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

\[ B(P_i) \sim N(0, \sigma^2_P) \] the 20 random effects

\[ \varepsilon_i \sim N(0, \sigma^2_R) \] 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></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></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 $\mathcal{H}(\hat{z})$ 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 $i = 1 \ldots 50$, $\eta_i \sim N(0, \sigma^2_{\lambda})$, and $\varepsilon_i \sim N(0, \sigma^2_Y)$ all independent.

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

- 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);

<table>
<thead>
<tr>
<th>index</th>
<th>name</th>
<th>value</th>
<th>std dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>logSdLam</td>
<td>-2.3576e-01</td>
<td>3.3713e-01</td>
</tr>
<tr>
<td>2</td>
<td>logSdy</td>
<td>8.1161e-01</td>
<td>1.1955e-01</td>
</tr>
<tr>
<td>3</td>
<td>lam</td>
<td>9.4885e-01</td>
<td>1.2231e+00</td>
</tr>
<tr>
<td>4</td>
<td>lam</td>
<td>1.0772e+00</td>
<td>1.0988e+00</td>
</tr>
<tr>
<td>5</td>
<td>lam</td>
<td>1.3274e+00</td>
<td>1.0810e+00</td>
</tr>
<tr>
<td>6</td>
<td>lam</td>
<td>1.1676e+00</td>
<td>1.0275e+00</td>
</tr>
<tr>
<td>7</td>
<td>lam</td>
<td>7.3510e-01</td>
<td>9.6103e-01</td>
</tr>
<tr>
<td>8</td>
<td>lam</td>
<td>4.1696e-01</td>
<td>9.4607e-01</td>
</tr>
<tr>
<td>9</td>
<td>lam</td>
<td>8.8186e-02</td>
<td>9.5341e-01</td>
</tr>
<tr>
<td>10</td>
<td>lam</td>
<td>-3.8547e-02</td>
<td>9.5028e-01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

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

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 + 0.000000005; // 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: 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 1 0 0 0 0 0 1 1 1 1 1
```
Solution

DATA_SECTION
init_int N;
init_vector r(1,N);
init_vector n(1,N);
init_vector x1(1,N);
init_vector 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)
objective_function_value jnll

PROCEDURE_SECTION

index name value std dev
1 alpha0 -5.4849e-01 1.6611e-01
2 alpha1 9.7427e-02 2.7739e-01
3 alpha2 1.3368e+00 2.3623e-01
4 alpha12 -8.1003e-01 3.8422e-01
5 logSigma -1.4497e+00 4.6691e-01
6 B -1.5854e-01 2.2444e-01
7 B 9.0476e-03 1.8999e-01
8 B -1.8273e-01 2.0536e-01
9 B 2.4140e-01 2.3428e-01
10 B 9.9434e-02 2.0804e-01
11 B 4.5013e-02 2.3020e-01
12 B 6.2799e-02 1.8985e-01
13 B 1.6606e-01 2.0487e-01
14 B -1.0436e-01 2.0653e-01
15 B -2.3195e-01 2.2041e-01
16 B 5.0781e-02 2.2308e-01
17 B 8.0648e-02 2.2664e-01
18 B -6.6290e-02 2.1167e-01
19 B -1.1708e-01 2.2198e-01
20 B 1.8894e-01 2.3589e-01
21 B -8.1461e-02 2.3978e-01
22 B -1.5248e-01 2.4853e-01
23 B 2.5509e-02 2.0398e-01
24 B -2.2122e-02 2.0649e-01
25 B 1.7960e-01 2.2983e-01
26 B -3.1760e-02 2.2604e-01
27 sigma 2.3463e-01 1.0955e-01

\[
\sigma = \exp(\log\sigma);
\]
\[
\logitp = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_{12} \text{elem}_\prod(x_1, x_2) + B;
\]
\[
p = \text{elem}_\div(\exp(\logitp), (1.0 + \exp(\logitp)));
\]
\[
\text{jnll} = 0.0;
\]
for(int i=1; i<=N; ++i){
    \[
    \text{jnll} += -\log\text{comb}(n(i), r(i)) - \log(p(i)) * r(i) - \log(1.0 - p(i)) * (n(i) - r(i));
    \]
    \[
    \text{jnll} += 0.5*(\log(2.0*\text{M_PI}*\text{square}(\sigma)) + \text{square}(B(i)/\sigma));
    \]
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, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
     x2 = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1),
     N = 21)

list(alpha0 = 0, alpha1 = 0, alpha2 = 0, alpha12 = 0, tau = 10)