Datorlaboration 3 - Multivariata fördelningar och Maximum likelihood

Statistik och dataanalys III, 15 hp

Mona Sfaxi och Mattias Villani

I. Introduktion

I den här datorlabben kommer vi först att uppehålla oss lite kort vid matriser och sedan arbeta med multivariata fördelningar, där särskilt fokus kommer riktas mot den multivariata normalfördelningen och hur man kan visualisera den grafiskt. Därefter kommer vi att arbeta med olika likelihoodfunktioner och maximum likelihood (ML) skattningar följt av deras samplingfördelningar. Vi kommer att använda datorn som hjälp för att slippa krångliga derivator.

När du är klar med den här datorlabben så kommer du kunna börja med del 2 i inlämningsuppgiften.

II. Installera paket

Den här labben använder multivariat normalfördelning från paketet `mvtnorm`. Vi kommer även att använda oss av paketen `plotly` för att göra 3D grafer. Paketen `remotes` och `SUdatasets` (som kan laddas ner via github) kommer behövas för att ladda ner dataset. För att få ut det mesta av den här datorlabben bör du rendera den till en html-fil. På så sätt kan du få interaktiva 3D-plottar när du använder dig av `plotly`.

- ☐ Skapa en code-chunk där du laddar ner alla paket. Skriv `#| output: false` längst upp i din code-chunk för att filtera bort störande meddelanden som dyker upp då man laddar paket. Dessa bör inte vara med i ditt färdiga dokument. Får du ändå dessa meddelanden efter att du har renderat ditt Quarto-dokument så beror det förmodligen på en felstavning.

Tänk på att man endast behöver installera ett paket *en* gång, sedan bör man kommentera bort den koden med hjälp av #. Annars kommer paketet att installeras på nytt varje gång man renderar sitt dokument. Förutom att det tar extra tid att rendera kan det också ge upphov till felmeddelanden som stoppar renderingen om man har otur.

1. Något kort om matriser

Vi har tidigare sett i Datorlabb 1 hur man kan skapa lite olika matriser och använda sig av loopar för att exempelvis fylla dem med olika värden. I det här avsnittet ska vi ägna oss åt ett par grundläggande beräkningar av matriser.

Matriser betecknas ofta i text med fetstilade stora bokstäver medan vektorer ofta skrivs med fetstilade små bokstäver. Säg att du har matrisen X

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

> **Uppgift 1.1** Skapa en matris `X` i R av dimension 2x2 som den ovan. (Se datorlabb 1, sida 10 ifall du har glömt hur man gör).

Determinanten av en matris A betecknas ibland som $\det(A)$ eller $|A|$ (det betyder alltså inte absolutbelopp inom linjär algebra). Vi ska först titta på hur man kan beräkna determinanten av en 2x2 matris A, där

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

För en 2x2 matris kan determinanten alltid beräknas som $|A| = ad - bc$.

Diagonalen av en matris är ett annat begrepp som är viktigt. Det kan betecknas som $\text{diag}(A)$ och är en vektor bestående av de diagonala elementen från matrisen. I det här fallet är $\text{diag}(A) = [a \;\; d]$.

> **Uppgift 1.2** Beräkna för hand deterimanten och diagonalen av matrisen X som du skapade i R.

Ett annat viktigt begrepp är **inversen** av en matris och det betecknas som A^{-1}. Det skiljer sig från inversen av en funktion som vi tittade på i datorlabb 2. Du kommer senare i den här Datorlabben att använda dig av inversen av en matris för dina beräkningar.

Inversen av en 2x2 matris kan beräknas genom

$$\begin{aligned} A^{-1} &= \frac{1}{ad - bc} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \\ &= \frac{1}{|A|} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \end{aligned}$$

Vi ser alltså att bråket $\frac{1}{|A|}$ multipliceras med varje element i matrisen:

$$\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

och att resultatet A^{-1} är en ny matris.

Uppgift 1.3* Beräkna (för hand) inversen av matrisen som du skapade i R.

Uppgift 1.4* Beräkna inversen av matrisen B, där

$$B = \begin{bmatrix} 3 & 6 \\ 6 & 12 \end{bmatrix}$$

för hand och i R med hjälp av funktionen `solve(min_matris)`. Vad är det som händer och varför?

Avslutningsvis, diagonalen av en matris beräknas med samma princip som det illustrerades ovan oavsett om det är en 2x2 eller en 20x20 matris. Däremot behöver man använda sig av andra metoder för att beräkna determinanten och inversen av en matris av högre dimension än 2x2. Men det är inget vi kommer behöva göra i den här kursen. Istället kommer vi att ge R detta nöje. Det kan dock vara betydelsefullt att förstå principen som ligger bakom dessa beräkningar så att man vet vad man använder sig av i funktionerna under Avsnitt 4.

2. Multivariata fördelningar

I det här avsnittet ska vi titta på bivariata fördelningar, där speciellt fokus kommer riktas mot den multivariata normalfördelningen, och hur man kan illustrera dem grafiskt. Låt oss först börja med ett exempel från en bivariat sannolikhetsfördelning där

$$f(y_1, y_2) = y_1 + y_2, \qquad 0 \leq y_1 \leq 1,\ 0 \leq y_2 \leq 1,$$

För att rita denna fördelning kan vi antingen göra en så kallad contour-plot eller en 3D-graf. Låt oss först börja med en contour-plot. Den första variabeln representeras där horisontellt och den andra variabeln representeras vertikalt i grafen. Lite likt en karta över ett berg ser man ett mönster i form av linjer, där linjernas värden blir större ju högre upp de befinner sig. Det markerar att mest massa finns där.

