

Table of Contents

How it works in principle.....	2
How to find improved settings at minimal risk.....	2
Comments and current limitations.....	2
Tested system settings.....	3
Installation.....	3
On Windows10.....	3
On the AAPS phone.....	3
Define your variant in the VDF (Variant Definition File).....	4
Execute the analysis in a DOS command window.....	5
Example.....	8
The „pred“ output option.....	10
The „flowchart“ output option.....	12
Execute the analysis using the GUI front end.....	13
Select inputs.....	13
Select graphics options.....	14
Execute the analysis.....	14
Inspect results.....	15
Executing the analysis on the AAPS phone.....	16
Start the emulation.....	16
The results.....	18
Stop the emulation.....	18
Tipps and tricks.....	19
Appendix – Some more examples of VDF entries	20

How it works in principle

The idea is to reduce risks when changing AAPS settings. I first test the impact of such changes offline on a PC by emulating in a python script what the loop would do. That python is a translation of the original AAPS „determineSMB-basal.js“ java version.

The code scans an existing AAPS logfile and extracts the results of the original run. Also the inputs for "determine-basal" like last glucose, IOB array or profile definition are extracted. Here profile is not the usual daily pattern of basals etc. but a collection of settings like which SMB feature is ON or OFF; for details go to the „SMB“ tab in AAPS and scan the section named „Profile“.

After collecting all these inputs the emulator will also read your definition of alternative AAPS settings like

- different temp target
- different SMB settings
- different basal safety factors
- different ISF (here named „sens“ !) or IC values

Settings like those related to meals already eaten or insulin already dosed are handled in other modules of AAPS and are therefore out of reach here. For the same reason changes to Autosense cannot be handled and the only meaningful change you can do is set it to „1“ which means switch it off.

Then determine_basal.py is called to rerun the analysis and produce results in the working folder for comparing the original situation with the emulated scenario.

How to find improved settings at minimal risk

Because this tool is a virtual analysis you can emulate many changes „in silico“ first without risk to your health should those changes turn out to be unfavorable. The steps to undertake with the help of this tool go as follows:

1. Identify a situation you want to improve and get the relevant logfiles
2. Read the related section of the docs
3. List the settings that may have an impact
4. Use this emulator on the PC to try which of those settings have the desired impact
5. Tune the level of the favorite setting to deliver the desired improvement
6. Rerun the emulator on the AAPS phone so you see the result quasi in real time and fine tune the setting. Here use the variant with the lower insulin delivery as the master in AAPS and the other one as the emulated version. This way you can decide each time an incremental insulin dose is suggested whether you really want to apply it manually.
7. Once things look OK and safe implement the setting in AAPS itself

Steps 4-6 are those not available without this tool and add extra safety when changing settings.

Comments and current limitations

This is is not a complete virtual loop! Please keep in mind that the variant shows a new and independent decision at each timestep. It is not a sequence of decisions building on each other because there is no forecasting what a different decision would have lead to at the next time step. Therefore you need to carefully consider the results at each timestep separately and consider whether it is safe enough to be activated in AAPS. Once this emulator gives me promising hints and the parameter is tuned I go to quasi "open loop", e.g. for a few days I manually apply an increase in SMB as a correction bolus before going live.

The code still contains dead sections because it is a spinoff from scanAPSlog.py, a more complete

analysis of the logfiles. I use that to populate my own local database for treatments, sgvs, etc. independant of Nightscout.

The code contains many (in)active print statements for debugging purposes. I am still learning python and have not yet taken the lesson on proper debugging.

Be aware that maxBolus=0 in the tabular output file means maxBolus was high enough to not limit the SMB and was therefore not listed explicitly in the original logfile by AAPS.

Glucose values are assumed to be in mg/dl.

The code only works for orefl with SMB mode. With some effort it could be adapted to AMA.

Tested system settings

- Windows10, 64 bit
- python 3.7, 3.8
- some python libs like matplotlib (see the import statements in vary_settings_core.py)
- AAPS versions for logfile: orefl & SMB for 2.3 / 2.5 / 2.5.1 / 2.6.1 / 2.7
- AAPS versions for determine_basal: orefl & SMB for 2.3 / 2.5 / 2.5.1 / 2.6.1 / 2.7

Installation

Download the Github repo as a zip archive.

On Windows10

Extract it to a folder containing

- "determine_basal.py" as the emulation of the original java version
- "vary_settings_GUI.py" which manages the process in an interactive window
- create a shortcut on your desktop to point to this GUI file.
- "vary_settings_batch.py" which alternatively manages the process in a DOS window
- "vary_settings_core.py" which contains the core programme
- "<AAPS logfile(s)>" (zipped or unzipped) for the period to be investigated
- "<variant_label>.dat" which contains the definition of the settings you want to change

On the AAPS phone

- From the playstore download and install qpython3 („QPython 3L – Python for Android“ by QpythonLab).
- This should create a folder „qpython“ at the top level
- Go to its subfolder „scripts3“
- From the above Github archive downloaded above extract the following scriptfiles (i.e. the python programmes) to this subfolder
 - „determine_basal.py“
 - „vary_ISF_batch.py“
 - „vary_ISF_core.py“
- "<variant_label>.dat" , the VDF (see below), is the same as on Windows and is copied to the same folder containing the AAPS logfiles.

Define your variant in the VDF (Variant Definition File)

Create or edit "<variant_label>.dat", your VDF. It defines the changes for your what-if scenario. You have access to the relevant lists or dictionaries as shown in the tab „SMB“, sections „Glucose-data“ through to „Autosense-data“. The general format per row is:

```
.  
.   
<array> <item in array> <new value or formula> [<optionally: #### any comment>]  
.   
.
```

The entries may be separated by several BLANKS and aligned for better readability. Their meaning is as follows:

- *<array>*
name of python dictionary; the available names are
 - autosens_data do not change; gets calculated inside AAPS elsewhere
 - glucose_status not really meaningful
 - currenttemp not really meaningful
 - iob_data do not change; gets calculated inside AAPS elsewhere
 - meal_data do not change; gets calculated inside AAPS elsewhere
 - profile most variations will happen here
 - new_parameter to hold emulator specific information like AAPS version

If array is not a recognised name the row will be skipped.

No BLANKS allowed here.

- *<item in array>*
Item in <array>
No BLANKS allowed here.
- *<new value or formula>*
Boolean, numeric or string value or expression that evaluates accordingly
May contain BLANKS but not '####'

optionally also:

- *<#### any comment>* an optional comment after a block of '####'

For detailed content of the arrays and their correct spelling see the original logfile and look for rows containing "[DetermineBasalAdapterSMBJS.invoke():" or check the appropriate sections in the AAPS „SMB“ tab.

CAUTION:

Whereas AAPS will check for limits and deactivate some SMB settings this emulator does not know about them and just goes ahead. Therefore it is your responsibility to check whether AAPS would accept the settings from your variant definition file.

Example: This repo contains the variant definition file „Demo_Sports_Adaptations.dat“. It will be used for a logfile covering 2-3 hours of playing badminton. I like to stay below 140mg/dl because otherwise I lose concentration and react too slow. In this case it worked quite well but what if I tweak things a bit?

```
profile min_bg 101 ### just above the temp target threshold for SMB  
profile max_bg profile['min_bg'] ### redefine target  
#profile target_bg int((profile['min_bg'] + profile['max_bg'])/2) ### example of numerical expression  
profile tempTargetset True ### spelling error  
profile temptargetset True ### i.e. not using NS profile target  
profile allowSMB_with_high_temptarget False ### was True at the time
```

1. The first row changes the upper target from 100 to 101 to move it just outside of the „enableSMB_with_temptarget“ range.
2. The second row sets the upper target value the same as the lower one with a simple expression.

