AD Model Builder introduction course

Random effects models

AD Model Builder foundation

anders@nielsensweb.org
About random effect models

- In purely fixed effects models we have
  - Random variables we observe
  - Model parameters we want to estimate

- In random effects models we have
  - Random variables we observe
  - Random variables we do **NOT** observe
  - Model parameters we want to estimate

- This model class is very useful and goes by many names: random effects models, mixed models, latent variable models, state-space models, frailty models, hierarchical models, ...

- Many tools can handle linear Gaussian models.

- No other tool handles non-linear non-Gaussian random effect models like ADMB.
Example: Paired observations

- Two methods A and B to measure blood cell count (to check for the use of doping).
- Paired study.

<table>
<thead>
<tr>
<th>Person ID</th>
<th>Method A</th>
<th>Method B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.5</td>
<td>5.4</td>
</tr>
<tr>
<td>2</td>
<td>4.4</td>
<td>4.9</td>
</tr>
<tr>
<td>3</td>
<td>4.6</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>5.4</td>
<td>4.9</td>
</tr>
<tr>
<td>5</td>
<td>7.6</td>
<td>7.2</td>
</tr>
<tr>
<td>6</td>
<td>5.9</td>
<td>5.5</td>
</tr>
<tr>
<td>7</td>
<td>6.1</td>
<td>6.1</td>
</tr>
<tr>
<td>8</td>
<td>7.8</td>
<td>7.5</td>
</tr>
<tr>
<td>9</td>
<td>6.7</td>
<td>6.3</td>
</tr>
<tr>
<td>10</td>
<td>4.7</td>
<td>4.2</td>
</tr>
</tbody>
</table>

- It must be expected that two measurements from the same person are correlated, so a paired t-test is the correct analysis
- The t-test gives a p-value of 5.1%, which is a borderline result...
- But more data is available
• In addition to the planned study 10 persons were measured with only one method

• Want to use all data, which is possible with random effects

• Assume these 20 are randomly selected from a population where the blood cell count is normally distributed

• Consider the following model:

\[ C_i = \alpha(M_i) + B(P_i) + \varepsilon_i, \quad i = 1 \ldots 30 \]

- \(\alpha(M_i)\) the 2 fixed method effects
- \(B(P_i) \sim \mathcal{N}(0, \sigma^2_P)\) the 20 random effects
- \(\varepsilon_i \sim \mathcal{N}(0, \sigma^2_R)\) measurement noise

All \(B(P_i)\) and \(\varepsilon_i\) are assumed independent

• This model uses all data and gives a 95% c. i. for the method bias \(\alpha(A) - \alpha(B)\) which is: \((0.04; 0.41)\).

• Notice that now there is a (slightly) significant method bias.

<table>
<thead>
<tr>
<th>Person ID</th>
<th>Method A</th>
<th>Method B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.5</td>
<td>5.4</td>
</tr>
<tr>
<td>2</td>
<td>4.4</td>
<td>4.9</td>
</tr>
<tr>
<td>3</td>
<td>4.6</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>5.4</td>
<td>4.9</td>
</tr>
<tr>
<td>5</td>
<td>7.6</td>
<td>7.2</td>
</tr>
<tr>
<td>6</td>
<td>5.9</td>
<td>5.5</td>
</tr>
<tr>
<td>7</td>
<td>6.1</td>
<td>6.1</td>
</tr>
<tr>
<td>8</td>
<td>7.8</td>
<td>7.5</td>
</tr>
<tr>
<td>9</td>
<td>6.7</td>
<td>6.3</td>
</tr>
<tr>
<td>10</td>
<td>4.7</td>
<td>4.2</td>
</tr>
<tr>
<td>11</td>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>6.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>4.2</td>
<td></td>
</tr>
</tbody>
</table>
DATA_SECTION
  init_int nrow;
  init_int ncol;
  init_matrix obs(1,nrow,1,ncol);
  vector C(1,nrow);
  ivector P(1,nrow);
  ivector M(1,nrow);
 !! C=column(obs,3);
  !! P=(ivector)column(obs,1);
  !! M=(ivector)column(obs,2);
PARAMETER_SECTION
  init_number logSigmaP;
  init_number logSigmaR;
  init_vector alpha(1,2);
  random_effects_vector B(1,20);
  sdreport_number sigmaP;
  sdreport_number sigmaR;
  sdreport_number diffAB;
  vector pred(1,nrow);
  objective_function_value nll;
PROCEDURE_SECTION
  sigmaR=exp(logSigmaR);
  sigmaP=exp(logSigmaP);
  dvariable ss;
  nll=0.0;
  ss=square(sigmaR);
  for(int i=1; i<=nrow; ++i){
    pred(i)=alpha(M(i))+B(P(i));
    nll+=0.5*(log(2*M_PI*ss)+square(C(i)-pred(i))/ss);
  }
  ss=square(sigmaP);
  for(int i=1; i<=20; ++i){
    nll+=0.5*(log(2*M_PI*ss)+square(B(i))/ss);
  }
  diffAB=alpha(1)-alpha(2);
Random effects in AD Model Builder

