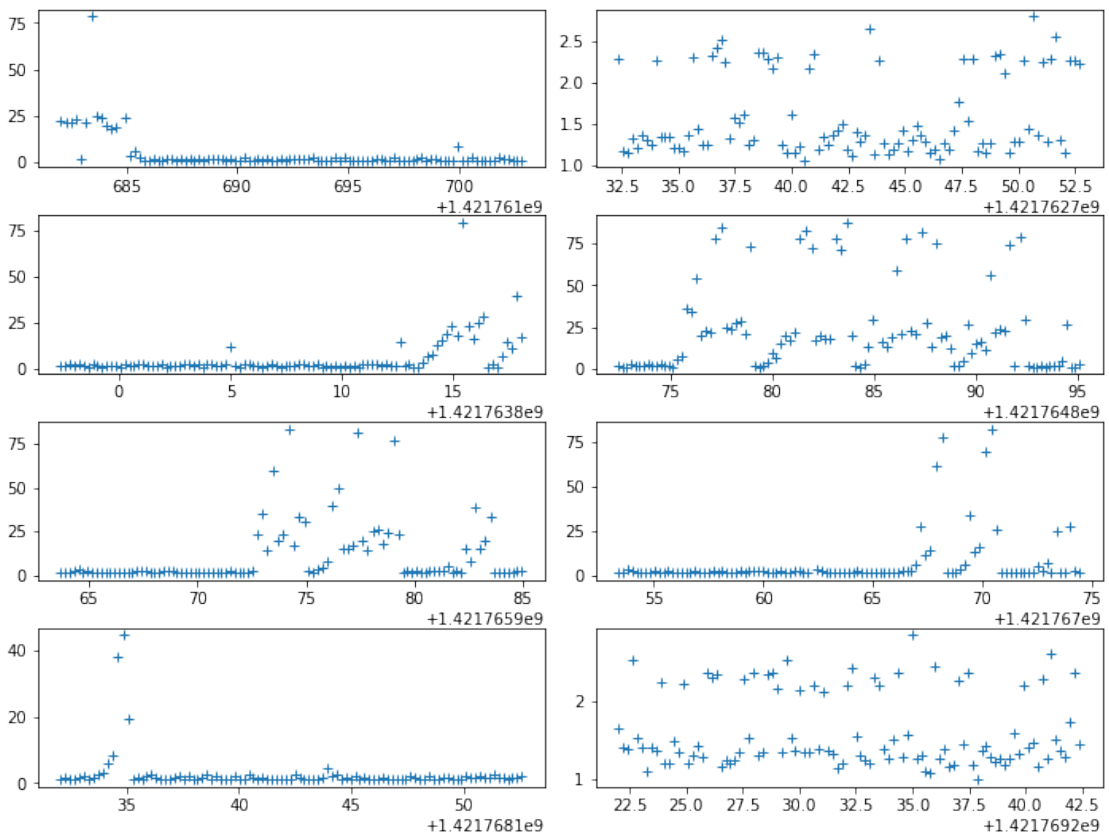
[59]:
```
plt.figure(figsize=(12,9))
plt.plot(data['time_passed'],data['duration'],'+')
```

[59]:
```
[<matplotlib.lines.Line2D at 0x7f6d95e759b0>]
```



A première vue, le temps ne semble pas impacter la durée d'envoi. Les zooms à plusieurs endroits renforcent cette opinion. En effet, les durées d'envoi sont généralement inférieures à 3 ms et des pics apparaissent de temps en temps peu importe le temps écoulé.

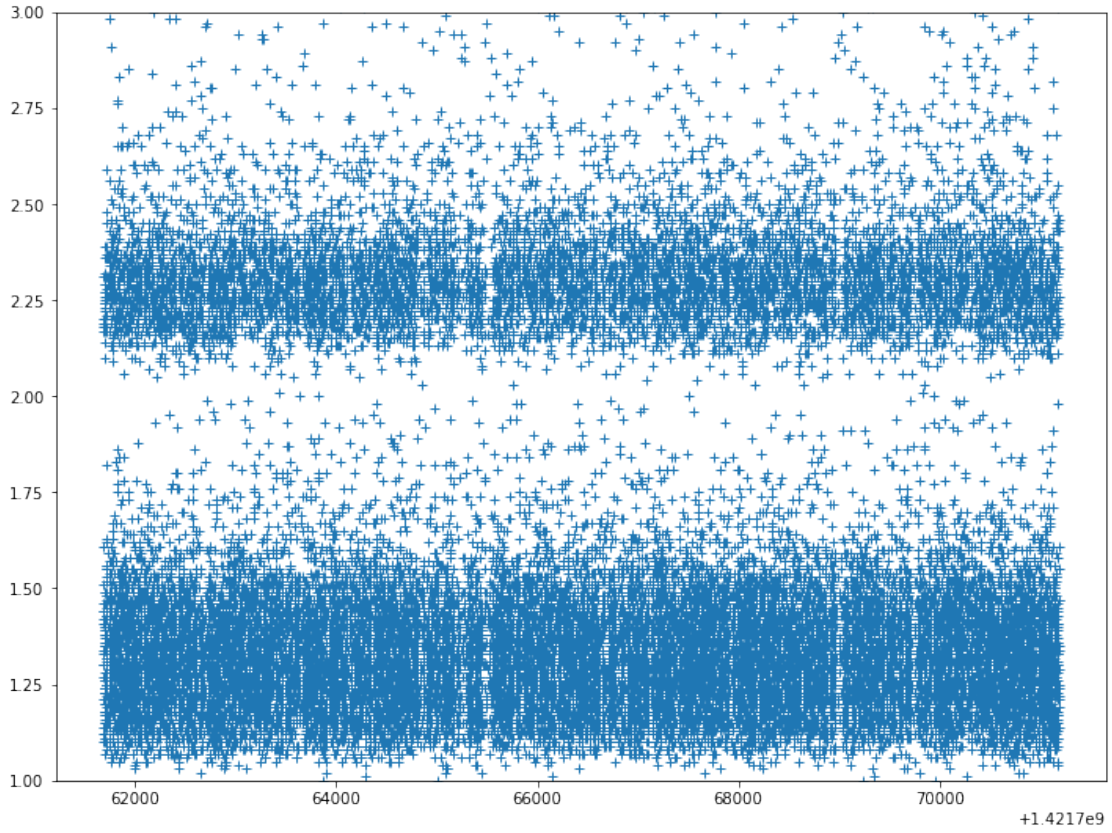
[60]:
```
plt.figure(figsize=(12,9))

plt.subplots_adjust(wspace=0.1, hspace=0.3)

plt.subplot(421)
plt.plot(data['time_passed'][0:100],data['duration'][0:100],'+')

plt.subplot(422)
plt.plot(data['time_passed'][5000:5100],data['duration'][5000:5100],'+')

plt.subplot(423)
plt.plot(data['time_passed'][10000:10100],data['duration'][10000:10100],'+')

plt.subplot(424)
plt.plot(data['time_passed'][15000:15100],data['duration'][15000:15100],'+')

plt.subplot(425)
plt.plot(data['time_passed'][20000:20100],data['duration'][20000:20100],'+')

plt.subplot(426)
plt.plot(data['time_passed'][25000:25100],data['duration'][25000:25100],'+')

plt.subplot(427)
plt.plot(data['time_passed'][30000:30100],data['duration'][30000:30100],'+')

plt.subplot(428)
plt.plot(data['time_passed'][35000:35100],data['duration'][35000:35100],'+')

plt.show()
```



Regardons l'évolution seulement entre 1 et 3. Pour se rendre compte qu'il n'y a pas de lieu entre durée d'envoi et date d'acquisition.

