Dataset

Supplementary material for the paper "Committees, sequential voting and transparency"

Author(s):
Hahn, Volker

Publication Date:
2008

Permanent Link:

https://doi.org/10.3929/ethz-a-005570816 →

This file contains supplementary material
for the paper "Committees, Sequential Voting and Transparency"
by Volker Hahn (2008),
forthcoming in "Mathematical Social Sciences".

Contents:

* C source code for the program
  that was used to generate data for Figure 1

* output of this program (data used for Figure 1)

* C source code of the program
  used for Numerical Findings 1 and 2

```
/*      $Id: Figure1.c,v 1.8 2008-02-22 13:55:28 vhahn Exp $     */

/*
  Author:    Volker Hahn
  Type:      C source code

  Objective: This creates the data for Figure 1 in the paper
  "Committees, Sequential Voting and Transparency", 2008. Forthcoming in
  "Mathematical Social Sciences".

  Compiler: compiled by gcc, version 4.1.2, available from gcc.gnu.org,
            command "gcc -lm Figure1.c -o Figure1"
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* declare functions */
long double OptimalBehavior3T();
long double computeUtilPrincSeq();
long double computeUtilPrincSim();

/* global variables */
long double pH=0.8, pL=0.51;

int main()
{
   pH=.8;

   /* compute the principal's utility in the first period under sequential
      voting and simultaneous voting for all admissibile values of pL */
   for (pL=.5;pL<=pH+0.00001;pL+=0.005)
      printf("%Lf %Lf %Lf\n",
             pL, computeUtilPrincSeq(),computeUtilPrincSim());

   return(0);
}

/*
  Function: computeUtilPrincSeq

  Objective: returns the probability of a correct decision being
  reached in the first period, i.e.
  the utility of the principal in the first period,
  under sequential voting
*/

long double computeUtilPrincSeq()
{
/* assume w.l.o.g d^* = +1 */

/* v[0] vote of member 1,
   v[1] vote of member 2,
   v[2] vote of member 3 */
   char v[3];

   /* probability of member 2 following his signal s=+1
      if less efficient and v=(-1) */
   long double followSignalIfUnsure2;

   /* probability of member 3 following his signal s=+1
      if less efficient and v=(-1,+1) */
   long double followSignalIfUnsure3;

   /* the result is stored in this variable */
   long double result=0;

   long double Prob=0;

   /* parameters describing the optimal behavior
      of less effficient members 2 and 3, compare Proposition 2
      of the paper */
   followSignalIfUnsure2 = (1.-pH)/(1.-pL);
   followSignalIfUnsure3 = OptimalBehavior3T();

   /* loop over all possible patterns of votes */
   for (v[0]=-1;v[0]<=+1;v[0]+=2)
      for (v[1]=-1;v[1]<=+1;v[1]+=2)
         for (v[2]=-1;v[2]<=+1;v[2]+=2)
         {
            /* Prob will give the probability of a particular
               pattern of votes (v_1,v_2,v_3)
               It will be computed step by step:

               the probablity of (v_1,v_2,v_3)
               =
               the probability of v_1
               times
               the probability of v_2, given v_1,
               times
               the probability of v_3, given v_1 and v_2
            */

            /* First step: Prob = probability of 1 voting for v[0] */
            if (v[0]==-1)
               Prob=1-.5*(pH+pL);
            else
               Prob=.5*(pH+pL);

            /* Second step: Prob is multiplied by the
               probability of 2 voting for v[1],
               given that 1 has chosen v[0] */
            if (v[0]==-1 && v[1]==-1)
               Prob*=(.5*(1-pH)+.5*(1-pL)+.5*pL*(1-followSignalIfUnsure2));
            else if (v[0]==-1 && v[1]==+1)
               Prob*=(.5*pH + .5*pL*followSignalIfUnsure2);
            else if (v[0]==+1 && v[1]==-1)
               Prob*=(.5*(1-pH) + .5*(1-pL)*followSignalIfUnsure2);
            else if (v[0]==+1 && v[1]==+1)
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure2));


            /* Third step: Prob is multiplied by the
               probability of 3 voting for v[2],
               given that 1 has chosen v[0] and 2 has chosen v[1] */
