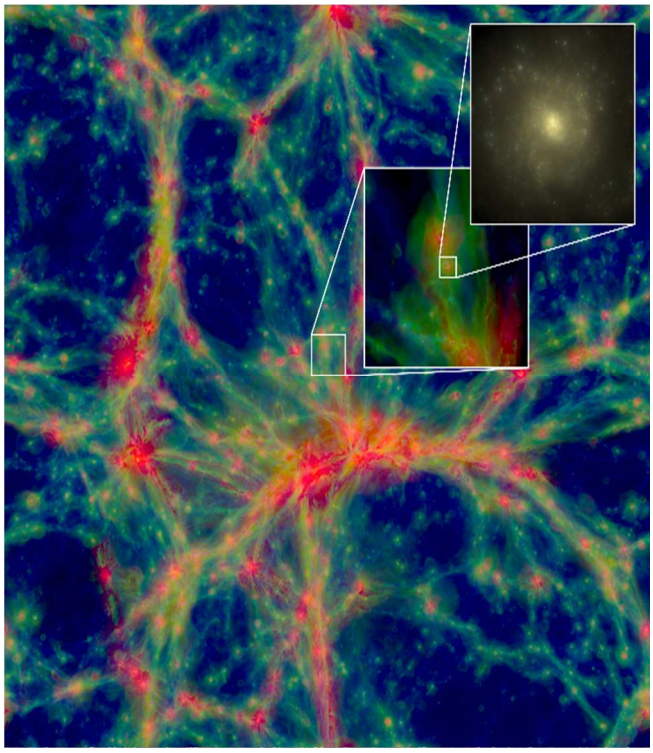


Simulering av smittspridning inom en viss population av individer

Simulering och modulering

SAMMANFATTNING

Simulering och modellering av smittspridning av en sjukdom inom en begränsad population av individer med hjälp av JAVA programmering under begränsande omständigheter

ben mul

KTH, II1304

Innehållsförteckning

1. Abstrakt

Rapporten som du har nu i dina händer, behandlar simuleringen av smittspridning inom en begränsad population. I populationen finns det en sjuk individ eller flera sjuka individer med sannolikheten ps att smitta sina närmaste grannar inom deras geografiska område. Syftet här är att kunna förutsäga eller analysera hur en sådan sjukdom skulle kunna utvecklas under en viss tid i den givna populationen. Vidare kommer vi att se vilka parameter som påverkar sannolikhet att sjukdomen dö ut inom population det vill säga sannolikhet att alla levande individer är antingen friska eller osmittade. Population har implementerats som en 2D matris där varje punkt representerar en individ. Varje individ har ett grannskap, detta definieras som de individerna som ligger närmaste, avstånd 1. För att kunna simulera detta har vi använt oss av programmeringen med programmeringsspråket Java. För att hantera data på ett enkelt sätt och förenkla beräkningar har vi begränsat oss till en population av 9 respektive 25 individer. Programmet återges i bilaga och kan köras för flera individer. Vi har kommit fram till att i en population av 9 individer där det initialt finns en sjuk individ så dör sjukdomen ut i medelvärdet efter 11 dagar. Resultatet skiljer sig åt mellan olika körningar och olika initiala sannolikheter.

2. Introduktion

Det givna problemet [1] är att simulera hur en smittspridning skulle kunna utvecklas efter ett visst antal dagar, givet att det finns N*N individer placerade inom ett visst geografiskt område och att bland de individerna finns det M sjuka individer placerade inom samma geografiska område. Varje individ har en sannolikhet ps att smittar sina närmaste grannar. Varje sjuk individ har sannolikhet px att dör och ingen smittad individ kan spridda sjukdomen vidare under samma dag som den själv blev infekterad. En sjuk individ kan förbli sjuk minst efter min dagar och högst efter max dagar.

Samma frågeställning finnes i [2] där författarna diskuterat modellering och lösningen till problemet som vi i den här rapporten har försökt implementerat. I [2] har man modellerat problemet med en lång lista som i programkod representeras med en endimensionell vektor [1] där vissa sjuka individer har placeras i godtyckliga positioner i vektorn. Sedan använder de en kvot för att beräkna hastighet eller farten med vilken en viss individ kan smitta andra, alltså hur snabbt smittspridningen utvecklas. Efter en viss tid så dör sjukdomen ut under vissa fall [2].

Vi har i vårt arbete använt ett annorlunda tillvägagångssätt. Problemet har modellerats som en stor matris M[N][N] av storlek N där det totala antalet individer är N*N, Varje individ I, i position (X,Y) placeras i matrisen på plats M[X][Y]. En individ kan bara smitta sina närmaste grannar. Dessutom har vi en funktion som itererar igenom Matris eller populationen en gång och ändrar värden i matrisen utifrån de ovan beskrivna villkor. Varje sådana anrop till den givna funktionen utgör en dag i vår modell.

[1] Array i java

[2] Eftersom vi använder oss av en slumptalsgeneratorer kan sjukdomen dör ut i vissa fall och teoretiskt köra oändligt lång utan att sjukdomen helt dör ut.

3. Metod

3.1 Modellering

I problembeskrivning står det inte tydligt vilken modell man ska implementationsmässigt använda: Varje val av modell påverkar dels vilka resultat man får des om de erhållna resultaten är tillräckliga tillförlitliga för att dra slutsatser om ursprungsproblemet. En bra modell ska i så fall eftersträva att återspegla så bra som möjligt det verkliga problemet - bara med mindre detaljer. Problemets egentliga natur ska förbli detsamma. Vi har kommit fram till 5 olika modeller som man skulle kunna implementera och använda för att simulera det givna problemet.

3.1.1 En endimensionell vektor

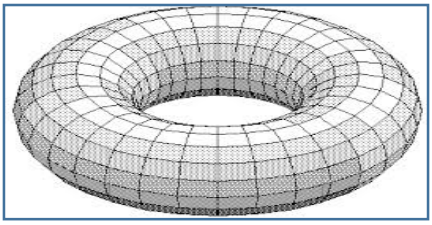
Precis som [2] skulle vi kunna använda oss en endimensionell vektor med ett system som gör det möjligt att en individ kan smitta alla sina närmaste grannar [3] . Tyvärr så stred den modell problembeskrivning i [1].

3.1.2 En enkel matris

En enkel matris kan användas för modellera problemet på så sätt att en individ i plats M[X][Y] kan bara smittar sina grannar på plats M[X - 1][Y], M[X+1][Y], M[X][Y +1], M[X][X-1], M[X-1][Y +1], M[X-1][Y-1], M[X+1][Y -1], M[X+1][X+1] om M[X][Y] befinner sig i mitten av matris och om individen M[X][Y] befinner sig på en av kanterna av matrisen så har individen bara 5 respektive 3 grannar att eventuellt smittar. Grannskapet är större för vissa individer och mindre för andra

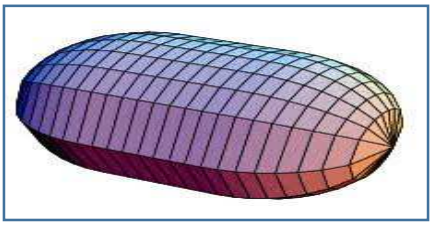
3.1.3 En torusmodell

En tredje modell vore att använda matris och representerar relationerna mellan olika individer i matrisen som figur X nedan visar.



3.1.4 En cylindrisk modell

En annan modell är att beskriva relationerna mellan olika individer i matrisen som figuren X visar.



[3] I det här fallet vad som bedöms närmast är en implementationsspecifik fråga

3.1.5 Sfärisk modell med matris

Slutligen kan man också använda sig av en sfärisk modell där varje individ har samma sannolikhet att smitta sina närmaste grannar som att smitta avlägsna individer.

