Affirma™ Spectre® DC Device Matching Analysis Tutorial

The procedure described in this application note are deliberately broad and generic. Requirements for your specific design may dictate procedures slightly different from those described here.

Purpose

This application note describes how to

- Use the dcmatch command in SPECTRE.
- Examine the dcmatch results generated by SPECTRE.
- Describe the models, equations, and theory of dcmatch analysis.

Audience

Circuit designers who want to find out dc variation and predict dc accuracy of their designs.

Overview

The DCMATCH analysis performs DC device mismatch analysis for a given output. It computes the deviation in the DC operating point of the circuit caused by mismatch in the devices. Users need to specify mismatch parameters in their model cards for each device contributing to the deviation. The analysis uses the device mismatch models to construct equivalent mismatch current sources in parallel to all the devices that have mismatch modeled. These current sources will have zero mean and some variance. The analysis computes the 3-sigma variance of dc voltages or currents at user specified outputs due to the mismatch current sources. The simulation results displays the devices rank ordered by their contributions to the outputs. In addition, for mosfet devices (limited to bsim3v3 now), it displays threshold voltage mismatch, current factor mismatch, gate voltage mismatch, and drain current mismatch. For bipolar devices (limited to vbic now), it displays base-emitter junction voltage mismatch. For resistors (limited to two terminal resistance), it displays resistor mismatches.

The analysis replaces multiple simulation runs in the estimation of circuit accuracies. It automatically identifies the set of critical matched components during circuit design. For example, when there are matched pairs in the circuit, the contribution of two matched transistors will be equal in magnitude and opposite in sign.

Typical usage are to simulate the output offset voltage of operational amplifiers, estimate the variation in bandgap voltages, and predict the accuracy of current steering DACs.

Help syntax

To obtain the command syntax on-line, users can type

spectre -h dcmatch

Command syntax

The syntax of the dcmatch analysis is as follows:

Name [pnode nnode] dcmatch <parameter=value> ...

where the parameters are

1 mth

Relative mismatch contribution threshold value.

2 where=screen

Where DC-Mismatch analysis results should be printed. Possible values are screen, logfile, file,or rawfile.

3 file

File name for results to be printed if where=file is used.

Probe parameters

4 oprobe

Compute mismatch at the output defined by this component.

Port parameters

5 portv

Voltage across this probe port is output of the analysis. Instead of using pnode and nnode to identify output, users can use oprobe and portv.

6 porti

Current through this probe port is output of the analysis.

Notice that porti allows users to select a current associated with a specific device (component) given in oprobe as an output. This device, however, has to have its terminal current(s) as network variable(s), i.e. the device has to be an inductor, a switch, a tline, a controlled voltage source, an iprobe, or other type of device which has branch currents as network variables. Otherwise, Spectre will invalidate the request. Furthermore, for inductor, vsource, switch, controlled voltage source and iprobe, porti can only be set to one, since these devices are one port devices (two terminal); and for tline porti can be set to one or two, since it is a two port device (four terminals).

Sweep interval parameters

7 start=0

Start sweep limit.

8 stop

Stop sweep limit.

9 center

Center of sweep.

10 span=0

Sweep limit span.

11 step

Step size, linear sweep.

12 lin=50

Number of steps, linear sweep.

13 dec

Points per decade.

14 log=50

Number of steps, log sweep.

15 values=[...]

Array of sweep values.

Sweep variable parameters

16 dev

Device instance whose parameter value is to be swept.

17 mod

Model whose parameter value is to be swept.

18 param

Name of parameter to sweep.

State-file parameters

19 readns

File that contains estimate of DC solution (nodeset).

Output parameters

20 save

Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

21 nestlvl

Levels of subcircuits to output.

22 oppoint=no

Should operating point information be computed, and if so, where should it be sent. Possible values are no, screen, logfile, or rawfile.

Convergence parameters

23 prevoppoint=no

Use operating point computed on the previous analysis. Possible values are no or yes.

24 restart=yes

Do not use previous DC solution as initial guess. Possible values are no or yes.

Annotation parameters

25 annotate=sweep

Degree of annotation. Possible values are no, title, sweep, status, or steps.

26 stats=no

Analysis statistics. Possible values are no or yes.

27 title

Analysis title.

The dcmatch analysis will find a dc operating point first. If the dc analysis fails, then the dcmatch analysis will fail also. The parameter *mth* is a threshold value relative to the maximum contribution. Any device contribution less than (*mth* * maximum) will not be reported. Where maximum is the maximum contribution among all the devices of a given type.

Example

dcmm1 dcmatch mth=1e-3 oprobe=vd porti=1

model n1 bsim3v3 type=n ...

+ mvtwl=6.15e-9 mvtwl2=2.5e-12 mvt0=0.0 mbewl=16.5e-9 mbe0=0.0

In the above example, a dcmatch analysis is to be performed to investigate the 3-sigma dc variation at the output of the current flowing through the device vd. The porti=1 specifies that the current is flowing through the first port of vd (vd has only one port obviously, thus porti=2 will be illegal). All device mismatch contributions less than 1e-3 * maximum contribution among devices to the output will not be reported. dcmm1 is the name of the analysis, which can be used to identify output among analyses. The mismatch (i.e. the equivalent mismatch current sources in parallel to all the devices that uses model n1) is modeled by the model parameters *mvtwl*, *mvtwl2*, *mvt0*, *mbew*l, and *mbe0*. The meaning of these parameters is described in the modeling section.

The output of the dcmatch analysis is displayed like this:

```
**********************************************
DC Device Matching Analysis ‘mismatch1’ at vd
**********************************************

Local Variation = 3-sigma random device variation
```

sigmaOut	sigmaVth	sigmaBeta	sigmaVg	sigmaIds	
-13.8 uA	2.21 mV	357 m%	2.26 mV	1.71 %	mp6
-6.99 uA	1.63 mV	269 m%	1.68 mV	1.08 %	m01
-2.71 uA	1.11 mV	187 m%	1.16 mV	648 m%	m02
-999 nA	769 uV	131 m%	807 uV	428 m%	m04x4_4
-999 nA	769 uV	131 m%	807 uV	428 m%	m04x4_3
-999 nA	769 uV	131 m%	807 uV	428 m%	m04x4_2
-999 nA	769 uV	131 m%	807 uV	428 m%	m04x4_1
-999 nA	769 uV	131 m%	807 uV	428 m%	m04
-718 nA	1.09 mV	185 m%	1.15 mV	599 m%	m40x04
-520 nA	1.55 mV	263 m%	1.63 mV	835 m%	m20x04
-378 nA	2.21 mV	376 m%	2.34 mV	1.16 %	m10x04
-363 nA	539 uV	92 m%	567 uV	293 m%	m08
-131 nA	379 uV	64.9 m%	400 uV	203 m%	m16
-46.7 nA	267 uV	45.9 m%	283 uV	142 m%	m32

vd =-3.477 mA +/- 15.91 uA (3-sigma variation)

It says that the 3-sigma variation at i(vdd) is -3.477mA +/- 15.91uA due to the mismatch models. The -3.477mA is the dc operating value of the i(vdd), whereas the 15.91uA is the 3-sigma variation due to the device mismatches. The device mp6 contributes the most to the output at -13.8 uA followed by m01 which contributes -6.99uA. The equivalent 3-sigma Vth variation of mp6 is 2.21mV. The relative 3-sigma beta (current factor) variation of mp6 is 0.357%. The equivalent 3-sigma gate voltage variation is 2.26mV. The relative 3-sigma Ids variation of mp6 is 1.71%.

More Examples

```
dcmm2 n1 n2 dcmatch mth=1e-3 where=rawfile stats=yes
```

In the above example, a dcmatch analysis is to be performed to investigate the 3-sigma dc variation on output v(n1,n2). The result of the analysis will be printed in a psf file. The cpu statistics of the analysis will also be generated.

```
dcmm3 dcmatch mth=1e-3 oprobe=r3 portv=1
```

In the above example, the output is the voltage drop across the 1st port of r3. Since r3 has only one port, portv=2 will be illegal.

```
dcmm4 n3 0 dcmatch mth=1e-3 where=file file="%C:r.info.what"
```

In the above example, the output of the analysis will be printed in a file of .info.what prepended with circuit name.

```
sweep1 sweep dev=mp6 param=w start=80e-6 stop=90e-6 step=2e-6 {
dcmm5 n3 0 dcmatch mth=1e-3 where=rawfile
}
```

In the above example, a set of analyses will be performed on output v(n3,0) by sweeping the device parameter w of the device mp6 from 80um to 90um at each increment of 2um. The results will be printed as psf files. The psf files will be named as **sweep1-000_dcmm5.dcmatch** to **sweep1-005_dcmm5.dcmatch**.

```
dcmm6 n3 0 dcmatch mth=0.01 dev=x1.mp2 param=w
  + start=15e-6 stop=20e-6 step=1e-6 where=rawfile
```

In the above example, an internal sweep of dcmatch analyses will be performed by sweeping the device parameter w of the device x1.mp2 from 15um to 20um at each increment of 1um. The results will be printed as psf files. The psf files will be named as **dcmm6_0.dcmatch** to **dcmm6_5.dcmatch**. Notice that even it is an internal sweep, six files still will be generated. This is contradictory to a dc or an ac analysis, where only a single psf file is generated while sweeping a parameter. This is due to the fact while sweeping a parameter, the rank order of the output contribution for each devices may become different. Thus we need to store the results in multiple psf files.

```
dcmm7 n3 0 dcmatch mth=0.01 param=temp

+ start=25 stop=100 step=25
```

In the above example, temperature is swept from 25C to 100C at increment of 25C.

Mismatch modeling

1. Use Model

After identifying the output, the user needs to specify the parameters for each device contributing to the mismatch. Currently, The mismatch model implemented in Spectre for Mosfet transistors introduces five additional model parameters, one for bjts and nine for resistors.

To start the dc-mismatch analysis, the user types the analysis statement with the output voltage on which the dc-mismatch analysis is to be performed. The results are:

1) The 3-sigma random output variation

2) The dc operating point information

3) For each Mosfet:

- 3-sigma output variation, $3\sigma(\Delta Ids)\frac{\partial}{\partial \Delta Ids}Vout$
- 3-sigma Beta variation, $\frac{3\sigma(\Delta\beta)}{\beta_0}$
- 3-sigma Threshold voltage variation, $3\sigma(\Delta Vth)$
- 3-sigma gate voltage variation, $3\sigma(\Delta Vg)$
- 3-sigma current mismatch to nominal current ratio, $\frac{3\sigma(\Delta Ids)}{Ids_0}$

4) For each BJT

- 3-sigma output variation, $3\sigma(\Delta Ic)\frac{\partial}{\partial \Delta Ic}Vout$
- 3-sigma Vbe Variation, $3\sigma(\Delta Vbe)$

5) For each Resistor

- 3-sigma output variation, $3\sigma(\Delta Ir)\frac{\partial}{\partial \Delta Ir}Vout$
- 3-sigma Resistor current mismatch to nominal resistance current ratio, $\frac{3\sigma(\Delta Ir)}{Ir_0}$
- 3-sigma VR variation, 3σ(ΔVr) = 3σ(ΔIr).R
- 3-sigma IR variation, 3σ(ΔIr)
- 3-sigma Resistor variation, $3\frac{\sigma(\Delta Ir)}{Ir_0}R$

We define a threshold mth, below which the contributions will not be shown, currently, in spectre mth = 0.1%. Mth is an analysis parameter.

2. Summary of Theory.

Statistical variation of drain current in a MOSFET is modeled by:

$$Ids = Ids_o + \Delta Ids \quad , \tag{EQ 1}$$

where:

Ids -- is the total drain to source current.

Ids_o -- is the nominal current.

ΔIds -- is the variation in drain to source current due to local device variation.

Let $Vout$ be the output signal of interest, then the variance of $Vout$ due to the i^{th} MOSFET is approximated by:

$$Vout)_i = \left(\frac{\partial}{\partial \Delta Ids_i}Vout\right)^2\Bigg|_{Ids_o} \sigma^2(\Delta Ids_i) \tag{EQ 2}$$

The term $\frac{\partial}{\partial \Delta Ids_i} Vout$ in (2) is the sensitivity of the output to the drain to source current and can be efficiently obtained in a standard way as outlined in part b.

a- Mismatch models

The term $\sigma^2(\Delta Ids_i)$ is the variance of the mismatch current in Mosfet transistors. The mismatch in the current is assumed to be due to a mismatch in the threshold voltage (Vth) and the mismatch in the width to length ratio (β). It is approximated as:

$$\frac{\sigma^2(\Delta Ids)}{(Ids_0)^2} = \frac{gm_o^2}{(Ids_0)^2}\sigma^2(\Delta Vth) + \frac{\sigma^2(\Delta\beta)}{\beta_o^2} \quad , \qquad \text{(EQ 3)}$$

where

$$gm_o = \left.\frac{\partial Ids}{\partial Vth}\right|_{Ids_o} ,$$

and currently, as implemented in Spectre:

$$\sigma^2(\Delta Vth) = \frac{mvtwl^2}{WL} + \frac{mvtwl2^2}{WL^2} + mvt0^2$$

$$\frac{\sigma^2(\Delta\beta)}{(\beta 0)^2} = \frac{mbewl^2}{WL} + mbe0^2$$

Note, that gm_o is computed at the DC bias solution from the device model equations, the values a, b, c, d and e are the mismatch parameters while W and L are device parameters.

Another way of representing the mismatch current is using the gate voltage variation defined as:

$$\sigma^2(\Delta Vg) \equiv \frac{\sigma^2(\Delta Ids)}{gm_o^2} \qquad \text{(EQ 4)}$$

Similarly to the Mosfet, the bipolar $\sigma^2(\Delta Ici)$ and the resistor $\sigma^2(\Delta Iri)$ are computed for each device provided that the device size, bias point and mismatch parameters are known.

$$\frac{\sigma^2(\Delta Ir)}{Ir_0^2} = mr + \frac{mrl}{L^{mrlp}} + \frac{mrw}{W^{mrwp}} + \frac{mrlw1}{(LW)^{mrlw1p}} + \frac{mrlw2}{(LW)^{mrlw2p}}$$

$$\frac{\sigma^2(\Delta Ic)}{(Ic_0)^2} = \frac{(g\dot{m}_0)^2}{(Ic_0)^2} \cdot (mvt0)^2$$

where gm_0 is $\frac{\partial}{\partial VBE}Ic$ and Ic_0 is the nominal collector current, and ΔVbe =$(mvt0)^2$

b-Mismatch analysis

The goal is to compute $\frac{\partial}{\partial \Delta Ids_i}Vout$ for each MOSFET i. The network is characterized with the standard MNA equations with the addition of the term for the mismatch current:

$$(x) + \sum_i m_i \Delta Ids_i = \mathbf{0} \quad , \qquad \text{(EQ 5)}$$

where:

f -- is the vector of equations,

x -- is the vector of unknowns,

ΔIds_i -- is the mismatch current of the i^{th} MOSFET,

$\boldsymbol{m}_i$ -- is incidence column vector that maps the mismatch current of the ith device to the proper MNA equations.

The output is defined as:

$$Vout = \boldsymbol{y}^T \boldsymbol{x}, \qquad \text{(EQ 6)}$$

where $\boldsymbol{y}$ is a column vector and superscript “T” represents the transpose.

In terms of (6), the sensitivity becomes:

$$\frac{\partial}{\partial \Delta Ids_i} Vout = \boldsymbol{y}^T \frac{\partial \boldsymbol{x}}{\partial \Delta Ids_i}. \qquad \text{(EQ 7)}$$

The term $\frac{\partial \boldsymbol{x}}{\partial \Delta Ids_i}$ is obtained by differentiating (5) with respect to ΔIds_i --

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \Delta Ids_i} + \boldsymbol{m}_i = \boldsymbol{0}, \qquad \text{(EQ 8)}$$

and then solving

$$\frac{\partial \boldsymbol{x}}{\partial \Delta Ids_i} = -\left[\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right]^{-1} \boldsymbol{m}_i. \qquad \text{(EQ 9)}$$

Combining (9) with (7), the expression for sensitivity becomes:

$$\frac{\partial}{\partial \Delta Ids_i} Vout = -\boldsymbol{y}^T \left[\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right]^{-1} \boldsymbol{m}_i \qquad \text{(EQ 10)}$$

Since there is only one output and many MOSFETS, it is efficient to define a vector z such that

$$z^T = y^T \left[\frac{\partial f(x)}{\partial x} \right]^{-1} \quad \text{(EQ 11)}$$

or

$$\left[\frac{\partial f(x)}{\partial x} \right]^T z = y \quad \text{(EQ 12)}$$

After computing the vector z through a “transpose solve” in (12), the sensitivity of the output with respect to each transistor current can be computed with one extra vector product:

$$\frac{\partial}{\Delta Ids_i} Vout = -z^T m_i \quad \text{(EQ 13)}$$

and finally, $\sigma^2(Vout) = \sum_{i=1}^{n} (z^T m_i)^2 \sigma^2(\Delta Ids_i)$ would be the variance of the output, where n is the number of devices.

In the equations above, the bjt and the resistor contributions to the mismatch should be added.

3. Finding Model Parameters.

To determine dcmatch model parameters, users have to go to their foundry or modeling group, study the statistical process, then do a regression analysis. We will give an example of how to use the concept of curve fitting to find model parameters. The actual procedure may vary significantly and is a matter of art. We will only introduce the concept. In general, to find parameters is not a trivial task.

Lets assume we want to find out the parameters *mvtwl*, *mvtwl2*, and *mvt0*, which are used in the model equation

$$\sigma^2(\Delta Vth) = \frac{mvtwl^2}{WL} + \frac{mvtwl2^2}{WL^2} + mvt0^2 \qquad \text{(EQ 14)}$$

The modeling group needs to make, say 200 statistical measurements of the values of $\sigma^2(\Delta Vth)$ by varying values of channel length *L* and width *W*. Provided that threshold voltage extraction is a known procedure. Typically, threshold voltage is the value of gate voltage that corresponds to a drain current of 1mA. Extraction of this parameter is done at low drain bias (about 0.1V) although in modern technologies this value may be even lower. Gate bias is increased gradually until this level of current is achieved. After the 200 samples are collected, the task becomes to find parameters *p1*, *p2*, and *p3* to minimize the curve fitting error via the least square fit formula:

$$error = \sum_{i=1}^{200}\left(y_i - \left(\frac{p_1}{W_iL_i} + \frac{p_2}{W_iL_i^2} + p_3\right)\right)^2 \qquad \text{(EQ 15)}$$

where y_i is the i-th measurement value (among the 200 measurements) of $\sigma^2(\Delta Vth)$ at $L = L_i$ and $W = W_i$. Once the *p1*, *p2*, and *p3* are found, we can take the square root of them to find *mvtwl*, *mvtwl2* and *mvt0*, repectively. The minimum of *error* in Equation (15) occurs where the derivative of *error* with respect to *p1*, *p2*, and *p3* vanishes.

$$\frac{\partial}{\partial p_1}error = 2\sum_{i=1}^{200}\left(y_i - \left(\frac{p_1}{W_iL_i} + \frac{p_2}{W_iL_i^2} + p_3\right)\right)\frac{-1}{W_iL_i} = 0 \qquad \text{(EQ 16)}$$

$$\frac{\partial}{\partial p_2}error = 2\sum_{i=1}^{200}\left(y_i - \left(\frac{p_1}{W_iL_i} + \frac{p_2}{W_iL_i^2} + p_3\right)\right)\frac{-1}{W_iL_i^2} = 0 \qquad \text{(EQ 17)}$$

$$\frac{\partial}{\partial p_3} error = 2\sum_{i=1}^{200}\left(y_i - \left(\frac{p_1}{W_i L_i} + \frac{p_2}{W_i L_i^2} + p_3\right)\right)(-1) = 0 \qquad \text{(EQ 18)}$$

The Equations (16), (17), and (18) can be rearranged in normal form as:

$$\begin{bmatrix} \sum \frac{1}{W_i^2 L_i^2} & \sum \frac{1}{W_i^2 L_i^3} & \sum \frac{1}{W_i L_i} \\ \sum \frac{1}{W_i^2 L_i^3} & \sum \frac{1}{W_i^2 L_i^4} & \sum \frac{1}{W_i L_i^2} \\ \sum \frac{1}{W_i L_i} & \sum \frac{1}{W_i L_i^2} & \sum 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \sum \frac{y_i}{W_i L_i} \\ \sum \frac{y_i}{W_i L_i^2} \\ \sum y_i \end{bmatrix} \qquad \text{(EQ 19)}$$

Assuming that the matrix in the Equation (19) is not singular, then (19) can be solved for *p1*, *p2*, and *p3*, using standard LU decomposition and backsubstitution. If the matrix is singular, then different method such as singular value decomposition can be used. The model parameters *mvtwl*, *mvtwl2*, and *mvt0* are simply the square root of the *p1*, *p2*, and *p3*, respectively.

As for illustration, we attach a sample C program here. The first half portion of the program is to cook up 200 sample data points of y_i to mimic measurement data. We also create the right hand side vector rhs as in Equation (19). The last half portion creates the matrix, and calls dense matrix solver routines *dgeco* and *dgesl* from LINPACK (a popular dense matrix package) to obtain the fitted parameters. Upon executing the program, we got this result:

model parameter mvtwl = 0.01
model parameter mvtwl2 = 0.003
model parameter mvt0 = 0

```
generating 200 samples using the above parameters ...

fitted parameter 0 = 0.0102933
fitted parameter 1 = 0.00290799
fitted parameter 2 = 0
```

The program is by no means to be robust or efficient. Nevertheless, it illustrates the concept of curve fitting. The exact procedure for fitting your measurements to obtain proper model parameters robustly has to be developed by your modeling group.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define F77CONV(array,i,j,lda) (array[ (i) + (lda) * (j) ])

main() {

   int i, j, k, samples, n, lda, one, zero;
   double y, sigmavth2; /* sigma^2(dvth) */
   double mvtwl, mvtwl2, mvt0;
   double w, l;
   double r, t, rcond, tmp;
   double alpha[3][3], rhs[3], work[3];
   double farray[9];
   int ipvt[3];
   double drand_();

   /* the following are the model parameters to fit */
```

```
mvtwl = 10e-3;
mvtwl2 = 3e-3;
mvt0 = 0;
printf(" model parameter mvtwl = %g\n", mvtwl);
printf(" model parameter mvtwl2 = %g\n", mvtwl2);
printf(" model parameter mvt0 = %g\n", mvt0);
n = lda = 3;
samples = 200;
zero = 0;
printf(" generating %d samples using the above parameters ...\n",samples);

for (i = 0; i<n; i++) {
    for (j = 0; j<n; j++) {
        alpha[i][j] = 0.0;
}
    rhs[i] = 0.0;
}
r = drand_(&one);
l = 0.18;
/*********************************************/
/* portion1: generate fake measurement data */
/*********************************************/
k = 0;
for (i = 1; i<=samples; i++) {
    w = 1.0 * i; /* different channel widths */
    if (i >= samples/2) { /* different channel lengths */
```

```
            k++;
            l = 0.36;
            w = 1.0 * k;
        }
        sigmavth2 = mvtwl*mvtwl/(w*l) + mvtwl2*mvtwl2/(w*l*l) + mvt0*mvt0;
        r = drand_(&zero)*5e-2;
        /* generate fake measurement data */
        y = (1.0 +r) * sigmavth2;
        /* create least square fit right rhs */
        rhs[0] += y/(w*l);
        rhs[1] += y/(w*l*l);
        rhs[2] += y;
    }
    /*********************************************/
    /* portion2: solve for p                     */
/*********************************************/
    /* create the curve fittling matrix  */
    l = 0.18;
    k = 0;
    for (i = 0; i<samples; i++) {
        w = 1.0 * (i+1);
        if ((i+1) >= samples/2) {
            k++;
            l = 0.36;
            w = 1.0 * k;
        }
```

```
        tmp = 1.0/(w*l);
        alpha[0][0] += (tmp*tmp);
        alpha[0][1] += (tmp*tmp)/l;
        alpha[0][2] += tmp;
        alpha[1][0] += (tmp*tmp)/l;
        alpha[1][1] += (tmp*tmp)/(l*l);
        alpha[1][2] += (tmp/l);
        alpha[2][0] += tmp;
        alpha[2][1] += tmp/l;
        alpha[2][2] += 1.0;
    }

    /* convert to fortran array, since we will be calling fortran solvers */
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            F77CONV( farray, i, j, lda ) = alpha[i][j];
        }
    }

    /* solve the matrix to obtain the three fitted model parameters */
    dgeco_( farray, &n, &lda, ipvt, &rcond, work);
t = 1.0 + rcond;
    if (t == 1.0) {
        printf("singluar matrix of alpha encountered rcond = %g\n",rcond);
    }
    dgesl_( farray, &n, &lda, ipvt, rhs, &zero);
```

```
    /* after dgesl, rhs contains p1, p2, p3 now */
    printf("\n");
    for (i = 0; i< 3; i++) {
        if (rhs[i] <= 0) rhs[i] = 0.0;
        if (rhs[i] >= 0) {
            printf(" fitted parameter %d = %g\n", i, sqrt(rhs[i]));
        }
    }
}
```