

# Datorlaboration 8

Måns Magnusson

31 januari 2015

---

## Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändra grupper.
  - En av ska vara **navigatör** och den andra **programmerar**. Navigatörens ansvar är att ha ett helhetsperspektiv över koden. Byt position var 30:e minut.
  - Det är tillåtet att diskutera med andra grupper, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
  - Utgå från laborationsfilen som går att ladda ned [här](#)
  - Laborationen består av två delar:
    - Datorlaborationen
    - Inlämningsuppgifter
  - I laborationen finns det extrauppgifter markerade med \*. Dessa kan hoppas över.
  - Deadline för labben framgår på [kurshemsidan](#)
-

# Innehåll

<b>I Datorlaboration</b>	<b>3</b>
<b>1 Introduktion till ggplot2</b>	<b>4</b>
1.1 Grunden i ggplot2 . . . . .	4
1.1.1 Skapa en ggplot (linje eller scatter) . . . . .	5
1.1.2 Enklare modifikationer av ett ggplot-objekt . . . . .	6
1.1.3 Barplot, histogram och boxplot . . . . .	6
1.1.4 Histogram . . . . .	7
1.1.5 Boxplot . . . . .	8
1.2 Grafiska teman/profiler . . . . .	9
<b>2 Introduktion till spatiala data och visualisering</b>	<b>11</b>
2.1 Förberedelser . . . . .	11
2.1.1 Rpaket . . . . .	11
2.1.2 Data . . . . .	11
2.2 Läsa in shape-filer och spatiala datastrukturer . . . . .	12
2.2.1 Statistik på karta . . . . .	13
2.2.2 Kombinera polygoner till större geografiska områden. . . . .	14
2.2.3 Skapa delmängder av spatiala objekt . . . . .	15
2.2.4 * Extraproblem: Statistik över Östergötland . . . . .	15
2.3 Skapa kartor med ggmap . . . . .	16
2.4 Kombinera spatiala data med olika projektioner/geografiska referenssystem . . . . .	19
<b>II Inlämningsuppgifter</b>	<b>23</b>
<b>3 Inlämningsuppgifter</b>	<b>25</b>
3.1 dynamic.linear.system() . . . . .	25
3.2 Miniprojektet del II . . . . .	26

**Del I**

**Datorlaboration**

# Kapitel 1

## Introduktion till ggplot2

Paketet `ggplot2` skiljer sig från den grundläggande grafikfunktionaliteten som finns implementerat i R. Paketet bygger på vad som brukar kallas “The grammar of graphics” (därav `gg` i `ggplot2`) och är ett försök till ett formellt språk för att uttrycka hur en visualisering ska se ut. Mer teori bakom denna grammatik går att finna i [2, 1] och är grunden bakom exempelvis SPSS grafiksystem. Genom att ha en grundläggande förståelse för denna grammatik kan vi enkelt och snabbt skapa mycket komplicerade visualiseringar.

I R:s basgrafiksysteem kunde man se grafikfunktionaliteten lite som ett papper vi ritar på. Vi ritar initialt upp vår graf och kan sedan lägga till/rita ”ovanpå” det befintliga pappret. `ggplot` är annorlunda. Med `ggplot` skapar vi ett grafikobjekt och vi kan lägga till bit för bit av grafen för att när vi sedan är klar med vår graf visualisera den. Det gör det enklare att bygga upp komplicerade grafer utan att behöva använda särskilt mycket kod.

### 1.1 Grunden i ggplot2

Till skillnad från basgrafen utgår `ggplot` alltid från en `data.frame`. Baserat på denna `data.frame` skapas sedan grafen med två huvudsakliga komponenter:

- `aes` (aesthetic) som handlar om utseendet på grafen, färger, former m.m.
- `geom` (geometrics) som beskriver vilken typ av graf vi vill ha (bar, line, points)

Vi lägger sedan till dessa komponenter till vår graf och `data.frame`.

När det gäller de olika geometriska argumenten, d.v.s. de olika typer av grafer som går att skapa, finns det ett mycket stor antal vi kan använda oss av. Några exempel är:

geom	Beskrivning
<code>geom_point</code>	Scatterplot
<code>geom_line</code>	Line graph
<code>geom_bar</code>	Barplot
<code>geom_boxplot</code>	Boxplot
<code>geom_histogram</code>	Histogram

Exakt hur dessa geometriska figurer ska se ut styrs sedan med `aes`. Nedan finns några exempel:

aes	Beskrivning
<code>x</code>	x-axel
<code>y</code>	y-axel
<code>size</code>	storlek
<code>col</code>	färg
<code>shape</code>	form

De enskilda geometriska figurerna kan i sin tur ha ett antal olika aesthetics. Nedan finns lite exempel.

geom	Specifika aesthetics
<code>geom_points</code>	point <code>shape</code> , point <code>size</code>
<code>geom_line</code>	<code>linetype</code> , <code>line size</code>
<code>geom_bar</code>	<code>y min</code> , <code>y max</code> , <code>fill color</code> , <code>outline color</code>

Med dessa verktyg har vi en grund för att bygga upp ett mycket stort antal visualiseringar.

### 1.1.1 Skapa en ggplot (linje eller scatter)

1. Vi börjar med att läsa in the datamaterialet `Nile`.

```
library(ggplot2)

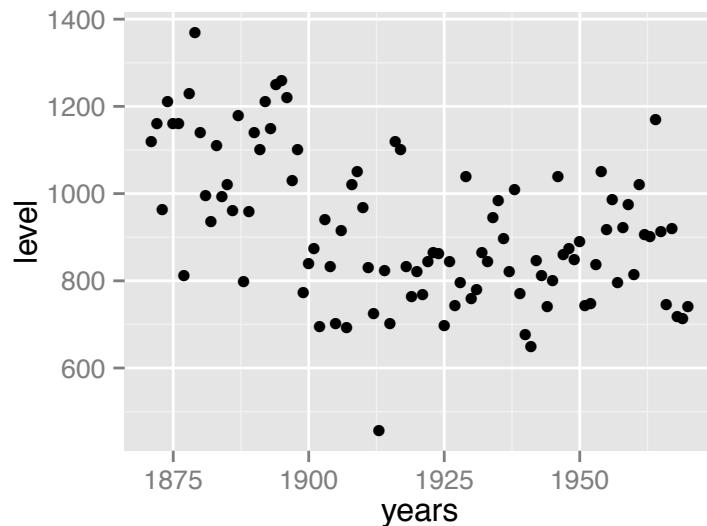
Loading required package: methods

data(Nile)
Nile <- as.data.frame(Nile)
colnames(Nile) <- "level"
Nile$years <- 1871:1970
```

