Fouille de données – TD 2

M2 Informatique, Université de Paris

RAPPEL: Installez les packages si vous ne l'avez pas déja fait. Pense-bête de trucs en python.
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-là 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, num_buckets):
  """Simulates several coin toss (heads/tails) experiments, and outputs
     the observed distribution of the ratio of "tails", discretized in
     num_buckets buckets. See the description of the arguments below.

  Args:
     num_exp: an integer. Number of experiments.
     num_coins_per_exp: an integer. Number of coin tosses per experiment.
     num_buckets: an integer. Number of main buckets of the discretized
         output distribution. Each bucket has size 1/num_bucket.

  Returns:
      A list of (num_buckets+1) integers: element #i (0-indexed) will be the
      number of experiments that yielded a ratio of "tails" in the
      [i/num_buckets, (i+1)/num_buckets[  half-closed interval.
      There are num_buckets+1 elements because we need a last, additional
      bucket for the value 1, which is not included in the last interval
      since it's half-closed.
      NOTE: to get the bucket index #i from the ratio of tails
            r = num_tails/num_coins_per_exp, use this formula:
            i = int(r * num_buckets).
  """
```

Exemples / Explications:
Si `num_exp = 1000, num_coins_per_exp = 10 et num_buckets = 4,` on va simuler 1000 expériences. Dans chaque expérience on tire 10 pieces, on calcule le ratio de "tails" qui va donc être soit 0, soit 0.1, soit 0.2, …, soit 0.9, soit 1 (11 valeurs possibles), et on va incrémenter le **bucket** correspondant à ce ratio. Là on a 4(+1) buckets représentant ces intervalles de ratios:

- Bucket #0: ratio ∈ [0, 0.25[# note: contient les tirages avec 0, 1, 2 tails sur 10
- Bucket #1: ratio ∈ [0.25, 0.5[# note: contient les tirages avec 3, 4 tails sur 10

- Bucket #2: ratio ∈ [0.5, 0.75[# note: contient les tirages avec 5, 6, 7 tails sur 10
- Bucket #3: ratio ∈ [0.75, 1[# note: contient les tirages avec 8, 9 tails sur 10
- Bucket #4: ratio = 1   # note: contient les tirages avec 10 tails sur 10

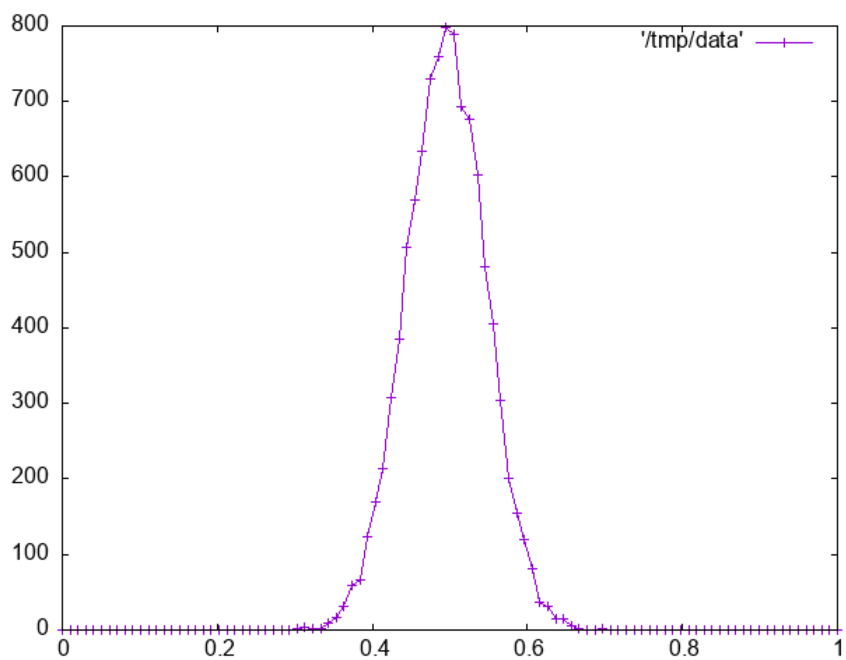
Donc on a notre liste de 5 éléments, qui commence à [0, 0, 0, 0, 0] et à chaque expérience, on va ajouter un à l'élément i si le ratio est dans le bucket #i.
Du coup, après 1000 expériences, on s'attend à obtenir une liste qui ressemblerait à:
[55, 322, 568, 54, 1]   # C'est un **exemple**! Les vraies valeurs varient un peu.
Vous remarquerez que la somme vaut num_exp = 1000, forcément.

Autre exemple: si `num_exp = 1000, num_coins_per_exp = 100 et num_buckets = 10,` on s'attend à ce que `simulation_coin(1000, 100, 10)` nous renvoie une liste de 11 entiers qui ressemble à: [0, 0, 0, 24, 442, 499, 34, 1, 0, 0, 0]   # Encore une fois c'est un **exemple**!

Pour tester votre programme, on va tracer la distribution:

- Ouvrez un python interactif (tapez "python")
- `import td2`
- `NB=99   # num_buckets`
- `data=td2.simulation_coin(10000, 100, NB)`
- `open('/tmp/data', 'w').write('\n'.join(['%f %f' % (i/NB, data[i]) for i in range(NB+1)]))`
