# Fouille de données – TD 2

M2 Informatique, Université de Paris

**TESTS:**  
Télécharger <http://fabien.viger.free.fr/ml/test.py> (c’est le même que pour le TD1), et <http://fabien.viger.free.fr/ml/test2.py> (celui la est nouveau), et les mettre dans le même répertoire que votre fichier td2.py, et tapez: **python3 test2.py**

#### **Exercice 0**

Implémentez la fonction suivante.

| def simulation\_coin(num\_exp, num\_coins\_per\_exp):  """Simulates many coin toss (heads/tails) experiments, and output the  observed distribution of the ratios of "tails", discretized.  See the description of the returned list below.  Args:  num\_exp: an integer. Number of experiments.  num\_coins\_per\_exp: an integer. Number of coin tosses per experiment.  Returns:  A list of 1001 elements: element #i will be the number of experiments  that yielded an observed ratio of "tails" approximately equal to  0.001\*i (i.e. to get the index corresponding to given ratio, use:  int(1000\*ratio) ).  """ |
| --- |

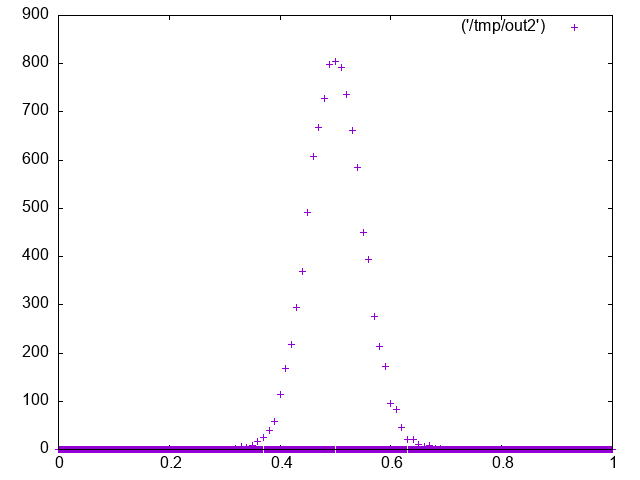
**Exemple:** si à la place de “1000” on utilisait 10, donc en renvoyant une liste de 10+1=11 elements, on s’attend à ce que simulation\_coin(1000, 100) donne quelquechose comme:

[0, 0, 0, 24, 442, 499, 34, 1, 0, 0, 0] # C’est un exemple.

Pour tester votre programme, on va tracer la distribution:

* Ouvrez un python interactif (tapez “python”)
* import td2
* data=td2.simulation\_coin(10000, 100)
* open('/tmp/data', 'w').write('\n'.join(['%f %f' % (0.001\*i, data[i]) for i in range(1001)]))
* Dans un terminal parallèle, tapez:  
  gnuplot -e "set term png; set out '/tmp/data.png'; plot('/tmp/data')"
* Visualisez! Par exemple, dans Chrome, allez à l’URL file:///tmp/data.png

Ca devrait ressembler à ca:



#### **Exercice 1**

Implémentez la fonction suivante, à l'aide de la fonction [math.erf()](https://docs.python.org/3/library/math.html#math.erf).  
(\*): Rendez-la plus encore plus précise pour des valeurs très grandes de "value" en utilisant aussi math.erfc(). Essayez avec value=**20** par exemple!

| def proba\_normal\_var\_above(value):  """Returns the probability that a random variable following a normal  distribution with mean 0 and standard deviation 1 is above "value".   Args:  value: a float. See the top-level comment.  Returns:  a float. See the top-level comment.  """ |
| --- |

#### **Exercice 2**

Implémentez la fonction suivante. C'est très tordu! L'idée est:

* Vous avez un échantillon de valeurs observées (par exemple, les tailles en cm d'un certain nombre d'européens).
* Vous voulez estimer quelle est la probabilité que votre estimation de la taille moyenne soit surestimée de delta cm ou plus. Par exemple, vous mesurez une moyenne de 167 cm sur un échantillon de 1000 personnes, et vous voulez estimer la probabilité de surestimer la *vraie* moyenne (celle sur l’ensemble de la population, pas seulement les 1000) de 2 cm ou plus. On fait donc l'hypothèse que la vraie moyenne est 165 cm, et on estime la probabilité d'observer une moyenne de 167 cm ou plus sur 1000 personnes.

