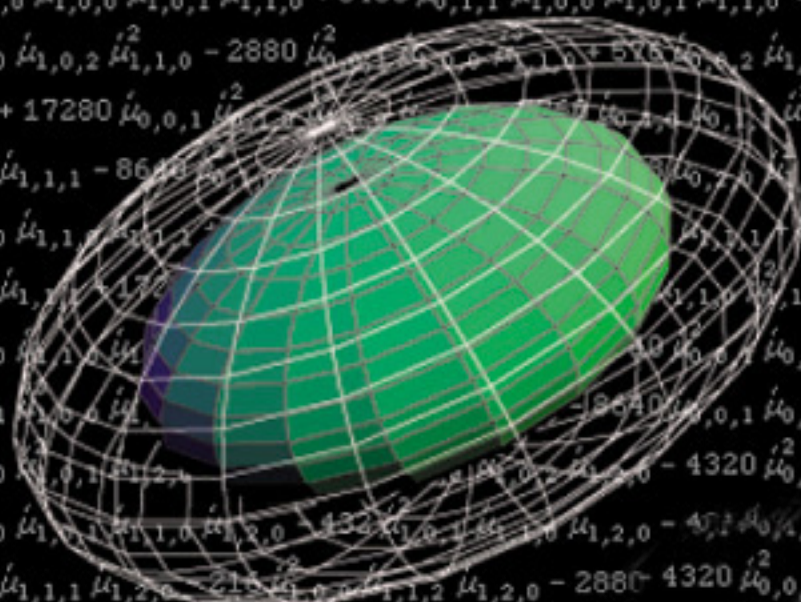
SPRINGER TEXTS IN STATISTICS

# MATHEMATICAL STATISTICS *with* *Mathematica*®



COLIN ROSE
MURRAY D. SMITH

# Mathematical Statistics with *Mathematica*

## Chapter 12 – Maximum Likelihood Estimation in Practice


12.1 Introduction 379

12.2 `FindMaximum` 380

12.3 A Journey with `FindMaximum` 384

12.4 Asymptotic Inference 392
- A Hypothesis Testing 392
- B Standard Errors and *t*-statistics 395

12.5 Optimisation Algorithms 399
- A Preliminaries 399
- B Gradient Method Algorithms 401

12.6 The BFGS Algorithm 405

12.7 The Newton–Raphson Algorithm 412

12.8 Exercises 418


## Please reference this 2002 edition as:


Rose, C. and Smith, M.D. (2002)
*Mathematical Statistics with Mathematica*, Springer-Verlag, New York.


## Latest edition

For the latest up-to-date edition, please visit: www.mathStatica.com

# Chapter 12

## Maximum Likelihood Estimation in Practice

### 12.1 Introduction

The previous chapter focused on the theory of maximum likelihood (ML) estimation, using examples for which analytic closed form solutions were possible. In practice, however, ML problems rarely yield closed form solutions. Consequently, ML estimation generally requires numerical methods that iterate progressively from one potential solution to the next, designed to terminate (at some pre-specified tolerance) at the point that maximises the likelihood.

This chapter emphasises the numerical aspects of ML estimation, using illustrations that have appeared in statistical practice. In §12.2, ML estimation is tackled using **mathStatica**'s `FindMaximum` function; this function is the mirror image of *Mathematica*'s built-in minimiser `FindMinimum`. Following this, §12.3 examines the performance of `FindMinimum` / `FindMaximum` as both a constrained and an unconstrained optimiser. We come away from this with the firm opinion that `FindMinimum` / `FindMaximum` should only be used for unconstrained optimisation. §12.4 discusses statistical inference applied to an estimated statistical model. We emphasise asymptotic methods, mainly because the asymptotic distribution of the ML estimator, being Normal, is simple to use. We then encounter a significant weakness in `FindMinimum` / `FindMaximum`, in that it only yields ML estimates. Further effort is required to estimate the (asymptotic) variance-covariance matrix of the ML estimator, which is required for inference. The remaining three sections focus on details of optimisation algorithms, especially the so-called gradient-method algorithms implemented in `FindMinimum` / `FindMaximum`. §12.5 describes how these algorithms are built, while §12.6 and §12.7 give code for the more popular algorithms of this family, namely the BFGS algorithm and the Newton–Raphson algorithm.

This chapter requires that we activate the **mathStatica** function `SuperLog`:

```
SuperLog[On]
```

```
— SuperLog is now On.
```

`SuperLog` modifies *Mathematica*'s `Log` function so that `Log[Product[]]` 'objects' or 'terms' get converted into sums of logarithms; see §11.1 B for more detail on `SuperLog`.

## 12.2 FindMaximum

Optimisation plays an important role throughout statistics, just as it does across a broad spectrum of sciences. When analytic solutions for ML estimators are not possible, as is typically the case in statistical practice, we must resort to numerical methods. There are numerous optimisation algorithms, a number of which are implemented in *Mathematica*'s `FindMinimum` function. However, we want to maximise an observed log-likelihood, not minimise it, so **mathStatica**'s `FindMaximum` function is designed for this purpose. `FindMaximum` is a simple mirror image of `FindMinimum`:

```
? FindMaximum
```

```
FindMaximum is identical to
  the built-in function FindMinimum, except
  that it finds a Max rather than a Min.
```

To illustrate usage of `FindMaximum`, we use a random sample of biometric data attributed to Fatt and Katz by Cox and Lewis (1966):

```
xdata = ReadList["nerve.dat", Number];
```

The data represents a random sample of size $n = 799$ observations on a continuous random variable $X$, where $X$ is defined as the time interval (measured in units of one second) between successive pulses along a nerve fibre. We term this the 'Nerve data'. A frequency polygon of the data is drawn in Fig. 1 using **mathStatica**'s `FrequencyPlot` function. The statistical model for $X$ that generated the data is unknown; however, its appearance resembles an Exponential distribution (*Example 1*), or a generalisation of it to the Gamma distribution (*Example 2*).

```
FrequencyPlot[xdata];
```

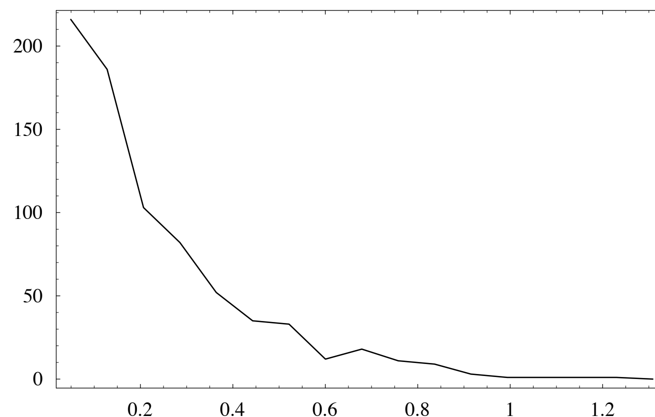



**Fig. 1:** The Nerve data

⊕ ***Example 1:*** `FindMaximum` — Part I

Assume $X \sim \text{Exponential}(\lambda)$, with pdf $f(x; \lambda)$, where $\lambda \in \mathbb{R}_+$:

```
f = (1/λ) ⅇ^(-x/λ);        domain[f] = {x, 0, ∞} && {λ > 0};
```

For $(X_1, \ldots, X_n)$, a size $n$ random sample drawn on $X$, the log-likelihood is given by:

$$\text{logL}\lambda = \text{Log}\left[\prod_{i=1}^{n} (\text{f} \;/.\; \text{x} \to \text{x}_i)\right]$$

$$-\frac{n\,\lambda\,\text{Log}[\lambda] + \sum_{i=1}^{n} x_i}{\lambda}$$

For the Nerve data, the observed log-likelihood is given by:

```
obslogLλ = logLλ /. {n → Length[xdata], x_i_ :→ xdata[[i]]}
```

$$-\frac{174.64 + 799\,\lambda\,\text{Log}[\lambda]}{\lambda}$$

To obtain the MLE of $\lambda$, we use `FindMaximum` to numerically maximise `obslogLλ`.[1] For example:

```
solλ = FindMaximum[obslogLλ, {λ, {0.1, 1}}]

{415.987, {λ → 0.218573}}
```

The output states that the ML estimate of $\lambda$ is 0.218573, and that the maximised value of the observed log-likelihood is 415.987. Here is a plot of the data overlaid with the fitted model:

```
FrequencyPlot[xdata, f /. solλ[[2]]];
```

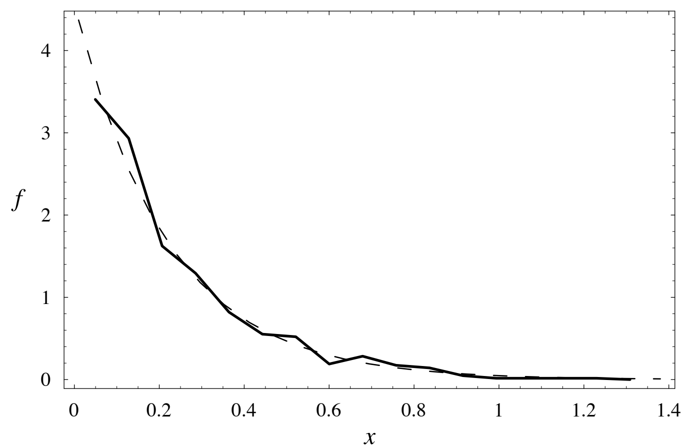



**Fig. 2:** Nerve data (——) and fitted Exponential model (– – –)

The Exponential model yields a close fit to the data, except in the neighbourhood of zero where the fit over-predicts. In the next example, we specify a more general model in an attempt to overcome this weakness. ■

⊕ ***Example 2:*** `FindMaximum` — Part II

Assume that $X \sim \text{Gamma}(\alpha, \beta)$, with pdf $f(x; \alpha, \beta)$:

```
f = x^(α-1) ⅇ^(-x/β) / (Γ[α] β^α);     domain[f] = {x, 0, ∞} && {α > 0, β > 0};
```

where $\alpha \in \mathbb{R}_+$ and $\beta \in \mathbb{R}_+$. ML estimation of the parameter $\theta = (\alpha, \beta)$ proceeds in two steps. First, we obtain a closed form solution for either $\hat{\alpha}$ or $\hat{\beta}$ in terms of the other parameter (*i.e.* we can obtain either $\hat{\alpha}(\beta)$ or $\hat{\beta}(\alpha)$). We then estimate the remaining parameter using the appropriate concentrated log-likelihood via numerical methods (`FindMaximum`).

The log-likelihood $\log L(\alpha, \beta)$ is:

```
logLθ = Log[∏_{i=1}^n (f /. x → x_i)]
```

$$-\frac{1}{\beta}\left(n\,\beta\,(\alpha\,\text{Log}[\beta] + \text{Log}[\Gamma[\alpha]]) + (\beta - \alpha\,\beta)\sum_{i=1}^{n}\text{Log}[x_i] + \sum_{i=1}^{n} x_i\right)$$

The score vector $\frac{\partial}{\partial\theta}\log L(\theta)$ is derived using **mathStatica**'s `Grad` function:

```
score = Grad[logLθ, {α, β}]
```

$$\left\{-n\,(\text{Log}[\beta] + \text{PolyGamma}[0, \alpha]) + \sum_{i=1}^{n}\text{Log}[x_i],\ \frac{-n\,\alpha\,\beta + \sum_{i=1}^{n} x_i}{\beta^2}\right\}$$

The ML estimator of $\alpha$ in terms of $\beta$ is obtained as:

```
solα = Solve[score〚2〛 == 0, α] // Flatten
```

$$\left\{\alpha \to \frac{\sum_{i=1}^{n} x_i}{n\,\beta}\right\}$$

That is,

$$\hat{\alpha}(\beta) = \frac{1}{n\,\beta}\sum_{i=1}^{n} X_i.$$

Substituting this solution into the log-likelihood yields the concentrated log-likelihood $\log L(\hat{\alpha}(\beta), \beta)$, which we denote `logLc`:

```
logLc = logLθ /. solα
```

$$-\frac{1}{\beta}\left(\sum_{i=1}^{n} x_i + \left(\sum_{i=1}^{n} \mathrm{Log}[x_i]\right)\left(\beta - \frac{\sum_{i=1}^{n} x_i}{n}\right) + n\,\beta\left(\mathrm{Log}\left[\Gamma\left[\frac{\sum_{i=1}^{n} x_i}{n\,\beta}\right]\right] + \frac{\mathrm{Log}[\beta]\sum_{i=1}^{n} x_i}{n\,\beta}\right)\right)$$

Next, we substitute the data into the concentrated log-likelihood:

```
obslogLc = logLc /. {n → Length[xdata], x_i_ :→ xdata⟦i⟧};
```

Then, we estimate $\beta$ using `FindMaximum`:

```
solβ = FindMaximum[obslogLc, {β, {0.1, 1}}]⟦2⟧
```

```
{β → 0.186206}
```

For the Nerve data, and assuming $X \sim \text{Gamma}(\alpha, \beta)$, the ML estimate of $\beta$ is $\hat{\beta} = 0.186206$. Therefore, the ML estimate of $\alpha$, $\hat{\alpha}(\hat{\beta})$, is:

```
solα /.
 Flatten[{solβ, n → Length[xdata], x_i_ :→ xdata⟦i⟧}]
```

```
{α → 1.17382}
```

Here is a plot of the data overlaid by the fitted model:

```
FrequencyPlot[xdata, f /. {α → 1.17382, β → 0.186206}];
```

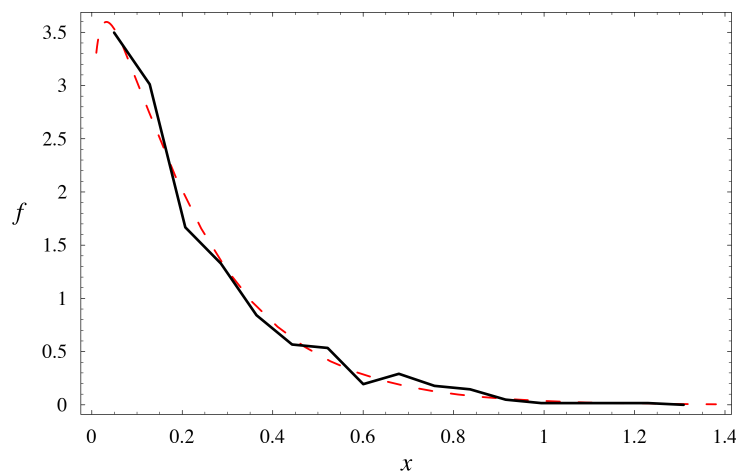



**Fig. 3:** Nerve data (——) and fitted Gamma model (– – –)

The Gamma model (see Fig. 3) achieves a better fit than the Exponential model (see Fig. 2), especially in the neighbourhood of zero. ■

## 12.3 A Journey with `FindMaximum`

In this section, we take a closer look at the performance of `FindMaximum`. This is done in the context of a statistical model that has become popular amongst analysts of financial time series data — the so-called autoregressive conditional heteroscedasticity model (ARCH model). Originally proposed by Engle (1982), the ARCH model is designed for situations in which the variance of a random variable seems to alternate between periods of relative stability and periods of pronounced volatility. We will consider only the simplest member of the ARCH suite, known as the ARCH(1) model.

