ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

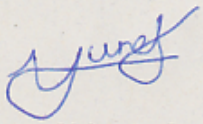
Stable Marriage Problem

Valentin Junet & Samuel Imfeld

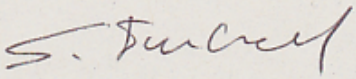
Zurich
December 2014

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.



Valentin Junet



Samuel Imfeld

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Stable Marriage Problem

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	First name(s):
Junet	Valentin
Imfeld	Samuel

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	Signature(s)
Zurich, 10.12.2014	
	S. Imfeld

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

1 Abstract

Online dating or match-making websites are flourishing these days. More and more people rely on their algorithms when searching for their Mr. Right, or Mrs. Right, respectively. Algorithms for match-making are therefore of quite some interest.

An important result in this concern is the so-called Gale Shapley algorithm proposed by Gale and Shapley [1962]. In 2012, Shapley even received the Nobel Prize in Economics for his work. The goal of this paper is to discuss the original model described by Gale and Shapley [1962] and advance the model in some sense. Namely we are going to introduce two changes to the model: The incompleteness of information and the possibility of preference changes. The modifications will be discussed further in section 4.2.

It is however not our claim that these changes applied to the model will make it an exact description of reality. Our goal is to study the repercussions on the stability and other significant indicators that show up when applying the modifications.

2 Individual contributions

Code was written by both. In the report, Valentin focused on the chapters 3, 4 and 6.1 whereas Samuel concentrated on chapters 1, 5 and 6.2.

3 Introduction and Motivations

The stable marriage problem is quite well known; it is originally an interesting mathematical problem, but it also has something catchy: Everyone has his own idea of what stability is, how we think it is optimal, and how we would like to be matched, or how we think it should be done. Then there comes this emotionless algorithm that pretends – with the support of some definitions and mathematical proofs – to find all of this for us and to do it as least as good as we would. Of course this is a little exaggerated. But when we think about those internet sites that match people according to some criteria, is't it what we're doing? Letting some numbers find an objective answer to those questions, to our subjective questions?

In order to prevent misunderstandings one must very precisely answer the following questions: What is stability? How can a stable situation be optimal? Of course answering those questions would result in a rather philosophical discussion which is not the goal of our paper. We will only be considering some very abstract and theoretical nodes which, for some practical and rhetorical reasons, will be called men or women. Nevertheless stability and other related indicators (which were going to introduce soon enough) will be the center of our attention.

The key points of our work presented here are to generate random links (or "friendship") between men and women (which in the following discussions will be referred to by nodes), to classify those links in a random way in order to get a preference list and to apply the algorithm of Gale and Shapley which matches the nodes according to their preferences. Then we modify some parameters and interpret the results. For a more precise description of the procedure, you may refer to the next section.

4 Description of the Model

4.1 Gale and Shapley's algorithm

Let's first describe the algorithm of Gale and Shapley. The setting is of a very abstract nature. There is a certain number of nodes, half of them are men and the other half are women. The first step is that every node makes a preference list, i. e. a ranking of all the nodes of oppposite sex. Then the algorithm proceeds as follows:

1. single men propose to their favorite women
2.
   - women accept their favorite suitor or their only suitor
   - if a woman has a fiance and a suitor, she chooses the one she likes better
   - if a woman rejects a man, then the next woman on the rejected man's list is his new favorite
3. if the single men still have women on their list of preferences, go to step 1

So the algorithm terminates when all nodes have a partner. This is obviously always achieved if the length of all the preference lists corresponds to the number of nodes of opposite sex (see Gale and Shapley [1962, p. 14, Theorem 1]). Gale and Shapley also proved that this algorithm always terminates with an optimal stability. For this purpose they gave precise definitions of stability and optimality:

Definition 1: *An assignment of applicants to colleges will be called unstable if there are two applicants α and β who are assigned to colleges A and B, respectively, although β prefers A to B and A prefers β to α.*[1]

Definition 2: *A stable assignment is called optimal if every applicant is at least as well off under it as under any other assignment.*[2]

These definitions are in terms of colleges and applicants because in their paper they first

[1] Gale and Shapley [1962, p. 10]
[2] Gale and Shapley [1962, p. 10]

start with this setting and then go on to the special case of stable marriages. Translated to the words of match-making the definitions would be something like: "A set of marriages is called unstable if under it there are a man and a woman who are not married to each other but prefer each other to their actual mates"[3] and "A stable assignment is called optimal if every man is at least as well off under it as under any other assignment".

4.2 Modifications

We will now present in more detail the modifications we made to this algorithm. Our purpose was to modify and adapt it so that it represents more accurately the reality. Of course it is fundamentally impossible to make a perfect representation of reality through this: If for example a man goes into a bar, he doesn't particularly make a list of the women he prefers and propose to each woman according to his list (or at least this is not systematically observed). However, there are some modifications we can make to make it more realistic. One of these is that a man doesn't necessarily know every women (and vice versa). To improve this point, we modified the list of preferences so that men and women only know each other in a restricted area. For example, a man who lives in China doesn't know a woman who lives in Switzerland, so he won't propose to her (we didn't consider the case where the man in China could know the Swiss woman though internet or by traveling, he only knows the women "around" him). We also considered the case in which a man knows a woman in his area but she doesn't know him and then this man can propose to her and she might accept or not. The other modification we made is that people, obviously, can change their minds. We modified it so that during the process either only the men, only the women or both can change their mind. These changes of preferences are made each "round" (each time a single man proposes to a woman) with a certain probability which is fixed.

We also introduced some new concepts, namely the number of "dumps" and the "optimality index". The number of "dumps" is the number of times a woman rejected her fiance for another man. This can be seen has an indication of time, the more people "get dumped", the longer it takes to reach a final assignment. The optimality index is defined as the sum over all ranks of the actual partners in the preference list of their respective partners, divided by two times the number of men/women squared (resulting in a normalization, e. g. it only takes values between zero and one). This should somehow give information about the optimality of the outcome, although it does not correspond to definition 2 in section 4.1. A low optimality index corresponds to a rather optimal assignment whereas a high optimality index indicates a not so optimal assignment.

[3] Gale and Shapley [1962, p. 11]

5 Implementation

5.1 Conventions

The nodes (men resp. women) are referenced by integers from one to n where n is the input size (number of men/women).

Accordingly, a preference list is a permutation of the first n integers: The element $\sigma(1)$ is the most preferred node, $\sigma(2)$ the second most preferred and so on. In the case of non-complete information, there are $m < n$ distinct integers followed by zeros to complete the sequence.

5.2 Generating the Preference Lists

5.2.1 Random

For testing purposes we first coded a generator for random preference matrices which basically calls `randperm(n)` n times, resulting in a matrix whose rows represent the preference list of a single node. The generated matrix can be used to simulate the classic case with complete information.

5.2.2 The Plane Algorithm

The question that now arises is the following: How can I generate a preference matrix with non-complete information that represents reality in an appropriate manner? In particular there should be a mechanism for controlling the number of non-zero entries in the preference lists (i. e. controlling the degree of completeness of information). The number of non-zero elements should, however, not be the same for each node (this would mean that each node knows the same fixed number of people, which is clearly not a good representation of real situations). Additionally, situations like 'man x knows woman y but woman y does not know man x' should be considered. To achieve these characteristics we developed the 'generatePlane' algorithm: It is based on the idea that a node only knows his neighbours (e. g. the people in his town or community). Therefore his preference list should consist only of nodes that are closer than a certain distance, which we will call the visibility radius in the further discussions. We realized this by assigning to each node a random position in a two dimensional plane. For simplicity they are distributed in $[0, 1] \times [0, 1]$.

Definition 3: *The visibility radius defines the neighbourhood of a node in* $[0, 1] \times [0, 1]$. *A node can only know other nodes that have a distance smaller than the visibility radius.*

To avoid border effects, the edges are connected (one could view it as a torus). For the actual generation of the preference lists one just has to iterate through all nodes and

determine all nodes of opposite sex that are in the disc of visibility radius. We then chose to use a random permutation of these neighbourhood nodes to keep it simple.

A constant visibility radius implies in particular that if node x knows node y, then also node y knows node x. But as we don't want this to be the case all the time we also made an option for a random visibility radius that is updated every step.