2. För att skapa en `ggplot` börjar vi med att skapa grunden för plotten med funktionen `ggplot()`. Nedan är ett exempel på att skapa en `ggplot` med `Nile`, sedan lägger vi till att `x` ska utgöras av variabeln `years` och `level`. Sedan lägger vi till att plotten ska utgöras av punkter. Vi sparar grafen som variabeln `p`. För att skapa grafen tittar vi bara på `p`:

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_point()
p

Don't know how to automatically pick scale for object of type ts. Defaulting to continuous
```



3. Vill vi ändra till en linjegraf (vilket känns bättre) här byter vi bara ut geometrin:

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line()
```

4. Vill vi lägga till både punkter och linjer i samma graf kan vi bara ta `p` och lägga till punkter. Här blir det tydligt hur vi i `ggplot` lägger till lager på lager och sedan producerar en visualisering:

```
p <- p + geom_point()
```

5. På samma sätt kan vi också lägga till rubriker och axelettiketter:

```
p <- p + xlab("Years") + ylab("Water level") + ggtitle("Nile series")
Nile$period <- "- 1900" Nile$period[Nile$years >= 1900] <- "1900 - 1945" Nile$period[Nile$years
```

### 1.1.2 Enklare modifikationer av ett ggplot-objekt

1. Vill vi ändra färg och form på olika delar i en graf behöver vi ange exakt var dessa förändringar ska ske.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line(color="red", size=3) + geom_point(col
```

2. Om vi nu vill förtydliga vissa delar av grafen med olika färger eller använder vi `aes` i den del av grafen vi vill ändra. Först ska vi skapa en ny faktorvariabel vi vill visualisera.

```
Nile$period <- "- 1900"
Nile$period[Nile$years >= 1900] <- "1900 - 1945"
Nile$period[Nile$years > 1945] <- "1945 + "
Nile$period <- as.factor(Nile$period)
```

3. Vill vi nu exempelvis lyfta in visualiseringen i linjerna måste vi lägga `aes` där.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line(aes(color=period)) + geom_point()
```

4. Vill vi istället modifiera punkterna lägger vi till det i `geom_point()`.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line() + geom_point(aes(color=period))
```

5. Vill vi lägga det i hela grafen kan vi lägga till färgen i den huvudsakliga styrningen av aesthetics i grafen.

```
p <- ggplot(data=Nile) + aes(x=years, y=level, color=period) + geom_line() + geom_point()
```

6. Baserat på graferna ovan pröva att göra följande förändringar:

- (a) Ändra typ av linje i grafen [**Tips!** `linetype`]
- (b) Ändra typ av punkter i grafen [**Tips!** `shape`]
- (c) Gör punkterna transparaenta [**Tips!** `alpha`]

### 1.1.3 Barplot, histogram och boxplot

För att pröva dessa diagram använder vi oss av datamaterialet mtcars. Vi börjar med att läsa in datamaterialet mtcars. För att få mer information om detta datamaterial, använd `?mtcars`. Vi gör också om

```

data(mtcars)
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)

```

Till skillnad från basgrafen använder vi inte olika funktioner för olika plottar utan vi använder bara olika `geoms`.

- Vill vi exempelvis skapa ett stapeldiagram anger vi bara en axel och ett annat geom, men i övrigt är det inge större skillnad mot en linjegraf:

```
p <- ggplot(data=mtcars) + aes(x=cyl) + geom_bar()
```

- Vi kan också enkelt lägga till funktionen `coord_flip()` för att skapa ett liggande stapeldiagram istället för ett stående.

```
p + coord_flip()
```

- Skillnaden ligger i att det finns lite andra aesthetics för stapeldiagram än för övriga diagram som `fill`.

```
p <- ggplot(data=mtcars) + aes(x=cyl) + geom_bar(fill="darkblue", colour="red")
```

- För att skapa stapeldiagram med flera grupper behöver vi dels lägga till en till variabel som indikerar att vi vill ha ex. olika färger för olika grupper samt ange hur dessa diagram ska se ut. Pröva exemplen nedan:

```

p <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="stack")
p <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="dodge")
p + scale_fill_discrete(name="Testa\nDetta")

```

#### 1.1.4 Histogram

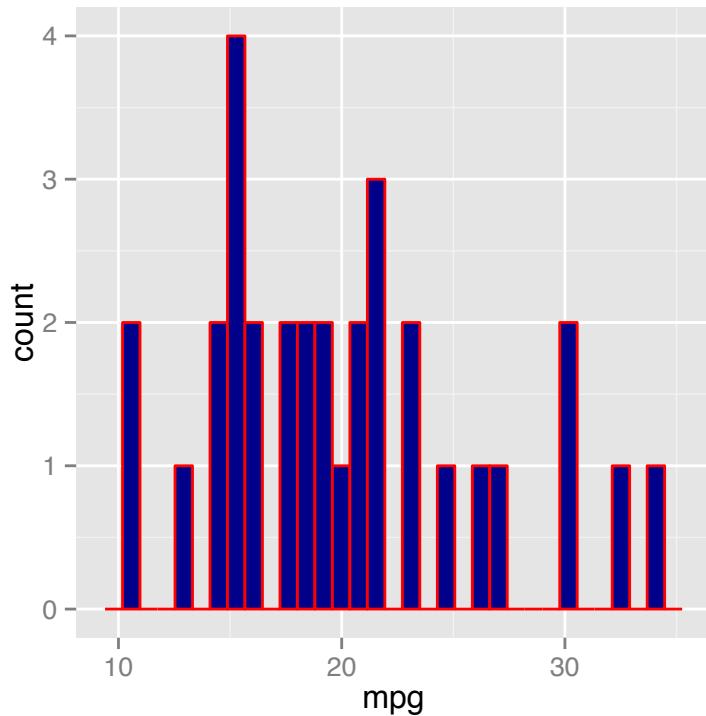
- Den egentliga skillnaden mellan ett stapeldiagram och ett histogram är bara huruvida variabeln är kontinuerlig eller inte. Detta gör att för att skapa ett histogram gör vi på exakt samma sätt, men vi använder oss av en kontinuerlig variabel:

```

p <- ggplot(data=mtcars) + aes(x=mpg) + geom_bar(fill="darkblue", colour="red")
p

```

*stat\_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.*



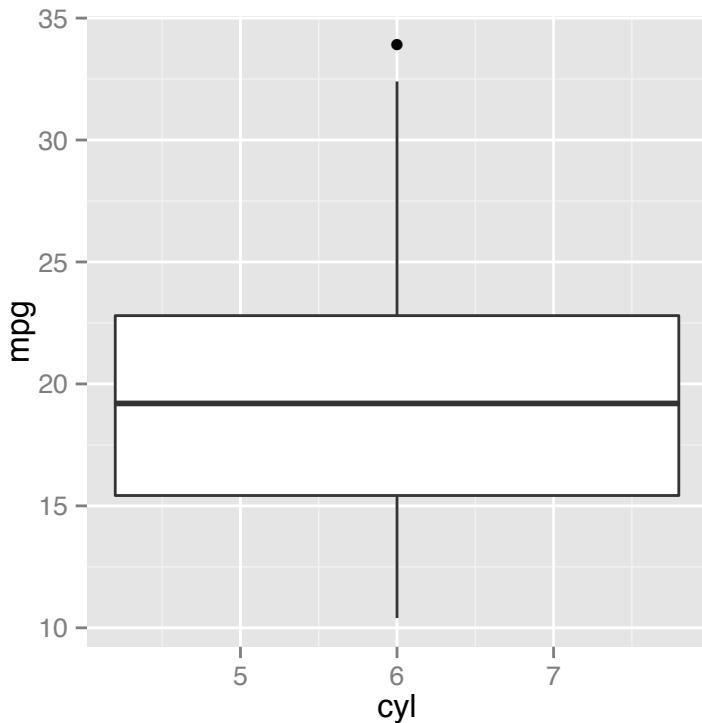
2. Sättet ovan är ett sätt att skapa ett histogram. Vi kan också använda den specialgjorda geometriska funktionen `geom_histogram()` om vi vill kunna hantera histogram enklare.

```
p <- ggplot(data=mtcars) + aes(x=mpg) + geom_histogram(fill="darkblue", colour="red", binwidth=1)
```

### 1.1.5 Boxplot

1. Boxplottar är egentligen en kombination av kontinuerliga variabler. Precis som tidigare inleder vi skapa en `ggplot` med ett datamaterial och definierar vilka variabler vi vill använda.

```
p <- ggplot(data=mtcars) + aes(x=cyl, y=mpg) + geom_boxplot()
p
```



- Vill vi sedan göra förändringar kan vi lägga till det till och från.

```
p + coord_flip() + xlab("X") + ggtitle("Hejsan")
```

## 1.2 Grafiska teman/profiler

En av de stora fördelarna med att `ggplot` skiljer ut själva plotten från utseendet är att det är enkelt att skapa strukturer för olika delar av en graf som vi vill använda flera gånger. En av de bästa exemplen på detta är teman i `ggplot`. Ett tema är en uppsättning med inställningar för en grafisk profil som vi vill använda i `ggplot2`.

Den stora fördelen är att har vi väl skapat ett tema (vilket kan ta lite tid) kan temat läggas till mycket enkelt till samtliga grafer. Detta underlättar kopplingen mellan exempelvis grafiska profiler och de grafer som produceras, vilket gör att `ggplot2` är mycket populärt i företag och organisationer. Ett exempel på rapport som använder `ggplot2` genomgående är Pensionsmyndighetens [Orange rapport].

- Med `ggplot2` kommer en del teman förinstallerade och precis som allt annat i R är det enkelt att bara lägga till den grafiska profilen efter att vi skapat en graf.

```
p <- ggplot(data=Nile) + aes(x=years, y=level, color=period) + geom_line() + geom_point()
p <- p + theme_bw()
```

- Pröva på liknande sätt följande teman: `theme_grey()`, `theme_classic()`.
- Ett tema i `ggplot2` är bara en funktion, så det är enkelt att titta på hur temat ser ut och sedan utgå från ett befintligt tema för att anpassa det till det utseende vi själva vill ha. Sedan kan detta tema enkelt spridas till alla som arbetar med visualisering med `ggplot`.

theme\_bw

4. Att ändra den grafiska profilen innebär då bara att ändra denna temafunktion (även om det kan innehålla en del jobb).

## Kapitel 2

# Introduktion till spatiala data och visualisering

Spatiala data , geografiska data eller geotaggad data är data som innehåller information om lokalitet eller plats. Det är en datastruktur som blir allt mer vanligt i och med att allt mer data digitaliseras. Det kan vara data om positioner (från exempelvis mobiltelefoner eller twittermeddelanden), det kan vara kartor för att skapa geografiska referenser till exempelvis data på kommunal nivå och det kan vara eller det kan vara spatial visualisering där exempelvis områden med större brottslighet visualiseras.

Samlingsnamnet för denna typ av analyser är spatiala data eller spatial analys. En mycket bra hemsida med fördjupningar finns [[här](#)].

Tidigare har denna typ av analys och databearbetning varit reserverade för personer med särskilt kunnande i geografiska informationsystem eller GIS, men på senare tid har R seglat upp som en allt större konkurrens när det gäller spatial visualisering och analys.

I detta avsnitt kommer vi göra en kortare djupdykning i detta fält. Djupdykningen kommer vara långt ifrån fullständig, men ge en känsla för att det ofta inte behöver vara särskilt svårt att skapa kartvisualiseringar.

### 2.1 Förberedelser

#### 2.1.1 Rpaket

Som ett första steg behöver vi installera en hel del paket som kan hantera och arbeta med spatiala datastrukturer. Vi kommer därför gå igenom den mest grundläggande funktionaliteten i följande R-paket:

Paket	Innehåll
<code>maptools</code>	Verktyg för att använda och läsa in spatiala data
<code>rgeos</code>	Kopplingar från R till GEOS (Geometry engine open source)
<code>rgdal</code>	Verktyg för att konvertera mellan projektioner
<code>ggmap</code>	Spatial visualisering med google maps och opens street map.
<code>sp</code>	Klasser och metoder för spatiala datastrukturer

När det gäller paketen rgeos och rgdal bygger dessa paket dessutom på andra implementationer i språk som Java och C++. Det kan göra det något mer krångligt att installera dessa paket i vissa fall. För att installera dessa paket på Mac OS kan jag rekommendera [[denna](#)] och [[denna](#)] guide för installation.

#### 2.1.2 Data

I denna laboration behöver vi också ladda ned den data vi ska använda oss av. Detta är exempel på svenska data över sveriges kommunindelning i form av shape-fil. Vi laddar ned dessa shapefiler med paketet `downloader`.