- In random effects models we have
  - Random variables we observe: $x$
  - Random variables we do not observe: $z$
  - Model parameters we want to estimate: $\theta$

- If we had observed $x$ and $z$ we would have a joint likelihood $L(x, z, \theta)$

- But $z$ is unobserved so we have to estimate $\theta$ in the marginal likelihood:

  $$L(x, \theta) = \int L(x, z, \theta) dz$$

- This requires a high dimensional integral — which is difficult

- This is (part of) the reason MCMC methods are so widely used
  - MCMC can be slow, difficult to judge convergence, and in tools like winBugs a prior must be assigned to everything — even when you have no prior information.

- AD Model Builder has a better solution
**Laplace approximation**

- Want to compute the marginal likelihood for a given $\theta$ value:

\[
L(x, \theta) = \int L(x, z, \theta) dz
\]

- First the joint likelihood $L(x, z, \theta)$ is optimized w.r.t. $z$.

- This optimization yields an estimate $\hat{z}$, and an estimated hessian $\mathcal{H}(\hat{z})$.

- Next a Gaussian approximation is assumed and the result (apart from a constant) is:

\[
L(x, \theta) \approx |\text{det}(\mathcal{H}(\hat{z}))|^{-0.5} L(x, \hat{z}, \theta)
\]

- Notice that when defined in this way $\hat{z}$ and $\mathcal{H}(\hat{z})$ and also depend on $\theta$, which makes AD of this pretty difficult, but all solved for us in AD Model Builder.

- Actually this is all very simple to use. All we have to do is:
  - Code up the joint negative log likelihood
  - declare as `random_effects_vector z(1,n);`
Example: Estimating latent random walk

- Observation vector $Y$ generated from:
  - $\lambda_i = \lambda_{i-1} + \eta_i$
  - $Y_i = \lambda_i + \varepsilon_i$
  - where $i = 1 \ldots 50$, $\eta_i \sim \mathcal{N}(0, \sigma_{\lambda}^2)$, and $\varepsilon_i \sim \mathcal{N}(0, \sigma_Y^2)$ all independent.

- Notice $\lambda$ vector unobserved, and here we wish to estimate $\lambda$

- Knowing what we know now — how should we model this?

- Consider $\lambda$ as unobserved random variable
  - Estimate model parameters ($\sigma_{\lambda}$ and $\sigma_{\varepsilon}$) in marginal distribution $\int p(\lambda, Y) d\lambda$
  - Predict $\lambda$ via distribution of $\lambda|Y$
DATA_SECTION
init_int N
init_vector y(1,N)

PARAMETER_SECTION
init_number logSdLam
init_number logSdy
random_effects_vector lam(1,N);
objective_function_value jnll;

