King, Stephen (STFC,RAL,ISIS)

From: Paul Kienzle <paul.kienzle@nist.gov>
Sent: 14 July 2014 20:55
To: King, Stephen (STFC,RAL,ISIS)
Cc: sasview-developers@lists.sourceforge.net
Subject: Re: [Sasview-developers] Error estimates from fits

The output below shows that forward difference is consistent with central difference. numdifftools is also consistent. These values are consistent with those returned by the Levenberg-Marquardt solver from scipy.

Park is giving the wrong answer because sasview redefined residuals as residuals/sqrt(N) in sasview/src/sans/fit/ParkFitting.py rather than setting

```
chisq = sum(self.residuals**2)/self.degrees_of_freedom
```

The stated reason in the code is to be consistent with Scipy, which is strange since scipy does not return chisq. The scipy wrapper in sasview uses the incorrect normalization sum(residuals**2)/N as the normalized chisq, rather than:

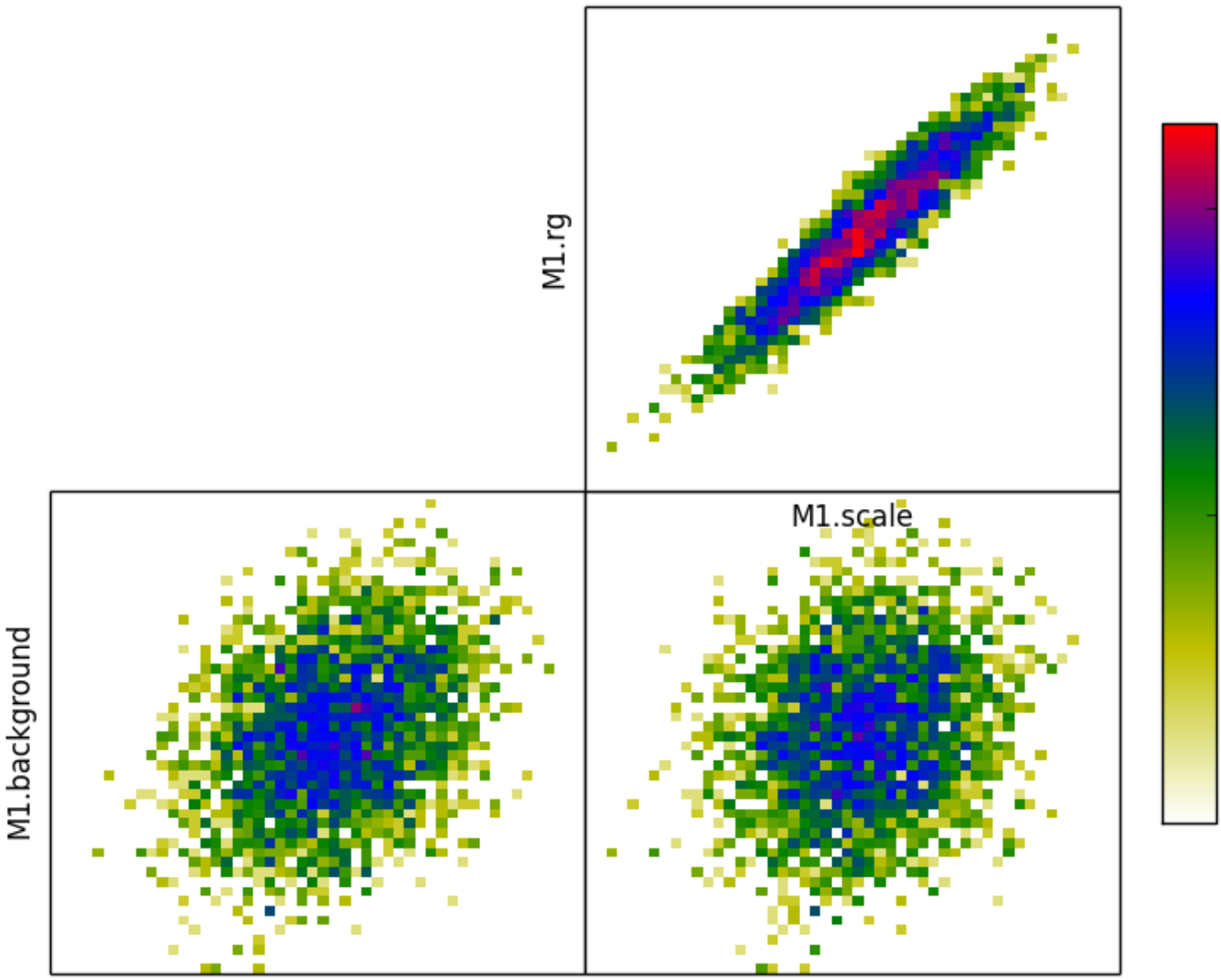
```
chisq = sum(self.residuals**2)/(len(data) - len(parameters))
```