```
library(downloader)
# Download and read data
komuner_remote <- "https://raw.githubusercontent.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/Kommun_SCB.zip"
```

```
komuner_local <- "Kommun_SCB.zip"  
download(komuner_remote, destfile = komuner_local)  
unzip(komuner_local)
```

## 2.2 Läsa in shape-filer och spatiala datastrukturer

För att skapa kartor behöver vi en så kallad shapefil. Dessa innehåller geografisk data i vektoriserat format vilket vi kan läsa in och modifiera i R. Vektoriserade geografiska data innebär att vår data representeras som punkter, vektorer eller polygoner. Mer information om shapefiler finns [\[här\]](#) och om vektoriserad GIS-information finns [\[här\]](#).

Shapefiler som är aktuella för just era problem beror på detaljeringsgrad, men församlingsgränser eller kommunindelningsgränser av intresse finns på de flesta kommuner och länsstyrelser. Ofta finns informationen att tillgå gratis som shapefiler. Exempelvis tillhandahåller valmyndigheten samtliga valdistrikts i form av en shapefil.

1. Läs in shapefilen med hjälp av `readShapePoly()` på följande sätt i R.

```
library(maptools)  
  
Loading required package: sp  
Checking rgeos availability: TRUE  
  
swe_municip <- readShapePoly("Kommun_SCB/Kommun_SCB_07.shp")
```

2. Vi har nu läst in shapefilen i R. Vi kan få en snabb beskrivning av våra geografiska data med `summary()`

```
class(swe_municip)  
summary(swe_municip)
```

3. Som ett första steg kan vi också skapa en snabb grafisk plot baserat på vårt spatiala objekt.

```
plot(swe_municip)
```



4. Nu har vi läst in och studerat vår första shapefil!

### 2.2.1 Statistik på karta

Nästa steg är att integrera statistik med vår karta, för att på detta sätt visualisera kommunala data. Med Koden nedan laddar jag ned kommunala data från SCB (antalet mjölkkor per kommun) och städar datat något.

```
library(pxweb)

pxweb: R tools for PX-WEB API.
Copyright (C) 2014 Mans Magnusson, Leo Lahti, Love Hansson
https://github.com/ropengov/pxweb

my_data <-
  get_pxweb_data(url = "http://api.scb.se/0V0104/v1/doris/sv/ssd/J0/J00103/HusdjurK",
                 dims = list(Region = c('*'),
                             Djurslag = c('05'),
                             ContentsCode = c('J00103K2'),
                             Tid = c('2007')),
                 clean = TRUE)
my_data <- my_data[, c(1,5)]
my_data[, 1] <- substr(as.character(my_data$region), 1, 4)
my_data[is.na(my_data[, 2]), 2] <- 0
```

1. När det gäller spatiala data finns vårt dataset (kopplade till varje polygon - i detta fall kommun) som en egen `data.frame` i vårt spatiala objekt. Vi kan komma åt datat med @ på följande sätt:

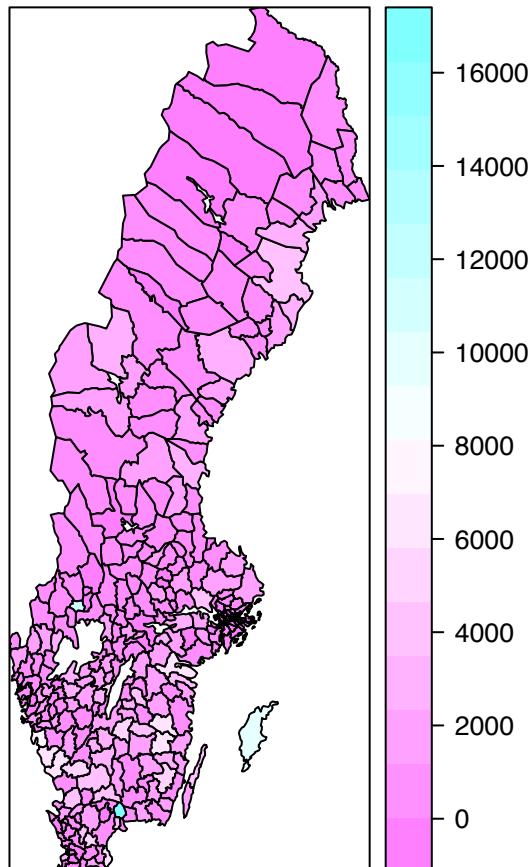
```
head(swe_municip@data)
```

2. Denna `data.frame` kan vi hantera precis som vanliga `data.frames`. Vill vi lägga på data använder vi därför `merge()` och lägger till våra data om kor till respektive kommun.

```
swe_municip@data <- merge(swe_municip@data, my_data, by.x="KNKOD", by.y="region")
```

- Nu har vi lagt till våra estimat och kan använda `spplot()` för att skapa en karta baserad på vår statistik. Pröva följande kod.

```
spplot(swe_municip, "values")
```



### 2.2.2 Kombinera polygoner till större geografiska områden.

Nu har vi arbetat med svenska kommuner. Vill vi arbeta med län går detta att göra genom att läsa in en separat shapefil för län. Vi kan dock kombinera polygoner till större områden direkt i R. I detta fall är det län, men det går att kombinera områden på ett godtyckligt sätt.

- Vi börjar att skapa en variabel som heter `LN` som indikerar vilket län respektive kommun ligger i. Tack vara kommunkoderna är det möjligt att se detta direkt. De två första positionerna anger län.
- Nästa steg blir att slå samman de kommunala polygonerna för att bilda länspolygoner. För detta använder vi `unionSpatialPolygons()`. I detta fall verkar det som några kommuner hamnat i fel län. Detta beror på att länskoden för vissa kommuner blivigt är felaktig.

