TIPE : Propagation de rumeurs dans un réseau social
Listing des programmes

Hugo LEVY-FALK

2017

Table des matières

1 graph.ml **2**

2 experiment.ml **4**

3 sortFirst.ml **6**

4 spread.ml **6**

5 experimentSpreadingRandom.ml **7**

6 experimentSpreadingDegree.ml **9**

7 experimentSpreadingBetween.ml **11**

8 drawExperimentSpreadingRandom.py **14**

9 drawExperimentSpreadingRandomInitial.py **15**

10 drawExperimentSpreadingDegreeInitial.py **17**

1 graph.ml

```
open Core.Std;;

(* Builds a random graph with the Watts and Strogatz method.
*)

Random.self_init ();;
let wattsStrogatzMatrix n k beta =
  let l = Array.make_matrix n n false in
  let rec wire i j = if i < 0 then wire (n+i) j
    else if i >= n then wire (i-n) j
    else if j < 0 then wire i (n+j)
    else if j >= n then wire i (j-n)
    else (l.(i).(j) <- true;l.(j).(i) <- true)
  in
  let rec unwire i j = if i < 0 then unwire (n+i) j
    else if i >= n then unwire (i-n) j
    else if j < 0 then unwire i (n+j)
    else if j >= n then unwire i (j-n)
    else (l.(i).(j) <- false;l.(j).(i) <- false)
  in
  let rec wired i j = if i < 0 then wired (n+i) j
    else if i >= n then wired (i-n) j
    else if j < 0 then wired i (n+j)
    else if j >= n then wired i (j-n)
    else l.(i).(j)
  in
  for i=0 to n-1 do
    for j = i-k/2 to i+k/2 do
      if j != i then wire i j
    done
  done;
  for i = 0 to n-1 do
    for j = i+1 to (i+k/2) do
      let r = Random.float 1.0 in
      if r < beta then begin
        unwire i j;
        let k = ref (Random.int n) in
        while (wired i !k) || (!k = i) do
          k := Random.int n
        done;
        wire i !k
      end
    done;
  done;
  l
;;
```

```
(* Betweenness centrality of a graph via its adjacency matrix*)
let betweenness g =
  let n = Array.length g in
  let cB = Array.create n (0.0) in
  for s = 0 to n-1 do
    let stack = Stack.create() in
    let p = Array.create n ([]) in
    let sigma = Array.create n ( 0.0) in
    sigma.(s) <- 1.0;
    let d = Array.create n ((-1)) in
    d.(s) <- 0;
    let q = Queue.create() in
    Queue.enqueue q s;
    while not (Queue.is_empty q) do
      let v = Queue.dequeue_exn q in
      Stack.push stack v;
      for w = 0 to (n-1) do
        let iw = g.(v).(w) in
        if iw then
          if d.(w) < 0 then begin
            Queue.enqueue q w;
            d.(w) <- d.(v) + 1;
          end;
          if d.(w) = (d.(v) + 1) then begin
            sigma.(w) <- sigma.(w) +. sigma.(v);
            p.(w) <- v::p.(w);
          end;
      done;
    done;
    let delta = Array.create n (0.0) in
    while not (Stack.is_empty stack) do
      let w = Stack.pop_exn stack in
      List.iter ~f:(fun v -> delta.(v) <-
        delta.(v) +. sigma.(v) /. sigma.(w) *. (1.0 +. delta.(w))) p.(w);
      if w != s then begin cB.(w) <- cB.(w) +. delta.(w); end;
    done;
  done;
  cB
;;

let degree g i =
  let n = Array.length g in
  let r = ref 0 in
  for j = 0 to (n-1) do
     if g.(i).(j) then incr r
  done;
  !r
;;
```

```
let maxDegree g n =
  let deg = degree g in
  let size = Array.length g in
  let rec loop i r = if i >= size then r else
    loop (i+1) ((deg i, i)::r)
  in
  SortFirst.sortFirst n (loop 0 [])
;;

let maxBetweenness g n =
  let a = betweenness g in
  let size = Array.length g in
  let rec loop i r = if i >= size then r else
    loop (i+1) ((a.(i), i)::r)
  in
  SortFirst.sortFirst n (loop 0 [])
;;
```