            if (v[0]==-1 && v[1]==-1 && v[2]==-1)
               Prob*=(.5*(1-pH)+.5*(1-pL)+.5*pL*(1-followSignalIfUnsure2));
            else if (v[0]==-1 && v[1]==-1 && v[2]==+1)
               Prob*=(.5*pH+.5*pL*followSignalIfUnsure2);
            else if (v[0]==-1 && v[1]==+1 && v[2]==-1)
               Prob*=(.5*(1-pH)+.5*(1-pL)*followSignalIfUnsure3);
            else if (v[0]==-1 && v[1]==+1 && v[2]==+1)
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure3));
```

```
            else if (v[0]==+1 && v[1]==-1 && v[2]==-1)
               Prob*=(.5*(1-pH)+.5*pL*(1-followSignalIfUnsure3)+.5*(1-pL));
            else if (v[0]==+1 && v[1]==-1 && v[2]==+1)
               Prob*=(.5*pH+.5*pL*followSignalIfUnsure3);
            else if (v[0]==+1 && v[1]==+1 && v[2]==-1)
               Prob*=(.5*(1-pH)+.5*(1-pL)*followSignalIfUnsure2);
            else if (v[0]==+1 && v[1]==+1 && v[2]==+1)
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure2));

            /* Check if there is a majority of votes in favor of the correct
               decision. If this is the case, increase "result" by Prob */
            if (v[0]+v[1]+v[2]>0)
               result+=Prob;
         }

   return(result);
}

/*
  Function: computeUtilPrincSim

  Objective: returns the probability of a correct decision being
  reached in the first period, i.e. the utility of the principal
  in the first period, under simultaneous voting
*/

long double computeUtilPrincSim()
{
   /* the result is stored in this variable */
   long double result=0;

   long double Prob=1;

   int i=0;

   /* v: vector of the members' votes
      for example v[0] is member 1's vote */
   char v[3];

   /* Note: the analysis is comparably simple, because
      the probabilities of individual members voting
      correctly are independent under simultaneous voting */

   for (v[0]=-1;v[0]<=+1;v[0]+=2)
      for (v[1]=-1;v[1]<=+1;v[1]+=2)
         for (v[2]=-1;v[2]<=+1;v[2]+=2)
         {
            Prob=1;

            for(i=0;i<=2;i++)
               if (v[i]==-1)
                  Prob*=1-.5*(pH+pL); /* 1-.5*(pH+pL) is the probability
                                         of a member choosing a wrong vote */
               else
                  Prob*=.5*(pH+pL); /* .5*(pH+pL) is the probability
                                       of a member choosing a correct vote */

            /* Check if there is a majority of votes in favor of the correct
               decision. If this is the case, increase "result" by Prob */
            if (v[0]+v[1]+v[2]>0)
               result+=Prob;
         }
```



```
   return(result);
}

/*
  ------------------------------------------------
  Function: OptimalBehavior3T

  Objective: compute the probability of member 3 choosing v=+1,
  given s=+1, v_1=-1, and v_2=+1 (compare Proposition 2 in the paper)
  ------------------------------------------------
*/

long double OptimalBehavior3T()
{
   long double followSignalIfUnsure3;
   long double zmp=0;

   /* compute z_(-1,+1) and compare to pL */

   zmp = (1./(1.+ (1.-pL)*(1.-pH)*(.5*(pH+pL))/(1.- .5*(pH+pL))
               / (.5*(pH*(1.-pL)+pL*(1.-pH)))));

   if (pL>zmp)
      /* private signal superior to information from
         pattern of previous votes */
      followSignalIfUnsure3 = 1.;
   else
      followSignalIfUnsure3 =(1-pH)/(1-pL);

   return(followSignalIfUnsure3);
}
```

```
0.500000 0.725000 0.718250
0.505000 0.727075 0.721657
0.510000 0.729161 0.725052
0.515000 0.731258 0.728436
0.520000 0.733367 0.731808
0.525000 0.735487 0.735168
0.530000 0.737619 0.738516
0.535000 0.739764 0.741851
0.540000 0.741922 0.745174
0.545000 0.744093 0.748484
0.550000 0.746278 0.751781
0.555000 0.748477 0.755065
0.560000 0.751215 0.758336
0.565000 0.753984 0.761593
0.570000 0.756758 0.764837
0.575000 0.759536 0.768066
0.580000 0.762319 0.771282
0.585000 0.765106 0.774483
0.590000 0.767899 0.777670
0.595000 0.770697 0.780843
0.600000 0.773500 0.784000
0.605000 0.776309 0.787142
0.610000 0.779124 0.790270
0.615000 0.781946 0.793382
0.620000 0.784774 0.796478
