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.

Person ID	Method A	Method B
1	5.5	5.4
2	4.4	4.9
3	4.6	4.5
4	5.4	4.9
5	7.6	7.2
6	5.9	5.5
7	6.1	6.1
8	7.8	7.5
9	6.7	6.3
10	4.7	4.2

- 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

Person ID	Method A	Method B
1	5.5	5.4
2	4.4	4.9
3	4.6	4.5
4	5.4	4.9
5	7.6	7.2
6	5.9	5.5
7	6.1	6.1
8	7.8	7.5
9	6.7	6.3
10	4.7	4.2
11		5.1
12		4.4
13		4.5
14		5.3
15		7.5
16	5.7	
17	6.0	
18	7.5	
19	6.5	
20	4.2	

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

- Assume these 20 are ramdomly selected from a population where the blod cell count is normally distributed

- Consider the following model:
  $C_i = \alpha(M_i) + B(P_i) + \varepsilon_i, \quad i = 1 \dots 30$
  $\alpha(M_i)$ the 2 fixed method effects
  $B(P_i) \sim \mathcal{N}(0, \sigma_P^2)$ the 20 random effects
  $\varepsilon_i \sim \mathcal{N}(0, \sigma_R^2)$ measurement noise
  All $B(P_i)$ and ε_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.

```
#No rows
 30
#No cols
 3
#The obs matrix
#P M C
1 1 5.5
2 1 4.4
3 1 4.6
4 1 5.4
5 1 7.6
6 1 5.9
7 1 6.1
8 1 7.8
9 1 6.7
10 1 4.7
16 1 5.7
17 1 6
18 1 7.5
19 1 6.5
20 1 4.2
1 2 5.4
2 2 4.9
3 2 4.5
4 2 4.9
5 2 7.2
6 2 5.5
7 2 6.1
8 2 7.5
9 2 6.3
10 2 4.2
11 2 5.1
12 2 4.4
13 2 4.5
14 2 5.3
15 2 7.5
```

```
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: θ
- 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 θ 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 θ value:

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

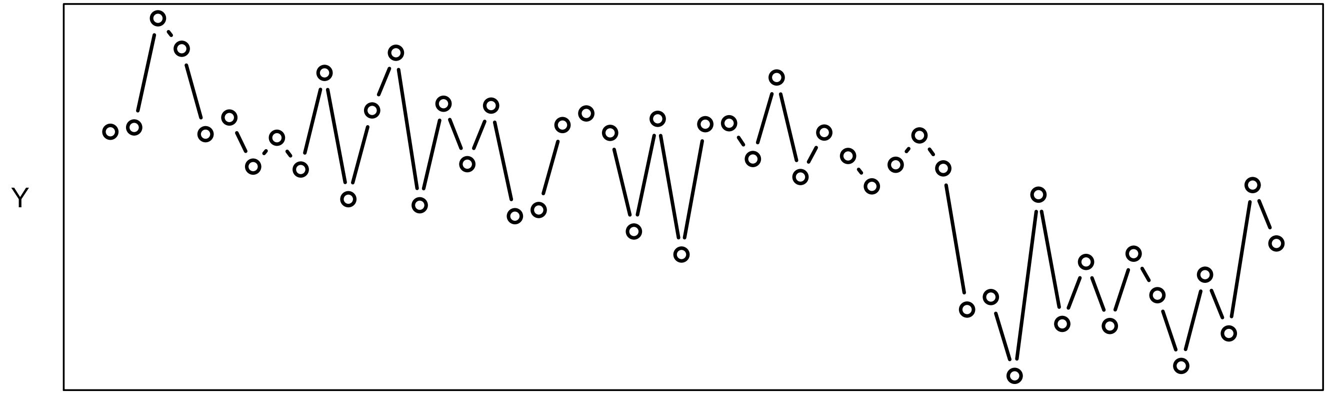
- First the joint likelihood $L(x,z,\theta)$ is optimized w.r.t. z.
- This optimization yields an estimate $\hat{z}$, and an estimated hessian $\mathcal{H}(\hat{z})$.
- Next a Gaussian approximation is assumed and the result (apart from a constant) is:

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

- Notice that when defined in this way $\hat{z}$ and $\mathcal{H}(\hat{z})$ and also depend on θ, 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 λ vector unobserved, and here we wish to estimate λ
- Knowing what we know now — how should we model this?
- Consider λ as unobserved random variable
  - Estimate model parameters (σ_λ and σ_ε) in marginal distribution $\int p(\lambda, Y) d\lambda$
  - Predict λ via distribution of $\lambda | Y$

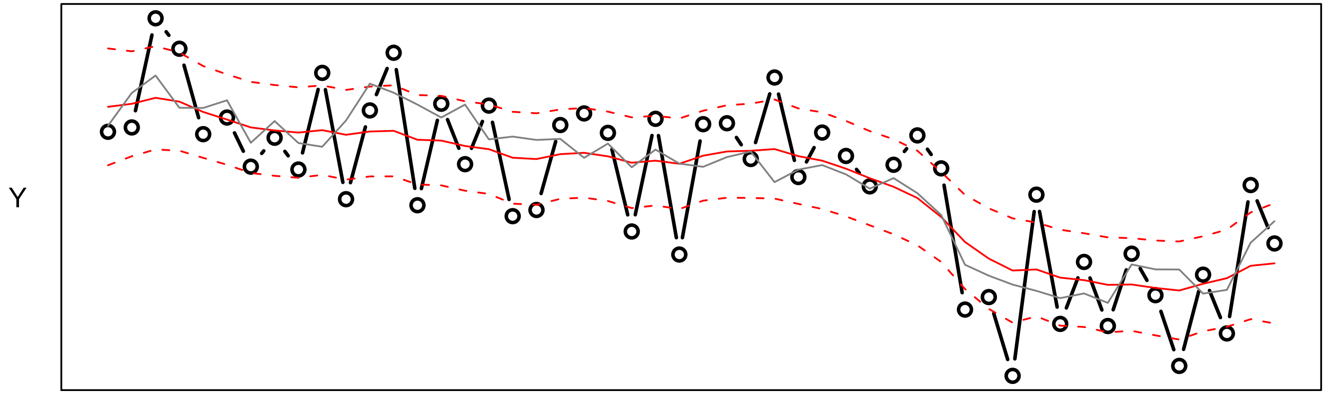
```
DATA_SECTION
  init_int N
  init_vector y(1,N)

