

## Specifying model parameters

Anders Nielsen & Arni Magnusson

## PARAMETER\_SECTION

- Is where model parameters should be specified
- Model parameters can be fixed at their initial values
- Bounds can be specified
- Phases can be specified (Start by estimating these, then include those, and so on)
- Additional variables for intermediate calculations, and additional outputs can be declared
- Finally the it should name the function to be minimized by a line like:  
`objective_function_value nll;`
- AD Model Builder will keep track of the derivatives for the quantities declared here

```
objective_function_value nll;
```

# A single parameter

**Unbounded, and active parameter:** The most basic parameter is declared by:

```
init_number theta;
```

It is initialized to zero, if no other initialization is done (in program, or via a .pin file).

**Bounded, and active parameter:** Bounds are added by:

```
init_bounded_number p(0,1);
```

It is initialized to the mid-point of the interval (here 0.5), if no other initialization is done.

**Fixed parameter:** To fix a parameter at its initial value, simply add a '-1':

```
init_number theta(-1);
```

```
init_bounded_number p(0,1,-1);
```

Then its value will not be estimated.

**Optimization in phases:** In models with multiple parameters it is possible to have a parameter fixed during the first optimization (of some other parameters), and the active in the second or third. The phase from which it is set active ('1','2',or '3') is specified by:

```
init_number theta(2);
```

estimated in second phase

```
init_bounded_number p(0,1,3);
```

estimated in third phase

# Vectors of parameters

**Unbounded, and always active:** A vector with 3 elements and valid index from 1 to 3

```
init_vector theta(1,3);
```

It is initialized to zero by default

**Bounded, and active:** A vector with valid index from 0 to 5 and each element in  $]-1,3[$

```
init_bounded_vector theta(0,5,-1,3);
```

It is initialized to the mid-point of the interval (here 1), if no other initialization is done.

**Fixed:** To fix all elements of a parameter vector we add a '-1' argument:

```
init_vector theta(1,3,-1);
```

```
init_bounded_vector theta(0,5,-1,3,-1);
```

Then they are kept at initialization.

**Optimization in phases:** The active optimization phase can be set for vectors too:

```
init_vector theta(1,3,2);
```

estimated in second phase

```
init_bounded_vector theta(0,5,-1,3,3);
```

estimated in third phase

By default three phases are available, but more is possible (see ADMB manual).

**Parameter vector summing to zero:** To declare a parameter vector that is optimized such that it sums to zero:

```
init_bounded_dev_vector epsilon(1,20,-10,10,2)
```

**Vectors of individual parameters:** Consider the following example:

<pre>DATA_SECTION   init_int N   init_vector Y(1,N)   init_vector x(1,N)   init_vector lb(1,3)   init_vector ub(1,3)   init_ivector phase(1,3)  PARAMETER_SECTION   init_bounded_number_vector th(1,3,lb,ub,phase)   objective_function_value nll  PROCEDURE_SECTION   nll=0.5*(N*log(2*M_PI*square(th(3)))     +sum(square(Y-(th(1)+th(2)*x)))/square(th(3)));</pre>	<pre># number of observations 10 # observed Y values 1.4 4.7 5.1 8.3 9.0 14.5 14.0 13.4 19.2 18 # observed x values -1 0 1 2 3 4 5 6 7 8 # lower bounds a b sigma -100 -100 0 # upper bounds a b sigma 100 100 100 # phase a b sigma 1 1 2</pre>
---	--

The logarithm of the determinant of the hessian = 7.64036

index	name	value	std dev	1	2	3
1	th[1]	4.0782e+00	7.0394e-01	1.0000		
2	th[2]	1.9091e+00	1.5547e-01	-0.7730	1.0000	
3	th[3]	1.4122e+00	3.1577e-01	0.0000	0.0000	1.0000

An unbounded version `init_number_vector` is also available.

# Now we start to see a pattern

Declaration	type of object in DATA SECTION	type of object in PARAMETER SECTION
[init_]int	int	int
[init_] [bounded_]number	double	dvariable
[init_] [bounded_] [dev_]vector	vector of doubles(dvector)	vector of dvariables(dvar_vector)
[init_] [bounded_]matrix	matrix of doubles(dmatrix)	matrix of dvariables(dvar_matrix)
[init_]3darray	3 dimensional array of doubles	3 dimensional array of dvariables
4darray	4 dimensional array of doubles	4 dimensional array of dvariables
5darray	5 dimensional array of doubles	5 dimensional array of dvariables
6darray	6 dimensional array of doubles	6 dimensional array of dvariables
7darray	7 dimensional array of doubles	7 dimensional array of dvariables
sdreport_number	na	dvariable
sdreport_vector	na	vector of dvariables(dvar_vector)
sdreport_matrix	na	matrix of dvariables(dvar_matrix)