0.625000 0.787609 0.799559
0.630000 0.790452 0.802623
0.635000 0.793302 0.805672
0.640000 0.796160 0.808704
0.645000 0.799026 0.811720
0.650000 0.801902 0.814719
0.655000 0.804786 0.817701
0.660000 0.807681 0.820666
0.665000 0.810585 0.823614
0.670000 0.813500 0.826544
0.675000 0.816427 0.829457
0.680000 0.819365 0.832352
0.685000 0.822316 0.835229
0.690000 0.825280 0.838088
0.695000 0.828258 0.840928
0.700000 0.831250 0.843750
0.705000 0.834258 0.846553
0.710000 0.837282 0.849337
0.715000 0.840323 0.852102
0.720000 0.843383 0.854848
0.725000 0.846462 0.857574
0.730000 0.849561 0.860281
0.735000 0.852682 0.862967
0.740000 0.855825 0.865634
0.745000 0.858993 0.868280
0.750000 0.862188 0.870906
0.755000 0.865409 0.873512
0.760000 0.868660 0.876096
0.765000 0.871942 0.878659
0.770000 0.875258 0.881202
0.775000 0.878609 0.883723
0.780000 0.881999 0.886222
0.785000 0.885430 0.888700
0.790000 0.888905 0.891155
0.795000 0.892427 0.893589
0.800000 0.896000 0.896000
```

```
/*      $Id: NumericalFinding_1+2.c,v 1.32 2008-02-22 13:55:46 vhahn Exp $
 */

/*
  Author:    Volker Hahn
  Type:      C source code

  Objective: This program verifies Numerical Findings 1 and 2 in the paper
  "Committees, Sequential Voting and Transparency", 2008. Forthcoming in
  "Mathematical Social Sciences".

  Compiler: compiled by gcc, version 4.1.2, available from gcc.gnu.org,
            command "gcc -lm NumericalFinding_1+2.c -o NumericalFinding_1+2"
*/


/* The program prints information about the constellations for which
   first-order stochastic dominance (FOSD) does not hold

   If it prints nothing, FOSD always holds.

   Numerical Findings 1 and 2 are checked simultaneously.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define COMMITTEE_SIZE 3
#define MAXCOEFF 4
/* take care of roundoff errors */
#define EPSILON 0.0000000001

/* declare functions */
long double OptimalBehavior3T();
void computeDistrSim(long double *frequency);
void computeDistrSeq(long double *frequency);
void computeDistrOpac(long double *frequency);

char optOpac(int n, int nHPlus, int nLPlus);

int bincoeff(int n, int k);
void bincoeffinit();
void checkFOSD();
void makeCDF(long double *frequency);

/* global variables */
int BinomialCoefficient[MAXCOEFF][MAXCOEFF];

long double pH=0.8, pL=0.6;

int main()
{
   /* initialize binomial coefficients */
   bincoeffinit();

   /* check whether first-order stochastic dominance holds */
   checkFOSD();

   return(0);
}
```



```
/*
  ---------------------------------------------
  Function: checkFOSD

  check whether first-order stochastic dominance holds in the scenarios
  described in Numerical Findings 1 and 2

  If it is violated, checkFOSD prints information, for example,
  about pH, pL, and the number of efficient members
  ---------------------------------------------
*/

void checkFOSD()
{
   int n;
   long double frequencySeq[COMMITTEE_SIZE+1];
   long double frequencySim[COMMITTEE_SIZE+1];
   long double frequencyOpac[COMMITTEE_SIZE+1];

   /* check FOSD for all admissible values of pL and pH */

   for (pH=0.5;pH<=1.00001;pH+=0.001)
      for (pL=.5;pL<=pH+0.00001;pL+=0.001)
      {

         /* compute the probability distribution function of highly efficient
            members in the second period for three scenarios ... */

         /* ... opaque voting in the first period */
         computeDistrOpac(frequencyOpac);
         /* ... transparent sequential voting in the first period */
         computeDistrSeq(frequencySeq);
         /* ... transparent simultaneous voting in the first period */
         computeDistrSim(frequencySim);

         /* compute the cumulative probability distributions, i.e. integrate
          the probability distribution functions */
         makeCDF(frequencyOpac);
         makeCDF(frequencySeq);
         makeCDF(frequencySim);

         /* check FOSD */

         /* n: number of highly efficient members */
         for (n=0;n<COMMITTEE_SIZE;n++)
         {
            /* check Numerical Finding 1 */
            if (frequencyOpac[n]<frequencySeq[n]-EPSILON)
