How-to: Bayes with *Mathematica*

Romke Bontekoe

Bontekoe Research, Amsterdam
romke@bontekoe.nl

Abstract. *Mathematica* is unique in its integrated symbolic and numerical solving capacities. Prof. Phil Gregory states "..., the time required to develop and test programs with Mathematica is approximately 20 times shorter than the time required to write and debug the same program in Fortran or C, ...". However, *Mathematica* has a steep learning curve. In this paper the *Mathematica* code for Bayesian linear regression and model selection is provided as a take-off for novice users.

Keywords: Bayesian linear regression, Bayesian model selection, Mathematica
PACS: 02.50.Tt

LINEAR REGRESSION

In this paper the basics of Bayesian linear regression are exposed side-by-side with the corresponding *Mathematica*[1] code. *Mathematica* is organised in Notebooks. The corresponding Notebook can be obtained by sending an email to the author.

The regression example is taken from Bishop [1]. The $N = 10$ data points are taken from a Sine curve with added Gaussian noise ($\sigma_{noise} = 0.3$). The aim is to construct a function by linear regression, which approximates the $t = Sin(x)$ function in Figure 1 as good as we can; the t stands for target value.

The actual data consist of two parts, a vector of measurement, or target, values $\boldsymbol{t} = (t_1, ..., t_n)^T$ and a vector of the corresponding sampling positions $\boldsymbol{x} = (x_1, ...x_n)^T$.

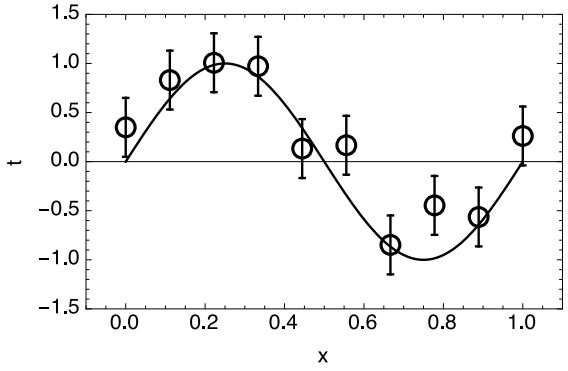

FIGURE 1.

[1] Mathematica is a trademark of Wolfram Research Inc.

The standard deviation of the noise is expressed by the inverse variance $\beta = \sigma_{\text{noise}}^{-2}$, or the *precision* of the data.

Non-linear basis functions give flexibility in linear regression problems. There are many possible choices, e.g. polynomials, Gaussian kernel functions, sigmoids, wavelets, etc. The basis functions need not to be a set of orthogonal functions. Writing a vector of M basis functions $\boldsymbol{\phi}(x) = (1, \phi_1(x), ..., \phi_{M-1}(x))^T$ and corresponding weights $\boldsymbol{w} = (w_0, ...w_{M-1})^T$, the regression model can be written as a dot-product

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}). \tag{1}$$

The target function $y(\boldsymbol{x}, \boldsymbol{w})$ can be non-linear in $\boldsymbol{x}$ but is linear in $\boldsymbol{w}$. Since the measurement positions $\boldsymbol{x}$ are fixed also the basis functions $\boldsymbol{\phi}(\boldsymbol{x})$ are fixed. The optimization is over the linear variables $\boldsymbol{w}$, hence *linear regression*.

For polynomial basis functions $\boldsymbol{\phi}(x) = \left(1, x, x^2, ..., x^{M-1}\right)^T$ the model becomes

$$y(x_n, \boldsymbol{w}) = w_0 + w_1 x_n + w_2 x_n^2 + ... + w_{M-1} x_n^{M-1} = \sum_{j=0}^{M-1} w_j x_n^j, \tag{2}$$

where x_n denotes the sample position of the n-th datum. Note that $\boldsymbol{w}$-vectors of length M correspond to polynomials of degree $m = M - 1$. The *Mathematica* code for the basis function is

```
basisPhiPoly[ m_Integer, x_ ] := Table[ x^ i, {i,0,m} ]
```

which defines a function basisPhiPoly[.] depending on an integer variable m and an unspecified x, which can be numeric or symbolic. The {i,0,m} is the table iterator. It is customary that user defined functions begin with a lower case letter; all *Mathematica* library functions begin with an upper case letter.