```
swe_county <- unionSpatialPolygons(SpP = swe_municip, IDs = swe_municip@data$LN)
```

```
Loading required package: rgeos  
rgeos version: 0.3-4, (SVN revision 438)
```

```
GEOS runtime version: 3.4.2-CAPI-1.8.2 r3921
Polygon checking: TRUE
```

3. Vi kan sedan studera resultatet med `plot()` och `summary()`.

```
plot(swe_county)
summary(swe_county)
```

4. När vi lägger ihop olika polygoner försvinner vårt datamaterial i vårt spatiala objekt. För att lägga till nya data behöver vi därför göra detta ”manuellt”. Vi börjar med att aggregera upp våra data till länsnivå.

```
aggr_data <- aggregate(x=swe_municip@data$values, by=list(swe_municip@data$LN), FUN=sum)
names(aggr_data) <- c("LN", "kossor")
```

5. Nästa steg är att lägga till dessa nya aggregerade data till vårt nya spatiala objekt. För att detta ska gå måste rad-id i vår data.frame stämma överens med polygon-id i det spatiala objektet. Därför börjar vi med att lägga till radnamn och därefter använder vi `SpatialPolygonsDataFrame()` för att kombinera data på länsnivå.

```
rownames(aggr_data) <- aggr_data$LN
swe_county <- SpatialPolygonsDataFrame(Sr=swe_county, data=aggr_data)
```

6. Nu kan vi återigen skapa en karta med lite större områden där antalet kor framgår mer tydligt. (Men på grund av kommunkodproblem så har några kommuner hamnat fel.)

```
spplot(swe_county, "kossor")
```

### 2.2.3 Skapa delmängder av spatiala objekt

Som ett sista steg ska vi plocka ut endast delar av en karta för att skapa kartor över mindre delområden. Detta blir extra intressant när vi visualisera data på kartor. Vi utgår än en gång från våra kommuner som spatialt objekt.

1. Vi ska nu istället plocka ut en delmängd av dessa kommuner. Än en gång använder vi länskoderna vi skapat. För att plocka ut ett visst antal kommuner gör vi precis som för att indexera rader i en data.frame i R. För att välja ut lännet ”01” (Stockholms län) används följande kod.

```
spatial_sthlm <- swe_municip[swe_municip@data[['LN']]=="01",]
```

2. Vi kan nu studera dels hur vårt spatiala objekt ser ut med `plot()` samt få en bild av var kossorna bor i Stockholms län med `spplot()`.

### 2.2.4 \* Extraproblem: Statistik över Östergötland

Vi ska nu skapa en karta över Östergötland för en godtycklig statistisk storhet. Ta reda på vilken länskod Östergötland har. Gå in på SCB och hämta ned data på kommunnivå för Östergötlands län. Du kan välja vilken statistik du vill (ex. valresultat finns på kommunal nivå) och skapa en karta över denna variabel för Östergötland.

För att snabbt ladda ned data till R kan du använda paketet `pxweb`.

```
library(pxweb)
my_data <- interactive_pxweb(api="scb", lang="sv")
```

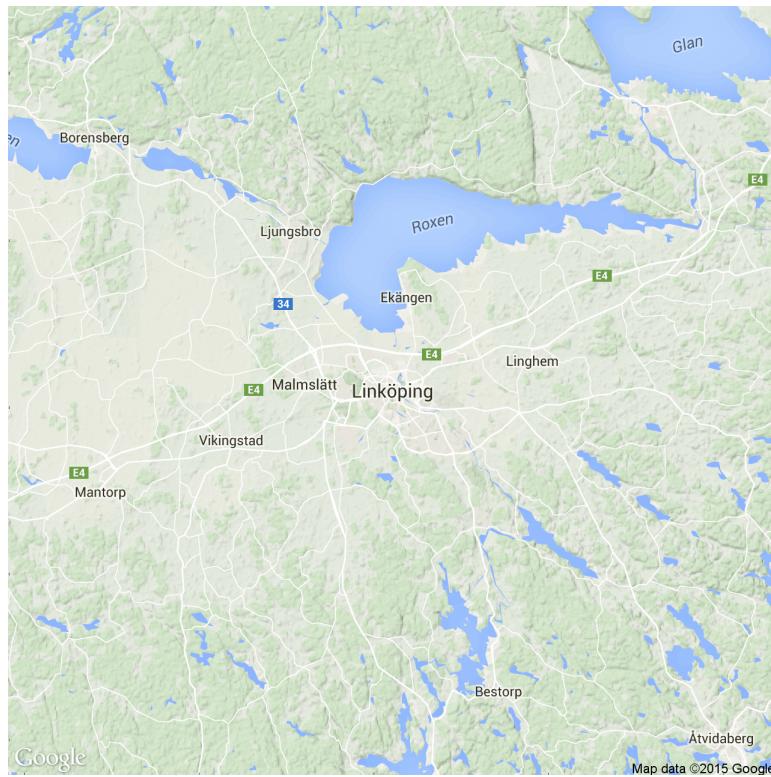
## 2.3 Skapa kartor med ggmap

Vi har hittills arbetat med shapefiler som innehåller spatiala data. Inte sällan vill vi också sätta våra spatiala data i relation till kartor. Särskilt om vi har data som är geotaggade, d.v.s. innehåller koordinater är det av intresse att visualisera dessa på en karta. För att använda kartor i R är det enklaste att använda oss av de kartor som finns öppet tillgängliga på webben, som Google Maps, Open Street Map m.fl. Med paketet **ggmap** kan vi enkelt läsa in en godtycklig kvadratisk karta.

1. Vi börjar med att läsa in paketet **ggmap** och skapar en första karta över Linköping på följande sätt.

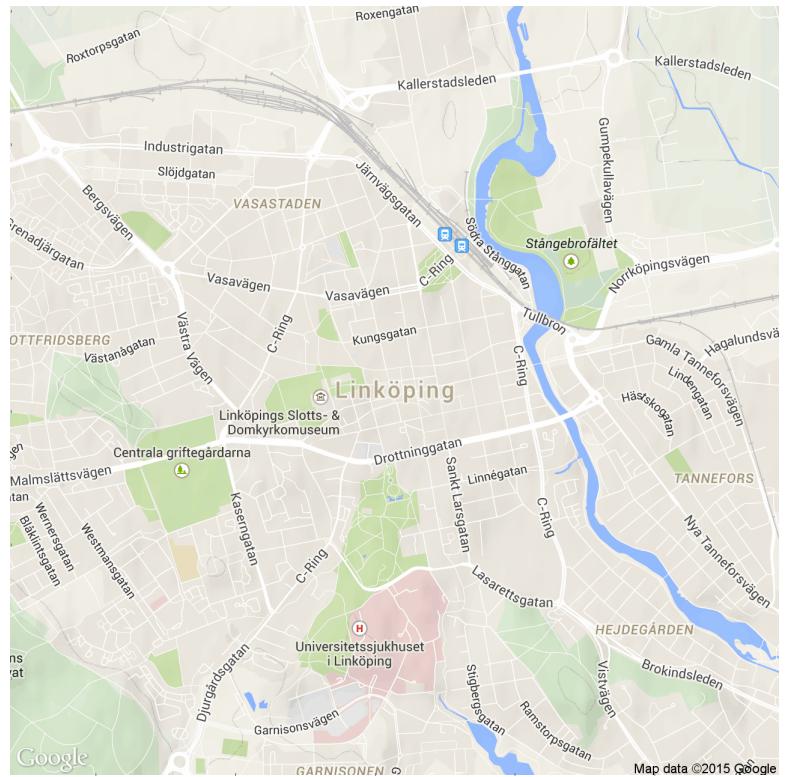
```
library(ggmap)
qmap("Linkoping")
```

*Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=10&size=%2062x496
Google Maps API Terms of Service : http://developers.google.com/maps/terms
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false
Google Maps API Terms of Service : http://developers.google.com/maps/terms*



2. Som framgår ovan så får vi dels upp en karta över Linköping (på en relativt hög upplösning) och dels information om att vi använt Google Maps samt information om vilka villkor som gäller för användadet av kartan. Framöver kommer denna extra information döljas. Vill vi zomma in eller ut använder vi argumentet **zoom**.