3.2 Vår modell

För att modellera det här problemet hade vi använt oss av modellen beskriven i 3.1.2.

3.2.1 Modellens begränsningar

Det finns flera anledningar till vårt val att använda den andra modellen framför alla andra modeller. Modellen 1 strider mot programspecifikation [1]. I vår modell så har individerna inte samma antal grannar, vissa individer kommer att vara mer utsatta än andra och vissa kommer att kunna smitta flera individer och andra färre. Så smittspridningen blir olika beroende på var individen befinner sig i matrisen. Den modellen speglar ganska bra en verklig situation. I en verklig situation kan man först bara smitta de individer man kommer till direkt kontakt med, de individerna kan i sin tur föra vidare sjukdomen till sina bekantskap.

I sista modellen så är sannolikhet lita stor för varje individ att smittar så som att låta sig smittas. Detta kan vara bra för modellera en mängd människor eller en mängd individer som befinner sig i ett väl begränsat område (by) där varje individer har kontakt med varandra.

Modellen 2 återspeglar vårt problem på bästa sättet, jämför med modellen 3 och 4 där man måste tar hänsyn till alla fall. Implementation blir ganska komplex för alla sådana fall man ska koppla till varje individ i en viss position.

3.3 Verktyg

För att kunna arbetat med vår modell och simulera den har vi använt flera olika hjälpmedel.

- Programmering – JAVA 8
- Jackson strukturerad programmering –
- UML
- TDD

Programmeringsspråket Java har använts flitigt för att implementerat vår modell med alla dess funktioner. Eftersom Java är lättillgänglig, användarvänlig och dessutom objektorienterad så passade det väldigt bra implementationen av vår modell.

Jackson Strukturerad programmering har använt för att specificera vårt problem och modellera vad vårt program ska klara av att göra. UML har använts för att specificera och dokumentera alla våra klasser med dess datatyper.

TDD står för en test driven utveckling (test driven developpement) varje moduler har testat var för sig (modultest) sedan i samband med andra moduler (integrationstest) efter eventuella bug så har koden analyseras på nytt och sedan testats (regressionstest).

3.4 Arbetet

Arbetet påbörjades med en kravanalys av vad programmet ska klara av att göra samt vad det inte ska göra. Sedan studerade vi olika fall som kan uppstå om en individ är i position (i, j) av matrisen. Vi implementerade sedan möjligheten att snabbt få informationen om grannskap för varje individ genom att få individens positions som parameter. Efter studier av alla dessa fall implementerade vi en pseudokod med hjälp av JSP[4]. Dessa sedan översättas till Java kod i utvecklingsmiljö Intellj.

Varje implementerad klass och objekt testades med hjälp av test driven utveckling med dessa steg

- Modultest
- Integrationstest
- Regressionstest

3.5 Programspecifikation

Vi ska här redogöra för vår analys av problemet och redovisa vårt förslag på till hur simulering skulle ha kunnat implementeras [1]. Alla andra steg hittas i Bilaga D.

3.5.1 Indata och utdata

Enligt det som står i kravspecifikation [1] så ska programmet har följande:

Indata

Användaren skall kunna sätta värden för följande parametrar:

- Storlek på populationen (N)
- Smittsannolikhet (per dygn) att en sjuk individ smittar en frisk granne som inte är immun (S)
- Längd på den tid man är sjuk, rektangulär fördelat i ett intervall [minDygn, maxDygn]
- Sannolikhet att en individ avlider per dygn individen är sjuk (L)
- Hur många individer som är sjuka initialt och var de är placerade

Utdata från simuleringen skall vara:

- Antal smittade varje dygn
- Antal som avlidit varje dygn
- Antal individer som tillfrisknar varje dygn
- Antalet sjuka varje dygn
- Ackumulerat antal smittade varje dygn
- Ackumulerat antal avlidna varje dygn

[4] Jackson strukturerad programmering

3.5.2 Förslag till lösning och dess begränsning

Vårt tillvägagångsätt var att implementera vår lösning till problemet som en 2D matris. Efter långa reflektioner hade vi identifierat tre övergripande klasser som skulle hjälpa oss att simulera smittspridningen. Dessa klasser är följande:

- Individ()
- Pos()
- Population()
- Main()

I klassen individ() finns det olika attributet som en individ kan tänkas ha t.ex. levande, död, placering, max dag, smittad m.m.

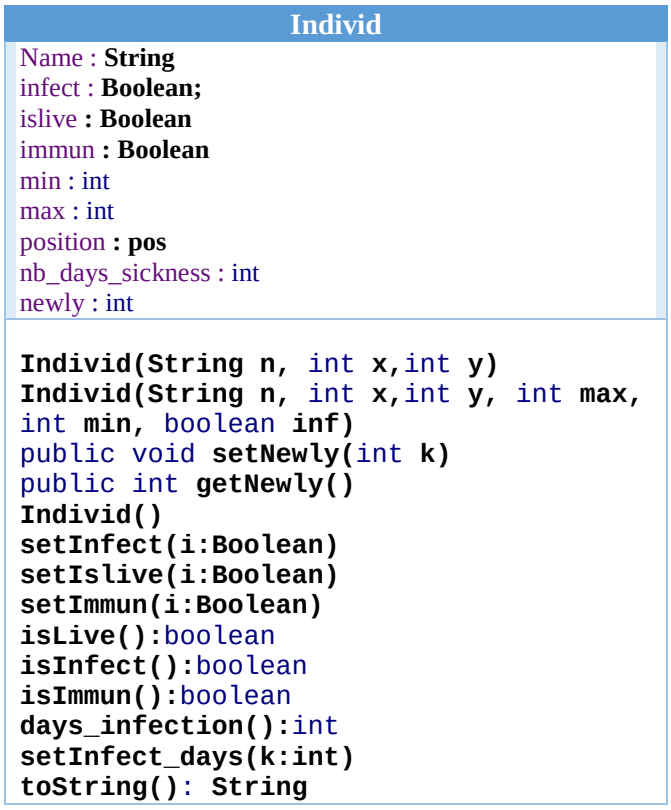
I klassen Pos() så får vi grannskapet till varje individ given dess position.

I klassen Population() så har vi en 2D matris av ett objekt individ(). Matrisen representerar den population som Main() ska arbeta med för att simulera hela smittspridningen. Begränsningen här är att individer har olika antal grannar beroende på var de i matrisen befinner sig. Dessutom kan en individ vara immun men ändå bära på sjukdomen[5].

3.5.3 UML för alla klasser

UML står för **Unified Modeling Language[6],** det är ett modelleringsspråk som används inom mjukvaruutveckling för ge ett standardiserat sätt att visualisera design av mjukvarusystem. Jag ska i den här delen bara använda UML för beskriva alla mina klasser, så kallad UML Klassdiagram [4]. Vi ska inte beskriva hur olika steg i programmet körs. Detta redogörs i stället med JSP i 3.5.4.

