```
C http://arxiv.org/abs/0908.2519
C Krzysztof Malarz <malarz@agh.edu.pl>, Sat, 13 Mar 2010, 15:49:17 CET
      program zaller

      implicit none
      integer i,j,k,it,n,ibm,nnn,kk,ihist,irun,
     +   NMAX,TMAX,NRUN,NHIST,SIZE
      real x,y,r1,r2,a,rr,ppp,xm,ym,nomin,denom,ran0
      logical CON
      parameter(CON=.true.,NMAX=1e3,TMAX=1e2,NRUN=1000,NHIST=50,
     +   a=0.1,ppp=10.0/(1.0*NMAX),SIZE=NMAX*(TMAX+1))
      dimension x(NMAX,SIZE),y(NMAX,SIZE),n(NMAX),xm(NMAX),ym(NMAX),
     +   nomin(NMAX),denom(NMAX),ihist(NHIST)
      data ibm,ihist/1,NHIST*0/

      if(CON) then
         print *,'# with interation among agents, p=',ppp
      else
         print *,'# without interaction among agents'
      endif

      print *,'# NMAX  TMAX  NRUN  mu'
      print *,'#',NMAX,TMAX,NRUN,a
      print *,'# p       N(p)'
      print *,'################################################'

      do 999 irun=1,NRUN

      do i=1,NMAX
        nomin(i)=0.0
        denom(i)=0.0
      enddo

C initial message
      do 001 i=1,NMAX
        x(i,1)=2.0*ran0(ibm)-1.0
        y(i,1)=2.0*ran0(ibm)-1.0
        n(i)=1
c       print *,x(i,1),y(i,1)
 001  enddo

C it = message ID, i = agent Id, n(i) = current number of accepted messages by i
C - 002 - time evolution, gathering new messages
      do 002 it=2,TMAX

        r1=2.0*ran0(ibm)-1.0
        r2=2.0*ran0(ibm)-1.0

        do 003 i=1,NMAX
          nnn=n(i)
          do 004 j=1,nnn
            rr=((r1-x(i,j))**2)+((r2-y(i,j))**2)
            if(rr.le.a*a) then
              n(i)=n(i)+1
              x(i,n(i))=r1
              y(i,n(i))=r2
              goto 005
            endif
 004      enddo
 005      continue
 003    enddo

C for model with interaction among agents
      if(CON) then
C evaluate 'average message' for each agent
        do i=1,NMAX
```

```
          xm(i)=0.0
          ym(i)=0.0
          do j=1,n(i)
            xm(i)=xm(i)+x(i,j)/(1.0*n(i))
            ym(i)=ym(i)+y(i,j)/(1.0*n(i))
          enddo
        enddo

c 'i' sends his average message to 'j' with probability ppp
        do 303 i=1,NMAX
          if(ran0(ibm).le.ppp) then
            do 304 j=1,NMAX
              if(.not.(i.eq.j)) then
                nnn=n(j)
                do 305 k=1,nnn
                  rr=((xm(i)-x(j,k))**2)+((ym(i)-y(j,k))**2)
                  if(rr.le.a*a) then
                    n(j)=n(j)+1
                    x(j,n(j))=xm(i)
                    y(j,n(j))=ym(i)
                    goto 306
                  endif
 305            enddo
              endif
 306        continue
 304        enddo
          endif
 303    enddo
      endif

 002  enddo

C prepare a histogram
      do i=1,NMAX
        do j=1,n(i)
          if(x(i,j).gt.0.0) nomin(i)=nomin(i)+x(i,j)
          denom(i)=denom(i)+abs(x(i,j))
        enddo
      enddo

      do i=1,NMAX
        j=1+(NHIST-1)*nomin(i)/denom(i)
        ihist(j)=ihist(j)+1
      enddo

 999  enddo

      do i=1,NHIST
        print *,(1.0*i)/(1.0*NHIST),ihist(i)
      enddo

      end
```

```
/*
 ============================================================================
 Name        : RAS_Model_2d_net.c
 Version     : 1.0
 Copyright   : Piotr Gronek, 2009
 Description : 2-dimensional simulation of opinion dynamics network
               with proposals treated as messages - corrected 24.06.2009
               Linux edition
 ============================================================================
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <time.h>
#include <errno.h>
#include <string.h>

int ibm=0;
// use this first function to seed the random number generator,
// call this before any of the other functions
void initrand(int seed)
{
        ibm = seed;
}

// generates a pseudo-random double between -0.999 and 0.999...