               /* print information if Num. Find. 1 is violated */
               printf("Opac - Seq pH=%Lf pL=%Lf n=%d \n", pH, pL, n);
            /* check Numerical Finding 2 */
            if (frequencySeq[n]<frequencySim[n]-EPSILON)
               /* print information if Num. Find. 2 is violated */
               printf("Seq - Sim pH=%Lf pL=%Lf n=%d \n", pH, pL, n);
         }
      }
}

/*
  ---------------------------------------------
  Function: computeDistrSim
```

```
  Objective: compute the probability of n members
  being highly efficient in the second period under
  simultaneous voting in the first period

  the results are in the vector frequency

  for example, frequency[0] is the probability of
  no member being highly efficient
  ------------------------------------------------
*/

void computeDistrSim(long double *frequency)
{
   int n=0;

   long double compIndi;

   /* compIndi: average competence of an individual member after re-appointment

      .5*(pH+pL) .... probability of a member choosing the correct vote
                      in the first period and thus being re-appointed
      pH/(pH+pL) .... probability of a member who has chosen the correct
                      vote being highly efficient
      (1-.5*(pH+pL))  probability of a member choosing the wrong vote
                      in the first period and thus being dismissed
      .5 ............ probability of a newly appointed member being
                      highly efficient
   */

   compIndi = .5*(pH+pL) * pH/(pH+pL) + (1-.5*(pH+pL)) * .5;

   /* Note that the efficiency levels of individual members
      are independent for simultaneous voting */
   for (n=0;n<=COMMITTEE_SIZE;n++)
      frequency[n]=bincoeff(3,n)*pow(compIndi,n)*pow(1-compIndi,COMMITTEE_SIZE-n
);
}



/*
  ------------------------------------------------
  Function: computeDistrSeq

  Objective: compute the probability of n members
  being highly efficient in the second period under
  sequential voting in the first period

  the results are in the vector frequency

  for example, frequency[0] is the probability
  of no member being highly efficient
  ------------------------------------------------
*/

void computeDistrSeq(long double *frequency)
{
   /* assume w.l.o.g. d^* = +1 */

   /* v[] : votes of members 1,2,3
      v[0] is the vote of member 1 (v_1)
      v[1] is the vote of member 2 (v_2)
```



```
      v[2] is the vote of member 3 (v_3)
   */

   char v[3];
   int i,n;

   long double Prob;

   /* probability of member 2 following his signal s=+1
      if less efficient and v=(-1) */
   long double followSignalIfUnsure2;

   /* probability of member 3 following his signal s=+1
      if less efficient and v=(-1,+1) */
   long double followSignalIfUnsure3;

   /* kappa[] competencies of members 1,2,3
      kappa[0] is the probability of member 1 being highly efficient
      kappa[1] is the probability of member 2 being highly efficient
      kappa[2] is the probability of member 3 being highly efficient */
   long double kappa[3];

   /* parameters describing the optimal behavior
      of less efficient members 2 and 3, compare Proposition 2
      of the paper */
   followSignalIfUnsure2 = (1.-pH)/(1.-pL);
   followSignalIfUnsure3 = OptimalBehavior3T();

   /* erase old values */
   for (n=0;n<=COMMITTEE_SIZE;n++)
      frequency[n]=0;

   /* loop over all possible patterns of votes */
   for (v[0]=-1;v[0]<=+1;v[0]+=2)
      for (v[1]=-1;v[1]<=+1;v[1]+=2)
         for (v[2]=-1;v[2]<=+1;v[2]+=2)
         {
            /* Prob will give the probability of a particular
               pattern of votes (v_1,v_2,v_3)
               It will be computed step by step:

               the probability of (v_1,v_2,v_3)
               =
               the probability of v_1
               times
               the probability of v_2, given v_1,
               times
               the probability of v_3, given v_1 and v_2
            */

            /* First step: Prob = probability of 1 voting for v[0];
             compute kappa[0] */
