How-to ...

get larger SMBs

This document is useful for you if you think you should be able to get larger SMBs but you do not understand why they stay so low. The first two chapter s are useful for all users of AAPS, the later ones require extra features introduced by autoISF. Here is the sequence of checks to go through to identify the current bottlenecks.

1. Get the most out of regular AAPS

Most users are not aware if and when they hit the standard AAPS limit called *maxBolus*. Its value can be seen at the end of the *script debug* section in the SMB tab. It is derived from your profiles current basal rate and the value of *maxSMBBasalMinutes* or *maxUAMSMBBasalMinutes*, respectively, depending on whether you are in UAM mode or not. The value is calculated by this formula:

maxBolus = profile current basal rate * max(UAM)SMBBasalMinutes / 60

As mentioned before, your current value of *maxBolus* is listed in the SMB tab at the end of the Script debug section. Next you need to check whether this limit was reached or you stayed below. To find out you have to read the reason part of the *results* section following the *script debug* in the SMB tab. If it does not mention anything about *maxBolus* then you are fine. If however it explicitly lists the value of *maxBolus* then you know you hit that limit for that loop event.

If you missed reading the SMB tab you can still find the value and the same reason printed in your logfile. If you are not sure in which logfile to search or whether it happened several times within a short time frame you may use my python script "*find_string_batch.py*" (see https://github.com/ga-zelle/Scan-APS-logfiles) which will scan a list of logfiles. That list of logfiles is defined just like the list definition used for the emulator. The DOS command to search for active maxBolus limits being reached looks like this:

```
find_string_batch <wild-card-AAPS-logfile-name(s)> dummy "; maxBolus "
```

If scanning the logfiles is too cumbersome you can run the afore mentioned emulator (see https://github.com/ga-zelle/APS-what-if) for that time window and see the *maxBolus* message there in much shorter form in the ...orig.txt-file like in this example:

```
…
naive_eventualBG 54, 60m 0U/h temp needed; last bolus 5.5m ago; maxBolus: 0.1
---------- Reason --------------------------
COB: 8.539714285714298, Dev: 60, BGI: -3, ISF: 70, CR: 12, Target: 100, minPredBG 117,
minGuardBG 119, IOBpredBG 108, COBpredBG 98, UAMpredBG 155;  insulinReq 0.24; maxBolus 0.1;
setting 60m low temp of 0U/h. Microbolusing 0.1U.
```

In the example above the regular *maxBolus* info is highlighted in yellow. The note highlighted a bit later in green shows that the *maxBolus* limit was active.

What can you do to increase your *maxBolus?*

- Check your settings of both, *maxSMBBasalMinutes and maxUAMSMBBasalMinutes*
  Their default values are 30 minutes but you can increase them as far as 120 minutes which is 4 times as strong.
- You may think about increasing your profile basal rates just for meal times but that is risky. Just imagine you take the meal at another time then you get too much insulin while the meal is missing and too little when you actually take it. In summary not a good idea.

2. SMB limited by max_iob

The loop algorithm may initially come up with a bolus recommendation which would exceed the upper limit as set by max_iob. In such a case the insulin required is reduced to stay within range. However, it is hard to see whether this actually happened or not. Like in the previous section there is only an indirect sign in the Reason statement. I scanned my logfiles for the whole month of April 2023 with the findstr method mentioned before by issuing the command

```
s:\find_string_batch.py y:\2023-04\AndroidAPS._2023-04-*.zip dummy ", max_iob "
```

To my surprise I found 23 occurrences. Here is one example:

```
22:41:32.933 [androidx.work-2] D/APS: [DetermineBasalAdapterSMBJS.invoke():127]:
Result: {"temp":"absolute", …(etc.)… , max_iob 8,  insulinReq 4.95. Microbolusing
2.1U. …(etc.)
```

The part highlighted in green is the only indicator that the *max_iob* limit was active. To make it more visible the SMB tab in autoISF 2.2.8.1 includes an extra line explaining the case explicitly:

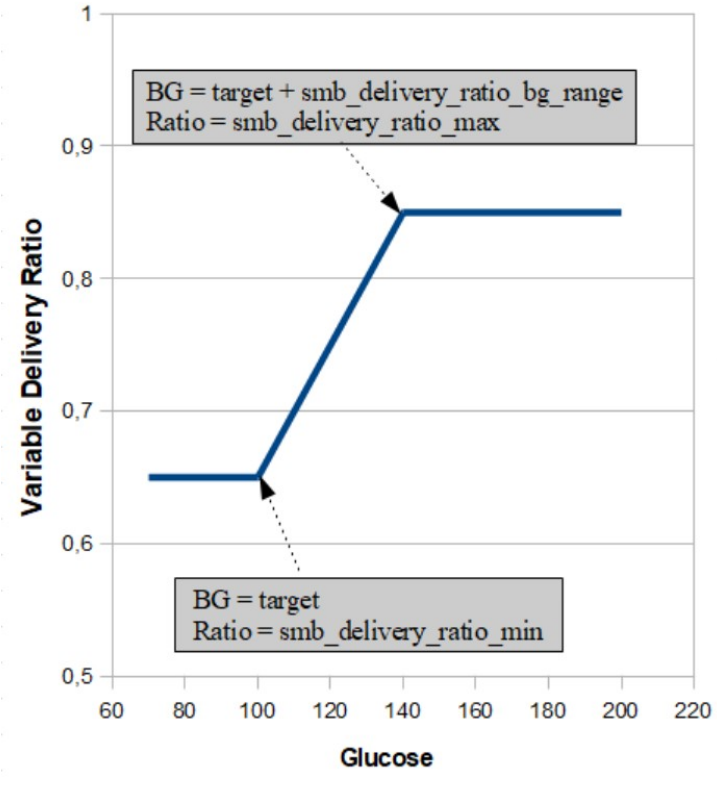
```
…
InsReq 5.37 capped at 4.95 to not exceed max_iob
IOB 3.05 > COB 0; mealInsulinReq = 0
…
```

Before increasing *max_iob* just because of that think about whether that is still safe for you to do. I kept mine at 8U as a safety net.

3. Get a larger share of the insulin required (*insReq*)

Note: As mentioned at the beginning the actions described in this section and all following ones are only available with the extensions included in autoISF. All the new parameters can be set and changed in the autoISF submenu included at the end of the OpenAPS SMB Preferences. For finding them in the menu system I recommend an alternative, namely using the filter field at the top of the Preferences Screen. There you enter any string like “smb_delivery” and it will present a short list of the matching preferences.

One of the important interim results of the loop is this value of *insReq* to get to target safely. By default 50% of that *insReq* will be delivered as SMB. This is a safety measure to leave some room for error and especially with follower setup it might happen that the SMB gets requested twice, namely from master and follower. Without follower setup the *smb_delivery_ratio* can be increased to a different but fixed value ranging from 0.5 up to 1.0.



If you want to be careful or your CGM is unreliable you may want to have a ratio closer to 0.5 at low glucose and allow higher ratios at higher glucose. For that purpose you can use the alternative definition of a linearly increasing ratio starting at target with

smb_delivery_ratio_min and growing up to *smb_delivery_ratio_max* at target+*smb_delivery_ratio_bg_range*.

