



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



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):

**Authored by** (in block letters):

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

**Name(s):**

**First name(s):**


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)**


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

# Contents

<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Individual contributions</b>	<b>1</b>
<b>3</b>	<b>Introduction and Motivations</b>	<b>1</b>
<b>4</b>	<b>Description of the Model</b>	<b>2</b>
4.1	Gale and Shapleys algorithm . . . . .	2
4.2	Modifications . . . . .	3
<b>5</b>	<b>Implementation</b>	<b>4</b>
5.1	Conventions . . . . .	4
5.2	Generating the Preference Lists . . . . .	4
5.2.1	Random . . . . .	4
5.2.2	The Plane Algorithm . . . . .	5
5.3	Making Matches . . . . .	5
5.4	Checking the Engagements . . . . .	8
5.5	simulation . . . . .	9
<b>6</b>	<b>Simulation Results and Discussion</b>	<b>9</b>
6.1	part valentin . . . . .	9
6.2	part samuel . . . . .	9
<b>7</b>	<b>Summary and Outlook</b>	<b>9</b>
<b>8</b>	<b>References</b>	<b>9</b>
<b>9</b>	<b>Appendix: MATLAB Codes</b>	<b>10</b>

# 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:

1. In the original model every node knows all the other nodes of the opposite gender. In a setting like a database of some match-making site this may be true. But as soon as the number of nodes gets big, it costs a huge amount of computation time to consider all nodes. In reality, information about the nodes of opposite is never complete (this would mean knowing about 3.5 billion people).
2. It is also conceivable that at some point a node might change its opinion about other nodes and rearrange them in his preference rating

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

What is stability? How to find its optimality? -Well, actually, whats an optimal stability?-. Of course, answering those questions would be the work of a philosophical essay which is not our pretention. We will only be considering some very abstract

and theoretical nodes which, for some practical and rhetorical reasons, will be called men or women. Although not in some philosophical sense stability and other related parameters (which were going to introduce soon enough) will be the center of our attention. We will see how they relate to our lovely nodes and interpret their correlations. The key point of this work is to generate random links or friendship between men and women, to classify those in a random order, to apply the algorithm of Gale and Shapley which match together in an optimal way men and women according to their preferences, to modify some parameters and then interpret the results. For a more precise description of the procedure, you may refer to chapter 4, Description of the Model.

The stable marriage problem is quite well known; it is originally an interesting mathematical problem, but it also has something catchy, everyone has its own idea of what is stability, how we think it is optimal, and how we would like to be matched, or how we think it should be done and there comes this emotionless algorithm who pretends -with the support of some mathematical proofs- to find all of this for us and to do it as least as good as we would. Of course, this is exaggerated but when we think about those internet sites who match people according to some theoretical criteria, isn't it what we're doing? To let some numbers find an objective answer to those questions, to our subjective questions? So...lets take one of the most fundamental of those matching algorithm and instead of letting it answer our questions, see what question we can extract from it.

## 4 Description of the Model

### 4.1 Gale and Shapleys algorithm

Lets first describe the algorithm of Gale and Shapley. Every man, resp. woman, classes the women, resp. men, according to his, resp. her, preferences. Then every man propose to his first choice, if a woman has more than one proposal she chooses her favorite and if she only has one proposal she accepts it. We do then the same with all the men which are still single; they propose to their second choices and the women accept the proposal if she likes him better than are actual fianc. This algorithm terminates when the single men have no women left on their list of preferences. Gale and Shapley proved that this algorithm always terminates with an optimal stability. According to their definitions an unstable assignment occurs when

there is a man or a woman likes someone else than his/her wife/husband better and this someone likes him/her better to than her/his husband/wife. If there isnt any unstable assignment, then the marriages are called stable. They also precise:

**Definition 1:** *A stable assignment is called optimal if every applicant is at least as well off under it as under any other assignment.*<sup>1</sup>

**Definition 2:** *An assignment of applicants to colleges will be called unstable if there are two applicants  $\alpha$  and  $\beta$  who are assigned to colleges  $A$  and  $B$ , respectively, although  $\beta$  prefers  $A$  to  $B$  and  $A$  prefers  $\beta$  to  $\alpha$ .*<sup>2</sup>

For more precise information you may refer to the quoted paper of Gale and Shapley.