2 experiment.ml

```
(* Structure de la table experiments:
CREATE TABLE "experiments" ("name" TEXT,"last_id" INT,"infos" TEXT DEFAULT (null))
*)

let silent = true;;


let print_return m r = if silent then () else
  Printf.printf ("%s %s") m (Sqlite3.Rc.to_string r)
;;

let load_db () =
  Sqlite3.db_open "experiments.sqlite"
;;

let close_db db =
  Sqlite3.db_close db
;;

let cleaner target row =
  let l = Array.length row in
  for i = 0 to l-1 do
    target := Some(row.(i))
  done;
;;

let get_exp_last_id db name =
  let last_id = ref None in
```

```
  let get_last_id () =
    print_return "Recherche du dernier identifiant."
    (Sqlite3.exec_not_null_no_headers db ~cb:(cleaner last_id)
      ("SELECT last_id FROM experiments WHERE name=\""^name^"\";"))
  in
  get_last_id ();
  ! last_id
;;

let get_experiment db name =
  let last_id = get_exp_last_id db name in
  let last_result = ref None in
  let create_table () =
    print_return ("Création de la table " ^ name)
    (Sqlite3.exec db ("CREATE TABLE "^name^" (id INT, value TEXT);"));
    print_return ("Enregistrement dans 'experiments' de la table " ^ name) (
      Sqlite3.exec db ("INSERT INTO experiments VALUES (\""^name^"\", 0, \"\");"
    ))
  in
  let get_last_result id =
    print_return "Recherche du dernier résultat."
    (Sqlite3.exec_not_null_no_headers db ~cb:(cleaner last_result)
      ("SELECT value FROM "^name^" WHERE id=\""^id^"\";"))
  in
  let get_last_step () = match last_id with
  | None -> create_table (); !last_result
  | Some s -> get_last_result s; !last_result
  in
  get_last_step ()
;;

let add_step_id db exp id str =
  print_return (Printf.sprintf ("Ajout de l'étape %d à %s") id exp)
  (Sqlite3.exec db
    ("INSERT INTO "^exp^" VALUES ("^(string_of_int id)^",\""^str^"\");"))
;;

let change_last_id db exp id =
  print_return (Printf.sprintf
    ("Mise à jour du dernier identifiant (%d) de %s") id exp)
  (Sqlite3.exec db
    ("UPDATE experiments SET last_id = "
      ^(string_of_int id)^" WHERE name=\""^exp^"\";"))
;;
let add_step db exp str = let id =
  match get_exp_last_id db exp with
  | None -> 0
  | Some(s) ->  (1 + (int_of_string s))
  in
```

```
  add_step_id db exp id str;
  change_last_id db exp id;
;;
```

3 sortFirst.ml

```
open Core.Std;;

let sortFirst n l =
  let rec sep p l left right len_l = match l with
  | [] -> left, right, len_l
  | (a,b)::tl when a > p -> sep p tl ((a,b)::left) right (len_l + 1)
  | (a,b)::tl -> sep p tl left ((a,b)::right) len_l
  in
  let rec loop n l = match l with
  | [] -> []
  | (a,b)::tl ->
    let left, right, len_l = sep a tl [] [] 0 in
    if len_l >= n then (loop n left)
    else
    (loop n left)@[(a,b)]@(loop (n-len_l-1) right)
  in
  List.map (loop n l) ~f:(fun (a,b) -> b)
;;
```

4 spread.ml

```
open Core.Std;;


(*
  Spread the rumor and return the amount of nodes aware of it.
*)
let step_p graph a b s =
  let n = Array.length graph in
  let p_lim = b /. (a +. b) in
  let nb = ref 0 in
  for i = 0 to n-1 do
    if not s.(i) then begin
      let aware = ref 0 in
      let d = ref 0 in
      for j = 0 to n-1 do
        if graph.(i).(j) then(
          incr d;
          if s.(j) then incr aware;
        )
      done;
      let p = (float_of_int !aware) /. (float_of_int !d) in
```

```
      if p > p_lim then (s.(i) <- true; incr nb)
    end
    else incr nb
  done;
  !nb
;;
```