5.3 Making Matches

This is the part where the algorithm proposed by Gale and Shapley [1962] actually comes into play. An implementation in MATLAB is shown in listing 1 (The basic ideas for the implementation are taken from RosettaCode [2014] and adapted to MATLAB).

```
function [ engaged, stable ] = makeMatch( m, f )
%makeMatch finds engagements for preferences according to Gale–Shapley ...
    algorithm
...
freemen = [(1:n)',ones(n,1)];
engaged = zeros(n,2);
while ~isempty(find(freemen(:,2)==1,1))
    theman = find(freemen(:,2)==1,1);
    thegirl = m(theman,1);
    index = find(engaged(:,2)==thegirl,1);
    if(isempty(index) )
        engaged(theman,1) = theman;
        engaged(theman,2) = thegirl;
        freemen(theman,2) = 0;
    else
        fiance = engaged(index,1);
        girlprefers = f(thegirl,:);
        if(find(girlprefers==theman,1)<find(girlprefers==fiance,1))
            engaged(theman,1) = theman;
            engaged(theman,2) = thegirl;
            engaged(fiance,1) = 0;
            engaged(fiance,2) = 0;
            freemen(theman,2) = 0;
            freemen(fiance,2) = 1;
        else
            m(theman,:) = [m(theman,2:n) 0];
        end
    end
end
stable = checkEngagements(engaged,m,f);
...
end
```

Listing 1 : makeMatch code skeleton

The main structures are the arrays `freemen` and `engaged`. The array `freemen` contains all free men (first column: indices of men, second column: 1 for free, 0 for not free) and `engaged` contains the engagements produced by the algorithm (first column: indices of men, second column: indices of women). The main loop starts in line 6 and continues until all men are in an engagement. Then in the loop the first free man is picked (line 7) and his most preferred girl is determined (line 8). Now one has to distinguish between two cases:

- The girl has no engagement: Everything is fine, the man and the girl are now engaged (lines 11 and 12). Also the man is not free anymore (line 13).
- The girl alread has a fiance: In this case one has to do another distinction:
  - The girl prefers the new man to her fiance: A new engagement is made and the old one is cancelled (lines 18-21). One also has to update `freemen`.
  - The engagements remain unchanged, but the girl is removed from the man's preference list because she is not attainable for him.

In the end the engagements are checked with the `checkEngagements` algorithm described in section 5.4. However, when applying the modifications described in section 4.2 and still using this algorithm one runs into problems (as expected). Therefore we had to adapt the `makeMatch` algorithm to be able to handle the following situations:

- An unknown man proposes to a woman who is not engaged: We decided that in this case the proposing man should have a chance to succeed with his proposal, but this should not always be the case. Therefore we implemented a random decision with a certain probability for accepting (typically around 0.25). This can be seen as a simulation of the real-life situation 'the woman gets to know the unknown man and gets to like him (or not)'.
- An unknown node proposes to a woman who is engaged: We decided to apply the same procedure as above. One could argue that the probability for accepting should be lower because she already has a fiance, but we left that out for the sake of simplicity.
- The preference list of a man is empty but he is not engaged: In this case the man is just left with no partner.

The implementation of the above points will not be discussed here any further. The final algorithm is in the appendix for reference.

Finally we added the preference change functionality: In each iteration step a random decision is made between changing preferences or not, using the probability given in the additional parameter `changerate`. If the answer is positive, the preference list of one

randomly chosen node is perturbed a little: A random node in the preference list is chosen and switched with the node one rank lower (for example `[4,2,3,1]` becomes `[4,3,2,1]` after switching nodes 2 and 3). There is also a parameter to determine whether only men, only women or both should change their preferences.

5.4 Checking the Engagements

An important indicator for the later discussion is the stability of the engagements. It can be checked using this algorithm. The main loop is shown in listing 2 (again the basic structure is inspired by RosettaCode [2014])

```
1  while he<=n
2      she = engaged(he,2);
3      % make sure that the man has a partner i. e. she is not 0
4      % otherwise continue because nothing to check
5      ...
6      hisindex = find(f(she,:)==he,1);
7      herindex = find(m(he,:)==she,1);
8      helikesbetter = m(he,1:herindex);
9      shelikesbetter = f(she,1:hisindex);
10     % check for her
11     for i=1:size(shelikesbetter)
12         guy = shelikesbetter(i);
13         guysgirl = engaged(guy,2);
14         guylikes = m(guy,:);
15         if (find(guylikes==she,1)<find(guylikes==guysgirl,1))
16             stable = false;
17         end
18     end
19     % check for him
20     ...
21     he=he+1;
22 end
```

Listing 2 : checkEngagements code fragment

The main loop is an iteration over all men (line 1), but only those who are engaged because having no partner is not considered as an instability (lines 3-5). After having retrieved all the indices and preference lists, one iterates over all nodes that appear before the actual partner in their respective preference lists and checks whether there is an instability (see definition 1 in section 4.1). This results in two for loops, one for the man and one for the woman he is engaged to.

5.5 simulation

The actual simulation makes iterated calls to `generatePlane` and `makeMatch` to simulate the modified Gale-Shapley Algorithm for different parameter settings. The input parameters are:

- `n`: the input size $n \in \mathbb{N}$
- `mode, radius`: mode determines the choice of the visibility radius (either random or constant), and if the radius is constant then it can be set with the parameter radius $r \in [0, 0.5]$
- `changerate`: the rate at which preference changes are performed $c \in [0, 1]$
- `p`: determines whose preferences are changed $p \in \{0, 0.5, 1\}$

The resulting output variables are:

- no. of instabilities: number of instabilities according to definition 1 in section 4.1
- no. of dumps: number of fiances that have been dumped during match-making, see section 4.2
- no. of singles: number of single nodes that remain after match-making
- optimality index: as defined above, see section 4.2

6 Simulation Results and Discussion

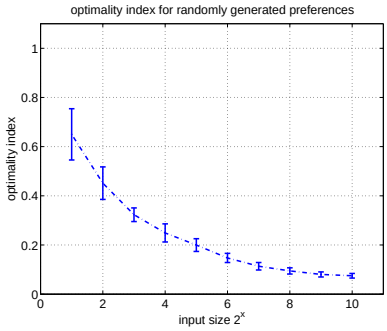
6.1 Instabilities

6.2 Singles

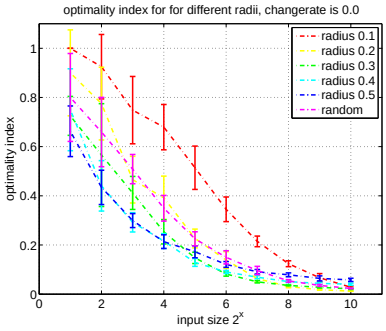
6.3 Optimality Index

Question: How does the optimality index change when the visibility radius and the preference change rate are varied? Is there a dependence on the input size?

We expect to find that that the optimality index increases for higher preference change rates.



(a) bla bla bla



(b) bla bla bla

7 Summary and Outlook

8 References

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):pp. 9–15, 1962. ISSN 00029890. URL `http://www.jstor.org/stable/2312726`.

RosettaCode. Stable marriage problem, 2014. URL `http://rosettacode.org/wiki/Stable_marriage_problem#Python`.

9 Appendix: MATLAB Codes

generateRandom.m

```
function [ m, f ] = generateRandom( n )
%GENERATERANDOM generates random preference matrices
m = zeros(n,n);
f = zeros(n,n);
for i=1:n
    m(i,:) = randperm(n,n);
    f(i,:) = randperm(n,n);

end
```