PROCEDURE_SECTION
jnll=0.0;
dvariable var;
var=exp(2.0*logSdLam);
for(int i=2; i<=N; ++i){
    jnll+=0.5*(log(2.0*M_PI*var) + square(lam(i)-lam(i-1))/var);
}
var=exp(2.0*logSdy);
for(int i=1; i<=N; ++i){
    jnll+=0.5*(log(2.0*M_PI*var) + square(lam(i)-y(i))/var);
}

TOP_OF_MAIN_SECTION
gradient_structure::set_MAX_NVAR_OFFSET(3000);
More efficient coding

- The idea is to reduce the likelihood calculation to a sum of function calls, where each call only uses a few random effects.
- Each function call must include the parameters needed, and the random effects needed, and not much more (no need to pass data)
- Function headers must be one line — even when they get too long.
Example: Discrete valued time series

- One of the examples from the AD Model Builder site (from Kuk & Cheng (1999)).
- The model:

\[
y_i \sim \text{Pois}(\lambda_i) , \text{ where } \\
\log(\lambda_i) = X_i b + u_i , \text{ and } \\
u_i = au_{i-1} + \varepsilon_i
\]

Here, $X_i$ is a covariate vector, $b$ is a vector of regression parameters and $u_i$ is an AR(1). The dimension of $b$ is 6 and $i=1,\ldots,168$.

<table>
<thead>
<tr>
<th></th>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
<th>$\beta_3$</th>
<th>$\beta_4$</th>
<th>$\beta_5$</th>
<th>$\beta_6$</th>
<th>$a$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMB-RE</td>
<td>0.242</td>
<td>-3.81</td>
<td>0.162</td>
<td>-0.482</td>
<td>0.413</td>
<td>-0.0109</td>
<td>0.627</td>
<td>0.538</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.270</td>
<td>2.76</td>
<td>0.15</td>
<td>0.16</td>
<td>0.13</td>
<td>0.13</td>
<td>0.19</td>
<td>0.15</td>
</tr>
<tr>
<td>Kuk &amp; Cheng</td>
<td>0.244</td>
<td>-3.82</td>
<td>0.162</td>
<td>-0.478</td>
<td>0.413</td>
<td>-0.0109</td>
<td>0.665</td>
<td>0.519</td>
</tr>
</tbody>
</table>
Discrete valued time series code

DATA_SECTION
    init_int n
    init_vector y(1,n)
    init_int p
    init_matrix X(1,n,1,p)

PARAMETER_SECTION
    init_vector b(1,p,1)
    init_bounded_number a(-1,1,2)
    init_number log_sigma(2)
    random_effects_vector u(1,n,2)
    objective_function_value g

PROCEDURE_SECTION
    g=0.0; int i;
    sf1(log_sigma,a,u(1));
    for (i=2;i<=n;i++)
    { sf2(log_sigma,a,u(i),u(i-1),i); }
    for (i=1;i<=n;i++)
    { sf3(u(i),b,i); }

SEPARABLE_FUNCTION void sf1(const dvariable& ls,const dvariable& aa,const dvariable& u_1)
    g += ls - 0.5*log(1-square(aa)) +0.5*square(u_1/exp(ls))*(1-square(aa));

SEPARABLE_FUNCTION void sf2(const dvariable& ls, const dvariable& aa,const dvariable& u_i, const dvariable& u_i1, int i)
    g += ls +.5*square((u_i-aa*u_i1)/exp(ls));

SEPARABLE_FUNCTION void sf3(const dvariable& u_i ,const dvar_vector& bb, int i)
    dvariable eta = X(i)*bb + u_i;
    dvariable lambda = exp(eta);
    g -= y(i)*eta - lambda;

TOP_OF_MAIN_SECTION
    gradient_structure::set_MAX_NVAR_OFFSET(1000);
Non-Gaussian random effects