5 experimentSpreadingRandom.ml

```
open Core.Std;;
open Yojson.Basic.Util;;


type exp_stat = {
  graph_no:int;
  prop_spread:float array;
};;


let json_of_exp_stat e =
  `Assoc [
  ("graph_no", `Int e.graph_no);
  ("prop_spread",
    `List (List.map (Array.to_list e.prop_spread) ~f:(fun x -> `Float x)))
];;

let exp_stat_of_json json =
  {
    graph_no = json |> member "graph_no" |> to_int;
    prop_spread = json |> member "prop_spread" |> to_list
    |> List.map ~f:(fun x -> x |> to_float) |> Array.of_list
  }
;;

let escape_double_quotes s =
  let exp = Str.regexp "\"" in
  Str.global_replace exp "\"\"" s
;;

let save_step db e exp_name=
  json_of_exp_stat e
  |> Yojson.Basic.to_string
  |> escape_double_quotes
  |> Experiment.add_step_id db exp_name e.graph_no;
  Experiment.change_last_id db exp_name e.graph_no
;;
```

```
let process db graph_size nb_gen k beta max_spread_step a b =
  let choose_spread init =
    let s = Array.create graph_size false in
    let k = ref 0 in
    let n = int_of_float ((float_of_int graph_size) *. init) in
    while !k <= n do
      let i = Random.int graph_size in
      if not s.(i) then (incr k; s.(i) <- true)
    done;
    s
  in
  let step i init=
    let g = Graph.wattsStrogatzMatrix graph_size k beta in
    let prop_spread = Array.create max_spread_step 0.0 in
    let spread = choose_spread init in
    let j = ref 0 in
    let p = ref (-1.0) in
    while !j <= (max_spread_step-1) && prop_spread.(!j) != !p do
      prop_spread.(!j) <-
      (float_of_int (Spread.step_p g a b spread)) /. (float_of_int graph_size);
      if !j > 0 then (p := prop_spread.(!j-1));
      incr j;
    done;
    {graph_no=i; prop_spread=prop_spread}
  in
  let experiment init db=
    let exp_name = Printf.sprintf ("r_spreading_random_%d_%d_%d_%d_%d_%d_%d_%d")
      graph_size (int_of_float (beta*.100.0))
      k nb_gen init max_spread_step (int_of_float a) (int_of_float b) in
    print_endline ("Nom de l'expérience : "^exp_name);
    let cur_step = ref (match Experiment.get_experiment db exp_name
    with
    | None -> {graph_no= (-1);prop_spread=[||]}
    | Some(s) -> s |> Yojson.Basic.from_string |> exp_stat_of_json
    ) in
    let beg = !cur_step.graph_no + 1 in
    for i = beg to nb_gen - 1 do
      cur_step := step i ((float_of_int init)/.100.0);
      save_step db !cur_step exp_name
    done;
  in
  for i = 1 to 99 do
    experiment i db;
    print_newline ()
  done
;;


let graph_size = 500;;
```

```
let nb_gen = 100;;
let max_spread_step = 500;;

let () =
  print_endline "Initialisation de Random.";
  Random.self_init ();
  print_endline "Ouverture de la base de données.";
  let db = Experiment.load_db () in
  for b = 0 to 4 do
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 1.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 3.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 1.0 3.0;
  done;
  print_endline "Fermeture de la base de données.";
  if (Experiment.close_db db) then
    print_endline "Fermeture réussie."
  else
    print_endline "Echec de la fermeture."
```