The basis function is called inside the model function

```
yModelPoly[w_List] := Function[ Evaluate[ w . basisPhiPoly[ Length[w]-1, # ] ] ] ]
```

where the length of the input vector of weights w defines the degree of the polynomial. The variable x is represented by the operator #. The result is the dot-product of the two vectors.

The entire problem can be formulated by the $N \times M$ design matrix $\boldsymbol{\Phi}$, composed of N the row-vectors of basis functions $\boldsymbol{\phi}(x_n) = (1, \phi_1(x_n), ..., \phi_{M-1}(x_n))$

```
designPhiPoly[ m_Integer, xData_List ] := basisPhiPoly[ m,# ] & /@ xData
```

where the xData_List is the list of $\boldsymbol{x} = (x_1, ...x_n)^T$. The Map (/@) operator applies the basisPhiPoly[.] function to the list xData and yields a vector for each data point x_n. The # and & pair are part of the *Mathematica* function syntax.

These three *Mathematica* functions form the building blocks for the regression problem.

MAXIMUM LIKELIHOOD SOLUTION

The likelihood function gives a numerical measure on how good (or how bad) a model $y(\boldsymbol{x},\boldsymbol{w})$ fits the data. For independent Gaussian noise, the likelihood function is the product of N Gaussians

$$p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w},\beta) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|y\left(x_n,\boldsymbol{w}\right),\beta^{-1}\right) \ . \tag{3}$$

The full expression for the log-likelihood is

$$\ln p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w},\beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\left(y\left(x_n,\boldsymbol{w}\right)-t_n\right)^2 \ + \frac{N}{2}\ln\beta \ \ - \frac{N}{2}\ln(2\pi), \tag{4}$$

from which immediately the *sum of squares* is recognised. And indeed, in the maximum (log-)likelihood solution the weights vector $\boldsymbol{w}$ is identical to the least squares solution.

The maximum is found by differentiating the log-likelihood with respect to $\boldsymbol{w}$. For Gaussian noise, this can is done analytically. This yields the Maximum Likelihood solution $\boldsymbol{w}_{\mathrm{ML}}$

$$\boldsymbol{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} \ \boldsymbol{\Phi}^T\boldsymbol{t}, \tag{5}$$

where the $\boldsymbol{\Phi}$ is the design matrix.

```
wMLpoly[ m_Integer, tData_List, xData_List ] := Block[ { designMatrix, wML },
designMatrix = designPhiPoly[ m, xData ];
wML = Inverse[ Transpose[designMatrix] . designMatrix ] . Transpose[designMatrix] .
    tData
]
```

The Block[.] isolates the internal variables {designMatrix, wML} of this function from the rest of the *Mathematica* code. Note the three vector-matrix Dot[.] products. *Mathematica* does not distinguish between vector-matrix-tensor multiplication as long as the dimensions are compatible.

The maximum likelihood solution model function $y_{\mathrm{ML}}(\boldsymbol{x},\boldsymbol{w}_{\mathrm{ML}})$ is obtained by substituting the $\boldsymbol{w}_{\mathrm{ML}}$ in the regression model

```
yModelPoly[ wMLpoly[ m, tData, xData ] ] [ x ]
```

The value of the log-likelihood solution is readily available in *Mathematica*. The residuals, i.e. the differences between the target model $y(\boldsymbol{x},\boldsymbol{w}_{ML})$ and the target values $\boldsymbol{t}$ are obtained by

```
misfit = yModelPoly[ wMLpoly[ m, tData, xData ] ][ # ] & xData - tData.
```

First the $\boldsymbol{w}_{\mathrm{ML}}$ coefficients are computed (wMLpoly[.]). Next the function yModelPoly[.] is defined and Map[.]-ped onto the list of sampling points (xData), yielding the model values $y(x_n,\boldsymbol{w}_{ML})$ at the sampling positions. Finally the difference between the model values $y(x_n,\boldsymbol{w}_{ML})$ and the target values t_n is taken. The result is stored in the variable misfit and is a list of N numbers.

