PAUL SCHERRER INSTITUT

Memorandum

Datum: March 26, 2008

Von: Andreas Suter
Telefon: +41 (0)56 310 4238
Raum: WLGA / 119
e-mail: andreas.suter@psi.ch

An:

cc:

Realistic Meissner Profile Test

In order to cross-check that a simple Gaussian fit to measured Meissner screening profiles $B(z)$ is a reasonable approach, I performed some tests which will be summarized in this memo. For the test it is assumed that Meissner screening profile $B(z)$ is given by the following formula (symmetric thin film model)

$$B(z) = B_{\text{ext}} \frac{\cosh\left[\frac{z - d/2}{\lambda}\right]}{\cosh\left[\frac{d}{2\lambda}\right]} \tag{1}$$

with B_{ext} the applied magnetic field, d the thickness of the film, and λ the London penetration depth. This formula is just the superposition of the exponential penetrating from both sides of the film, i.e. $B(z) \propto e^{-z/\lambda} + e^{-(d-z)/\lambda}$.

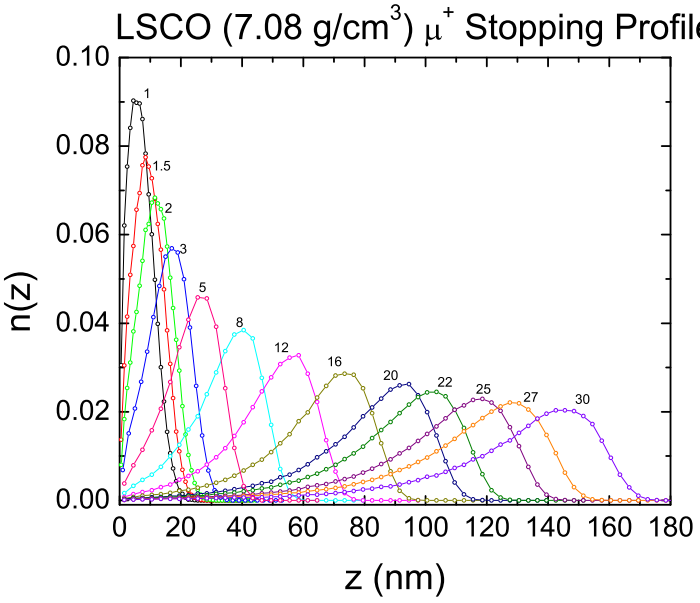

Figure 1: μ^+ stopping profile for LSCO ($\rho = 7.08\,(\text{g/cm}^3)$) for various energies calculated by TRIM.SP. The labels denote the implantation energies.

Since the muon stopping distribution $n(z)$ is rather broad, it needs explicitly takes care

of. I have calculated using TRIM.SP some stopping profiles for LSCO ($La_{2-x}Sr_xCuO_4$ with $\rho = 7.08\,(g/cm^3)$) which are shown in Fig.1.

As a further ingredients the relation between the muon stopping distribution and the field probability distribution $p(B)$ is needed. This is given by

$$p(B)dB = n(z)dz \tag{2}$$

where $p(B)$ is the probability distribution to find a field in the range $[B, B+dB]$. Eq.(2) is only piecewise applicable in the ranges where $B(z)$ is monotonic. For the ansatz from Eq.(1) this means one has to split it into 2 regions, namely $z \leq d/2$ and $z > d/2$. The routine calculating $p(B)$ for a given $n(z)$ and the parameters describing $B(z)$ (B_{ext}, d, λ) is given in the Appendix A. Some typical obtained $p(B)$'s are shown in Fig.2. Interesting to note is that for $E \gtrsim 12\,(\text{keV})$ there is a singular feature visible on the low field side of $p(B)$. The origin of this feature is given due to the divergence of $\left|\frac{dB}{dz}\right|^{-1}_{z\to d/2}$ and is found at the field $B_{\text{min}} = B_{\text{ext}}/\cosh[d/(2\lambda)]$. In addition to the field distribution given by the Meissner screening profile, the ROOT macro (see Appendix A) can add a background $p_{\text{bkg}}(B)$ simulating muons precessing closely around the external field B_{ext}. $p_{\text{bkg}}(B)$ is implemented as a Gaussian and is simulating muons not hitting the sample and hence precessing in the vicinity of B_{ext}.