It needs to be the same because I want to simulate a temp target which does not allow for a finite bandwidth if defined in the AAPS GUI.

3. The third row is ignored because „#rofile“ is not a recognised dictionary. But just look at the rather complex expression assignment possible. As far as targets are concerned the „target_bg“ is defined as the average of „min_bg“ and „max_bg“ anyway inside determine_basal meaning this row is not required.
4. The fourth row tries to redefine the flag for „tempTargetset“ as True. However, that spelling is not really correct because there are 2 swaps in lower case and upper case spelling.
5. The fifth row keeps the flag for „temptargetSet“ as True and this time in correct spelling. There is no check to verify the target bandwidth is zero in such a case.
6. Finally the sixth row disables SMB for higher TempTargets, i.e. above 100.

In summary the target increases from 100 to 101 which is now in line with the general recommendation for sport by disabling SMB.

For first time use I recommend to leave things unchanged and verify that the results agree with the original. Minor numerical deviations are probably due to roundings in the original logfile.

There are three special cases of quasy arrays that were introduced to handle interim or time varying assignments:

- Temp <var_name> <value or expression>
- You can use your own names for variables to hold interim values which can be referenced in statements following it
- STAIR <UTC-date/time> < value from this time onwards>
This is like defining a basal profile
- INTERPOL <UTC-date/time 1> < value at this time>
INTERPOL <UTC-date/time 2> < value at this time>
This uses linear interpolation and extrapolation to define a time dependent function

An example shows possible usage of how to define and how to reference the functions:

STAIR	2020-04-14T00:00:00Z	111
STAIR	2020-04-14T12:50:00Z	125
STAIR	2020-04-14T13:10:00Z	131
INTERPOL	2020-04-14T13	100
INTERPOL	2020-04-14T13:30:00Z	140
profile	min_bg	round(INTERPOL) - 20 ### time varying value
profile	max_bg	STAIR - 10 ### time varying value

Execute the analysis in a DOS command window

For those who are less fit in DOS commands I created a GUI front end which collects the inputs in a form (see appendix). Otherwise, open a DOS command window, navigate to the working folder containing "<variant_label>.dat" and enter

DOS-prompt>[python] vary_settings_batch.py <logfile(s)> <options> <VDF> [<from> [<to>]]
The initial „python“ can be ommitted if the DOS path definition includes the python folder.

The meaning of the arguments is as follows:

- <logfile(s)>
the AAPS logfile (still zipped or unzipped) downloaded from Android.
Optionally you may use the wild characters (? for matching any single character; * for matching any string) to match several logfiles in the directory which provides a longer time window to evaluate. An example could be „AndroidAPS._2020-04-02_*“ to handle all logfiles of that day.
- <options>
"All" or a slash separated list of things to be shown so it does not overload the plot.

Things to be shown are items from this list:

- `insReq` is the total insulin required at the current time step
- `maxBolus` is the upper limit of allowed SMB
- `SMB` is the actual SMB to be delivered
- `basal` is the actual basal insulin to be delivered
- `target` is the lower and upper target
- `bg` is the original glucose value
- `cob` is the original COB value to help understanding the situation
- `iob` is the original IOB value to help understanding the situation
- `activity` is the original insulin activity to help understanding the situation
- `as_ratio` is the Autosense ratio
- `ISF1` is the insulin sensitivity factor
- `range1` parameters of nearly unchanged BG
- `slope1` parameters of nearly linear change in BG
- `pred` plots the prediction curves per timestep of the emulated scenario including hints if SMB needed to be disabled; you get a quasi animation by using forward/backward keys in the pdf-file created with one page per loop execution
- `flowchart` in the graphics output only it plots a flowchart of how determine-basal was transversed with all the if-branches executed.

With the special flag „-“ in front of any option you can suppress individual <plot-options>. I normally use „All/-pred/-flowchart“ which results in everything apart from „pred“ and „flowchart“ in order to speed up the programme significantly in routine analyses.

- <variant_label> or alternatively:
- <variant_label>.dat

A label for the result file names and plot. It is also the base name of the VDF "`<variant_label>.dat`".

The alternative method of giving the filename directly may be easier especially when using the DOS type ahead method

The optional parameters may be used to define a time window to be evaluated, whether for a single logfile or a list of them. Please be aware that the date/time combination is given in UTC format, i.e. in Greenwich time without daylight savings effect:

- <from>
Define the beginning of the time window like „2020-04-02T22:00:00Z“ or for short „2020-04-02T22“ which is midnight of April 2. If you want to skip the start time then just use „2000“ as a very early year.
- <to>
Optionally also define the end of the time window, again in the readable UTC format used in the logfile system.

The process will create the following files:

- <logfile>.<variant>.log
echo of parameter changes as requested by <variant>.dat
- <logfile>.orig.txt
extract of original logfile with entries relevant for the loop result; for the same logfile this will always be the same name and be independent of the variant; therefore there is no harm when a different variant is emulated and it gets overwritten.
- <logfile>.<variant>.txt
equivalent, emulated subset for the variant loop result
- <logfile>.<variant>.tab
comparison of all key values showing original versus emulated run; summary lines at the

¹ For details see my branch autoISF

bottom show minimum and maximum values found within the time window as well as total basal and total SMB delivered

- `<logfile>.<variant>.pdf`

plot of the tabular results; if the previous file with that name is still open in pdf-viewer a warning will be printed until it is clear to proceed.

A table with the subset key values as selected by the options list is also output to the screen directly.

The plot is saved in a predefined resolution(200dpi) and size(9 x up to 12 inches, portrait). Without the „pred“ option is is also displayed interactively and things like aspect ratio or margins can be modified. After such modifications the graph can be saved manually by clicking the disk symbol, preferably in jpeg or png format which makes it easier to later include it in other documents. With the “pred“ option you can inspect it in a pdf-viewer and transverse the history of the logfile with the forward and backward keys.

A file compare between the two txt-files is possible, but mainly the formating is still different and flags too many lines although numerically equivalent. Personally I find the table and plot more useful.

Example

What do the result files look like for the example from above?

First let us check „AndroidAPS. 2019-11-13 00-00-00 .7.Demo Sports Adaptations.log“:

```
===== Echo of what-if definitions actioned for variant Demo_Sports_Adaptations
===== created on Wed, 22 Apr 2020 12:02:11 +0200
===== for loop events found in logfile AndroidAPS. 2019-11-13_00-00-00_7

loop execution in row=2050 of logfile AndroidAPS. 2019-11-13_00-00-00_7 at= 2019-11-13T18:23:30Z
edited old value of 100 in profile with min_bg=101
edited old value of 100 in profile with max_bg=101
not actioned: [#rofile], [target_bg], [int((profile['min_bg'] + profile['max_bg'])/2)]
appended new entry to profile with tempTargetset=True
edited old value of True in profile with temptargetSet=True
edited old value of True in profile with allowSMB_with_high_temptarget=False

loop execution in row=5288 of logfile AndroidAPS. 2019-11-13_00-00-00_7 at= 2019-11-13T18:28:30Z
edited old value of 100 in profile with min_bg=101
edited old value of 100 in profile with max_bg=101
not actioned: [#rofile], [target_bg], [int((profile['min_bg'] + profile['max_bg'])/2)]
appended new entry to profile with tempTargetset=True
edited old value of True in profile with temptargetSet=True
edited old value of True in profile with allowSMB_with_high_temptarget=False
...
```