3.5.3.1 Individ()



[5] Detta är för att få en simulering som liknar verkligheten. Det kan ändras i koden om så önskas
[6] https://en.wikipedia.org/wiki/Unified_Modeling_Language

3.5.3.2 Pos()

Pos

```
posX; int
posY; int
```

```
pos(int x,int y)
getUp(): int []
getDown(): int []
getRight() : int []
getLeft() : int []
getQ1() : int []
getQ2() : int []
getQ3() : int []
getQ4() : int []
toString() : String
```

3.5.3.3 Population()

Population

```
NEW_LINE_SEPARATOR : String
N : int
ps : double
min : int
max : int
ps_x : double
Ns : int
posX : int
posY : int
Individ [][] pop
SM,X,F,S,AS,AX,TF,Frisk :int
epidemi : int
isEpidem    boolean i;
FileWriter fileWriter = null;
```

```
population()
population(m:int,mi:int, p_s:int NN:int, pxx: int:antal_sjuka,
places:int[])
getDataOutput() : int []
getDataOutputString() : String
getDataOutputString(d : int) : void
getNear(x1 : int, y1 : int) : ArrayList<int[]> neighbours
neighbours(posX : int, posY : int, P : Individ [][]):ArrayList<int[]>
situation(x : int, y : int, n : int) : int
display()
random_choose(univ : ArrayList<int[]>,prob : double) : ArrayList<int[]>
infect()
stop(): int
simul()
simulate(N : int)
simulate()
```

3.5.4 JSP för programmet

ÖÖ

3.6 Implementation och Simulering

3.6.1 Pos()

Som det står i tidigare paragrafer ska funktionen pos() returnerar position av grannskapet av ett objekt individ. Den fullständiga implementation hittas i bilaga A.

3.6.1.1 Tester

I Main() programmet så skrev vi den här:

I en population av storlek 25,

```
population pop = new population();
ArrayList<int[]> p = pop.getNear(2,2);
for(int [] c : p)
    System.out.println(Arrays.toString(c));
```

Utdata blev :

[1, 2],[2, 3],[3, 2],[2, 1],[1, 3],[1, 1],[3, 1],[3, 3].

På så sätt kunna vi dra slutsatsen att programmet var implementerat på rätt sätt. I bilaga A kan man se programmet i sin helhet.

3.6.2 Individ()

Här testade vi i samband med population att objekt individen var implementerat på rätt sätt. Några logiska fel upptäcktes och rättades därefter.

3.6.2.1 Implementation av individer

Den fullständiga implementation hittas i bilaga A.

3.6.2.2 Tester

Alla datatyper som finns i objekt individ testades för att säkerställa att individer använde alla sina attributet på rätt sätt. Dessutom infördes en ny int variabel ”newly” som gjorde det möjligt att skilja mellan de som nyligen blev infekterade och de som har varit infekterade för längesen. För icke infekterade så är newly = -1, för nyligen smittade så är det 0 och för alla andra sjuka 1.

3.6.3 Population()

Population representerar vårt simuleringsunderlag, det vill säga alla möjliga individer som ska finns inom det geografiska området som smittan verkar i. Population är en 2D matris av objekt Individ.

3.6.3.1 Implementation av population

Klassdiagram visar den fullständiga metods datatyper och koden finns i bilaga A.

3.6.3.2 Tester

Här testade vi först funktionen random_choose(). Den använder Math.random() [7] i java som returnera ett flyttal mellan 0.0 till 1.0. Det flyttalet har sedan används för att välja ut på måfå vilka av en smittad individs grannar ska smittas vid en viss tidpunkt.

Så här testades random_choose :

```
population pop = new population();
//pop.simul();
ArrayList<int[]> p = pop.getNear(2,2);
ArrayList<int[]> q = pop.random_choose(p,0.5);
for(int [] c : q)
    System.out.println(Arrays.toString(c));
```

körning 1

[3, 3],[3, 1],[2, 3],[2, 1]

Körning 2

[2, 3],[3, 1],[1, 1],[3, 3]

Här ser vi att körning 1 skiljer sig från körning 2. Stokastiskhet har alltså implementerat på rätt sätt. Vi kan vara säkra på att vi inte kommer att smitta samma individer flera gånger. Implementation hittas i bilaga A.

Sedan testades infect() och flera logiska fel hittades och åtgärdades.

3.6.4 Simulering av programmet

För att simulera programmet skapade vi tre olika funktioner i klassen population:

- simul()
- simulate(N:int)
- simulate()

Anledning till vi skapade flera tillsinnes lika funktioner är dels för att underlätta felsökning dels för att underlätta olika simuleringar. Den första funktionen löser problemet enligt programspecifikation [1]. I den andra funktionen kan man stega igenom varje iteration t.ex. jag vill simulera för 18 eller 100 dagar. I den tredje funktionen körs programmet fram till antal daga är lika med N = indata för storleken av population.

[7] https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

3.6.4.1 Simulering

Genom att köra simul() för ps= 0.3 px = 0.2 [5,2] N = 3 får man resultat som i Bilaga B.

3.6.4.2 Tester

Olika resultat och analyser för ps= 0.2 px = 0.2 [5,2] population= 9 finns under rubriken Resultat och i bilaga C.

4. Resultat

4.1. Population av 9 individer

4.1.1 Data från en körning

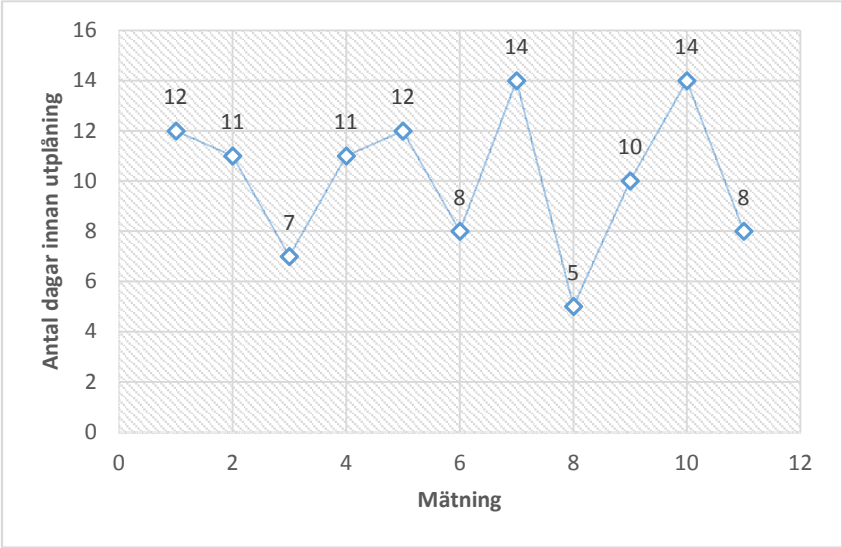
För den är här körning så är Ps ([8]) = 0.2 och Px ([9]) = 0.2. Maximal respektive minimala antal dagar för sjukdomen är 5 respektive 3.

Dag	Döda	Antal friska	Antal sjuka	Immuna	Smittade per dag
1	0	8	1	0	0
2	0	7	2	0	1
3	0	6	3	0	1
4	1	5	3	0	1
5	2	4	3	0	1
6	3	4	2	1	1
7	3	2	4	1	2
8	4	1	4	1	0
9	5	1	3	1	0
10	5	2	2	2	0
11	5	3	1	3	0
12	5	4	0	4	0

[8] Sannolikheten att en sjuk individ smittar individer i sitt grannskap

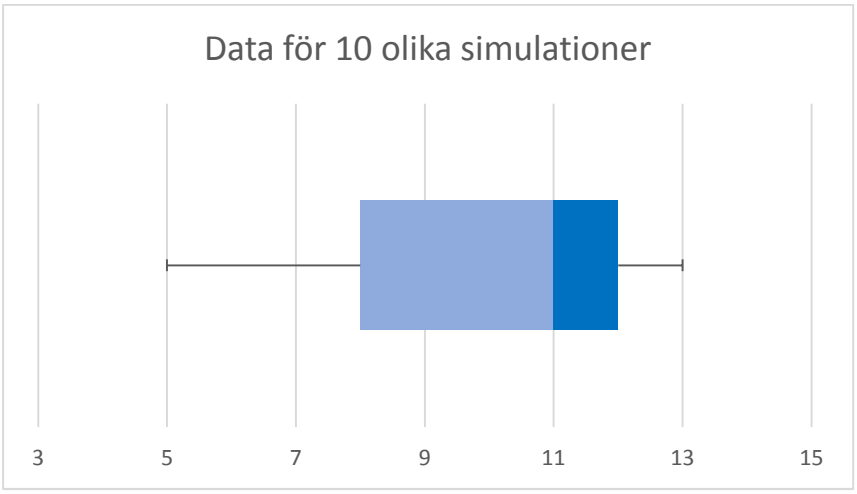
[9] Sannolikhet att en sjuk individ avlider