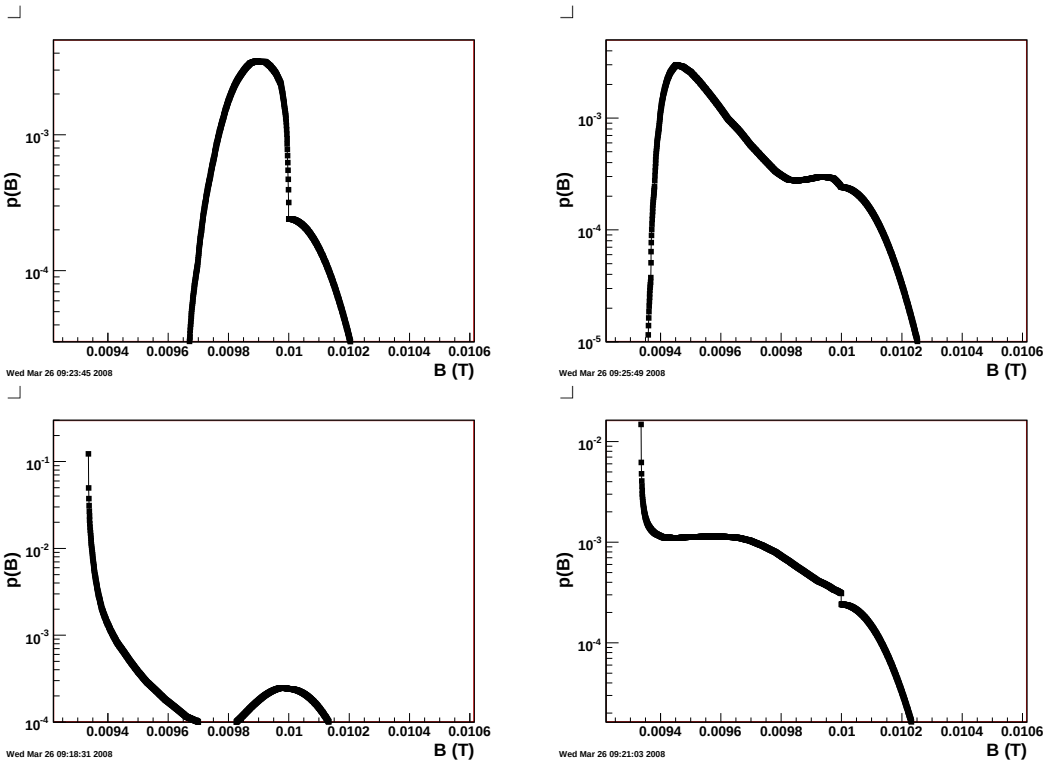

Figure 2: $p(B)$'s for various implantation energies. Top left: $E = 1.0$ (keV), top right: $E = 8.0$ (keV), bottom left: $E = 16.0$ (keV), bottom right: $E = 27.0$ (keV).

Having calculated the $p(B)$, I used the `fakeDataNemu` program which is generating fake decay histograms from an input $p(B)$ and some additional parameters telling what statistics is wished, how many histograms shall be generated, what phase, etc.[1]

The fake data histograms where fitted by the following simple model

$$A_i(t) = A_0 \exp\left[-1/2(\sigma t)^2\right] \cos(\gamma_\mu B_{\text{sg}} t + \phi_i) + A_{\text{bkg}} \exp\left[-1/2(\sigma_{\text{bkg}} t)^2\right] \cos(\gamma_\mu B_{\text{ext}} t + \phi_i) \tag{3}$$

[1]There are two versions to generate fake data from a given $p(B)$, namely `fakeDataNemu` which is generating nemu files, and `fakeData` which is generating native LEM ROOT output files.

using single histogram fits of WKM. The index i is referring to the specific histogram. For the fit $A_{\rm bkg}$, $\sigma_{\rm bkg}$, and $B_{\rm ext}$ were fixed. This is a reasonable approach since the same can be done in real experiments. Fig.3 shows $B_{\rm sg}$ versus $\langle z\rangle$, where $\langle z\rangle$ is the mean range of $n(z)$, i.e. $\langle z\rangle = \int z\, n(z)\, dz / \int n(z)\, dz$. The curve shown in Fig.3 is a fit to Eq.(1) with all parameters free. As can be seen the deviation of the fitted parameters (inset of Fig.3) in the 1-2% range, which means the simple Gaussian fit approach is very good (at least for the parameters used in this test).