6 experimentSpreadingDegree.ml

```
open Core.Std;;
open Yojson.Basic.Util;;


type exp_stat = {
  graph_no:int;
  prop_spread:float array;
};;


let json_of_exp_stat e =
  `Assoc [
  ("graph_no", `Int e.graph_no);
  ("prop_spread", `List (List.map (Array.to_list e.prop_spread)
    ~f:(fun x -> `Float x)))
];;

let exp_stat_of_json json =
  {
    graph_no = json |> member "graph_no" |> to_int;
    prop_spread = json |> member "prop_spread" |> to_list
    |> List.map ~f:(fun x -> x |> to_float) |> Array.of_list
  }
;;
```

```
let escape_double_quotes s =
  let exp = Str.regexp "\"" in
  Str.global_replace exp "\"\"" s
;;

let save_step db e exp_name=
  json_of_exp_stat e
  |> Yojson.Basic.to_string
  |> escape_double_quotes
  |> Experiment.add_step_id db exp_name e.graph_no;
  Experiment.change_last_id db exp_name e.graph_no
;;

let process db graph_size nb_gen k beta max_spread_step a b =
  let choose_spread g init =
    let s = Array.create graph_size false in
    let n = int_of_float ((float_of_int graph_size) *. init) in
    let rec loop l = match l with
    | [] -> ()
    | hd::tl -> s.(hd) <- true; loop tl
    in
    loop (Graph.maxDegree g n); s
  in
  let step i init=
    let g = Graph.wattsStrogatzMatrix graph_size k beta in
    let prop_spread = Array.create max_spread_step 0.0 in
    let spread = choose_spread g init in
    let j = ref 0 in
    let p = ref (-1.0) in
    while !j <= (max_spread_step-1) && prop_spread.(!j) != !p do
      prop_spread.(!j) <-
      (float_of_int (Spread.step_p g a b spread)) /. (float_of_int graph_size);
      if !j > 0 then (p := prop_spread.(!j-1));
      incr j;
    done;
    {graph_no=i; prop_spread=prop_spread}
  in
  let experiment init db=
    let exp_name = Printf.sprintf ("r_spreading_degree_%d_%d_%d_%d_%d_%d_%d_%d")
      graph_size (int_of_float (beta*.100.0))
      k nb_gen init max_spread_step (int_of_float a) (int_of_float b) in
    print_endline ("Nom de l'expérience : "^exp_name);
    let cur_step = ref (match Experiment.get_experiment db exp_name
    with
    | None -> {graph_no= (-1);prop_spread=[||]}
    | Some(s) -> s |> Yojson.Basic.from_string |> exp_stat_of_json
    ) in
    let beg = !cur_step.graph_no + 1 in
```

```
    for i = beg to nb_gen - 1 do
      cur_step := step i ((float_of_int init)/.100.0);
      save_step db !cur_step exp_name
    done;
  in
  for i = 1 to 99 do
    experiment i db;
    print_newline ()
  done
;;


let graph_size = 500;;
let nb_gen = 100;;
let max_spread_step = 500;;

let rec (^) k n = if n=0 then 1 else
  if (n mod 2) = 0 then let r = k ^ (n/2) in r*r
  else
    k * (k ^ (n-1))
;;


let () =
  print_endline "Initialisation de Random.";
  Random.self_init ();
  print_endline "Ouverture de la base de données.";
  let db = Experiment.load_db () in
  for b = 0 to 4 do
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 1.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 3.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 1.0 3.0;
  done;
  print_endline "Fermeture de la base de données.";
  if (Experiment.close_db db) then
    print_endline "Fermeture réussie."
  else
    print_endline "Echec de la fermeture."
```

7 experimentSpreadingBetween.ml

```
open Core.Std;;
open Yojson.Basic.Util;;
```

```
type exp_stat = {
  graph_no:int;
  prop_spread:float array;
};;


let json_of_exp_stat e =
  `Assoc [
  ("graph_no", `Int e.graph_no);
  ("prop_spread",
    `List (List.map (Array.to_list e.prop_spread) ~f:(fun x -> `Float x)))
];;

let exp_stat_of_json json =
  {
    graph_no = json |> member "graph_no" |> to_int;
    prop_spread = json |> member "prop_spread" |> to_list
    |> List.map ~f:(fun x -> x |> to_float) |> Array.of_list
  }
;;

let escape_double_quotes s =
  let exp = Str.regexp "\"" in
  Str.global_replace exp "\"\"" s
;;

let save_step db e exp_name=
  json_of_exp_stat e
  |> Yojson.Basic.to_string
  |> escape_double_quotes
  |> Experiment.add_step_id db exp_name e.graph_no;
  Experiment.change_last_id db exp_name e.graph_no
;;

let process db graph_size nb_gen k beta max_spread_step a b =
  let choose_spread g init =
    let s = Array.create graph_size false in
    let n = int_of_float ((float_of_int graph_size) *. init) in
    let rec loop l = match l with
    | [] -> ()
    | hd::tl -> s.(hd) <- true; loop tl
    in
    loop (Graph.maxBetweenness g n); s
  in
  let step i init=
    let g = Graph.wattsStrogatzMatrix graph_size k beta in
    let prop_spread = Array.create max_spread_step 0.0 in
    let spread = choose_spread g init in
    let j = ref 0 in
```

```
    let p = ref (-1.0) in
    while !j <= (max_spread_step-1) && prop_spread.(!j) != !p do
      prop_spread.(!j) <-
      (float_of_int (Spread.step_p g a b spread)) /. (float_of_int graph_size);
      if !j > 0 then (p := prop_spread.(!j-1));
      incr j;
    done;
    {graph_no=i; prop_spread=prop_spread}
  in
  let experiment init db=
    let exp_name = Printf.sprintf ("r_spreading_between_%d_%d_%d_%d_%d_%d_%d_%d")
      graph_size (int_of_float (beta*.100.0))
      k nb_gen init max_spread_step (int_of_float a) (int_of_float b) in
    print_endline ("Nom de l'expérience : "^exp_name);
    let cur_step = ref (match Experiment.get_experiment db exp_name
    with
    | None -> {graph_no= (-1);prop_spread=[||]}
    | Some(s) -> s |> Yojson.Basic.from_string |> exp_stat_of_json
    ) in
    let beg = !cur_step.graph_no + 1 in
    for i = beg to nb_gen - 1 do
      cur_step := step i ((float_of_int init)/.100.0);
      save_step db !cur_step exp_name
    done;
  in
  for i = 1 to 99 do
    experiment i db;
  done
;;


let graph_size = 500;;
let nb_gen = 100;;
let max_spread_step = 500;;

let rec (^) k n = if n=0 then 1 else
  if (n mod 2) = 0 then let r = k ^ (n/2) in r*r
  else
    k * (k ^ (n-1))
;;


let () =
  print_endline "Initialisation de Random.";
  Random.self_init ();
  print_endline "Ouverture de la base de données.";
  let db = Experiment.load_db () in
  for b = 0 to 4 do
    process db graph_size nb_gen 50
```

```
      ((float_of_int b) *. 0.25) max_spread_step 1.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 3.0 1.0;
    process db graph_size nb_gen 50
      ((float_of_int b) *. 0.25) max_spread_step 1.0 3.0;
  done;
  print_endline "Fermeture de la base de données.";
  if (Experiment.close_db db) then
    print_endline "Fermeture réussie."
  else
    print_endline "Echec de la fermeture."
```