[61]:
```
plt.figure(figsize=(12,9))
plt.plot(data['time_passed'],data['duration'],'+')
plt.ylim(1,3)
```

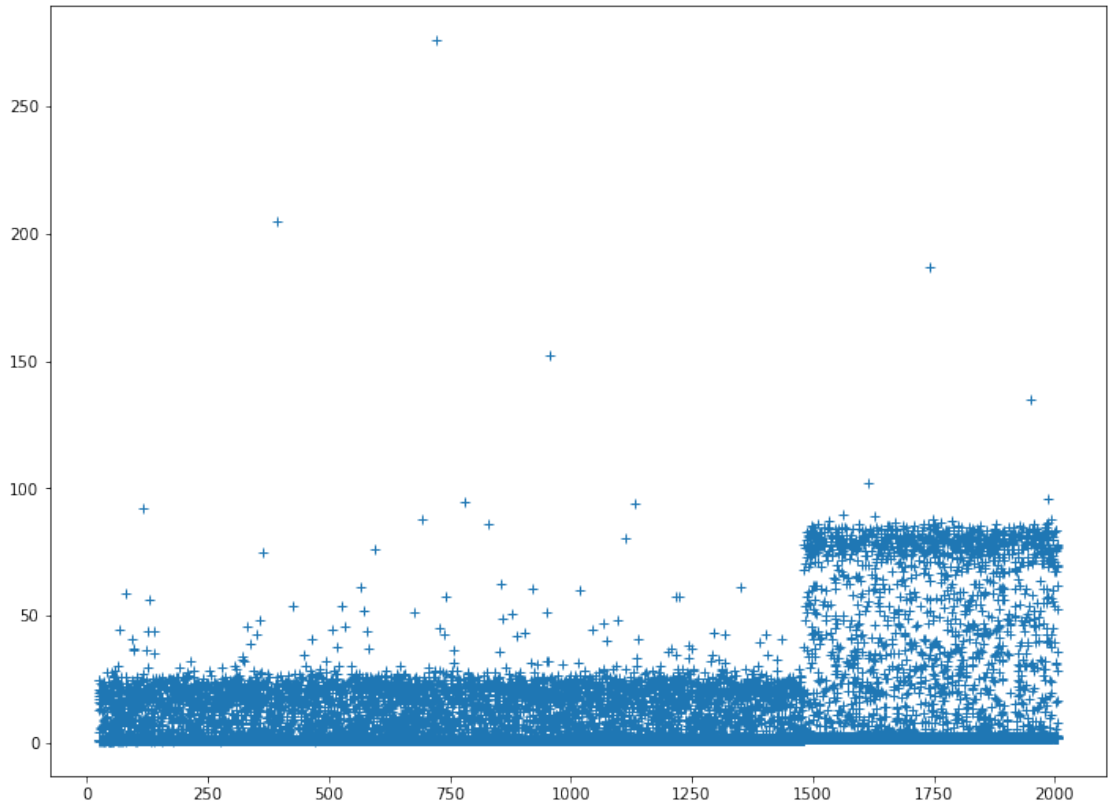
[61]: (1, 3)



Nous allons maintenant voir si la taille du message à une importance sur la durée d'envoi.

[62]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'+')
```

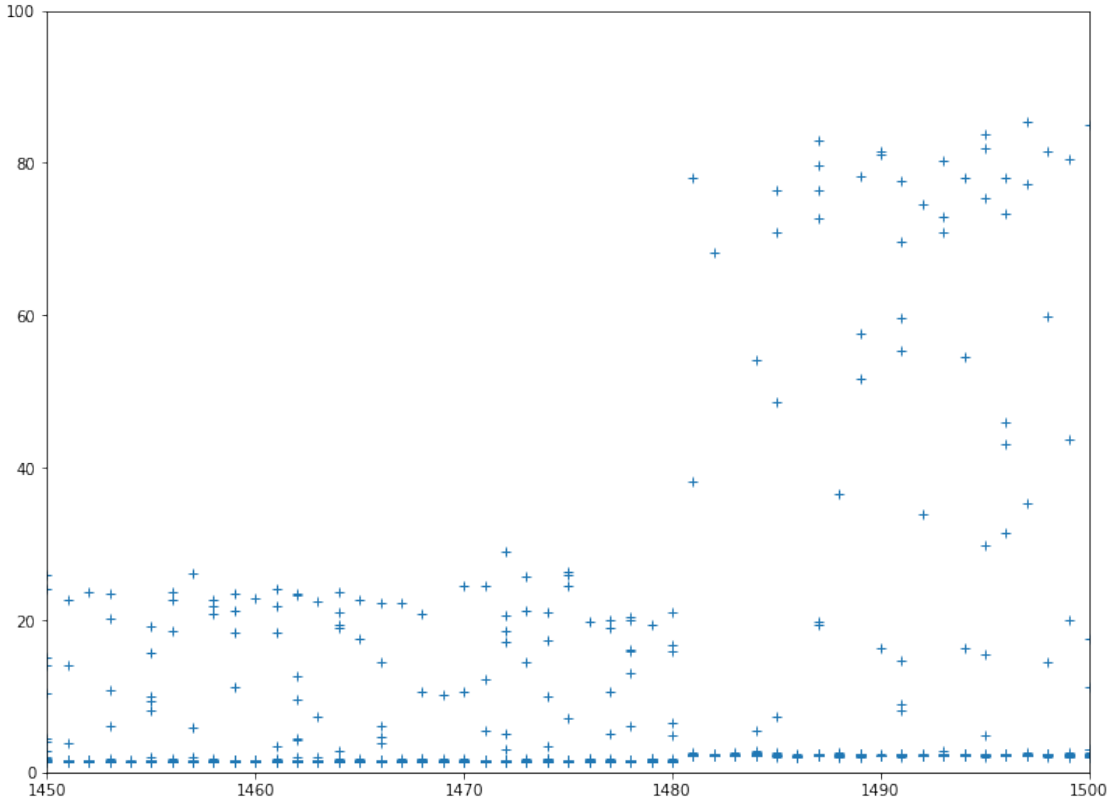
[62]: `[<matplotlib.lines.Line2D at 0x7f6d95e390b8>]`



A première vue, il semble y avoir un seuil sur la taille des messages à partir duquel les durées sont plus importes. Les deux données semblent liées. Estimons graphiquement ce seuil en zoomant sur la zone en question.

[63]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'+')
plt.axis([1450,1500,0,100])
```

[63]: [1450, 1500, 0, 100]



Sur le graphique, nous voyons que le seuil est situé entre 1480 et 1481 octets.

1.3 Régression linéaire des deux zones

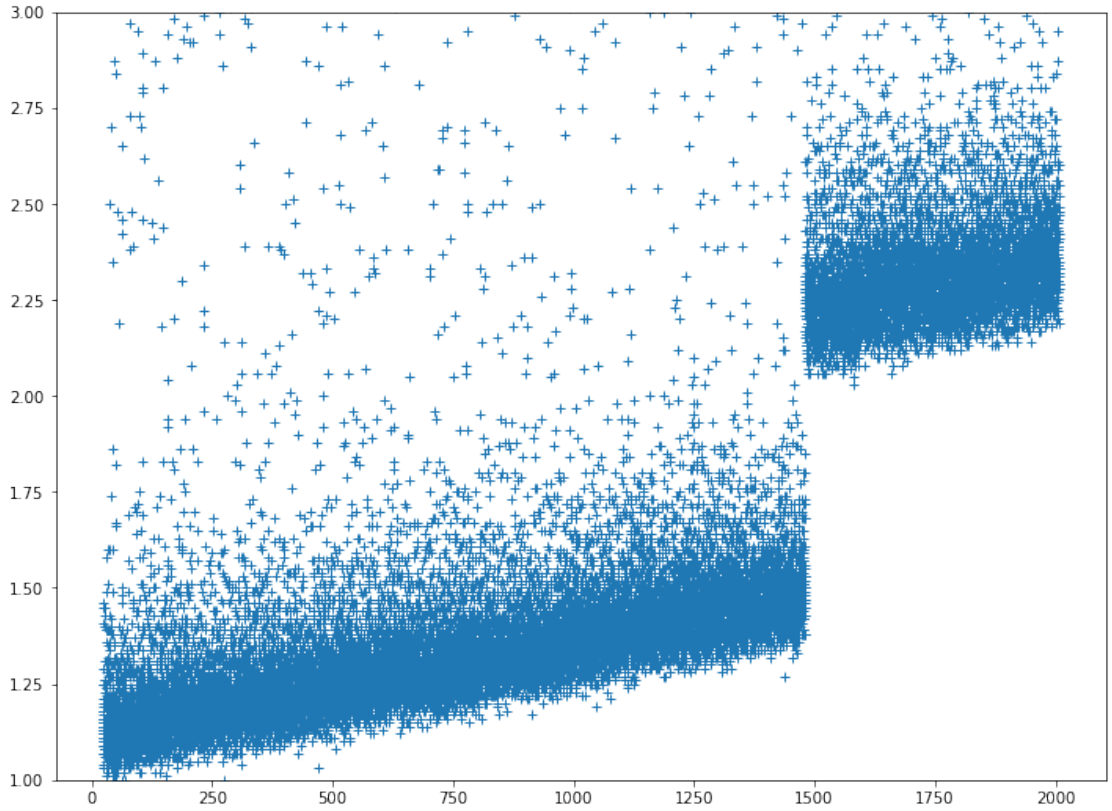
Nous allons évaluer ces deux zones séparément. Il faut donc commencer par séparer les données en deux groupes.

[64]:
```
data_low=data[data['size_message']<=1480]
data_high=data[data['size_message']>=1481]
```

Nous allons maintenant effectué une régression. Pour savoir le type de régression, nous regardons l'évolution de la durée d'envoi par rapport à la taille du message entre 1 et 3, dans la partie où nous avons le plus de données.