double randd()
{
        ibm=ibm*16807;
        return 2.0 * fabs(1.0*ibm/2147483647) - 1.0;
}

void addmessage( struct buddy_body *somebody, double mx, double my )
{
        int k;
        struct msgchain {
                double mx;
                double my;
                struct msgchain *nxtchain;
        };
        struct msgchain *chain,  *newchain;
        struct buddy_body {
                struct msgchain *mchain;
                int mchnbr;
                double mxsumplus;
                double mysumplus;
                double mxsummod;
                double mysummod;
                double sumx;
                double sumy;
                long count;
        };

        if ((*somebody).mchnbr == 0) { // initially empty chain - add element
                newchain = (struct msgchain *) malloc(sizeof(struct msgchain));
                if( !newchain ) { printf("2 malloc error no. %d\n", errno); exit(2); }
                (*newchain).mx = mx;
                (*newchain).my = my;
                (*newchain).nxtchain = NULL;
                (*somebody).mchain = newchain;
                newchain = NULL;
        }
        else // otherwise add element at the end of chain
        {
```

```
                // skip to the last chain element
                chain = (*somebody).mchain;
                for(k = 1; k < (*somebody).mchnbr; k++ )
                {
                        chain = (*chain).nxtchain;
                }
                // add new chain element
                newchain = (struct msgchain *) malloc(sizeof(struct msgchain));
                if( !newchain ) { printf("2 malloc error no. %d\n", errno); exit(2); }
                (*newchain).mx = mx;
                (*newchain).my = my;
                (*newchain).nxtchain = NULL;
                (*chain).nxtchain = newchain;
                newchain = NULL;
        }
        (*somebody).mchnbr++; // update chain length
}

// main stuff... :)
int main(int argc, char **argv) {
        int BUDDIES = 2;
        int MESSAGES = 2;
        int INIT_MSGS = 1;
        double LAMBDA = 1.0;
        double RMAX = 4.0;
        int INIT_TYPE = 0;
        int SEED = 0;
        int i, j, k, lmax, l;
        short **edge, *moved;
        int *npacc;
        double x, y, mx, my, kx, ky, px, py, distx[10], disty[10], distxsum, dis
tysum;
        struct msgchain {
                double mx;
                double my;
                struct msgchain *nxtchain;
        };
        struct msgchain *chain, *newchain;
        struct buddy_body {
                struct msgchain *mchain;
                int mchnbr;
                double mxsumplus;
                double mysumplus;
                double mxsummod;
                double mysummod;
                double sumx;
                double sumy;
                long count;
        } *buddy;

        puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */

        if (argc > 1)
        {
                for (i = 1; i < argc;  i += 2 )
                {
                        if ( ! strcmp(argv[i], "-B") ) BUDDIES  = atoi(argv[i+1
]);
                        if ( ! strcmp(argv[i], "-M") ) MESSAGES  = atoi(argv[i+
1]);
                        if ( ! strcmp(argv[i], "-I") ) INIT_MSGS = atoi(argv[i+1
]);
                        if ( ! strcmp(argv[i], "-L") ) LAMBDA = (double) atof(ar
gv[i+1]);
                        if ( ! strcmp(argv[i], "-R") ) RMAX   = (double) atof(a
rgv[i+1]);
```

```
                        if ( ! strcmp(argv[i], "-T") ) INIT_TYPE = atoi(argv[i+1
]);
                        if ( ! strcmp(argv[i], "-S") ) SEED = atoi(argv[i+1]);
                }
        }

        printf("BUDDIES  %d\n", BUDDIES);          // number of agents
        printf("MESSAGES %d\n", MESSAGES);         // number of messages
        printf("INIT_MSGS %d\n", INIT_MSGS);       // number of initializing messag
es, if any
        printf("LAMBDA   %lf\n", LAMBDA);          // mean agent network degree (ag
ent interaction)
        printf("RMAX     %lf\n", RMAX);            // message acceptance / agent in
teraction radius (a.k.a. mu)
        printf("INIT_TYPE %d\n", INIT_TYPE);       // initial agent distribution mo
de
        printf("SEED %d\n", SEED);                 // initialization for pseudorand
om generator

        initrand(SEED);

        buddy = (struct buddy_body *) malloc(sizeof(struct buddy_body) * BUDDIES
);
        if( !buddy ) { printf("1 malloc error no. %d\n", errno); exit(1); }

        chain = (struct msgchain *) malloc(sizeof(struct msgchain));
        if( !chain ) { printf("2 malloc error no. %d\n", errno); exit(2); }
        (*chain).mx = 0.0;
        (*chain).my = 0.0;
        (*chain).nxtchain = NULL;
        free(chain);
        for (i = 0; i < BUDDIES; i++)
        {
                buddy[i].mchain = NULL;
                buddy[i].mchnbr = 0;
                buddy[i].mxsumplus = 0.0;
                buddy[i].mysumplus = 0.0;
                buddy[i].mxsummod = 0.0;
                buddy[i].mysummod = 0.0;
                buddy[i].sumx = 0.0;
                buddy[i].sumy = 0.0;
                buddy[i].count = 0;
        }

        edge = (short **) malloc(sizeof(short *) * BUDDIES);
        if( !edge ) { printf("3 malloc error no. %d\n", errno);  exit(3); }
        for (i = 0; i < BUDDIES; i++)
        {
                edge[i] = (short *) malloc(sizeof(short) * BUDDIES);
                if( !edge[i]) { printf("4 malloc error no. %d\n", errno);  exit(4); }
        }

        // generating network between buddies a.k.a. agents
