Review (round 2) gmd-2021-175

Author: Benoît Pasquier

I have found the responses and revisions of the authors appropriate and **recommend publication after some minor revisions**, listed below. (Apologies for the likely frustrating notation comments.)

- I should have been clearer in my first review point 38:

  > Eq. (30) + many lines: "f_{global}". Usually non-variable subscripts are typeset upright.

  This comment should have mentioned all non-variable subscript and superscript. E.g., I would replace `f^{Base}_\textrm{T}` and `r^{Base}_{qj}` with `f^\textrm{base}_\textrm{T}` and `r^\textrm{base}_{qj}` (i.e., f^{Base} and r^{Base} with $f^{\textrm{base}}$ and $r^{\textrm{base}}$) where I also avoided having a capital "B" for consistency. The authors should check the entire manuscript for any non-variable subscript/superscript and correct them.

- As per GMD's guidelines (https://www.geoscientific-model-development.net/submission.html#math) vectors such as **x**, which I guess was LaTeX'd from `\mathbf{x}` (first appearance l. 117) should be typeset "in boldface italics", i.e., ***x***. This is easily done using the `\vec{x}` command provided by the Copernicus LaTeX template.

- Units are missing in almost every figure and should be added.

- l. 178: Eq. (2): the sums should not start at i=1. They should be written as `\sum_{i\in j}` instead of `\sum_{i=1}^{i\in j}` ($\sum_{i\in j}$ instead of $\sum_{i=1}^{i\in j}$).

- l. 132: Mathematical symbol D should be in italics, i.e., D. (Use "D".)

- l. 168: "the length of the vector **r**" should be spelled out for clarity with maybe something like "d, the number of r_i terms". (Also note that otherwise the vector **r**, which should be boldface italic, is not even defined.)

- Table 1: In retrospect, my suggestions for experiment names were not great. For readability, I think it might be better to have shorter names and avoid underscores. What about:

  - "CTL" for the CMA-ES run (the control run),
  - "$\textrm{SMOOTH}_1$" for the "D_smooth_1 run" and so on,
  - "$\textrm{NOISY}_1$" for the "D_noise_rand1" run and so on, and
  - "$\textrm{SPARSE}_1$" for the "D_smooth_sparse_1" run and so on?

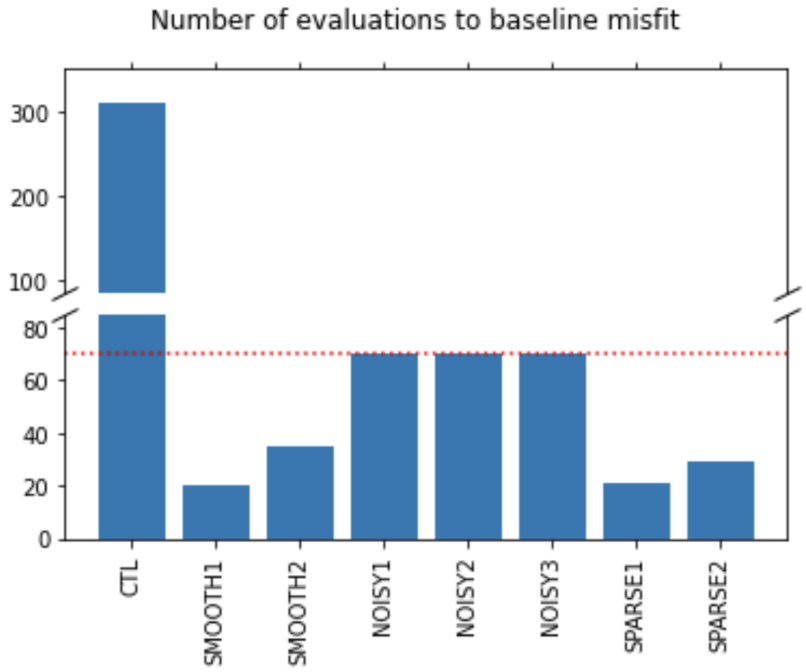
- l. 221: Use the `\times` symbol ("×") rather than the letter "x".