Men innan vi börjar rita måste vi först definiera funktionen i R samt koda fram definitionsområdet för respektive variabel så att R vet var funktionen ska ritas någonstans. Detta görs smidigt med funktionen `seq(min, max, length)`, som låter oss skapa en fin så kallad "grid" av värden. Vi ser ovan att $0 \leq y_1 \leq 1$ och $0 \leq y_2 \leq 1$. I det här fallet räcker det med endast 100 värden var.

```
fy1y2 <- function(y1, y2) y1 + y2

y1Grid <- seq(0, 1, length = 100)
y2Grid <- seq(0, 1, length = 100)
```

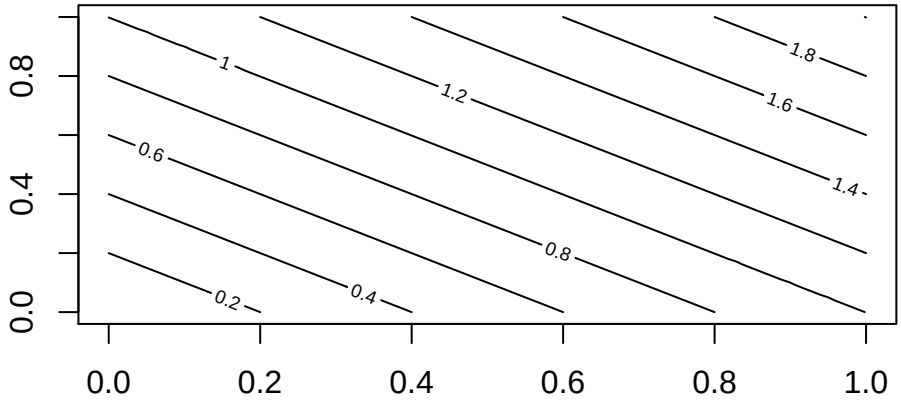
Därefter behöver vi beräkna funktionsvärdet givet alla olika kombinationer av värden på y_1 och y_2 som vi nyss har skapat. Eftersom vi har skapat 100 värden var för varje variabel så kommer

resultatet alltså att bli en matris av storlek 100×100 som består av olika funktionsväden för $f(y_1, y_2)$. Detta kan vi göra genom att använda oss av funktionen
`outer(X = variabel1, Y = variabel2, FUN)`.

```
fy1y2_values <- outer(X = y1Grid, Y = y2Grid, FUN = fy1y2)
```

Nu kan vi skapa vår contour-plot genom att använda funktionen contour().

```
contour(y1Grid, y2Grid, fy1y2_values)
```



För att skapa en 3D-plot går vi tillväga på liknande sätt. Vi använder oss av funktionen `plot_ly(x, y, z, type = "surface")` likt nedan

```
plot_ly(x = y2Grid, y = y1Grid, z = fy1y2_values, type = "surface")
```

2.1 Multivariat normalfördelning

Låt X vara en bivariat normalfördelelad matris. X är alltså en matris av dimension $n \times k$ där n är totalt antal observationer (rader) och k är antal variabler. Det innebär att varje variabel (kolumn) har ett väntevärde och att man kan representera alla väntevärden i en vektor av

storlek `k` dvs i detta fall har vi; $\mu = [\mu_1, \mu_2]$. Variablerna har även en varians och en kovarians med varandra. Detta kan man representera i en kovariansmatris Σ (uttalas Sigma) där

$$\Sigma = \begin{bmatrix} \sigma_1^2 & cov(x_1, x_2) \\ cov(x_2, x_1) & \sigma_2^2 \end{bmatrix}$$

Till att börja med ska du nu simulera från en bivariat normalfördelning och sedan plotta resultatet.

Uppgift 2.1* Skapa först en vektor `mu` som består av värdena 5 och 7 och skapa sedan en matris `Sigma` som består av varianserna 1 för x_1, 2 för x_2 och låt kovariansen mellan dem vara lika med -0.75.

Uppgift 2.2* Simulera därefter 2000 observationer med hjälp av funktionen `rmvnorm(Antal_observationer, mean, sigma)`. Rita dina nya simulerade observationer i ett spridningsdiagram med hjälp av funktionen `plot()`. Vad ser du för mönster? Ser det ut som du har förväntat dig?

Vi har ovan simulerat för att få en aning om hur den bivariata normalfördelningen ser ut i praktiken. Men hur gör man ifall man vill titta grafiskt på täthetsfunktionen och inte simulerade värden? Exempelvis kan man skapa en `contour` plot eller en 3D-graf. Vi börjar återigen med en `contour`-plot.

Uppgift 2.3 Börja med att skapa två gridar av x-värden med hjälp av funktionen `seq(min, max, length)`. Låt x_1 gå från 2 till 10 och x_2 gå från 0 till 14. Båda ska bestå av 100 observationer var. Kalla dem exempelvis för `xGrid1` och `xGrid2`. Du kommer att behöva dessa nedan för att senare beräkna funktionsvärdet av fördelningen.