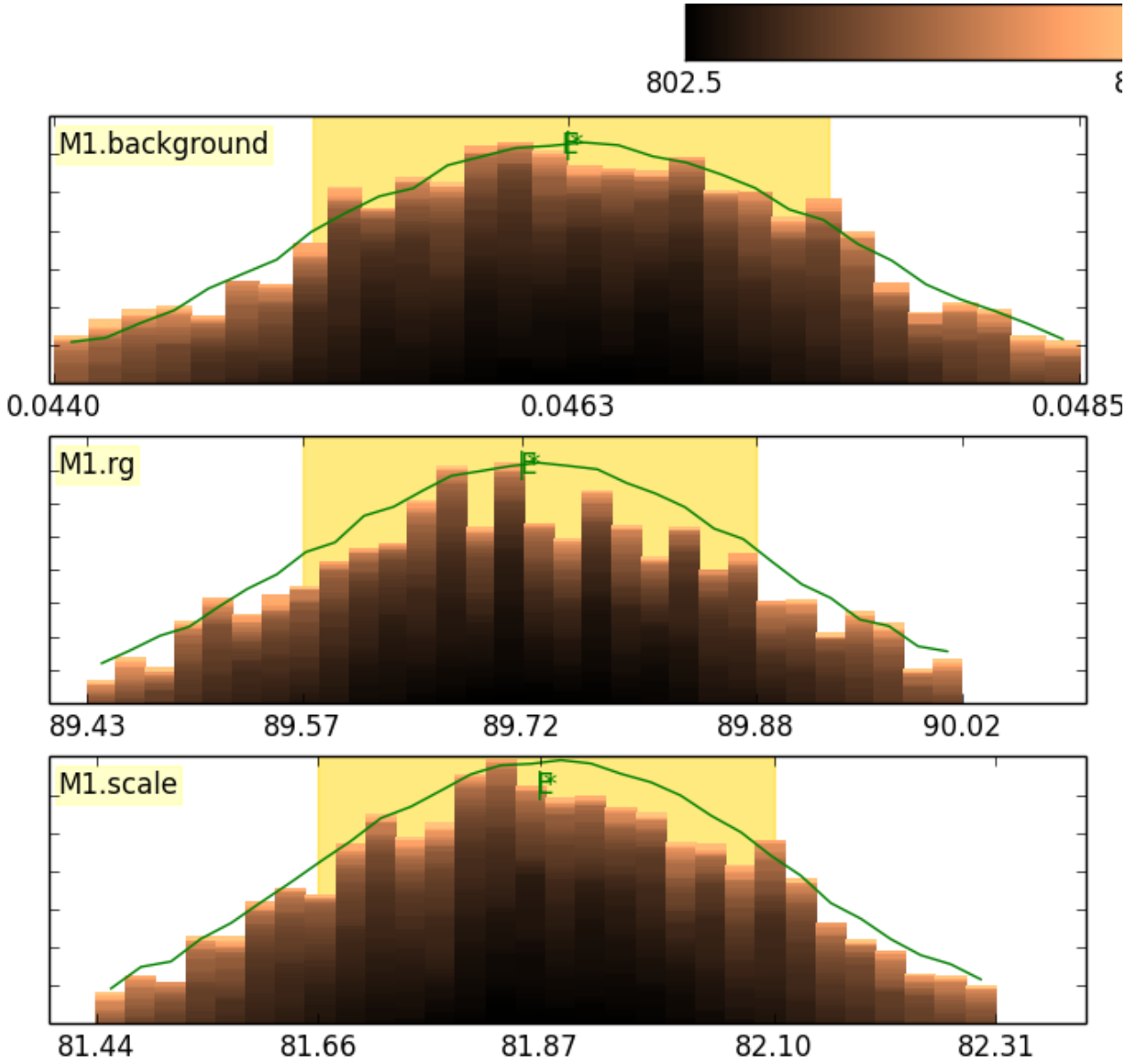
Since both park and scipy are to be replaced with bumps I'll leave it to someone else to decide if these errors need to be fixed.