- Fig. 4:

  - Maybe a line for the target value could be added in the background? (and a ±5% band?)
  - Maybe show the 10 CTL (C_smooth) starting points as tiny dots?
  - There is a lot of unused vertical space in each panel. Maybe the y-axis limits can be tightened a bit? E.g., Fig. 4c shows maximum K_PHY values of about 0.2, but the y-axis goes up to 0.5.

    - The legend could be simplified to only say that circles are starting points and crosses are optimized values? (Maybe use black for the legend and then give a different color than black for the smooth values.
    - Speaking of color, a color-blind-friendly palette could be used here instead of plain black, red, and blue (e.g., colorbrewer's qualitative colors (https://colorbrewer2.org/#type=qualitative&scheme=Dark2&n=3), but there are many others!)
    - The legend could be placed at the bottom rather than in the middle to avoid visually breaking the x-axis alignments of top and bottom panels.

- Fig. 5: This is a key figure that was added in response to the 1st round of review to replace the now Table C. Yet, the main message — that DFO-LS requires much less evaluations than CMA-ES — is now obfuscated by the use of different scales and 2 *y*-axes. Better to show both on the same scale and let the visual speak for itself! The broken-axis suggestion (from the 1st review round) was not used, although it would make this much clearer in my opinion. Here is what I had in mind, e.g., for Fig. 5a (The red dashed line shows the imposed limit on evaluations for DFO-LS runs.):



I understand MATLAB is not suited for broken-axis plots, so to be helpful I have provided below the python code that produces the broken-axis plot shown above. This code can easily be used as a template to reproduce each panel in Fig. 5. Python code:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(19680801)

experiment_names = ["CTL", "SMOOTH1", "SMOOTH2", "NOISY1", "NOISY2",
"NOISY3", "SPARSE1", "SPARSE2"]
nevals_to_basline_misfits = [309,20,35,70,70,70,21,29]

# If we were to simply plot pts, we'd lose most of the interesting
# details due to the outliers. So let's 'break' or 'cut-out' the y-
```

```
axis
# into two portions - use the top (ax1) for the outliers, and the
bottom
# (ax2) for the details of the majority of our data
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
fig.subplots_adjust(hspace=0.1)  # adjust space between axes

# plot the same data on both axes
ax1.bar(experiment_names, nevals_to_basline_misfits)
ax2.bar(experiment_names, nevals_to_basline_misfits)

# zoom-in / limit the view to different portions of the data
ax1.set_ylim(85, 350)  # outliers only
ax2.set_ylim(0, 85)  # most of the data

# hide the spines between ax and ax2
ax1.spines.bottom.set_visible(False)
ax2.spines.top.set_visible(False)
ax1.xaxis.tick_top()
ax1.tick_params(labeltop=False)  # don't put tick labels at the top
ax2.xaxis.tick_bottom()

# Now, let's turn towards the cut-out slanted lines.
# We create line objects in axes coordinates, in which (0,0), (0,1),
# (1,0), and (1,1) are the four corners of the axes.
# The slanted lines themselves are markers at those locations, such
that the
# lines keep their angle and position, independent of the axes size or
scale
# Finally, we need to disable clipping.

d = .5  # proportion of vertical to horizontal extent of the slanted
line
kwargs = dict(marker=[(-1, -d), (1, d)], markersize=12,
              linestyle="none", color='k', mec='k', mew=1,
clip_on=False)
ax1.plot([0, 1], [0, 0], transform=ax1.transAxes, **kwargs)
ax2.plot([0, 1], [1, 1], transform=ax2.transAxes, **kwargs)

plt.axhline(y=70, linestyle=":", color="red")

plt.xticks(rotation=90, ha='center', va='top')

plt.suptitle("Number of evaluations to baseline misfit")

plt.show()
```

- Note 1: Do not pay too much attention to the code comments in the snippet above because they are from the matplotlib broken-axis example (https://matplotlib.org/stable/gallery/subplots_axes_and_figures/broken_axis.html) from which this code was slightly edited.

- Note 2: No local python installation is needded: To produce the plot above this code was edited and ran online using Binder (https://mybinder.org/)).