            if (v[0]==-1)
            {
               Prob=1-.5*(pH+pL);
               kappa[0] = (1-pH)/(1-pH+1-pL); /* Note: 1 will be dismissed,
                                                 because kappa[0]<.5 */
            }
            else
            {
               Prob=.5*(pH+pL);
               kappa[0]=pH/(pH+pL); /* Note: 1 will be re-appointed,
```

```
                                because kappa[0]>.5 */
            }

            /* Second step: Prob is multiplied by the
               probability of 2 voting for v[1],
               given that 1 has chosen v[0];
               compute kappa[1]
            */
            if (v[0]==-1 && v[1]==-1)
            {
               Prob*=(.5*(1-pH)+.5*(1-pL)+.5*pL*(1-followSignalIfUnsure2));
               kappa[1]= (1-pH)/((1-pH)+(1-pL)+pL*(1-followSignalIfUnsure2));
            }
            else if (v[0]==-1 && v[1]==+1)
            {
               Prob*=(.5*pH + .5*pL*followSignalIfUnsure2);
               kappa[1]=pH/(pH+pL*followSignalIfUnsure2);
            }
            else if (v[0]==+1 && v[1]==-1)
            {
               Prob*=(.5*(1-pH) + .5*(1-pL)*followSignalIfUnsure2);
               kappa[1]=(1-pH)/((1-pH)+ (1-pL)*followSignalIfUnsure2);
            }
            else if (v[0]==+1 && v[1]==+1)
            {
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure2));
               kappa[1]=pH/(pH+pL+(1-pL)*(1-followSignalIfUnsure2));
            }


            /* Third step: Prob is multiplied by the
               probability of 3 voting for v[2],
               given that 1 has chosen v[0] and 2 has chosen v[1];
               compute kappa[2]
            */
            if (v[0]==-1 && v[1]==-1 && v[2]==-1)
            {
               Prob*=(.5*(1-pH)+.5*(1-pL)+.5*pL*(1-followSignalIfUnsure2));
               kappa[2]=(1-pH)/((1-pH)+(1-pL)+pL*(1-followSignalIfUnsure2));
            }
            else if (v[0]==-1 && v[1]==-1 && v[2]==+1)
            {
               Prob*=(.5*pH+.5*pL*followSignalIfUnsure2);
               kappa[2]=pH/(pH+pL*followSignalIfUnsure2);
            }
            else if (v[0]==-1 && v[1]==+1 && v[2]==-1)
            {
               Prob*=(.5*(1-pH)+.5*(1-pL)*followSignalIfUnsure3);
               kappa[2]=(1-pH)/((1-pH)+(1-pL)*followSignalIfUnsure3);
            }
            else if (v[0]==-1 && v[1]==+1 && v[2]==+1)
            {
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure3));
               kappa[2]=pH/(pH+pL+(1-pL)*(1-followSignalIfUnsure3));
            }
            else if (v[0]==+1 && v[1]==-1 && v[2]==-1)
            {
               Prob*=(.5*(1-pH)+.5*pL*(1-followSignalIfUnsure3)+.5*(1-pL));
               kappa[2]=(1-pH)/((1-pH)+pL*(1-followSignalIfUnsure3)+(1-pL));
            }
            else if (v[0]==+1 && v[1]==-1 && v[2]==+1)
            {
```



```
               Prob*=(.5*pH+.5*pL*followSignalIfUnsure3);
               kappa[2]=pH/(pH+pL*followSignalIfUnsure3);
            }
            else if (v[0]==+1 && v[1]==+1 && v[2]==-1)
            {
               Prob*=(.5*(1-pH)+.5*(1-pL)*followSignalIfUnsure2);
               kappa[2]=(1-pH)/((1-pH)+(1-pL)*followSignalIfUnsure2);
            }
            else if (v[0]==+1 && v[1]==+1 && v[2]==+1)
            {
               Prob*=(.5*pH+.5*pL+.5*(1-pL)*(1-followSignalIfUnsure2));
               kappa[2]=pH/(pH+pL+(1-pL)*(1-followSignalIfUnsure2));
            }

            /* re-appoint all members whose level of efficiency exceeds 0.5 */
            for (i=0;i<=2;i++)
               if (kappa[i]<0.5)
                  kappa[i]=0.5;

            /* There is only one constellation where no member chooses
               the correct vote

               (1-kappa[0])*(1-kappa[1])*(1-kappa[2]) gives the probability
               of no member being highly efficient, given that the pattern of
               votes is v */
            frequency[0]+=Prob*(1-kappa[0])*(1-kappa[1])*(1-kappa[2]);

            /* There are three constellations where one member chooses the
               correct vote */