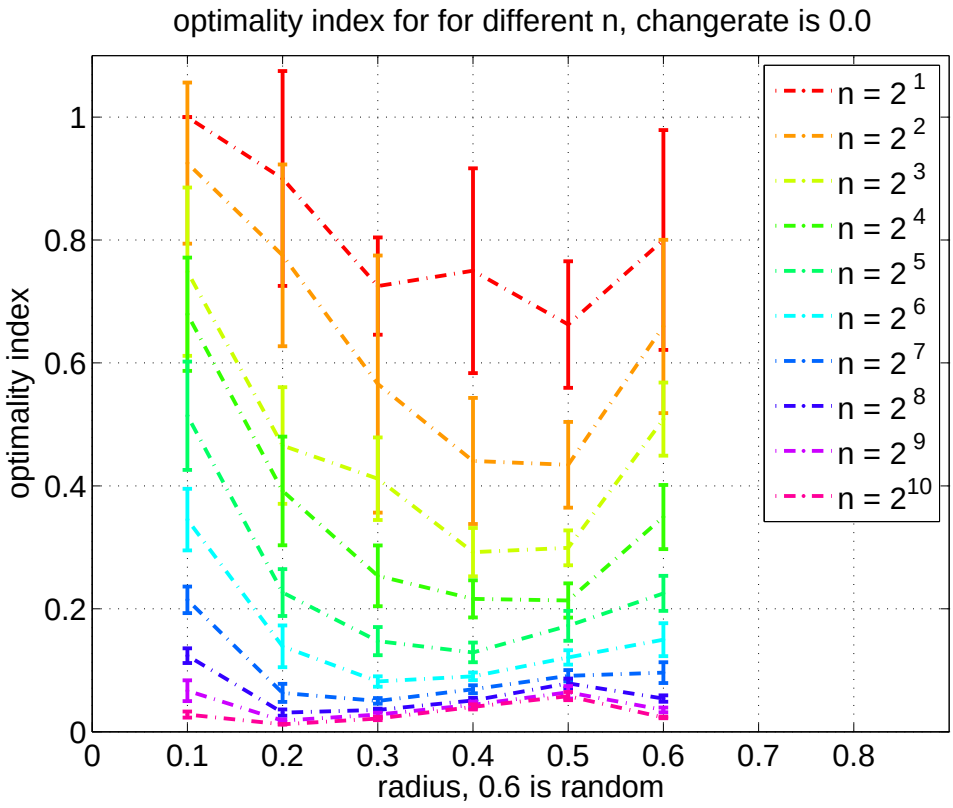


Figure 2: bla bla bla

generatePlane.m

```
function [ mpref,fpref ] = generatePlane( n ,mode, radius)
%GENERATEPLANE generates preference lists for men and women
%   based on a plane where women and men are represented by points
%   they have a limited visibility radius
%   n: number of men and women
%   mode: visibility radius mode, optional argument
%     1 --> const, one constant radius for all nodes
%     2 --> random, a new random radius is generated in each iteration
%           value is between 0.1 and 0.5
%   default mode is const
%   mpref: mens preferences in nxn matrix
```

```
%   fpref: womens preferences in nxn matrix

global verbosity

if (nargin >= 2 && mode == 1)
    assert(nargin==3);
    r = radius;
end
if(nargin < 2)
    mode = 1;
    r = 0.2;%default value
end

% generate random coordinates
% and extend to torus
men = zeros(3,9*n);
rnd = rand(2,n);
men(:,(0*n)+1:1*n)=[(1:n);rnd];
men(:,(1*n)+1:2*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
men(:,(2*n)+1:3*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
men(:,(3*n)+1:4*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
men(:,(4*n)+1:5*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
men(:,(5*n)+1:6*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
men(:,(6*n)+1:7*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
men(:,(7*n)+1:8*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
men(:,(8*n)+1:9*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];

women = zeros(3,9*n);
rnd = rand(2,n);
women(:,(0*n)+1:1*n)=[(1:n);rnd];
women(:,(1*n)+1:2*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
women(:,(2*n)+1:3*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
women(:,(3*n)+1:4*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
women(:,(4*n)+1:5*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
women(:,(5*n)+1:6*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
women(:,(6*n)+1:7*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
women(:,(7*n)+1:8*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
women(:,(8*n)+1:9*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];

%plotting
% if verbosity~=0
%     plot(men(2,1:n),men(3,1:n),'o',women(2,1:n),women(3,1:n),'o');
%     label1 = cellstr( num2str(women(1,1:n)') );
%     label2 = cellstr( num2str(men(1,1:n)') );
%     text(women(2,1:n),women(3,1:n),label1);
%     text(men(2,1:n),men(3,1:n),label2);
%     title('nodes in plane');
%     legend('men','women');
% end

```

```
d = zeros(2,9*n);
mpref = zeros(n,n);
fpref = zeros(n,n);


for i=1:n
    man = men(:,i);
    for j=1:9*n
        woman = women(:,j);
        d(:,j) = [woman(1,1);norm(man(2:3)-woman(2:3),2)];
    end
    if mode==2
        r = rand*0.4+0.1;
    end
    index = find(d(2,:)<r);
    available = women(:,index);
    sz = size(available,2);
    if sz>n
        available = available(:,1:n);
        sz = n;
    end
    perm = randperm(sz);
    mpref(i,1:sz) = available(1,perm);
end
for i=1:n
    woman = women(:,i);
    for j=1:9*n
        man = men(:,j);
        d(:,j) = [man(1,1);norm(man(2:3)-woman(2:3),2)];
    end
    if mode==2
        r = rand*0.4+0.1;
    end
    index = find(d(2,:)<r);
    available = men(:,index);
    sz = size(available,2);
    if sz>n
        available = available(:,1:n);
        sz = n;
    end
    perm = randperm(sz);
    fpref(i,1:sz) = available(1,perm);
end
end
```

vprintf.m

```
function vprintf(varargin)
% VPRINTF controlled printing
```

```
%
global verbosity
if verbosity~=0
    fprintf(varargin{:});
end
```