```
qmap("Linkoping", zoom=14)
```



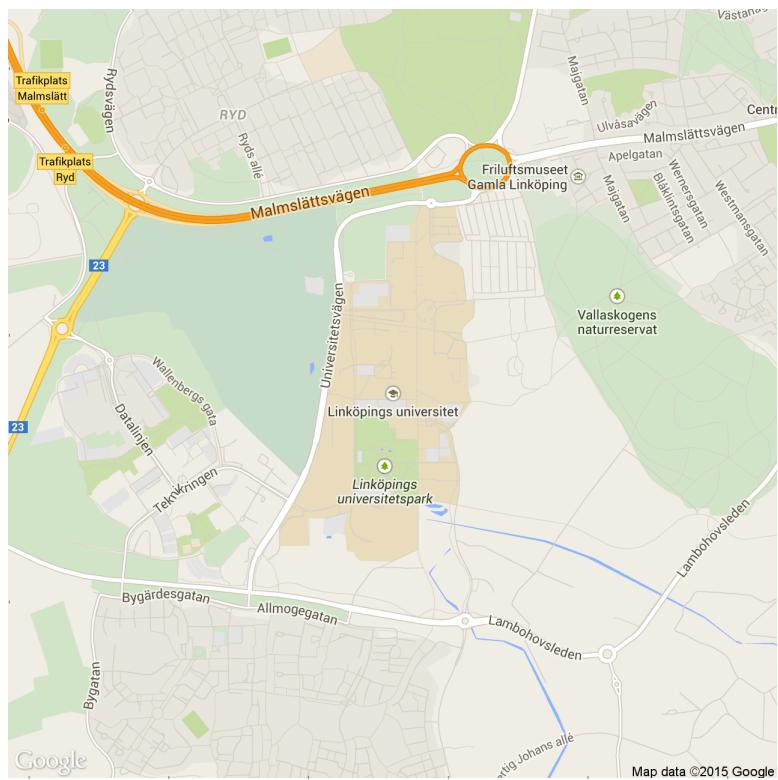
3. Vi kan också använda Google Maps för att få ut koordinater för enskilda platser med `geocode()`.

```
linkpg_uni <- unlist(geocode("Linkopings Universitet"))
linkpg_uni
```

lon	lat
15.576	58.398

4. Då `qmap()` använder sig av koordinater kan vi också ange koordinater för att skapa kartor. Med argumentet `maptype` van vi välja mellan google maps kartor "roadmap", "satellite", "hybrid", "terrain".

```
qmap(location=linkpg_uni, zoom=14, maptype="roadmap")
```



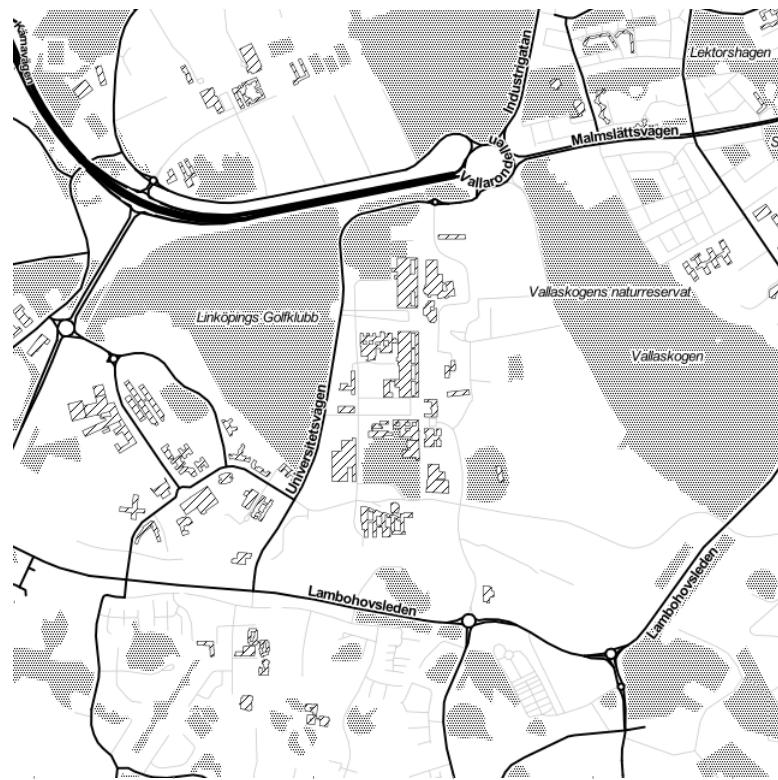
5. Vi kan också välja svartvita kartor med argumentet color. Precis som i `ggplot2` kan vi spara våra kartobjekt för att skriva ut dem senare.