Load the following data:

```
pdata = ReadList["BML.dat"];
```

The data lists the daily closing price (in Australian dollars) of Bank of Melbourne shares on the Australian Stock Exchange from October 30, 1996, until October 10, 1997 (non-trading days have been removed). Figure 4 illustrates the data.

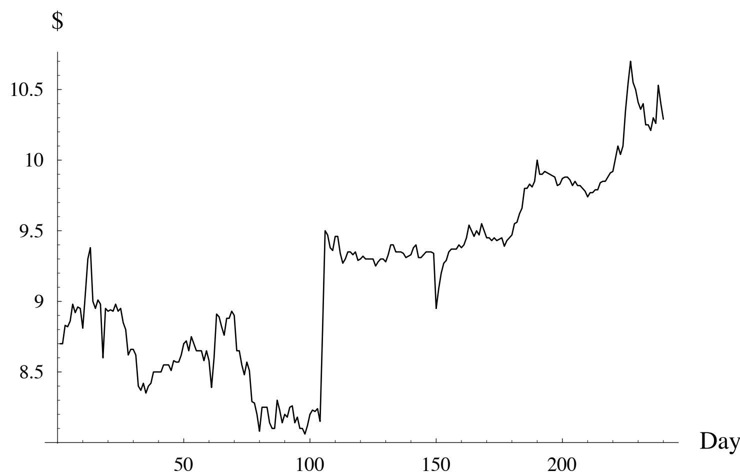



**Fig. 4:** The Bank of Melbourne data

Evidently, there are two dramatic increases in price: +\$0.65 on day 105, and +\$0.70 on day 106. These movements were caused by a takeover rumour that swept the market on those days, which was officially confirmed by the bank during day 106. Further important dates in the takeover process included: day 185, when approval was granted by the government regulator; day 226, when complete details of the financial offer were announced to shareholders; day 231, when shareholders voted to accept the offer; and day 240, the bank's final trading day.

Our analysis begins by specifying a variant of the random walk with drift model (see *Example 4* in Chapter 11) which, as we shall see upon examining the estimated residuals, leads us to specify an ARCH model later to improve fit. Let variable $P_t$ denote the closing price on day $t$, and let $\vec{x}_t$ denote a vector of regressors. Then, the random walk model we consider is

$$\Delta P_t = \vec{x}_t . \beta + U_t \tag{12.1}$$

where $\Delta P_t = P_t - P_{t-1}$, and the notation $\vec{x}_t . \beta$ indicates the dot product between the vectors $\vec{x}_t$ and $\beta$. We assume $U_t \sim N(0, \sigma^2)$; thus, $\Delta P_t \sim N(\vec{x}_t . \beta, \sigma^2)$. For this example, we specify a model with five regressors for vector $\vec{x}_t$, all of which are dummy variables: they consist of a constant intercept (the drift term), and day-specific intercept dummies corresponding to the takeover, the regulator, the disclosure and the vote. We denote the regression function by

$$\vec{x}_t . \beta = \beta_1 + x_2 \beta_2 + x_3 \beta_3 + x_4 \beta_4 + x_5 \beta_5 .$$

For all $n$ observations, we enter the price change:

```
Δp = Drop[pdata, 1] - Drop[pdata, -1];
```

and then the regressors: `x2` for the takeover, `x3` for the regulator, `x4` for the disclosure and `x5` for the vote:

```
x2 = x3 = x4 = x5 = Table[0, {239}];
x2⟦104⟧ = x2⟦105⟧ = x3⟦184⟧ = x4⟦225⟧ = x5⟦230⟧ = 1;
```

Note that the estimation period is from day 2 to day 240; hence, the reduction of 1 in the day-specific dummies. The statistical model (12.1) is in the form of a *Normal linear regression model*. To estimate the parameters of our model, we apply the `Regress` function given in *Mathematica*'s `Statistics`LinearRegression`` package. The `Regress` function is built using an ordinary least squares (OLS) estimator. The differences between OLS and ML estimates of the parameters of our model are minimal.[2] To use `Regress`, we must first load the Statistics add-on:

```
<< Statistics`
```

and then manipulate the data to the required format:

```
rdata = Transpose[{x2, x3, x4, x5, Δp}];
```

The estimation results are collected in `olsθ`:

```
olsθ = Regress[rdata,
   {takeover, regulator, disclosure, vote},
   {takeover, regulator, disclosure, vote},
   RegressionReport → {ParameterTable,
     EstimatedVariance, FitResiduals}];
```

Table 1 lists the OLS estimates of the parameters $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$; the estimates correspond to the coefficients of drift (labelled 1) and the day-specific dummies (labelled takeover, regulator, disclosure and vote).

| | Estimate | SE | TStat |
|---|---|---|---|
| 1 | −0.000171 | 0.0059496 | −0.02873 |
| takeover | 0.675171 | 0.0646293 | 10.44680 |
| regulator | 0.140171 | 0.0912057 | 1.53687 |
| disclosure | 0.190171 | 0.0912057 | 2.08508 |
| vote | −0.049829 | 0.0912057 | −0.54633 |
| $\sigma^2$ | 0.008283 | | |

**Table 1:** OLS estimates of the Random Walk with Drift model

Notice that the only regressors to have *t*-statistics that exceed 2 in absolute value are the takeover and disclosure day-specific dummies. Figure 5 plots the time series of fitted residuals.

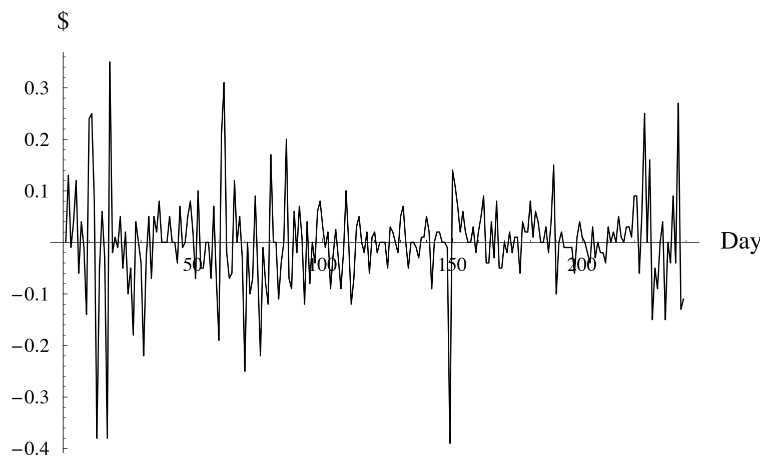



**Fig. 5:** Fitted OLS residuals

The residuals exhibit clusters of variability (approximately, days 2–30, 60–100, 220–240) interspersed with periods of stability (day 150 providing an exception to this). This suggests that an ARCH specification for the model disturbance $U_t$ may improve the fit of (12.1); for details on formal statistical testing procedures for ARCH disturbances, see Engle (1982).

To specify an ARCH(1) model for the disturbances, we extend (12.1) to

$$\Delta P_t = \vec{x}_t . \beta + U_t \tag{12.2}$$

$$U_t = V_t \sqrt{\alpha_1 + \alpha_2 U_{t-1}^2} \tag{12.3}$$

where $V_t \sim N(0, 1)$. We now deduce conditional moments of the disturbance $U_t$ holding $U_{t-1}$ fixed at a specific value $u_{t-1}$. The conditional mean and variance of $U_t$ are $E[U_t \mid U_{t-1} = u_{t-1}] = 0$ and $\text{Var}(U_t \mid U_{t-1} = u_{t-1}) = \alpha_1 + \alpha_2 u_{t-1}^2$, respectively. These results imply that $\Delta P_t \mid (U_{t-1} = u_{t-1}) \sim N(\vec{x}_t . \beta, \alpha_1 + \alpha_2 u_{t-1}^2)$. The likelihood function is

the product of the distribution of the initial condition and the conditional distributions; the theory behind this construction is similar to that discussed in *Example 4* of Chapter 11. Given the initial condition $U_0 = 0$, the likelihood is

$$L(\theta) = \frac{1}{\sqrt{2\pi\alpha_1}} \exp\left( \frac{-(\Delta p_1 - \vec{x}_1 . \beta)^2}{2\alpha_1} \right) \times \prod_{t=2}^{n} \frac{1}{\sqrt{2\pi(\alpha_1 + \alpha_2 u_{t-1}^2)}} \exp\left( \frac{-(\Delta p_t - \vec{x}_t . \beta)^2}{2(\alpha_1 + \alpha_2 u_{t-1}^2)} \right) \tag{12.4}$$

where $\Delta p_t$ is the datum observed on $\Delta P_t$ and $u_t = \Delta p_t - \vec{x}_t . \beta$, for $t = 1, \ldots, n$. We now enter the log-likelihood into *Mathematica*. It is convenient to express the log-likelihood in terms of $u_t$:

```
Clear[n];
logLθ =
  FullSimplify[Log[ Exp[-u1^2/(2 α1)]/Sqrt[2 π α1] ∏_{t=2}^{n} Exp[-ut^2/(2 (α1 + α2 u_{t-1}^2))]/Sqrt[2 π (α1 + α2 u_{t-1}^2)] ],
    {u1 ∈ Reals, α1 > 0}] // Expand
```

$$-\frac{1}{2} \text{n Log}[2\pi] - \frac{\text{Log}[\alpha 1]}{2} - \frac{u_1^2}{2\,\alpha 1} - \frac{1}{2}\sum_{t=2}^{n} \text{Log}[\alpha 1 + \alpha 2\, u_{-1+t}^2] - \frac{1}{2}\sum_{t=2}^{n} \frac{u_t^2}{\alpha 1 + \alpha 2\, u_{-1+t}^2}$$

To obtain the observed log-likelihood, we first enter in the value of $n$; we then re-define the regressors in $\vec{x}$, reducing the number of regressors from five down to just the two significant regressors (takeover and disclosure) from the random walk model fitted previously:

```
n = 239;    xdata = Transpose[{x2, x4}];
```

Next, we enter the disturbances $u_t$ defined, via (12.2), as $U_t = \Delta P_t - \vec{x}_t . \beta$:

```
uvec = Δp - xdata.{β2, β4};
```

Finally, we create a set of replacement rules called `urules`:[3]

```
urules = Table[u_i → uvec[[i]], {i, n}];     Short[urules]

{u1 → 0., u2 → 0.13, <<236>>, u239 → -0.11}
```

Substituting `urules` into the log-likelihood yields the observed log-likelihood:

```
obslogLθ = logLθ /. urules;
```

Note that our *Mathematica* inputs use the parameter notation (β2, β4, α1, α2) rather than the neater subscript form $(\beta_2, \beta_4, \alpha_1, \alpha_2)$; this is because `FindMinimum` / `FindMaximum` does not handle subscripts well.[4]

We undertake maximum likelihood estimation of the parameters $(\beta_2, \beta_4, \alpha_1, \alpha_2)$ with `FindMaximum`. To begin, we apply it blindly, selecting as initial values for $(\beta_2, \beta_4)$ the estimates from the random walk model, and choosing arbitrary initial values for $\alpha_1$ and $\alpha_2$:

```
sol = FindMaximum[obslogLθ,
  {β2, .675171}, {β4, .190171}, {α1, 1}, {α2, 1}]
```

```
— FindMinimum::fmnum :
   Objective function 122.878 + 375.42 ⅈ is not real at
     {β2, β4, α1, α2} = {0.675165, 0.190167, -≪19≫, 0.988813}.
— FindMinimum::fmnum :
   Objective function 122.878 + 375.42 ⅈ is not real at
     {β2, β4, α1, α2} = {0.675165, 0.190167, -≪19≫, 0.988813}.
— FindMinimum::fmnum :
   Objective function 122.878 + 375.42 ⅈ is not real at
     {β2, β4, α1, α2} = {0.675165, 0.190167, -≪19≫, 0.988813}.
— General::stop : Further output of FindMinimum::fmnum will
     be suppressed during this calculation.
```

Why has it crashed? Our first clue comes from the error message, which tells us that the observed log-likelihood ‘is not real’ for some set of values assigned to the parameters. Of course, all log-likelihoods *must* be real-valued at all points in the *parameter space*, so the problem must be that `FindMaximum` has drifted outside the parameter space. Indeed, from the error message we see that $\alpha_1$ has become negative, which may in turn cause the conditional variance, $\mathrm{Var}(\Delta P_t \mid (U_{t-1} = u_{t-1})) = \alpha_1 + \alpha_2\, u_{t-1}^2$ to become negative, causing *Mathematica* to report a complex value for $\log(\alpha_1 + \alpha_2\, u_{t-1}^2)$. It is easy to see that if $\alpha_2 = 0$, the ARCH model, (12.2) and (12.3), reduces to the random walk model (12.1) in which case $\alpha_1 = \sigma^2$, so we require $\alpha_1 > 0$. Similarly, we must insist on $\alpha_2 \geq 0$. Finally, Engle (1982, Theorem 1) derives an upper bound for $\alpha_2$ which must hold if higher order even moments of the ARCH(1) process are to exist. Imposing $\alpha_2 < 1$ ensures that the unconditional variance, $\mathrm{Var}(U_t)$, exists.