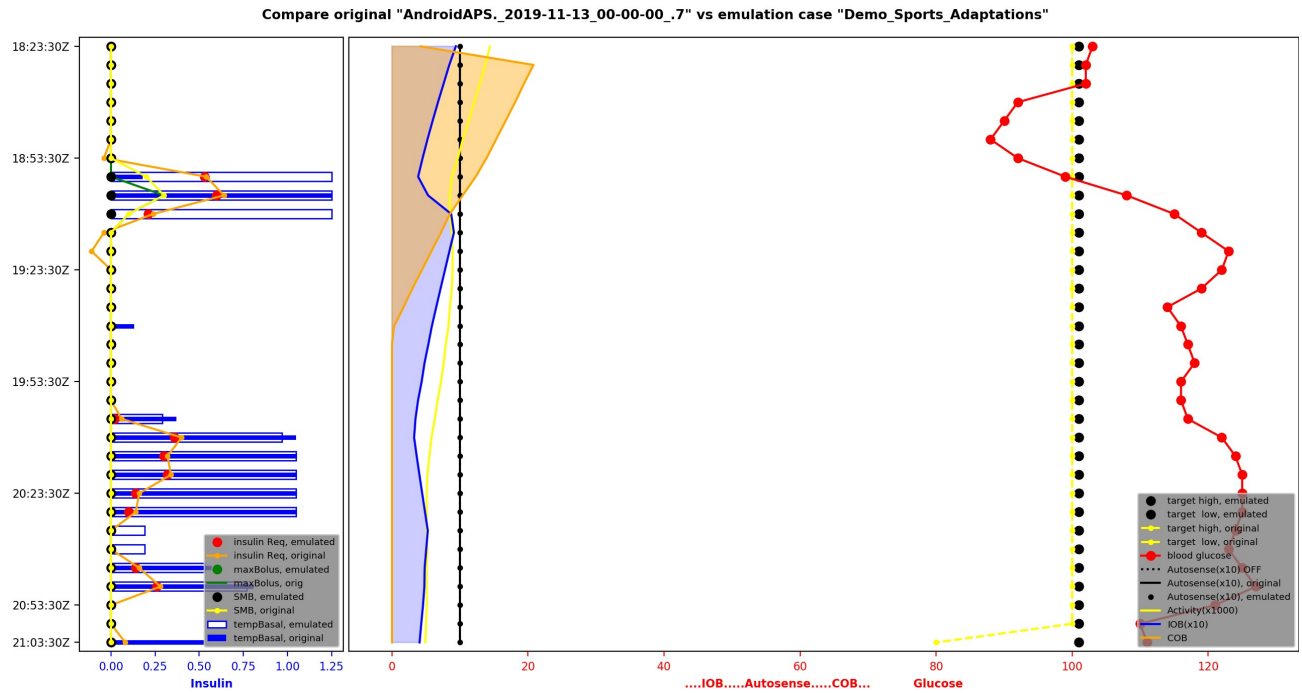
This is an echo of the changes assigned at each timestep. It is useful to check results of complex expressions for values and verify that there were no spelling errors like in the „appended ...“ rows

Next check „AndroidAPS. 2019-11-13 00-00-00 .7.Demo Sports Adaptations.tab“:

id	time	--UNIXtime--	bg	cob	iob	act	-Autosens-		----target----		insulin Req		-maxBolus-		---SMB---		---tmpBasal---		
							orig	emul	orig	emul	orig	emul	orig	emul	orig	emul	orig	emul	
0	18:23:30Z	1573669410.3	103	4.2	0.94	0.014	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
1	18:28:30Z	1573669710.2	102	20.8	0.84	0.014	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
2	18:33:30Z	1573670010.3	102	19.4	0.76	0.013	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
3	18:38:30Z	1573670310.3	92	18.1	0.68	0.012	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
4	18:43:30Z	1573670610.3	90	16.7	0.6	0.011	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
5	18:48:30Z	1573670910.3	88	15.3	0.52	0.01	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
6	18:53:30Z	1573671210.3	92	13.9	0.45	0.01	1.0	1.0	100-100	101-101	-0.04	0	0	0	0	0	0	0	
7	18:58:30Z	1573671510.3	99	12.3	0.38	0.009	1.0	1.0	100-100	101-101	0.54	0.53	0	0	0.2	0	0.18	1.25	
8	19:03:30Z	1573671810.6	108	10.4	0.53	0.009	1.0	1.0	100-100	101-101	0.64	0.6	0.3	0	0.3	0	1.25	1.25	
9	19:08:30Z	1573672110.3	115	8.5	0.87	0.009	1.0	1.0	100-100	101-101	0.24	0.21	0.1	0	0.1	0	0	1.25	
10	19:13:30Z	1573672410.3	119	7.2	0.91	0.009	1.0	1.0	100-100	101-101	-0.04	0	0	0	0	0	0	0	
11	19:18:30Z	1573672710.3	123	5.8	0.84	0.009	1.0	1.0	100-100	101-101	-0.11	0	0	0	0	0	0	0	
12	19:23:30Z	1573673010.3	122	4.4	0.78	0.009	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
13	19:28:30Z	1573673310.3	119	3.0	0.71	0.009	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
14	19:33:30Z	1573673610.2	114	1.7	0.65	0.009	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
15	19:38:40Z	1573673920.6	116	0.3	0.58	0.008	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0.13	0	
16	19:43:30Z	1573674210.4	117	0	0.53	0.008	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0.03	0.01	
17	19:48:30Z	1573674510.4	118	0	0.48	0.008	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
18	19:53:30Z	1573674810.3	116	0	0.44	0.007	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
19	19:58:30Z	1573675110.4	116	0	0.38	0.007	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
20	20:03:30Z	1573675410.6	117	0	0.35	0.006	1.0	1.0	100-100	101-101	0.06	0.02	0	0	0	0	0.37	0.29	
21	20:08:30Z	1573675710.4	122	0	0.33	0.006	1.0	1.0	100-100	101-101	0.4	0.36	0	0	0	0	1.05	0.97	
22	20:13:30Z	1573676010.3	124	0	0.36	0.006	1.0	1.0	100-100	101-101	0.32	0.3	0	0	0	0	1.05	1.05	
23	20:18:30Z	1573676310.4	125	0	0.4	0.005	1.0	1.0	100-100	101-101	0.34	0.32	0	0	0	0	1.05	1.05	
24	20:23:30Z	1573676610.3	125	0	0.44	0.005	1.0	1.0	100-100	101-101	0.16	0.14	0	0	0	0	1.05	1.05	
25	20:28:30Z	1573676910.8	125	0	0.48	0.005	1.0	1.0	100-100	101-101	0.14	0.1	0	0	0	0	1.05	1.05	
26	20:33:30Z	1573677210.3	124	0	0.53	0.005	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0.19	
27	20:38:30Z	1573677510.4	123	0	0.5	0.005	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0.19	
28	20:43:30Z	1573677810.3	125	0	0.48	0.005	1.0	1.0	100-100	101-101	0.16	0.14	0	0	0	0	0.57	0.53	
29	20:48:30Z	1573678110.7	127	0	0.48	0.005	1.0	1.0	100-100	101-101	0.28	0.26	0	0	0	0	0.81	0.77	
30	20:53:30Z	1573678410.3	121	0	0.46	0.005	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
31	20:58:30Z	1573678710.3	110	0	0.43	0.005	1.0	1.0	100-100	101-101	0	0	0	0	0	0	0	0	
32	21:03:30Z	1573679010.7	111	0	0.41	0.005	1.0	1.0	80- 80	101-101	0.08	0	0	0	0	0	0.53	0	
Totals:																0.6	0.0	0.76	0.91

This table lists the main comparisons between the original case and the emulated case. In the target columns the last row shows that the original target was reduced from 100 to 80 because sport was over. As the variant continued with 101 just ignore that last timestep in the final judgement. The Insulin required is slightly lower as expected due to a slightly higher target. The SMB columns confirm that all of the original SMBs were disabled. The tempBasal now has to offset that lack of insulin around those timesteps and is otherwise slightly reduced because of reduced requirements. The totals row at the bottom gives an indication of the changes but the timing of the major differences is important. While the SMBs of 0.6U were delivered the basals added up to 0.12U in those 15 minutes giving a total of 0.72U. In the variant case the total basal insulin adds up to 0.31U in that short period. So in summary this difference is too small to justify a change in settings which worked well on that day and the next week will be different anyway because the body will be in a different condition. But the process could be repeated for the logfiles of the equivalent activities on other days.