```
karta <- qmap(linkpg_uni, zoom = 14, maptype="satellite", color="bw")
```

6. Det finns fyra olika kartsystem som går att använda från `ggmap`. Vill vi använda något annat kartsystem använder vi `source`.

```
qmap(linkpg_uni, zoom = 14, source = "stamen", maptype = "toner")
```

*Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=58.397836,15.576007&zoom=14  
Google Maps API Terms of Service : http://developers.google.com/maps/terms*



- Nu har vi skapat kartor i R. Nästa steg är att kombinera kartor och spatiala data.

## 2.4 Kombinera spatiala data med olika projektioner/geografiska referenssystem

När vi har många olika typer av data vi vill kombinera på samma karta måste de ha samma projektion och samma koordinatsystem. En bra och kort introduktion till kartprojektioner hittar du [\[här\]](#).

Alla data vi vill kombinera behöver vi känna till vilket geografiskt referenssystem som används. Exempelvis Google Maps använder sig av formatet EPSG 4326 (vilket vi kallar longitud och latitud). För att kolla upp vilket EPSG-nummer ett format kan vi använda <http://spatialreference.org/>.

Vi ska nu prova att visualisera våra kommungränser vi läst in i shapefilen ovan på Google Maps.

- Det första steget vi måste ta är att vi måste konvertera våra kommunkartor till EPSG 4326 för att visualisera dem med Google Maps. Vi börjar med att tillskriva vår shapefil det EPSG-format filen har. (Filens användare SWEREF 99 TM - slå upp det på <http://spatialreference.org/> så framgår EPSG-koden).

```
proj4string(swe_municip) <- CRS("+init=epsg:3006")
```

- Vi har nu tillskrivit vårt spatiala objekt ett geografiskt referenssystem. Nästa steg är att konvertera den till longitud och latitud. För att konvertera mellan geografiska referenssystem behövs rgdal-paketet. Vi vet att EPSG-koden för detta är 4326 och använder spTransform() för att göra konverteringen på följande sätt.

```
library(rgdal)
```

```
rgdal: version: 0.8-16, (SVN revision 498)
Geospatial Data Abstraction Library extensions to R successfully loaded
Loaded GDAL runtime: GDAL 1.11.0, released 2014/04/16
Path to GDAL shared files: /Users/manna97/Library/R/3.1/library/rgdal/gdal
```

```
Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
Path to PROJ.4 shared files: /Users/manma97/Library/R/3.1/library/rgdal/proj

swe_municip_4326 <- spTransform(swe_municip, CRS("+init=epsg:4326"))
```

3. Nu har vi konverterat vår shapefil till rätt format och kan visualisera våra data på en karta. För att `ggplot2` ska kunna läsa ett spatialt polygonobjekt behöver vi först göra om det spatala objektet till en `data.frame` som ggplot kan använda med

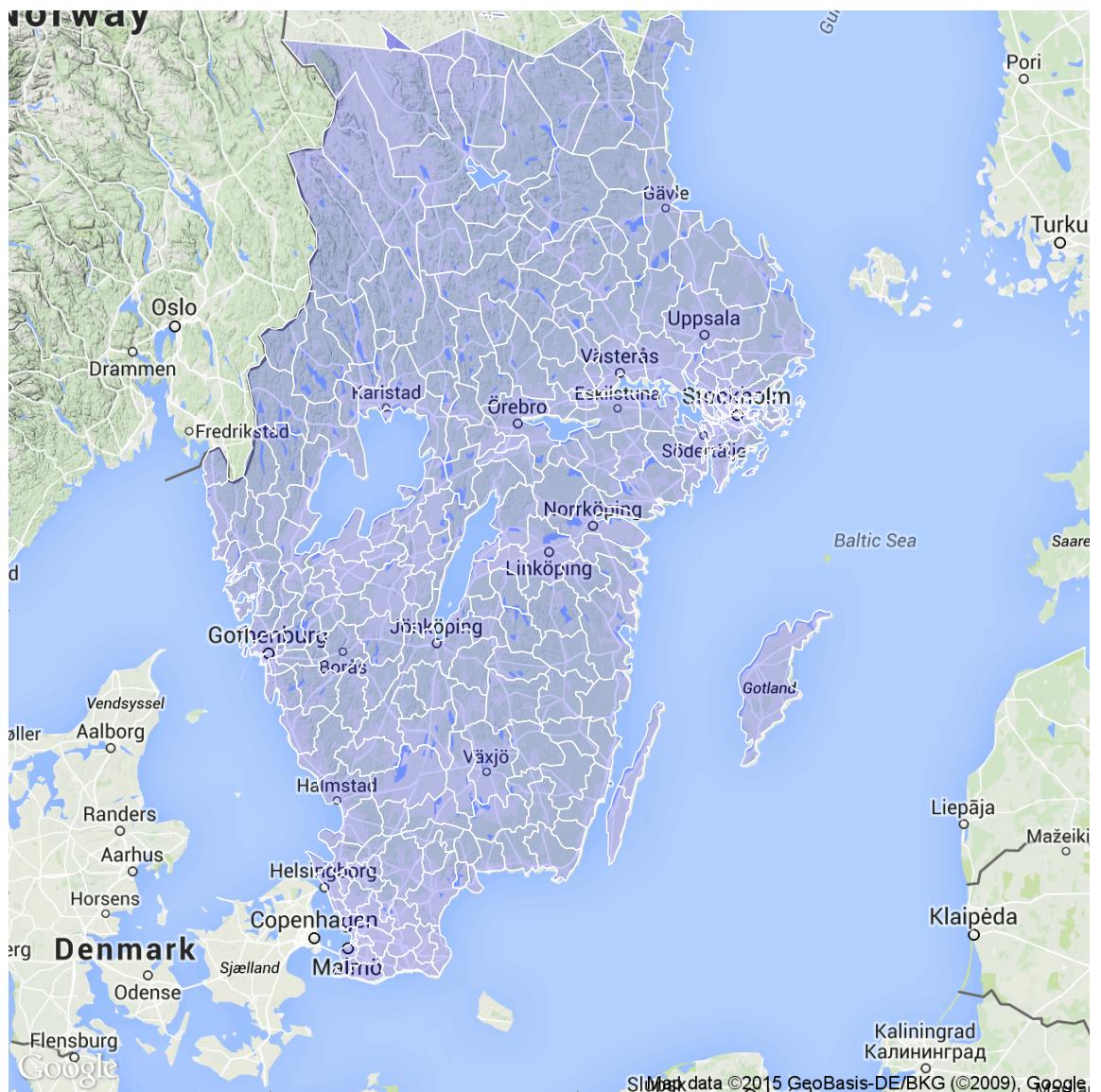
```
data <- fortify(swe_municip_4326)