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);
```

index	name	value	std dev
1	logSdLam	-2.3576e-01	3.3713e-01
2	logSdy	8.1161e-01	1.1955e-01
3	lam	9.4885e-01	1.2231e+00
4	lam	1.0772e+00	1.0988e+00
5	lam	1.3274e+00	1.0810e+00
6	lam	1.1676e+00	1.0275e+00
7	lam	7.3510e-01	9.6103e-01
8	lam	4.1696e-01	9.4607e-01
9	lam	8.8186e-02	9.5341e-01
10	lam	-3.8547e-02	9.5028e-01
.			
.			
.			
43	lam	-6.2033e+00	1.0043e+00
44	lam	-6.3070e+00	9.8098e-01
45	lam	-6.5045e+00	9.9328e-01
46	lam	-6.4900e+00	9.7044e-01
47	lam	-6.6344e+00	9.9289e-01
48	lam	-6.7417e+00	1.0266e+00
49	lam	-6.4604e+00	9.9465e-01
50	lam	-6.2260e+00	1.0168e+00
51	lam	-5.7070e+00	1.1180e+00
52	lam	-5.6044e+00	1.2585e+00



More efficient coding

```
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;

  for(int i=2; i<=N; ++i){
    step(lam(i-1),lam(i),logSdLam);
  }

  for(int i=1; i<=N; ++i){
    obs(lam(i),logSdy,i);
  }
```

- 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.

```
SEPARABLE_FUNCTION void step(const dvariable& lam1, const dvariable& lam2, const dvariable& logSdLam)
  dvariable var=exp(2.0*logSdLam);
  jnll+=0.5*(log(2.0*M_PI*var)+square(lam2-lam1)/var);
SEPARABLE_FUNCTION void obs(const dvariable& lam, const dvariable& logSdy, int i)
  dvariable var=exp(2.0*logSdy);
  jnll+=0.5*(log(2.0*M_PI*var)+square(lam-y(i))/var);
TOP_OF_MAIN_SECTION
  gradient_structure::set_MAX_NVAR_OFFSET(3000);
```

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)
- Restricted maximum likelihood estimation can be obtained (at least for linear Gaussian models) by making the mean parameters random effects with flat priors.
- Non-linear models and models with non-Gaussian random effects can be approximated.
- 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: λ_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 λ_0 was part of the model, but never defined.
- Here that is equivalent with assuming it has a uniform prior on $(-\infty, \infty)$.

? Estimate λ_0 via pure maximum likelihood estimation

? Estimate λ_0 with a Bayesian prior of $\lambda_0 \sim \mathcal{N}(0, 1)$

- Data for this exercise is:

```
#No obs
 50
#Y
 -0.09399342 0.08762907 4.657932 3.38314 -0.1941568 0.5034158 -1.553094 -0.3431696
-1.673901 2.372934 -2.917300 0.8004703 3.21504 -3.170574 1.081191 -1.449991
1.001843 -3.627856 -3.369206 0.1883197 0.6740543 -0.1392156 -4.269124 0.4490485
-5.234534 0.2239184 0.2639806 -1.233715 2.179709 -1.988403 -0.1270127 -1.106568
-2.379884 -1.475134 -0.2455092 -1.625744 -7.538624 -7.015322 -10.31427 -2.727188
-8.139333 -5.544363 -8.227553 -5.198673 -6.936379 -9.898509 -6.07848 -8.538303
-2.325157 -4.770373
```

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*var)
             +square(lam(1)-lam0)/var);

  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);
```

```
DATA_SECTION
  init_int N
  init_vector y(1,N)

PARAMETER_SECTION
  init_number logSdLam
  init_number logSdy
  init_number lam0;
  random_effects_vector lam(1,N);
  objective_function_value jnll;

PROCEDURE_SECTION
  jnll=0.0;
  dvariable var;

             //--- Add prior N(0,1) to lam0 ---
  jnll+=0.5*(log(2.0*M_PI)+square(lam0));

  var=exp(2.0*logSdLam);
  jnll+=0.5*(log(2.0*M_PI*var)
             +square(lam(1)-lam0)/var);

  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);
```

Exercise: Random effect logistic regression

- Read through the example at:

  http://mathstat.helsinki.fi/openbugs/Examples/Seeds.html

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

? Compare results.

- The data for this exercise is:

```
#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_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)

  objective_function_value jnll
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));
  }
```

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