For Gaussian noise with standard deviation σ_{noise} the log-likelihood is now

```
LogLikelihood[ NormalDistribution[ 0.0, dataStDev ], misfit ].
```

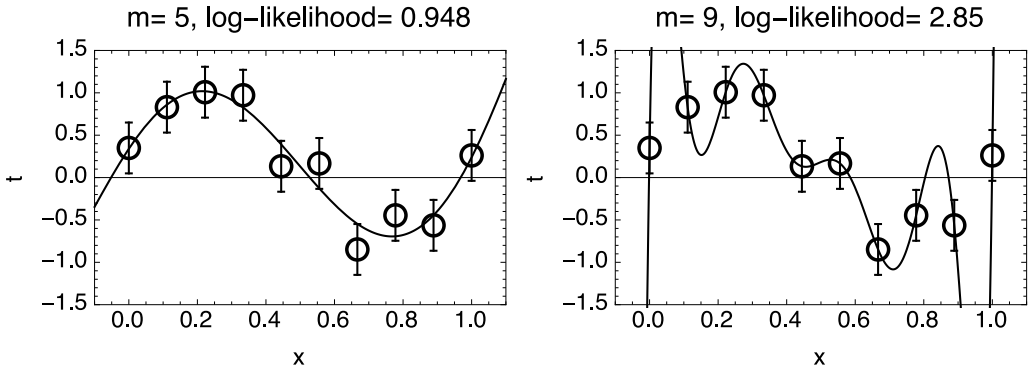

FIGURE 2.

Figure 2 shows the maximum likelihood solution for polynomials with $m = 5$ and $m = 9$. For the polynomial of degree $m = 9$, we have an excellent fit through the training points, but a very poor fit elsewhere, especially for x values near the ends.

The maximum likelihood solution weights $\boldsymbol{w}$ for all polynomial degrees are shown in Table 1. Note the large and nearly cancelling coefficients for degree $m = 9$. Also note that log-likelihood values (bottom row of Table 1) steadily increase with with polynomial degree. The value of the log-likelihood gives no indication when overfitting is lurking. The risk of overfitting is always present when using the maximum likelihood method.

Nearly singular matrices occur in Equation (5) for polynomial degrees $m = 8$ and $m = 9$ for Bishop's data. These are signalled by *Mathematica* and a warning is issued in these cases.

BAYESIAN PARAMETER ESTIMATION

To simplify the treatment, we consider a zero-mean, isotropic Gaussian prior probability distribution for $\boldsymbol{w}$, with a single precision parameter $\alpha = \sigma_{\text{prior}}^{-2}$

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}\left(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}\right), \tag{6}$$

with $\boldsymbol{I}$ the identity covariance matrix. Now a single value α controls the width of the prior distribution for all dimensions of $\boldsymbol{w}$.

TABLE 1. Values of maximum likelihood solution weights $\boldsymbol{w}$ as a function of the polynomial degree. The bottom line lists the log-likelihood values.

	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9
w_0	0.19	0.82	0.91	0.31	0.32	0.34	0.35	0.35	0.35	0.35
w_1		−1.27	−1.90	7.99	7.67	6.08	2.62	7.40	−19.43	232.46
w_2			0.64	−25.43	−23.83	−10.55	32.10	−46.78	494.28	−5323.95
w_3				17.37	14.82	−22.74	−206.27	261.45	−3819.51	48587.36
w_4					1.28	44.33	399.00	−923.52	14471.69	−231728.38
w_5						−17.22	−332.71	1595.91	−30455.45	640283.91
w_6							105.16	−1294.45	36098.80	−1062194.59
w_7								399.89	−22493.94	1042780.94
w_8									5723.46	−557883.74
w_9										125245.90
Log[LH]	−18.11	−9.02	−8.84	0.91	0.91	0.95	1.02	1.06	1.28	2.85

With a Gaussian likelihood function and a Gaussian prior, the posterior distribution for $\boldsymbol{w}$ from Bayes' theorem is the (normalised) product of two Gaussian distributions. The full solution is

$$\begin{aligned} p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{x},\alpha,\beta) &= \frac{p(\boldsymbol{t}|\boldsymbol{w},\boldsymbol{x},\beta)*p(\boldsymbol{w}|\alpha)}{p(\boldsymbol{t}|\boldsymbol{x},\beta)} \\ &= \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N,\boldsymbol{S}_N) \end{aligned} \tag{7}$$

with

$$\begin{aligned} \boldsymbol{m}_N &= \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}, \\ \boldsymbol{S}_N^{-1} &= \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}. \end{aligned} \tag{8}$$