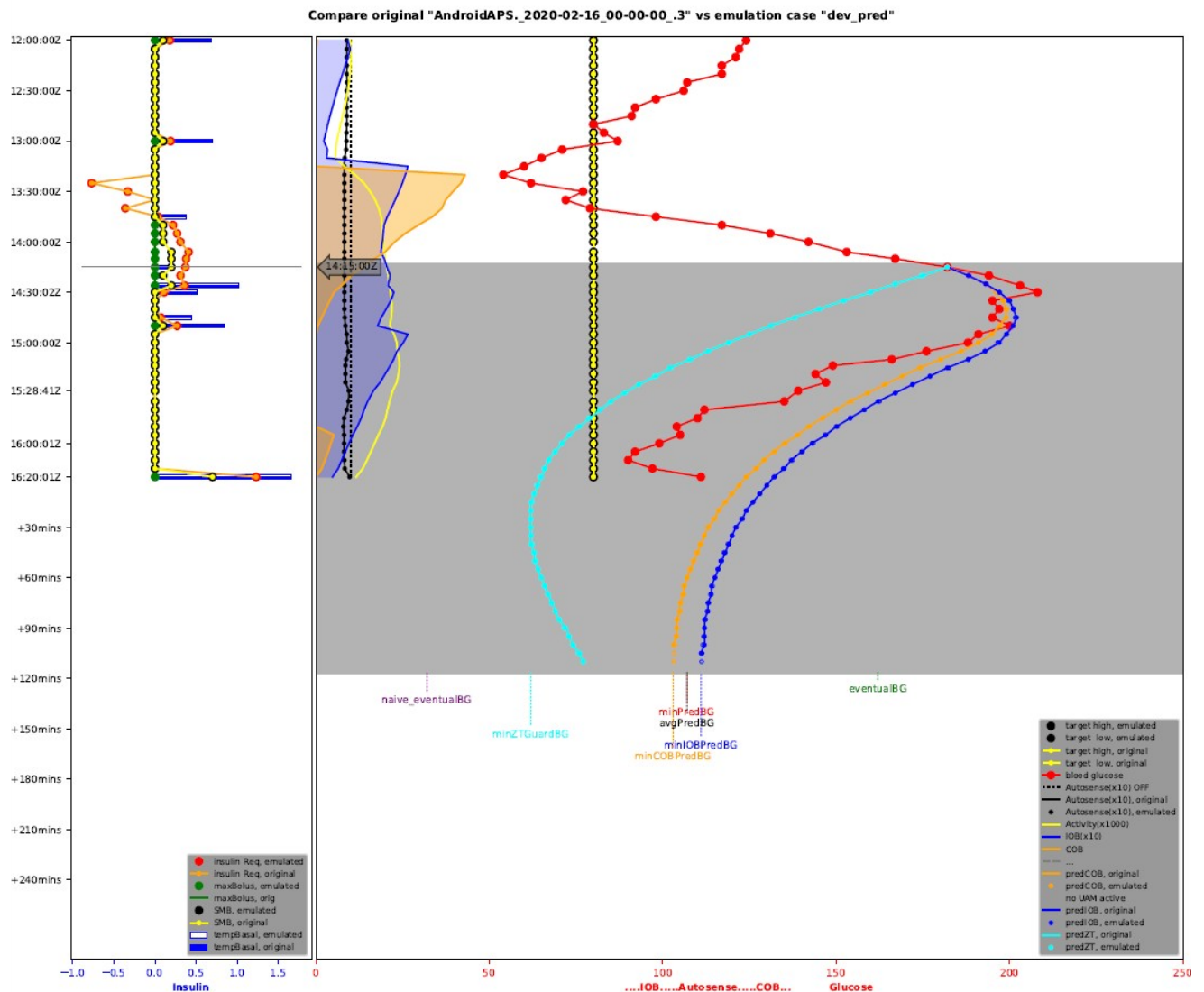
The graphical representation of the comparison tells the same story. The best evaluation method is to look at both, the graph and the table when assessing the comparison. The graph for the example looks like this:



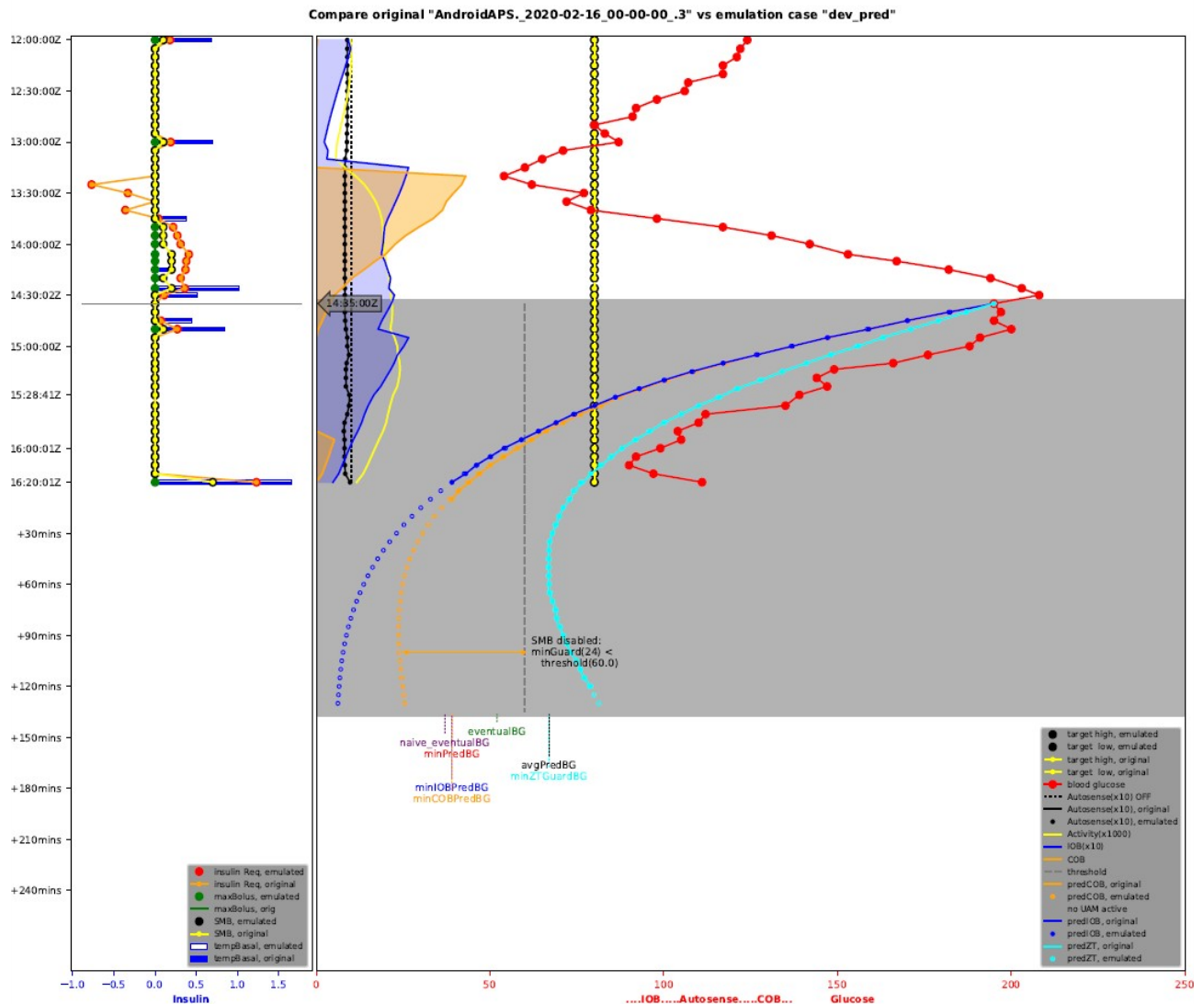
The „pred“ output option

It overlays the graph with the prediction lines known from the AAPS “Home“ tab. With the backward and forward arrows in the pdf-viewer you can get a slow motion animation of the history. The hope is that this will improve the understanding of how the loop works.