In order to obtain the ML estimates, we need to incorporate the parameter restrictions $\alpha_1 > 0$ and $0 \leq \alpha_2 < 1$ into *Mathematica*. There are two possibilities open to us:

(i) to use `FindMaximum` as a constrained optimiser, or

(ii) to re-parameterise the observed log-likelihood function so that the constraints are not needed.

For approach (i), we implement `FindMaximum` with the constraints entered at the command line; for example, we might enter:

```
sol1 = FindMaximum[obslogLθ, {β2, .675171}, {β4, .190171},
  {α1, 1, 0.00001, 100}, {α2, 0.5, 0, 1}, MaxIterations → 40]
```

```
{243.226, {β2 → 0.693842,
  β4 → 0.191731, α1 → 0.00651728, α2 → 0.192958}}
```

In this way, FindMinimum / FindMaximum is being used as a *constrained* optimiser. The constraints entered above correspond to $0.00001 \leq \alpha_1 \leq 100$ and $0 \leq \alpha_2 \leq 1$. Also, note that we had to increase the maximum possible number of iterations to 40 (10 more than the default) to enable FindMinimum / FindMaximum to report convergence. Unfortunately, FindMinimum / FindMaximum often encounters difficulties when parameter constraints are entered at the command line.

Approach (ii) improves on the previous method by re-parameterising the observed log-likelihood in such a way that the constraints are eliminated. In doing so, FindMinimum / FindMaximum is implemented as an *unconstrained* optimiser, which is a task it can cope with. Firstly, the constraint $\alpha_1 > 0$ is satisfied for all real $\gamma_1$ provided $\alpha_1 = e^{\gamma_1}$. Secondly, the constraint $0 \leq \alpha_2 < 1$ is (almost) satisfied for all real $\gamma_2$ provided $\alpha_2 = (1 + \exp(\gamma_2))^{-1}$. A replacement rule is all that is needed to re-parameterise the observed log-likelihood:

```
obslogLλ = obslogLθ /. {α1 → e^γ1, α2 → 1/(1 + e^γ2)};
```

We now attempt:

```
sol2 = FindMaximum[obslogLλ,
  {β2, .675171}, {β4, .190171}, {γ1, 0}, {γ2, 0}]
```

```
{243.534, {β2 → 0.677367,
  β4 → 0.305868, γ1 → -5.07541, γ2 → 1.02915}}
```

The striking feature of this result is that even though the starting points of this and our earlier effort are effectively the same, the maximised value of the observed log-likelihood yielded by the current solution sol2 is strictly superior to that of the former sol1:

```
sol2[[1]] > sol1[[1]]
```

```
True
```

It would, however, be unwise to state unreservedly that sol2 represents the ML estimates! In practice, it is advisable to experiment with different starting values. Suppose, for example, that the algorithm is started from a different location in the parameter space:

```
sol3 = FindMaximum[obslogLλ,
  {β2, .675171}, {β4, .190171}, {γ1, -5}, {γ2, 0}]
```

```
{243.534, {β2 → 0.677263,
  β4 → 0.305021, γ1 → -5.07498, γ2 → 1.03053}}
```

This solution is slightly better than the former one, the difference being detectable at the $5^{\text{th}}$ decimal place:

```
NumberForm[sol2[[1]], 9]
NumberForm[sol3[[1]], 9]
```

```
243.53372

243.533752
```

Nevertheless, we observe that the parameter estimates output from both runs are fairly close, so it seems reasonable enough to expect that `sol3` is in the *neighbourhood* of the solution.[5]

There are still two features of the proposed solution that need to be checked, and these concern the gradient:

```
g = Grad[obslogLλ, {β2, β4, γ1, γ2}];
g /. sol3⟦2⟧
```

```
{0.0553552, 0.000195139, 0.0116302, -0.000123497}
```

and the Hessian:

```
h = Hessian[obslogLλ, {β2, β4, γ1, γ2}];
Eigenvalues[h /. sol3⟦2⟧]
```

```
{-359.682, -96.3175, -79.1461, -2.60905}
```

The gradient at the maximum (or minimum or saddle point) should disappear — but this is far from true here. It would therefore be a mistake to claim that `sol3` is the ML estimate! On the other hand, all eigenvalues at `sol3` are negative, so the observed log-likelihood is concave in this neighbourhood. This is useful information, as we shall see later on. For now, let us return to the puzzle of the non-zero gradient!

Why does `FindMinimum` / `FindMaximum` fail to detect a non-zero gradient at what it claims is the optimum? The answer lies with the algorithm's stopping rule. Quite clearly, `FindMinimum` / `FindMaximum` does not check the magnitude of the gradient, for if it did, further iterations would be performed. So what criterion does `FindMinimum` use in deciding whether to stop or proceed to a new iteration? After searching the documentation on `FindMinimum`, the criterion is not revealed. So, at this stage, our answer is incomplete; we can only say for certain what criterion is *not* used. Perhaps, like many optimisers, `FindMinimum` iterates until the improvement in the objective function is smaller than some critical number? Alternatively, perhaps `FindMinimum` iterates until the absolute change in the choice variables is smaller than some critical value? Further discussion of stopping rule criteria appears in §12.5.

Our final optimisation assault utilises the fact that, at `sol3` (our current best 'solution'), we have reached a neighbourhood of the parameter space in which the observed log-likelihood is concave, since the eigenvalues of the Hessian matrix are negative at `sol3`. In practice, it is nearly always advisable to 'finish off' an optimisation with iterations of the Newton–Raphson algorithm, provided it is computationally feasible to do so. This algorithm can often be costly to perform, for it requires computation of the Hessian matrix at each iteration, but this is exactly where *Mathematica* comes into its own because it is a wonderful differentiator! And for our particular problem, provided that we do not print it to screen, the Hessian matrix takes less than no time for *Mathematica* to compute — as we have already witnessed when it was computed for the re-parameterised observed log-likelihood and stored as `h`. The Newton–Raphson algorithm can be run by supplying an option to `FindMaximum`. Starting our search at `sol3`, we find:

```
sol4 = FindMaximum[obslogLλ,
                        {β2, 0.677263}, {β4, 0.305021},
                        {γ1, -5.07498}, {γ2, 1.03053},
                                              Method → Newton]
```

```
{243.534, {β2 → 0.677416,
  β4 → 0.304999, γ1 → -5.07483, γ2 → 1.03084}}
```

Not much appears to have changed in going from `sol3` to `sol4`. The value of the observed log-likelihood increases slightly at the 6th decimal place:

```
NumberForm[sol3⟦1⟧, 10]
NumberForm[sol4⟦1⟧, 10]
```

```
243.5337516

243.5337567
```

which necessarily forces us to replace `sol3` with `sol4`, the latter now being a possible contender for the maximum. The parameter estimates alter slightly too:

```
sol3⟦2⟧
```

```
{β2 → 0.677263, β4 → 0.305021,
 γ1 → -5.07498, γ2 → 1.03053}
```

```
sol4⟦2⟧
```

```
{β2 → 0.677416, β4 → 0.304999,
 γ1 → -5.07483, γ2 → 1.03084}
```

But what about our concerns over the gradient and the Hessian?

```
g /. sol4⟦2⟧
```

```
{0.0000131549, -6.90917×10^-7,
 4.09054×10^-6, 3.40381×10^-7}
```

```
Eigenvalues[h /. sol4⟦2⟧]
```

```
{-359.569, -96.3068, -79.1165, -2.60887}
```

Wonderful! All elements of the gradient are numerically much closer to zero, and the eigenvalues of the Hessian matrix are all negative, indicating that it is negative definite. `FindMinimum / FindMaximum` has, with some effort on our part, successfully navigated its way through the numerical optimisation maze and presented to us the point estimates that maximise the re-parameterised observed log-likelihood. However, our work is not yet finished! The ML estimates of the parameters of the original ARCH(1) model must be determined:

$$\left\{\beta 2,\ \beta 4,\ e^{\gamma 1},\ \frac{1}{1+e^{\gamma 2}}\right\}\ /.\ \text{sol4}[[2]]$$

```
{0.677416, 0.304999, 0.00625216, 0.262921}
```

We conclude our 'journey' by presenting the ML estimates in Table 2.

| | Estimate |
|---|---|
| takeover | 0.677416 |
| disclosure | 0.304999 |
| $\alpha_1$ | 0.006252 |
| $\alpha_2$ | 0.262921 |

**Table 2:** ML estimates of the ARCH(1) model

# 12.4 Asymptotic Inference

Inference refers to topics such as hypothesis testing and diagnostic checking of fitted models, confidence interval construction, within-sample prediction, and out-of-sample forecasting. For statistical models fitted using ML methods, inference is often based on large sample results, as ML estimators (suitably standardised) have a limiting Normal distribution.

## 12.4 A Hypothesis Testing

Asymptotic inference is operationalised by replacing unknowns with consistent estimates. To illustrate, consider the Gamma($\alpha$, $\beta$) model, with mean $\mu = \alpha\beta$. Suppose we want to test

$$H_0 : \mu = \mu_0 \quad \text{against} \quad H_1 : \mu \neq \mu_0$$

where $\mu_0 \in \mathbb{R}_+$ is known. Letting $\hat{\mu}$ denote the ML estimator of $\mu$, we find (see *Example 4* for the derivation):

$$\sqrt{n}\,(\hat{\mu} - \mu) \xrightarrow{d} N(0, \alpha\,\beta^2).$$

Assuming $H_0$ to be true, we can (to give just two possibilities) base our hypothesis test on either of the following asymptotic distributions for $\hat{\mu}$:

$$\hat{\mu} \overset{a}{\sim} N\left(\mu_0, \frac{1}{n}\,\hat{\alpha}\hat{\beta}^2\right) \quad \text{or} \quad \hat{\mu} \overset{a}{\sim} N\left(\mu_0, \frac{1}{n}\,\mu_0\hat{\beta}\right).$$

Depending on which distribution is used, it is quite possible to obtain conflicting outcomes to the tests. The potential for arbitrary outcomes in asymptotic inference has, on occasion, 'ruffled the feathers' of those advocating that inference should be based on small sample performance!

⊕ ***Example 3:*** The Gamma or the Exponential?

In this example, we consider whether there is a statistically significant improvement in using the Gamma($\alpha$, $\beta$) model to fit the Nerve data (*Example 2*) when compared to the Exponential($\lambda$) model (*Example 1*). In a Gamma distribution, restricting the shape parameter $\alpha$ to unity yields an Exponential distribution; that is, Gamma$(1, \beta) =$ Exponential($\beta$). Hence, we shall conduct a hypothesis test of

$$H_0 : \alpha = 1 \quad \text{against} \quad H_1 : \alpha \neq 1.$$

We use the asymptotic theory of ML estimators to perform the test of $H_0$ against $H_1$. Here is the pdf of $X \sim$ Gamma($\alpha$, $\beta$):

```
f = x^(α-1) ⅇ^(-x/β) / (Γ[α] β^α);      domain[f] = {x, 0, ∞} && {α > 0, β > 0};
```

Since the MLE is regular (conditions 1a, 2, 3, 4a, and 5a are satisfied; see §11.4 D and §11.5 A),

$$\sqrt{n}\left(\hat{\theta} - \theta_0\right) \xrightarrow{d} N(0, i_0^{-1})$$

where $\hat{\theta}$ denotes the MLE of $\theta_0 = (\alpha, \beta)$. We can evaluate $i_0^{-1}$:

```
Inverse[FisherInformation[{α, β}, f]] // Simplify
```

$$\begin{pmatrix} \frac{\alpha}{-1+\alpha\,\text{PolyGamma}[1,\alpha]} & \frac{\beta}{1-\alpha\,\text{PolyGamma}[1,\alpha]} \\ \frac{\beta}{1-\alpha\,\text{PolyGamma}[1,\alpha]} & \frac{\beta^2\,\text{PolyGamma}[1,\alpha]}{-1+\alpha\,\text{PolyGamma}[1,\alpha]} \end{pmatrix}$$

Let $\sigma^2$ denote the top left element of $i_0^{-1}$; note that $\sigma^2$ depends only on $\alpha$. From the (joint) asymptotic distribution of $\hat{\theta}$, we find

$$\hat{\alpha} \overset{a}{\sim} N\left(\alpha, \tfrac{1}{n}\sigma^2\right).$$

We may base our test statistic on this asymptotic distribution for $\hat{\alpha}$, for when $\alpha = 1$ (*i.e.* $H_0$ is true), it has mean 1, and standard deviation:

```
s = Sqrt[(1/n) α/(-1 + α PolyGamma[1, α])] /. {α → 1, n → 799} // N

0.0440523
```

Because the alternative hypothesis $H_1$ is uninformative (two-sided), $H_0$ will be rejected if the observed value of $\hat{\alpha}$ (1.17382 was obtained in *Example 2*) is either much larger than unity, or much smaller than unity. The $p$-value (short for 'probability value'; see, for example, Mittelhammer (1996, pp. 535–538)) for the test is given by

$$P\left(\left|\hat{\alpha}-1\right| > 1.17382-1\right) = 1 - P(0.82618 < \hat{\alpha} < 1.17382)$$

which equals:

```
g = 1/(s Sqrt[2 π]) Exp[-(α̂ - 1)^2/(2 s^2)];     domain[g] = {α̂, -∞, ∞};

1 - (Prob[1.17382, g] - Prob[0.82618, g])
```

```
0.0000795469
```

As the $p$-value is very small, this is strong evidence against $H_0$. ■

⊕ ***Example 4:*** Constructing a Confidence Interval

In this example, we construct an approximate confidence interval for the mean $\mu = \alpha\beta$ of the Gamma($\alpha$, $\beta$) distribution using an asymptotic distribution for the MLE of the mean.