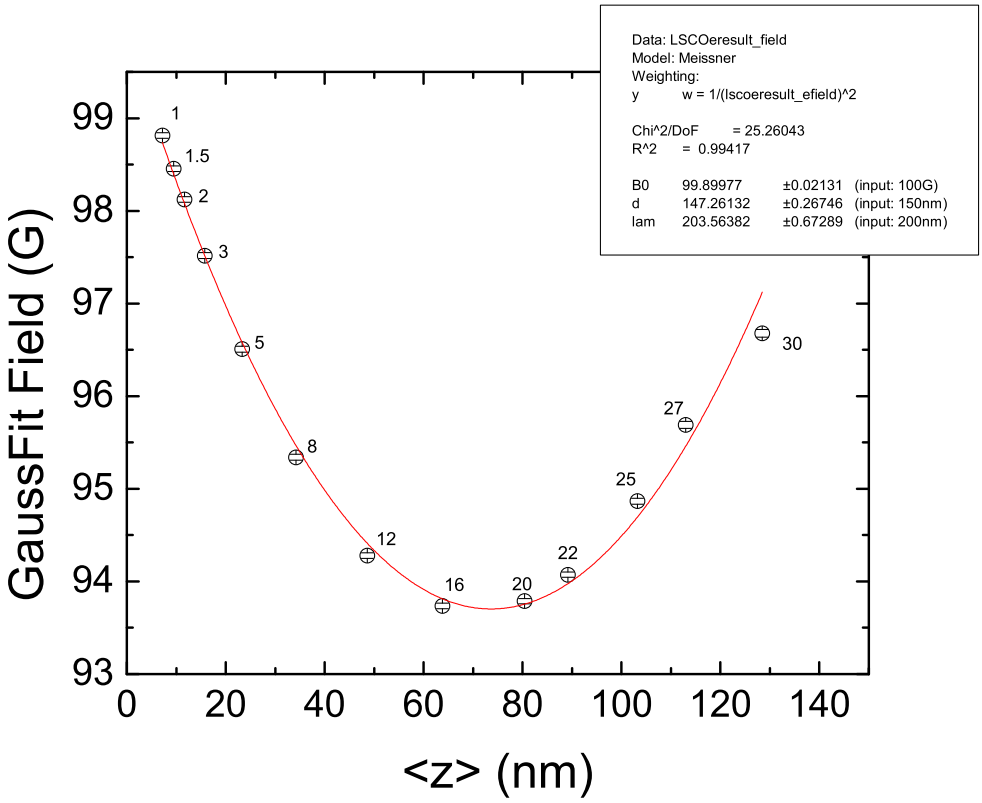

Figure 3: Field profile as found by a Gaussian fit. To be more precise: the figure shows $B_{\rm sg}$ versus $\langle z\rangle$, where $B_{\rm sg}$ is the field obtained from a simple Gaussian fit to the asymmetry (for details see the text), and $\langle z\rangle$ is the mean range obtained from TRIM.SP. The numbers placed next to the symbols are denoting the implantation energy.

Fig.4 shows the depolarization rate σ as function of $\langle z\rangle$. $\sigma(\langle z\rangle)$ has a very peculiar shape which is originating from the interplay between the stronger broadening of $n(z)$ for larger energies and the slope of $B(z)$. Naturally $\sigma(\langle z\rangle)$ has a minimum close to the center of the film.