[65]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'+')
plt.ylim(1,3)
```

[65]: (1, 3)



Nous nous rendons compte que le plus adapté est une régression linéaire.

[66]:
```
from sklearn.linear_model import LinearRegression

modele_low=LinearRegression()
modele_low.fit(data_low['size_message'].values.reshape(-1,␣
↪1),data_low['duration'])

modele_high=LinearRegression()
modele_high.fit(data_high['size_message'].values.reshape(-1,␣
↪1),data_high['duration'])
```

[66]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Si nous calculons le R^2 score, nous obtenons ceci :

[67]:
```
print(modele_low.score(data_low['size_message'].values.reshape(-1,␣
↪1),data_low['duration']))

print(modele_high.score(data_high['size_message'].values.reshape(-1,␣
↪1),data_high['duration']))
```

```
0.0004513532467899095
0.0003561582707053423 4
```

Les modèles sont très mauvais. Deux idées nous viennent en tête : ne conserver que les durées d'envoi faibles (là où nous avons le plus de données) ou faire la moyenne des durées d'envoi pour chaque taille de messages.

Commençons par le cas de la moyenne. Nous commençons par créer les durées d'envoi moyennes pour chaque taille de message.

[68]:
```
size_low=list(set(data_low['size_message']))

duration_low=np.empty(len(size_low))
for k in range(len(size_low)) :
    duration_low[k]=np.
 ↪mean(data_low['duration'][data_low['size_message']==size_low[k]])

size_high=list(set(data_high['size_message']))

duration_high=np.empty(len(size_high))
for k in range(len(size_high)) :
    duration_high[k]=np.
 ↪mean(data_high['duration'][data_high['size_message']==size_high[k]])
```

Effectuons de nouveau les modèles linéaires.

[69]:
```
modele_low_mean=LinearRegression()
modele_low_mean.fit(np.reshape(size_low,(-1, 1)),duration_low)

modele_high_mean=LinearRegression()
modele_high_mean.fit(np.reshape(size_high,(-1, 1)),duration_high)

print(modele_low_mean.score(np.reshape(size_low,(-1, 1)),duration_low))
print(modele_high_mean.score(np.reshape(size_high,(-1, 1)),duration_high))
```

```
0.010371058628793395
0.006217305568337994
```

Les résultats se sont améliorés mais ne sont toujours pas bons.

Nous allons maintenant essayer de supprimer les durées d'envoi trop élevées. Nous supprimons les durées inférieurs à 2 pour les messages inférieurs à 1480 octets et inférieurs à 3 pour les messages supérieurs à 1481 octets.

[70]:
```
data_low_reduced=data_low[data_low['duration']<=2]
data_high_reduced=data_high[data_high['duration']<=2.6]
```

Regardons combien de données ont été supprimées.

[71]:
```
print(len(data_low_reduced)/len(data_low))
print(len(data_high_reduced)/len(data_high))
```

```
0.8454709645819941
0.8163426862520891
```

Nous avons perdu plus de 15% des données. Notre modèle sera surement mieux adapté aux données réduites mais il ne prendra pas en compte une grande partie des données donc sera moins fiable sur les données complètes.

[72]:
```
modele_low_reduced=LinearRegression()
modele_low_reduced.fit(data_low_reduced['size_message'].values.reshape(-1,␣
↪1),data_low_reduced['duration'])

modele_high_reduced=LinearRegression()
modele_high_reduced.fit(data_high_reduced['size_message'].values.reshape(-1,␣
↪1),data_high_reduced['duration'])

print(modele_low_reduced.score(data_low_reduced['size_message'].values.
↪reshape(-1, 1),data_low_reduced['duration']))
print(modele_high_reduced.score(data_high_reduced['size_message'].values.
↪reshape(-1, 1),data_high_reduced['duration']))
```

```
0.5108687354918388
0.12732864666562382
```

Le modèle dans ce cas n'est pas mauvais. Mais comment se comporte-t-il si nous revenons à la totalité des données ?

[73]:
```
print(modele_low_reduced.score(data_low['size_message'].values.reshape(-1,␣
↪1),data_low['duration']))
print(modele_high_reduced.score(data_high['size_message'].values.reshape(-1,␣
↪1),data_high['duration']))
```

```
-0.11559915224098294
-0.13025418584545734
```

Lorsque nous utilisons toutes les données, ce modèle est inutile.

1.4 Détermination de la latence et de la capacité de la connexion.

Nous allons maintenant déterminer la latence et la capacité pour les trois modèles (complet, moyenne et réduit). Nous avons la relation T=L+S/C où T est la durée, L la latence, S la taille du message et C la capacité. Ayant nos données en millisecondes et octet. La latence s'exprimera en millisecondes et la capacité en octet par millisecondes.

Commençons par le cas du modèle complet.

[74]:
```
print(modele_low.intercept_)
print(1/modele_low.coef_[0])

print(modele_high.intercept_)
print(1/modele_high.coef_[0])
```

```
3.275674199393943
3064.5146168986353
5.289832573707132
387.76039978488814
```

Nous avons une latence de 3.276 millisecondes et une capacité de 3065 octets par millisecondes pour les petits messages et une latence de 5.290 millisecondes et une capacité de 388 octets par millisecondes pour les gros messages.

Etudions le cas du modèle avec les moyennes.

[75]:
```
print(modele_low_mean.intercept_)
print(1/modele_low_mean.coef_[0])

print(modele_high_mean.intercept_)
print(1/modele_high_mean.coef_[0])
```

```
3.2752258866275303
2995.5783326928704
5.666776618402003
424.14888748876956
```

Nous avons une latence de 3.275 millisecondes et une capacité de 2996 octets par millisecondes pour les petits messages et une latence de 5.667 millisecondes et une capacité de 424 octets par millisecondes pour les gros messages.

Regardons maintenant le cas avec le modèle réduit.

[76]:
```
print(modele_low_reduced.intercept_)
print(1/modele_low_reduced.coef_[0])

print(modele_high_reduced.intercept_)
print(1/modele_high_reduced.coef_[0])
```

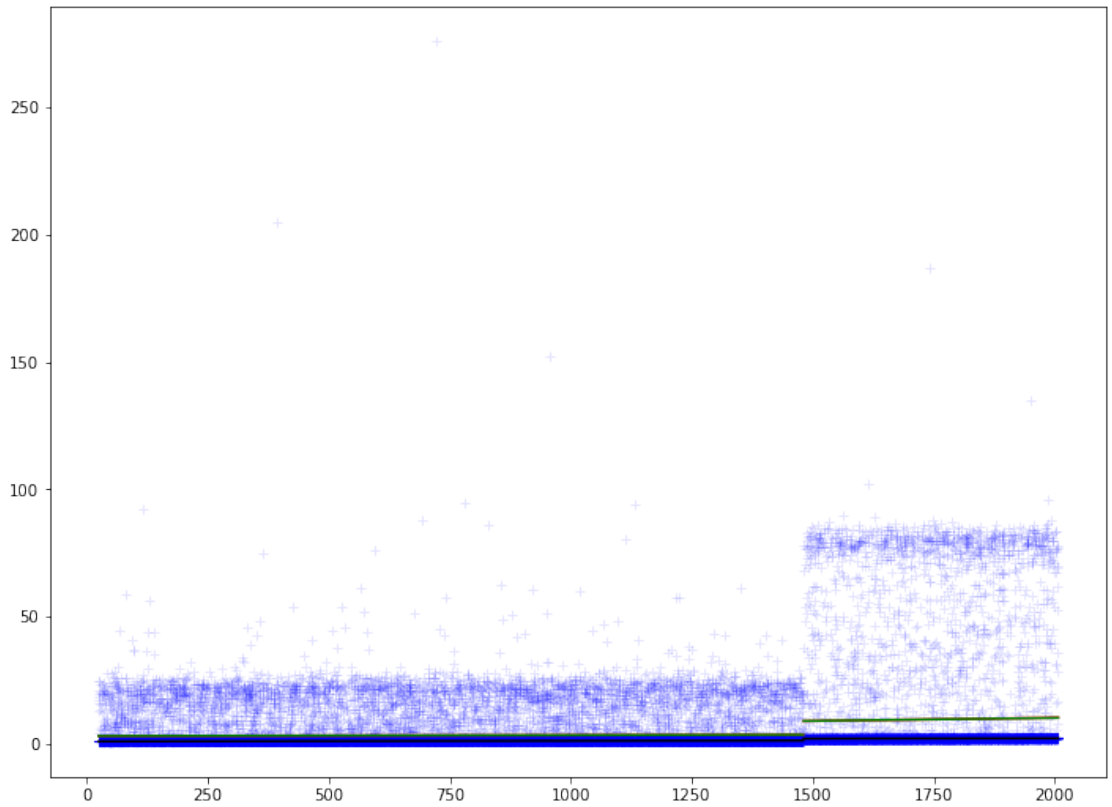
```
1.1466097128990944
4124.963057028056
1.9007662669402698
4370.636833457074
```

Nous avons une latence de 1.147 millisecondes et une capacité de 4125 octets par millisecondes pour les petits messages et une latence de 1.901 millisecondes et une capacité de 4371 octets par millisecondes pour les gros messages.

Traçons maintenant les trois régressions avec les données expérimentales. La courbe rouge correspond au modèle complet, la verte au modèle moyen et la noir au modèle réduit.

[77]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.1)
plt.plot(size_low,modele_low.predict(np.reshape(size_low,(-1, 1))),'r')
plt.plot(size_high,modele_high.predict(np.reshape(size_high,(-1, 1))),'r')
plt.plot(size_low,modele_low_mean.predict(np.reshape(size_low,(-1, 1))),'g')
plt.plot(size_high,modele_high_mean.predict(np.reshape(size_high,(-1, 1))),'g')
plt.plot(size_low,modele_low_reduced.predict(np.reshape(size_low,(-1, 1))),'k')
plt.plot(size_high,modele_high_reduced.predict(np.reshape(size_high,(-1,␣
↪1))),'k')
```