Uppgift 2.4 Skriv sedan en funktion vid namn `fX` som tar argumenten `x1` och `x2` och använd dig av funktionen `dmvnorm()` som finns i paketet `mvtnorm`. Precis som vid den univariata normalfördelningen så representerar ett `d` framför funktionsnamnet täthetsfunktionen. `dmvnorm()` tar argumenten `x`, som består av dina faktiska observationer för dina olika variabler (x tar alltså en matris av storlek $n \times k$, i detta fall är n = 100 och k = 2). Funktionen tar även argumenten `mean` och `sigma`. Låt dessa vara samma värden som du använde dig av tidigare när du simulerade. Tips: I just det här specifika fallet bör du använd dig av `x = cbind(xGrid1, xGrid2)` inne i `dmvnorm()`.

Uppgift 2.5 Använd dig därefter av funktionen `outer(x, y, min_funktion)` där x är `xGrid1`, y är `xGrid2` och min_funktion är `fX` som du skapade precis ovan. Spara dina observationer under ett passande namn som exempelvis `z`. `z` bör vara en matris av dimension 100x100 som består av olika funktionsvärden från en bivariat normalfördelning.

Uppgift 2.6 Plotta sedan din fördelning, använd gärna pseudo-koden:

```
contour(mina_x1_värden, mina_x2_värden, mina_funktionsvärden)
```

Om du vill justera antal ringar i din plot kan du använda dig av argumentet `nlevels =`.

Vi kan nu fortsätta med att skapa en 3D graf över normalfördelningen ovan. Allt förarbete har också redan gjorts då funktionen använder sig av i princip samma argument som `contour()` funktionen.

Uppgift 2.7 Använd dig av funktionen `plot_ly()`. För att få en 3D-plot behöver du skriva type = "surface" likt nedan. Lägg märke till att du också kan skriva type = "contour" för att istället få en mer informativ contour-plot än vad du fick tidigare.

```
plot_ly(x = mina_x2_värden, y = mina_x1_värden, z = mina_funktionsvärden,
        type = "surface")
```

Här måste man alltså skriva ut argumenten x, y och z i funktionen `plotly()`. Annars så fungerar det inte.

2.2 Vad gör koden ovan?

Den multivariata normalfördelningen ges av formeln

$$f(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} e^{(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))}$$

Där x är en vektor bestående av k stycken variabler och μ är en vektor, också av storlek k som består av väntevärdet av varje variabel. Σ (uttalas sigma) är en kovariansmatris av dimension $k \times k$, $|\Sigma|$ är determinanten av kovariansmatrisen och Σ^{-1} är inversen av matrisen. Har du inte läst linjär algebra tidigare så kanske uttrycket ovan kommer se främmande ut. Men oroa dig inte.

Eftersom vi nu tittar på en bivariat normalfördelning där k = 2 så kan vi istället skriva uttrycket ovan utan att använda oss av matriser och vektorer och skriva täthetsfunktionen $f(x_1, x_2)$ som

$$\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2(1-\rho^2)}\left(\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2}\right)\right]$$

Där ρ (uttalas `rho`) är korrelationen mellan x_1 och x_2.

Uppgift 2.8* Använd dig av uttrycket ovan och skriv en funktion i R som tar argumenten: x1, x2, mu1, mu2, sigma1, sigma2 och rho. Låt de olika Parametrarna ha defaultvärden i funktionen där $\mu_1 = 5, \mu_2 = 7, \sigma_1 = 1, \sigma_2 = 2$ och $\rho = -0.75$ precis som tidigare. Kalla funktionen för `Bivariate_Normal`. Alternativt så kan du direkt använda dig av funktionen `dmvnorm()`.

Uppgift 2.9* Bilda en tom matris av dimension 100x100 som du fyller med nollor och kallar för `fx1x2`.

Uppgift 2.10* Skriv sedan en dubbelloop som fyller varje cell i `fx1x2` med funktionsvärden som fås från funktionen `Bivariate_Normal` (alternativt från funktionen `dmvnorm()`) genom att låta x1 vara lika med `xGrid1` och x2 vara lika med `xGrid2` som du skapade innan. (Tips: i dubbellopen, låt
`fx1x2[i,j] = Bivariate_Normal(x1 = xGrid1[i], x2 = xGrid2[j]))`.

Uppgift 2.11* Rita nu en 3D-graf med dina värden som du nyss skapat genom att använda dig av funktionen plot_ly()), där argumenten i funktionen ges av: `plot_ly(x = mina_x2_värden, y = mina_x1_värden, z = mina_funktionsvärden, type = "surface")`.

2.3 Extrauppgifter

Låt Y_1 och Y_2 ha den simultana sannolikhetsfördelningen

$$f(y_1, y_2) = e^{-(y_1+y_2)}, \qquad y_1 > 0,\ y_2 > 0$$

Uppgift 2.12 Skapa en funktion i R som representerar $f(y_1, y_2)$ ovan och rita den i en 3D-plot på liknande sätt som du gjorde med den bivariata normalfördelningen. Vi ser att båda variabler är strikt positiva och att de i princip går mot oändligheten. Men det går inte bra att koda på det sättet. Här måste du själv bestämma dig för rimliga värden som de högst kan ta när du ritar. Pröva dig gärna fram med olika värden.