makeMatch.m

```
function [ engaged, output ] = makeMatch( m, f, changerate, p )
%makeMatch finds engagements for preferences according to Gale–Shapley ...
    algorithm
%
%   men an women encoded as integers from 1 to n
%
%   input:
%   m: preference matrix of the men. Each row corresponds to a man and
%   the elements are the women listed according to his preferences.
%   f: preference matrix of the women. Each row corresponds to a woman and
%   the elements are the men listed according to her preferences.
%   changerate: rate at which preference changes are performed, e. g. if
%   changerate=0.2 then only in 20% of iterations preferences are changed
%   p: change preferences for men (p=1) / women (p=0.5) / both (p=0.5)
%
%   dimensions must be correct, m=nxn, f=nxn.
%
%   returns:
%   engaged: nx2 Matrix containing matches
%   output: output data -->
%   output(1,1): number of instabilities
%   output(1,2): number of singles
%   output(1,3): number of dumps
%   output(1,4): optimality index

% optional test prints
global verbosity
vprintf('mens preferences:\n');
if verbosity~=0 disp(m); end
vprintf('womens preferences:\n')
if verbosity~=0 disp(f); end
% assign local variables
initialm = m;
initialf = f;
n = size(m,1);
n2 = size(f,1);
% make sure dimensions agree
assert(n == size(m,2));
assert(n==n2);
if nargin > 2
    assert(nargin==4);
```

```
41      assert(changerate<=1);
42      assert(changerate>=0);
43      assert(~isempty(find(p==[0,1,0.5],1)));
44  end
45  % more local variables
46  freemen = [(1:n)',ones(n,1)]; % column 1= men; column 2= 1 -> man is free, ...
        0 -> man isn't free
47  engaged = zeros(n,2);% column 1= men; column 2= women
48  dumped=0; % no of dumps
49  acceptrate = 0.75; % rate at which unknown nodes are accepted
50  % main loop
51  while ~isempty(find(freemen(:,2)==1,1)) % iterate as long as there are free men
52      % preference changes
53      if nargin > 2 % only if changerate and p are given
54          if rand < changerate % change prefs?
55              node = randi(n); % node whose prefs to change
56              if rand < p % change for men or women
57                  pref = nonzeros(m(node,:))';
58                  len = size(pref,2);
59                  if len>1
60                      k = randi([2,len]); % where in pref to change
61                      girl1 = pref(k); % the girl to swap
62                      i1 = find(initialm(node,:)==girl1,1); % index of girl1 ...
                            in initialm
63                      girl2 = m(node, k-1); % girl to be swapped with
64                      i2 = find(initialm(node,:)==girl2,1); % index of girl2 ...
                            in initialm
65                      initialm(node, i2) = girl1;
66                      initialm(node, i1) = girl2;
67                      m(node, i1) = girl2;
68                      m(node, i1-1) = girl1;
69                  end
70              else
71                  pref = nonzeros(f(node,:))';
72                  len = size(pref,2);
73                  if len>1
74                      k = randi([2,len]); % where in pref to change
75                      man1 = pref(k); % the man to swap
76                      i1 = find(initialf(node,:)==man1,1); % index of man1 in ...
                            initialf
77                      man2 = f(node, k-1); % man to be swapped with
78                      i2 = find(initialf(node,:)==man2,1); % index of man2 in ...
                            initialf
79                      initialf(node, i2) = man1;
80                      initialf(node, i1) = man2;
81                      f(node, i1) = man2;
82                      f(node, i1-1) = man1;
83                  end
84              end %if_2
85              vprintf('preferences changed\n');
```

```
86          end %if_1
87      end
88      % +++
89      theman = find(freemen(:,2)==1,1); % the first man free on the list
90      thegirl = m(theman,1); % his first choice
91          if thegirl==0; % theman doesn't know any free girls who want him, ...
                he'll be alone :(
92              freemen(theman,2)=0;
93              engaged(theman,:)=0;
94          else
95              index = find(engaged(:,2)==thegirl,1); % index of possible ...
                    fiance of his first choice
96              if(isempty(index) ) % thegirl is free -> theman will be engaged ...
                    to thegirl
97                  if isempty(find(f(thegirl,:)==theman,1))
98                      vprintf('man %d proposed to women %d, she does not know ...
                            him\n', theman, thegirl);
99                      if rand>acceptrate % man accepts with a certain rate
100                         engaged(theman,1) = theman; % make new engagement
101                         engaged(theman,2) = thegirl;
102                         vprintf('she accepts\nman %d is engaged to girl ...
                                %d\n', theman, thegirl);
103                         freemen(theman,2) = 0; % man is not free anymore
104                         f(thegirl,:) = [theman, f(thegirl,1:n-1)]; % update ...
                                preferences
105                         initialf(thegirl,:) = [theman, ...
                                initialf(thegirl,1:n-1)]; % also in initial ...
                                matrix (will be used for checking)
106                     else
107                         vprintf('she declines\n');
108                         m(theman,:) = [m(theman,2:n) 0]; % make pref list ...
                                of theman smaller
109                     end % if_4
110                 else
111                     engaged(theman,1) = theman; % make new engagement
112                     engaged(theman,2) = thegirl;
113                     vprintf('man %d is engaged to girl %d\n', theman, thegirl);
114                     freemen(theman,2) = 0; % man is not free anymore
115                 end % if_3
116             else % thegirl is already engaged -> check if thegirl prefers ...
                    theman to her fiance
117                 fiance = engaged(index,1); % her fiance
118                 girlprefers = f(thegirl,:); % pref list of thegirl
119                 howgirllikestheman=find(girlprefers==theman,1); % themans ...
                        index on thegirls preferences list
120                 howgirllikesfiance=find(girlprefers==fiance,1); % fiances ...
                        index on thegirls preferences list
121                 if(isempty(howgirllikestheman)) % thegirl doesn't know ...
                        theman -> thegirl accepts with a certain rate
122                     if rand > 0.75
```

```
123                         % thegirl prefers theman -> update pref list
124                         f(thegirl,:) = [f(thegirl,1:howgirllikesfiance), ...
                                theman, f(thegirl,howgirllikesfiance+1:n-1)];
125                         initialf(thegirl,:) = ...
                                [initialf(thegirl,1:howgirllikesfiance), ...
                                theman, ...
                                initialf(thegirl,howgirllikesfiance+1:n-1)]; % ...
                                also initial
126                     end % if_4
127                 end % if_3
128                 if(find(girlprefers==theman,1)<find(girlprefers==fiance,1)) ...
                        % thegirl prefers theman ->change engagement
129                     engaged(theman,1) = theman; % change fiance of the girl
130                     engaged(theman,2) = thegirl;
131                     engaged(fiance,1) = 0; % fiance is free again
132                     engaged(fiance,2) = 0;
133                     vprintf('girl %d dumped man %d for man %d\n', thegirl, ...
                            fiance, theman);
134                     dumped=dumped+1;
135                     freemen(theman,2) = 0;
136                     freemen(fiance,2) = 1;
137                 else
138                     m(theman,:) = [m(theman,2:n) 0]; % thegirl prefers her ...
                            fiance -> take thegirl out of themans preference list
139                 end % if_3
140             end % if_2
141         end % if_1
142 end % while
143 % result printing (suppressed if verbositiy set to 0)
144 if dumped==1
145     vprintf('\n%d man has been dumped for another\n\n', dumped);
146 else
147     vprintf('\n%d men have been dumped for others\n\n', dumped);
148 end % if
149 single = size(find(engaged(:,2)==0),1); % number of single nodes
150 if single==1
151     vprintf('There is %d single man/woman\n\n', single);
152 else
153     vprintf('There are %d single men/women\n\n', single);
154 end % if
155 [stable, counter] = checkEngagements(engaged,initialm,initialf); % check ...
        the engagements
156 if (stable)
157     vprintf('marriages are stable\n');
158 else
159     vprintf('marriages are unstable\n');
160     if counter==1
161         vprintf('there is %d unstable mariage\n', counter);
162     else
163        vprintf('there are %d unstable mariages\n', counter);
```

```
    end % if_2
end % if
% calculate optimality index
opt = 0;
for i = 1:n
    he = i;
    she = engaged(he,2);
    if she~=0
        hisindex = find(initialf(she,:)==he,1);
        herindex = find(initialm(he,:)==she,1);
    else
        hisindex = n;
        herindex = n;
    end
    opt = opt + hisindex + herindex;
end
opt = opt/(2*n*n);
vprintf('optimality index is %1.2f\n',opt);
% set output
output = zeros(1,4);
output(1,1) = counter;
output(1,2) = single;
output(1,3) = dumped;
output(1,4) = opt;
end
```

checkEngagements.m

```
function [ stable,counter ] = checkEngagements( engaged, m, f )
%checkEngagements checks whether a set of engagements is stable
%
%   men an women encoded as integers from 1 to n
%
%   input:
%   engaged: engagement matrix
%   m,f: preference matrices
%
%   dimensions must be correct, m=nxn, f=nxn, engaged=nx2
%
%   returns:
%   stable: true for stable engagements, false otherwise
%   counter: the number of unstable mariages

n = size(m,1); % input size
% reverse the engaged matrix such that the new matrix has the index of the
% women on the column one and those of their respective husbands in row two
invengaged=zeros(n,2);
copy = engaged(:,[2,1]);
i=1;
```

```
22 while i~=n+1
23     index=copy(i,1);
24     while index==0 && i~=n % find first index that is nonzero
25         i=i+1;
26         index=copy(i,1);
27     end % while
28     if index==0 && i==n
29         break;
30     end % if
31     invengaged(index,:)=copy(i,:);
32     i=i+1;
33 end % while
34 % assign local variables
35 stable=true;
36 he=1;
37 counter=0;
38 inst = [0,0];
39 % main loop
40 while he<=n
41     she = engaged(he,2); % she is engaged to he
42     while (she==0 && he~=n) % he is not engaged, so there is no instability ...
           -> check the next man
43         he = he+1;
44         she = engaged(he,2);
45     end % while
46     if she==0 % -> he=n is not engaged, nothing to check.
47         break;
48     end %if
49     % get indexes in pref lists
50     hisindex = find(f(she,:)==he,1);
51     herindex = find(m(he,:)==she,1);
52     helikesbetter = m(he,1:herindex);
53     shelikesbetter = f(she,1:hisindex);
54     % check for her
55     if ~isempty(shelikesbetter) % there is no one on earth she likes better
56         for i=1:size(shelikesbetter) % loop to check if there is ...
               unstability for the girl
57             guy = shelikesbetter(i); % all the guys she likes better
58             guysgirl = engaged(guy,2); % the guy she is engaged to
59             if guysgirl == 0 && ~isempty(find(m(guy,:)== she,1)) % if this ...
                   guy isn't engaged, then she could be with him -> unstable, ...
                   unless he doesn't know her.
60                 stable = false;
61                 vprintf('man %d and woman %d like each other better\n', guy, ...
                       she);
62                 inst = [guy,she;inst];
63
64             else
65                 guylikes = m(guy,:); % the ordered preferences of guy
66                 if (find(guylikes==she,1)<find(guylikes==guysgirl,1)) % if ...
```

```
                    guy also likes she better than his wife -> unstable
67                  stable = false;
68                  vprintf('man %d and woman %d like each other better\n', ...
                        guy, she);
69                  inst = [guy,she;inst];
70              end % if_3
71          end % if_2
72      end % for
73  end % if_1
74  % now the other way round, check for him
75  if ~isempty(helikesbetter) % there is no one on earth he likes better
76      for i=1:size(helikesbetter) % loop to check if there is unstability ...
            for the man
77          girl = helikesbetter(i); % all the girls he likes better
78          girlsguy = invengaged(girl,2); % the girl he is engaged to
79          if girlsguy == 0 && ~isempty(find(f(girl,:)== he,1))% if this ...
                girl isn't engaged, then she could be with her -> unstable
80              stable = false;
81              vprintf('man %d and woman %d like each other better\n', he, ...
                    girl);
82              inst = [he,girl;inst];
83          else
84              girllikes = f(girl,:);% the ordered preferences of girl
85              if (find(girllikes==he,1)<find(girllikes==girlsguy,1)) % if ...
                    guy also likes she better than his wife -> unstable
86                  stable = false;
87                  vprintf('man %d and woman %d like each other better\n', ...
                        he, girl);
88                  inst = [he,girl;inst];
89              end % if_3
90          end % if_2
91      end % for
92  end % if_1
93
94      he=he+1; % go to the next man
95  end % while
96  % delete duplicate instabilities
97  inst = unique(inst, 'rows');
98  counter = size(inst,1)-1;
99  end
```