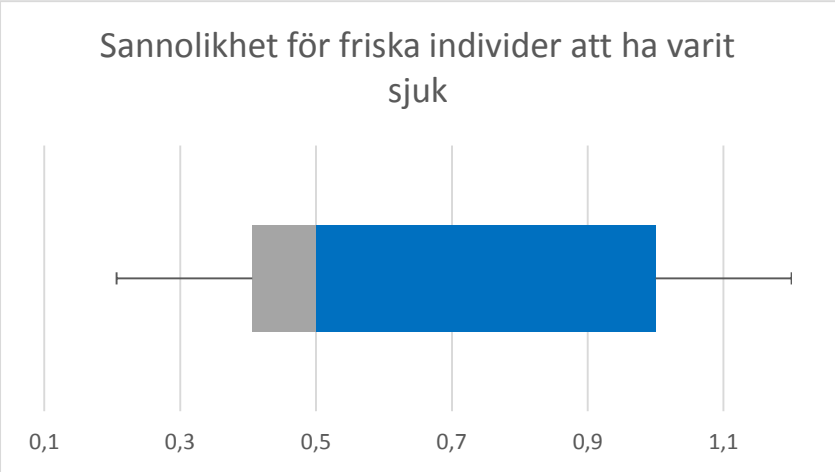
4.1.2 Tid det tar för sjukdomen att utplånas för 11 körningar



I alla de 11 körningar som visas i tabell X så är ps=px= 0.2 och [max,min] = [5,2]. För de 10 första mätning så är medelvärde = 10.4 och standard avvikelse s = 2.95. Alla beräkningar har gjorts utifrån statiska definitioner som kan hittas i [3]. Medianen är m = 11 box ritning för det visas nedan

.





Räknar man konfidens med 95% procent säkerhet så blir intervallet [8.3,12.5]. Beräkningarna hittas i Bilaga E. Vi kan vara ganska säkra på att sjukdomen kommer att dör ut mellan 8 till 13 dagar givet att ps=px=0.2, [max,min] =[5,3] och population = 9.

4.2 Population av 25

Här kommer vi att visa bara de 10 första iterationer samt hur de förhåller sig med varandra.

4.2.1 Ps = 0.2 Px = 0.2 [max,min] = [5,3]

I det här fallet så dör sjukdomen inte ut. Om man antar att en sjuk individ ändå kan vara immun efter ett visst antal dagar. I Längden kommer det att återstår efter flera körningar en sjukdom bärande individ som är immun men kan ej smittar sina grannar eftersom alla individer i sitt grannskap är antingen immuna eller döda, vilket ledder till en oändlig körning. Utanför det antagande så kommer alla körningar att bete sig som på fallet beskrivet i 4.2.2. Alla körningar kommer då att avslutas efter ungefär 10 dagar.

Dag	*Döda*	*Antal friska*	*Antal sjuka*	*Immuna*	*Smittade per dag*
1	0	24	1	0	0
2	0	23	2	0	1
3	1	21	3	0	1
4	1	19	5	0	1
5	2	16	7	0	1
6	3	13	9	1	1
7	6	12	7	1	2
8	8	11	6	1	0
9	9	11	5	1	0
10	10	13	2	2	0
….	…	…	…	…	…
100	10	14	1	4	0

4.2.2 Ps = 1 Px = 0.5 [max,min] = [5,3]

I det här fallet så dör sjukdomen ut efter 11 dagar.

Dag	*Döda*	*Antal friska*	*Antal sjuka*	*Immuna*	*Smittade per dag*
1	0	24	1	0	0
2	1	19	5	0	4
3	10	12	3	0	1
4	12	10	3	0	1
5	13	8	4	0	1
6	13	7	5	0	1
7	15	7	3	2	1
8	16	7	2	2	0
9	16	8	1	3	0
10	16	8	1	3	0
11	16	9	0	4	0

4.3 Population 81 respektive 1024 individer

Här är resultatet nästan samma som för en population av 25 individer. Sjukdomen dör inte ut för fall 1 och sjukdomen dör ut efter ungefär 10 dagar för 81 individer. För 1024 individer så dör sjukdomen ut ungefär efter 23 till 36 dagar för fall 2 men sjukdomen dör inte för fall 1. Alla resultat hittas i bilaga C tillsammans med koden.

5. Diskussion

För ps <= 0.3 så har det visats sig att sjukdomen inte nödvändigtvis dör ut. För en population som är mindre 9, så dör sjukdomen ut med största sannolikhet oavsett parameter för ps med px >= 0.3. Olika samlade data [10]med olika sannolikheter för px pch ps, visar att storleken på population i sig har ingen inverka på sannolikhet för att sjukdomen ska dör ut. Däremot tar det naturligtvis längre tid för sjukdomen att dör ut i en population av 1024 individer jämfört med en population av 16 individer.

För en population >= 16 individer där är parametern för px helt avgörande för att sjukdomen ska ens får en chans att utplånas efter ett visst antal dagar. För en max dag tillräckligt stor så tar det naturligtvis längre tid för sjukdomen att utplånas. Men det maximala antalet av dagar en individ får vara sjuk är i sammanhanget inte avgörande för att smittan utplånas.

För ps < 0.2 så spelar px ingen roll i polulation där N*N <= 16 . För ps > = 0.2 så är px avgörande för att sjukdomen ska dö ut, här spelar storleken av population ingen roll. Alltså en förutsättning för att sjukdomen ska i längden dö ut är att px >= 0.5, i så fall blir ps obetydlig. T.ex. vid en population på 1024 individer med ps = 1 och px = 0.5 så dör sjukdomen efter 23 dagar – 36. Rent intuitivt verkar där här resultatet stämmar. Om vi antar att sannolikhet av sjukdom bärande individer att dö är hög så minskar det sannolikhet att sjukdomen sprids, vilket i sin tur resulterar i att sjukdomen så småningom dör ut och resten av populationen är antingen immuna eller inte alls smittade.

Det finns nämligen en betingning mellan det betingade sannolikhet Px = P(dör | sjuk) och P_ut = P(sannolikhet att sjukdomen dör ut). Sambandet mellan de två sannolikheter är tydligt i våra simulationer däremot ha något matematiskt samband inte härletts. Däremot kan vi observera att P(dör ∩ sjuk) ≠ P(dör)P(sjuk), de två sannolikheter är inte oberoende. Den första är lika med noll och den andra är olika beroende på körningar med vi vet att för alla körningar så är P(dör)P(sjuk) ≠ 0.

[10] Olika körningar kan fås i bilaga tillsammans med programkoden

Rent logiskt kan vi lista varför Ps är betingad till P_ut. Om vi antar Storlek på population är N och storlek på den sjuka populationen är X med $X \subset N$, B =(dör och sjuk). Vidare vet vi att Px = P(dör | sjuk), varje gång eventet B inträffar får det här sambandet X = X – 1 och N = N -1, därmed så minskas sannolikhet att sjukdomen ska spridas nämligen sannolikheten Ps minskar eftersom sjukdom bärande individer minskar när event B inträffar. Alltså kan vi dra slutsatsen att om sjukdomen ska i längden dör ut så måste den delen av population som bär på sjukdomen dör ut innan de hinner sprida sjukdomen vidare.