8 drawExperimentSpreadingRandom.py

```
import matplotlib.pyplot as pl
import numpy as np
import sqlite3
import json


graph_size = 500
nb_gen = 100
k = 50
max_spread_step = 500

X = np.arange(max_spread_step)
conn = sqlite3.connect('experiments.sqlite')
pl.close('all')
fig = pl.figure()
ax = fig.add_subplot(111)
fig.suptitle(
    "Avancement de la propagation pour une distribution initiale aléatoire.",
    fontweight='bold')
ax.set_title('Taille du graphe={} Nb gen={} K={}'.format(graph_size,nb_gen,k))
ax.set_ylabel("Proportion")
ax.set_xlabel("Étape")


def process_json_mean(j, Y, Y_low, Y_high):
    t = json.loads(j)['prop_spread']
    for x,i in enumerate(t):
        Y[x] += i
        Y_high[x] = max(Y_high[x],i)
        if Y_low[x] == 0:
            Y_low[x] = i
        else:
            Y_low[x] = min(Y_low[x],i)
```

```
def process_standard_deviation(j, Y_deviation, Y):
    t = json.loads(j)['prop_spread']
    for x,i in enumerate(t):
        Y_deviation[x] += (i - Y[x])**2

def process_one(init):

    Y = np.zeros(max_spread_step)

    Y_deviation = np.zeros(max_spread_step)
    Y_high = np.zeros(max_spread_step)
    Y_low = np.zeros(max_spread_step)

    cursor = conn.execute(
        "SELECT value FROM r_spreading_random_"+
        "{graph_size}_{beta}_{k}_{nb_gen}_{x}_{max_spread_step}_{a}_{b}".format(
            graph_size, nb_gen, int(init*10), max_spread_step))
    rows = cursor.fetchall()
    for row in rows:
        process_json_mean(row[1], Y, Y_low, Y_high)
    Y = [y/nb_gen for y in Y]
    pl.plot(X, Y, label="Proportion initiale {}%".format(init*10))

for i in range(6,15):
    process_one(i/2)
conn.close()

pl.legend(loc="best")
pl.grid()
pl.show()
```