The example below is from a different logfile with no variation active and shows a moment when the predictions matched the future glucose quite well. This means the settings matched real life quite well for about one hour and that there was no outside disturbance like new carbs or target changes.



Some timesteps later the loop reports in its „SMB“ tab a reason why SMB was disabled. Such a situation is included in the graph whenever it applies to help understanding of the reasoning. What I find interesting is that the loop compares the predictions against the threshold rather than the lower target. That threshold is defined as 50% of the lower target plus 20mg/dl, i.e. always below even the lower target!

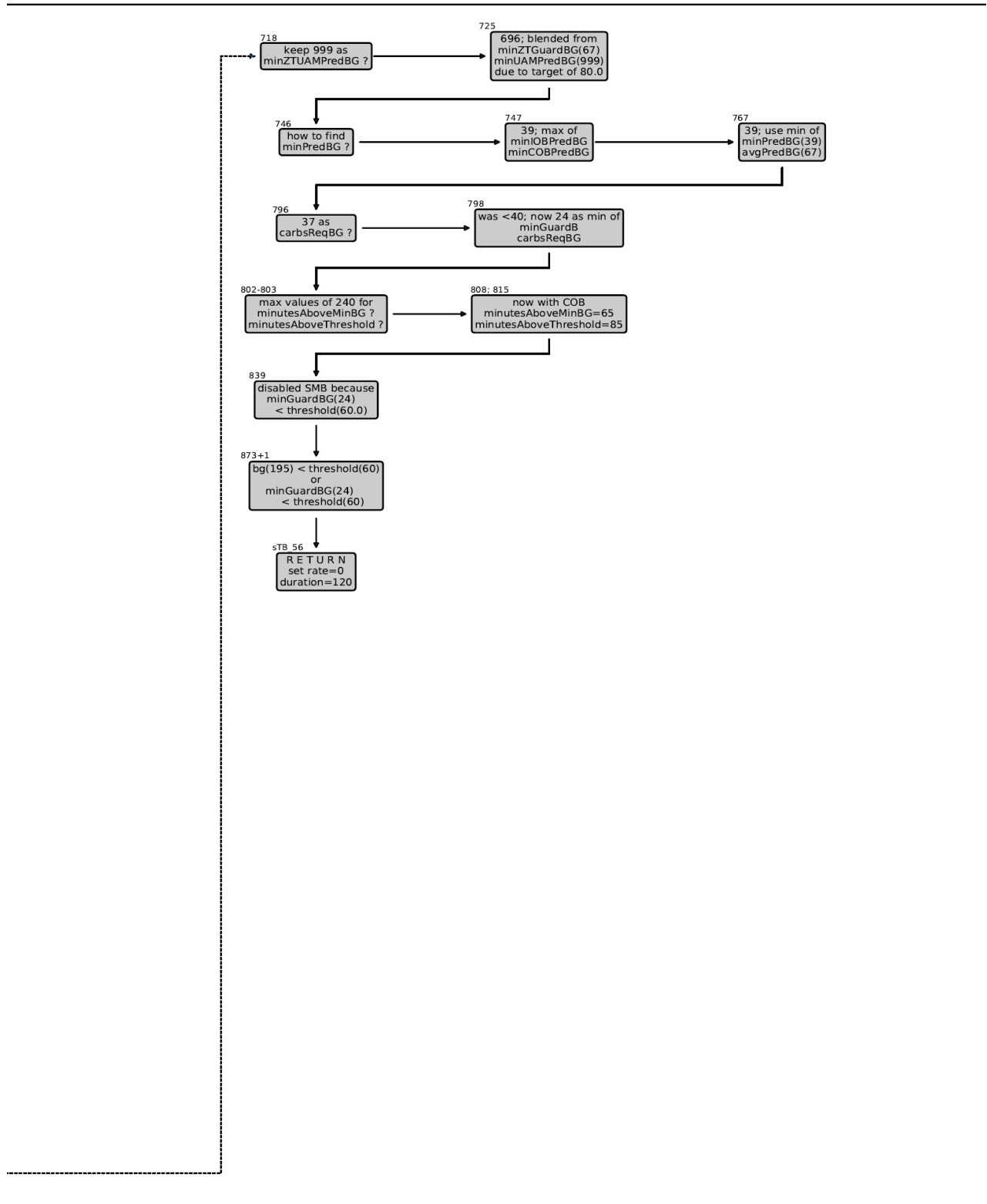


By the way, open circles in the predictions show the initial calculations which later get capped, truncated and rounded in AAPS and those final values are shown as filled circles.

The „flowchart“ output option

In order to better understand the logic and decisions in determine-basal I added this extra graphic. As for the „pred“ option a new flowchart is created at each time step. As an example here is a zoomed in region for the same timestep as before, i.e. the disabling of SMB at 14:35:00Z;

_00-00-00_3" vs emulation case "dev_pred"



Flowchart and decision logic at time 14:35:00Z

It shows the source of the decision in row 839² of the original determine-basal.js.

² This row numbering was valid in AAPS version 2.6 and not updated to version 2.7

Execute the analysis using the GUI front end

Start by clicking on the shortcut created on the desktop during the installation process. This opens a DOS window. The DOS window should remain empty but may be error messages will show up and I recommend to check for those before quitting the GUI. I tried to catch such messages in the code for display in the form itself but you never know.