        for (i = 0; i < BUDDIES; i++)
        {
                for (j = 0; j < BUDDIES; j++)
                {
                        edge[i][j] = 0;
                        if ( i > j ) // lower triangle
                        {
                                if ( ( ( randd() + 1.0 ) * 0.5 ) <= ( LAMBDA / (
double) BUDDIES ) )
                                        { edge[i][j] = edge[j][i] = 1; } // not
overwritten by lower triangle
                        }
                }
```

```
        }

        moved = (short *) malloc(sizeof(short) * BUDDIES);
        if( !moved ) { printf("5 malloc error no. %d\n", errno);  exit(5); }

        npacc = (int *) malloc(sizeof(int) * BUDDIES);
        if( !npacc ) { printf("6 malloc error no. %d\n", errno);  exit(6); }

        for (i = 0; i < BUDDIES; i++)
        {
                moved[i] = npacc[i] = 0;
        }

        distxsum = 0.0;
        distysum = 0.0;
        for (i = 0; i < 10; i++)
        {
                distx[i] = disty[i] = 0.0;
        }

        /**** the stuff .... ***/

        // initial set of random messages for each buddy separately
        for (i = 0; i < BUDDIES; i++) // loop over buddies a.k.a agents
        {
                for (j = 0; j < INIT_MSGS; j++)
                {
                        // new message mx,my arrives
                        mx = randd();
                        my = randd();

                        // update average position
                        buddy[i].sumx += mx;
                        buddy[i].sumy += my;
                        buddy[i].count++;

                        if ( INIT_TYPE == 1 ) // include impact from initial mes
sages
                        {
                                // update message distribution data for later st
atistics
                                if ( mx > 0.0 ) buddy[i].mxsumplus += mx;
                                buddy[i].mxsummod += fabs( mx );
                                if ( my > 0.0 ) buddy[i].mysumplus += my;
                                buddy[i].mysummod += fabs( my );
                        }

                        // add the message to the end of chain
                        addmessage( &buddy[i], mx, my );

                }
        }

        // message processing
        for (i = 0; i < MESSAGES; i++) // loop over messages
        {
                // new message mx,my arrives
                mx = randd();
                my = randd();
                /**/printf("new message[%d] mx= %lf my= %lf\n", i, mx, my);

                for (j = 0; j < BUDDIES; j++) // loop over buddies a.k.a agents
                {
                        chain = buddy[j].mchain;
                        lmax = buddy[j].mchnbr;
                        // if first message
```

```
                        if (chain == NULL) // add first message
                        {
                                x = buddy[j].sumx; // count presumed zero
                                y = buddy[j].sumy;

                                if ( RMAX > sqrt( (x - mx) * (x - mx)
                                                + (y - my) * (y - my) ) )
                                {
                                        // message mx,my within range radius

                                        // mark j's move
                                        moved[j] = 1;

                                        // update average position
                                        buddy[j].sumx += mx;
                                        buddy[j].sumy += my;
                                        buddy[j].count++;

                                        // update message distribution data for
later statistics
                                        if ( mx > 0.0 ) buddy[j].mxsumplus += mx
;
                                        buddy[j].mxsummod += fabs( mx );
                                        if ( my > 0.0 ) buddy[j].mysumplus += my
;
                                        buddy[j].mysummod += fabs( my );

                                        // add the message to the end of chain
                                        addmessage( &buddy[j], mx, my );

                                        //printf("message[%d] mx= %lf my= %lf ac
cepted\n", j, mx, my);
                                }
                        }
                        // if non-first message
                        for (k = 0; k < lmax; k++ ) // loop over buddy's chain
                        {
                                x = (*chain).mx;
                                y = (*chain).my;
                                chain = (*chain).nxtchain;

                                if ( RMAX > sqrt( (x - mx) * (x - mx)
                                                + (y - my) * (y - my) ) )
                                {
                                        // message mx,my within range radius

                                        // mark j's move
                                        moved[j] = 1;

                                        // update average position
                                        buddy[j].sumx += mx;
                                        buddy[j].sumy += my;
                                        buddy[j].count++;

                                        // update message distribution data for
later statistics
                                        if ( mx > 0.0 ) buddy[j].mxsumplus += mx
;
                                        buddy[j].mxsummod += fabs( mx );