Låt nu istället Y_1 och Y_2 ha den simultana sannolikhetsfördelningen

$$f(y_1, y_2) = 2y_1, \qquad 0 \leq y_1 \leq 1,\ 0 \leq y_2 \leq 1$$

Uppgift 2.13 Rita fördelningen i en 3D-plot samt i en contour-plot och jämför dem båda.

3. Likelihoodfunktioner

Låt $X_1, X_2, ..., X_n$ vara oberoende variabler från samma fördelning. Likelihoodfunktionen ges av $\mathcal{L}(\theta|x)$, där θ är en parameter. (Observera att du inte får skriva likelihoodfunktionen som $P(\theta|x)$ då detta är en aposteriorifördelning och något vi kommer titta på i laboration 4 då vi kommer syssla med Bayesiansk inferens). θ kan också vara en vektor och bestå av flera parametrar som exempelvis μ och σ om vi tittar på en normalfördelning. I likelihoodfunktionen behandlas data som fixt medan parametrarna kan variera. Särskilt intressant blir det när vi försöker finna det parametervärde som ger oss det mest troliga värdet på vår likelihoodfunktion, men det kommer vi att (o)roa oss för i nästa avsnitt.

Låt oss först titta på ett exempel på log-likelihoodfunktionen för en Bernoullifördelning. Bernoullifördelningen ges av

$$p(y) = p^y(1-p)^{1-y}$$

Så likelihoodfunktionen ges av

$$L(p|y_i) = \prod_{i=1}^{n} p(y) = p^{y_i}(1-p)^{1-y_i}$$
$$L(p|y_i) = p^{\sum y_i}(1-p)^{n-\sum y_i}$$

och loglikelihoodfunktionen ges av

$$\sum_{i=1}^{n} y_i ln(p) + \left(n - \sum_{i=1}^{n} y_i\right) ln(1-p)$$

Vi börjar med att skapa ett dataset som vi vet följer en Bernoullifördelning genom att simulera 400 observationer från en Bernoullifördelning med $p = 0.25$. Det finns ingen inbyggd funktion för Bernoullifördelningen i R. Däremot kan vi utnyttja funktionen för Binomialfördelningen och låta size = 1, så får vi automatiskt en massa observationer som antingen är 0 eller 1, dvs alla observationer kommer från en Bernoullifördelning med $p = 0.25$.

```
set.seed(3134)
ySim <- rbinom(n = 400, size = 1, prob = 0.25)
```

För att kontrollera att allting stämmer så kan vi ta en titt på våra simulerade värden på `ySim`. Vi vet exempelvis att väntevärdet för en Bernoullifördelning $= p$ därför bör medelvärdet av våra simulerade observationer från `ySim` ligga runt 0.25. Vi testar om så är fallet:

```
mean(ySim)
```

```
[1] 0.26
```

Och bekräftar att det ligger väldigt nära det teoretiska värdet. Så allt ser ut att stämmma (det kommer inte att vara exakt 0.25 pga att det är slumpmässiga dragningar).

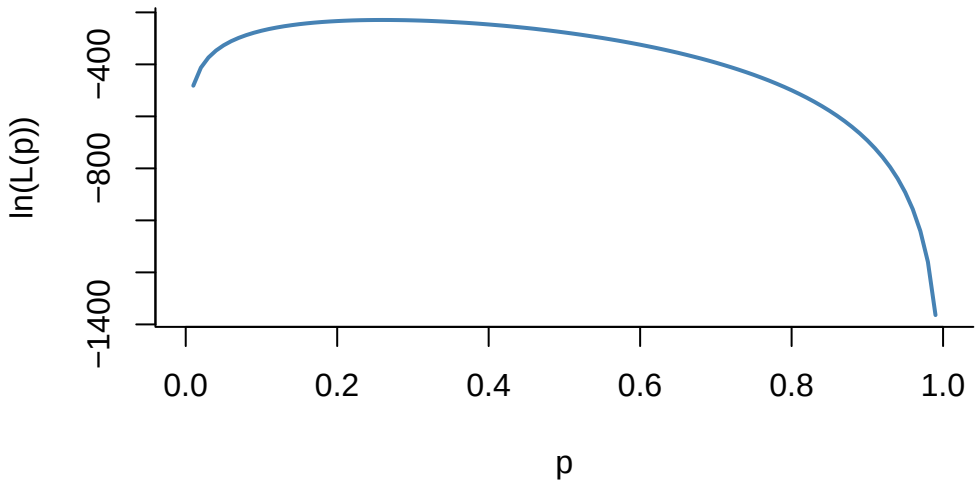
Nu kan vi skriva en funktion för loglikelihoodfunktionen i R som tar argumenten `p` och `y` utifrån ekvationen ovan.

```
loglik_bern <- function(p, y){
  n <- length(y)
  loglik <- sum(y)*log(p) + (n - sum(y))*log(1-p)
  return(loglik)
}
```