- If the random effects are non-Gaussian, then the Laplace approximation may be inaccurate.
- Can use transformation \( g = F^{-1}(\Phi(u)) \), where \( u \sim \mathcal{N}(0, 1) \).
- E.g. part of a larger example:

```
PARAMETER_SECTION
  random_effects_vector u(1,nh,2)
PROCEDURE_SECTION
  for (i=1;i<=nh;i++){  
    fun(i,j,u(i),log_theta1,beta);
  }
SEPARABLE_FUNCTION void fun(int i,int & j ,const prevariable& ui, const prevariable& log_theta1, const dvar_vector& beta)
  f += 0.9189385 + 0.5*square(ui); // N(0,1) likelihood contribution from u's
  dvariable z=cumd_norm(ui); // z has uniform (0,1) distribution
  z = 0.99999999*z + 0.00000005; // To gain numerical stability
  dvariable gi = theta1*inv_cumd_gamma(z,1.0/theta1);
```

- In situations where we fear the Laplace approximation may be inaccurate, we can improve it by importance sampling. Simply by:

  ```
  .model -is 100
  ```
REML via random effects?

- The following non-linear model is assumed to describe the relation between density $D$ within pot and yield $Y$ per plant:

$$\log(Y_i) = -\log(\alpha + \beta D_i) + \varepsilon_i, \quad \text{where } \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

DATA_SECTION
init_int N
init_vector density(1,N)
init_vector yield(1,N)
vector logYield(1,N)
!! logYield=log(yield);

PARAMETER_SECTION
init_number logA
init_number logB
init_number logSigma
objective_function_value nll
sdreport_number a
sdreport_number b
sdreport_number sigma
vector pred(1,N)
number ss

PROCEDURE_SECTION
b=exp(logB);
a=exp(logA)-b*min(density);
sigma=exp(logSigma);
ss=square(sigma);
pred=-log(a+b*density);
nll=0.5*(N*log(2*M_PI*ss)+
    sum(square(logYield-pred))/ss);
Simulation study

<table>
<thead>
<tr>
<th>Parameter</th>
<th>True value</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.022</td>
<td>0.022</td>
<td>0.025</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.025</td>
<td>0.0249</td>
<td>0.0251</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.150</td>
<td>0.126</td>
<td>0.13</td>
</tr>
</tbody>
</table>

This is an expected result, but can we fix it?
Results with random effects (flat prior) on $\alpha$ and $\beta$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>True value</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.022</td>
<td>0.022</td>
<td>0.025</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.025</td>
<td>0.0249</td>
<td>0.0251</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.150</td>
<td>0.141</td>
<td>0.145</td>
</tr>
</tbody>
</table>

Almost!
What else?

- Large collection of examples at http://www.otter-rsch.com/admbre/examples.html
- It is possible to add priors on parameters (see exercise)
- The quality of the approximation can be checked and improved by importance sampling — without additional coding!
- Lots of flags for optimizing performance e.g. memory options — see manual.
- ...
Exercise: $\lambda_0$ in the latent random walk example

- As you may have noticed the model in the latent random walk example was not fully specified, as $\lambda_0$ was part of the model, but never defined.

- Here that is equivalent with assuming it has a uniform prior on $(-\infty, \infty)$.

? Estimate $\lambda_0$ via pure maximum likelihood estimation

? Estimate $\lambda_0$ with a Bayesian prior of $\lambda_0 \sim \mathcal{N}(0, 1)$

- Data for this exercise is:

<table>
<thead>
<tr>
<th>#No obs</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Y</td>
<td></td>
</tr>
<tr>
<td>-0.09399342 0.08762907 4.657932 3.38314 -0.1941568 0.5034158 -1.553094 -0.3431696</td>
<td></td>
</tr>
<tr>
<td>-1.673901 2.372934 -2.917300 0.8004703 3.21504 -3.170574 1.081191 -1.449991</td>
<td></td>
</tr>
<tr>
<td>1.001843 -3.627856 -3.369206 0.1883197 0.6740543 -0.1392156 -4.269124 0.4490485</td>
<td></td>
</tr>
<tr>
<td>-5.234534 0.2239184 0.2639806 -1.233715 2.179709 -1.988403 -0.1270127 -1.106568</td>
<td></td>
</tr>
<tr>
<td>-2.379884 -1.475134 -0.2455092 -1.625744 -7.538624 -7.015322 -10.31427 -2.727188</td>
<td></td>
</tr>
<tr>
<td>-2.325157 -4.770373</td>
<td></td>
</tr>
</tbody>
</table>
Solution