                                        if ( my > 0.0 ) buddy[j].mysumplus += my
;
                                        buddy[j].mysummod += fabs( my );

                                        // add the message to the end of chain
                                        addmessage( &buddy[j], mx, my );
```

```
                                        break;
                                }
                        }
                }

                // interacting agents proposal processing
                for (j = 0; j < BUDDIES; j++) // loop over buddies a.k.a agents
for proposals
                {
                        // proposal from j'th as a message
                        mx = buddy[j].sumx / (double) buddy[j].count;
                        my = buddy[j].sumy / (double) buddy[j].count;
                        for (k = j + 1; k < BUDDIES; k++) // go over upper trian
gle only
                        {
                                if( edge[j][k] == 1 ) // edge between given two
buddies
                                {
                                        if ( moved[j] == 1 ) // proposal from j'
th to k'th
                                        {
                                                chain = buddy[k].mchain;
                                                lmax = buddy[k].mchnbr - npacc[k
]; // !!!!!!!!!!
                                                for (l = 0; l < lmax; l++ ) // l
oop over k's chain
                                                {
                                                        x = (*chain).mx;
                                                        y = (*chain).my;
                                                        chain = (*chain).nxtchai
n;

                                                        if ( RMAX > sqrt( (x - m
x) * (x - mx)
                                                                        + (y - m
y) * (y - my) ) )
                                                        {
                                                                // proposal mx,m
y within range radius
                                                                // add the propo
sal to the end of message chain
                                                                addmessage( &bud
dy[k], mx, my );
                                                                // count accepte
d proposals
                                                                npacc[k]++; // !
!!!!!!!!!!!!
                                                                break;
                                                        }
                                                }
                                        }
                                        if ( moved[k] == 1 ) // proposal from k'
th to j'th
                                        {
                                                // proposal from k'th as a messa
ge
                                                kx = buddy[k].sumx / (double) bu
ddy[k].count;
                                                ky = buddy[k].sumy / (double) bu
ddy[k].count;

                                                chain = buddy[j].mchain;
                                                lmax = buddy[j].mchnbr - npacc[j
]; // !!!!!!!!!!
                                                for (l = 0; l < lmax; l++ ) // l
oop over j's chain
                                                {
```

```
                                                        x = (*chain).mx;
                                                        y = (*chain).my;
                                                        chain = (*chain).nxtchai
n;
                                                        if ( RMAX > sqrt( (x - k
x) * (x - kx)
                                                                        + (y - k
y) * (y - ky) ) )
                                                        {
                                                                // proposal mx,m
y within range radius
                                                                // add the messa
ge to the end of chain
                                                                addmessage( &bud
dy[j], kx, ky );
                                                                // count accepte
d proposals
                                                                npacc[j]++; // !
!!!!!!!!!!!!
                                                                break;
                                                        }
                                                }
                                        }
                                }
                        } // end loop over buddies for proposals
                        // finishing proposal statistics
                        for (j = 0; j < BUDDIES; j++) // loop over buddies
                        {
                                moved[j] = 0;
                                for (k = j + 1; k < BUDDIES; k++) // go over upper trian
gle only
                                {
                                        moved[k] = 0;
                                        if ( edge[j][k] == 1 ) // edge between given two
 buddies
                                        {
                                                // update k's statistics
                                                chain = buddy[k].mchain;
                                                lmax = buddy[k].mchnbr - npacc[k]; // !!
!!!!!!!!