First $\boldsymbol{S}_N$ must be solved for:

```
covarPoly[ dataStdDev_, m_Integer, priorStdDev_, xData_List ] :=
Block[ { covar, designMatrix, hyperα, hyperβ },
hyperα = 1/priorStdDev^2;
hyperβ = 1/dataStdDev^2;
designMatrix = designPhiPoly[ m, xData ];
covar = hyperα*IdentityMatrix[ m+1 ] + hyperβ*Transpose[designMatrix].designMatrix;
covar = Inverse[ covar ];
covar = 0.5*( Transpose[ covar ] + covar ) ]
```

As before, the Block[.] defines some local variables. The covariance matrix covar is computed in three lines. First, the RHS of Equation (8) is computed. Next this matrix is inverted by Inverse[.]. However, this covar matrix is not always symmetric due to numerical rounding errors. Since strict symmetry is required by the MultinormalDistribution[.] function, the covariance matrix is forced to be symmetric in the third line.

Next we can solve for $\boldsymbol{m}_N$:

```
meanPoly[ dataStdDev_, m_Integer, priorStdDev_, tData_List, xData_List ] :=
Block[ {covar, designMatrix, hyperβ, mean },
hyperβ = 1/dataStdDev^2;
covar = covarPoly[ dataStdDev, m, priorStdDev, xData ];
designMatrix = designPhiPoly[ m, xData ];
mean = hyperβ * covar . Transpose[designMatrix] . tData ]
```

Note the two matrix-vector dot products in the last line.

The maximum of the posterior probability distribution, $\boldsymbol{w}_{\mathrm{MAP}}$, is equal to the mean (or mode) $\boldsymbol{m}_N$ for a multinormal distribution. This is the best point estimate we can make. The uncertainty in the values of $\boldsymbol{w}_{\mathrm{MAP}}$ are the variances on the diagonal of the covariance matrix $\boldsymbol{S}_N$.

The $\boldsymbol{w}_{\mathrm{MAP}}$ solution and the corresponding target function $y(\boldsymbol{x},\boldsymbol{w}_{\mathrm{MAP}})$ for the linear regression problem is obtained by:

```
wMAP = meanPoly[ dataStdDev, m, priorStdDev, tData, xData ];
yModelPoly[ wMAP ][ x ];
```

Figure 3 shows the MAP solutions for two polynomials with $m = 5$ and $m = 9$. $\sigma_{\mathrm{prior}} = 20$ is chosen for the prior width, which covers a range of reasonable values for $\boldsymbol{w}$. This demonstrates that the wild overfitting can be controlled by a suitable prior.

However, the choice of the best degree m for the polynomial is still unsettled.

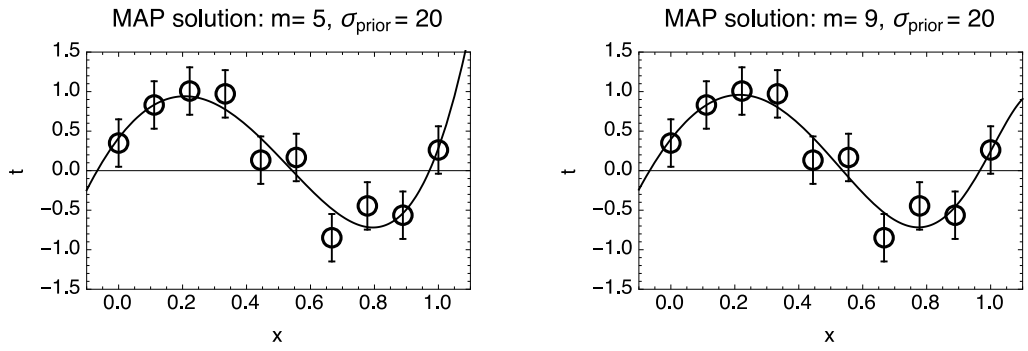

FIGURE 3.

BAYESIAN MODEL SELECTION

Suppose we didn't know that Bishop's regression data were drawn from a Sine function, but we were told that they came from one of the 10 polynomials. But we weren't told from which one. Bayesian model selection allows us to find the polynomial with the largest probability.