From the previous example, we know $\sqrt{n}\left(\hat{\theta}-\theta_0\right) \xrightarrow{d} N(0, i_0^{-1})$, where $\hat{\theta}$ is the MLE of $\theta_0 = (\alpha, \beta)$. As $\mu$ is a function of the elements of $\theta_0$, we may apply the Invariance Property (see §11.4 E) to find

$$\sqrt{n}\,(\hat{\mu}-\mu) \xrightarrow{d} N\left(0, \frac{\partial\mu}{\partial\theta_0^{\mathrm{T}}} \times i_0^{-1} \times \frac{\partial\mu}{\partial\theta_0}\right).$$

**mathStatica** derives the variance of the limit distribution as:

```
f = x^(α-1) e^(-x/β)/(Γ[α] β^α);     domain[f] = {x, 0, ∞} && {α > 0, β > 0};

Grad[α β, {α, β}].Inverse[FisherInformation[{α, β}, f]].
  Grad[α β, {α, β}] // Simplify
```

$\alpha\,\beta^2$

Consequently, we may write $\sqrt{n}\,(\hat{\mu}-\mu) \overset{a}{\sim} N(0, \alpha\beta^2)$. Unfortunately, a confidence interval for $\mu$ cannot be constructed from this asymptotic distribution, due to the presence of the unknown parameters $\alpha$ and $\beta$. However, if we replace $\alpha$ and $\beta$ with, respectively, the estimates $\hat{\alpha}$ and $\hat{\beta}$, we find[6]

$$\hat{\mu} \overset{a}{\sim} N\left(\mu, \frac{1}{n}\,\hat{\alpha}\hat{\beta}^2\right).$$

From this asymptotic distribution, an approximate $100\,(1-\omega)\,\%$ confidence interval for $\mu$ can be constructed; it is given by

$$\hat{\mu} \pm z_{1-\omega/2}\sqrt{\hat{\alpha}\hat{\beta}^2/n}$$

where $z_{1-\omega/2}$ is the inverse cdf of the $N(0, 1)$ distribution evaluated at $1 - \omega/2$.

For the Nerve data of *Example 2*, with ML estimates of 1.17382 for $\alpha$, and 0.186206 for $\beta$, an approximate 95% confidence interval for $\mu$ is:[7]

```
α̂ = 1.17382;     β̂ = 0.186206;     μ̂ = α̂ β̂;

z = √2 InverseErf[0, -1 + 2 (1 - 0.05/2)];

{μ̂ - z √(α̂ β̂² / 799), μ̂ + z √(α̂ β̂² / 799)}

{0.204584, 0.232561}
```

## 12.4 B Standard Errors and *t*-statistics

When reporting estimation results, it is important to mention, at the very least, the estimates, the standard errors of the estimators, and the *t*-statistics (*e.g.* see Table 1). For ML estimation, such details can be obtained from an asymptotic distribution for the estimator. It is insufficient to present just the parameter estimates. This, for example, occurred for the ARCH model estimated in §12.3, where standard errors and *t*-statistics were not presented (see Table 2). This is because `FindMinimum` / `FindMaximum` only returns point estimates of the parameters, and the optimised value of the observed log-likelihood. To report standard errors and *t*-statistics, further programming must be done.

For regular ML estimators such that

$$\sqrt{n}\,\left(\hat{\theta} - \theta_0\right) \xrightarrow{d} N(0,\, i_0^{-1})$$

with an asymptotic distribution:

$$\hat{\theta} \overset{a}{\sim} N\!\left(\theta_0,\, (n\, i_0)^{-1}\right)$$

we require a consistent estimator of the matrix $(n\, i_0)^{-1}$ in order to operationalise asymptotic inference, and to report estimation results. Table 3 lists three such estimators.

| | |
|---|---|
| Fisher | $\left(n\, i_{\hat{\theta}}\right)^{-1}$ |
| Hessian | $\left(-\frac{\partial^2}{\partial\theta\, \partial\theta^{\mathrm{T}}} \log\, L(\hat{\theta})\right)^{-1}$ |
| Outer-product | $\left(\sum_{i=1}^{n} \left(\frac{\partial}{\partial\theta} \log f\left(X_i; \hat{\theta}\right)\right)\left(\frac{\partial}{\partial\theta} \log f\left(X_i; \hat{\theta}\right)\right)^{\mathrm{T}}\right)^{-1}$ |

**Table 3:** Three asymptotically equivalent estimators of $(n\, i_0)^{-1}$

Each estimator relies on the consistency of the MLE $\hat{\theta}$ for $\theta_0$. All three are asymptotically equivalent in the sense that $n$ times each estimator converges in probability to $i_0^{-1}$. The first estimator, labelled ‘Fisher’, was used in *Example 4*. The second, ‘Hessian’, is based on regularity condition 5a(ii) (see §11.5 A). This estimator is quite popular in practice, having the advantage over the Fisher estimator that it does *not* require solving an expectation. The ‘Outer-product’ estimator is based on the definition of Fisher Information (see §10.2 D, and condition 4a in §11.5 A). While it would appear more complicated than the others, it can come in handy if computation of the Hessian estimator becomes costly, for it requires only one round of differentiation.

If the MLE in a *non-identically distributed* sample (see §11.5 B) is such that,

$$\sqrt{n}\,\left(\hat{\theta} - \theta_0\right) \xrightarrow{d} N\left(0,\, (i_0^{(\infty)})^{-1}\right)$$

then to operationalise asymptotic inference, the Hessian and Outer-product estimators given in Table 3 may be used to estimate $(n\, i_0^{(\infty)})^{-1}$; however, the Fisher estimator is now $I_{\hat{\theta}}^{-1}$, where $I_\theta$ denotes the Sample Information on $\theta$ (see §10.2 E).

⊕ ***Example 5:*** Income and Education: An Exponential Regression Model

In *Example 15* of Chapter 11, we considered the simple Exponential regression model:

$$Y \,\big|\, (X = x) \sim \text{Exponential}(\exp(\alpha + \beta\, x)) \tag{12.5}$$

where regressor $X = x \in \mathbb{R}$, and parameter $\theta = (\alpha, \beta) \in \mathbb{R}^2$. Here is the pdf $f(y \,|\, X = x; \theta)$:

```
f = 1/Exp[α + β x] Exp[-y/Exp[α + β x]];

domain[f] = {y, 0, ∞} && {α ∈ Reals, β ∈ Reals, x ∈ Reals};
```

Greene (2000, Table A4.1) gives hypothetical data on the pair $(Y_i, X_i)$ for $n = 20$ individuals, where $Y$ denotes Income ($000s per annum) and $X$ denotes years of Education. Here is the Income data:

```
Income = {20.5, 31.5, 47.7, 26.2, 44.0, 8.28,
   30.8, 17.2, 19.9, 9.96, 55.8, 25.2, 29.0,
   85.5, 15.1, 28.5, 21.4, 17.7, 6.42, 84.9};
```

… and here is the Education data:

```
Education = {12, 16, 18, 16, 12, 12, 16, 12,
   10, 12, 16, 20, 12, 16, 10, 18, 16, 20, 12, 16};
```

Figure 6 illustrates the data in the form of a scatter diagram.

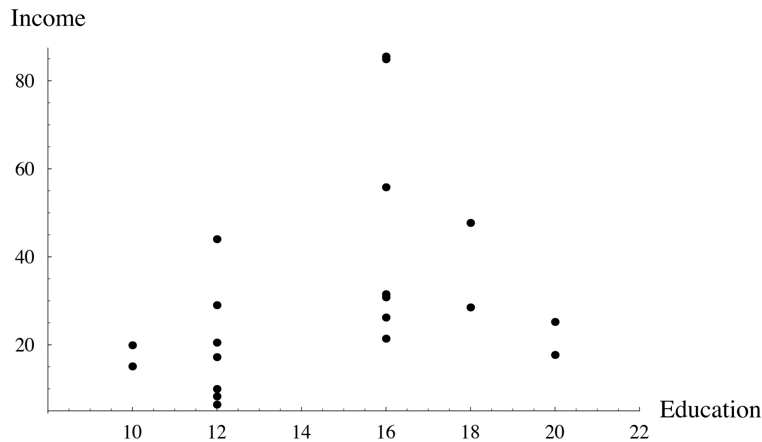



**Fig. 6:** The Income–Education data

Using ML methods, we fit the Exponential regression model (12.5) to this data. We begin by entering the observed log-likelihood:

```
obslogLθ = Log[∏_{i=1}^{n} (f /. {y → y_i, x → x_i})] /.
   {n → 20, y_i_ :→ Income⟦i⟧, x_i_ :→ Education⟦i⟧}
```

$$-42.9\, e^{-\alpha-20\beta} - 76.2\, e^{-\alpha-18\beta} - 336.1\, e^{-\alpha-16\beta} - 135.36\, e^{-\alpha-12\beta} - 35.\, e^{-\alpha-10\beta} - 20\,\alpha - 292\,\beta$$

We obtain the ML estimates using `FindMaximum`'s Newton–Raphson algorithm:

```
solθ = FindMaximum[obslogLθ, {α, 0.1}, {β, 0.2},
                    Method → Newton]
```

```
{-88.1034, {α → 1.88734, β → 0.103961}}
```

Thus, the observed log-likelihood is maximised at a value of $-88.1034$, with ML estimates of $\alpha$ and $\beta$ reported as 1.88734 and 0.103961, respectively.

Next, we compute the Fisher, Hessian and Outer-product estimators given in Table 3. The Fisher estimator corresponds to the inverse of the $(2\times2)$ Sample Information matrix derived in *Example 15* of Chapter 11. It is given by:

```
Fisher = Inverse[
   ( n              Σ_{i=1}^{n} x_i   )
   ( Σ_{i=1}^{n} x_i  Σ_{i=1}^{n} x_i^2 ) /. {n → 20, x_i_ :→ Education⟦i⟧}] // N
```

$$\begin{pmatrix} 1.20346 & -0.0790043 \\ -0.0790043 & 0.00541126 \end{pmatrix}$$

The Hessian estimator is easily computed using **mathStatica**'s `Hessian` function:

```
hessian = Inverse[-Hessian[obslogLθ, {α, β}] /. solθ⟦2⟧]
```

$$\begin{pmatrix} 1.54467 & -0.102375 \\ -0.102375 & 0.00701196 \end{pmatrix}$$

Calculating the Outer-product estimator is more involved, so we evaluate it in four stages. First, we calculate the score $\frac{\partial}{\partial\theta} \log f(x_i; \theta)$ using **mathStatica**'s `Grad` function:

```
grad = Grad[Log[f /. {y → yi, x → xi}], {α, β}]
```

$$\{-1 + e^{-\alpha-\beta x_i} y_i,\ x_i\,(-1 + e^{-\alpha-\beta x_i} y_i)\}$$

Next, we form the outer product of this vector with itself — the distinctive operation that gives the estimator its name:

```
op = Outer[Times, grad, grad]
```

$$\begin{pmatrix} (-1 + e^{-\alpha-\beta x_i} y_i)^2 & x_i\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 \\ x_i\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 & x_i^2\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 \end{pmatrix}$$

We then `Map` the sample summation operator across each element of this matrix (this is achieved by using the level specification `{2}`):

$$\textbf{opS = Map}\left[\sum_{i=1}^{n} \textbf{\# \&, op, \{2\}}\right]$$

$$\begin{pmatrix} \sum_{i=1}^{n} (-1 + e^{-\alpha-\beta x_i} y_i)^2 & \sum_{i=1}^{n} x_i\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 \\ \sum_{i=1}^{n} x_i\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 & \sum_{i=1}^{n} x_i^2\,(-1 + e^{-\alpha-\beta x_i} y_i)^2 \end{pmatrix}$$

Finally, we substitute the ML estimates and the data into `opS`, and then invert the resulting matrix:

```
outer = Inverse[opS /. Flatten[{solθ⟦2⟧,
      n → 20, yi_ :→ Income⟦i⟧, xi_ :→ Education⟦i⟧}]]
```

$$\begin{pmatrix} 5.34805 & -0.342767 \\ -0.342767 & 0.0225022 \end{pmatrix}$$

In this particular case, the three estimators yield different estimates of the asymptotic variance-covariance matrix (this generally occurs). The estimation results for the trio of estimators are given in Table 4.

| | | Fisher | | Hessian | | Outer | |
|---|---|---|---|---|---|---|---|
| | Estimate | SE | TStat | SE | TStat | SE | TStat |
| $\alpha$ | 1.88734 | 1.09702 | 1.72042 | 1.24285 | 1.51856 | 2.31259 | 0.81612 |
| $\beta$ | 0.10396 | 0.07356 | 1.41326 | 0.08374 | 1.24151 | 0.15001 | 0.69304 |

**Table 4:** Estimation results for the Income–Education data

The ML estimates appear in the first column (Estimate). The associated estimated asymptotic standard errors appear in the SE columns (these correspond to the square root of the elements of the leading diagonal of the estimated asymptotic variance-covariance matrices). The $t$-statistics are in the TStat columns (these correspond to the estimates divided by the estimated asymptotic standard errors; these are statistics for the tests of $H_0 : \alpha = 0$ and $H_0 : \beta = 0$). These results suggest that Education is not a significant explanator of Income, assuming model (12.5). ■

# 12.5 Optimisation Algorithms

## 12.5 A Preliminaries

The numerical optimisation of a function (along with the related task of solving for the roots of an equation) is a problem that has attracted considerable interest in many areas of science and technology. Mathematical statistics is no exception, for as we have seen, optimisation is fundamental to estimation, whether it be for ML estimation or for other estimation methods such as the method of moments or the method of least squares. Optimisation algorithms abound, as even a cursory glance through Polak's (1971) classic reference work will reveal. Some of these have been coded into `FindMinimum`, but for each one that has been implemented in that function, there are dozens of others omitted. Of course, the fact that there exist so many different types of algorithms is testament to the fact that every problem is unique, and its solution cannot necessarily be found by applying one algorithm. The various attempts at estimating the ARCH model in §12.3 provide a good illustration of this.

We want to solve two estimation problems. The first is to maximise a real, single-valued observed log-likelihood function with respect to the parameter $\theta$. The point estimate of $\theta_0$ is to be returned, where $\theta_0$ denotes (as always) the true parameter value. The second is to estimate the asymptotic standard errors of the parameter estimator and the asymptotic $t$-statistics. This can be achieved by returning, for example, the Hessian evaluated at the point estimate of $\theta_0$ (*i.e.* the Hessian estimator given in Table 3 in §12.4). It is fair to say that obtaining ML estimates is the more important task; however, the two taken together permit inference using the asymptotic distribution.