                                                for (l = 0; l < lmax; l++ ) // loop over
 k's chain
                                                {
                                                        chain = (*chain).nxtchain;
                                                }
                                                for (l = lmax; l < buddy[k].mchnbr; l++)
                                                {
                                                        x = (*chain).mx;
                                                        y = (*chain).my;
                                                        chain = (*chain).nxtchain;
                                                        // update j'th average position
                                                        buddy[k].sumx += x;
                                                        buddy[k].sumy += y;
                                                        buddy[k].count++;

                                                        // update message distribution d
ata for later statistics
                                                        if ( x > 0.0 ) buddy[k].mxsumplu
s += x;
                                                        buddy[k].mxsummod += fabs( x );
                                                        if ( y > 0.0 ) buddy[k].mysumplu
s += y;
                                                        buddy[k].mysummod += fabs( y );
                                                        // clear accepted proposals
                                                        npacc[k]--; // !!!!!!!!!!!!!
```

```
                                        }
                                        // update j's statistics
                                        chain = buddy[j].mchain;
                                        lmax = buddy[j].mchnbr - npacc[j]; // !!
!!!!!!!!
                                        for (l = 0; l < lmax; l++ ) // loop over
 j's chain
                                        {
                                                chain = (*chain).nxtchain;
                                        }
                                        for (l = lmax; l < buddy[j].mchnbr; l++)
                                        {
                                                x = (*chain).mx;
                                                y = (*chain).my;
                                                chain = (*chain).nxtchain;
                                                // update j'th average position
                                                buddy[j].sumx += x;
                                                buddy[j].sumy += y;
                                                buddy[j].count++;

                                                // update message distribution d
ata for later statistics
                                                if ( x > 0.0 ) buddy[j].mxsumplu
s += x;
                                                buddy[j].mxsummod += fabs( x );
                                                if ( y > 0.0 ) buddy[j].mysumplu
s += y;
                                                buddy[j].mysummod += fabs( y );
                                                // clear accepted proposals
                                                npacc[j]--; // !!!!!!!!!!!!!
                                        }
                                }
                        }
                } // end loop over buddies
        } // end loop over messages

        // final printing
        for (i = 0; i < BUDDIES; i++) // loop over buddies
        {
                chain = buddy[i].mchain;
                if (chain == NULL) { printf("end buddy[%d] empty chain\n", i); break;
}
                for(k = 0; k < buddy[i].mchnbr; k++ )
                {
                        x = (*chain).mx;
                        y = (*chain).my;
                        chain = (*chain).nxtchain;
                }

                // calculate message probabilities
                px = py = 0.0;
                if ( buddy[i].mxsummod > 0.0 ) px = buddy[i].mxsumplus / buddy[i
].mxsummod;
                if ( buddy[i].mysummod > 0.0 ) py = buddy[i].mysumplus / buddy[i
].mysummod;

                printf("end buddy[%d] avg pos x= %lf y= %lf px= %lf py= %lf\n",
                                i, buddy[i].sumx / (double) buddy[i].count,
                                        buddy[i].sumy / (double) buddy[i].count,
 px, py);

                // record input into probability distribution beans
                for (j=0; j < 10; j++)
                {
                        if ( 10.0 * px > j + 1 ) continue;
                        distx[j] += ( 1.0 / BUDDIES );
```

```
                        break;
                }
                for (j=0; j < 10; j++)
                {
                        if ( 10.0 * py > j + 1 ) continue;
                        disty[j] += ( 1.0 / BUDDIES );
                        break;
                }
        }

        for (i = 0; i < 10; i++) // loop over distribution beans
        {
                distxsum += distx[i];
                distysum += disty[i];
        } // end of loop over distribution beans

        for (i = 0; i < 10; i++) // loop over distribution beans
        {
                printf("P(px=[%d]): %lf, P(py=[%d]): %lf\n",
                                        i, distxsum ? distx[i] / distxsum : 0.0,

                                        i, distysum ? disty[i] / distysum : 0.0)
;
        } // end of loop over distribution beans

        return EXIT_SUCCESS;

}
```

```
#  64-bit binary code name:
#  ras_8.2_64
#  runtime parameters:
#  -B 1000 -M 100 -I 1 -L 5.0 -R 0.2 -T 0 -S $i


for (( i = 65757555 ; i < 65757555 + 2000 ; i+=2 )); \
        do ./ras_8.2_64 -B 1000 -M 100 -I 1 -L 5.0 -R 0.2 -T 0 -S $i | \
        grep 'P(px=' | awk '{ print $2 " " $4 ;}' | \
        sed -s s/,// >> out.ras_v8.2_B1kM100I1L5R0.2T0.csv ; done
```