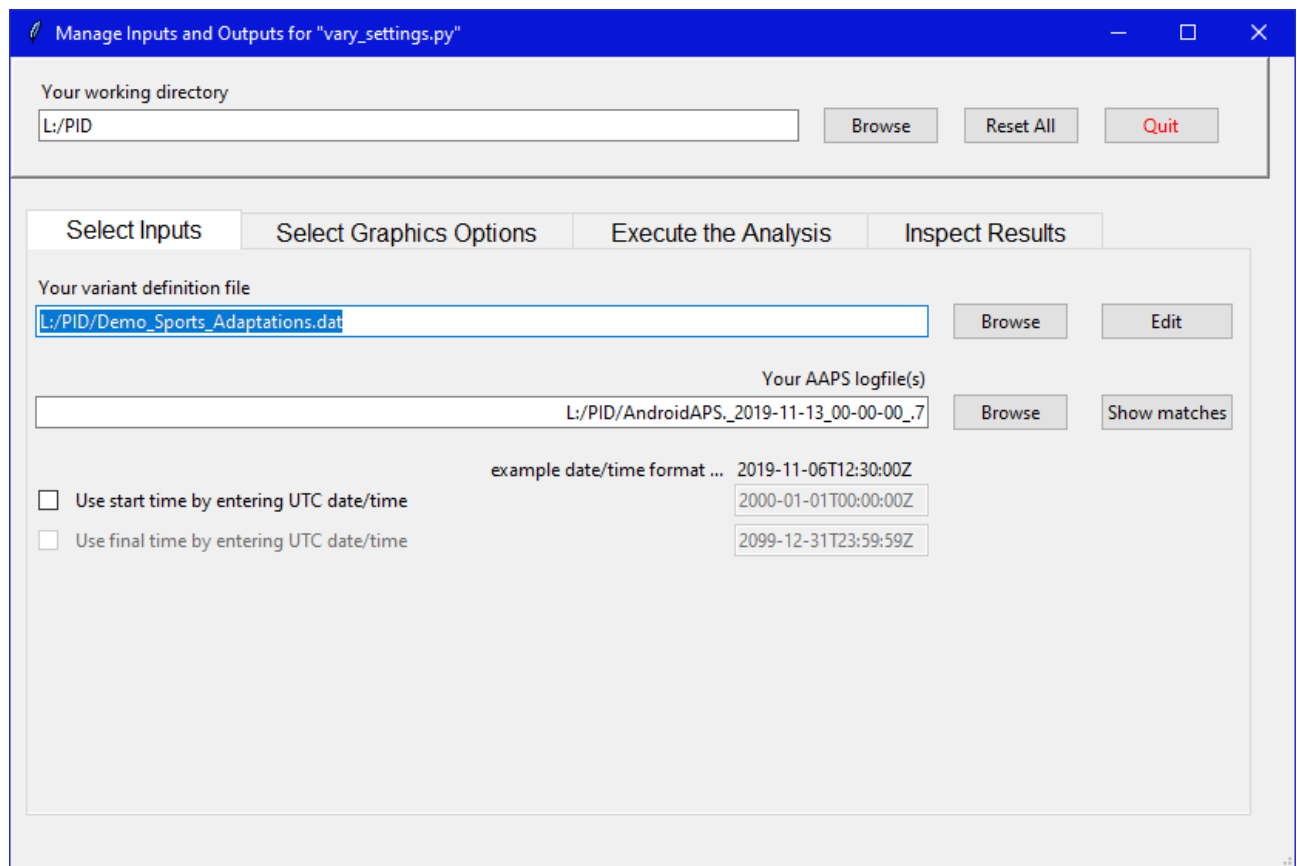
After opening that DOS window another window opens which contains a form itself to manage the whole process. The size of the window may be adjusted so longer filenames are displayed without scrolling. In general, boxes with white background are used to enter text, mostly filenames. Boxes with rounded corners and grey background are command buttons to start certain actions:

- Browse – will start the standard file selection dialog to find the respective file
- Edit - loads that file into your standard Editor (as long as your system has assigned a standard app for that file type)
- Show – similar to „Edit“ apart from the result PDF where it uses your standard PDF viewer

The process goes from top to bottom and from left to right but of course you can jump to where ever. First, you select the working folder, i.e. the one containing your logfiles, the variant definition file and later the result files. This input is not absolutely required but it preselects the folder for all later file browser actions.

Select inputs

Click the „Select Inputs“ tab if not already active. The two required inputs are the names of the variant definition file and the AAPS logfile(s).



If you need to scan several logfiles for longer time windows you edit the filename by using „*“ or „?“ in the suitable position of the filename. In such cases you can check which files will match the wild card specification by clicking the button „Show matches“.

For specifying an optional start date/time you activate the tick box. The time field will no longer be greyed out and you can enter/edit the start date and time. Above the field you see the format of how to specify the date and time. Remember AAPS uses the internal UTC format which is Greenwich time without daylight savings offset., e.g. in MESZ (Middle-European-SummerTime) it is numerically 2 hours behind. Normally I use copy and paste from that part of the logfile field and then do the detailed adjustments. If final time is also wanted, again tick the box, copy and paste from the start time and adapt it.

Clicking „Reset All“ will clear all input fields and return to the defaults.

Select graphics options

Click the „Select Graphics Options“ tab if not already active. This form selects what information to show in the graphic output and a table on screen. It uses radio buttons for the coarse selection and where appropriate shows tick boxes for details to be included or excluded. The resulting string as used in the DOS method is dynamically updated to reflect the tick actions but could also be edited manually.

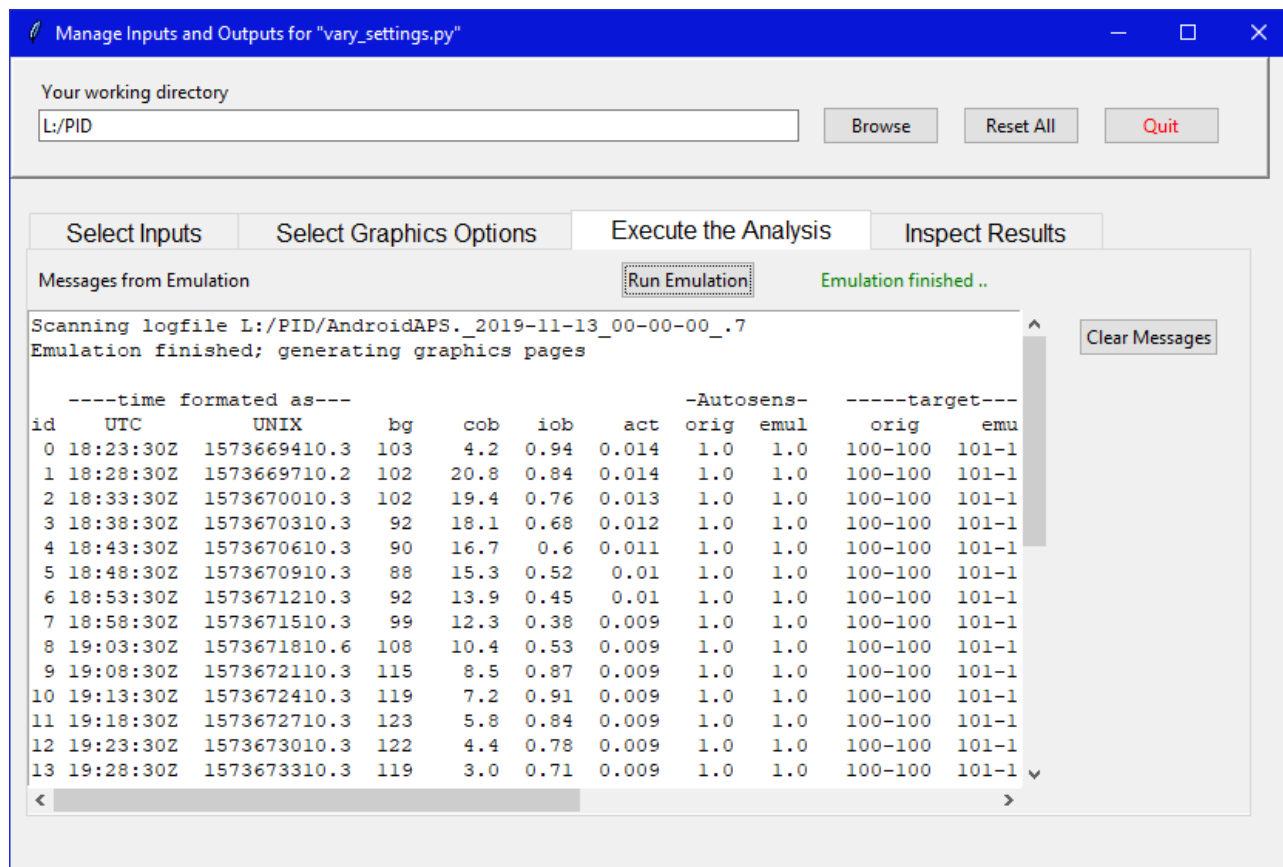
The initial selection is „most“ which is the most frequent option. In mode „most“ you tick those options you are not interested in.

In mode „just a few“ it is the opposite, i.e. you select those options you are specifically interested in.

In mode „All“ the fine grained selections in the bottom are disabled and hidden because they make no sense in that context.