The algorithms that we discuss in this section address the dual needs of the estimation problem; in particular, we illustrate the *Newton–Raphson* (NR) and the *Broydon–Fletcher–Goldfarb–Shanno* (BFGS) algorithms. The NR and BFGS algorithms are options in `FindMinimum` using `Method→Newton` and `Method→QuasiNewton`, respectively. However, in its Version 4 incarnation, `FindMinimum`

returns only a point estimate of $\theta_0$; other important pieces of information such as the final Hessian matrix are not recoverable from its output. This is clearly a weakness of `FindMinimum` which will hopefully be rectified in a later version of *Mathematica*.

Both the NR and BFGS algorithms, and every other algorithm implemented in `FindMinimum`, come under the broad category of *gradient methods*. Gradient methods form the backbone of the literature on optimisation and include a multitude of approaches including quadratic hill-climbing, conjugate gradients, and quasi-Newton methods. Put simply, gradient methods work by estimating the gradient and Hessian of the observed log-likelihood at a given point, and then jumping to a superior solution which estimates the optimum. This process is then repeated until convergence. Amongst the extensive literature on optimisation using gradient methods, we refer in particular to Polak (1971), Luenberger (1984), Gill *et al*. (1981) and Press *et al*. (1992). A discussion of gradient methods applied to optimisation in a statistical context appears in Judge *et al*. (1985).

Alternatives to optimisation based on gradient methods include direct search methods, simulated annealing methods, taboo search methods, and genetic algorithms. The first — direct search — involves the adoption of a search pattern through parameter space comparing values of the observed log-likelihood at each step. Because it ignores information (such as gradient and Hessian), direct search methods are generally regarded as inferior. However, the others — simulated annealing, taboo search, and genetic algorithms — fare better and have much to recommend them. Motivation for alternative methods comes primarily from the fact that a gradient method algorithm is unable to escape from regions in parameter space corresponding to local optima, for once at a local optimum a gradient algorithm will not widen its search to find the global optimum — this is termed the problem of multiple local optima.[8]

The method of simulated annealing (Kirkpatrick *et al*. (1983)) attempts to overcome this by allowing the algorithm to move to worse locations in parameter space, thereby skirting across local optima; the method performs a slow but thorough search. An attempt to improve upon the convergence speed of the annealing algorithm is Ingber's (1996) simulated quenching algorithm. Yet another approach is the taboo method (Glover *et al*. (1993)) which is a strategy that forces an algorithm (typically a gradient method) to move through regions of parameter space that have not previously been visited. Genetic algorithms (Davis (1991)) offer an entirely different approach again. Based on the evolutionary notion of natural selection, combinations of the best intermediate solutions are paired together repeatedly until a single dominant optimum emerges.

When applying a gradient method to an observed log-likelihood which has, or may have, multiple local optima, it is advisable to initiate the algorithm from different locations in parameter space. This approach is adequate for the examples we present here, but it can become untenable in higher-dimensional parameter spaces.

As outlined, the estimation problem has two components: estimating parameters and estimating the associated standard errors of the estimator. Fortunately, by focusing on the solution for the first component, we will, as a by-product, achieve the solution for the second. We begin by defining the *penalty function*, which is the negative of the observed log-likelihood function:

$$p(\theta) = -\log L(\theta). \tag{12.6}$$

Minimising the penalty function for choices of $\theta$ yields the equivalent result to maximum likelihood. The reason for defining the penalty function is purely because the optimisation literature is couched in terms of minimisation, rather than maximisation. Finally, we assume the parameter $\theta$, a $(k \times 1)$ vector, is of dimension $k \geq 2$ and such that $\theta \in \Theta = \mathbb{R}^k$. Accordingly, optimisation corresponds to unconstrained minimisation over choice variables defined everywhere in two- or higher-dimensional real space.

Before proceeding further, we make brief points about the $k = 1$ case. The case of numerical optimisation over the real line (*i.e.* corresponding to just one parameter, since $k = 1$) is of lesser importance in practice. If univariate optimisation is needed, line search algorithms such as Golden Search and methods due to Brent (1973) should be applied; see Luenberger (1984) for discussion of these and other possibilities. *Mathematica*'s `FindMinimum` function utilises versions of these algorithms, and our experience of its performance has, on the whole, been good. Determining in advance whether the derivative of the penalty function can be constructed (equivalent to the negative of the score) will usually cut down the number of iterations, and can save time. If so, a single starting point need only be supplied (*i.e.* it is unnecessary to compute the gradient and supply it to the function through the `Gradient` option). Univariate optimisation can, however, play an important role in multivariate optimisation by determining step-length optimally.

Finally, as we have seen, parametric constraints often arise in statistical models. In these cases, the parameter space $\Theta = \{\theta : \theta \in \Theta\}$ is a proper subset of $k$-dimensional real space or may be degenerate upon it (*i.e.* $\Theta \subset \mathbb{R}^k$). This means that maximum likelihood/minimum penalty estimation requires constrained optimisation methods. Our opinion on `FindMinimum` as a *constrained* optimiser — in its Version 4 incarnation — is that we cannot recommend its use. The approach that we advocate is to transform a constrained optimisation into an unconstrained optimisation, and use `FindMinimum` on the latter. This can be achieved by re-defining parameter $\theta$ to a new parameter $\lambda = g(\theta)$ in a manner such that $\lambda \in \mathbb{R}^q$, where $q \leq k$. Of course, the trick is to determine the appropriate functional form for the transformation $g$. Once we have determined $g$ and optimised with respect to $\lambda$, recovery of estimation results (via the Invariance Property) pertinent to $\theta$ can be achieved by using replacement rules, as well as by exploiting *Mathematica*'s excellent differentiator.

## 12.5 B Gradient Method Algorithms

An algorithm (gradient method or otherwise) generates a finite-length sequence such as the following one:

$$\hat{\theta}_{(0)}, \hat{\theta}_{(1)}, \ldots, \hat{\theta}_{(r)} \tag{12.7}$$

where the bracketed subscript indicates the iteration number. Each $\hat{\theta}_{(j)} \in \mathbb{R}^k$, $j = 0, \ldots, r$, resides in the same space as the $\theta$, and each can be regarded as an estimate of $\hat{\theta}$:

$$\hat{\theta} = \arg\max_{\theta \in \mathbb{R}^k} \log L(\theta) = \arg\min_{\theta \in \mathbb{R}^k} p(\theta).$$

The sequence (12.7) generally depends on three factors: (i) the point at which the algorithm starts $\hat{\theta}_{(0)}$, (ii) how the algorithm progresses through the sequence; that is, how

$\hat{\theta}_{(j+1)}$ is obtained from $\hat{\theta}_{(j)}$, and (iii) when the process stops. Of the three factors, our attention focuses mainly on the second — the iteration method.

### ○ *Initialisation*

Starting values are important in all types of optimisation methods — more so, perhaps, for gradient method algorithms because of the multiple local optima problem. One remedy is to start from different locations in the parameter space in order to trace out the surface of the observed log-likelihood, but this may not appeal to the purist. Alternative methods have already been discussed in §12.5 A, with simulated annealing methods probably worthy of first consideration.

### ○ *The Iteration Method*

Typically, the link between iterations takes the following form,

$$\hat{\theta}_{(j+1)} \;=\; \hat{\theta}_{(j)} + \mu_{(j)}\, d_{(j)} \tag{12.8}$$

where the step-length $\mu_{(j)} \in \mathbb{R}_+$ is a scalar, and the direction $d_{(j)} \in \mathbb{R}^k$ is a vector lying in the parameter space. In words, we update our estimate obtained at iteration $j$, namely $\hat{\theta}_{(j)}$, by moving in the direction $d_{(j)}$ by a step of length $\mu_{(j)}$.

The fundamental feature of algorithms coming under the umbrella of gradient methods is that they are *never worsening*. That is,

$$p(\hat{\theta}_{(0)}) \;\geq\; p(\hat{\theta}_{(1)}) \;\geq\; \cdots \;\geq\; p(\hat{\theta}_{(r-1)}) \;\geq\; p(\hat{\theta}_{(r)}).$$

Thus, each member in the sequence (12.7) traces out an increasingly better approximation for minimising the penalty function. Using these inequalities and the relationship (12.8), for any $j = 0, \ldots, r$, we must have

$$p\big(\hat{\theta}_{(j)} + \mu_{(j)}\, d_{(j)}\big) - p(\hat{\theta}_{(j)}) \leq 0. \tag{12.9}$$

The structure of a gradient method algorithm is determined by approximating the left-hand side of (12.9) by truncating its Taylor series expansion. To see this, replace $\mu_{(j)}$ in (12.9) with $\mu$, and take a (truncated) Taylor series expansion of the first term about $\mu = 0$, to yield

$$\mu\, p_g(\hat{\theta}_{(j)}) \,.\, d_{(j)}$$

where the $(k \times 1)$ vector $p_g$ denotes the gradient of the penalty function. Of course, $p_g$ is equivalent to the negative of the score, and like the score, it too disappears at $\hat{\theta}$; that is, $p_g(\hat{\theta}) = \vec{0}$. Replacing the left-hand side of (12.9) with the Taylor approximation, finds

$$p_g(\hat{\theta}_{(j)}) \,.\, d_{(j)} \leq 0 \tag{12.10}$$

for $\mu$ is a positive scalar. Expression (12.10) enables us to construct a range of differing possibilities for the direction vector. For example, for a symmetric matrix $W_{(j)}$, we might

select direction according to

$$d_{(j)} = -W_{(j)} \,.\, p_g(\hat{\theta}_{(j)}) \tag{12.11}$$

because then the left-hand side of (12.10) is a weighted quadratic form in the elements of vector $p_g$, the weights being the elements of matrix $W_{(j)}$; that is,

$$p_g(\hat{\theta}_{(j)}) \,.\, d_{(j)} \;=\; -p_g(\hat{\theta}_{(j)}) \,.\, W_{(j)} \,.\, p_g(\hat{\theta}_{(j)}). \tag{12.12}$$

This quadratic form will be non-positive provided that the matrix of weights $W_{(j)}$ is positive semi-definite (in practice, $W_{(j)}$ is taken positive definite to ensure strict improvement). Thus, the algorithm improves from one iteration to the next until a point $p_g(\hat{\theta}_{(r)}) = \vec{0}$ is reached within numerical tolerance.

Selecting different weight matrices defines various iterating procedures. In particular, four choices are NR = Newton–Raphson, Score = Method of Scoring, DFP = Davidon–Fletcher–Powell and BFGS = Broydon–Fletcher–Goldfarb–Shanno:

$$\text{NR}: \quad W_{(j)} = -H_{(j)}{}^{-1} \tag{12.13}$$

$$\text{Score}: \quad W_{(j)} = I_{(j)}{}^{-1} \tag{12.14}$$

$$\text{DFP}: \quad W_{(j+1)} \;=\; W_{(j)} \;+\; \frac{\Delta\hat{\theta} \times \left(\Delta\hat{\theta}\right)^{\mathrm{T}}}{\Delta\hat{\theta} \,.\, \Delta p_g} \;-\; \frac{\left(W_{(j)} \,.\, \Delta p_g\right) \times \left(W_{(j)} \,.\, \Delta p_g\right)^{\mathrm{T}}}{\Delta p_g \,.\, W_{(j)} \,.\, \Delta p_g} \tag{12.15}$$

$$\begin{aligned} \text{BFGS}: \quad W_{(j+1)} \;=\; & (\text{as for DFP}) \;+\; \left(\Delta p_g \,.\, W_{(j)} \,.\, \Delta p_g\right) \\ & \times \left(\frac{\Delta\hat{\theta}}{\Delta\hat{\theta} \,.\, \Delta p_g} - \frac{W_{(j)} \,.\, \Delta p_g}{\Delta p_g \,.\, W_{(j)} \,.\, \Delta p_g}\right) \times \left(\frac{\Delta\hat{\theta}}{\Delta\hat{\theta} \,.\, \Delta p_g} - \frac{W_{(j)} \,.\, \Delta p_g}{\Delta p_g \,.\, W_{(j)} \,.\, \Delta p_g}\right)^{\mathrm{T}} \end{aligned} \tag{12.16}$$

The notation used here is the following:

$H_{(j)}$ is the Hessian of the observed log-likelihood function evaluated at $\theta = \hat{\theta}_{(j)}$

$I_{(j)}$ is the Sample Information matrix evaluated at $\theta = \hat{\theta}_{(j)}$

$\Delta\hat{\theta} \;=\; \hat{\theta}_{(j)} - \hat{\theta}_{(j-1)}$ is the change in the estimate from the previous iteration, and

$\Delta p_g = p_g(\hat{\theta}_{(j)}) - p_g(\hat{\theta}_{(j-1)})$ is the change in the gradient.

The DFP and BFGS weighting matrices appear complicated, but as we shall see in the following section, implementing them with *Mathematica* is reasonably straightforward. Of the algorithms (12.13) – (12.16), `FindMinimum` includes the NR algorithm (`Method` → `Newton`) and the BFGS algorithm (`Method` → `QuasiNewton`).

To illustrate, we shall obtain the iterator for the Method of Scoring. Combine (12.8), (12.11) and (12.14), to yield

$$\hat{\theta}_{(j+1)} = \hat{\theta}_{(j)} - \mu_{(j)} \, I_{(j)}^{-1} \, . \, p_g(\hat{\theta}_{(j)})$$

which, to be complete, requires us to supply a step-length $\mu_{(j)}$. We might, for instance, select step-length to optimally improve the penalty function when moving in direction $d_{(j)} = -I_{(j)}^{-1} \, . \, p_g(\hat{\theta}_{(j)})$ from $\hat{\theta}_{(j)}$; this is achieved by solving

$$\mu_{(j)} = \arg\min_{\mu \in \mathbb{R}_+} \, p\big(\hat{\theta}_{(j)} - \mu \, I_{(j)}^{-1} \, . \, p_g(\hat{\theta}_{(j)})\big).$$

Of course, this is a univariate optimisation problem that can be solved by numerical means using `FindMinimum`. Unfortunately, experience suggests that determining step-length in this manner can be computationally inefficient, and so a number of alternatives have been proposed. In particular, one due to Armijo is implemented in the examples given below.