            frequency[1]+=Prob*kappa[0]*(1-kappa[1])*(1-kappa[2]);
            frequency[1]+=Prob*(1-kappa[0])*kappa[1]*(1-kappa[2]);
            frequency[1]+=Prob*(1-kappa[0])*(1-kappa[1])*kappa[2];

            /* There are three constellations where two member choose the
               correct vote */
            frequency[2]+=Prob*(1-kappa[0])*kappa[1]*kappa[2];
            frequency[2]+=Prob*kappa[0]*(1-kappa[1])*kappa[2];
            frequency[2]+=Prob*kappa[0]*kappa[1]*(1-kappa[2]);

            /* There is only one constellation where all members choose
               the correct vote */
            frequency[3]+=Prob*kappa[0]*kappa[1]*kappa[2];
         }
}

/*
  ------------------------------------------
  Function: computeDistrOpac

  Objective: compute the probability of n members being
  highly efficient in the second period under opaque
  voting in the first period

  the results are in the vector frequency

  for example, frequency[0] is the probability
  of no member being highly efficient
  ------------------------------------------
*/

void computeDistrOpac(long double *frequency)
{
```

```
   /* s: signals of members 1,2,3 */
   char s[3];

   int i=0;

   /* comp: actual efficiency levels of members 1,2,3 */
   /* H = 1; L= 0 */
   char comp[3];

   /* assume w.l.o.g d^* = 1 */

   /* n: number of efficient members,
      nHPlus: number of efficient members with signal +1,
      nLPlus: number of less efficient members with signal +1 */

   int n=0,nHPlus=0,nLPlus=0;

   long double Prob=0;

   /* erase old results */
   for(i=0;i<=3;i++)
      frequency[i]=0;

   /* loop over all possible vectors of competencies and signals */
   for (comp[0]=0;comp[0]<=1;comp[0]++)
      for (comp[1]=0;comp[1]<=1;comp[1]++)
         for (comp[2]=0;comp[2]<=1;comp[2]++)
            for (s[0]=-1;s[0]<=1;s[0]+=2)
               for (s[1]=-1;s[1]<=1;s[1]+=2)
                  for (s[2]=-1;s[2]<=1;s[2]+=2)
                  {
                     /* Prob gives the probability of a particular
                        combination of signals
                        s and efficiency levels comp */

                     Prob=1;

                     for (i=0;i<=3;i++)
                     {
                        if ((comp[i]==1) && (s[i]==1))
                           Prob*=.5*pH;
                        else if ((comp[i]==1) && (s[i]==-1))
                           Prob*=.5*(1-pH);
                        else if ((comp[i]==0) && (s[i]==1))
                           Prob*=.5*pL;
                        else if ((comp[i]==0) && (s[i]==-1))
                           Prob*=.5*(1-pL);
                     }

                     n=0;

                     /* compute the number of highly efficient
                        members, i.e. n */
                     for (i=0;i<3;i++)
                        n+=comp[i];

                     nHPlus=0;

                     /* compute the number of highly efficient members
                        with a correct signal s=+1 */
                     for (i=0;i<3;i++)
                        if ((comp[i]==1) && s[i]==1)
```



```
                           nHPlus++;

                     nLPlus=0;

                     /* compute the number of less efficient members
                        with a correct signal s=+1 */
                     for (i=0;i<3;i++)
                        if ((comp[i]==0) && s[i]==1)
                           nLPlus++;

                     /* We check whether the committee chooses a correct decisio
n,
                        given n, nHPlus,nLPlus */
                     if (optOpac(n,nHPlus,nLPlus)==1) /* correct outcome */
                        frequency[n]+=Prob;
                     else if (optOpac(n,nHPlus,nLPlus)==-1) /* wrong outcome,
                                                               dismissal
                                                               of all members */
                        for (i=0;i<=3;i++)
                           frequency[i]+=bincoeff(3,i)*pow(.5,3)*Prob;