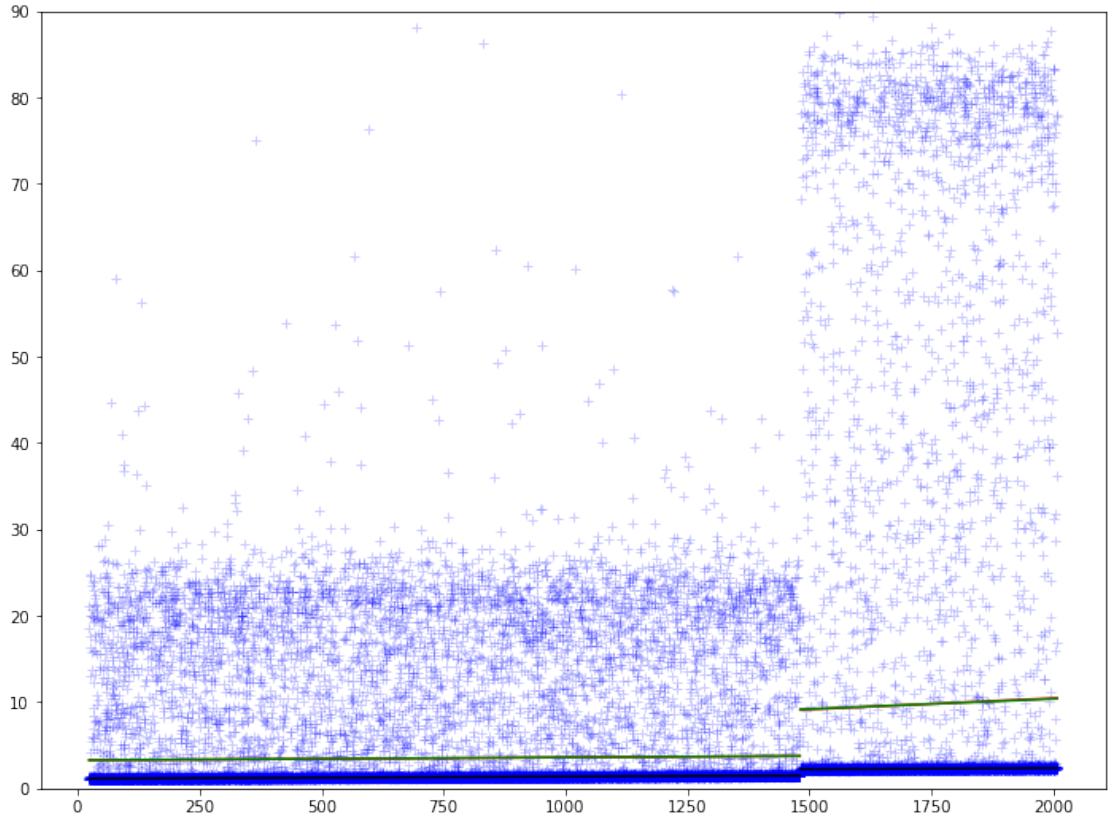
[77]:
```
[<matplotlib.lines.Line2D at 0x7f6d95838a58>]
```

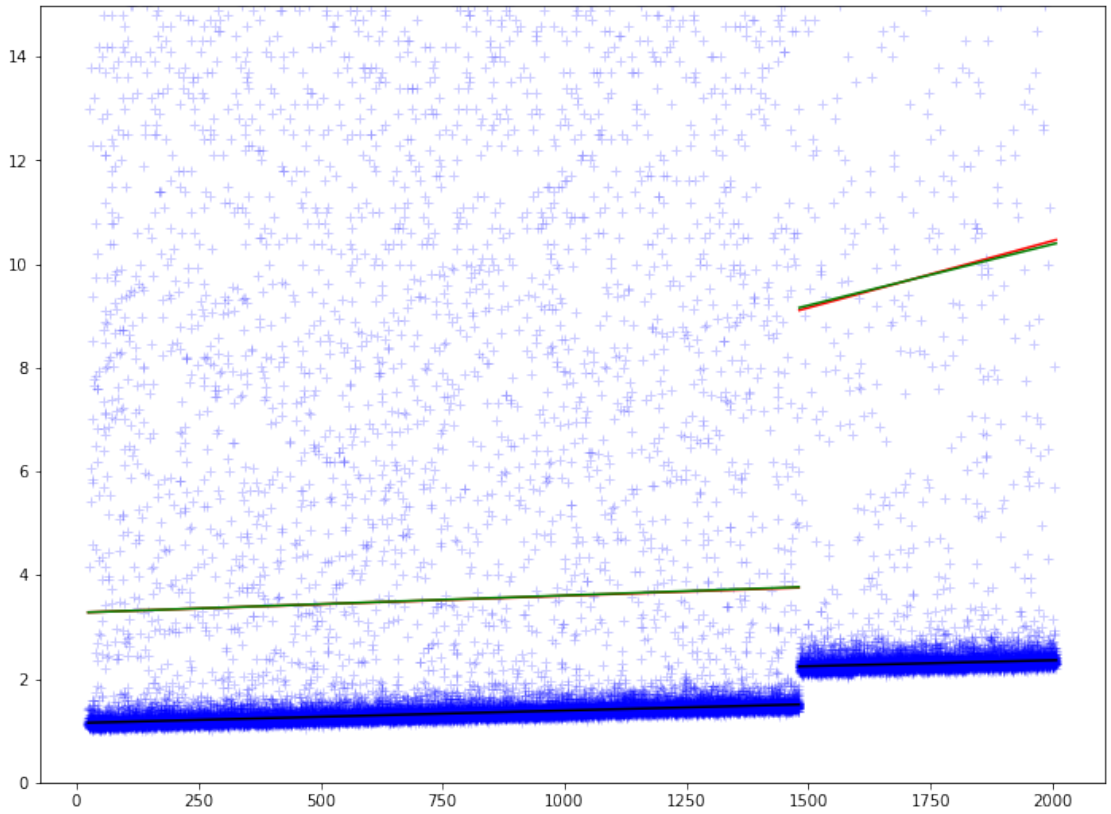


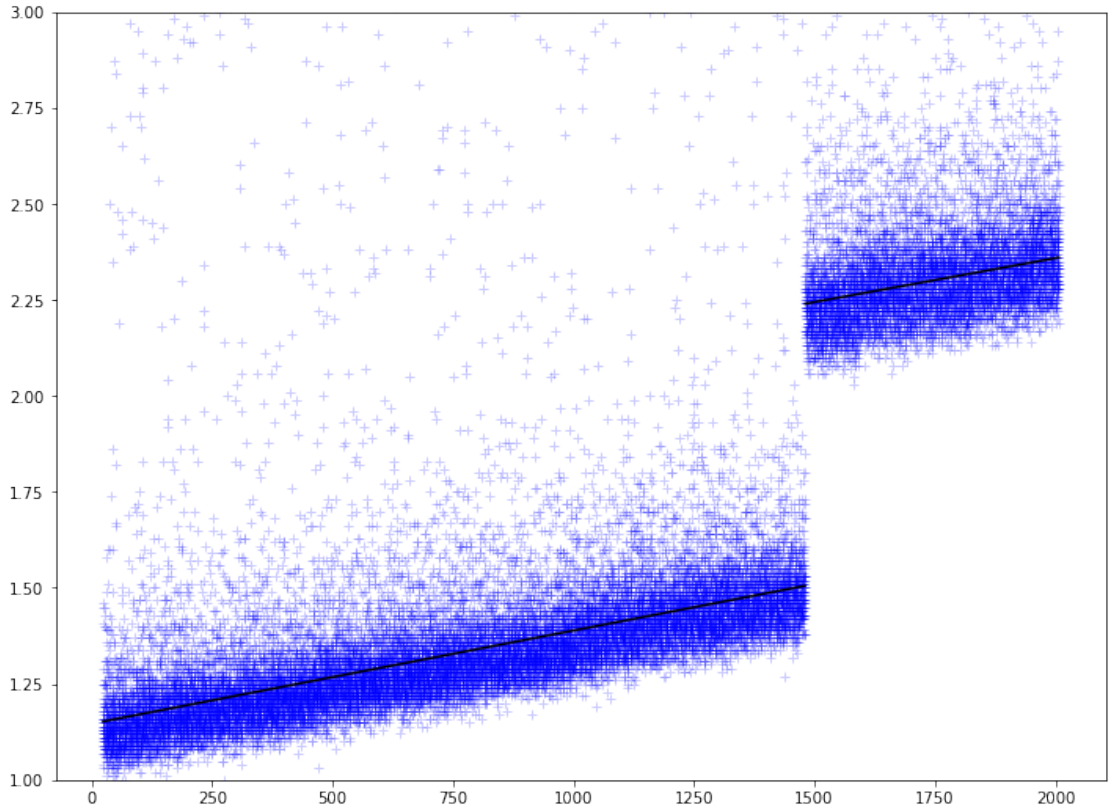
Effectuons un léger zoom, un zoom plus conséquent et un zoom très important.