Execute the analysis

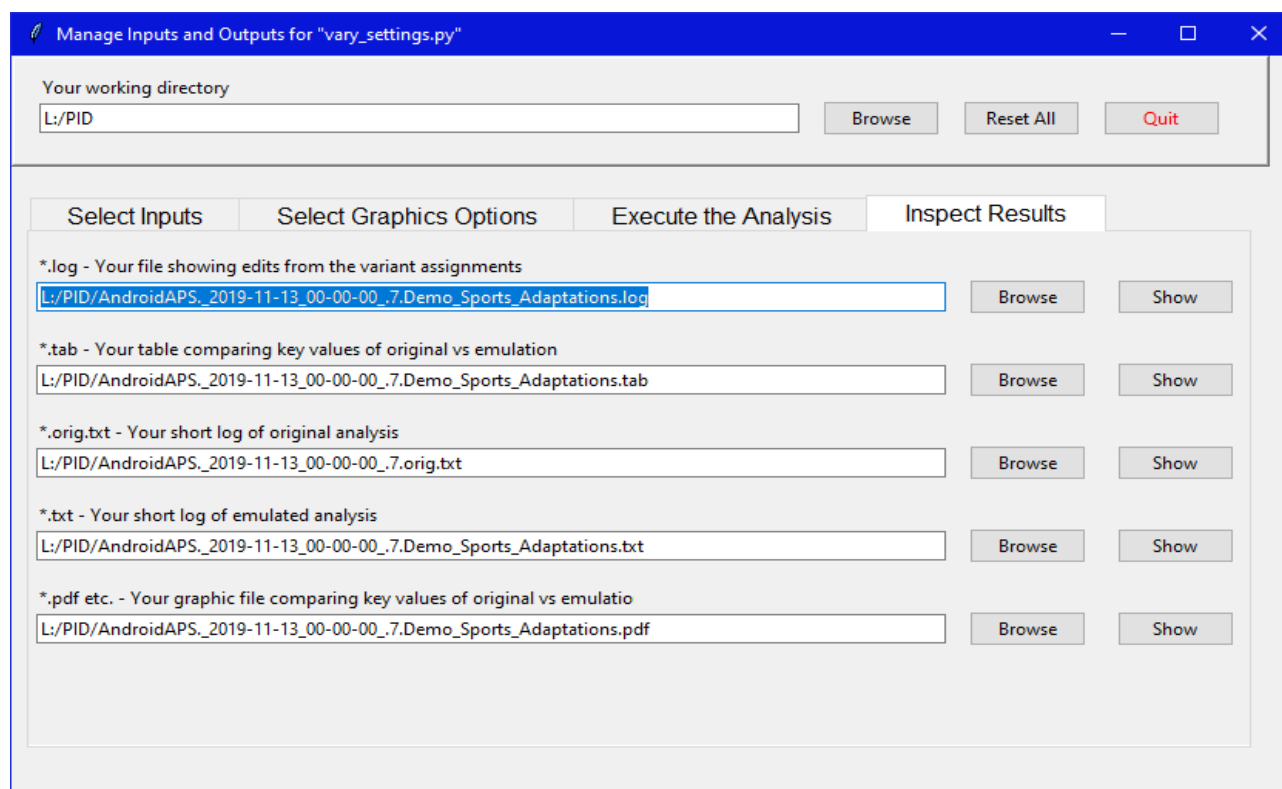
Click the „Execute the Analysis“ tab if not already active. The button „Run Emulation“ will kick off the analysis. The large white box shows messages otherwise displayed in the DOS window. Here it may make sense to adjust the window size to show more content without scrolling. The vertical resizing is limited to 30 lines so the window fits on a normal monitor.



Error messages or hints about missing inputs are displayed in red. The button „Clear Messages“ obviously will clear all contents in the message box. This is useful before starting the next analysis.

Inspect results

Click the „Inspect Results“ tab if not already active. This tab will also be activated automatically once the analysis is finished. Also the filenames are already prefilled after the analysis. However, you may use the „Browse“ buttons to select and see content of analyses run before.



Executing the analysis on the AAPS phone

Using the emulator on the phone is much easier than on Windows because only very few input parameters are required. These few are input via initial dialogs. Also no logfile input is needed because it always works with the currently active logfile. This way the emulator is effectively shadowing the active loop and you can see immediately whether the alternative settings would have delivered different SMBs or TBRs.

If SMB values are higher than in the master the user will be informed by a message from the speech synthesis. Thus you can carry the phone hidden away until alarmed and if so judge whether you want to apply the incremental SMB as an extra bolus manually.

As a further safety measure the logfile is scanned regularly for those „add'l carbs req“. If found for the last loop execution then again a message is output via the speech synthesis.

Start the emulation

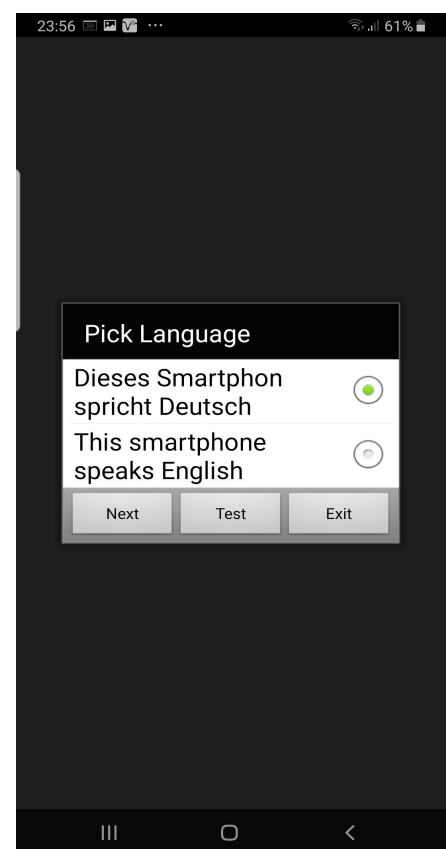
On the phone press the „QPython3L“ button created during installation. There, press Programs , select „vary_ISF_batch.py“ and finally select „Run“.

The first dialog is used to select the language for the speech synthesis. Why that? Initially I set it to German but sometimes the lady insisted on pronouncing everything in English. Occasionally she even switched in the middle of a session.

That meant I had to restart the script. I suspect those switches were triggered by xDrip+ when „Speak Readings“ was active.

Click „Test“ to listen to a sample speech synthesis.

Click „Next“ to proceed.



The second dialog window which opens is for picking all those items you want listed on the phone. You can see the initial default selection. Depending on the size of your display in landscape orientation you may not be able to fit everything into a single row per loop and may accordingly deselect less interesting columns.

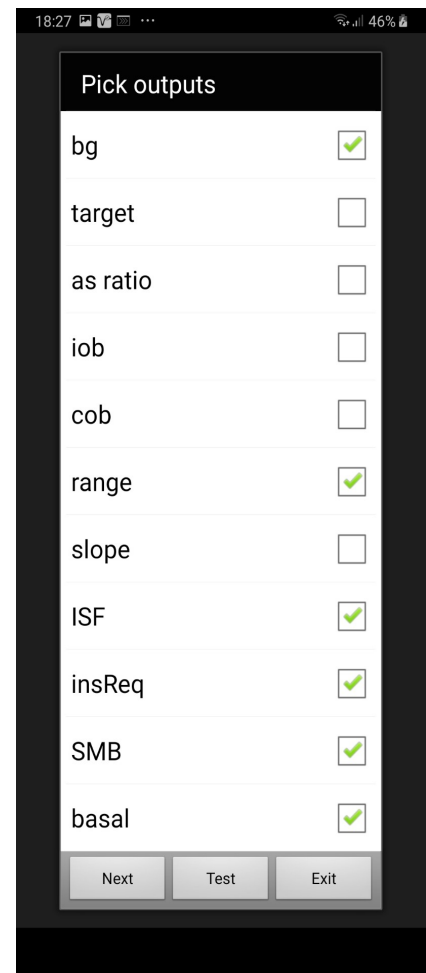
You may have to scroll the screen up and down to see all the options.