A final point worth noting concerns estimating the asymptotic standard error of the estimator. As mentioned previously, this estimate is obtained as a by-product of the optimisation. This is because an estimate of the asymptotic variance-covariance matrix is given by the final weighting matrix $W_{(r)}$, since the estimates of the asymptotic standard error are the square root of the main diagonal of this matrix. The NR weight (12.13) corresponds to the Hessian estimator, and the Score weight (12.14) to the Fisher estimator (see Table 3); the DFP and BFGS weights are other (consistent) estimators. However, the default algorithm implemented in `FindMinimum` (the conjugate gradient algorithm) does not yield, as a by-product, the estimate of the asymptotic variance-covariance matrix.

○ ***Stopping Rules***

Algorithms converge (asymptotically) to $\hat{\theta}$; nevertheless, from a practical view, the sequence (12.7) must be terminated in finite time, and the estimate $\hat{\theta}_{(r)}$ of $\hat{\theta}$ must be reported. This therefore requires that we define numerical convergence. How this is done may vary. Possibilities include the following:

(i) convergence defined according to epsilon change in parameter estimates:

$$\text{stop if } \left\| \hat{\theta}_{(r)} - \hat{\theta}_{(r-1)} \right\| < \epsilon_1$$

(ii) convergence defined according to epsilon change in the penalty function:

$$\text{stop if } \left| \, p(\hat{\theta}_{(r)}) - p(\hat{\theta}_{(r-1)}) \, \right| < \epsilon_2$$

(iii) convergence defined according to the gradient being close to zero:

$$\text{stop if } \left\| p_g(\hat{\theta}_{(r)}) \right\| < \epsilon_3$$

(iv) convergence defined according to the gradient element with the largest absolute value being close to zero:

$$\text{stop if } \max\big( \left| \, p_g(\hat{\theta}_{(r)}) \, \right| \big) < \epsilon_4$$

where $\epsilon_1, \epsilon_2, \epsilon_3$ and $\epsilon_4$ are small positive numbers, $|\cdot|$ denotes the absolute value of the argument, and $\|\cdot\|$ denotes the Euclidean distance of the argument vector from the origin (the square root of the dot product of the argument vector with itself). The method we favour is (iv).

Of course, picking just one rule out of this list may be inappropriate as a stopping rule, in which case numerical convergence can be defined according to combinations of (i), (ii), (iii) or (iv) holding simultaneously. Finally, (i)–(iv) hold if $\hat{\theta}_{(r)}$ happens to locate either a local maximum or a saddle point of the penalty function, so it is usually necessary to check that the Hessian of the penalty function (equal to the negative of the Hessian of the observed log-likelihood) is positive definite at $\hat{\theta}_{(r)}$.

## 12.6 The BFGS Algorithm

In this section, we employ the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm to estimate a Poisson two-component-mix model proposed by Hasselblad (1969).

### ○ *Data, Statistical Model and Log-likelihood*

Our data — the Death Notice data — appears in Table 5. The data records the number of death notices for women aged 80 or over, each day, in the English newspaper, *The Times*, during the three-year period, 1910–1912.

| Death Notices per day ($X$) : | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Frequency (no. of days) : | 162 | 267 | 271 | 185 | 111 | 61 | 27 | 8 | 3 | 1 |

**Table 5:** Death Notice data

The data is interpreted as follows: there were 162 days in which no death notices appeared, 267 days in which one notice appeared, … and finally, just 1 day on which the newspaper listed nine death notices. We enter the data as follows:

```
count = {162, 267, 271, 185, 111, 61, 27, 8, 3, 1};
```

As the true distribution of $X$ = 'the number of death notices published daily' is unknown, we shall begin by specifying the Poisson($\gamma$) model for $X$,

$$P(X = x) = \frac{e^{-\gamma}\,\gamma^x}{x!}, \qquad x \in \{0, 1, 2, \ldots\} \quad \text{and} \quad \gamma \in \mathbb{R}_+$$

Then, the log-likelihood is:

```
Clear[G];          logLγ = Log[∏_{x=0}^{G} ( e^{-γ} γ^x / x! )^{n_x} ]
```

$$-\gamma \sum_{x=0}^{G} n_x + \text{Log}[\gamma] \sum_{x=0}^{G} x\, n_x - \sum_{x=0}^{G} \text{Log}[x!]\, n_x$$

where, in order for `SuperLog` to perform its magic, we have introduced the `Subscript` $n_x$ to index, element by element, the data in list `count` (so $n_0 = 162$, $n_1 = 267$, …, $n_9 = 1$).[9] Define $G < \infty$ to be the largest number of death notices observed in the sample, so $G = 9$ for our data.[10] ML estimation in this model is straightforward because the log-likelihood is concave with respect to $\gamma$.[11] This ensures that the ML estimator is given by the solution to the first-order condition:

```
solγ = Solve[Grad[logLγ, γ] == 0, γ] // Flatten
```

$$\left\{\gamma \to \frac{\sum_{x=0}^{G} x\, n_x}{\sum_{x=0}^{G} n_x}\right\}$$

For our data, the ML estimate is obtained by inputting the data into the ML estimator, `solγ`, using a replacement rule:

```
solγ = solγ /. {G → 9, n_x_ :→ count[[x + 1]]} // N
```

```
{γ → 2.15693}
```

We leave estimation of the standard error of the estimator as an exercise for the reader.[12]

When Hasselblad (1969) examined the Death Notice data, he suggested that the sampled population was in fact made up of two sub-populations distinguished according to season, since death rates in winter and summer months might differ. As the data does not discriminate between seasons, Hasselblad proceeded by specifying an unknown mixing parameter between the two sub-populations. We denote this parameter by $\omega$ (for details on component-mix models, see §3.4 A). He also specified Poisson distributions for the sub-populations. We denote their parameters by $\phi$ and $\psi$. Hasselblad's Poisson two-component mix model is

$$P(X = x) \;=\; \omega\, \frac{e^{-\phi}\, \phi^x}{x!} \;+\; (1-\omega)\, \frac{e^{-\psi}\, \psi^x}{x!}, \qquad x \in \{0, 1, 2, \ldots\}$$

where the mixing parameter $\omega$ is such that $0 < \omega < 1$, and the Poisson parameters satisfy $\phi > 0$ and $\psi > 0$. For Hasselblad's model, the observed log-likelihood can be entered as:

```
obslogLλ = Log[ ∏_{x=0}^{G} ( ω (e^-ϕ ϕ^x)/(x!) + (1 - ω) (e^-ψ ψ^x)/(x!) )^(n_x) ] /.
    {G → 9, n_x_ :→ count[[x + 1]], ω → 1/(1 + e^a), ϕ → e^b, ψ → e^c};
```

Note that we have implemented a re-parameterisation of $\theta = (\omega, \phi, \psi)$ to $\lambda = g(\theta) = (a, b, c) \in \mathbb{R}^3$ by using a replacement rule (see the second line of input).

Due to the non-linear nature of the first-order conditions, ML estimation of the unknown parameters requires iterative methods for which we choose the BFGS algorithm.[13] Using `FindMaximum`, initialised at $(a, b, c) = (0.0, 0.1, 0.2)$, finds:[14]

```
FindMaximum[obslogLλ, {a, 0.0}, {b, 0.1}, {c, 0.2},
            Method → QuasiNewton]
```

```
{-1989.95, {a → 0.575902, b → 0.227997, c → 0.9796}}
```

Using the estimates 0.575902, 0.227997 and 0.9796, for $a$, $b$ and $c$, respectively, we can obtain the ML estimates for $\omega$, $\phi$ and $\psi$. Alas, with `FindMinimum / FindMaximum`, it is not possible to inspect the results from each iteration in the optimisation procedure; nor, more importantly, can we recover estimates of the asymptotic variance-covariance matrix of the ML estimator of $\lambda$. Without the asymptotic variance-covariance matrix, we cannot, for example, undertake the inference described in §12.4. Thus, `FindMinimum/ FindMaximum` does not do all that we might hope for.

### ○ *BFGS Algorithm*

We now code the BFGS algorithm, and then apply it to estimate the parameters of Hasselblad's model. We begin by converting the re-parameterised observed log-likelihood into a penalty function:

```
p = -obslogLλ;
```

Our task requires the unconstrained minimisation of the penalty `p` with respect to $\lambda$. Our BFGS code requires that we define the penalty function `pf` and its gradient `gradpf` as *Mathematica* functions of the parameters, using an immediate evaluation:

```
    pf[{a_, b_, c_}] = p;
gradpf[{a_, b_, c_}] = Grad[p, {a, b, c}];
```

To see that this has worked, evaluate the gradient of the penalty function at $a = b = c = 0$:

```
g = gradpf[{0, 0, 0}]
```

```
{0, -634, -634}
```

We now present some simple code for each part of the BFGS algorithm (12.16). The following module returns the updated approximation $W_{(j)}$ to the negative of the inverse Hessian matrix at each iteration:

```
BFGS[Δθ_, Δgrad_, W_] :=
 Module[{t1, t2, t3, t4, t5, t6, t7},
            t1 = Outer[Times, Δθ, Δθ];
            t2 = Δθ.Δgrad;
            t3 = W.Δgrad;
            t4 = Outer[Times, t3, t3];
            t5 = Δgrad.t3;
  (* For DFP ignore the remaining lines
                 and return W+t1/t2-t4/t5 *)
            t6 = Δθ / t2 - t3 / t5;
            t7 = Outer[Times, t6, t6];
            W + t1 / t2 - t4 / t5 + t5 t7]
```

The BFGS updating expression can, of course, be coded as a one-line command. However, this would be inefficient as a number of terms are repeated; hence, the terms `t1` to `t7` in `BFGS`.

The next component that is needed is a line search method for determining step-length $\mu$. There happen to be quite a few to choose from. For simplicity, we select a relatively easy version of Armijo's method as given in Polak (1971) (for a more detailed version, see Luenberger (1984)):

```
Armijo[f_, θ_, grad_, dir_] :=
 Module[{α = 0.5, β = 0.65, μ = 1., f0, gd},
    f0 = f[θ];   gd = grad.dir;
    While[ f[θ + μ dir] - f0 - μ α gd > 0, μ = β μ];  μ]
```

This module essentially determines a feasible step-length $\mu$, but not necessarily an optimal one. The first argument, `f`, denotes the objective function (our penalty function). Because `Armijo` needs to evaluate `f` at many points, the `Armijo` function assumes that `f` is a *Mathematica* function like `pf` (not `p`). A more advanced method is Goldstein's (again, see Polak (1971) or Luenberger (1984)), where bounds are determined within which an optimising search can be performed using, for example, `FindMinimum` (but remember to transform to an unconstrained optimisation). The cost in undertaking this method is the additional time it takes to determine an optimal step-length.

To set BFGS on its way, there are two initialisation choices required—$\hat{\lambda}_{(0)}$ and $W_{(0)}$—which are the beginning parameter vector 'guess' and the beginning inverse Hessian matrix 'guess', respectively. The success of our search can depend crucially on these two factors. To illustrate, suppose we set $\hat{\lambda}_{(0)} = (0, 0, 0)$ and $W_{(0)} = I_3$. From (12.8), (12.11), and our earlier output, it follows that $\hat{\lambda}_{(1)} = \mu_{(0)} \times (0, 634, 634)$. Determining step-length, we find:

```
Armijo[pf, {0, 0, 0}, g, {0, 634, 634}]

— General::unfl : Underflow occurred in computation.
— General::unfl : Underflow occurred in computation.
— General::unfl : Underflow occurred in computation.
— General::stop : Further output of
    General::unfl will be suppressed during this calculation.

1.
```

… and the algorithm has immediately run into troubles. The cause of these difficulties is scaling (quantities such as `Exp[-634]` are involved in numeric computations). Fortunately, a heuristic that can help to overcome this type of ill-conditioning is to enforce scale dependence onto $W_{(0)}$. A simple one that can often work is:

```
W0[θ_, grad_] := Sqrt[θ.θ / grad.grad] IdentityMatrix[Length[θ]]
```

$$W0[\theta_, grad_] := \sqrt{\frac{\theta.\theta}{grad.grad}}\ IdentityMatrix[Length[\theta]]$$

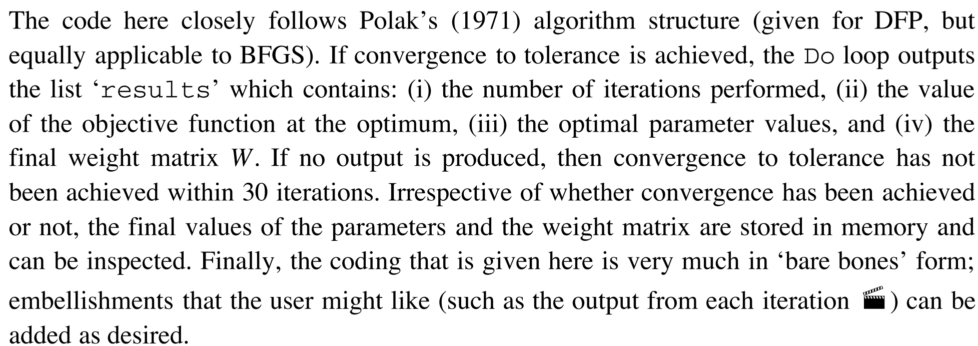
`W0` ensures that the Euclidean length of the initial direction vector from the origin matches that of the initial starting parameter; that is, $W_{(0)}$ is forced to be such that direction $d_{(0)} = -W_{(0)} . g(\hat{\lambda}_{(0)})$ satisfies

$$\sqrt{d_{(0)} . d_{(0)}} = \sqrt{\hat{\lambda}_{(0)} . \hat{\lambda}_{(0)}} \ .$$

Of course, forcing $W_{(0)}$ to behave in this way always rules out selecting $\hat{\lambda}_{(0)}$ as a zero vector as the initial parameter guess. For further details on other generally better methods of scaling and pre-conditioning, see Luenberger (1984).