Tänk på att vi skriver loglikelihood funktionen som en funktion av vår parameter `p`. Därför kan vi sedan skapa en grid av värden för vårt `p` och se vilka funktionsvärden detta resulterar i för vår loglikelihoodfunktion. För att få en bra visuell bild av hur loglikelihoodfunktionen ändras givet olika värden på `p` kan vi plotta våra värden av p på x-axeln och $ln(\mathcal{L}(p))$ på y-axeln.

```
pGrid <- seq(0, 1, by = 0.01)
L_p <- loglik_bern(p = pGrid, y = ySim)

plot(pGrid, L_p, type = "l", col = "steelblue", lwd = 2, bty = "l",
     xlab ="p",ylab ="ln(L(p))",)
```



Vi ser att funktionen stiger lite smått för små värden på p och börjar sedan sjunka någonstans vid p = 0.2-0.3 för att sedan sjunka allt snabbare och snabbare ju närmre p är till 1. Men är detta rimligt? I det här fallet så vet vi att alla våra observatiner $y_i \sim Bern(p = 0.25)$ så det är rimligt att likelihoodfunktionen har sitt maxvärde någonstans där $p = 0.25$!

Överkurs:

Låt oss försöka se vilket värde på likelihoodfunktionen som är störst och sedan identifiera vilket värde på *pGrid* detta motsvaras av. Därefter kan vi använda dessa kooridnater i funktionen `points()` för att rita ut var detta max-värde befinner sig någonstans och sedan används funktionen `text()` för att skriva ut exakt vad dessa värden motsvaras av. Det första argumentet i funktionen `text()` ger oss x-koordinaten för var vi vill placera vår text och det andra argumentet ger oss y-koordinaten för var vi vill placera vår text. Sedan använder vi oss av funktionen `paste0()` för att kombiera siffror och text i en och samma textsträng.

```
my_index <- which.max(L_p)
my_index
```

```
[1] 27
```

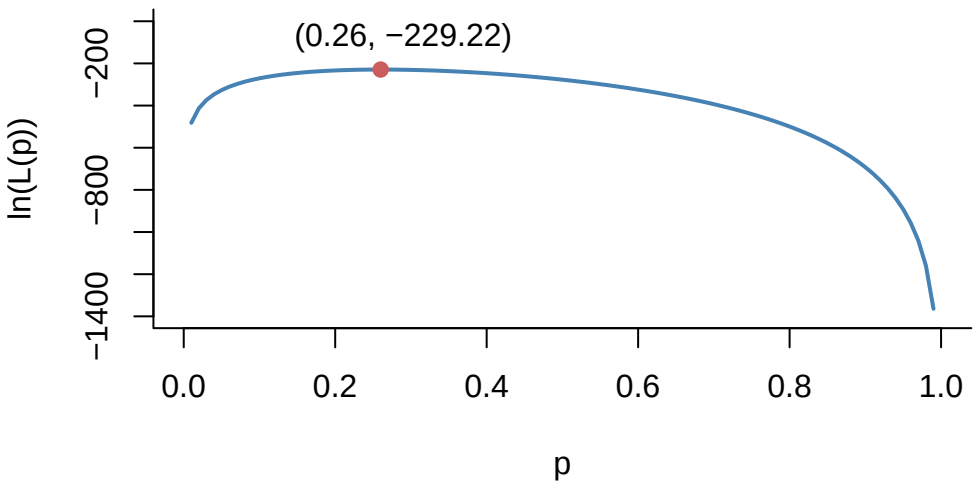
```
maxVal_L_p <- L_p[my_index]
maxVal_p <- pGrid[my_index]
maxVal_L_p # max value of the loglikelihood function
```

```
[1] -229.2228
```

```
maxVal_p #the value of p resulting in the max value of the loglikelihood function
```

```
[1] 0.26
```

```
plot(pGrid, L_p, type = "l", col = "steelblue", lwd = 2, bty = "l",
     xlab = "p", ylab = "ln(L(p))", ylim = c(-1400, 0))
points(x = maxVal_p, y = maxVal_L_p, col = "indianred", pch = 19)
text(x = (maxVal_p + 0.03), y = (maxVal_L_p + 150),
     paste0("(", maxVal_p, ", ", round(maxVal_L_p, 2), ")"))
```



Vi ser från ovan att värdet på p som maximerar likelihoodfunktionen ligger väldigt nära 0.25, i detta fall 0.26, vilket var samma värde som vi fick när vi beräknade medelvärdet för `ySim`. Är det en slump?

3.1 Likelihoodfunktionen av en exponentialfördelning

Låt $X_1, X_2, ..., X_n$ vara oberoende exponentialfördelade variabler, täthetsfunktionen för en exponentialfördelad variabel ges av

$$f(x_i) = \frac{1}{\beta} e^{-\frac{x_i}{\beta}} \tag{1}$$

Uppgift 3.1* Härled uttrycket för log likelihoodfunktionen utifrån täthetsfunktionen ovan (Utan att söka online).

Uppgift 3.2* Sätt ett seed till 1597 med funktionen `set.seed(1597)` så att du kan replikera dina resultat. Simulera därefter 50 observationer från en exponentialfördelning med parameter $\beta = 5$ med hjälp av funktionen `rexp`. Tänk på att du behöver skriva *rate = 1/β* i funktionen. Spara vektorn under ett passande namn som exempelvis `x`.