[78]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.2)
plt.plot(size_low,modele_low.predict(np.reshape(size_low,(-1, 1))),'r')
plt.plot(size_high,modele_high.predict(np.reshape(size_high,(-1, 1))),'r')
plt.plot(size_low,modele_low_mean.predict(np.reshape(size_low,(-1, 1))),'g')
plt.plot(size_high,modele_high_mean.predict(np.reshape(size_high,(-1, 1))),'g')
plt.plot(size_low,modele_low_reduced.predict(np.reshape(size_low,(-1, 1))),'k')
```

```
plt.plot(size_high,modele_high_reduced.predict(np.reshape(size_high,(-1,␣
↪1))),'k')
plt.ylim(0,90)
plt.show()
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.2)
plt.plot(size_low,modele_low.predict(np.reshape(size_low,(-1, 1))),'r')
plt.plot(size_high,modele_high.predict(np.reshape(size_high,(-1, 1))),'r')
plt.plot(size_low,modele_low_mean.predict(np.reshape(size_low,(-1, 1))),'g')
plt.plot(size_high,modele_high_mean.predict(np.reshape(size_high,(-1, 1))),'g')
plt.plot(size_low,modele_low_reduced.predict(np.reshape(size_low,(-1, 1))),'k')
plt.plot(size_high,modele_high_reduced.predict(np.reshape(size_high,(-1,␣
↪1))),'k')
plt.ylim(0,15)
plt.show()
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.2)
plt.plot(size_low,modele_low.predict(np.reshape(size_low,(-1, 1))),'r')
plt.plot(size_high,modele_high.predict(np.reshape(size_high,(-1, 1))),'r')
plt.plot(size_low,modele_low_mean.predict(np.reshape(size_low,(-1, 1))),'g')
plt.plot(size_high,modele_high_mean.predict(np.reshape(size_high,(-1, 1))),'g')
plt.plot(size_low,modele_low_reduced.predict(np.reshape(size_low,(-1, 1))),'k')
plt.plot(size_high,modele_high_reduced.predict(np.reshape(size_high,(-1,␣
↪1))),'k')
plt.ylim(1,3)
plt.show()
```


90
80
70
60
50
40
30
20
10
0
0
250
500
750
1000
1250
1500
1750
2000


0
2
4
6
8
10
12
14
0
250
500
750
1000
1250
1500
1750
2000



Le modèle complet et le modèle par moyenne sont très proches. Comme le montre le dernier graphique, le modèle réduit convient très bien au faible durée d'envoi.

Le modèle réduit est de mon point de vue plus intéressant pour déterminer la latence et la capacité car il est moins sensible au bruit. Il sera ainsi plus simple de comparer la latence et la capacité entre deux connexions de réseaux. Néanmoins, ce modèle n'est pas bon pour prédire la durée d'envoi à partir de nouvelle information. Les deux autres modèles auront en moyenne même d'erreur.

1.5 Second jeu de données

Nous allons effectuer le même travail avec un second jeu de données :

[79]:
```
data_url = 'http://mescal.imag.fr/membres/arnaud.legrand/teaching/2014/
 ↪RICM4_EP_ping/stackoverflow.log.gz'
raw_data = pd.read_table(data_url,sep=' ',header=None)
raw_data
```

[79]:

	0	1	2	3	4	\
0	[1421771203.082701]	1257	bytes	from	stackoverflow.com	
1	[1421771203.408254]	454	bytes	from	stackoverflow.com	
2	[1421771203.739730]	775	bytes	from	stackoverflow.com	

```
3     [1421771204.056630]  1334  bytes  from  stackoverflow.com
4     [1421771204.372224]    83  bytes  from  stackoverflow.com
5     [1421771204.688367]   694  bytes  from  stackoverflow.com
6     [1421771205.005514]  1577  bytes  from  stackoverflow.com
7     [1421771205.321112]   632  bytes  from  stackoverflow.com
8     [1421771205.637464]   405  bytes  from  stackoverflow.com
9     [1421771205.953472]  1419  bytes  from  stackoverflow.com
10    [1421771206.269163]   329  bytes  from  stackoverflow.com
11    [1421771206.585098]   868  bytes  from  stackoverflow.com
12    [1421771206.901972]  1714  bytes  from  stackoverflow.com
13    [1421771207.217863]  1053  bytes  from  stackoverflow.com
14    [1421771207.533900]   349  bytes  from  stackoverflow.com
15    [1421771207.851148]  1598  bytes  from  stackoverflow.com
16    [1421771208.166794]  1412  bytes  from  stackoverflow.com
17    [1421771208.482159]   167  bytes  from  stackoverflow.com
18    [1421771208.798155]    60  bytes  from  stackoverflow.com
19    [1421771209.114480]  1038  bytes  from  stackoverflow.com
20    [1421771209.430586]   949  bytes  from  stackoverflow.com
21    [1421771209.746729]   279  bytes  from  stackoverflow.com
22    [1421771210.062322]   757  bytes  from  stackoverflow.com
23    [1421771210.378113]  1355  bytes  from  stackoverflow.com
24    [1421771210.694015]  1151  bytes  from  stackoverflow.com
25    [1421771211.009670]   237  bytes  from  stackoverflow.com
26    [1421771211.324856]  1221  bytes  from  stackoverflow.com
27    [1421771211.640544]  1063  bytes  from  stackoverflow.com
28    [1421771211.956109]   445  bytes  from  stackoverflow.com
29    [1421771212.272504]  1619  bytes  from  stackoverflow.com
…                    …   …    …   …               …
6857  [1421773451.530711]   234  bytes  from  stackoverflow.com
6858  [1421773451.847515]   231  bytes  from  stackoverflow.com
6859  [1421773452.163837]  1495  bytes  from  stackoverflow.com
6860  [1421773452.479834]  1313  bytes  from  stackoverflow.com
6861  [1421773452.795239]   182  bytes  from  stackoverflow.com
6862  [1421773453.111570]  2000  bytes  from  stackoverflow.com
6863  [1421773453.427110]  1396  bytes  from  stackoverflow.com
6864  [1421773453.742351]   515  bytes  from  stackoverflow.com
6865  [1421773454.058100]   590  bytes  from  stackoverflow.com
6866  [1421773454.373566]   229  bytes  from  stackoverflow.com
6867  [1421773454.689196]   806  bytes  from  stackoverflow.com
6868  [1421773455.007766]   422  bytes  from  stackoverflow.com
6869  [1421773455.324571]  1939  bytes  from  stackoverflow.com
6870  [1421773455.639814]   365  bytes  from  stackoverflow.com
6871  [1421773455.954957]   502  bytes  from  stackoverflow.com
6872  [1421773456.272951]  1738  bytes  from  stackoverflow.com
6873  [1421773456.591915]  1148  bytes  from  stackoverflow.com
6874  [1421773456.915868]   294  bytes  from  stackoverflow.com
6875  [1421773457.231617]  1534  bytes  from  stackoverflow.com
```

```
6876  [1421773457.546404]  1103  bytes  from  stackoverflow.com
6877  [1421773457.861499]  1121  bytes  from  stackoverflow.com
6878  [1421773458.177030]  1219  bytes  from  stackoverflow.com
6879  [1421773458.493444]  1880  bytes  from  stackoverflow.com
6880  [1421773458.808864]   986  bytes  from  stackoverflow.com
6881  [1421773459.124524]   357  bytes  from  stackoverflow.com
6882  [1421773459.440517]  1696  bytes  from  stackoverflow.com
6883  [1421773459.756250]   561  bytes  from  stackoverflow.com
6884  [1421773460.071820]   773  bytes  from  stackoverflow.com
6885  [1421773460.387385]  1009  bytes  from  stackoverflow.com
6886  [1421773460.704382]  1948  bytes  from  stackoverflow.com