In the **PARAMETER\_SECTION** the following rules apply:

- Everything starting with **init\_** is optimized (unless phase is set to '-1').
- For everything starting with **sdreport\_** AD Model Builder is instructed to estimate standard errors and correlations.
- Other **number**, **vector**, **matrix**, ... variables are used to store intermediate calculations

# How are parameters initialized

- If none of the following methods are used, the default is to set unbounded parameters to **zero** and bounded parameters to the **interval midpoint**
- This default behaviour can be overruled by using the **INITIALIZATION\_SECTION** as in:

```
PARAMETER_SECTION
  init_bounded_number_vector th(1,3,lb,ub,phase)
  objective_function_value nll
INITIALIZATION_SECTION
  th 1;
```

- Whatever is specified by default or in the **INITIALIZATION\_SECTION** can be overwritten by supplying initial values in the **<modelname>.pin** file. Such a file could look like:

```
#th
4 2 1.5
```

with the values appearing in the same order as the parameters in the **PARAMETER\_SECTION**

- A final way to set initial values, which will overwrite all methods above is to use the **PRELIMINARY\_CALCS\_SECTION** as in:

```
PARAMETER_SECTION
  init_bounded_number_vector th(1,3,lb,ub,phase)
  objective_function_value nll
PRELIMINARY_CALCS_SECTION
  th(1)=2;
  th(2)=2;
  th(3)=2;
```

# Transformations

- Bounded optimization in ADMB works great, but sometimes we need something different, or are in a situation where we prefer to a parameter transformation. Consider:

```
DATA_SECTION
  init_int N
  init_vector Y(1,N)
  init_vector x(1,N)

PARAMETER_SECTION
  init_number a
  init_number b
  init_number logSigma
  sdreport_number sigmasq
  objective_function_value nll

PROCEDURE_SECTION
  sigmasq=exp(2*logSigma);
  nll=0.5*(N*log(2*M_PI*sigmasq)+sum(square(Y-(a+b*x)))/sigmasq);
```

- We know  $\sigma$  must be positive, but we don't know the upper limit
- After estimation we want to supply a confidence interval. If we calculate it as  $\sigma \in ]\hat{\sigma} - 2sd(\hat{\sigma}); \hat{\sigma} + 2sd(\hat{\sigma})[$  it could get a negative lower bound.
- If we use the transformed variable  $\xi = \log(\sigma)$  we get:  $\sigma \in ]e^{\hat{\xi}-2sd(\hat{\xi})}; e^{\hat{\xi}+2sd(\hat{\xi})}[$ , which we know is entirely positive.



# Probability vector

- For a single probability parameter we can use the inverse logit transformation

$$p = \exp(\alpha) / (1 + \exp(\alpha)), \quad \text{where } \alpha \in \mathcal{R}$$

- For a probability vector  $p = (p_1, \dots, p_n) \in ]0, 1[^n$  with  $\sum p = 1$  we can use the following transformation:

$$p = \begin{pmatrix} \exp(\alpha_1) / (1 + \sum_{i=1}^{n-1} \exp(\alpha_i)) \\ \exp(\alpha_2) / (1 + \sum_{i=1}^{n-1} \exp(\alpha_i)) \\ \vdots \\ \exp(\alpha_{n-1}) / (1 + \sum_{i=1}^{n-1} \exp(\alpha_i)) \\ 1 - \sum_{i=1}^{n-1} \exp(\alpha_i) / (1 + \sum_{i=1}^{n-1} \exp(\alpha_i)) \end{pmatrix} \quad \text{where } \alpha \in \mathcal{R}^{n-1}$$

- Example using a probability vector:

```
DATA_SECTION
  init_int dim;
  init_vector X(1,dim);

PARAMETER_SECTION
  init_vector a(1,dim-1);
  sdreport_vector p(1,dim);
  objective_function_value nll;

PROCEDURE_SECTION
  p=a2p(a);
  nll=-gammln(sum(X)+1.0)+sum(gammln(X+1.0))
    -sum(elem_prod(X,log(p)));

FUNCTION dvar_vector a2p(const dvar_vector& a)
  dvar_vector p(1,dim);
  dvar_vector expa=exp(a);
  p(1,dim-1)=expa/(1+sum(expa));
  p(dim)=1-sum(p(1,dim-1));
  return p;
```