                     else /* members are indifferent and randomize
                             between both possibilities */
                     {
                        /* As a consequence, with probability .5, the committee
                           chooses the wrong decision and is dismissed ... */
                        frequency[n]+=.5*Prob;
                        /* ... and with probability .5, the committee chooses
                           the correct decision and is re-appointed ... */
                        for (i=0;i<=3;i++)
                           frequency[i]+=bincoeff(3,i)*pow(.5,3)*.5*Prob;
                     }
                  }
   }


   /*
     ------------------------------------------------
     Function: OptimalBehavior3T

     Objective: compute the probability of member 3 choosing v=+1,
     given s=+1, v_1=-1, and v_2=+1 (compare Proposition 2 in the paper)
     ------------------------------------------------
   */

   long double OptimalBehavior3T()
   {
      long double followSignalIfUnsure3;
      long double zmp=0;

      /* compute z_(-1,+1) and compare to pL; */

      /* zmp corresponds to z_(-1,+1); for the derivation see Equation (20) in the
paper
       and note that z_(-1) = 1-.5*(pH+pL)  */
      zmp = (1./(1.+ (1.-pL)*(1.-pH)*(.5*(pH+pL))/(1.- .5*(pH+pL))
                  / (.5*(pH*(1.-pL)+pL*(1.-pH)))));

      if (pL>zmp)
         /* pL> z_(-1,+1):
            private signal superior to information from
            pattern of previous votes */
         followSignalIfUnsure3 = 1.;
```

```
   else
      followSignalIfUnsure3 =(1-pH)/(1-pL);

   return(followSignalIfUnsure3);
}

/*
  ------------------------------------------------
  Function: optOpac

  Objective: returns the optimal decision of the committee under
  opacity, given

  n, the number of highly efficient members,
  nHPlus, the number of highly efficient members with a correct signal (s=+1),
  nLPlus, the number of less efficient members with a correct signal (s=+1)
  ------------------------------------------------
*/

char optOpac(int n, int nHPlus, int nLPlus)
{
   char result=0;

   /* probability of observing nHplus and nLplus, given n and d^* = +1 */
   long double PrPlus=0;

   /* probability of observing nHplus and nLplus, given n and d^* = -1 */
   long double PrMinus=0;

   PrPlus=bincoeff(n,nHPlus)*pow(pH,nHPlus)*pow(1-pH,n-nHPlus);
   PrPlus*=bincoeff(COMMITTEE_SIZE-n,nLPlus)*pow(pL,nLPlus)*pow(1-pL,COMMITTEE_S
IZE-n-nLPlus);

   PrMinus=bincoeff(n,nHPlus)*pow(1-pH,nHPlus)*pow(pH,n-nHPlus);
   PrMinus*=bincoeff(COMMITTEE_SIZE-n,nLPlus)*pow(1-pL,nLPlus)*pow(pL,COMMITTEE_
SIZE-n-nLPlus);

   if (PrPlus>PrMinus+EPSILON) /* +1 is more likely to be correct */
      result=+1;
   else if (PrPlus<PrMinus-EPSILON) /* -1 is more likely to be correct */
      result=-1;
   else /* both +1 and -1 are equally likely */
      result=0;

   return(result);
}

/*
  ------------------------------------------------
  Function: makeCDF

  Objective: compute the cumulative probability distribution
  function from the probability distribution function
  ------------------------------------------------
*/


void makeCDF(long double *frequency)
{
   int i;

   for (i=1;i<=3;i++)
```



```
      frequency[i]+=frequency[i-1];

}

/*
  ------------------------------------------------
  Function: bincoeffinit

  Objective: compute binomial coefficients, called only
  once, at the beginning
  ------------------------------------------------
*/

void bincoeffinit()
{
  int k,n;

  BinomialCoefficient[0][0] = 1;

  for (n=1;n<=MAXCOEFF-1;n++) {
    BinomialCoefficient[n][0] = 1;
    BinomialCoefficient[n][n] = 1;
        for (k=1; k<n;k++)
          BinomialCoefficient[n][k] =
             BinomialCoefficient[n-1][k-1] + BinomialCoefficient[n-1][k];
  }
}


/*
  ------------------------------------------------
  Function: bincoeff

  Objective: returns binomial coefficients
  ------------------------------------------------
*/

int bincoeff(int n, int k)
{
  if ((k<0) || k>n) return 0;

  return(BinomialCoefficient[n][k]);
}
```