Vidare har våra resultat visat att av dem som överlever sjukdomen så är det i genomsnitt 50% som har varit smittade men har tillfrisknat under smittspridningsperioden. För att sjukdomen helt ska utplånas så måste ps >= 0.5 i vårt fall och tillfriskningsperioden << 10.

6. Avslutande Reflektioner om modulering

Som det står i högskoleförordningen [6] så ska ingenjör har förmåga att kunna förutsäga och fatta beslut även begränsade underlag eller data. För att kunna klara av detta, ska ingenjörer har förmåga att modellera problemet, simulera den för att på så sätt veta hur det skulle ha förhållit efter en viss tid. Den förmåga är kritisk för alla ingenjörer som Leslie [5] beskriver. För att kunna bygga ett hus ska man först analysera vad det är som ska byggas och hur det ska byggas. Sedan ska man skapa en modell för hur det slutliga huset ska se ut. Den modellen ska sedan implementeras i det här fallet byggas och man kan alltid gå tillbaka till ursprungsmodellen för att se om våra tidigare antagande håller.

Den uppgiften fick oss studenter att analysera över ett problem, komma man fram till en modell, implementera modellen för att sedan dra slutsatser utifrån de resultat man får. För att det ska vara korrekt så måste den framtagna modellen stämmar överens med verklighet eller problemet man försöker modellera. Sedan ska modellen granskas, valideras eller förkastas. Den är en naturlig vetenskaplig process som ingenjörer och blivande ingenjörer bör känna till.

Kunskaper kopplade till simulering och modellering kan även användas i andra sammanhang till exempel när man försöker veta riktighet av en hypotes då kan man modellera situation och se om hypotesen håller eller inte. Detta bidrar till mer kritiskt tänkande och eftertanke genom hela tiden se om resultat stämmer överens med den verklighet man har försökt abstrahera. Sedan bidrar allmänt till en bättre problemlösning. Problem som går att lösa med naturvetenskap såklart inte existentiella problem.

Referenser

[1] R. Rönngren, “Simulering av smittspridning v1.1,” pp. 0–12.

[2] Alan W. Biermann and Dietolf Ramm, *Great Ideas in Computer Science with Java*, 1st ed. The MIT Press, 2016.

[3] A. Hayter, *Probability and statistics for engineers and scientists*. 2012.

[4] Y. D. Liang, *Introduction To Java Programming Comprehensive*. 2011.

BILAGA A

Pos

```
/**
 * Created by ben on 2017-09-27.
 */
public class pos
{
    int posX;
    int posY;
    pos(int x,int y)
    {
        this.posX = x;
        this.posY = y;
    }

    public int [] getUp()
    {
        int [] f = {posX - 1, posY};
        return f;
    }
    public  int [] getDown()
    {
        int [] f = {posX + 1, posY};
        return f;
    }

    public int [] getRight()
    {
        int [] f = {posX, posY + 1};
        return f;
    }
    public  int [] getLeft()
    {
        int [] f = {posX, posY - 1};
        return f;
    }
    public  int [] getQ1()
    {
        int [] f = {posX - 1, posY + 1};
        return f;
    }
    public  int [] getQ2()
    {
        int [] f = {posX - 1, posY - 1};
        return f;
    }
    public  int [] getQ3()
    {
        int [] f = {posX + 1, posY - 1};
        return f;
    }
    public  int [] getQ4()
    {
        int [] f = {posX + 1, posY + 1};
        return f;
    }
    public String toString()
```

```
    {
        return "[ X = " + posX + " , Y = " + posY + " ] ";
    }
}
```

Individ

```
/**
 * Created by ben on 2017-09-26.
 */
public class Individ
{
    String name;
    Boolean infect;
    Boolean islive;
    Boolean immun;
    int min;
    int max;
    pos position;
    int nb_days_sickness;
    int newly;

    Individ(String n, int x,int y)
    {
        this.name = n;
        this.infect = false;
        this.islive = true;
        this.immun = false;
        //this.lifetime = new int [2];
        //lifetime[0] = 5;
        this.min = 3;
        //lifetime[1] = 10;
        this.max = 5;
        this.position = new pos(x,y);
        //numbers of days elapsed since the individ were infected
        this.nb_days_sickness = 0;
        newly = -1;
    }

    Individ(String n, int x,int y, boolean inf)
    {
        this.name = n;
        this.infect = inf;
        this.islive = true;
        this.immun = false;
        //this.lifetime = new int [2];
        //lifetime[0] = 5;
        //lifetime[1] = 10;
        this.min = 3;
        //lifetime[1] = 10;
        this.max = 5;
        this.position = new pos(x,y);
        //numbers of days elapsed since the individ were infected
        //make sure to make to 2 so that people newly infected with
        // value 1 cannot infect theirs neighbours until their values
        // increment to 2 or more in the next days of infection
        this.nb_days_sickness = 1;
        this.newly = 0;
    }
    public void setNewly(int k)
    {
```

```
    this.newly = k;
}
public int getNewly()
{
    return this.newly;
}


Individ()
{
    this.infect = false;
    this.islive = true;
    this.immun = false;
    this.min = 5;
    //lifetime[1] = 10;
    this.max = 10;
}

public void setInfect(Boolean f)
{
    this.infect = f;
    this.nb_days_sickness++;
    this.newly = 1;
}
public void setInfectImmun(Boolean f)
{
    this.infect = f;
}
public void setIslive(Boolean l)
{
    this.islive = l;
}
public void setLifetime(int m, int mx)
{
    this.min = m;
    //lifetime[1] = 10;
    this.max = mx;
}
public int getMax()
{
    return this.max;
}
public int getMin()
{
    return this.min;
}

public void setImmun(Boolean i)
{
    this.immun = i;
}

public boolean isLive()
{
    return islive;
}
public boolean isInfect()
{
    return infect;
}
public boolean isImmun()
```

```
    {
        return immun;
    }
    public void days_infection()
    {
        this.nb_days_sickness++;
    }
    public void setInfect_days(int f)
    {
        this.nb_days_sickness = f;
    }

    public String toString()
    {
        return "[" + " inf = " + infect + " , sick_d = " + nb_days_sickness
+ ", im = " + this.immun + ", isliv " + isLive() +  " ] ";
    }
}
```

Population

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * Created by ben on 2017-09-26.
 */
public class population
{

//Delimiter used in CSV file
private static final String COMMA_DELIMITER = ",";
    private static final String NEW_LINE_SEPARATOR = "\n";
    //****************** data input ************************
    //size of the population
    int N;
    //probability that a sick individ infects its neighbours
    double ps;
    //the lifetime of the disease for each individ
    int min;
    int max;
    //probability that an individ X die iff X is sick
    double ps_x;
    //number of sick individs and their position
    int Ns;
    int posX;
    int posY;
    Individ [][] pop;
    //******************************************************
    int SM,X,F,S,AS,AX,TF,Frisk;
    int epidemi;
    boolean isEpidemi;

    ArrayList<Object> coord;
    FileWriter fileWriter = null;
    population()
    {
```