simulation.m

```
1   function [ data ] = simulation( saveit )
2   %simulation perform simulation
3   %
4   %   input:
5   %   saveit: if 1 data is saved, if 0 not
6   %
```

```
%   returns:
%   data: simulation data


%simulation

% simulate match making
% n is 2et, t from 1 to 6
% radius is either constant or random
%   when constant, in 0.1:0.05:0.5
% frequency

global verbosity
verbosity = 0;

assert(~isempty(find(saveit==[0,1],1)));
tmax = 6;
t = 2.^(1:tmax);
r = 0.1:0.05:0.5;
data = zeros(tmax,10,4);
seed = rng;
if saveit==1
    dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
    mkdir(dirname);
end

% radius random
for i=1:tmax
   n = t(i);
   [a,b] = generatePlane(n,2);
   [x,y] = makeMatch(a,b);
   data(i,10,:) = y;
end

% radius const
for i=1:tmax
    for j=1:9
        n = t(i);
        radius = r(j);
        [a,b] = generatePlane(n,1,radius);
        [x,y] = makeMatch(a,b);
        data(i,j,:) = y;
    end
end
% plot optimality index for each radius
hold on
handle = figure(1);
col = hsv(10);
%set(groot,'defaultAxesLineStyleOrder',{'-*',':','o'});
for i=1:10
```

```
57      plot(1:tmax,data(:,i,4),'color', col(i,:), 'marker', '*','linestyle','--');
58      title('optimality index for for different radiuses');
59
60  end
61  arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' ',' '];
62  xlabel('input size 2^x');
63  ylabel('optimality index');
64  legend([num2str(r','radius %1.3f');arr]);
65  if saveit==1
66      saveas(handle,sprintf('%s/figure_1.pdf', dirname));
67  end
68  hold off
69
70  % plot no of dumps for each radius
71  handle = figure(2);
72  for i=1:10
73      subplot(3,4,i);
74      bar(1:tmax,data(:,i,3));
75      xlabel('input size 2^x');
76      ylabel('number of dumps');
77      ylim([0,100]);
78      if i~=10
79          title(sprintf('plotting #dumps for radius %1.3f',r(i)));
80      else
81          title('plotting #dumps for radius random');
82      end
83
84  end
85  if saveit==1
86      saveas(handle,sprintf('%s/figure_2.pdf', dirname));
87  end
88  % saving
89  if (saveit==1)
90      save(sprintf('%s/data.mat',dirname),'data','seed');
91  end
92
93  end
```

simulation2.m

```
1  function [ data ] = simulation2( saveit )
2  %simulation perform simulation
3  %
4  %   input:
5  %   saveit: if 1 data is saved, if 0 not
6  %
7  %   returns:
8  %   data: simulation data
9
```

```
10
11 %simulation
12
13 % simulate match making
14 % n is 2et, t from 1 to 6
15 % radius is either constant or random
16 %   when constant, r=0.4
17 %makeMatch with preference changes for men, women, men/women and
18 %probability of changing in p=0.1:0.1:0.9
19
20 global verbosity
21 verbosity = 0;
22
23 assert(~isempty(find(saveit==[0,1],1)));
24 tmax = 11;
25 t = 2.^(1:tmax);
26 p = 0.0:0.25:1.0;
27 pmax=size(p,2);
28 m=12; %number of different makeMatch called in simulation2
29 data = zeros(tmax,m*pmax,4);
30 r= 0.1:0.15:0.4;
31 seed = rng;
32 if saveit==1
33     dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
34     mkdir(dirname);
35 end
36
37 % radius random
38 for i=1:tmax
39     n = t(i);
40     [a,b] = generatePlane(n,2);
41     fprintf('Plane with rand radius, size %d is generated\n', n);
42     for j=1:pmax
43        [x1,y1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
              #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
              optimality index
44        [x2,y2] = makeMatch(a,b, p(j),0);
45        [x3,y3] = makeMatch(a,b, p(j),0.5);
46        data(i,j,:) = y1;
47        data(i,pmax+j,:)=y2;
48        data(i,2*pmax+j,:)=y3;
49     end
50 end
51
52 %constant radius in r=[0.1,0.25,0.4]
53 for i=1:tmax
54     n = t(i);
55     [a,b] = generatePlane(n,1, r(1));
56     fprintf('Plane with radius r=0.1, size %d is generated\n', n);
57     for j=1:pmax
```

```
58          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                optimality index
59          [v2,w2] = makeMatch(a,b, p(j),0);
60          [v3,w3] = makeMatch(a,b, p(j),0.5);
61          data(i,3*pmax+j,:) = w1;
62          data(i,4*pmax+j,:) = w2;
63          data(i,5*pmax+j,:) = w3;
64      end
65  end
66
67  for i=1:tmax
68      n = t(i);
69      [a,b] = generatePlane(n,1, r(2));
70      fprintf('Plane with radius r=0.25, size %d is generated\n', n);
71      for j=1:pmax
72          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                optimality index
73          [v2,w2] = makeMatch(a,b, p(j),0);
74          [v3,w3] = makeMatch(a,b, p(j),0.5);
75          data(i,6*pmax+j,:) = w1;
76          data(i,7*pmax+j,:) = w2;
77          data(i,8*pmax+j,:) = w3;
78      end
79  end
80
81  for i=1:tmax
82      n = t(i);
83      [a,b] = generatePlane(n,1, r(3));
84      fprintf('Plane with radius r=0.4, size %d is generated\n', n);
85      for j=1:pmax
86          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                optimality index
87          [v2,w2] = makeMatch(a,b, p(j),0);
88          [v3,w3] = makeMatch(a,b, p(j),0.5);
89          data(i,9*pmax+j,:) = w1;
90          data(i,10*pmax+j,:) = w2;
91          data(i,11*pmax+j,:) = w3;
92      end
93  end
94
95
96
97
98  % saving
99  if (saveit==1)
100     save(sprintf('%s/data.mat',dirname),'data','seed');
101 end
```

```
102
103 end
```

simulation3.m

```
1  function [ data ] = simulation3( saveit , tmax)
2  %simulation perform simulation
3  %
4  %   input:
5  %   saveit: if 1 data is saved, if 0 not
6  %   tmax: maximal exponent form input size (base 2)
7  %
8  %   returns:
9  %   data: simulation data
10
11 % simulate match making
12 % n is 2^t, t from 1 to tmax
13 % radius is either constant or random
14 %   when constant, in 0.1:rstep:0.5
15 % frequency changerate 0:fstep:1
16 % in makeMatch argument list p=0.5 -> change pref for both men and women
17
18 global verbosity
19 verbosity = 0;
20
21 assert(~isempty(find(saveit==[0,1],1)));
22 if nargin < 2
23     tmax = 6;
24 end
25 t = 2.^(1:tmax);
26 rstep = 0.1;
27 r = 0.1:rstep:0.5;
28 fstep = 0.5;
29 f = 0:fstep:1;
30 sizef = size(f,2);
31 sizer = size(r,2)+1;
32 m = 10; % number of iterations
33 data = zeros(tmax,sizer,sizef,m,4);
34 % data dimensions:
35 % 1 input size n, tmax
36 % 2 radius, sizer
37 % 3 frequency, sizef
38 % 4 iterations, m
39 % 5 output values, 4
40 seed = rng;
41 disp(seed);
42 if saveit==1
43     dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
44     mkdir(dirname);
```

```
end
tic
fprintf('simulating for radius random\n');
% radius random
for i=1:tmax
    n = t(i);
    for k=1:sizef
        freq = f(k);
        for l=1:m
            %[a,b] = generatePlane(n,2);
            [a,b] = generateRandom(n);
            fprintf('.');
            [x,y] = makeMatch(a,b,freq,0.5);
            data(i,sizer,k,l,:) = y;
        end
        fprintf('\n');
    end
    fprintf('n = %4d complete after %5.1f\n', n, toc);
end
if 1==0
fprintf('simuation for radius const\n')
% radius const
for i=1:tmax
    n = t(i);
    for j=1:sizer-1
        radius = r(j);
        for k=1:sizef
            freq = f(k);
            for l=1:m
                %[a,b] = generatePlane(n,1,radius);
                [a,b] = generateRandom(n);
                fprintf('.');
                [x,y] = makeMatch(a,b,freq,0.5);
                data(i,j,k,l,:) = y;
            end
            fprintf('\n');
        end
        fprintf('radius %1.1f complete\n', radius);
    end
    fprintf('n = %4d complete after %5.1f\n', n, toc);
end
end
% plotting
plot3(data);
% saving
if (saveit==1)
    save(sprintf('%s/data.mat',dirname),'data','seed');
end

end
```