Running the dream MCMC from bumps, the 68% interval is consistent with 2*uncertainty:

```
              Parameter       mean  median    best [   68% interval]
[   95% interval]
 1        M1.background 0.0463(11) 0.04625 0.04629 [ 0.0451  0.0474] [
0.0440  0.0485]
 2                M1.rg  89.72(15)  89.719  89.727 [  89.57   89.88]
[  89.43   90.02]
 3             M1.scale  81.88(22)  81.873  81.882 [  81.66   82.10]
[  81.44   82.31]
```

As you can see from the attached MCMC plots, the uncertainties hide a strong correlation between scale and Rg.


M1.rg
M1.background
M1.scale



On Jul 14, 2014, at 12:14 PM, <stephen.king@stfc.ac.uk> <stephen.king@stfc.ac.uk> wrote:

Hi Paul,

I get this much from the script before it falls over:

execfile('nd.py')
forward difference svd err [0.00046188 0.21106138 0.16122371]
forward difference qr err [0.00046188 0.21106138 0.16122371]
forward difference ||svd - qr|| = 4.52439784432e-17
central difference svd err [0.00046188 0.2110611   0.1612235]
central difference qr err [0.00046188 0.2110611   0.1612235]
central difference ||svd - qr|| = 2.9039372874e-17

because numdifftools isn't in the Python shipped with SasView.

My options would therefore seem to be a) I install numdifftools and build SasView, or b) I just send you the datafile and modified script...

As it's the end of the day here, do you mind if I do b)?! See attached.

I also ran the fit through Richard Heenan's FISH program (least squares optimiser). I didn't have the polydisperse coil model to hand but at least the output from the ideal model should be indicative:

Gaussian coil for polymers

1	Const	7.936600E+01	9.070E-01
2	Rg	8.487478E+01	6.226E-01
3	BKG	4.860956E-02	4.846E-03

So these uncertainties are larger than the least-squares from SasView, but smaller than those from Park.

Steve

-----Original Message-----
From: Paul Kienzle [mailto:paul.kienzle@nist.gov]
Sent: 12 July 2014 00:48
To: King, Stephen (STFC,RAL,ISIS)
Cc: sasview-developers@lists.sourceforge.net
Subject: Re: FW: [Sasview-developers] Error estimates from fits

On Jul 11, 2014, at 6:47 AM, <stephen.king@stfc.ac.uk> <stephen.king@stfc.ac.uk> wrote:

Thanks, Paul.

With Peter's help - the data was in cansas format so the way your script tried to load it didn't work at first - we got it running.

Attached are the plots using the two sets of uncertainties.

Do I interpret this as the simple case is providing uncertainties that are unfeasibly small?

I interpret this as park giving unreasonably large error bars, but I can't support it (see below).

Instead of forward simulation, we can try a to improve our estimate of the parameter uncertainties, and see if we can track down why the two results differ.

I am confident that QR decomposition and SVD give similar error bars, having tested both algorithms against calibrated least squares problems from the NIST Statistical Reference Datasets (http://www.itl.nist.gov/div898/strd/), which were computed using high precision floating point numbers (128 bit quad precision rather than the usual 64 bit double precision). As they point out on the background page for nonlinear regression (http://www.itl.nist.gov/div898/strd/nls/nls_info.shtml) numerical differentiation will make it hard to get accurate values. It is still possible that the calculation of the covariance matrix as inv(R'R) is faulty, but we can check whether it makes a difference in your particular case.

Instead, I think the most likely difference is coming from the calculation of the Jacobian. With the model and data, you can try computing the Jacobian with the same h = abs(p)*sqrt(numpy.finfo(float).eps) using both the forward difference formula and the center point formula to see if they give radically different

answers. As to which one is more correct, the numdifftools package has a jacobian calculator that uses dynamic stepping.

Please try the attached nd.py which computes these, but first update it with your data loader.

An even better option is to download the bumps-integration branch from svn and set the optimizer to dream. Start with burn=300 and steps=1000 since you already have a fit.