We now implement the BFGS algorithm using the parts constructed above. As the starting point, we shall select:

```
λ0 = {0.0, 0.1, 0.2};
```



The code here closely follows Polak's (1971) algorithm structure (given for DFP, but equally applicable to BFGS). If convergence to tolerance is achieved, the `Do` loop outputs the list '`results`' which contains: (i) the number of iterations performed, (ii) the value of the objective function at the optimum, (iii) the optimal parameter values, and (iv) the final weight matrix $W$. If no output is produced, then convergence to tolerance has not been achieved within 30 iterations. Irrespective of whether convergence has been achieved or not, the final values of the parameters and the weight matrix are stored in memory and can be inspected. Finally, the coding that is given here is very much in 'bare bones' form; embellishments that the user might like (such as the output from each iteration) can be added as desired.

```
(* Start iteration (iter=0) *)
λ0 = {0.0, 0.1, 0.2};
g0 = gradpf[λ0];
W = W0[λ0, g0];

Do[ (* Subsequent iterations (maximum 30) *)
            d = -W.g0;
           λ1 = λ0 + Armijo[pf, λ0, g0, d] d;
           g1 = gradpf[λ1];
    If[ Max[Abs[g1]] < 10^-6,
        W = BFGS[λ1 - λ0, g1 - g0, W];
        Break[results = {iter, -pf[λ1], λ1, W}] ];
           Δλ = λ1 - λ0;
           Δg = g1 - g0;
  (* Reset λ0 and g0 for the next iteration *)
           λ0 = λ1;  g0 = g1;
           W = BFGS[Δλ, Δg, W],      {iter, 30}]
```

```
{26, -1989.95, {0.575862, 0.228008, 0.979605},
 ( 0.775792   -0.245487  -0.0810482
  -0.245487    0.084435   0.0246111
  -0.0810482   0.0246111  0.0093631 )}
```

The output states that the BFGS algorithm converged to tolerance after 26 iterations. The ML estimates are $\hat{a} = 0.575862$, $\hat{b} = 0.228008$ and $\hat{c} = 0.979605$; almost equivalent to the point estimates returned by `FindMaximum`. At the estimates, the observed log-likelihood is maximised at a value of $-1989.95$. The BFGS estimate of the asymptotic variance-covariance matrix is the $(3 \times 3)$ matrix in the output. Table 6 summarises the results.

| | Estimate | SE | TStat |
|---|---|---|---|
| $a$ | 0.575862 | 0.880791 | 0.653801 |
| $b$ | 0.228008 | 0.290577 | 0.784673 |
| $c$ | 0.979605 | 0.0967631 | 10.1237 |

**Table 6:** ML estimation results for the unrestricted parameters

Our stopping rule focuses on the gradient, stopping if the element with the largest magnitude is smaller than $10^{-6}$. Our choice of $10^{-6}$ corresponds to the default for `AccuracyGoal` in `FindMinimum`. It would not pay to go much smaller than this, and may even be wise to increase it with larger numbers of parameters.[15] Other stopping rules can be tried.[16] Finally, the outputted $W$ is an estimate of the asymptotic variance-covariance matrix.

To finish, we present a summary of the ML estimates and their associated standard errors and $t$-statistics for the parameters of the original Poisson two-component-mix model. To do this, we use the Invariance Property, since the unrestricted parameters $\lambda$ are linked to the restricted parameters $\theta$ by the re-parameterisation $\lambda = g(\theta)$. Here, then, are the ML estimates of the Poisson two-component-mix parameters $\theta = (\omega, \phi, \psi)$:

```
solλ = { a → results⟦3, 1⟧,
         b → results⟦3, 2⟧,
         c → results⟦3, 3⟧ };

solθ = {ω → 1/(1 + ⅇ^a), ϕ → ⅇ^b, ψ → ⅇ^c} /. solλ

{ω → 0.359885, ϕ → 1.2561, ψ → 2.6634}
```

That is, the ML estimate of the mixing parameter is $\hat{\omega} = 0.359885$, and the ML estimates of the Poisson component parameters are $\hat{\phi} = 1.2561$ and $\hat{\psi} = 2.6634$. Here is the estimate of the asymptotic variance-covariance matrix (see (11.17)):

```
G = Grad[{1/(1 + ⅇ^a), ⅇ^b, ⅇ^c}, {a, b, c}];

G.W.Transpose[G] /. solλ

⎛ 0.0411708  0.0710351  0.0497281 ⎞
⎜ 0.0710351  0.133219   0.0823362 ⎟
⎝ 0.0497281  0.0823362  0.0664192 ⎠
```

We summarise the ML estimation results obtained using the BFGS algorithm in Table 7.

| | Estimate | SE | TStat |
|---|---|---|---|
| $\omega$ | 0.359885 | 0.202906 | 1.77366 |
| $\phi$ | 1.2561 | 0.364992 | 3.44143 |
| $\psi$ | 2.6634 | 0.257719 | 10.3345 |

**Table 7:** ML estimation results for the Poisson two-component-mix model

Finally, it is interesting to contrast the fit of the Poisson model with that of the Poisson two-component-mix model. Here, as a function of $x \in \{0, 1, 2, \ldots\}$, is the fitted Poisson model:

$$\texttt{fitP} = \frac{e^{-\gamma}\,\gamma^{x}}{x!}\ \texttt{/. sol}\gamma$$

$$\frac{0.115679\ 2.15693^{x}}{x!}$$

… and here is the fitted Poisson two-component-mix model:

$$\texttt{fitPcm} = \left(\omega\,\frac{e^{-\phi}\,\phi^{x}}{x!} + (1-\omega)\,\frac{e^{-\psi}\,\psi^{x}}{x!}\right)\ \texttt{/. sol}\theta$$

$$\frac{0.102482\ 1.2561^{x}}{x!} + \frac{0.0446227\ 2.6634^{x}}{x!}$$

Table 8 compares the fit obtained by each model to the data. Evidently, the Poisson two-component-mix model gives a closer fit to the data than the Poisson model in every category. This improvement has been achieved as a result of introducing two additional parameters, but it has come at the cost of requiring a more complicated estimation procedure.

| | Count | Mixed | Poisson |
|---|---|---|---|
| 0 | 162 | 161.227 | 126.784 |
| 1 | 267 | 271.343 | 273.466 |
| 2 | 271 | 262.073 | 294.924 |
| 3 | 185 | 191.102 | 212.044 |
| 4 | 111 | 114.193 | 114.341 |
| 5 | 61 | 57.549 | 49.325 |
| 6 | 27 | 24.860 | 17.732 |
| 7 | 8 | 9.336 | 5.464 |
| 8 | 3 | 3.089 | 1.473 |
| 9 | 1 | 0.911 | 0.353 |

**Table 8:** Fitted Poisson and Poisson two-component-mix models

## 12.7 The Newton–Raphson Algorithm

In this section, we employ the Newton–Raphson (NR) algorithm to estimate the parameters of an Ordered Probit model.

○ ***Data, Statistical Model and Log-likelihood***

Random variables that cannot be observed are termed *latent*. A common source of such variables is individual sentiment because, in the absence of a rating scale common to all individuals, sentiment cannot be measured. Even without an absolute measurement of sentiment, it is often possible to obtain partial information by using categorisation; a sampling device that can achieve this is the ubiquitous ‘opinion survey’. Responses to such surveys are typically ordered — *e.g.* choose one of ‘disliked Brand X’, ‘indifferent to Brand X’, or ‘liked Brand X’ — which reflects the ordinal nature of sentiment. Such latent, ordered random variables are typically modelled using cumulative response probabilities. Well-known models of this type include the *proportional-odds* model and the *proportional-hazards* model (*e.g.* see McCullagh and Nelder (1989)), and the *Ordered Probit* model due to McKelvey and Zavoina (1975) (see also Maddala (1983) and Becker and Kennedy (1992)). In this section, we develop a simple form of the ordered probit model (with cross-classification), estimating parameters using the Newton–Raphson (NR) algorithm.

During consultations with a general medical practitioner, patients were asked a large number of lifestyle questions. One of these was (the somewhat morbid), “Have you recently found that the idea of taking your own life kept coming into your mind?”. Goldberg (1972) reports count data for 295 individuals answering this question in Table 9.

| *Illness class* | Definitely not ($j = 1$) | Do not think so ($j = 2$) | Has crossed my mind ($j = 3$) | Definitely has ($j = 4$) |
|---|---|---|---|---|
| Normal ($i = 1$) | 90 | 5 | 3 | 1 |
| Mild ($i = 2$) | 43 | 18 | 21 | 15 |
| Severe ($i = 3$) | 34 | 8 | 21 | 36 |

**Table 9:** Psychiatric data — cross-classified by illness

The data is assumed to represent a categorisation of the ‘propensity to suicidal thought’, a latent, ordered random variable. Responses are indexed by $j$, running across the columns of the table. In addition, all individuals had been cross-classified into one of three psychiatric classes: normal ($i = 1$), mild psychiatric illness ($i = 2$), and severe psychiatric illness ($i = 3$). For example, of the 167 individuals responding “Definitely not”, 90 were classified as normal, 43 as having mild psychiatric illness and 34 as suffering severe psychiatric illness. Enter the data:

```
freq = {{90, 5, 3, 1}, {43, 18, 21, 15}, {34, 8, 21, 36}};
```

Due to the cross-classification, the issue of interest is whether the propensity to suicidal thought can be ranked according to illness. Upon inspection, the data seems to

suggest that the propensity to suicidal thought increases with mental illness. In order to quantify this view, we define three latent, ordered random variables,

$Y_i^*$ = Propensity to suicidal thought of an individual classified with illness $i$

and we specify the following linear model for each,

$$Y_i^* = \beta_i + U_i\,, \qquad i \in \{1, 2, 3\} \tag{12.17}$$

where $U_i$ is an unknown disturbance term with zero mean. The (cross-classified) Ordered Probit model is characterised by assuming a trivariate Normal distribution (see §6.4 B) with independent components for the disturbances, namely,

$$\begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} \sim N(\vec{0}, I_3) \tag{12.18}$$

which is scale invariant because observations are categorical. The class-specific parameter $\beta_i$ enables us to quantify the differences between the psychiatric classes. In parametric terms, if propensity to suicidal thought can be ranked increasingly in respect of psychiatric illness, then we would expect $\beta_1 < \beta_2 < \beta_3$ —a testable hypothesis.[17]

Of course, it is *not* $Y_i^*$ that is observed in Table 9; rather, observations have been recorded on another trio of random variables which we define as

$Y_i$ = the response to the survey question of an individual classified with illness $i$.

To establish the link between response $Y_i$ and propensity $Y_i^*$, we assume $Y_i$ is a categorisation of $Y_i^*$, and that

$$P(Y_i = j) = P(\alpha_{j-1} < Y_i^* < \alpha_j) \tag{12.19}$$

for all combinations of indexes $i$ and $j$. The parameters $\alpha_0, \ldots, \alpha_4$ are cut-off parameters which, because of the ordered nature of $Y_i^*$, satisfy the inequalities $\alpha_0 < \alpha_1 < \cdots < \alpha_4$. Given the Normality assumption (12.18), we immediately require $\alpha_0 = -\infty$ and $\alpha_4 = \infty$ to ensure that probabilities sum to unity. Substituting (12.17) into (12.19), yields

$$\begin{aligned} P(Y_i = j) &= P(\alpha_{j-1} - \beta_i < U_i < \alpha_j - \beta_i) \\ &= \Phi(\alpha_j - \beta_i) - \Phi(\alpha_{j-1} - \beta_i) \end{aligned} \tag{12.20}$$

where $\Phi$ denotes the cdf of a $N(0, 1)$ random variable (which is the marginal cdf of $U_i$):[18]

```
Clear[Φ];          Φ[x_] = 1/2 (1 + Erf[x/Sqrt[2]]);
```

Then, the observed log-likelihood is given by:

```
obslogLθ =
  Log[∏_{i=1}^{3} (Φ[α1 - βi])^freq[[i,1]] (Φ[α2 - βi] - Φ[α1 - βi])^freq[[i,2]]
     (Φ[α3 - βi] - Φ[α2 - βi])^freq[[i,3]] (1 - Φ[α3 - βi])^freq[[i,4]]];
```

As it stands, the parameters of this model cannot be estimated uniquely. To see this, notice that in the absence of any restriction, it is trivially true that for any non-zero constant $\gamma$, the categorical probability in the ordered probit model satisfies

$$\Phi(\alpha_j - \beta_i) - \Phi(\alpha_{j-1} - \beta_i) \;=\; \Phi\big((\alpha_j + \gamma) - (\beta_i + \gamma)\big) - \Phi\big((\alpha_{j-1} + \gamma) - (\beta_i + \gamma)\big)$$

for *all possible* $i$ and $j$. Thus, the probability determined from values assigned to the parameters $(\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3)$ cannot be distinguished from the probability resulting from values assigned as per $(\alpha_1 + \gamma, \alpha_2 + \gamma, \alpha_3 + \gamma, \beta_1 + \gamma, \beta_2 + \gamma, \beta_3 + \gamma)$ for any arbitrary $\gamma \neq 0$. This phenomenon is known as a *parameter identification problem.* To overcome it, we must break the equivalence in probabilities for at least one combination of $i$ and $j$. This can be achieved by fixing one of the parameters, thus effectively removing it from the parameter set. Any parameter will do, and any value can be chosen. In practical terms, it is better to remove one of the cut-off parameters $(\alpha_1, \alpha_2, \alpha_3)$, for this reduces by one the number of inequalities to which these parameters must adhere. Conventionally, the identifying restriction is taken to be:

```
α1 = 0;
```

The parameter $\theta = (\alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3)$ is defined over the space

$$\Theta = \{(\alpha_2, \alpha_3) : (\alpha_2, \alpha_3) \in \mathbb{R}_+^2, 0 < \alpha_2 < \alpha_3\} \times \{(\beta_1, \beta_2, \beta_3) : (\beta_1, \beta_2, \beta_3) \in \mathbb{R}^3\}$$

and therefore $\Theta$ is a proper subset of $\mathbb{R}^5$. For unconstrained optimisation, a transformation to new parameters $\lambda = g(\theta) \in \mathbb{R}^5$ is required. Clearly, the transformation need only act on the cut-off parameters, and one that satisfies our requirements is:

```
prm = {α2 == α3/(1 + ⅇ^a2), α3 == ⅇ^a3, β1 == b1, β2 == b2, β3 == b3};
```

where $\lambda = (\text{a2, a3, b1, b2, b3}) \in \mathbb{R}^5$. Notice that $\alpha_3$ will be positive for all a3, and that it will always be larger than $\alpha_2$ for all a2, so the constraints $0 < \alpha_2 < \alpha_3$ will always be satisfied.[19] From inspection of `prm`, it is apparent that we have not yet determined $g(\theta)$, for $\alpha_2$ depends on $\alpha_3$. However, by inputting:

```
θToλ = Solve[prm, {α2, α3, β1, β2, β3}] // Flatten

{α2 → ⅇ^a3/(1 + ⅇ^a2), β1 → b1, β2 → b2, β3 → b3, α3 → ⅇ^a3}
```

we now have $g(\theta)$ in the form of a replacement rule. We now enter into *Mathematica* the observed log-likelihood function in terms of $\lambda$ (`obslogLλ`):

```
obslogLλ = obslogLθ /. θToλ;
```

Similar to §12.6, we can use `FindMaximum` to estimate the parameters using the NR algorithm:

```
FindMaximum[obslogLλ, {a2, 0}, {a3, 0},
 {b1, 0}, {b2, 0}, {b3, 0}, Method → Newton]

{-292.329, {a2 → 0.532781, a3 → -0.0507304,
  b1 → -1.34434, b2 → 0.0563239, b3 → 0.518914}}
```

But, as has previously been stated, the main drawback to using `FindMaximum` is that it does not supply the final Hessian matrix — we cannot construct an estimate of the asymptotic variance-covariance matrix of the ML estimator of $\lambda$ from `FindMaximum`'s output.