iteratesimulation.m

```
function [data, dataplot] = iteratesimulation(saveit, n)
    global verbosity
    verbosity=0;

    %!!!tmax, m, p, t, pmax must correspond to simulation2!!!
    assert(~isempty(find(saveit==[0,1],1)));
    seed = rng;
    tmax=11;
    m=12; %number of makeMatch called in simulation2
    p = 0.0:0.25:1.0;
    pmax=size(p,2);
    t = 2.^(1:tmax);
    data = zeros(tmax,m*pmax,4,n);
    dataplot=zeros(tmax,m*pmax,4);
    if saveit==1
        dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
        mkdir(dirname);
    end
    for i=1:n
        data(:,:,:,i)=simulation2(0);
        dataplot=dataplot+data(:,:,:,i);
        fprintf('iteration %d is done\n', i);
    end
    dataplot=dataplot/(n*1.);

% plot no of unstability for each probability, with change preferences for
% men, with random radius
hold on
handle = figure(1);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,1:pmax,1));
        if i==1
            xlabel('probability (man)');
            ylabel('# instabilities');
        end
        ylim([0,10]);
        xlim([0,1]);
        title(sprintf('input size %1.0f',t(i)));
    end

    if i<=2*tmax && i>tmax
        bar(p,dataplot(i-tmax,pmax+1:2*pmax,1));
        if i==tmax+1
            xlabel('probability (women)');
            ylabel('# instabilities');
```

```
        end
        ylim([0,10]);
        xlim([0,1]);
    end

    if i>2*tmax
        bar(p,dataplot(i-2*tmax,2*pmax+1:3*pmax,1));
        if i==2*tmax+1
            xlabel('probability (men/women)');
            ylabel('# instabilities');
        end
        ylim([0,10]);
        xlim([0,1]);
    end


end
if saveit==1
    saveas(handle,sprintf('%s/(1)instability with random radius.pdf', ...
        dirname));
end
hold off

% plot no of unstability for each probability, with change preferences for
% men, with constant radius=0.1
hold on
handle = figure(2);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,3*pmax+1:4*pmax,1));
        if i==1
            xlabel('probability (man)');
            ylabel('# instabilities');
        end
        ylim([0,10]);
        xlim([0,1]);
        title(sprintf('input size %1.0f',t(i)));
    end

    if i<=2*tmax && i>tmax
        bar(p,dataplot(i-tmax,4*pmax+1:5*pmax,1));
        if i==tmax+1
            xlabel('probability (women)');
            ylabel('# instabilities');
        end
        ylim([0,10]);
        xlim([0,1]);
    end