Uppgift 3.3* Skriv en funktion vid namn `loglik_exp` som först tar argumentet `beta` och sedan x. Använd dig av ditt uttryck från ovan för att definiera log

likelihood funktionen. Glöm inte att returnera uttrycket från funktionen i R! (Tips: Du kan också ha med ett argument för `n` om du vill, men det fungerar alldeles utmärkt att definiera `n` i din funktion).

Nedan syns ett exempel på vilken output funktionen bör ge då vi endast har 2 observationer och $\beta = 2$.

```
loglik_exp(beta = 2, x = c(3, 8.77))
```

```
[1] -7.271294
```

Uppgift 3.4* Skapa en grid av β-värden som går från 0.1 till 8 med hjälp av funktionen `seq(min, max, length)` eller `seq(min, max, by)` bestående av ca 100 värden. Döp vektorn till något passande som exempelvis `betaGrid`

Uppgift 3.5* Bilda en ny variabel i R vid namn `L_beta` genom att använda dig av funktionen `loglik_exp()` och låt `x` vara lika med de simulerade obsevationerna från exponentialfördelningen och låt `beta` vara lika med din grid av β-värden som du definierade ovan.

Uppgift 3.6* Illustrera sedan funktionen i en linjeplot med `betaGrid` på x-axeln och dina likelihoodvärden; `L_beta` på y-axeln. Ser du något intressant med funktionen?

3.2 Likelihoodfunktioner för geometriska fördelningar

Låt nu istället $Y_1, Y_2, ..., Y_n$ vara oberoende geometriskt fördelade variabler med parametern p. Frekvensfunktionen för Y_i där $i = 1, ..., n$ ges av

$$P(y_i) = (1-p)^{y_i - 1}p, \qquad y = 1, 2, ..., \infty$$

Uppgift 3.7 Finn (helst utan att söka online) uttrycket för log likelihoodfunktionen för den geometriska födelningen ovan.

Uppgift 3.8 Skapa nu två olika vektorer, `y1` och `y2`, genom att simulera 1000 observationer från en geometrisk fördelning med $p = 0.45$ respektive $p = 0.7$. Kom ihåg att R använder en annan definition av den geometriska fördelningen. För att lösa detta kan vi använda oss av ett trick i R; låt y1 = y1 + 1 och y2 = y2+1 så får du den definition som vi använder oss utav i kursboken. Kontrollera gärna att det stämmer genom att beräkna $\frac{1}{\bar{y}}$ för respektive variabel. Vad bör de teoretiska värdena vara?

Uppgift 3.9 Skriv en funktion, `loglik_geo` som tar argumenten `p` och `y` i den ordningen. Funktionen ska returnera det matematiska uttrycket för log likelihood funktionen för den geometriska fördelningen som du nyss tagit fram. Exempel på hur funktionen fungerar ges nedan för värdena $p = 0.2$ och y = 1, 3 och 8.

```
loglik_geo(p = 0.2, y = c(1, 3, 8))
```

```
[1] -6.836606
```

Uppgift 3.10 Skapa en grid av p-värden som går mellan 0 och 1 och som består av minst 100 värden med hjälp av funktionen `seq(min, max, length)` (eller alternativt `seq(min, max, by)`. Kalla din vektor för något passande som exempelvis `pGrid`.

Uppgift 3.11 Definiera två nya variabler `L_p_data1` och `L_p_data2` genom att använda dig av funktionen `loglik_geo()` där y är lika med `y1` för `L_p_data1` respektive `y2` för `L_p_data2`. Låt `p` vara lika med `pGrid` för dem båda.

Uppgift 3.12 Illustrera båda funktionerna i ett linjediagram genom att ha `pGrid` på x-axeln och Likelihoodfunktionerna på y-axeln. (Tips: använd dig av funktionen `plot()` och `lines()` för att rita dem båda i samma graf).

Uppgift 3.13 Vilket värde på `p` ger det mest sannolika likelihoodvärdet för respektive datamaterial? Det går bra att endast gissa här utifrån grafen men försök gärna att hitta dessa två värden och markera sedan dessa punkter i ditt diagram ovan om du får tid över. Tips: använd funktionerna `which.max(mina_likelihoodvärden)` och `max(mina_likelihoodvärden)` samt funktionen `points(min_x_koordinat, min_y_koordinat)` för att lägga till punkterna i diagrammet.

4. Maximum likelihood skattningar

Vi såg ovan hur likelihoodfunktionen såg ut för olika värden på våra parametrar givet ett visst dataset. Maximum likelihood (ML) metoden syftar till att hitta de mest troliga parameterskattningarna för vårt data. Vi ska nu fortsätta med att analytiskt och med hjälp av R finna ML-skattningar. Avslutningsvis kommer vi titta på hur man kan använda sig av normalapproximation för att få fram sina ML-skattningar

4.1 ML-skattning av parametern λ i Poissonfördelningen

Uppgift 4.1 Börja med att ladda ner datasetet `doctorvisits` från paketet `SUdatasets()` som består av antal besök till läkaren. Bilda sedan en ny variabel `visits` från kolumnen `numvisits`, men ta endast med de observationer där `numvisits` är högst 9 stycken. Koden nedan visar principen för hur man skulle kunna gå tillväga.

```
Ny_variabel <- min_df$min_variabel[min_df$min_variabel <= något_önskat_värde]
```

Låt oss fortsätta med Poissonfördelningen eftesom dess likelihoodfunktion är så förtjusande.