Regions defined for each Polygons

linkpg_large <- qmap("Linkoping", zoom = 6, maptype = "terrain")

Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=6&size=%2064
Google Maps API Terms of Service : http://developers.google.com/maps/terms
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false
Google Maps API Terms of Service : http://developers.google.com/maps/terms

linkpg_large + geom_polygon(aes(x = long, y = lat, group = group), data = data,
```



4. På detta sätt kan vi visualisera de flesta spatiala datatyper på ett enkelt och snabbt sätt.

# Litteraturförteckning

- [1] Leland Wilkinson. *The grammar of graphics*. Springer, 2012.
- [2] Leland Wilkinson, D Wills, D Rope, A Norton, and R Dubbs. *The grammar of graphics*. Springer, 2006.

## Del II

# Inlämningsuppgifter

## Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

1. Lös uppgiften med vanlig kod direkt i R-Studio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

## Automatisk återkoppling med markmyassignment

Som ett komplement för att snabbt kunna få återkoppling på de olika arbetsuppgifterna finns paketet markmyassignment. Med detta är det möjligt att direkt få återkoppling på uppgifterna i laborationen, oavsett dator. Dock krävs internetsanalsutning.

För att installera markmyassignment krävs paketet devtools (som därför först måste installeras):

```
> install.packages("devtools")
> devtools::install_github("MansMeg/markmyassignment")
```

För att automatiskt återkoppla en laboration behöver du först ange vilken laboration det rör sig om på följande sätt:

```
> library(markmyassignment)
> set_assignment("[assignment path]")
```

där [assignment path] är en adress du får av läraren till varje laboration.

För att se vilka uppgifter som finns i laborationen kan du använda funktionen `show_tasks()` på följande sätt:

```
> show_tasks()
```

För att få återkoppling på en uppgift använder du funktionen `mark_my_assignment()`. För att rätta samtliga uppgifter i en laboration gör du på följande sätt:

```
> mark_my_assignment()
```

Tänk på att uppgifterna som ska kontrolleras måste finnas som funktioner i R:s globala miljö. Du kan också kontrollera en eller flera enskilda uppgifter på följande sätt:

```
> mark_my_assignment(tasks="foo")
> mark_my_assignment(tasks=c("foo", "bar"))
```

Det går också att rätta en hel R-fil med samtliga laborationer. Detta är bra att göra innan du lämnar in din laboration. För att rätta en hel fil gör du på följande sätt:

```
> mark_my_assignment(mark_file = "[my search path to file]")
```

där [my search path to file] är sökvägen till din fil.

**Obs!** När hela filer kontrolleras måste den globala miljön vara tom. Använd `rm(list=ls())` för att rensa den globala miljön.

## Kapitel 3

# Inlämningsuppgifter

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)

Loading required package: yaml
Loading required package: testthat
Loading required package: httr

lab_path <-
"https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/Tests/d8.yml"
set_assignment(lab_path)

Assignment set:
D8 : Statistisk programmering med R: Lab 8
```

### 3.1 dynamic\_linear\_system()

Vi ska nu skapa en funktion som beräknar ett linjärt system och kan visualisera detta system över tid. Vi utgår från systemet för grodor och flugor som finns i kap. 1.3 [här]. Funktionen ska ha följande argument

`f_0` en initial startposition ( $n = 0$ ) för ett linjärt system av godtycklig storlek `f_0`.

`A` transitionsmatris.

`n` för vilka steg beräkningar ska göras.

Med dessa funktioner ska funktionen beräkna nivån (ex. antal flugor och grodor) vid de tidpunkter som anges av i enlighet med

$$f_{n+1} = \mathbf{A}f_n$$

Funktionen ska returnera en `data.frame` med variabeln `n` samt  $f_n$  på respektive rad. Om vi anger en vektor `f_0` utan namn ska variabelnanen vara `var1`, `var2` ... o.s.v. Anger vi en namngiven vektor ska namnen vara variabelnamn i den `data.frame` som returneras. Variabeln `n` ska också kunna ta godtyckliga värden för  $n$ .

**Tips!** `diagonalize_matrix()`

Hur funktionen ska fungera framgår nedan.

```
A <- matrix(c(.38, .24, -.36, 1.22), ncol=2, byrow = TRUE)
f_0 <- c(10,10)
n <- c(1:3,5,10)
dynamic_linear_system(f_0, A, n)
```

```

n      var1      var2
1 0 10.0000 10.0000
2 1 6.2000 8.6000
3 2 4.4200 8.2600
4 3 3.6620 8.4860
5 5 3.4710 9.7881
6 10 5.1953 15.5664

f_0 <- c(1,3)
names(f_0) <- c("frogs", "flies")
dynamic_linear_system(f_0, A, n)

n frogs flies
1 0 1.0000 3.0000
2 1 1.1000 3.3000
3 2 1.2100 3.6300
4 3 1.3310 3.9930
5 5 1.6105 4.8315
6 10 2.5937 7.7812

A <- matrix((1/100)*c(80, 30, 30, 5, 55, 5, 15, 15, 65), ncol=3, byrow = TRUE)
companies <- c(20,40,40)
names(companies) <- c("Fido", "Telus", "Rogers")
dynamic_linear_system(f_0=companies, A, n=c(1,3, 5, 10, 1000))

n Fido Telus Rogers
1 0 20.000 40.000 40.000
2 1 40.000 25.000 35.000
3 3 55.000 13.750 31.250
4 5 58.750 10.938 30.312
5 10 59.961 10.029 30.010
6 1000 60.000 10.000 30.000

```

## 3.2 Miniprojektet del II

Den sista delen av denna laboration är att genomföra miniprojektet del II. Se kurshemsidan för detaljer.  
*Nu är du klar!*