The initial default selection is suited to the autoISF³ setup and you most likely change those picks.

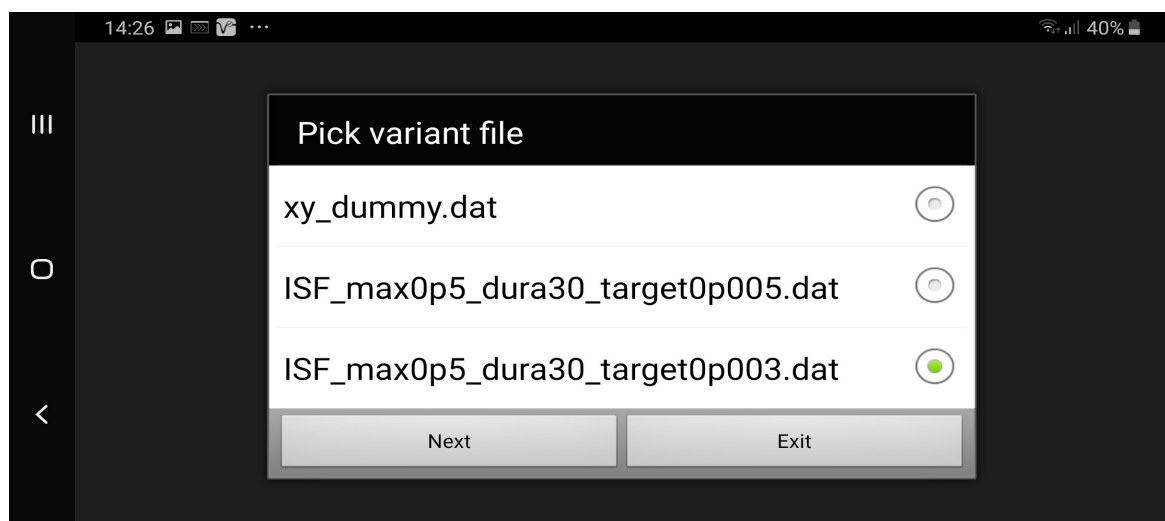
With all those columns ticked you need 92 characters per row.

Click „Test“ to generate a speech output announcing the required number of columns for the picks chosen.

Click „Next“ to proceed.



The last dialog window which opens is for picking your single variant definition file. This is a convenient time to rotate the phone to landscape to see nicely display longer filenames and to prepare for the result table display.



Click „Next“ to proceed.

3 See my branch autoISF

The results

The regular result files are saved in the logfile folder in case you need them. Please note that no graphics file is created as I could not find the matplotlib library for Android.

The main result of the Android version is the life table with key results. They cover the last hour or so. Older entries get deleted and newer ones get appended. This table is now also printed on Windows although not limited to max 14 rows.

No. 1														NEW	CTRL	TAKE WAKELOCK
UTC	bg	--5%	range-	--lin.fit-	-----ISFs-----	insulin	Req	---	SMB---	--tmpBasal--						
time	dura	avg.	dura	rate	orig	auto	emul	orig	emul	orig	emul	orig	emul	orig	emul	orig
08:37Z	104	20.0	104.0	10.0	-1.0	50	50.0	45.5	0	0	0	0	0	0	0	0
08:42Z	102	25.0	103.7	15.0	-1.2	50	50.0	44.5	0	0	0	0	0	0	0	0
08:47Z	106	30.0	104.0	0	0	50	50.0	43.5	0.32	0.32	0.2	0.2	1.04	1.04	1.04	1.04
08:47Z	111	0.0	108.5	0	0	50	50.0	50.0	0.56	0.54	0.3	0.3	1.52	1.48	1.52	1.48
08:52Z	105	0.0	105.0	0	0	50	50.0	50.0	0	0	0	0	0	0	0	0
08:57Z	101	5.0	103.0	10.0	-5.0	50	50.0	50.0	0	0	0	0	0	0	0	0
09:02Z	100	10.0	102.0	15.0	-3.7	55	55.0	52.6	0	0	0	0	0	0	0	0
09:07Z	95	0.0	95.0	20.0	-3.8	55	55.0	55.0	0	0	0	0	0	0	0	0
09:12Z	95	5.0	95.0	25.0	-3.2	55	55.0	55.0	0	0	0	0	0	0	0	0
09:17Z	96	20.0	97.4	10.0	0.5	55	55.0	51.3	0.02	0.02	0	0	0.54	0.54	0.54	0.54
09:22Z	93	15.0	94.8	0	0	55	55.0	52.6	0	0	0	0	0	0	0	0
09:27Z	86	0.0	86.0	10.0	-5.0	55	55.0	55.0	0	0	0	0	0	0	0	0
09:32Z	88	5.0	87.0	0	0	55	55.0	55.0	0	0	0	0	0.26	0.26	0.26	0.26
09:37Z	90	10.0	88.0	10.0	2.0	55	55.0	54.1	0.17	0.17	0.1	0.1	0.84	0.84	0.84	0.84

waiting 83 sec for next loop at 11:42



Sometimes when rotating the phone the column alignment was lost and I had to wait for the next loop update. The reason may be that on my phone the maximum width in portrait mode is 46, i.e. exactly half the above table width. Therefore I append a BLANK and rotating the phone is less annoying. If you have different screen dimensions and are annoyed let me know and I may consider adding a user specific config file.

Stop the emulation

Click the “Back“-button of the phone and then select „NO“, i.e. do not run it in the background.

Tipps and tricks

- If the keyboard overlays or pushes the result table off screen click the phones „Back“button
- If you rotate the phone between portrait and landscape orientation the column alignment may get screwed. This will be fixed upon the next loop execution and look nice again.
- If initially the table contains only a few lines then the script was started shortly after a new logfile was started. This fresh logfile covers less than 1 hour.
- The first column lists the loop execution times in UTC format. So for daylight savings time in central Europe this label is 2 hours behind your local time and that of the phone.
- Sometimes the qpython3-app disappears from the list of recent apps (on my phone there seems to be a limit of 4). In such cases I killed the app (Android Settings → Apps → Qpython3 → Force Stop) and restarted it.
- The message „add'l carbs req“ appears even although sufficient carbs were input in the bolus calculator. I included a filter to suppress it for smaller amunts and longer time spans. If the message still is output via the speech synthesis I check whether I really need fast carbs on top or whether the carbs on board are fast enough to keep me safe.

Status: 19-NOV-2020 @ 16:00

Appendix – Some more examples of VDF entries

This may serve as a collection of ideas and as proformas for your own studies.

Increase ISF for checking impact of higher insulin sensitivity

```
profile      sens      115          ### either define it directly, but fix
profile      sens      profile['sens']*1.10  ### alternatively increase it by 10%
```

Define CR as depending on basal rate like in circadian profiles

```
profile carb_ratio profile['current_basal']*20  ### 20 times basal rate
```

Switch version back from AAPS2.7 to AAPS 2.6.1 to check for different behaviour

```
new_parameter AAPS_Version "<2.7"  ### for a 2.7 original emulate what 2.6.1 would have done
```

Define TT reduction if BG change is at least 10mg/dl/5min

```
temp      delta10      glucose_status['delta']>=0  ### flag numeric value True:1, False:0
temp      lowerTT      -temp['delta10'] * 20        ### reduce TT by 0 or 20, respectively
profile    min_bg      profile['min_bg']+temp['lowerTT']
profile    max_bg      profile['min_bg']             ### same as lower target in TT case
profile    temptargetSet True                       ### just in case
```