9 drawExperimentSpreadingRandomInitial.py

```
import matplotlib.pyplot as pl
from matplotlib2tikz import save as tikz_save
import numpy as np
import sqlite3
import json


graph_size = 500
nb_gen = 100
k = 50
max_spread_step = 500
conn = sqlite3.connect('experiments.sqlite')
for a,b in [(1,1), (1,3), (3,1)]:
    for beta in [0,25,50,75,100]:
        X = np.arange(1,100)
```

```
pl.close('all')
fig = pl.figure()
ax = fig.add_subplot(111)
ax.set_title('Taille du graphe={} Nb gen={} K={}, q={}, Beta ={}\\%'
    .format(graph_size,nb_gen,k,b/(a+b),beta))
ax.set_ylabel("Proportion finale")
ax.set_xlabel("Proportion initiale (\\%)")


Y = np.zeros(100)
Y_high = np.zeros(100)
Y_low = np.zeros(100)


x_50 = 0

for x in X:
    cursor = conn.execute(
        "SELECT value FROM r_spreading_random_"+
        "{graph_size}_{beta}_{k}_{nb_gen}_{x}_{max_spread_step}_{a}_{b}"
        .format(**locals()))
    rows = cursor.fetchall()
    v = 0
    n = len(rows)
    for row in rows:
        v += json.loads(row[0])['prop_spread'][-1]
    Y[x] = v/n
    if abs(0.5-Y[x]) < abs(Y[x_50]-0.5):
        x_50 = x
    v = 0
    for row in rows:
        v += (Y[x] - json.loads(row[0])['prop_spread'][-1])**2
    v = (v/n)**(1/2)
    Y_high[x] = Y[x] + v
    Y_low[x] = Y[x] - v

pl.grid()

pl.plot(X,Y[1:], 'b', label="Propagation finale", lw=2.5)
pl.plot(X,X/100, 'g--', label="Identité", lw=2.5)
pl.plot(X, Y_high[1:], 'r--', label="Écart-type", lw=2.5)
pl.plot(X, Y_low[1:], 'r--', lw=2.5)
pl.axvline(x_50, c='orange', lw=2.5)
pl.axhline(0.5, xmin=0, xmax=100, c='orange', lw=2.5)

pl.text(x_50+1, 0.1, str(x_50)+" \\%",
    bbox=dict(facecolor='orange', alpha=0.95),
    size="large",
    color="white",
```

```
            fontweight="bold")

        pl.legend(loc="lower right")
        tikz_save("resultats/random_finale_f_initiale_q{}_Beta{}_ec.tex"
            .format(int(b/(a+b)*100), beta))
conn.close()
```

10 drawExperimentSpreadingDegreeInitial.py

```
import matplotlib.pyplot as pl
from matplotlib2tikz import save as tikz_save
import numpy as np
import sqlite3
import json


graph_size = 500
nb_gen = 100
k = 50
max_spread_step = 500
conn = sqlite3.connect('experiments.sqlite')
for a,b in [(1,1), (1,3), (3,1)]:
    for beta in [0,25,50,75,100]:
        X = np.arange(1,100)
        pl.close('all')
        fig = pl.figure()
        ax = fig.add_subplot(111)
        ax.set_title('Taille du graphe={} Nb gen={} K={}, q={}, Beta ={}\\%'
            .format(graph_size,nb_gen,k,b/(a+b),beta))
        ax.set_ylabel("Proportion finale")
        ax.set_xlabel("Proportion initiale (\\%)")


        Y = np.zeros(100)
        Y_high = np.zeros(100)
        Y_low = np.zeros(100)


        x_50 = 0

        for x in X:
            cursor = conn.execute(
                "SELECT value FROM r_spreading_degree_"+
                "{graph_size}_{beta}_{k}_{nb_gen}_{x}_{max_spread_step}_{a}_{b}"
                .format(**locals()))
            rows = cursor.fetchall()
            v = 0
            n = len(rows)
```

```
            for row in rows:
                v += json.loads(row[0])['prop_spread'][-1]
            Y[x] = v/n
            if abs(0.5-Y[x]) < abs(Y[x_50]-0.5):
                x_50 = x
            v = 0
            for row in rows:
                v += (Y[x] - json.loads(row[0])['prop_spread'][-1])**2
            v = (v/n)**(1/2)
            Y_high[x] = Y[x] + v
            Y_low[x] = Y[x] - v

        pl.plot(X,Y[1:], 'b', label="Propagation finale", lw=2.5)
        pl.plot(X,X/100, 'g--', label="Identité", lw=2.5)
        pl.plot(X, Y_high[1:], 'r--', label="Écart-type", lw=2.5)
        pl.plot(X, Y_low[1:], 'r--')

        pl.axvline(x_50, c='orange', lw=2.5)
        pl.axhline(0.5, xmin=0, xmax=100, c='orange', lw=2.5)
        pl.text(x_50+1, 0.1, str(x_50)+" \\%",
            bbox=dict(facecolor='orange', alpha=0.95),
            size="large",
            color="white",
            fontweight="bold")

        pl.legend(loc="lower right")
        pl.grid()
        tikz_save("resultats/degree_finale_f_initiale_q{}_Beta{}_ec.tex"
            .format(int(b/(a+b)*100), beta))

conn.close()
```