### ○ *NR Algorithm*

We shall estimate $\lambda$ and the asymptotic variance-covariance matrix using the NR algorithm. From (12.8), (12.11) and (12.13), the NR algorithm is based on the updating formulae:

$$\begin{aligned} \hat{\lambda}_{(k+1)} &= \hat{\lambda}_{(k)} + \mu_{(k)}\, d_{(k)} \\ d_{(k)} &= -W_{(k)} \,.\, p_g(\hat{\lambda}_{(k)}) \\ W_{(k)} &= -H_{(k)}{}^{-1} \end{aligned}$$

where $k$ is the iteration index, $p_g$ is the gradient of the penalty function, $W$ is the inverse of the Hessian of the penalty function and $H$ is the Hessian of the observed log-likelihood function. We obtain the penalty function, the gradient and the Hessian as follows:

```
p = -obslogLλ;
pf[{a2_, a3_, b1_, b2_, b3_}] = p;

g = Grad[p, {a2, a3, b1, b2, b3}];
gradpf[{a2_, a3_, b1_, b2_, b3_}] = g;

H = Hessian[obslogLλ, {a2, a3, b1, b2, b3}];
hessf[{a2_, a3_, b1_, b2_, b3_}] = H;
```

These are very complicated expressions, so unless your computer has loads of memory capacity, and you have loads of spare time, we strongly advise using the humble semi-colon '`;`' (as we have done) to suppress output to the screen! Here, `gradpf` and `hessf` are functions with a `List` of `Symbol` arguments matching exactly the elements of $\lambda$. The reason for constructing these two functions is to avoid coding the NR algorithm with numerous replacement rules, since such rules can be computationally inefficient and more cumbersome to code. The vast bulk of computation time is spent on the Hessian matrix.

This is why NR algorithms are costly, for they evaluate the Hessian matrix at every iteration. It is possible to improve computational efficiency by compiling the Hessian.[20]

Another way to proceed is to input the mathematical formula for the Hessian matrix directly into *Mathematica*; Maddala (1983), for instance, gives such formulae. This method has its cost too, not least of which is that it runs counter to the approach taken throughout this volume, which is to ask *Mathematica* to do the work. Yet another approach is to numerically evaluate/estimate the first- and second-order derivatives, for clearly there will exist statistical models with parameter numbers of such magnitude that there will be insufficient memory available for *Mathematica* to derive the symbolic Hessian — after all, our example only has five parameters, and yet computing the symbolic Hessian already requires around 7 MB (on our reference machine) of free memory. In this regard, the standard add-on package `NumericalMath`NLimit`` may assist, for its `ND` command performs numerical approximations of derivatives.

In §12.3, we noted that the NR algorithm is useful as a 'finishing-off' algorithm which fine tunes our estimates. This is because NR uses the actual Hessian matrix, whereas quasi-Newton algorithms (like BFGS) only use estimates of the Hessian matrix. But, for this example, we will apply the NR algorithm from scratch. Fortunately for us, the log-likelihood of the Ordered Probit model can be shown to be globally concave in its parameters; see Pratt (1981). Thus, the Hessian matrix is negative definite for all $\theta$, and therefore negative definite for all $\lambda$, as the two parameters are related one-to-one.

In principle, given concavity, the NR algorithm will reach the global maximum from wherever we choose to start in parameter space. Numerically, however, it is nearly always a different story! Sensible starting points nearly always need to be found when optimising, and the Ordered Probit model is no exception. For instance, if a starting value for `a3` equal to 3.0 is chosen, then one computation that we are performing is the integral under a standard Normal distribution curve up to $\exp(3) \simeq 20$. In this case, it would not be surprising to see the algorithm crash, as we will run out of numerical precision; see Sofroniou (1996) for a discussion of numerical precision in *Mathematica*. Sensible starting values usually require some thought and are typically problem-specific — even when we are fortunate enough to have an apparently ideal globally concave log-likelihood, as we do here.

Our implementation of the NR algorithm follows. Like our BFGS algorithm, we have left it very much without any bells and whistles. Upon convergence to tolerance, the output is recorded in `results` which has four components: `results⟦1⟧` is the number of iterations taken to achieve convergence to tolerance; `results⟦2⟧` is the value of the maximised observed log-likelihood; `results⟦3⟧` is the ML point estimates; and `results⟦4⟧` is the negative of the inverse Hessian evaluated at the ML point estimates. The origin would seem to be a sensible starting value at which to initiate the algorithm.

```
Armijo[f_, θ_, grad_, dir_] :=
 Module[{α = 0.5, β = 0.65, μ = 1., f0, gd},
    f0 = f[θ];   gd = grad.dir;
    While[ f[θ + μ dir] - f0 - μ α gd > 0, μ = β μ];  μ]
```

```
λ0 = {0., 0., 0., 0., 0.};
g0 = gradpf[λ0];

Do[        H0 = hessf[λ0];
           W0 = -Inverse[H0];
            d = -W0.g0;
           λ1 = λ0 + Armijo[pf, λ0, g0, d] d;
           g1 = gradpf[λ1];
If[Max[Abs[g1]] < 10^-6, Break[results =
     {iter, -pf[λ1], λ1, -Inverse[hessf[λ1]]}]];
           λ0 = λ1;
           g0 = g1,
                        {iter, 1, 20}];
```

From its starting point, the NR algorithm takes just over 10 seconds to converge to tolerance on our reference machine. In total, it takes five iterations:

```
results⟦1⟧
```

```
5
```

The returned estimate of $\lambda$ is:

```
results⟦3⟧
```

```
{0.532781, -0.0507304, -1.34434, 0.0563239, 0.518914}
```

at which the value of the observed log-likelihood is:

```
results⟦2⟧
```

```
-292.329
```

Table 10 gives estimation results for the parameters of our original Ordered Probit model (found using the Invariance Property). Because $\hat{\beta}_1 < \hat{\beta}_2 < \hat{\beta}_3$, our quantitative results lend support to the qualitative assessment made at the very beginning of this example — that propensity to suicidal thought increases with severity of psychiatric illness.[21]

| | Estimate | SE | TStat |
|---|---|---|---|
| $\alpha_2$ | 0.35157 | 0.059407 | 5.91804 |
| $\alpha_3$ | 0.95054 | 0.094983 | 10.00740 |
| $\beta_1$ | −1.34434 | 0.174219 | −7.71641 |
| $\beta_2$ | 0.05632 | 0.118849 | 0.47391 |
| $\beta_3$ | 0.51891 | 0.122836 | 4.22446 |

**Table 10:** ML estimation results for the Ordered Probit model

## 12.8 Exercises

**1.** Generate 10 pseudo-random drawings from $X \sim N(0, 1)$ as follows:

```
data = Table[√2 InverseErf[0, -1 + 2 Random[]], {10}]
```

Use ML estimation to fit the $N(\mu, \sigma^2)$ distribution to the artificial data using each of the following:

```
FindMaximum[obslogLθ, {μ, 0}, {σ, 1}]
FindMaximum[obslogLθ, {μ, {-1, 1}}, {σ, {0.5, 2}}]
FindMaximum[obslogLθ, {μ, 0, -3, 3}, {σ, 1, 0, 4}]
FindMaximum[obslogLθ, {μ, 0}, {σ, 1}, Method → Newton]
FindMaximum[obslogLθ, {μ, 0}, {σ, 1}, Method → QuasiNewton]
```

where `obslogLθ` is the observed log-likelihood for $\theta = (\mu, \sigma)$. Contrast your answers against the estimates computed from the exact ML estimator

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i \quad \text{and} \quad \hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{\mu})^2} .$$

**2.** Let $X \sim \text{Waring}(a, b)$ with pmf

$$P(X = x) = (b - a) \frac{\Gamma(x + a)\ \Gamma(b)}{\Gamma(a)\ \Gamma(x + b + 1)}, \qquad x \in \{0, 1, 2, \ldots\}$$

where the parameters are such that $b > a > 0$. Use `FindMaximum` to obtain ML estimates of $a$ and $b$ for the Word Count data, which is loaded using:

```
ReadList["WordCount.dat"]
```

Hint: re-parameterise $\theta = (a, b)$ to $\lambda = (c, d) \in \mathbb{R}^2$, where $a = e^c$ and $b = e^c(1 + e^d)$. Estimate the variance-covariance matrix of the asymptotic distribution of the ML estimator using the Hessian estimator.

**3.** Let $X \sim \text{NegativeBinomial}(r, p)$.

(i) Show that $\mu < \sigma^2$, where $\mu = E[X]$ and $\sigma^2 = \text{Var}(X)$.

(ii) Let $(X_1, X_2 \ldots, X_n)$ denote a random sample of size $n$ on $X$. Now it is generally accepted that $\overline{X}$, the sample mean, is the best estimator of $\mu$. Using the log-likelihood concentrated with respect to the estimator $\overline{X}$ for $\mu$, obtain the ML estimate of $r$ for the data sets `NB1` (enter as `ReadList["NB1.dat"]`) and `NB2` (enter as `ReadList["NB2.dat"]`).

(iii) Comment on the fit of each model. Can you find any reason why the ML estimate for the `NB2` data seems so erratic?

**4.** Answer the following using the Nerve data given in §12.2. Let $X \sim \text{Gamma}(\alpha, \beta)$ with pdf $f(x; \theta)$, where $\theta = (\alpha, \beta)$. *Example 2* derived the ML estimate as $\hat{\theta} = (\hat{\alpha}, \hat{\beta}) =$ (1.17382, 0.186206).

(i) Derive Fisher's estimate of the asymptotic variance-covariance matrix of the ML estimator of $\theta$ (hint: see *Example 3*).

(ii) Given $\hat{\theta}$, use the Invariance Property (§11.4 E) to derive ML estimates of:

(a) $\lambda = (\mu, \nu)$, where $\mu = E[X]$ and $\nu = \text{Var}(X)$, and

(b) the asymptotic variance-covariance matrix of the ML estimator of $\lambda$.

(iii) Re-parameterise the pdf of $X$ to $f(x; \lambda)$.

(a) Use `FindMaximum`'s BFGS algorithm (`Method` → `QuasiNewton`) to obtain the ML estimate of $\lambda$.

(b) Estimate the asymptotic variance-covariance matrix of the ML estimator of $\lambda$ using the Fisher, Hessian and Outer-product estimators.

(c) Compare your results for parts (a) and (b) to those obtained in (ii).

(iv) Using the Invariance Property (§11.4 E), report ML estimates of:

(a) $\delta = (\mu, \sigma)$, where $\mu = E[X]$ and $\sigma = \alpha$, and

(b) the asymptotic variance-covariance matrix of the ML estimator of $\delta$.

(v) Re-parameterise the pdf of $X$ to $f(x; \delta)$.

(a) Use `FindMaximum`'s BFGS algorithm to obtain the ML estimate of $\delta$.

(b) Estimate the asymptotic variance-covariance matrix of the ML estimator of $\delta$ using the Fisher, Hessian and Outer-product estimators.

(c) Compare your results for parts (a) and (b) to those obtained in (iv).

**5**. The Gamma regression model specifies the conditional distribution of $Y \mid X = x$, with pdf

$$f(y \mid X = x; \theta) = \frac{1}{\Gamma(\sigma)} \left(\frac{\mu}{\sigma}\right)^{-\sigma} \exp\left(-\frac{y\,\sigma}{\mu}\right) y^{\sigma-1}$$

where $\mu = \exp(\alpha + \beta\, x)$ is the regression function, $\sigma$ is a scaling factor and parameter $\theta = (\alpha, \beta, \sigma) \in \{\alpha \in \mathbb{R}, \beta \in \mathbb{R}, \sigma \in \mathbb{R}_+\}$. Use ML estimation to fit the Gamma regression model to Greene's data (see *Example 5*). By performing a suitable test on $\sigma$, determine whether the fitted model represents a significant improvement over the Exponential regression model for Greene's data.

**6**. Derive ML estimates of the ARCH model of §12.3 based on the BFGS algorithm of §12.6. Obtain an estimate of the variance-covariance matrix of the asymptotic distribution of the ML estimator. Report your ML estimates, associated asymptotic standard errors and $t$-statistics.