## 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 since, for example, when a man goes in a bar, he doesnt 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 is some modification we can make to make it more realistic. One of these is that every men dont 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 doesnt know a woman who lives in Switzerland, so he wont propose to her; but we didnt 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 doesnt 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

---

<sup>1</sup>Gale and Shapley [1962]

<sup>2</sup>Gale and Shapley [1962]

time a single man propose to a woman) with a certain probability which is fixed.

We also introduced some new concepts, like the number of dumps, or an optimality index. The number of dumps is the number of time the women rejected their fianc for another man, this can be seen has an indication of time, the more people get dumped, the longer it is to reach a final assignment.

The optimality index gives us an information about how good is the method we are using with regards to the list of preferences of each man and woman. For example the optimality index decrease if a man is married to his fifth choice, rather than his second. It takes in account all the men and women, it contains therefore an information about the method. A more precise description of how we did these modifications will be presented in the next chapter.

## 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 and 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.

### 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).

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

Listing 1 : makeMatch code skeleton

```
1 function [ engaged, stable ] = makeMatch( m, f )
2 %makeMatch finds engagements for preferences according to Gale-Shapley ...
   algorithm
3 ...
4 freemen = [ (1:n)', ones(n,1) ];
5 engaged = zeros(n,2);
6 while ~isempty(find(freemen(:,2)==1,1))
7     theman = find(freemen(:,2)==1,1);
8     thegirl = m(theman,1);
9     index = find(engaged(:,2)==thegirl,1);
10    if(isempty(index) )
11        engaged(theman,1) = theman;
12        engaged(theman,2) = thegirl;
13        freemen(theman,2) = 0;
14    else
15        fiance = engaged(index,1);
16        girlprefers = f(thegirl,:);
17        if(find(girlprefers==theman,1)<find(girlprefers==fiance,1))
18            engaged(theman,1) = theman;
19            engaged(theman,2) = thegirl;
20            engaged(fiance,1) = 0;
21            engaged(fiance,2) = 0;
22            freemen(theman,2) = 0;
23            freemen(fiance,2) = 1;
24        else
25            m(theman,:) = [m(theman,2:n) 0];
26        end
27    end
28 end
29 stable = checkEngagements(engaged,m,f);
30 ...
31 end
```

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 already has a fiancé: In this case one has to do another distinction:
  - The girl prefers the new man to her fiancé: 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 fiancé, 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.

## 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])

Listing 2 : checkEngagements code fragment

```
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
```

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 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  $c \in [0, 1]$

The resulting output variables are:

- no. of instabilities
- no. of dumps
- no. of singles
- optimality index, as defined above

## 6 Simulation Results and Discussion

### 6.1 part valentin

### 6.2 part samuel

## 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](http://rosettacode.org/wiki/Stable_marriage_problem#Python).

## 9 Appendix: MATLAB Codes

### generateRandom.m

```
1 function [ m, f ] = generateRandom( n )
2 %GENERATERANDOM generates random preference matrices
3 m = zeros(n,n);
4 f = zeros(n,n);
5 for i=1:n
6     m(i,:) = randperm(n,n);
7     f(i,:) = randperm(n,n);
8
9 end
```

### generatePlane.m

```
1 function [ mpref,fpref ] = generatePlane( n ,mode, radius)
2 %GENERATEPLANE generates preference lists for men and women
3 %   based on a plane where women and men are represented by points
4 %   they have a limited visibility radius
5 %   n: number of men and women
6 %   mode: visibility radius mode, optional argument
7 %       1 —> const, one constant radius for all nodes
8 %       2 —> random, a new random radius is generated in each iteration
9 %           value is between 0.1 and 0.5
10 %   default mode is const
11 %   mpref: mens preferences in nxn matrix
12 %   fpref: womens preferences in nxn matrix
13
14 global verbosity
15
16 if (nargin >= 2 && mode == 1)
17     assert(nargin==3);
18     r = radius;
```

```

19 end
20 if(nargin < 2)
21     mode = 1;
22     r = 0.2;%default value
23 end
24
25 % generate random coordinates
26 % and extend to torus
27 men = zeros(3,9*n);
28 rnd = rand(2,n);
29 men(:, (0*n)+1:1*n)=[ (1:n);rnd];
30 men(:, (1*n)+1:2*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
31 men(:, (2*n)+1:3*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
32 men(:, (3*n)+1:4*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
33 men(:, (4*n)+1:5*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
34 men(:, (5*n)+1:6*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
35 men(:, (6*n)+1:7*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
36 men(:, (7*n)+1:8*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
37 men(:, (8*n)+1:9*n)=men(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];
38
39 women = zeros(3,9*n);
40 rnd = rand(2,n);
41 women(:, (0*n)+1:1*n)=[ (1:n);rnd];
42 women(:, (1*n)+1:2*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
43 women(:, (2*n)+1:3*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
44 women(:, (3*n)+1:4*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
45 women(:, (4*n)+1:5*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
46 women(:, (5*n)+1:6*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
47 women(:, (6*n)+1:7*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
48 women(:, (7*n)+1:8*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
49 women(:, (8*n)+1:9*n)=women(:, (0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];
50
51 %plotting
52 % if verbosity~=0
53 %     plot(men(2,1:n),men(3,1:n),'o',women(2,1:n),women(3,1:n),'o');
54 %     label1 = cellstr( num2str(women(1,1:n)) );
55 %     label2 = cellstr( num2str(men(1,1:n)) );
56 %     text(women(2,1:n),women(3,1:n),label1);
57 %     text(men(2,1:n),men(3,1:n),label2);
58 %     title('nodes in plane');
59 %     legend('men','women');
60 % end
61
62 d = zeros(2,9*n);
63 mpref = zeros(n,n);
64 fpref = zeros(n,n);