```

```
97      if i>2*tmax
98          bar(p,dataplot(i-2*tmax,5*pmax+1:6*pmax,1));
99          if i==2*tmax+1
100             xlabel('probability (men/women)');
101             ylabel('# instabilities');
102         end
103         ylim([0,10]);
104         xlim([0,1]);
105     end
106 end
107 if saveit==1
108     saveas(handle,sprintf('%s/(2)instability with constant radius=0.1.pdf', ...
            dirname));
109 end
110 hold off
111
112 % plot no of unstability for each probability, with change preferences for
113 % men, with constant radius=0.25
114 hold on
115 handle = figure(3);
116 for i=1:3*tmax
117     subplot(3,tmax,i);
118     if i<=tmax
119         bar(p,dataplot(i,6*pmax+1:7*pmax,1));
120         if i==1
121             xlabel('probability (man)');
122             ylabel('# instabilities');
123         end
124         ylim([0,10]);
125         xlim([0,1]);
126         title(sprintf('input size %1.0f',t(i)));
127     end
128
129     if i<=2*tmax && i>tmax
130         bar(p,dataplot(i-tmax,7*pmax+1:8*pmax,1));
131         if i==tmax+1
132             xlabel('probability (women)');
133             ylabel('# instabilities');
134         end
135         ylim([0,10]);
136         xlim([0,1]);
137     end
138
139     if i>2*tmax
140         bar(p,dataplot(i-2*tmax,8*pmax+1:9*pmax,1));
141         if i==2*tmax+1
142             xlabel('probability (men/women)');
143             ylabel('# instabilities');
144         end
145         ylim([0,10]);
```

```
146         xlim([0,1]);
147     end
148 end
149 if saveit==1
150     saveas(handle,sprintf('%s/(3)instability with constant ...
        radius=0.25.pdf', dirname));
151 end
152 hold off
153
154
155 % plot no of unstability for each probability, with change preferences for
156 % men, with constant radius=0.4
157 hold on
158 handle = figure(4);
159 for i=1:3*tmax
160     subplot(3,tmax,i);
161     if i<=tmax
162         bar(p,dataplot(i,9*pmax+1:10*pmax,1));
163         if i==1
164             xlabel('probability (man)');
165             ylabel('# instabilities');
166         end
167         ylim([0,10]);
168         xlim([0,1]);
169         title(sprintf('input size %1.0f',t(i)));
170     end
171
172     if i<=2*tmax && i>tmax
173         bar(p,dataplot(i-tmax,10*pmax+1:11*pmax,1));
174         if i==tmax+1
175             xlabel('probability (women)');
176             ylabel('# instabilities');
177         end
178         ylim([0,10]);
179         xlim([0,1]);
180     end
181
182     if i>2*tmax
183         bar(p,dataplot(i-2*tmax,11*pmax+1:12*pmax,1));
184         if i==2*tmax+1
185             xlabel('probability (men/women)');
186             ylabel('# instabilities');
187         end
188         ylim([0,10]);
189         xlim([0,1]);
190     end
191 end
192
193 if saveit==1
194     saveas(handle,sprintf('%s/(4)instability with constant radius=0.4.pdf', ...
```

```
        dirname));
end
hold off


% plot no of dumps for each probability, with change preferences for
% men, with random radius
hold on
handle = figure(5);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,1:pmax,3));
        if i==1
            xlabel('probability (man)');
            ylabel('# dumps');
        end
        xlim([0,1]);
        title(sprintf('input size %1.0f',t(i)));
    end

    if i<=2*tmax && i>tmax
        bar(p,dataplot(i-tmax,pmax+1:2*pmax,3));
        if i==tmax+1
            xlabel('probability (women)');
            ylabel('# dumps');
        end
        xlim([0,1]);
    end

    if i>2*tmax
        bar(p,dataplot(i-2*tmax,2*pmax+1:3*pmax,3));
        if i==2*tmax+1
            xlabel('probability (men/women)');
            ylabel('# dumps');
        end
        xlim([0,1]);
    end


end
if saveit==1
    saveas(handle,sprintf('%s/(5)dumps with random radius.pdf', dirname));
end
hold off

% plot no of dumps for each probability, with change preferences for
% men, with constant radius=0.1
hold on
```

```
handle = figure(6);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,3*pmax+1:4*pmax,3));
        if i==1
            xlabel('probability (man)');
            ylabel('# dumps');
        end
        xlim([0,1]);
        title(sprintf('input size %1.0f',t(i)));
    end

    if i<=2*tmax && i>tmax
        bar(p,dataplot(i-tmax,4*pmax+1:5*pmax,3));
        if i==tmax+1
            xlabel('probability (women)');
            ylabel('# dumps');
        end
        xlim([0,1]);
    end

    if i>2*tmax
        bar(p,dataplot(i-2*tmax,5*pmax+1:6*pmax,3));
        if i==2*tmax+1
            xlabel('probability (men/women)');
            ylabel('# dumps');
        end
        xlim([0,1]);
    end
end
if saveit==1
    saveas(handle,sprintf('%s/(6)dumps with constant radius=0.1.pdf', ...
        dirname));
end
hold off

% plot no of dumps for each probability, with change preferences for
% men, with constant radius=0.25
hold on
handle = figure(7);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,6*pmax+1:7*pmax,3));
        if i==1
            xlabel('probability (man)');
            ylabel('# dumps');
        end
        xlim([0,1]);
```

```
293         title(sprintf('input size %1.0f',t(i)));
294     end
295
296     if i<=2*tmax && i>tmax
297         bar(p,dataplot(i-tmax,7*pmax+1:8*pmax,3));
298         if i==tmax+1
299             xlabel('probability (women)');
300             ylabel('# dumps');
301         end
302         xlim([0,1]);
303     end
304
305     if i>2*tmax
306         bar(p,dataplot(i-2*tmax,8*pmax+1:9*pmax,3));
307         if i==2*tmax+1
308             xlabel('probability (men/women)');
309             ylabel('# dumps');
310         end
311         xlim([0,1]);
312     end
313 end
314 if saveit==1
315     saveas(handle,sprintf('%s/(7)dumps with constant radius=0.25.pdf', ...
          dirname));
316 end
317 hold off
318
319
320 % plot no of dumps for each probability, with change preferences for
321 % men, with constant radius=0.4
322 hold on
323 handle = figure(8);
324 for i=1:3*tmax
325     subplot(3,tmax,i);
326     if i<=tmax
327         bar(p,dataplot(i,9*pmax+1:10*pmax,3));
328         if i==1
329             xlabel('probability (man)');
330             ylabel('# dumps');
331         end
332         xlim([0,1]);
333         title(sprintf('input size %1.0f',t(i)));
334     end
335
336     if i<=2*tmax && i>tmax
337         bar(p,dataplot(i-tmax,10*pmax+1:11*pmax,3));
338         if i==tmax+1
339             xlabel('probability (women)');
340             ylabel('# dumps');
341         end
```

```
342         xlim([0,1]);
343     end
344
345     if i>2*tmax
346         bar(p,dataplot(i-2*tmax,11*pmax+1:12*pmax,3));
347         if i==2*tmax+1
348             xlabel('probability (men/women)');
349             ylabel('# dumps');
350         end
351         xlim([0,1]);
352     end
353 end
354
355 if saveit==1
356     saveas(handle,sprintf('%s/(8)dumps with constant radius=0.4.pdf', ...
            dirname));
357 end
358 hold off
359
360 % plot no of single men/women for each probability, with change preferences for
361 % men, with random radius
362 hold on
363 handle = figure(9);
364 for i=1:3*tmax
365     subplot(3,tmax,i);
366     if i<=tmax
367         bar(p,dataplot(i,1:pmax,2));
368         if i==1
369             xlabel('probability (man)');
370             ylabel('# singles');
371         end
372         xlim([0,1]);
373         title(sprintf('input size %1.0f',t(i)));
374     end
375
376     if i<=2*tmax && i>tmax
377         bar(p,dataplot(i-tmax,pmax+1:2*pmax,2));
378         if i==tmax+1
379             xlabel('probability (women)');
380             ylabel('# singles');
381         end
382         xlim([0,1]);
383     end
384
385     if i>2*tmax
386         bar(p,dataplot(i-2*tmax,2*pmax+1:3*pmax,2));
387         if i==2*tmax+1
388             xlabel('probability (men/women)');
389             ylabel('# singles');
390         end
```

```
391         xlim([0,1]);
392     end
393
394
395 end
396 if saveit==1
397     saveas(handle,sprintf('%s/(9) singles with random radius.pdf', dirname));
398 end
399 hold off
400
401 % plot no of single for each probability, with change preferences for
402 % men, with constant radius=0.1
403 hold on
404 handle = figure(10);
405 for i=1:3*tmax
406     subplot(3,tmax,i);
407     if i<=tmax
408         bar(p,dataplot(i,3*pmax+1:4*pmax,2));
409         if i==1
410             xlabel('probability (man)');
411             ylabel('# singles');
412         end
413         xlim([0,1]);
414         title(sprintf('input size %1.0f',t(i)));
415     end
416
417     if i<=2*tmax && i>tmax
418         bar(p,dataplot(i-tmax,4*pmax+1:5*pmax,2));
419         if i==tmax+1
420             xlabel('probability (women)');
421             ylabel('# singles');
422         end
423         xlim([0,1]);
424     end
425
426     if i>2*tmax
427         bar(p,dataplot(i-2*tmax,5*pmax+1:6*pmax,2));
428         if i==2*tmax+1
429             xlabel('probability (men/women)');
430             ylabel('# singles');
431         end
432         xlim([0,1]);
433     end
434 end
435 if saveit==1
436     saveas(handle,sprintf('%s/(10)singles with constant radius=0.1.pdf', ...
        dirname));
437 end
438 hold off
439
```

```
% plot no of single for each probability, with change preferences for
% men, with constant radius=0.25
hold on
handle = figure(11);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,6*pmax+1:7*pmax,2));
        if i==1
            xlabel('probability (man)');
            ylabel('# singles');
        end
        xlim([0,1]);
        title(sprintf('input size %1.0f',t(i)));
    end

    if i<=2*tmax && i>tmax
        bar(p,dataplot(i-tmax,7*pmax+1:8*pmax,2));
        if i==tmax+1
            xlabel('probability (women)');
            ylabel('# singles');
        end
        xlim([0,1]);
    end

    if i>2*tmax
        bar(p,dataplot(i-2*tmax,8*pmax+1:9*pmax,2));
        if i==2*tmax+1
            xlabel('probability (men/women)');
            ylabel('# singles');
        end
        xlim([0,1]);
    end
end
if saveit==1
    saveas(handle,sprintf('%s/(11)singles with constant radius=0.25.pdf', ...
        dirname));
end
hold off


% plot no of single for each probability, with change preferences for
% men, with constant radius=0.4
hold on
handle = figure(12);
for i=1:3*tmax
    subplot(3,tmax,i);
    if i<=tmax
        bar(p,dataplot(i,9*pmax+1:10*pmax,2));
        if i==1
```

```
489             xlabel('probability (man)');
490             ylabel('# singles');
491         end
492         xlim([0,1]);
493         title(sprintf('input size %1.0f',t(i)));
494     end
495
496     if i<=2*tmax && i>tmax
497         bar(p,dataplot(i-tmax,10*pmax+1:11*pmax,2));
498         if i==tmax+1
499             xlabel('probability (women)');
500             ylabel('# singles');
501         end
502         xlim([0,1]);
503     end
504
505     if i>2*tmax
506         bar(p,dataplot(i-2*tmax,11*pmax+1:12*pmax,2));
507         if i==2*tmax+1
508             xlabel('probability (men/women)');
509             ylabel('# singles');
510         end
511         xlim([0,1]);
512     end
513 end
514
515 if saveit==1
516     saveas(handle,sprintf('%s/(12)singles with constant radius=0.4.pdf', ...
        dirname));
517 end
518 hold off
519
520 %plot no of single over no of person with  random radius
521 hold on
522 handle = figure(13);
523 for i=1:3*pmax
524     subplot(3,pmax,i);
525     if i<=pmax
526         bar(1:tmax,dataplot(:,i,2));
527         xlabel('input size 10^x');
528         ylabel('# singles');
529         title(sprintf('probability (man) %1.3f %', p(i)));
530     end
531
532     if i<=2*pmax && i>pmax
533         bar(1:tmax,dataplot(:,i,2));
534         xlabel('input size 10^x');
535         ylabel('# singles');
536         title(sprintf('probability (women) %1.3f %', p(i-pmax)));
537     end
```

```
538
539     if i>2*pmax
540         bar(1:tmax,dataplot(:,i,2));
541         xlabel('input size 10^x');
542         ylabel('# singles');
543         title(sprintf('probability (man/women) %1.3f %', p(i-2*pmax)));
544     end
545
546 end
547
548 if saveit==1
549     saveas(handle,sprintf('%s/(13)singles over person with random ...
        radius.pdf', dirname));
550 end
551 hold off
552
553
554
555 %plot no of single over no of person with radius=0.1
556 hold on
557 handle = figure(14);
558 for i=1:3*pmax
559     subplot(3,pmax,i);
560     if i<=pmax
561         bar(1:tmax,dataplot(:,3*pmax+i,2));
562         xlabel('input size 10^x');
563         ylabel('# singles');
564         title(sprintf('probability (man) %1.3f %', p(i)));
565     end
566
567     if i<=2*pmax && i>pmax
568         bar(1:tmax,dataplot(:,3*pmax+i,2));
569         xlabel('input size 10^x');
570         ylabel('# singles');
571         title(sprintf('probability (women) %1.3f %', p(i-pmax)));
572     end
573
574     if i>2*pmax
575         bar(1:tmax,dataplot(:,3*pmax+i,2));
576         xlabel('input size 10^x');
577         ylabel('# singles');
578         title(sprintf('probability (man/women) %1.3f %', p(i-2*pmax)));
579     end
580
581 end
582
583 if saveit==1
584     saveas(handle,sprintf('%s/(14)singles over person with radius=0.1.pdf', ...
        dirname));
585 end
```

```
586 hold off
587
588
589
590 %plot no of single over no of person with radius=0.25
591 hold on
592 handle = figure(15);
593 for i=1:3*pmax
594     subplot(3,pmax,i);
595     if i<=pmax
596         bar(1:tmax,dataplot(:,6*pmax+i,2));
597         xlabel('input size 10^x');
598         ylabel('# singles');
599         title(sprintf('probability (man) %1.3f %', p(i)));
600     end
601
602     if i<=2*pmax && i>pmax
603         bar(1:tmax,dataplot(:,6*pmax+i,2));
604         xlabel('input size 10^x');
605         ylabel('# singles');
606         title(sprintf('probability (women) %1.3f %', p(i-pmax)));
607     end
608
609     if i>2*pmax
610         bar(1:tmax,dataplot(:,6*pmax+i,2));
611         xlabel('input size 10^x');
612         ylabel('# singles');
613         title(sprintf('probability (man/women) %1.3f %', p(i-2*pmax)));
614     end
615
616 end
617
618 if saveit==1
619     saveas(handle,sprintf('%s/(15)singles over person with ...
        radius=0.25.pdf', dirname));
620 end
621 hold off
622
623 %plot no of single over no of person with radius=0.4
624 hold on
625 handle = figure(16);
626 for i=1:3*pmax
627     subplot(3,pmax,i);
628     if i<=pmax
629         bar(1:tmax,dataplot(:,9*pmax+i,2));
630         xlabel('input size 10^x');
631         ylabel('# singles');
632         title(sprintf('probability (man) %1.3f %', p(i)));
633     end
634
```

```
635     if i<=2*pmax && i>pmax
636         bar(1:tmax,dataplot(:,9*pmax+i,2));
637         xlabel('input size 10^x');
638         ylabel('# singles');
639         title(sprintf('probability (women) %1.3f %', p(i-pmax)));
640     end
641
642     if i>2*pmax
643         bar(1:tmax,dataplot(:,9*pmax+i,2));
644         xlabel('input size 10^x');
645         ylabel('# singles');
646         title(sprintf('probability (men/women) %1.3f %', p(i-2*pmax)));
647     end
648
649 end
650
651 if saveit==1
652     saveas(handle,sprintf('%s/(16)singles over person with radius=0.4.pdf', ...
          dirname));
653 end
654 hold off
655
656
657
658 % saving
659 if (saveit==1)
660     save(sprintf('%s/data.mat',dirname),'data','dataplot','seed');
661 end
662
663 end
```