                      5           6       7         8   9
0     (198.252.206.140):  icmp_seq=1  ttl=50  time=120  ms
1     (198.252.206.140):  icmp_seq=1  ttl=50  time=120  ms
2     (198.252.206.140):  icmp_seq=1  ttl=50  time=126  ms
3     (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
4     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
5     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6     (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
7     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
8     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
9     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
10    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
11    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
12    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
13    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
14    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
15    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
16    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
17    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
18    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
19    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
20    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
21    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
22    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
23    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
24    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
25    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
26    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
27    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
28    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
29    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
…                   …         …     …       …  ..
6857  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6858  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6859  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
```

```
6860  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6861  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6862  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6863  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6864  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6865  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6866  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6867  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6868  (198.252.206.140):  icmp_seq=1  ttl=50  time=113  ms
6869  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6870  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6871  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6872  (198.252.206.140):  icmp_seq=1  ttl=50  time=113  ms
6873  (198.252.206.140):  icmp_seq=1  ttl=50  time=114  ms
6874  (198.252.206.140):  icmp_seq=1  ttl=50  time=119  ms
6875  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6876  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6877  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6878  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6879  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6880  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6881  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6882  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6883  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6884  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6885  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6886  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms

[6887 rows x 10 columns]
```

Nous regardons si des lignes sont manquantes. Puis nous les supprimons, comme leur nombre est faible.

[80]:
```
np.shape(raw_data[raw_data.isnull().any(axis=1)])[0]
data = raw_data.dropna().copy()
data
```

[80]:
```
                        0     1      2     3                    4  \
0     [1421771203.082701]  1257  bytes  from  stackoverflow.com
1     [1421771203.408254]   454  bytes  from  stackoverflow.com
2     [1421771203.739730]   775  bytes  from  stackoverflow.com
3     [1421771204.056630]  1334  bytes  from  stackoverflow.com
4     [1421771204.372224]    83  bytes  from  stackoverflow.com
5     [1421771204.688367]   694  bytes  from  stackoverflow.com
6     [1421771205.005514]  1577  bytes  from  stackoverflow.com
7     [1421771205.321112]   632  bytes  from  stackoverflow.com
8     [1421771205.637464]   405  bytes  from  stackoverflow.com
```

```
9     [1421771205.953472]  1419  bytes  from  stackoverflow.com
10    [1421771206.269163]   329  bytes  from  stackoverflow.com
11    [1421771206.585098]   868  bytes  from  stackoverflow.com
12    [1421771206.901972]  1714  bytes  from  stackoverflow.com
13    [1421771207.217863]  1053  bytes  from  stackoverflow.com
14    [1421771207.533900]   349  bytes  from  stackoverflow.com
15    [1421771207.851148]  1598  bytes  from  stackoverflow.com
16    [1421771208.166794]  1412  bytes  from  stackoverflow.com
17    [1421771208.482159]   167  bytes  from  stackoverflow.com
18    [1421771208.798155]    60  bytes  from  stackoverflow.com
19    [1421771209.114480]  1038  bytes  from  stackoverflow.com
20    [1421771209.430586]   949  bytes  from  stackoverflow.com
21    [1421771209.746729]   279  bytes  from  stackoverflow.com
22    [1421771210.062322]   757  bytes  from  stackoverflow.com
23    [1421771210.378113]  1355  bytes  from  stackoverflow.com
24    [1421771210.694015]  1151  bytes  from  stackoverflow.com
25    [1421771211.009670]   237  bytes  from  stackoverflow.com
26    [1421771211.324856]  1221  bytes  from  stackoverflow.com
27    [1421771211.640544]  1063  bytes  from  stackoverflow.com
28    [1421771211.956109]   445  bytes  from  stackoverflow.com
29    [1421771212.272504]  1619  bytes  from  stackoverflow.com
…                    …    …     …    …              …
6857  [1421773451.530711]   234  bytes  from  stackoverflow.com
6858  [1421773451.847515]   231  bytes  from  stackoverflow.com
6859  [1421773452.163837]  1495  bytes  from  stackoverflow.com
6860  [1421773452.479834]  1313  bytes  from  stackoverflow.com
6861  [1421773452.795239]   182  bytes  from  stackoverflow.com
6862  [1421773453.111570]  2000  bytes  from  stackoverflow.com
6863  [1421773453.427110]  1396  bytes  from  stackoverflow.com
6864  [1421773453.742351]   515  bytes  from  stackoverflow.com
6865  [1421773454.058100]   590  bytes  from  stackoverflow.com
6866  [1421773454.373566]   229  bytes  from  stackoverflow.com
6867  [1421773454.689196]   806  bytes  from  stackoverflow.com
6868  [1421773455.007766]   422  bytes  from  stackoverflow.com
6869  [1421773455.324571]  1939  bytes  from  stackoverflow.com
6870  [1421773455.639814]   365  bytes  from  stackoverflow.com
6871  [1421773455.954957]   502  bytes  from  stackoverflow.com
6872  [1421773456.272951]  1738  bytes  from  stackoverflow.com
6873  [1421773456.591915]  1148  bytes  from  stackoverflow.com
6874  [1421773456.915868]   294  bytes  from  stackoverflow.com
6875  [1421773457.231617]  1534  bytes  from  stackoverflow.com
6876  [1421773457.546404]  1103  bytes  from  stackoverflow.com
6877  [1421773457.861499]  1121  bytes  from  stackoverflow.com
6878  [1421773458.177030]  1219  bytes  from  stackoverflow.com
6879  [1421773458.493444]  1880  bytes  from  stackoverflow.com
6880  [1421773458.808864]   986  bytes  from  stackoverflow.com
6881  [1421773459.124524]   357  bytes  from  stackoverflow.com
```

```
6882  [1421773459.440517]  1696  bytes  from  stackoverflow.com
6883  [1421773459.756250]   561  bytes  from  stackoverflow.com
6884  [1421773460.071820]   773  bytes  from  stackoverflow.com
6885  [1421773460.387385]  1009  bytes  from  stackoverflow.com
6886  [1421773460.704382]  1948  bytes  from  stackoverflow.com

                      5           6       7         8   9
0     (198.252.206.140):  icmp_seq=1  ttl=50  time=120  ms
1     (198.252.206.140):  icmp_seq=1  ttl=50  time=120  ms
2     (198.252.206.140):  icmp_seq=1  ttl=50  time=126  ms
3     (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
4     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
5     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6     (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
7     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
8     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
9     (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
10    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
11    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
12    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
13    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
14    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
15    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
16    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
17    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
18    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
19    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
20    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
21    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
22    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
23    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
24    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
25    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
26    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
27    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
28    (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
29    (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
…                    …        …     …       …  ..
6857  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6858  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6859  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6860  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6861  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6862  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6863  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6864  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6865  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
```

```
6866  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6867  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6868  (198.252.206.140):  icmp_seq=1  ttl=50  time=113  ms
6869  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6870  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6871  (198.252.206.140):  icmp_seq=1  ttl=50  time=110  ms
6872  (198.252.206.140):  icmp_seq=1  ttl=50  time=113  ms
6873  (198.252.206.140):  icmp_seq=1  ttl=50  time=114  ms
6874  (198.252.206.140):  icmp_seq=1  ttl=50  time=119  ms
6875  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6876  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6877  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6878  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6879  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms
6880  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6881  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6882  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6883  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6884  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6885  (198.252.206.140):  icmp_seq=1  ttl=50  time=111  ms
6886  (198.252.206.140):  icmp_seq=1  ttl=50  time=112  ms

[6824 rows x 10 columns]
```

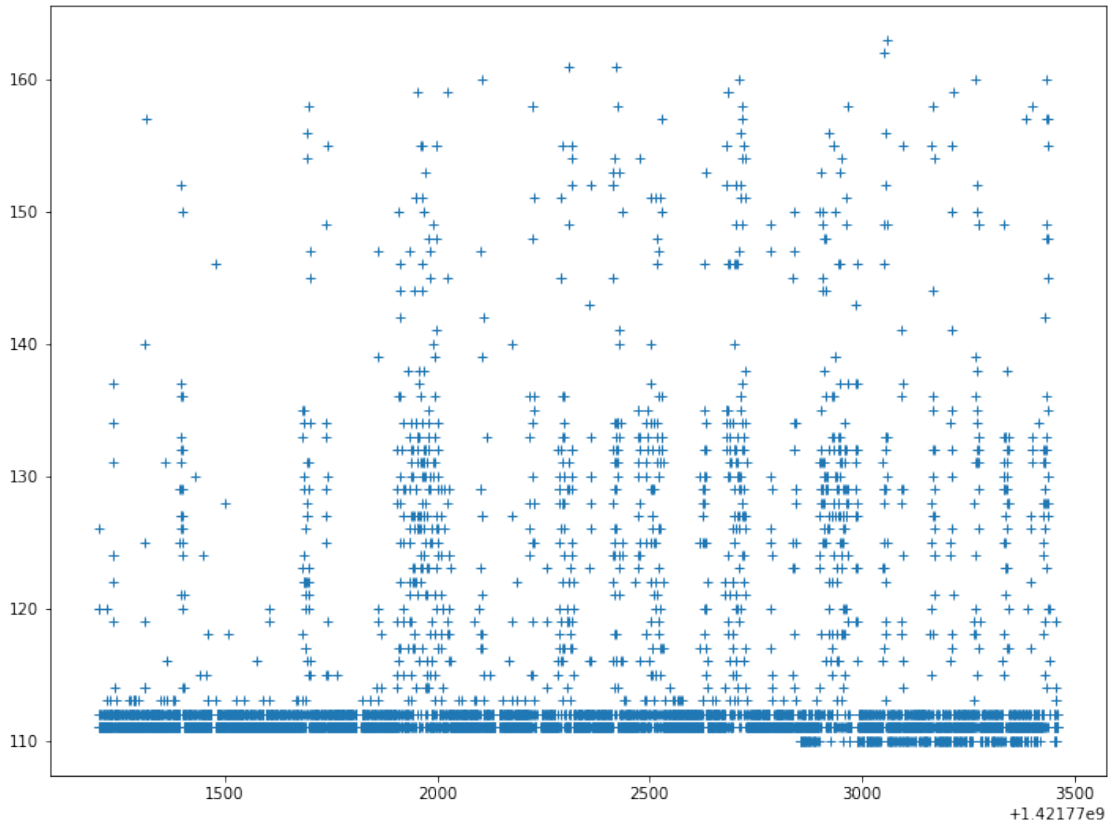
Nous changeons maintenant la forme des données utiles.

[81]:
```
def convert_time(time):
    return float(time[1:-1])