```
        //****************** data input ************************
        //size of the population
        this.N = 3;
        //probability that a sick individ infects its neighbours
        this.ps = 0.2;
        //the lifetime of the disease for each individ
        this.min = 3;
        this.max = 5;
        //probability that an individ X die iff X is sick
        this.ps_x = 0.2;
        //number of sick individs and their position
        this.Ns = 2;
        this.posX = (int)(this.N/2);
        this.posY = (int)(this.N/2);
        //*********************************************************
/*        System.out.println("X = "  + this.posX);
        System.out.println("X = "  + this.posY);*/

        this.coord = new ArrayList<Object>();
        this.pop = new Individ[N][N];
        for(int i = 0; i < N; i++)
        {
            for(int j = 0; j < N; j++)
            {
                pop[i][j] = new Individ( " " + i + "" + j, i,j);
                if( i == this.posX && j == this.posY)
                    pop[i][j] = new Individ( " " + i + "" + j, i,j,true);
            }
        }
        //******* initialize the data output parameter *************
        SM = 0;
        X = 0;
        F=0;
        S=1;
        AS=0;
        AX=0;
        //************** end of initialization **********************
        epidemi = (int)((this.N*this.N)/2) + 1;
        this.isEpidemi = false;
    }

    //****custom population constructor ***********
    population(int maxx,int minn, int p_s,int NN,int pxx, int antal_sjuka,
int [] dessplacering)
    {
        //Scanner in
        //****************** data input ************************
        //size of the population
        this.N = NN;
        //probability that a sick individ infects its neighbours
        this.ps = p_s;
        //the lifetime of the disease for each individ
        this.min = maxx;
        this.max = minn;
        //probability that an individ X die iff X is sick
        this.ps_x = pxx;
        //number of sick individs and their position
        this.Ns = antal_sjuka;
        this.posX = (int)(this.N/2);
        this.posY = (int)(this.N/2);
        //*********************************************************
```

```
        this.coord = new ArrayList<Object>();
        this.pop = new Individ[NN][NN];
        int x = 0, y = 1;
        for(int i = 0, j = 1; j < dessplacering.length; i++,j++) {
            int q = dessplacering[i];
            int  k = dessplacering[j];
            pop[q][k] = new Individ(" " + q + "" + k, q, k, true);
        }
        for(int i = 0; i < NN; i++)
        {
            for(int j = 0; j < NN; j++)
            {
                if(pop[i][j] == null) {
                    pop[i][j] = new Individ(" " + i + "" + j, i, j);
                }
            }
        }
        //******* initialize the data output parameter *************
        SM = 0;
        X = 0;
        F=0;
        S=antal_sjuka;
        AS=0;
        AX=0;
        //*************** end of initialization **********************
        epidemi = (int)((this.N*this.N)/2);
        this.isEpidemi = false;
    }
    public int [] getDataOutput()
    {
        int [] bark = {this.SM,this.X,this.F,this.S,this.AS,this.AX};
        return bark;
    }
    public String getDataOutputString()
    {
        StringBuilder br = new StringBuilder();
        br.append(" Smittade per day " + this.SM + " ," +
NEW_LINE_SEPARATOR);
        br.append(" Döda per day " + this.X + " ,"+ NEW_LINE_SEPARATOR);
        br.append(" Tillfriskna per day " + this.F + " ,"+
NEW_LINE_SEPARATOR);
        br.append(" Sjuka per day " + this.S + " ,"+ NEW_LINE_SEPARATOR);
/*        br.append(" Accumelerad Sjuka  " + this.AS + " ,"+
NEW_LINE_SEPARATOR);
        br.append(" Accumelerad döda  " + this.AX + " ,"+
NEW_LINE_SEPARATOR);*/
        br.append(" Totalt tillfriskna  " + this.TF + " ,"+
NEW_LINE_SEPARATOR);
        br.append(" Friska  " + this.Frisk + " ,"+ NEW_LINE_SEPARATOR);

        return br.toString();
    }

    public void getDataOutputString(int d)
    {
        try
        {
            String tr = "Dataoutput" + d + ".csv";
            this.fileWriter = new FileWriter(tr);
            //Write the CSV file header
```

```
            this.fileWriter.append("**************** DATA OUTPUT
******************");

            //Add a new line separator after the header
            this.fileWriter.append(NEW_LINE_SEPARATOR);

            //Write a new student object list to the CSV file
            this.fileWriter.append(" Smittade per day " + this.SM);
            this.fileWriter.append(NEW_LINE_SEPARATOR);
            this.fileWriter.append(" Döda per day " + this.X );
            this.fileWriter.append(NEW_LINE_SEPARATOR);
            this.fileWriter.append(" Tillfriskna per day " + this.F );
            this.fileWriter.append(NEW_LINE_SEPARATOR);
            this.fileWriter.append(" Sjuka per day " + this.S );
            this.fileWriter.append(NEW_LINE_SEPARATOR);
/*            this.fileWriter.append(" Accumelerad Sjuka  " + this.AS);
            this.fileWriter.append(NEW_LINE_SEPARATOR);
            this.fileWriter.append(" Accumelerad döda  " + this.AX);
            this.fileWriter.append(NEW_LINE_SEPARATOR);*/
            this.fileWriter.append(" Totalt tillfriskna  " + this.TF);
            this.fileWriter.append(NEW_LINE_SEPARATOR);
            this.fileWriter.append(" Friska  " + this.Frisk);
            this.fileWriter.append(NEW_LINE_SEPARATOR);

            System.out.println("CSV file was created successfully !!!");

        } catch (Exception e) {
            System.out.println("Error in CsvFileWriter !!!");
            e.printStackTrace();
        } finally {

            try {
                fileWriter.flush();
                fileWriter.close();
            } catch (IOException e) {
                System.out.println("Error while flushing/closing fileWriter
!!!");
                e.printStackTrace();
            }
        }
    }
    public ArrayList<int[]> getNear(int x1, int y1)
    {
        return neighbours(x1,y1,this.pop);
    }

    public ArrayList<int[]> neighbours(int posX, int posY, Individ [][] P)
    {
        int dim = P[0].length;
        pos point = new pos(posX,posY);
        ArrayList<Object> cd = new ArrayList<Object>();
        ArrayList<int[]> par = new ArrayList<int[]>();
        pos [] back;
        //get the situation and postion of object individ
        // around the individ in index posX and posY
        int sit = situation(posX,posY,dim);
        if((sit >= 0 ) && (sit <= 3)) {
            //case 1
            //back = new int[6];
            switch (sit)
            {
```

```
            case 0 :
                par.add(point.getRight());
                par.add(point.getQ4());
                par.add(point.getDown());
                break;
            case 1 :
                par.add(point.getLeft());
                par.add(point.getQ3());
                par.add(point.getDown());
                break;
            case 2 :
                par.add(point.getUp());
                par.add(point.getQ1());
                par.add(point.getRight());
                break;
            case 3 :
                par.add(point.getUp());
                par.add(point.getQ2());
                par.add(point.getLeft());
                break;
        }
    }
    else if((sit >= 4 ) && (sit <= 7)) {
        //case 2 or 1
        //back = new int[12];
        switch (sit)
        {
            case 4 :
                par.add(point.getQ3());
                par.add(point.getLeft());
                par.add(point.getRight());
                par.add(point.getQ4());
                par.add(point.getDown());
                break;
            case 5 :
                par.add(point.getLeft());
                par.add(point.getQ1());
                par.add(point.getUp());
                par.add(point.getRight());
                par.add(point.getQ2());
                break;
            case 6 :
                par.add(point.getUp());
                par.add(point.getQ1());
                par.add(point.getRight());
                par.add(point.getQ4());
                par.add(point.getDown());
                break;
            case 7 :
                par.add(point.getUp());
                par.add(point.getQ2());
                par.add(point.getLeft());
                par.add(point.getQ3());
                par.add(point.getDown());
                break;
        }
    }
    else {
        //back = new int[16];
        par.add(point.getUp());
        par.add(point.getRight());
```