plot3.m

```
1  function [  ] = plot3( data )
2  %UNTITLED Summary of this function goes here
3  %   Detailed explanation goes here
4  % plot optimality index for each radius
5  dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
6  mkdir(dirname);
7  % define arrays
8  sizer = size(data,2);
9  r=0.1:0.1:0.5;
10 f=0:0.5:1;
11 sizef=size(f,2);
12 tmax = size(data,1);
13 t=1:tmax;
14
15 % x-axis n, diff radius, plots freq
16 col = hsv(sizer);
```

```
for j=1:sizef
    freq = f(j);
    handle = figure(j);
    set(gca,'FontSize',16);
    hold on
    for i=1:sizer
        mm = squeeze(mean(data(:,i,j,:,4),4));
        st = squeeze(std(data(:,i,j,:,4),0,4));
        %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
            '*','linestyle','--');
        errorbar(1:tmax,mm,st,'color', col(i,:), 'marker', ...
            '.','linestyle','-.', 'LineWidth', 2);
    end
    hold off
    box on
    grid on
    title(sprintf('optimality index for for different radii, changerate is ...
        %1.1f',freq));
    arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' '];
    xlabel('input size 2^x');
    ylabel('optimality index');
    ylim([0,1.1]);
    xlim([0,11]);
    legend([num2str(r','radius %1.1f');arr]);
    saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j));
end

% x-axis radius, diff n, plots freq
col = hsv(tmax);
for j=1:sizef
    freq = f(j);
    handle = figure(j+3);
    set(gca,'FontSize',16);
    hold on
    for i=1:tmax
        mm = squeeze(mean(data(i,:,j,:,4),4));
        st = squeeze(std(data(i,:,j,:,4),0,4));
        %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
            '*','linestyle','--');
        errorbar(0.1:0.1:0.6,mm,st,'color', col(i,:), 'marker', ...
            '.','linestyle','-.', 'LineWidth', 2);
    end
    hold off
    box on
    grid on
    title(sprintf('optimality index for for different n, changerate is ...
        %1.1f',freq));
    %arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' ',' '];
    xlabel('radius, 0.6 is random');
    ylabel('optimality index');
```

```
    ylim([0,1.1]);
    xlim([0,0.9])
    legend(num2str(t','n = 2^{%2d}'));
    saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j+3));
end

% x-axis freq, diff n, plots radius
col = hsv(tmax);
for j=1:sizer
    if j~= sizer radius = r(j); end
    handle = figure(j+6);
    set(gca,'FontSize',16);
    hold on
    for i=1:tmax
        mm = squeeze(mean(data(i,j,:,:,4),4));
        st = squeeze(std(data(i,j,:,:,4),0,4));
        %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
            '*','linestyle','--');
        errorbar(f,mm,st,'color', col(i,:), 'marker', '.','linestyle','-.', ...
            'LineWidth', 2);
    end
    hold off
    box on
    grid on
    if j~= sizer
        title(sprintf('optimality index for for different n, radius ...
            %1.1f',radius));
    else
        title(sprintf('optimality index for for different n, radius random'));
    end
    %arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' ',' '];
    xlabel('changerate');
    ylabel('optimality index');
    ylim([0,1.1]);
    xlim([-0.1,1.4]);
    legend(num2str(t','n = 2^{%2d}'));
    saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j+6));
end

% surfs
handle = figure(13);
[a,b]=meshgrid(1:10,0.1:0.1:0.5);
hold on;
mm = squeeze(mean(data(:,1:5,3,:,4),4));
view(0,90);
box on
grid on
surf(a,b,mm','EdgeColor','none');
colorbar;
set(gca,'FontSize',16);
```

```
xlim([1,10]);
xlabel('input size2^x');
ylabel('radius');
saveas(handle,sprintf('%s/figure_13.pdf', dirname));
%generaterandom
handle = figure(14);
set(gca,'FontSize',16);
box on
dd=zeros(10,20,4);
for i=1:10
    n = 2^i;
    for j=1:20
        [a,b] = generateRandom(n);
        [x,y] = makeMatch(a,b);
        dd(i,j,:) = y;
        fprintf('.');
    end
    fprintf('\n');
end
mm = squeeze(mean(dd(:,:,4),2));
st = squeeze(std(dd(:,:,4),0,2));
errorbar(1:10,mm,st, 'marker', '.','linestyle','-.', 'LineWidth', 2);
xlabel('input size 2^x');
ylabel('optimality index');
title('optimality index for randomly generated preferences');
ylim([0,1.1]);
xlim([0,11]);
box on
grid on
saveas(handle,sprintf('%s/figure_14.pdf', dirname));
end
```