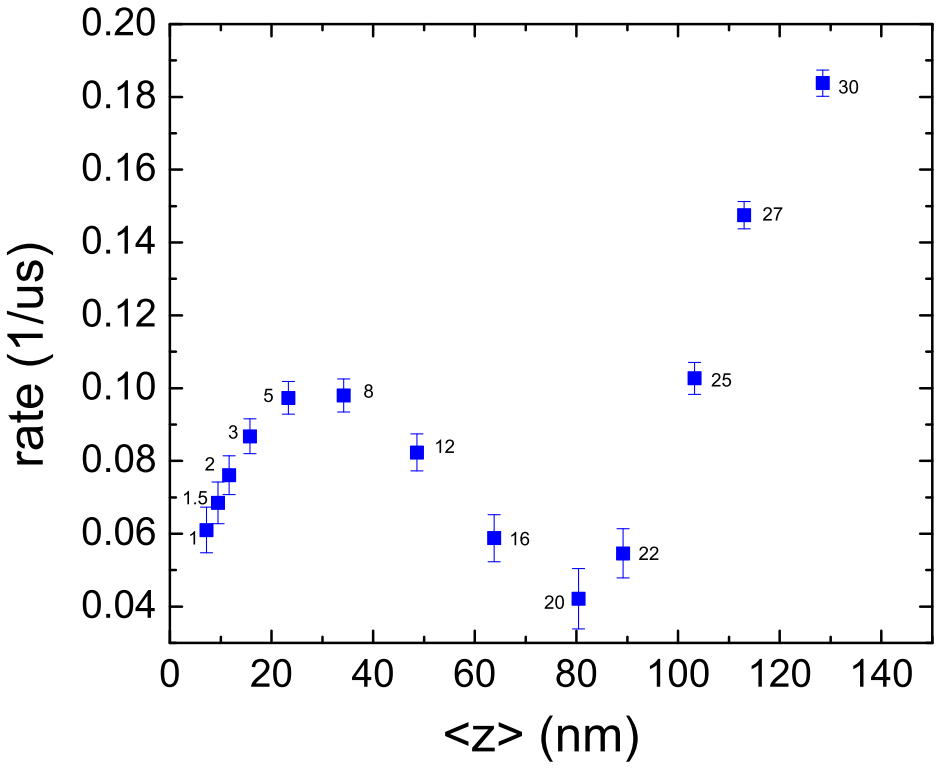

Figure 4: Depolarization rate (σ) profile. To be more precise: the figure shows σ versus $\langle z \rangle$, where σ is the variance of the simple Gaussian fit (for details see the text), and $\langle z \rangle$ is the mean range obtained from TRIM.SP. The numbers placed next to the symbols are denoting the implantation energy.