```
            par.add(point.getDown());
            par.add(point.getLeft());
            par.add(point.getQ1());
            par.add(point.getQ2());
            par.add(point.getQ3());
            par.add(point.getQ4());
        }

        return par;
    }

    public int situation(int x, int y, int n)
    {
        int c = 0;
        //case 1 4 cases
        if( x == 0 && y == 0)
            return 0;
        else if(x == 0 && y == n -1)
            return 1;
        else if(x == n - 1 && y == 0)
            return 2;
        else if (x ==  n - 1 && y == n - 1)
            return 3;

        //case 2 2 cases
        if( x == 0)
            return 4;
        else if(x == n - 1)
            return 5;

        if( y == 0)
            return 6;
        else if(y == n - 1)
            return 7;
        return -1;
    }

    public void display()
    {
        for(int i = 0; i < this.N; i++)
        {
            for(int j = 0; j < this.N; j++)
            {

                System.out.print(" " + pop[i][j]);
            }
            System.out.println(" ");
        }

    }
    //select a.size() * random
    public ArrayList<int[]> random_choose(ArrayList<int[]> univ, double
prob)
    {
        ArrayList<int[]> par = new ArrayList<int[]>();
        //choose no more than this number of element
        int limit = (int)(prob*univ.size());
        boolean [] check = new boolean[univ.size()];
        int t,k;
        int count = 0;
        for(int i = 0; (par.size() <= limit); i++ )
```

```
        {
            t = (int)(univ.size() * Math.random());
            if(count < limit)
            {
                if(check[t] == false) {
                    check[t] = true;
                    par.add(univ.get(t));
                    count++;
                }
            }
            else if(count == limit) {
                break;
            }
            else
                break;
        }
        return par;
    }

    //infect will be called from another function
    // every call will constituetes an iteration and
    // represent one day in our model
    public void infect()
    {
        for(int i = 0; i < this.N; i++)
        {
            for(int j = 0; j < this.N; j++)
            {
                if((this.pop[i][j].isInfect() && this.pop[i][j].isLive())
&& (this.pop[i][j].isImmun() == false && (this.pop[i][j].getNewly() == 0))
)
                {
                    //********************* handle neighbours around this
individ *********************************
                    ArrayList<int[]> ind_toInfect =
neighbours(i,j,this.pop);
                    ArrayList<int[]> ind_Infect =
random_choose(ind_toInfect,this.ps);
                    //to add check if p is already infected
                    for(int [] p : ind_Infect) {
                        if(this.pop[p[0]][p[1]].isInfect() == false)
                        {
                            this.pop[p[0]][p[1]].setInfect(true);
                        }

                    }

                    //other cases probability that X dies iff X is sick
                    // in the same loop get neigbours P
                    // in P random choose K individs with prob p_X
                    // and for each individ  p in K if p is infected
                    // set the p individ has dead setIslive = false;
                    ArrayList<int[]> ind_die =
random_choose(ind_toInfect,this.ps_x);
                    for(int [] p : ind_die) {
                        if(this.pop[p[0]][p[1]].isInfect()) {
                            this.pop[p[0]][p[1]].setIslive(false);
                            //augment the number of dead individ
                            //this.X++;
                        }
                    }
```

```
                    //************************************** end
*********************************************
                    this.pop[i][j].days_infection();
                    //********************* handle this individ in index
*************************************
                    //handle different cases min och max set immun
                    if(this.pop[i][j].nb_days_sickness >
this.pop[i][j].getMax())
                    {
                        this.pop[i][j].setImmun(true);
                        this.pop[i][j].setInfectImmun(false);
                }
                if(this.S >= epidemi)
                    this.isEpidemi = true;
            }
        }

        //update for the next day
        this.S = 0;
        this.SM = 0;
        this.F = 0;
        this.X = 0;
        this.TF = 0;
        this.Frisk = 0;
        for(int i = 0; i < this.N; i++)
        {
            for(int j = 0; j < this.N; j++)
            {
                if(this.pop[i][j].isInfect() && this.pop[i][j].isLive())
                    this.S = this.S + 1;
                if((!this.pop[i][j].isInfect()) && this.pop[i][j].isLive())
                    this.Frisk = this.Frisk + 1;

                if(this.pop[i][j].isInfect()) {
                    //peuples[i][j].days_infection();
                    this.pop[i][j].setNewly(0);

                    if(this.pop[i][j].nb_days_sickness == 1 &&
this.pop[i][j].isLive())
                    {
                        this.SM = this.SM + 1;
                    }
                    if(this.pop[i][j].isLive() == false)
                    {
                        this.X = this.X + 1;
                    }
                    if(this.pop[i][j].isImmun())
                    {
                        this.F = this.F + 1;
                    }
                }
                if(this.pop[i][j].isImmun())
                    this.TF = this.TF + 1;
            }
        }
    }

    public int stop()
    {
```

```
        int bk = (this.N) * (this.N);
        int count = 0;
        for(int i = 0; i < this.N; i++)
        {
            for(int j = 0; j < this.N; j++)
            {
                if(!this.pop[i][j].isInfect()) {
                    count = count + 1;
                }
            }
        }
        return count;
    }


    public void simul()
    {
        //call infect N times N = days
        int i;
        int count = 0;
        int bk = (this.N) * (this.N) - this.X;
        for(i = 1; count != bk ; i++)
        {
            System.out.println("************************** day " + i + "
****************************** ");
            infect();
            display();
            System.out.println();
            this.AS = this.AS + this.S;
            this.AX = this.AX + this.X;
            count = stop();
            bk = (this.N) * (this.N) - this.X;
            //System.out.println( "Count ==== " + count);
            System.out.println( "Data : \n" + getDataOutputString());
            getDataOutputString(i);
            System.out.println(" ************************  day " + i + "
finished ************************");
        }


        System.out.println("It is an epidemi ? " + this.isEpidemi);
    }

    public void simulate(int N)
    {
        //call infect N times N = days
        int i;
        for(i = 1; i  < N + 1; i++)
        {
            System.out.println("************************** day " + i + "
****************************** ");
            infect();
            display();
            System.out.println();
            this.AS = this.AS + this.S;
            this.AX = this.AX + this.X;

            System.out.println( "Data : \n" + getDataOutputString());
            System.out.println(" ************************  day " + i + "
finished ************************");
        }
```

```
        getDataOutputString(i);

        System.out.println("It is an epidemi ? " + this.isEpidemi);
    }


    public void simulate()
    {
        System.out.println("************************* day " + 0 + "
****************************** ");
        display();
        System.out.println( "Data : \n" + getDataOutputString());
        System.out.println(" **********************  day " + 0 + "
finished ************************");
        simulation();

    }
    public void simulation()
    {
        //call infect N times N = days
        int i;
        for(i = 1; i  < this.N + 1; i++)
        {
            System.out.println("************************* day " + i + "
****************************** ");
            infect();
            display();
            System.out.println();
            this.AS = this.AS + this.S;
            this.AX = this.AX + this.X;

            System.out.println( "Data : \n" + getDataOutputString());
            System.out.println(" **********************  day " + i + "
finished ************************");
        }
        getDataOutputString(i);

        System.out.println("It is an epidemi ? " + this.isEpidemi);
    }
}
```

Main

```
public class Main {

    public static void main(String[] args)
    {
   // write your code here
        population pop = new population();
        pop.simul();
/*        ArrayList<int[]> p = pop.getNear(2,2);
        ArrayList<int[]> q = pop.random_choose(p,0.5);
        for(int [] c : q)
            System.out.println(Arrays.toString(c));*/
```