Utilisez le théorème central limite + l'exo précédent.

| def proba\_sample\_mean\_above\_true\_mean\_by\_at\_least(sample, delta):  """Given a statistical sample (a list) of i.i.d. Values, returns the  probability there was of observing at least that mean, assuming that  the true mean was at most observed\_mean-delta.  Args:  sample: a list of numbers. See toplevel comment.  delta: a number. See toplevel comment.  Returns:  A float. See toplevel comment.  """ |
| --- |

#### Exercice 3

Implémentez la fonction suivante. On pourra utiliser une [recherche dichotomique](https://fr.wikipedia.org/wiki/Recherche_dichotomique) sur la fonction faite en exo 1.

| def standard\_percentile(p):  """Returns the value X such that the probability of drawing a  random variable following a normal distribution with mean 0 and standard  deviation 1 whose value is X or \*below\* is equal to p.  Example: If p=0.5, this should return 0 because there is a 0.5  probability of drawing a value above 0 in the normal  distribution with parameters (mean=0, stddev=1).  If p=0.84, this should return something close to 1 because  There is a ~0.84 probability to draw a value in [-infinity, 1].  Args:  p: a float. See toplevel comment.  Returns:  A float. See toplevel comment.  """ |
| --- |

#### 

#### Exercice 4

Implémentez la fonction suivante. C'est un peu comme de passer de l'exo 1 à l'exo 2, sauf qu'en plus on demande les 2 bornes : supérieure et inférieure.

| def confidence\_interval\_of\_mean(sample, pvalue):  """Assuming that sample is an i.i.d. random sample from a base  population, return the confidence interval of the mean value of  the underlying population, subject to the given pvalue.  Example: pvalue=0.05, sample=[23,15,17,22]. The "observed" mean of the  sample is X=19.25. The returned confidence interval [µ\_min, µ\_max] (as a  pair) is defined by:   * µ\_min should be the value of the "true" mean of the underlying population such that the probability of observing a mean equal or \*below\* X=19.25 on that population would be equal to 1-0.05=0.95. * µ\_max should be the value of the "true" mean of the underlying population such that the probability of observing a mean equal or \*above\* X=19.25 would be equal to 1-0.05=0.95.   In practice, we would get (µ\_min, µ\_max) ~= with these values.  Args:  sample: a list of numbers. See toplevel comment.  pvalue: a float in [0..1]. See toplevel comment.  Returns:  A pair of floats. See toplevel comment.  """ |
| --- |

#### 

#### Exercice 5 (optionnel: pour vous!): Attachement préférentiel

Implémentez la fonction suivante. Il s'agit de simuler un modèle statistique intéressant, une des premières explications empiriques/mathématiques de l'apparition des lois de puissance dans les grand réseaux d'interactions: l'attachement préférentiel. Ici, le modèle est un réseau social type twitter où chaque nouvel utilisateur "suit" un nombre fixe d'amis déjà présents sur le réseau (les noeux aléatoires), et "suit" en plus un nombre fixes d'utilisateurs, choisis préférentiellement selon le nombre d'utilisateurs qui les suivent déjà.

| def sim\_graph\_growth(num\_nodes, edges\_per\_new\_node, ratio\_follow\_edge):  """Simulates the growth of a graph, node by node. We start with a  single node. Then we add nodes one by one: the first  "edges\_per\_new\_node" nodes attach themselves to all the previous nodes  (meaning: we add an edge between them), then every time we add a node,  we add exactly ratio\_follow\_edge\*edges\_per\_new\_node edges by  "Following edges", and add (1-ratio\_follow\_edge)\*edges\_per\_new\_node by  attaching to a previously-created node, picked uniformly at random.  When we create an edge "following an edge", the idea is to pick one of  the already existing edges (uniformly at random), and to attach to  either one of its two nodes (50/50 chances).  Args:  num\_nodes: an integer. See toplevel comment.  edges\_per\_new\_node: an integer. See toplevel comment.  ratio\_follow\_edge: a float in [0..1]. See toplevel comment.  Returns:  A list of integers: the degrees of the nodes (number of edges).  """ |
| --- |

Bonus: produisez un graph (gnuplot?) de la distribution des degrés obtenue, par exemple avec

num\_nodes=10M, edges\_per\_new\_node=6, ratio\_follow\_edge=0.5.

Je vous laisse choisir le bon mode de représentation.. Pensez au cours!