Denoting the ten polynomials by their degree $(M_0, M_1, ..., M_9)$ we need to find the polynomial degree m for which $p(M_m|\boldsymbol{t},\boldsymbol{x})$ is the largest.
Applying Bayes' theorem for manipulation of probabilities

$$p(M_m|\boldsymbol{t},\boldsymbol{x}) = p(M_m|\boldsymbol{x}) * p(\boldsymbol{t}|\boldsymbol{x},M_m) \big/ p(\boldsymbol{t}|\boldsymbol{x}). \tag{9}$$

The $p(\boldsymbol{t}|\boldsymbol{x})$ is a normalization factor, independent on the model M_m and can be ignored in finding the maximum. The prior probability for each of the ten models ought to be independent of $\boldsymbol{x}$, hence $p(M_m|\boldsymbol{x}) = p(M_m)$. In absence of any other knowledge we assign the uniform prior over the models $p(M_m) = 1/10$, and thus can be ignored as well. The hyperparameters α and β are suppressed for readability.

However, the only way to connect the data with the model is through the weight vector $\boldsymbol{w}$ of regression coefficients corresponding to the model M_m

$$\begin{aligned} p(M_m|\boldsymbol{t},\boldsymbol{x}) &\propto p(\boldsymbol{t}|\boldsymbol{x},M_m) \\ &= \int p(\boldsymbol{w},\boldsymbol{t}|\boldsymbol{x},M_m)\, d\boldsymbol{w} \\ &= \int p(\boldsymbol{t}|\boldsymbol{w},\boldsymbol{x},M_m) * p(\boldsymbol{w}|\boldsymbol{x},M_m)\ d\boldsymbol{w} \\ &= \int \text{likelihood} * \text{prior}\ d\boldsymbol{w}. \end{aligned} \tag{10}$$

A few observations can be made. The $p(\boldsymbol{t}|\boldsymbol{x},M_m)$ already appeared before as the evidence $p(\boldsymbol{t}|\boldsymbol{x},\beta)$ in Equation (7). Earlier, the model dependence was concealed in the *length* of the vectors $\boldsymbol{t}$ and $\boldsymbol{x}$. Also the $p(\boldsymbol{t}|\boldsymbol{w},\boldsymbol{x},M_m)$ is the same likelihood function as before. The $p(\boldsymbol{w}|M_m)$ is the prior probability distribution over the weights for a given model M_m.

But most importantly, the model evidence in Equation (10) requires an $(m+1)$ dimensional integration over $d\boldsymbol{w}$. For every degree in the polynomial model M_m an *Ockham* factor $\frac{\Delta w_{posterior}}{\Delta w_{prior}} < 1$ appears, and the model evidence as a function of m becomes

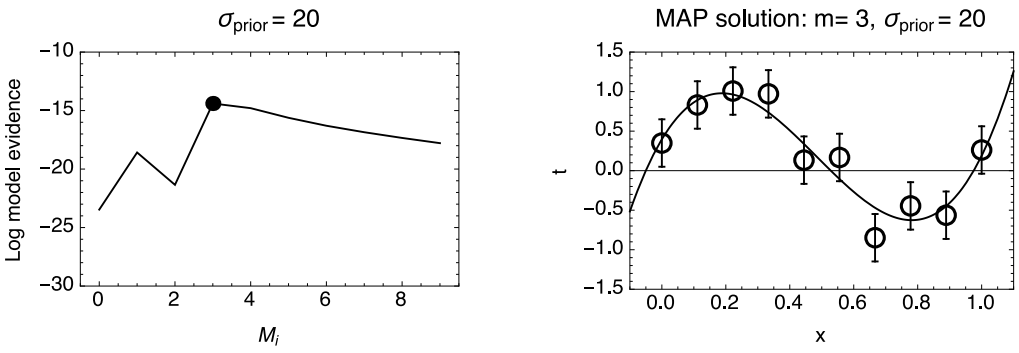

FIGURE 4.

$$p(\boldsymbol{t}|\boldsymbol{x},M_m) = p(\boldsymbol{t}|\boldsymbol{w}_{\mathrm{ML}},\boldsymbol{x},M_m) * \left(\frac{\Delta \mathrm{w}_{\mathrm{posterior}}}{\Delta \mathrm{w}_{\mathrm{prior}}}\right)^{m+1}. \tag{11}$$

The value of the likelihood increases with m due to the increasingly better fit with a higher polynomial degree (see Table 1). But the increase tapers off for say $m \geq 5$. On the other hand, the product of Ockham factors is a fast decreasing function with m. There is a trade-off in their product and the model evidence $p(\boldsymbol{t}|\boldsymbol{x},M_m)$ attains a maximum for a certain m. This model has the highest probability.