```
    }
}
```

BILAGA B

********** day 1 **********

[inf = false , sick_d = 0, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv true]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 2, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true]

[inf = true , sick_d = 1, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true]

Data :

Smittade per day 2 ,

Döda per day 0 ,

Tillfriskna per day 0 ,

Sjuka per day 3 ,

Totalt tillfriskna 0 ,

Friska 6 ,

********** day 1 finished **********

********** day 2 **********

[inf = false , sick_d = 0, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 3, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv true]

[inf = true , sick_d = 2, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true]

Data :

Smittade per day 1 ,

Döda per day 1 ,

Tillfriskna per day 0 ,

Sjuka per day 3 ,

Totalt tillfriskna 0 ,

Friska 5 ,

********** day 2 finished **********

********** day 3 **********

[inf = true , sick_d = 1, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 4, im = false, isliv true] [inf = true , sick_d = 2, im = false, isliv true]

[inf = true , sick_d = 3, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true]

Data :

Smittade per day 1 ,

Döda per day 2 ,

Tillfriskna per day 0 ,

Sjuka per day 4 ,

Totalt tillfriskna 0 ,

Friska 3 ,

********** day 3 finished **********

********** day 4 **********

[inf = true , sick_d = 2, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = true , sick_d = 3, im = false, isliv true]

[inf = true , sick_d = 4, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv true] [inf = false , sick_d = 0, im = false, isliv true]

Data :

Smittade per day 1 ,

Döda per day 3 ,

Tillfriskna per day 0 ,

Sjuka per day 4 ,

Totalt tillfriskna 0 ,

Friska 2 ,

********** day 4 finished **********

********** day 5 **********

[inf = true , sick_d = 3, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = true , sick_d = 4, im = false, isliv true]

[inf = true , sick_d = 5, im = false, isliv true] [inf = true , sick_d = 2, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv true]

Data :

Smittade per day 1 ,

Döda per day 3 ,

Tillfriskna per day 0 ,

Sjuka per day 5 ,

Totalt tillfriskna 0 ,

Friska 1 ,

********** day 5 finished **********

********** day 6 **********

[inf = true , sick_d = 4, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = true , sick_d = 5, im = false, isliv true]

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 2, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv true]

Data :

Smittade per day 0 ,

Döda per day 4 ,

Tillfriskna per day 0 ,

Sjuka per day 3 ,

Totalt tillfriskna 1 ,

Friska 2 ,

********** day 6 finished **********

********** day 7 **********

[inf = true , sick_d = 5, im = false, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = false , sick_d = 6, im = true, isliv true]

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 2, im = false, isliv false] [inf = true , sick_d = 3, im = false, isliv true]

Data :

Smittade per day 0 ,

Döda per day 4 ,

Tillfriskna per day 0 ,

Sjuka per day 2 ,

Totalt tillfriskna 2 ,

Friska 3 ,

********** day 7 finished **********

********** day 8 **********

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = false , sick_d = 6, im = true, isliv true]

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 2, im = false, isliv false] [inf = true , sick_d = 4, im = false, isliv true]

Data :

Smittade per day 0 ,

Döda per day 4 ,

Tillfriskna per day 0 ,

Sjuka per day 1 ,

Totalt tillfriskna 3 ,

Friska 4 ,

********** day 8 finished **********

********** day 9 **********

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = false , sick_d = 6, im = true, isliv true]

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 2, im = false, isliv false] [inf = true , sick_d = 5, im = false, isliv true]

Data :

Smittade per day 0 ,

Döda per day 4 ,

Tillfriskna per day 0 ,

Sjuka per day 1 ,

Totalt tillfriskna 3 ,

Friska 4 ,

********** day 9 finished **********

********** day 10 **********

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 1, im = false, isliv false] [inf = true , sick_d = 2, im = false, isliv false]

[inf = false , sick_d = 0, im = false, isliv true] [inf = true , sick_d = 5, im = false, isliv false] [inf = false , sick_d = 6, im = true, isliv true]

[inf = false , sick_d = 6, im = true, isliv true] [inf = true , sick_d = 2, im = false, isliv false] [inf = false , sick_d = 6, im = true, isliv true]

Data :

Smittade per day 0 ,

Döda per day 4 ,

Tillfriskna per day 0 ,

Sjuka per day 0 ,

Totalt tillfriskna 4 ,

Friska 5 ,

********** day 10 finished **********

It is an epidemi ? true

Process finished with exit code 0

BILAGA C

Rådata finnes på adressen : https://github.com/benedictmulongo/Simulation .

BILAGA D

Användaren skall kunna sätta värden för följande parametrar:

- Storlek på populationen (N)
- Smittsannolikhet (per dygn) att en sjuk individ smittar en frisk granne som inte är immun (S)
- Längd på den tid man är sjuk, rektangulärfördelat i ett intervall [minDygn, maxDygn]
- Sannolikhet att en individ avlider per dygn individen är sjuk (L)
- Hur många individer som är sjuka initialt och var de är placerade

Utdata från simuleringen skall vara:

- Antal smittade varje dygn
- Antal som avlidit varje dygn
- Antal individer som tillfrisknar varje dygn
- Antalet sjuka varje dygn
- Ackumulerat antal smittade varje dygn
- Ackumulerat antal avlidna varje dygn

Indata var lätt att ta fram via använde följande datatyper i klassen population

```
int N;
//probability that a sick individ infects its neighbours
double ps;
//the lifetime of the disease for each individ
int min;
int max;
//probability that an individ X die iff X is sick
double ps_x;
//number of sick individs and their position
int Ns;
int posX;
int posY;
Individ [][] pop;
//*******************************************************
int SM,X,F,S,AS,AX,TF,Frisk;
```

Dessutom skapades två konstruktörer :

- `population()`
- `population(int maxx,int minn, int p_s,int NN,int pxx, int antal_sjuka, int [] dessplacering)`

Den första har default variabler som gå att ändra och den andra för alla sina attributet från anropet som argument.
Utifrån vår analys kom vi fram till följande steg:

1. Bilda en klass av individer med dess attribut
2. Bilda en klass population och initierar vid anropet av population() en 2D matris med objekt individer
3. Initierar alla individer med default värde
4. Initiera variabel ps
5. Initiera variabel px
6. Definiera två globala variabler max och min
7. Skapa en funktion neigbours som returnerar grannskapet för varje individ
8. Implementera en funktion random_choose som väljer ut individer i grannskapet med sannolikhet px eller ps beroende på vilken del av koden som körs
9. Implementera en funktion infect() som använder sig av en iteration igenom population för att ändra värde på varje individer med hjälp av steg 7 och 8

10. Individer kan inte smitta varandra samma dag som de själv blev smittade
11. Representerar en dag som ett anrop till funktion infect()
12. Funktion infect() innehåller all logik för simulering
13. Simulera genom att kalla funktion infect() tills sjukdomen eventuellt dör ut.

BILAGA E

Medelvärdet

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

I vårt fall blev det 10.6

$$s = \sqrt{\frac{(\sum_{i=1}^{n} X_i^2) - \frac{(\sum_{i=1}^{n} X_i)^2}{n}}{n-1}}$$

Standar avvikelse

$$s = \sqrt{\frac{(\sum_{i=1}^{n} X_i^2) - \frac{(\sum_{i=1}^{n} X_i)^2}{n}}{n-1}}$$

I vår t fall är s = 2.95

Konfidensintervall

$$I = (\overline{X} - \frac{t_{\alpha/2,n-1} s}{\sqrt{n}}, \overline{X} + \frac{t_{\alpha/2,n-1} s}{\sqrt{n}})$$

I vårt fall I = [8.3,12.5].

Resten av beräkningar och tabeller har gjort med hjälp av Microsoft Excel. Det hittas på https://github.com/benedictmulongo/Simulation.