def convert_message(message):
    return int(message)

def convert_duration(duration):
    return float(duration[5:])

data['time_passed'] = [convert_time(time) for time in data[0]]
data['size_message'] = [convert_message(message) for message in data[1]]
data['duration'] = [convert_duration(duration) for duration in data[8]]
```

Nous allons vérifier que le temps n'a pas d'influence sur la durée d'envoi.

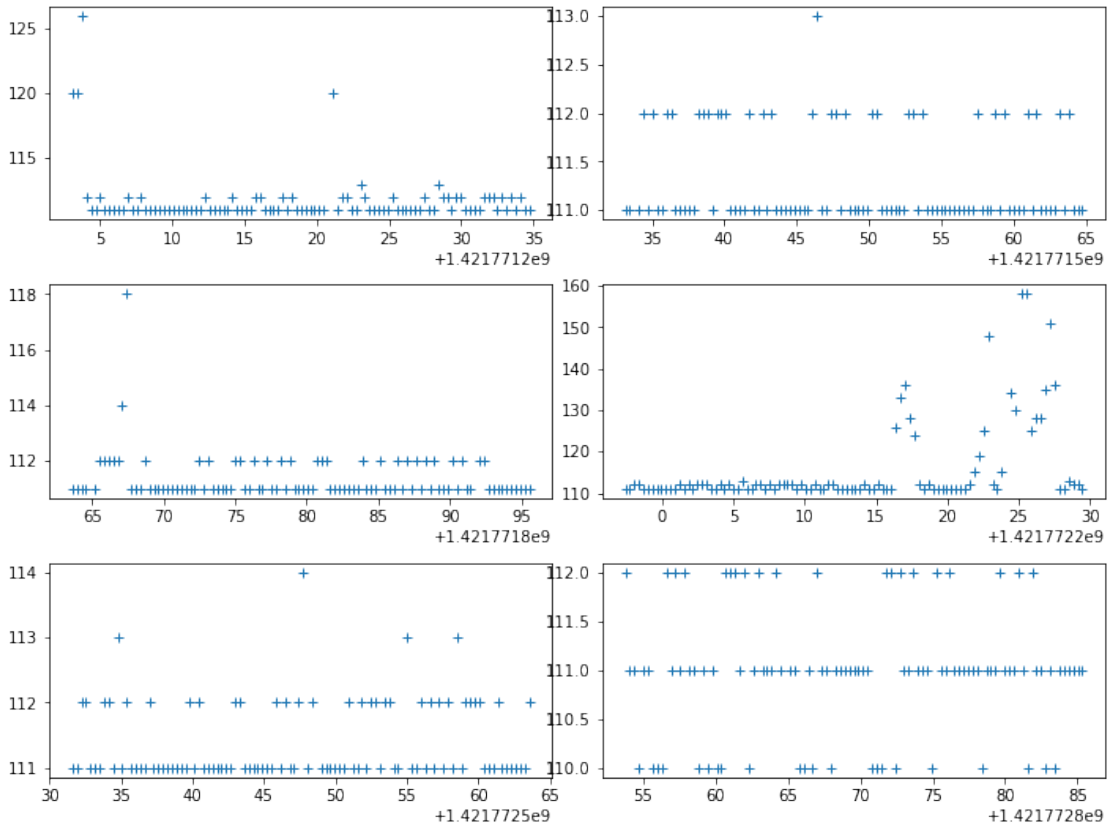
[82]:
```
plt.figure(figsize=(12,9))
plt.plot(data['time_passed'],data['duration'],'+')
```

[82]:
```
[<matplotlib.lines.Line2D at 0x7f6d9572b160>]
```



[83]:
```
plt.figure(figsize=(12,9))

plt.subplots_adjust(wspace=0.1, hspace=0.3)

plt.subplot(321)
plt.plot(data['time_passed'][0:100],data['duration'][0:100],'+')

plt.subplot(322)
plt.plot(data['time_passed'][1000:1100],data['duration'][1000:1100],'+')

plt.subplot(323)
plt.plot(data['time_passed'][2000:2100],data['duration'][2000:2100],'+')

plt.subplot(324)
plt.plot(data['time_passed'][3000:3100],data['duration'][3000:3100],'+')

plt.subplot(325)
plt.plot(data['time_passed'][4000:4100],data['duration'][4000:4100],'+')

plt.subplot(326)
```

```
plt.plot(data['time_passed'][5000:5100],data['duration'][5000:5100],'+')

plt.show()
```

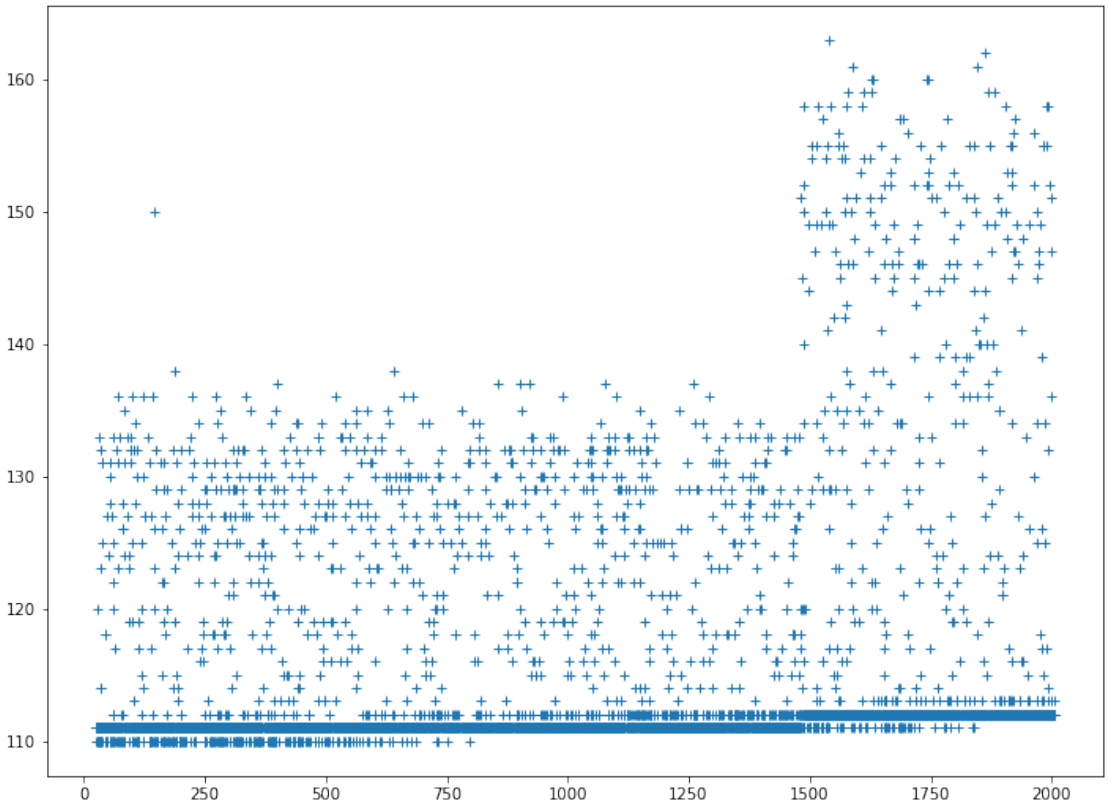


Comme pour le premier jeu de données, il ne semble pas y avoir de lien entre le temps et la durée d'envoi.

Nous allons maintenant regarder l'évolution de la durée d'envoi en fonction de la taille du message.

[84]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'+')
```

[84]:
```
[<matplotlib.lines.Line2D at 0x7f6d955932e8>]
```



Nous allons supposer ici qu'il n'y a pas de seuil, contrairement au premier jeu de données.