Har du tidigare inte installerat paketet `SUdatasets()` eller använder du en gammal version som inte har det här datasetet kan du behöva installera paketet igen med hjälp av koden:

```
library(remotes)
install_github("StatisticsSU/SUdatasets", force = TRUE)
library(SUdatasets)
```

Uppgift 4.2 Bevisa matematiskt att ML skattning för λ är $\hat{\lambda}_{ML} = \bar{X}$. Härled ett uttryck för standardavvikelsen för ML-skattningen. Beräkna sedan $\hat{\lambda}_{ML}$ och dess standardfel på datamaterialet. Tips: se gärna Denna tutorial för en mer utförlig förklaring kring det vi kommer syssla med i det här avsnittet eller ifall du fastnar.

Uppgift 4.3 Plotta data och den anpassade Poissonfördelningen där λ sätts lika med ML skattningen. Passar modellen data?

4.2 Negativ binomialfördelning och funktionen optim

En mer flexibel modell än Poissonmodellen är negativ binomialmodellen. En variant av negativ binomial är en generalisering av den geometriska modellen, se datorlabb 2. Negativ binomial med parametrar r och p räknar antal misslyckade Bernoulliförsök innan man får r lyckade försök. Om $r = 1$ får vi alltså geometrisk fördelning som specialfall. Vi kommer alltså nedan att använda oss utav R's definition av den negativa binomialfördelningen och inte bokens.

Vi ska också här använda en alternativ parametrisering av negativ binomialmodellen, som har en trevlig koppling till Poissonmodellen. I denna parametrisering sätter vi sannolikheten i Bernoulliförsöken till $p = \frac{r}{\mu+r}$, där μ visar sig vara väntevärdet för fördelningen. I den här nya parametriseringen är parametrarna r och μ istället för r och p, och vi tänker inte på modellen som något som kommer från ett antal Bernoulliförsök. Det är nu bara en praktisk fördelning för räknedata (counts) som har väntevärde lika med dessa andra parameter μ (precis som Poisson), men till skillnad från Poisson har vi nu ytterligare en parameter att leka med, r, vilket vi snart kommer se är trevligt.

Vi skriver denna negativ binomial modell som

$$X_i \overset{iid}{\sim} \text{NegBin}(r, \mu)$$

Notera följande trevliga egenskaper:

- När parametern r växer mot oändligheten så blir negativ binomialfördelningen alltmer lik Poissonfördelningen med parameter μ. Poissonfördelningen är alltså ett specialfall av negativ binomial. På matematiska säger vi att fördelningen $\text{NegBin}(r, \mu)$ konvergerar mot $\text{Poisson}(\mu)$ när $r \to \infty$.
- Väntevärdet i denna negativ binomial är alltid μ oavsett r, dvs samma väntevärde som en Poisson.
- Variansen är $\mu(1 + \frac{\mu}{r})$, vilket innebär att variansen för en $\text{NegBin}(r, \mu)$ variabel är alltid större än variansen för en $\text{Poisson}(\mu)$ variabel. Vi säger att negativ binomial är *overdispersed*. I många datamaterial ser man just det: stickprovsvariansen är större än medelvärdet. Då kan negativ binomial vara en bättre modell än Poisson (som ju alltid har samma väntevärde och varians).
- Nu när vi har släppt kopplingen till Bernoulliförsök kan vi t o m låta r vara vilket positivt tal som helst, dvs r behöver inte vara ett heltal. Så parametrarna r och μ är godtyckliga positiva tal.

Se negativ binomial för en interaktiv widget. Välj parameterization till 'mean' för att få den parameterisering vi använder här med r och μ.

> **Uppgift 4.4*** Som du kanske minns från datorlabb 2 så är `dnbinom(x, size = r, mu)` sannolikhetsfunktionen för den variant av negativ binomial som vi jobbar med här. Kom ihåg att argumentet r kallas `size` i R. Lek gärna runt med funktionen genom att exempelvis beräkna sannolikheten för $X = 1$ i en $\text{NegBin}(r = 1.5, \mu = 2)$ fördelning.

> **Tip**
>
> Du måste ange namnen på argumenten (`size` och `mu`) när du anropar `dnbinom`. Du kan t ex inte skriva `dnbinom(3, 2, 4)`.