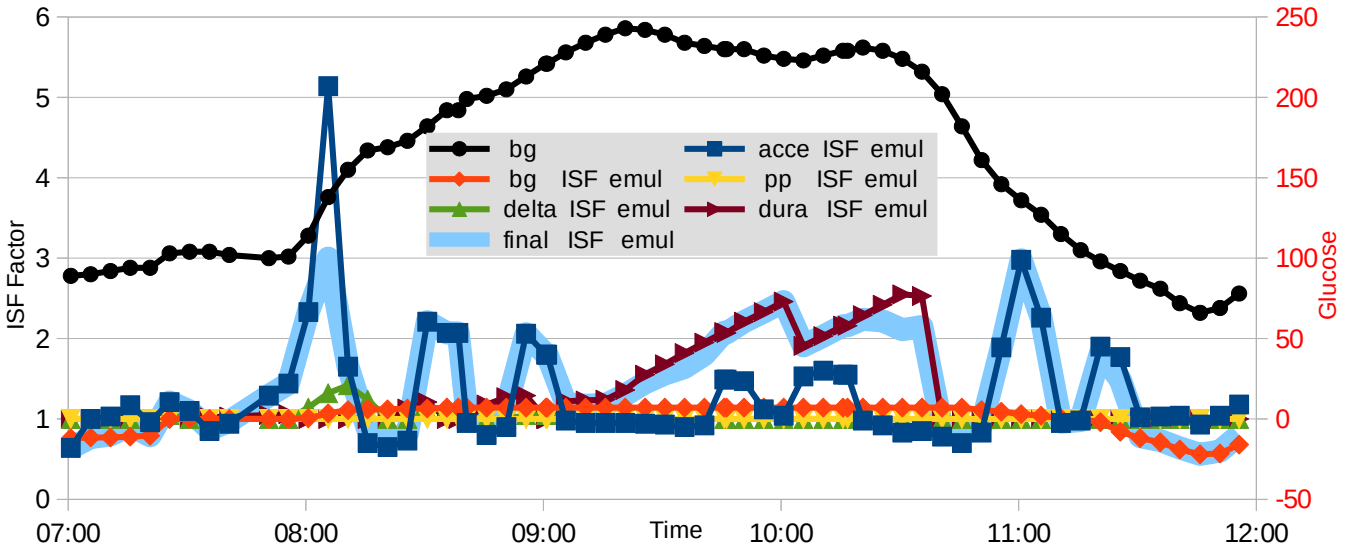
In the diagram above, when BG is at or below 100, AAPS will deliver 0.65 of the calculated *insReq*. As the BG rises, so will the delivery ratio to a maximum of 0.85 of the *insReq* at a BG of 140 or above. If the range was increased to the maximum value of 100, the 0.85 delivery ratio would not be achieved until the BG reached 200.

4. Go beyond the 120 minute limit of *max(UAM)SMBBasalMinutes*

If your *maxBolus* is still not sufficient because your basal rate is fairly low or you want to go for full loop and cannot satisfy the insReq needs you can use *smb_max_range_extension* as a factor to effectively multiply *maxUAMSMBBassalMinutes*. As outlined in chapter 1, you should check whether *maxBolus* is now sufficient.

5. The *insReq* may already be below expectations

In such a case some of the weights in autoISF may be too low and therefore you do not reach intended levels of SMB. The easiest way to find out which of the weights should be increased is to run the emulator with the no-change scenario. This will create a CSV-file which you load into your favourite spreadsheet and create an X-Y-Scatter diagram of the 6 autoISF weights. You can also include a scatter diagram for the BG values scaled on the secondary y-axis on the right. The final graph will look like this:



Irrespective whether this is the best modification and only for arguments sake let us assume you wanted to have more SMB between 9:00 and 10:30 UTC time. The wider band in light blue is the final ISF factor which is more or less the envelope of the individual factors. Here dura ISF is in control and you could obviously increase *dura_ISF_weight*. Alternatively you could raise *higher_ISFrange_weight* which looks very low compared to general formulas describing ISF as a function of BG like the 1800 rule or DynamicISF. You can now run a few iterations with the emulator to find a weight strong enough to lift the final ISF curve and check the *emulated insReq* and *emulated SMB* columns in the output of the emulator.

6. The final ISF factor is limited by autoISF_max

Obviously the better strategy is to get more SMB upfront like indicated by acce ISF around 8:00 in the diagram above. The SMB tab shows whether the final ISF was capped. Again, if you missed the 5 minutes where it was shown the easiest option is to check the emulator output. The diagram above is

already an indication because the final ISF seems to stop at 3 although acce ISF suggests more than 5. The TXT-file lists these lines at 08:05 and confirms that indication:

```
...
  created at= 2022-09-07T08:05:39.466Z
---------- Script Debug --------------------
Autosens ratio: 1;
Basal unchanged: 0.64;
ISF unchanged: 50.8
; CR: 9.230769230769232
acce_ISF adaptation is 5.14
bg_ISF adaptation is 1.07
delta_ISF adaptation is 1.32
dura_ISF by-passed; bg is only 0m at level 138
strongest ISF factor 5.14 limited by autoisf_max 3
final ISF factor is 3
…
```

As an action you may want to raise *autoISF_max* although in the above example the maximum value was already reached.

7. Summary of steps

1. Check that *maxSMBBasalMinutes* or *maxUAMSMBBasalMinutes* are sufficient.
2. Check that *max_iob* is sufficient but appropriate
3. Adjust delivery ratios and BG range settings
4. Adjust the range extension
5. Adjust the various autoISF weights
6. Adjust autoISF_max

Status: 2.May 2023 12:20