```

```

65
66
67 for i=1:n
68     man = men(:,i);
69     for j=1:9*n
70         woman = women(:,j);
71         d(:,j) = [woman(1,1);norm(man(2:3)-woman(2:3),2)];
72     end
73     if mode==2
74         r = rand*0.4+0.1;
75     end
76     index = find(d(2,:)<r);
77     available = women(:,index);
78     sz = size(available,2);
79     if sz>n
80         available = available(:,1:n);
81         sz = n;
82     end
83     perm = randperm(sz);
84     mpref(i,1:sz) = available(1,perm);
85 end
86 for i=1:n
87     woman = women(:,i);
88     for j=1:9*n
89         man = men(:,j);
90         d(:,j) = [man(1,1);norm(man(2:3)-woman(2:3),2)];
91     end
92     if mode==2
93         r = rand*0.4+0.1;
94     end
95     index = find(d(2,:)<r);
96     available = men(:,index);
97     sz = size(available,2);
98     if sz>n
99         available = available(:,1:n);
100        sz = n;
101    end
102    perm = randperm(sz);
103    fpref(i,1:sz) = available(1,perm);
104 end
105 end

```

**vprintf.m**



```

1 function vprintf(varargin)
2 % VPRINTF controlled printing
3 %
4 global verbosity
5 if verbosity~=0
6     fprintf(varargin{:});
7 end

```

## makeMatch.m

```

1 function [ engaged, output ] = makeMatch( m, f, changerate, p )
2 %makeMatch finds engagements for preferences according to Gale-Shapley ...
   algorithm
3 %
4 %   men an women encoded as integers from 1 to n
5 %
6 %   input:
7 %   m: preference matrix of the men. Each row corresponds to a man and
8 %   the elements are the women listed according to his preferences.
9 %   f: preference matrix of the women. Each row corresponds to a woman and
10 %   the elements are the men listed according to her preferences.
11 %   changerate: rate at which preference changes are performed, e. g. if
12 %   changerate=0.2 then only in 20% of iterations preferences are changed
13 %   p: change preferences for men (p=1) / women (p=0.5) / both (p=0.5)
14 %
15 %   dimensions must be correct, m=nxn, f=nxn.
16 %
17 %   returns:
18 %   engaged: nx2 Matrix containing matches
19 %   output: output data —>
20 %   output(1,1): number of instabilities
21 %   output(1,2): number of singles
22 %   output(1,3): number of dumps
23 %   output(1,4): optimality index
24
25 % optional test prints
26 global verbosity
27 vprintf('mens preferences:\n');
28 if verbosity~=0 disp(m); end
29 vprintf('womens preferences:\n')
30 if verbosity~=0 disp(f); end
31 % assign local variables
32 initialm = m;
33 initiaLf = f;