- Dans un terminal parallèle, tapez:
  `gnuplot -e "set term png; set out '/tmp/data.png'; plot('/tmp/data')"`
  - Si vous n'avez pas gnuplot: `sudo apt install gnuplot`
- Visualisez! Par exemple, dans Chrome, allez à l'URL `file:///tmp/data.png`

Pour NB = 99 ça devrait *ressembler* à ca:



Exercice 1

Implémentez la fonction suivante, à l'aide de la fonction `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 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 **sur**estimée 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 votre échantillon) 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.
- Pour cela, utilisez le théorème central limite du cours + 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] 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, returns the confidence interval of the mean value of
  the underlying population, subject to the given p-value.

  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 *above* X=19.25 on that population would be equal to 0.05.
    -  μ_max 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 would be equal to 0.05.
  In practice, we would get (μ_min, μ_max) ≈ (16.1, 22.4) 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 X.com (aka twitter.com), 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 avec une proba proportionnelle à leur renommée (renommée = 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 *undirected* 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 for every new node after that, we connect it (i.e. add edges)
    to exactly exactly edges_per_new_node other nodes, picked like this:
    - edges_per_new_node - int(ratio_follow_edge * edges_per_new_node)
      nodes to attach to are picked uniformly at random among all existing
      Nodes.
    - int(ratio_follow_edge * edges_per_new_node) nodes to attach to are
      picked by "following random edges": we pick an existing edge
      uniformly at random, then we attach to one of its two nodes (50/50
      chance to pick either node).

  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 dictionary (or collections.defaultdict) whose keys are the
      different degrees of nodes in the graph, and the values are the
      number of nodes with that degree in the graph.
  """
```

À titre d'illustration, voilà un output possible :
>>> sorted(td2.sim_graph_growth(1000, 3, 1).items())
[(3, 420), (4, 183), (5, 102), (6, 79), (7, 46), (8, 26), (9, 30), (10, 21), (11, 14), (12, 9), (13, 10), (14, 8), (15, 5), (16, 3), (17, 3), (18, 3), (19, 4), (20, 2), (21, 1), (22, 6), (23, 1), (24, 3), (25, 1), (26, 2), (27, 1), (28, 1), (29, 1), (30, 1), (31, 3), (32, 1), (33, 1), (36, 1), (38, 1), (40, 1), (41, 1), (66, 1), (67, 1), (75, 1), (79, 1), (115, 1)]

Ce qui signifie qu'en simulant un processus d'attachement préférentiel "pur" (`ratio_follow_edge`=1), sur un graphe de 1000 noeuds avec 3 arêtes par nouveau noeud, on obtient ≈420 noeuds de degré 3, ≈180 de degré 4, ≈100 de degré 5, …, et 1 noeud de degré > 100.

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!