A Used Root Macro to generate $p(B)$

```
#include <vector>
using namespace std;

Double_t GetNofZ(Double_t zz, vector<Double_t> &z, vector<Double_t> &nz)
{
  Bool_t found = false;
  UInt_t i;
  for (i=0; i<z.size(); i++) {
    if (z[i]/10.0 >= zz) {
     found = true;
     break;
    }
  }

  if (!found)
    return -1.0;

  if (i == 0)
    return nz[0]*10.0*zz/z[0];

  return TMath::Abs(nz[i-1]+(nz[i]-nz[i-1])*(10.0*zz-z[i-1])/(z[i]-z[i-1]));
}

Int_t FindBkgField(Double_t B, Double_t dB, vector<Double_t> &Bbkg)
{
  UInt_t i;
  Int_t result = -1;
  for (i=0; i<Bbkg.size(); i++) {
    if (TMath::Abs(Bbkg[i]-B) < dB) {
      break;
    }
  }

  if (i<Bbkg.size())
    result = (Int_t)i;

  return result;
}

void LSCO_test(TString &rge_fln, TString &out_fln, Double_t Bext, Double_t lambda,
               Double_t d, Double_t w, Double_t widthBkg)
{
  FILE *fp;
  char buffer[128];

  Double_t zz, nzz;
  vector<Double_t> z;
  vector<Double_t> nz;

  // read rge-file
  fp = fopen(rge_fln, "r");
  if (!fp) {
    cout << endl << "Couldn't open rge-file " << rge_fln.Data() << ", will quit.";
    cout << endl;
```

```
  return;
}

bool header = true;
while (!feof(fp)) {
  fgets(buffer, sizeof(buffer), fp);
  if (header) {
    memset(buffer, 0, sizeof(buffer));
    header = false;
    continue;
  }
  zz = nzz = 0.0;
  sscanf(buffer, "%lf %lf", &zz, &nzz);
  z.push_back(zz);
  nz.push_back(nzz);
}
fclose(fp);

// calculate pB - Meissner
Double_t N = TMath::CosH(d/2.0/lambda);
Double_t Bmin = Bext/N;
Double_t BB, BBnext;
Double_t dB = (Bext-Bmin)/1000.0;
Double_t dz;
Double_t zm, zp, zNext;
vector<Double_t> B;
vector<Double_t> pB;
Double_t nn;
for (UInt_t i=0; i<1000; i++) {
  BB = (Double_t)i/1000.0*(Bext-Bmin)+Bmin;
  BBnext = (Double_t)(i+1)/1000.0*(Bext-Bmin)+Bmin;
  B.push_back(BB);
  pB.push_back(0.0);
  // handle zm
  zm = d/2.0-lambda*TMath::ACosH(N*BB/Bext);
  zNext = d/2.0-lambda*TMath::ACosH(N*BBnext/Bext);
  dz = zNext-zm;
  nn = GetNofZ(zm, z, nz);
  if (nn != -1.0)
    pB[i] += nn*TMath::Abs(dz/dB);

  // handle zp
  zp = d/2.0+lambda*TMath::ACosH(N*BB/Bext);
  zNext = d/2.0+lambda*TMath::ACosH(N*BBnext/Bext);
  dz = zNext-zp;
  nn = GetNofZ(zp, z, nz);
  if (nn != -1.0)
    pB[i] += nn*TMath::Abs(dz/dB);
}

// normalize pB
Double_t pBsum = 0.0;
for (UInt_t i=0; i<B.size(); i++)
  pBsum += pB[i];
pBsum *= dB;
for (UInt_t i=0; i<B.size(); i++)
```

```
  pB[i] /= pBsum;

if (w > 0.0) { // only handle bkg if it makes sense
  // calculate pB-Bkg
  vector<Double_t> Bbkg;
  vector<Double_t> pBbkg;
  Int_t range = (Int_t)(5.0*widthBkg/dB);
  for (UInt_t i=0; i<2*range; i++) {
    Bbkg.push_back(Bext+i*dB-5.0*widthBkg);
    pBbkg.push_back(TMath::Exp(-0.5*TMath::Power((Bext-Bbkg[i])/widthBkg, 2.0)));
  }
  // normalize pBbkg
  pBsum = 0.0;
  for (UInt_t i=0; i<pBbkg.size(); i++)
    pBsum += pBbkg[i];
  pBsum *= dB/w;
  for (UInt_t i=0; i<pBbkg.size(); i++)
    pBbkg[i] /= pBsum;
}

if (w > 0.0) {
  UInt_t idx;
  TGraph gg(B.size()+Bbkg.size()/2);
  for (UInt_t i=0; i<B.size(); i++) {
    idx = FindBkgField(B[i], dB, Bbkg);
    if (idx == -1) {
      gg.SetPoint(i, B[i], pB[i]);
    } else {
      gg.SetPoint(i, B[i], pB[i]+pBbkg[idx]);
    }
  }
  for (UInt_t i=Bbkg.size()/2; i<Bbkg.size(); i++) {
    gg.SetPoint(i+B.size()-Bbkg.size()/2, Bbkg[i], pBbkg[i]);
  }
} else {
  TGraph gg(B.size());
  for (UInt_t i=0; i<B.size(); i++) {
    gg.SetPoint(i, B[i], pB[i]);
  }
}

gg.DrawClone("a*");

pBsum = 0.0;
Double_t valB, valpB;
for (UInt_t i=0; i<gg.GetN(); i++) {
  gg.GetPoint(i, valB, valpB);
  pBsum += valpB;
}
pBsum *= dB;

// save pB
fp = fopen(out_fln, "w");

fprintf(fp, "# title: Meissner + background\n");
fprintf(fp, "# parameter: B0 = %lf, lambda = %lf, d = %lf, w = %lf, widthBkg = %lf\n", Bext, lam
```

```
  fprintf(fp, "# ------------------------------------------------------\n");
  fprintf(fp, "# B, pB\n");
  for (UInt_t i=0; i<gg.GetN(); i++) {
    gg.GetPoint(i, valB, valpB);
    fprintf(fp, "%lf, %lf\n", valB, valpB/pBsum);
  }
  fprintf(fp, "# end of file\n");
  fclose(fp);

  // clean up
  z.clear();
  nz.clear();
  B.clear();
  pB.clear();
}
```