```

```

34 n = size(m,1);
35 n2 = size(f,1);
36 % make sure dimensions agree
37 assert(n == size(m,2));
38 assert(n==n2);
39 if nargin > 2
40     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 ...
           fiancé 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     howgirllikesthemans=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(howgirllikesthemans) % 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 verbatim 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 marriage\n', counter);
162     else
163         vprintf('there are %d unstable mariages\n', counter);
164     end % if_2
165 end % if
166 % calculate optimality index
167 opt = 0;
168 for i = 1:n
169     he = i;
170     she = engaged(he,2);
171     if she~=0
172         hisindex = find(initialf(she,:)==he,1);
173         herindex = find(initialm(he,:)==she,1);
174     else
175         hisindex = n;
176         herindex = n;
177     end
178     opt = opt + hisindex + herindex;
179 end
180 opt = opt/(2*n*n);
181 vprintf('optimality index is %1.2f\n',opt);
182 % set output
183 output = zeros(1,4);
184 output(1,1) = counter;
185 output(1,2) = single;
186 output(1,3) = dumped;
187 output(1,4) = opt;
188 end

```

## checkEngagements.m

```
1 function [ stable,counter ] = checkEngagements( engaged, m, f )
2 %checkEngagements checks whether a set of engagements is stable
3 %
4 %   men an women encoded as integers from 1 to n
5 %
6 %   input:
7 %   engaged: engagement matrix
8 %   m,f: preference matrices
9 %
10 %   dimensions must be correct, m=nxn, f=nxn, engaged=nx2
11 %
12 %   returns:
13 %   stable: true for stable engagements, false otherwise
14 %   counter: the number of unstable mariages
15
16 n = size(m,1); % input size
17 % reverse the engaged matrix such that the new matrix has the index of the
18 % women on the column one and those of their respective husbands in row two
19 invengaged=zeros(n,2);
20 copy = engaged(:,[2,1]);
21 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 ...
43         -> check the next man
44         he = he+1;
45         she = engaged(he,2);
46     end % while
47     if she==0 % -> he=n is not engaged, nothing to check.
48         break;
49     end %if
50     % get indexes in pref lists
51     hisindex = find(f(she,:)==he,1);
52     herindex = find(m(he,:)==she,1);
53     helikesbetter = m(he,1:herindex);
54     shelikesbetter = f(she,1:hisindex);
55     % check for her
56     if ~isempty(shelikesbetter) % there is no one on earth she likes better
57         for i=1:size(shelikesbetter) % loop to check if there is ...
58             unstability for the girl
59             guy = shelikesbetter(i); % all the guys she likes better
60             guysgirl = engaged(guy,2); % the guy she is engaged to
61             if guysgirl == 0 && ~isempty(find(m(guy,:)== she,1)) % if this ...
62                 guy isn't engaged, then she could be with him -> unstable, ...
63                 unless he doesn't know her.
64                 stable = false;
65                 vprintf('man %d and woman %d like each other better\n', guy, ...
66                     she);
67                 inst = [guy,she;inst];
68             else
69                 guylikes = m(guy,:); % the ordered preferences of guy
70                 if (find(guylikes==she,1)<find(guylikes==guysgirl,1)) % if ...
71                     guy also likes she better than his wife -> unstable
72                     stable = false;
73                     vprintf('man %d and woman %d like each other better\n', ...
74                         guy, she);
75                     inst = [guy,she;inst];
76                 end % if_3
77             end % if_2
78         end % for
79     end % if_1
80     % now the other way round, check for him
81     if ~isempty(helikesbetter) % there is no one on earth he likes better
82         for i=1:size(helikesbetter) % loop to check if there is unstability ...
83             for the man
84             girl = helikesbetter(i); % all the girls he likes better
85             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 %
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, in 0.1:0.05:0.5

```



```

17 % frequency
18
19 global verbosity
20 verbosity = 0;
21
22 assert(~isempty(find(saveit==[0,1],1)));
23 tmax = 6;
24 t = 2.^(1:tmax);
25 r = 0.1:0.05:0.5;
26 data = zeros(tmax,10,4);
27 seed = rng;
28 if saveit==1
29     dirname = sprintf('data/%s',datestr(now,'yyyy-mm-dd_HH-MM-SS'));
30     mkdir(dirname);
31 end
32
33 % radius random
34 for i=1:tmax
35     n = t(i);
36     [a,b] = generatePlane(n,2);
37     [x,y] = makeMatch(a,b);
38     data(i,10,:) = y;
39 end
40
41 % radius const
42 for i=1:tmax
43     for j=1:9
44         n = t(i);
45         radius = r(j);
46         [a,b] = generatePlane(n,1,radius);
47         [x,y] = makeMatch(a,b);
48         data(i,j,:) = y;
49     end
50 end
51 % plot optimality index for each radius
52 hold on
53 handle = figure(1);
54 col = hsv(10);
55 %set(groot,'defaultAxesLineStyleOrder',{'-*',':', 'o'});
56 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

```