- Paul

Re: theory uncertainty from forward simulation

There are a couple of concepts bundled together in this exercise: one is the confidence interval, which is how well you know the expected value at point x (the confidence interval should decrease to 0 as you perform more measurements) and the other is the prediction interval, which is how well you know the expected deviation at point x (the prediction interval should converge to the measurement uncertainty as you perform more measurements). The additional measurements do not need to be at the same x point; indeed in some models, measurements far away from x can do more to constrain the theory value at x than points near by. Imagine the case of fitting a straight line through a set of points, where you have a large amount of measurement error on each point. As you increase the number of measurements along the line your confidence in the expected value of the line at some point xc will increase, but the actual value for a new measurement at xc will be different from expected due to uncertainty in the measurement process.

The parameter uncertainties computed by sasview correspond to the confidence interval on I(q) for point q. The uncertainty in I(q) corresponds to the prediction interval. So the exercise of simulating a bunch of fits and looking at their distribution as I have done isn't directly comparable to the error bars.

With linear models, both the confidence interval and the prediction interval can be estimated from the fit, and the prediction interval should match the scatter in the data (which should arise from the gaussian uncertainty in the measurement as indicated by the error bars). It looks like we should be able to do the same thing with the covariance matrix to generate prediction intervals for non-linear least squares from the Jacobian at the minimum but I haven't done so. See LinearModel.ci and LinearModel.pi in bumps.wsolve for details (https://github.com/bumps/bumps/blob/master/bumps/wsolve.py)

Steve

-----Original Message-----
From: Paul Kienzle [mailto:paul.kienzle@nist.gov]
Sent: 10 July 2014 18:16
To: King, Stephen (STFC,RAL,ISIS)
Cc: sasview-developers@lists.sourceforge.net
Subject: Re: [Sasview-developers] FW: Error estimates from fits

On Jul 10, 2014, at 10:34 AM, <stephen.king@stfc.ac.uk> <stephen.king@stfc.ac.uk> wrote:

One of my users has been fitting some polymer scattering using the PolyGaussCoil model.

For some reason they decided to compare the simple and complex fitting engines. The good news is they got the same fit parameters! But the parameter uncertainties are significantly different for some parameters.

SIMPLE
Background 0.046288 +/- 0.0011378
Scale 81.882 +/- 0.22307
Poly_m 1.1
Rg 89.727 +/- 0.15731

ChiSq/Npts 21.939

COMPLEX
Background 0.046288 +/- 0.00097874
Scale 81.882 +/- 1.9189
Poly_m 1.1
Rg 89.727 +/- 1.3533

ChiSq/Npts 21.939

Their question is three-fold:
- Is this to be expected?
- Why the difference in uncertainties?
- Which uncertainties should they take as 'real'?

The uncertainties should be similar.

Park computes the Jacobian matrix using the center point formula:

J[i,j] = (f(p[i]+h) - f(p[i]-h)) / 2h for parameter p[i], data point Iq[j]

where

h = 1e-8 * (max - min) for bounds constrained parameters
h = p[i]*1e-8 for unconstrained parameters

The covariance matrix C is inv(J'J), computed using singular value decomposition as:

U V S = J
C = inv(J'J) = V inv(S S) V'

Similarly, scipy computes the jacobian using the forward difference formula:

J[i,j] = (f(p[i]+h) - f(p)) / h for parameter p[i], data point Iq[j]

where

h = abs(p[i])*1e-8

The covariance matrix C is computed using QR decomposition as:

Q R = J
C = inv(J'J) = inv(R'R)

The center point formula should be more accurate, though twice as expensive to calculate. SVD and QR decomposition are equally stable but R'R is ill-conditioned, so I trust the SVD calculation more. Even so, I don't expect a factor of 10 difference between the two.

To determine which uncertainties are more real, you could run a forward simulation on the model with parameters pulled at random from the two uncertainties and see which is a better match to the uncertainty of the data. Drop the attached sim.py into your data directory and change to the correct file name.

Using the > Tool > Python Shell menu you can start a python shell in which you can type:

```
pwd  # show what directory you are in
cd data # change to your data directory
execfile('sim.py')
```

This will show the 1-sigma uncertainty in the theory function values, which should be on the same order as the 1-sigma uncertainty on the data values. I haven't tested the script on windows, but it worked on my linux box.

- Paul

--
Scanned by iCritical.

<forward-simulation-simple-uncertainties_vs_data.png><forward-simulation-complex-uncertainties_vs_data.png>

--
Scanned by iCritical.

<25330-add_var_merged.xml><nd_hack-for-xml-format-data.py>