Taking again a conjugate prior for $p(\boldsymbol{w}|\alpha,M_m)$, i.e. a Gaussian distribution with zero mean $\boldsymbol{m}_0 = \boldsymbol{0}$ and isotropic covariance matrix $\boldsymbol{S}_0 = \alpha^{-1}\boldsymbol{I}$, then the model evidence can be integrated analytically over $\boldsymbol{w}$. The logarithm of the model evidence is

$$\ln\ p(\boldsymbol{t}|\boldsymbol{x},M_m)\ = \tfrac{m+1}{2}\ln\alpha + \tfrac{N}{2}\ln\beta - E(\boldsymbol{m}_N) - \tfrac{1}{2}\ln|\boldsymbol{A}| - \tfrac{N}{2}\ln(2\pi), \tag{12}$$

with the error term $E(\boldsymbol{m}_N) = \frac{\beta}{2}\|\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{m}_N\|^2 + \frac{\alpha}{2}\boldsymbol{m}_N.\boldsymbol{m}_N$. The mean is at $\boldsymbol{m}_N = \beta\boldsymbol{A}^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$, and $\boldsymbol{A} = \alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ is the Hessian matrix. Note the model complexity penalty factor $(m+1)/2\ln\alpha$ in the log-model evidence in Equation (12).

```
logEvidencePoly[ dataStDev_, m_Integer, priorStDev_, tData_List, xData_List ] :=
Block[{error, hessianA, hyperα, hyperβ, logEvidence, mean, misfit, nData, yModel},
hyperα = 1/priorStDev^2;
hyperβ = 1/dataStDev^2;
nData = Length[xData];
hessianA = hyperα*IdentityMatrix[m+1] +
hyperβ * Transpose[ designPhiPoly[m,xData] ] . designPhiPoly[m,xData] ;
mean = hyperβ*Inverse[hessianA] . Transpose[ designPhiPoly[m,xData] ] . tData;
yModel = designPhiPoly[m,xData] . mean;
misfit = yModel-tData;
error = 0.5*hyperβ*misfit.misfit + 0.5*hyperα*mean.mean;
logEvidence = 0.5*(m+1)*Log[ hyperα ] + 0.5*nData*Log[ hyperβ ] - error - 0.5*Log[
    Det[hessianA] ] - 0.5*nData*Log[2*Pi]
]
```

Again assuming a value $\sigma_{\mathrm{prior}} = 20$, we find that a third degree polynomial has the maximum evidence (Figure 4).

EPILOG

The aim of this paper is to demonstrate that coding in *Mathematica* requires relatively few lines. The code presented deals only with Gaussian models, which lie at the heart of almost all first attempts to solve problems in data analysis.

On the other hand, learning *Mathematica* is a considerable investment in time. Gregory [3] states that after this investment is made the development of writing software is sped op by an order of magnitude, mostly due to shortened test and debug cycles. His book is accompanied by a comprehensive *Mathematica* notebook.

Blower [2] has written two volumes on Bayesian information processing from a combinatorial point of view. In Volume 2 he points out some conceptual errors in the famous book by Jaynes [4]. Blower also provides some *Mathematica* code in his text.

Recently, the book by Von der Linden, Dose, and Von Toussaint [6] has appeared. It combines a thorough treatment of Bayesian probability theory with real applications from physics. The chapters on numerical techniques, as MCMC and Nested Sampling, are especially worth studying.

The list of references would not be complete without the mention of the book by Sivia and Skilling [5] which provides a concise and clear introduction into the subject.

The author's favourite book remains the excellent book by Bishop [1].

On a more personal note, the author of this paper regrets that he did not start using *Mathematica* earlier in his career. Anyone interested can obtain a *Mathematica* notebook with the code in this paper, including the plotting code, by sending an email to the author.

REFERENCES

1. Cristopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.
2. David J. Blower, *Information Processing*, 2 Volumes, CreateSpace Independent Publ., USA, 2013.
3. Phil Gregory, *Bayesian Logical Data Analysis for the Physical Sciences*, CUP, Cambridge, 2005.
4. Edwin T. Jaynes, *Probability Theory, the Logic of Science.*, Cambridge UP, Cambridge, 2003.
5. D. S. Sivia with J. Skilling, *Data Analysis, a Bayesian Tutorial*, Oxford UP, Oxford, 2006.
6. W. Von der Linden, V. Dose, U. Von Toussaint, *Bayesian Probability Theory.*, CUP, Cambridge, 2014.