DATA_SECTION
init_int N
init_vector y(1,N)

PARAMETER_SECTION
init_number logSdLam
init_number logSdy
init_number lam0; //--- add model parameter---
random_effects_vector lam(1,N);
objective_function_value jnll;

PROCEDURE_SECTION
jnll=0.0;
dvariable var;
var=exp(2.0*logSdLam);

    //--- Include it in jnll ---
    jnll+=0.5*(log(2.0*M_PI)+square(lam(1)-lam0)/var);

for(int i=2; i<=N; ++i){
    jnll+=0.5*(log(2.0*M_PI)+square(lam(i)-lam(i-1))/var);
}

var=exp(2.0*logSdy);
for(int i=1; i<=N; ++i){
    jnll+=0.5*(log(2.0*M_PI)+square(lam(i)-y(i))/var);
}

TOP_OF_MAIN_SECTION
gradient_structure::set_MAX_NVAR_OFFSET(3000);
Exercise: Random effect logistic regression

- Read through the example at:
  

- Implement the same model in ADMB, but without priors on the hyper parameters.

- Compare results.

- The data for this exercise is:

```plaintext
#N
21
#r
10 23 23 26 17 5 53 55 32 46 10 8 10 8 23 0 3 22 15 32 3
#n
39 62 81 51 39 6 74 72 51 79 13 16 30 28 45 4 12 41 30 51 7
#x1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
#x2
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
```
Solution

**DATA_SECTION**

init_int N;
init_ivector r(1,N);
init_ivector n(1,N);
init_ivector x1(1,N);
init_ivector x2(1,N);

**PARAMETER_SECTION**

init_number alpha0
init_number alpha1
init_number alpha2
init_number alpha12
init_number logSigma
random_effects_vector B(1,N)
sdreport_number sigma
vector logitp(1,N)
vector p(1,N)

**PROCEDURE_SECTION**

sigma=exp(logSigma);
logitp=alpha0+alpha1*x1+alpha2*x2+alpha12*elem_prod(x1,x2)+B;
p=elem_div(exp(logitp),(1.0+exp(logitp)));

jnll=0.0;
for(int i=1; i<=N; ++i){
    jnll+=log_comb(n(i),r(i))-log(p(i))*r(i)-log(1.0-p(i))*(n(i)-r(i));
    jnll+=0.5*(log(2.0*M_PI*square(sigma))+square(B(i)/sigma));
}

Winbugs code

model{
  for( i in 1 : N ) {
    r[i] ~ dbin(p[i],n[i])
    b[i] ~ dnorm(0.0,tau)
    logit(p[i]) <- alpha0 + alpha1 * x1[i] + alpha2 * x2[i] +
        alpha12 * x1[i] * x2[i] + b[i]
  }
  alpha0 ~ dnorm(0.0,1.0E-6)
  alpha1 ~ dnorm(0.0,1.0E-6)
  alpha2 ~ dnorm(0.0,1.0E-6)
  alpha12 ~ dnorm(0.0,1.0E-6)
  tau ~ dgamma(0.001,0.001)
  sigma <- 1 / sqrt(tau)
}
list(r = c(10, 23, 23, 26, 17, 5, 53, 55, 32, 46, 10, 8, 10, 8, 23, 0, 3, 22, 15, 32, 3),
  n = c(39, 62, 81, 51, 39, 6, 74, 72, 51, 79, 13, 16, 30, 28, 45, 4, 12, 41, 30, 51, 7),
  x1 = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
  x2 = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1),
  N = 21
)
list(alpha0 = 0, alpha1 = 0, alpha2 = 0, alpha12 = 0, tau = 10)