6  
10 16 31 13 14 16

index	name	value	std dev
1	a	-4.7000e-01	4.0311e-01
2	a	9.5044e-07	3.5355e-01
3	a	6.6140e-01	3.0783e-01
4	a	-2.0764e-01	3.7339e-01
5	a	-1.3353e-01	3.6596e-01
6	p	1.0000e-01	3.0000e-02
7	p	1.6000e-01	3.6661e-02
8	p	3.1000e-01	4.6249e-02
9	p	1.3000e-01	3.3630e-02
10	p	1.4000e-01	3.4699e-02
11	p	1.6000e-01	3.6661e-02

# Exercises

**Exercise 1:** Suggest how to use transformation to parametrize a parameter that is

- a) only negative
- b) between 2 and 5
- c) an increasing vector

**Solution:** Consider the following transformations

- a)  $\theta = -e^\alpha$ , where  $\alpha \in \mathcal{R}$
- b)  $\theta = 3e^\alpha / (1 + e^\alpha) + 2$ , where  $\alpha \in \mathcal{R}$
- c)  $\theta = (e^{\alpha_1}, e^{\alpha_1} + e^{\alpha_2}, \dots, e^{\alpha_1} + \dots + e^{\alpha_n})$ , where  $\alpha \in \mathcal{R}^n$

**Exercise 2:** To investigate the effect of a certain type of exposure in three doses (1,2,and 3) the following experiment was carried out. The experimental unit was a cage with 2 rats. Once per month in 10 months the activity was measured as number of crossing of a light beam. The data can be seen on the next page. It must be expected that measurements from same cage are correlated, and even that measurements close in time have higher correlations.

The following model was proposed:

$$\log(\text{count}) \sim \mathcal{N}(\mu, \Sigma), \quad \text{where}$$

$$\mu_i = \alpha(\text{dose}_i, \text{month}_i), \quad i = 1 \dots 300$$

$$\Sigma_{i,j} = \begin{cases} 0, & \text{if } \text{cage}_i \neq \text{cage}_j \\ \nu^2 + \tau^2 \exp\left\{\frac{-(\text{month}_i - \text{month}_j)^2}{\rho^2}\right\}, & \text{if } \text{cage}_i = \text{cage}_j \text{ and } i \neq j \\ \nu^2 + \tau^2 + \sigma^2, & \text{if } i = j \end{cases}$$

? Implement the model and remember that the variance parameters should be positive.

! The negative log density for the multivariate normal distribution is:

$$\ell(x, \mu, \Sigma) = \frac{1}{2} (N \log(2\pi) + \log |\Sigma| + (x - \mu)' \Sigma^{-1} (x - \mu))$$

Dose	Cage	Month									
		1	2	3	4	5	6	7	8	9	10
1	1	20584	15439	17376	14785	11189	10366	8725	9974	9576	6849
1	2	23265	16956	16200	12934	13763	11893	9949	10490	8674	7153
1	3	17065	12429	14757	10524	11783	8828	9016	9635	8028	8099
1	4	19265	19316	20598	16619	16092	13422	10532	10614	9466	9494
1	5	21062	14095	13267	12543	12734	12268	12219	11791	10379	8463
1	6	23456	10939	13270	14089	12986	13723	11878	13338	12442	10094
1	7	13383	11899	12531	15081	14295	13650	9988	11518	11915	7844
1	8	22717	22434	23151	13163	10029	10408	9119	10188	9549	11153
1	9	17437	13950	15535	14199	11540	9568	8481	9143	8117	5765
1	10	18546	12520	15394	10137	9218	7343	6702	7173	7257	5708
2	11	18536	16827	19185	12445	13227	10412	9855	9169	9639	6853
2	12	18831	14043	16493	12562	10397	8568	8599	8818	6011	5062
2	13	15016	13765	16648	14537	13929	10778	9897	9225	9491	5523
2	14	22276	15497	22024	15616	12440	11454	10290	9456	9567	7003
2	15	18943	14834	18403	16232	13085	12679	10489	9495	10896	8836
2	16	13598	10233	13392	10457	9236	8847	9445	9501	8509	5656
2	17	20498	22136	22094	19825	18157	11452	14809	14564	14503	10643
2	18	19586	12710	12745	7294	15757	15296	14097	14308	13933	10210
2	19	11474	8108	17714	16795	17364	16766	15016	13475	14349	8698
2	20	10284	10760	15628	10692	8420	5842	6138	10271	8435	4486
3	21	18459	15805	19924	18337	24197	18790	19333	22234	18291	11595
3	22	16186	11750	16470	18637	14862	14695	14458	14228	12909	9079
3	23	9614	8319	11375	9446	13157	11153	10540	11476	8976	6123
3	24	15688	15016	20929	12706	17351	15089	14605	15952	14795	10434
3	25	15864	13169	20991	20655	19763	19180	19003	18172	15025	11790
3	26	17721	14489	19085	21333	17011	16148	15280	14762	15745	10477
3	27	17606	7558	15646	15194	13036	10316	8172	8977	8378	3962
3	28	34907	29247	35831	15093	9754	10061	9042	11732	8716	4922
3	29	15189	14046	14909	14713	14999	14201	13184	13073	14639	10330
3	30	16388	14538	17548	19416	22034	17761	14488	16068	14773	10595

```

#noCages
30
#noMonths
10
# month
1 2 3 4 5 6 7 8 9 10
# dose
1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
# cage
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
# counts
20584 15439 17376 14785 11189 10366 8725 9974 9576 6849
23265 16956 16200 12934 13763 11893 9949 10490 8674 7153
17065 12429 14757 10524 11783 8828 9016 9635 8028 8099
19265 19316 20598 16619 16092 13422 10532 10614 9466 9494
21062 14095 13267 12543 12734 12268 12219 11791 10379 8463
23456 10939 13270 14089 12986 13723 11878 13338 12442 10094
13383 11899 12531 15081 14295 13650 9988 11518 11915 7844
22717 22434 23151 13163 10029 10408 9119 10188 9549 11153
17437 13950 15535 14199 11540 9568 8481 9143 8117 5765
18546 12520 15394 10137 9218 7343 6702 7173 7257 5708
18536 16827 19185 12445 13227 10412 9855 9169 9639 6853
18831 14043 16493 12562 10397 8568 8599 8818 6011 5062
15016 13765 16648 14537 13929 10778 9897 9225 9491 5523
22276 15497 22024 15616 12440 11454 10290 9456 9567 7003
18943 14834 18403 16232 13085 12679 10489 9495 10896 8836
13598 10233 13392 10457 9236 8847 9445 9501 8509 5656
20498 22136 22094 19825 18157 11452 14809 14564 14503 10643
19586 12710 12745 7294 15757 15296 14097 14308 13933 10210
11474 8108 17714 16795 17364 16766 15016 13475 14349 8698
10284 10760 15628 10692 8420 5842 6138 10271 8435 4486
18459 15805 19924 18337 24197 18790 19333 22234 18291 11595
16186 11750 16470 18637 14862 14695 14458 14228 12909 9079
9614 8319 11375 9446 13157 11153 10540 11476 8976 6123
15688 15016 20929 12706 17351 15089 14605 15952 14795 10434
15864 13169 20991 20655 19763 19180 19003 18172 15025 11790
17721 14489 19085 21333 17011 16148 15280 14762 15745 10477
17606 7558 15646 15194 13036 10316 8172 8977 8378 3962
34907 29247 35831 15093 9754 10061 9042 11732 8716 4922
15189 14046 14909 14713 14999 14201 13184 13073 14639 10330
16388 14538 17548 19416 22034 17761 14488 16068 14773 10595

```

## Solution: The following program implements the model

```
DATA_SECTION
  init_int noCages
  init_int noMonths
  init_ivector month(1,noMonths)
  init_ivector dose(1,noCages)
  init_ivector cage(1,noCages)
  init_matrix counts(1,noCages,1,noMonths)
  matrix lc(1,noCages,1,noMonths)
  !! lc=log(counts);

PARAMETER_SECTION
  init_matrix DxM(1,3,1,noMonths);
  init_number logNu;
  init_number logTau;
  init_number logSigma;
  init_number logRho;

  sdreport_number nu2;
  sdreport_number tau2;
  sdreport_number sigma2;
  sdreport_number rho2;

  objective_function_value nll;

PROCEDURE_SECTION
  nu2=exp(2*logNu);
  tau2=exp(2*logTau);
  sigma2=exp(2*logSigma);
  rho2=exp(2*logRho);

  dvar_matrix S(1,noMonths,1,noMonths);
  for(int i=1; i<=noMonths; ++i){
    S(i,i)=nu2+tau2+sigma2;
    for(int j=i+1; j<=noMonths; ++j){
      S(i,j)=nu2+tau2*exp(-square(month(i)-month(j))/rho2);
      S(j,i)=S(i,j);
    }
  }
  dvar_matrix Sinv=inv(S);
  dvariable logdet=log(det(S));

  nll=0.0;
  for(int i=1; i<=noCages; ++i){
    nll+=mvdnormi(lc(i),DxM(dose(i)),Sinv,logdet);
  }

FUNCTION dvariable mvdnormi(const dvector& x, const dvar_vector& mu, const dvar_matrix& Sinv, const dvariable& logdet)
  dvar_vector diff=x-mu;
  return 0.5*(log(2.0*M_PI)*noMonths+logdet+diff*Sinv*diff);
```