Nous pouvons effectuer une régression linéaire comme dans le cas précédent.

[85]:
```
modele=LinearRegression()
modele.fit(data['size_message'].values.reshape(-1, 1),data['duration'])

print(modele.score(data['size_message'].values.reshape(-1, 1),data['duration']))
```

```
0.023103989841964157
```

Nous allons essayer le modèle réduit comme dans le premier cas. Le modèle moyen donnant des résultats similaires nous n'allons pas le faire pour ce jeu de données. Pour cela, nous ne conservons que les durées d'envoi inférieures ou égales à 113 ms.

[86]:
```
data_reduced=data[data['duration']<=113]
print(len(data_reduced)/len(data))
```

```
0.8371922626025792
```

Puis, nous appliquons la régression linéaire.

[87]:
```
modele_reduced=LinearRegression()
modele_reduced.fit(data_reduced['size_message'].values.reshape(-1,␣
 ↪1),data_reduced['duration'])

print(modele_reduced.score(data_reduced['size_message'].values.reshape(-1,␣
 ↪1),data_reduced['duration']))
print(modele_reduced.score(data['size_message'].values.reshape(-1,␣
 ↪1),data['duration']))
```

```
0.47431238776980406
-0.12276793980489287
```

Nous obtenons des résultats similaires à ceux obtenus avec le premier jeu de données.

Nous pouvons maintenant déterminer la latence et la capacité pour les deux modèles.

[88]:
```
print(modele.intercept_)
print(1/modele.coef_[0])
print()
print(modele_reduced.intercept_)
print(1/modele_reduced.coef_[0])
```

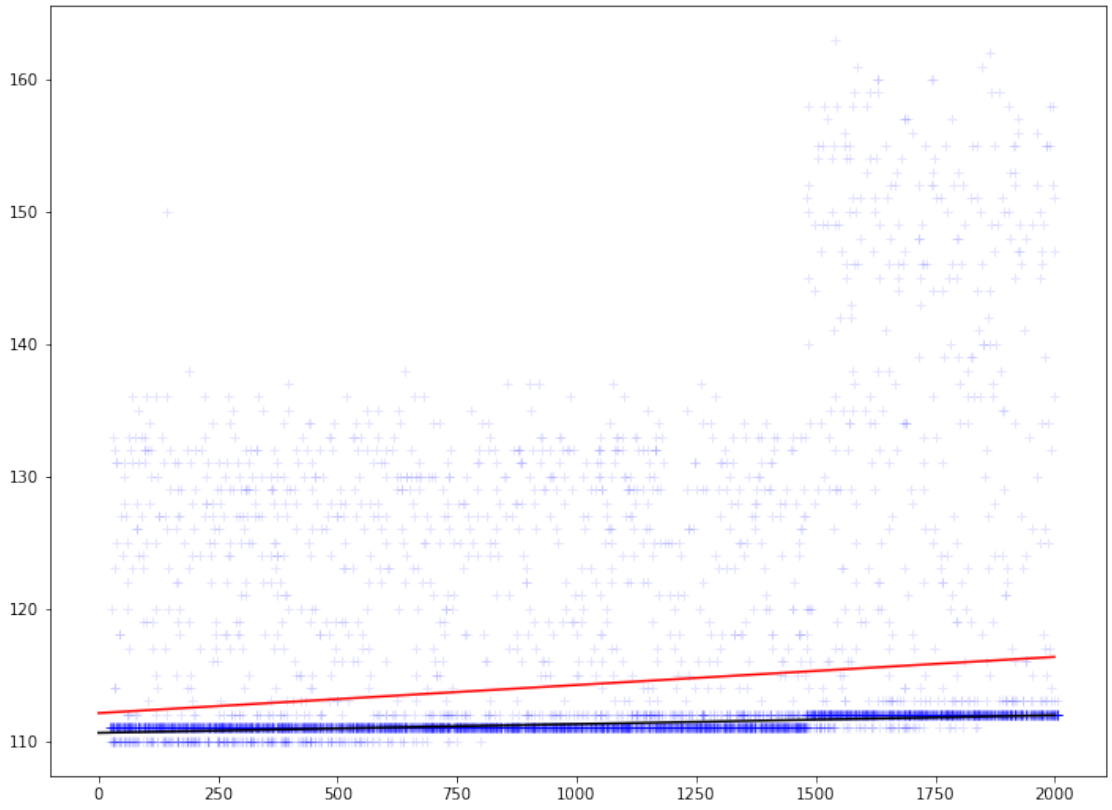
```
112.09420526325596
470.69432452502195

110.6015263223915
1501.9562382783452
```

Les latences sont assez similaires. Par contre, les capacités sont très différentes.

Traçons maintenant les trois régressions linéaires avec le jeu de données.

[89]:
```
size=np.arange(0,2000)
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.1)
plt.plot(size,modele.predict(np.reshape(size,(-1, 1))),'r')
plt.plot(size,modele_reduced.predict(np.reshape(size,(-1, 1))),'k')
```

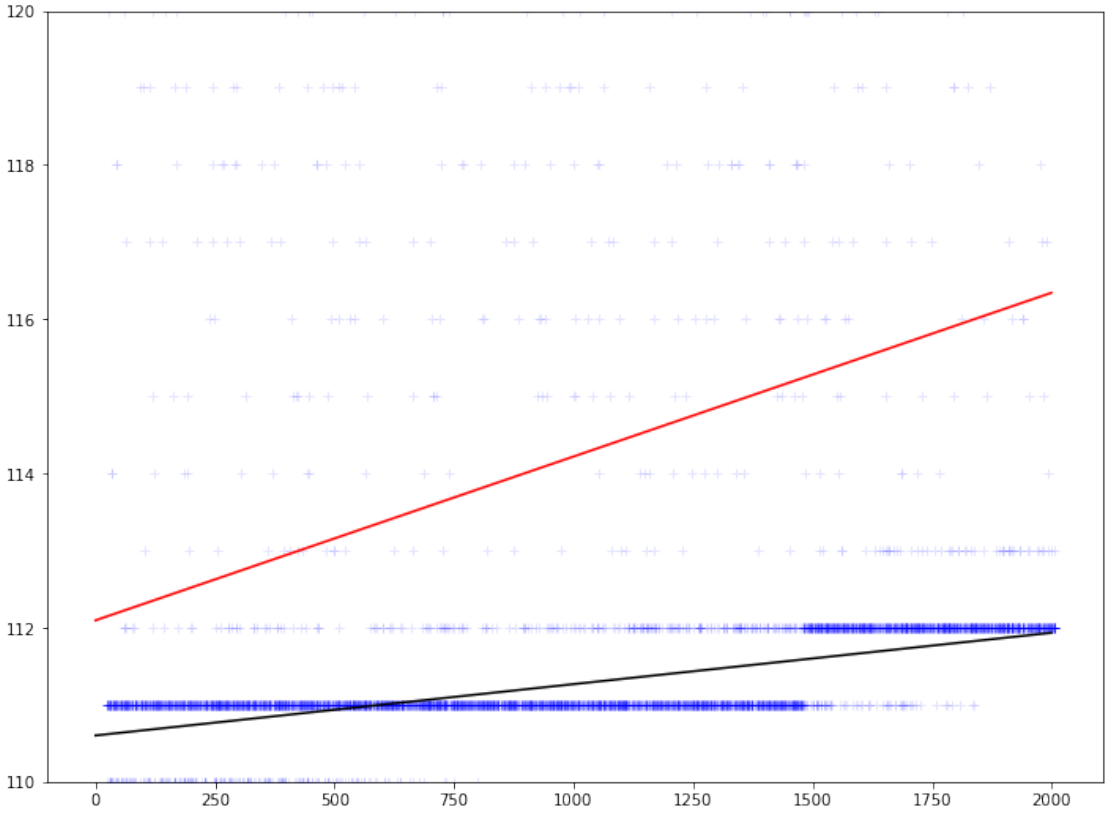
[89]:
```
[<matplotlib.lines.Line2D at 0x7f6d957f9be0>]
```



Avec un zoom.

[93]:
```
plt.figure(figsize=(12,9))
plt.plot(data['size_message'],data['duration'],'b+',alpha=0.1)
plt.plot(size,modele.predict(np.reshape(size,(-1, 1))),'r')
plt.plot(size,modele_reduced.predict(np.reshape(size,(-1, 1))),'k')
plt.ylim(110,120)
```

[93]: (110, 120)



Comme pour le premier jeu de données, il me semble que le modèle réduit est celui qui explique le mieux les données bien qu'il ne prenne pas les durées élevées en compte.

[ ]: