From 11/30/25

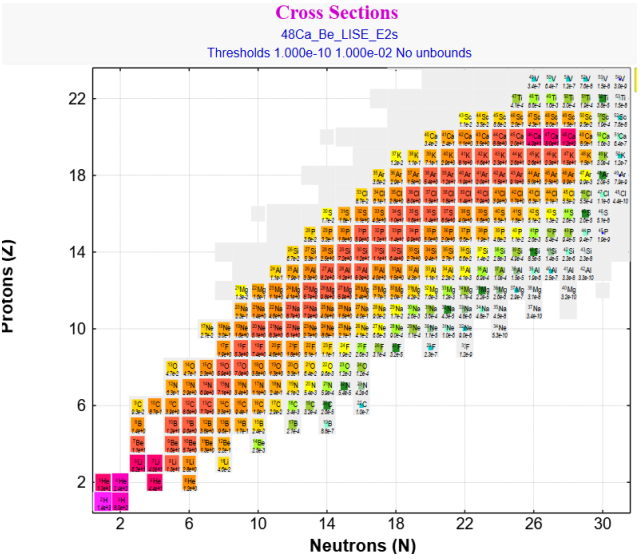
- USER cross-section files from current model
- Convolution model corrections (high energy)
- AA-minimization:
  -- Cycling local line to overcome local minima
  -- Diagonal scaling (DSCL) in Levenberg–Marquardt minimization
- Abrasion recommended-values plotting

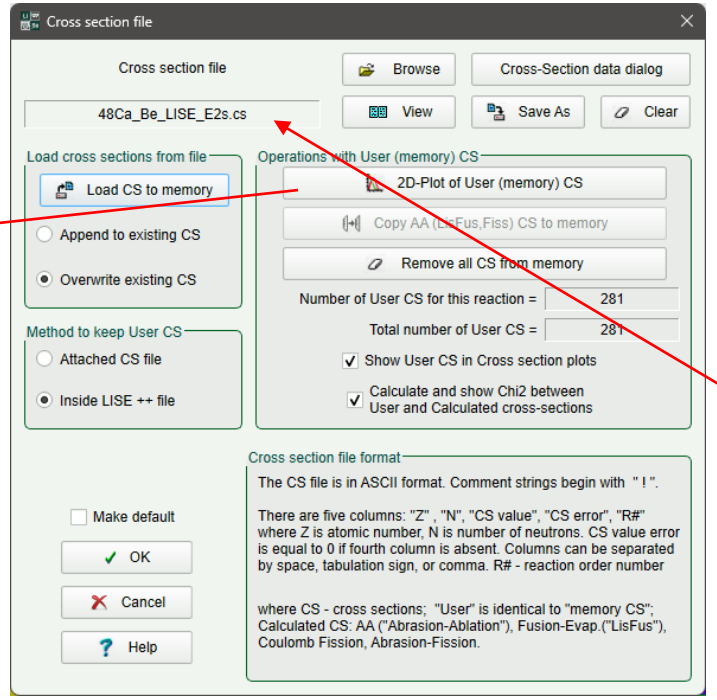
USER cross-section files from current model

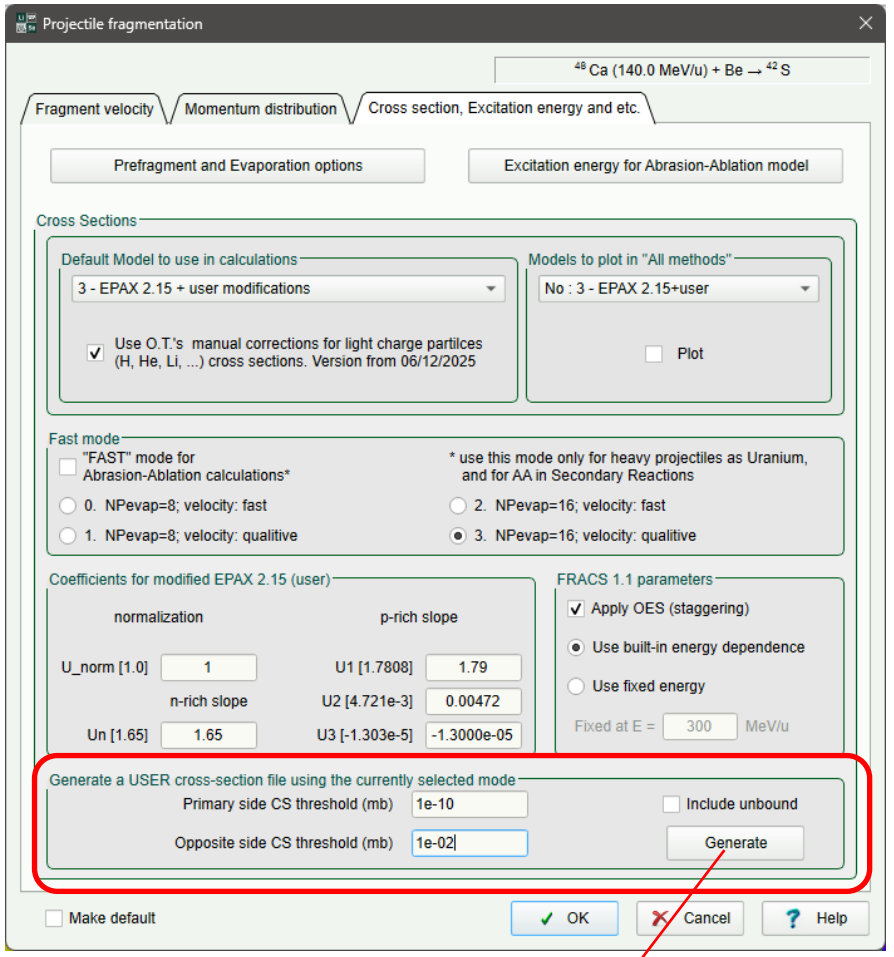
Relevant versions: 17.17.19, 17.17.20

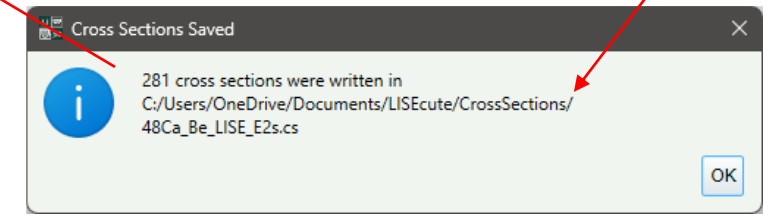
- Added option to generate a **USER cross-section file from the current model** in `d_MechanismFragmentation`
  - The file is produced directly from the active abrasion–ablation calculation (current project settings, model parameters, and filters)
- Added a toggle **"Include unbound"** in the USER CS file generator
  - When enabled, particle-unbound isotopes are also written to the user cross-section table
  - When disabled, only particle-bound isotopes are included, giving a cleaner table for many experimental analyses

This User CS file can be used for AA benchmarking









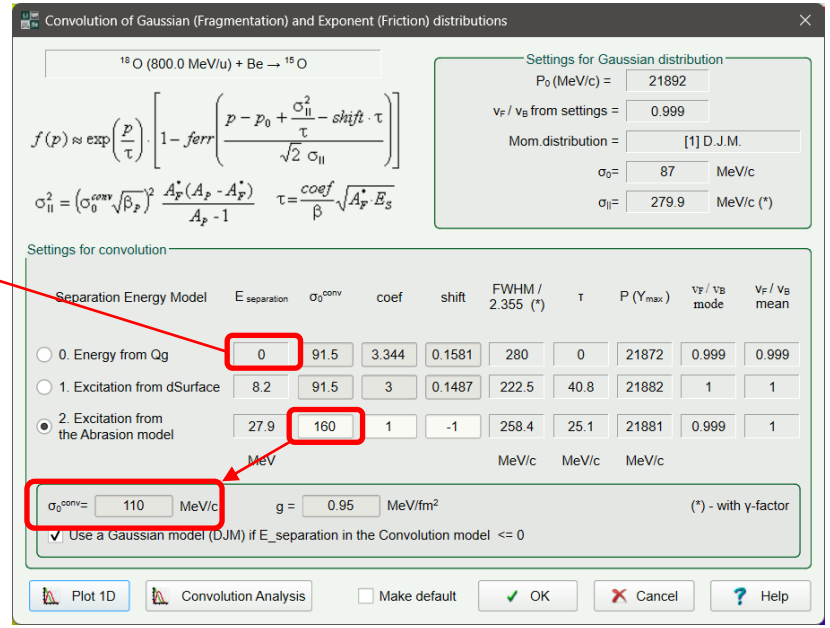
Convolution model corrections (high energy)

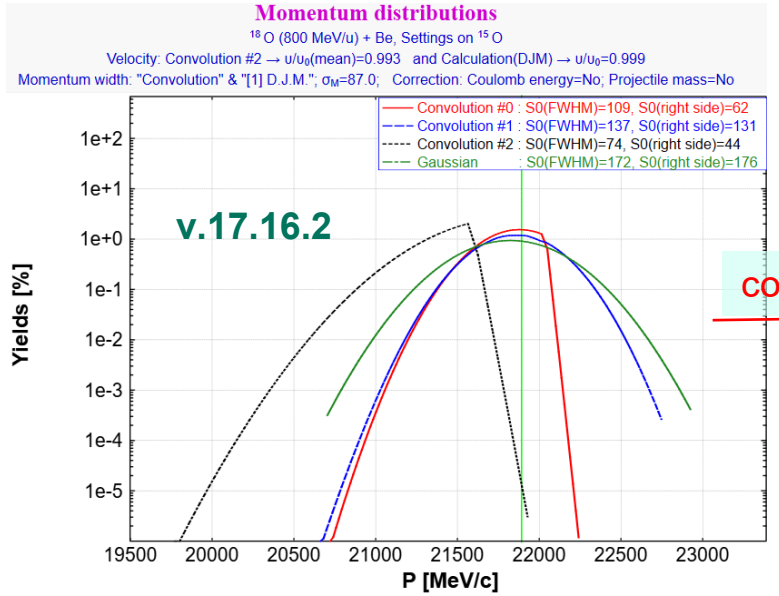
Relevant versions: 17.17.22, 17.17.23

- Corrected the **convolution model** behavior for **low separation energy**
  - Fixes truncation / shape issues in the tail region of the excitation-energy distribution at low *S*

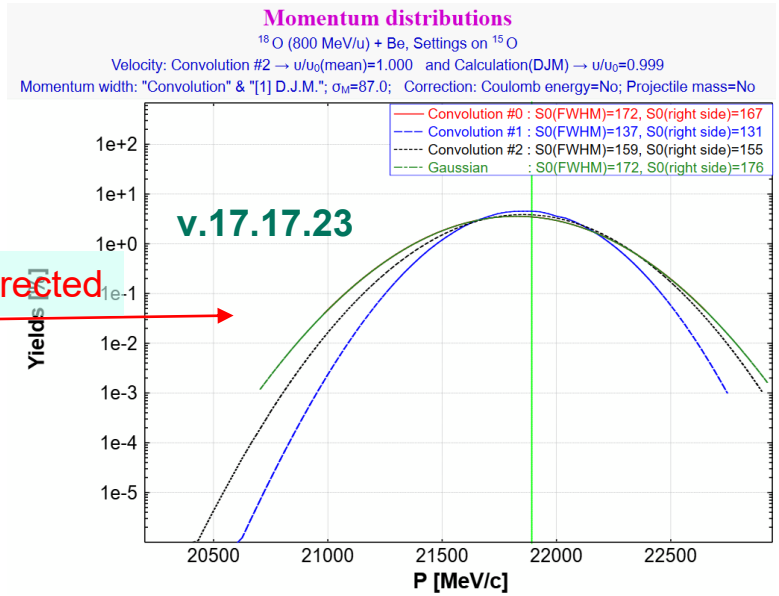
- Fixed the energy-dependent function for Convolution Method #2 at high projectile energies
  - Removed a bug that produced incorrect widths at high *E*
  - The $\sigma_0(E)$ dependence is now smooth across the full energy range

Using DJM model instead at *S*=0





corrected



Based on DJM feedback for extreme cases

AA-minimization: cycling local line to overcome local minima (1)

Relevant versions: 17.17.21, 17.17.24, 17.17.25, 17.17.26, 17.17.27, 17.17.28, 17.17.29

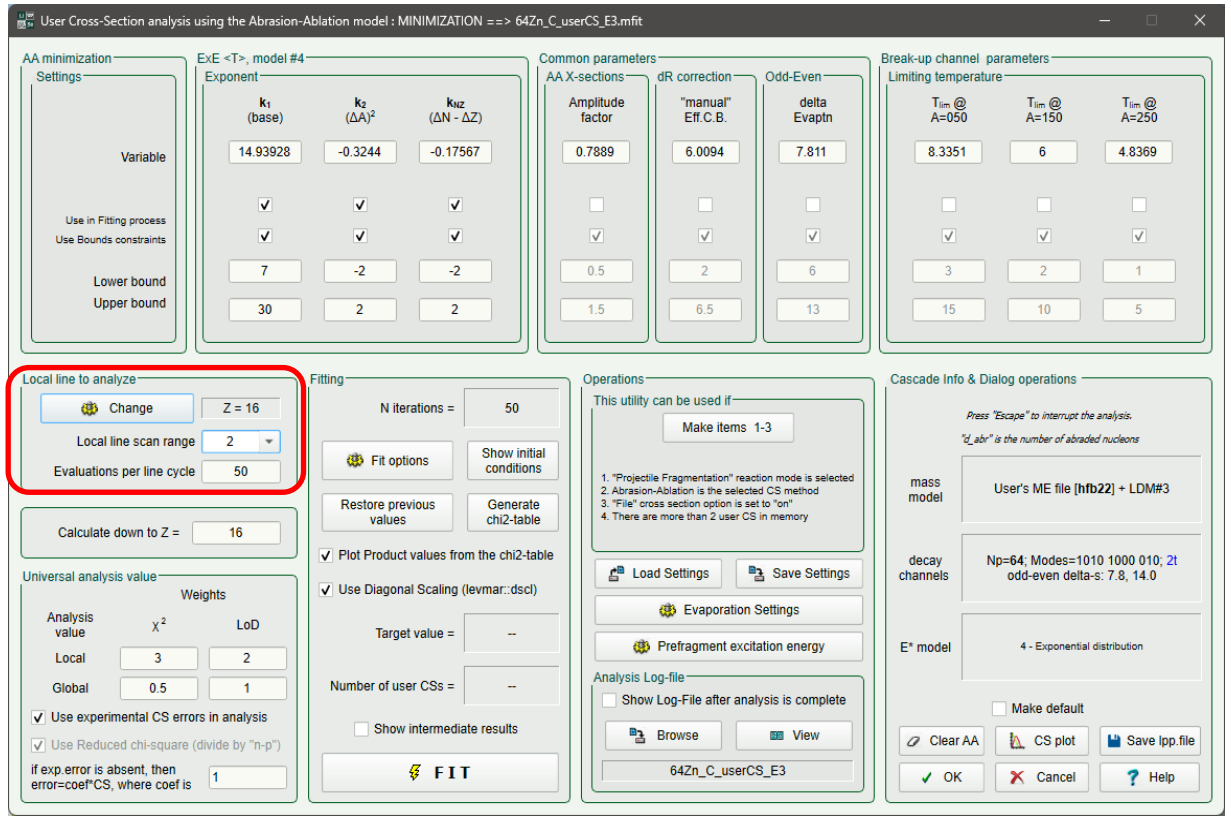
AA-minimization: cycling local line to overcome local minima

In AA-minimization the **target function** combines four contributions:

- χ^2 for all isotopes (global χ^2)
- $\log_{10}$–difference "LoD" for all isotopes (global LoD)
- χ^2 for the **local line** only (local χ^2)
- LoD for the local line only (local LoD)

The user chooses the local line (for example $Z = 24$, or $N = 36$) and the four weights.
Because the local line can have large weight, the minimizer may "lock" into a local minimum that describes only this line very well while the surrounding Z or N region remains poor.

To help escape such local minima, version 17.17.21–25 adds an option to **cycle the local line** during a single minimization run.



AA-minimization: cycling local line to overcome local minima (2)

Concept

Instead of keeping one fixed local line, LISE^{++} can periodically move it to neighboring lines and then return to the original one. The parameters are always updated **continuously**; only the definition of the "local" subset is changed.

Two new options are introduced:

1. **Local line scan range** (dN_cycle)
   - Integer Δ = 0…5 selected by combo box
   - Δ = 0 → no cycling, behavior identical to old versions
   - Δ = 1 → scan the initial line and its immediate neighbors (Z_0-1, Z_0, Z_0+1) or (N_0-1, N_0, N_0+1)
   - Δ = 2 → also include $Z_0\pm2$ (or $N_0\pm2$), etc.
2. **Evaluations per line cycle** (Eval_cycle)
   - Number of objective-function evaluations performed before the local line is moved to the next value in the scan sequence
   - Only visible and used when Δ ≠ 0

Together, these options define a **scan pattern** in Z or N during the Levenberg–Marquardt (LM) iterations.

Scan pattern

Let Z_0 be the initial local line chosen by the user.
For a given scan range Δ the internal helper function

int TDataFitMinimizationDlg::currentLine(int Z_init, int eval_index)

returns the Z (or N) value that is treated as "local" at a specific evaluation number.

For example, with Δ = 2 the pattern in one full cycle is:

- step group d = 0: Z_0
- d = 1: $Z_0-1 \rightarrow Z_0 \rightarrow Z_0+1$
- d = 2: $Z_0-2 \rightarrow Z_0 \rightarrow Z_0+2$

So the scan order is:

Z_0, Z_0-1, Z_0, Z_0+1, Z_0-2, Z_0, Z_0+2

Each "slot" in this sequence is held for **Eval_cycle** evaluations; then the local line is advanced to the next value. When the sequence is finished it wraps around and starts again from Z_0.

If a shifted line goes outside the allowed range (for example below the minimum Z with data), it is skipped.

The same logic is used whether the user selected a **Z-line** or an **N-line**; internally only the index of the line changes.

AA-minimization: cycling local line to overcome local minima (3)

Interaction with LM and parameters

- The **fit parameters themselves are never reset** when the local line changes.
  LM continues from the current best parameter vector p.
- Every time dlevmar_bc_dif_function_CSmin() is called, LISE^{++}:
  1. Determines the current evaluation index
  2. Computes the active local line with currentLine()
  3. Rebuilds the local contributions χ^2_local and LoD_local for that line
  4. Forms the total target value using the user weights
- This way one continuous LM run "feels" several slightly different landscapes, which helps the algorithm climb out of shallow or narrow minima centered on a single Z or N.

The cycling is **purely internal**:

- Output .mfit file still reports one final "Local line Z = …; Last Z = …" header using the originally selected line.
- Log messages such as Zcurrent changed! (used for debugging) can be disabled in production builds.

Practical use

1. **Simple fits**
   - Leave Δ = 0 (no cycling).
   - This reproduces the behavior of previous LISE^{++} versions.
2. **Difficult cases (strong local minima)**
   - Choose Δ = 1 or 2 to include immediate neighbor lines.
   - Set **Evaluations per line cycle** to a small number (for example 10–30) so that LM samples several lines during the run.
3. **Large scan ranges**
   - Δ = 3–5 are possible but may slow down each iteration because the "local" part is recomputed more often.
   - Use only when the minimum clearly jumps between more distant Z or N values.

Cycling the local line does not guarantee finding the global minimum, but in practice it:

- Smooths the target landscape seen by LM
- Reduces the chance that the fit is fully optimized for one Z (or N) row while neighboring rows remain poorly described

Diagonal scaling (DSCL) in Levenberg–Marquardt minimization (1)

(introduced in v17.17.28)

Why DSCL was added

In AA minimization the fitted parameters live on very different numerical scales:

- Hole energy E_{hole}: tens of MeV
- T_{mean} and limiting temperatures: about 1–20 MeV
- AA cross-section factor (G_AA_factor): around 1
- Odd–even Δ and dR: roughly 0.1–10

If you feed these directly into Levenberg–Marquardt (LM), you can get:

- Bad conditioning of the normal matrix J^TJ
- Very small steps for "large" parameters and too aggressive steps for "small" ones
- Stalls in shallow valleys or "singular matrix" warnings

To reduce these effects LISE++ now supports a **diagonal scaling** (DSCL) option that uses LM's `dscl` array to normalize parameters internally.

When DSCL is enabled, every active fit parameter is scaled according to its allowed range before LM sees it. This improves conditioning and makes step sizes more comparable.

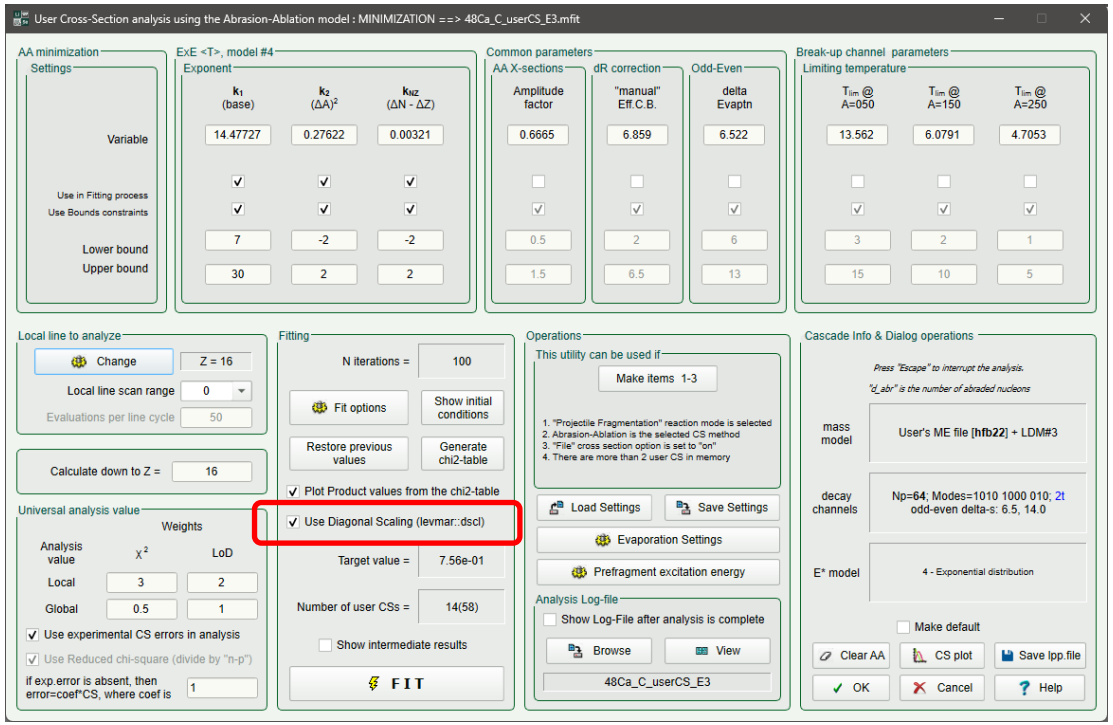
Basic idea

The LM library accepts an optional diagonal scaling vector `dscl[m]`:

For every parameter p_i LM works with scaled variables

> $q_i = d_i \cdot p_i$

where d_i = dscl[i].



In LISE++:

- For each parameter with **active bounds** [p_i_min, p_i_max] we compute the span
  $\Delta p_i = |p_i\_max - p_i\_min|$
- Then we set
  $d_i = 1 / \Delta p_i$
- If the span is zero or invalid, we fall back to
  $d_i = 1$

So in LM's internal space all bounded parameters have a typical scale of order 1, no matter what their physical units are.

This:

- Reduces anisotropy in parameter space
- Makes J^TJ less likely to be close to singular
- Allows LM to choose more reasonable global step sizes and often converge faster

Diagonal scaling (DSCL) in Levenberg–Marquardt minimization (2)

How it is implemented

1. DSCL storage and setup

In `CreateArrays()` a new array is allocated:

- `lm_dscl = new double[m];` for all m fit parameters
- It is initially filled with ones:
  `lm_dscl[k] = 1.0` for all k

For each parameter where bounds are active, LISE++ calls a helper like

```cpp
void TDataFitMinimizationDlg::setDSCL(int n)
{
  double span = qFabs(lm_lb[n] - lm_ub[n]);
  if (span <= 0.0) span = 1.0;   // safety
  lm_dscl[n] = 1.0 / span;
}
```

This is used for (when bounds are enabled):

- E-coefficients and σ-coefficients
- T_{mean} parameters
- Log-normal median and variance parameters
- Hole energy
- AA factor, thermalization coefficient, odd–even Δ, dR
- T_{lim} points
- dR_auto coefficients

Parameters without meaningful bounds simply keep dscl = 1.

2. Passing DSCL into LM

The main LM call in `CmFit()` now looks like:

```cpp
FitResult = dlevmar_bc_dif(
    dlevmar_bc_dif_function_CSmin,
    lm_p, lm_x, m, n,
    lm_lb, lm_ub,
    Levmar_fit_options.UseDSCL ? lm_dscl : nullptr,
    Levmar_fit_options.N,
    Levmar_fit_options.opt,
    info,
    lm_work, lm_covar,
    nullptr);
```

- If `Levmar_fit_options.UseDSCL` is **false**:
  `dscl = nullptr` and LM behaves exactly like older LISE++ versions
- If `UseDSCL` is **true**:
  `dscl = lm_dscl` and LM applies the diagonal scaling internally

The log header prints whether DSCL was used, for traceability, for example:

> dscl=yes

or

> dscl=no

on the first line of the `.mfit` file.

Diagonal scaling (DSCL) in Levenberg–Marquardt minimization (3)

User control and recommended usage

- GUI has a checkbox/option: **Use DSCL** (exact label as you implement it)
- When **enabled**: all bounded parameters are scaled using their current limits
- When **disabled**: behavior is identical to historical LISE++ fits

Recommendations:

- Keep DSCL **on** when
  - The parameter set mixes energies, factors, and small corrections
  - You see "Singular matrix A..." messages during fits
  - The target value decreases very slowly or behaves irregularly
- Turn DSCL **off** only if
  - You need to reproduce an old result bit-for-bit
  - You are debugging LM internals

Practical impact

From first tests in AA minimization:

- "Singular matrix A in dAx_eq_b_LU_noLapack()" appears less often
- Fits remain more stable while you **cycle the local line** (Z scan)
- Target value tends to decrease more smoothly in difficult fits

DSCL is not magic: it cannot fix a completely under-constrained problem or perfectly correlate parameters. But it gives the LM engine a numerically healthier problem and noticeably improves robustness of AA minimization in realistic LISE++ use cases.

Abrasion recommended-values plotting

Relevant versions: 17.17.18

•Fixed tiny issues in plotting the *Recommended abrasion-reaction settings* values

•Ensures that the curves and markers for the recommended settings are consistent with the underlying numerical values used in calculations