Vi ser alltså att täthetsfunktionen ges av `dnbinom(x, size, mu)` i R. I övningen ovan används endast ett värde. Om vi vill så kan vi sätta argumentet `x` till flera värden, dvs en vektor av värden och på så sätt få ut flera olika funktionsvärden samtidigt. I **Uppgift 3.3 och 3.9** skrev vi egna loglikelihood funktioner "från grunden" genom att först härleda det matematiska uttrycket för dessa. Men det finns ett enklare sätt att göra detta på och det är genom att använda sig av redan inbyggda sannolikhetsfördelningar i R. I dessa finns ett argument; `log = FALSE` sätter man det till `TRUE` istället och summerar alla funktionsvärden så får man loglikelihood funktionen! Koden nedan illustrerar principen då man har två parametrar som man vill skatta från en godtycklig fördelning.

```
log_lik <- sum(dmin_fördelning(x = data, parameter1, parameter2, log = TRUE))
```

Du ska nu använd funktionen `optim` för att beräkna ML skattningen av r och μ. Funktionen använder sig av olika numeriska optimeringsmetoder man kan välja mellan. Den kan söka efter maximum- eller minimumvärden. Men vi är endast intresserade av maximum. Låt ditt dataset återigen bestå av variabeln `visits` från **Uppgift 4.1**

> **Uppgift 4.5*** skapa först en funktion vid namn `loglik_nb` som tar argumenten `param` och `x`. Båda kommer att vara vektorer. Observera att här måste argumentet `param` komma först! Låt sedan `r` och μ vara lika med param[1] respektive param[2]. Definiera sedan loglikelihood funktionen genom att summera `dnbinom(x, size, mu, log)` och returnera slutligen detta värde från funktionen.

För att kunna använda `optim` behöver du först skapa två stycken startvärden för dina två parametrar så att algoritmen kan påbörja sin sökning efter maximumvärdet.

> **Uppgift 4.6*** Skapa en vektor `initVal` med två stycken positiva startvärden. Tänk alltså på att log(0) inte är definierat. Använd dig sedan av funktionen `optim()`. Den halvfärdiga koden nedan innehåller alla argument som du kommer behöva använda dig av. Det första argumentet `par` bör sättas till `initVal`. Det andra argumentet `fn` bör vara lika med din funktion `loglik_nb`. Argumentet `x` är dina faktiska observationer, i det här fallet bör `x` vara lika med vektorn `visits`.

```
optimRes <- optim(par, fn, gr = NULL, x, method=c("L-BFGS-B"), lower = c(0,0),
      control=list(fnscale=-1), hessian=TRUE)
```

En liten förklaring kring övriga argument ges nedan.

- ☐ Argumentet `gr` står för derivata, eftersom vi inte har räknat ut någon derivata så sätts den till `NULL`.
- ☐ Argumentet `method` bestämmer vilken typ av optimeringsmetod du kommer använda dig av. För den nyfkne står "L-BFGS-B" för att det är en typ av quasi-Newton metod som används (men detaljerna kring det är inget du kommer behöva oroa dig för i den här kursen).
- ☐ Efter det kommer argumentet `lower`, som i det här fallet sätts till en vektor av två nollor då vi endast letar efter positiva värden på `r` och μ.
- ☐ Med argumentet `control=list(fnscale=-1)` säger vi till funktionen att vi vill leta efter maximumvärden, default är att leta efter minimumvärden.
- ☐ Slutligen låter vi argumentet `hessian` vara lika med `TRUE` så att vi också kan skatta en matris med andra derivatan av loglikelihood funktionen med hänseende på våra parametrar. Det kommer vara viktigt när vi kommer till **Uppgift 4.9 och 4.10** .

För en mer detaljerad förklaring av argumenten se det här avsnittet om numerisk optimering och optim

`optim()` returnerar alltid en lista av olika komponenter. Du kan nå de olika komponenterna med hjälp av $-tecknet (se datorlabb 1). Komponenterna i listan som är av störst intresse är `par` och `hessian`, där `par` är värdena på dina parametrar som maximerar din funktion.

> **Uppgift 4.7*** Skapa två nya variabler `thetaHat` och `J` som är lika med `par`, respektive `hessian`. Vilka värden har du fått på dina ML-skattningar av r och μ?

> **Uppgift 4.8** Plotta en anpassning av negativ binomialfördelning utifrån dina ML-skattningar tillsammans med ditt dataset.

4.3 Normalapproximation av ML-skattningarna

Låt $\theta = (r, \mu)^\top$ vara en vektor med de två parametrarna och låt $\hat{\theta} = (\hat{r}, \hat{\mu})^\top$ vara en vektor av ML skattningen. I stora stickprov vet vi att samplingfördelningen för ML-skattningen kan approximeras med en multivariat normalfördelning:

$$\hat{\theta} \overset{approx}{\sim} N(\theta, -J^{-1}),$$

där J är den observerade informationmatrisen.

Tidigare i **Uppgift 4.7*** sparade du ner Hessianen J, som består av andraderivatan av log-likelihoodfunktionen med avseende på de två parametrarna, från `optim()`. Den approximativa kovariansmatrisen för ML-skattningen ges av $-J^{-1}$, dvs minus inversen av J.

> **Uppgift 4.9*** Beräkna den approximativa kovariansmatrisen ovan och spara den under namnet `Cov`.

> **Uppgift 4.10*** Beräkna approximativa standardfel från denna kovariansmatris och skapa approximativa 95%-iga konfidensintervall för båda parametrarna. (Tips: beräkna först diagonalen av matrisen `Cov`).

5. Sammanfattning

I den här datorlabben har vi först ägnat oss lite åt matriser. Sedan har vi använt bivariata fördelningar och ritat dem i 3D-grafer i R. Vi har även härlett och skrivit loglikelihood funktioner och illustrerat dessa grafiskt. Slutligen har vi använt funktionen `optim` som verktyg för att finna ML-skattningar.