Code Map and Program Guide for the HPZ Code Package

August 13, 2018

This guide is meant for programmers that wish to make alterations or adaptations to the code or add new features, preferences classes, functional forms, aggregators and etc. to the current code.

The First Part of the guide contains graphical illustrations of the code as a whole (code map), with additional literal explanations about how the code flow works.

The Second Part contains explanations about several modules that exist within the code that have a wide effect on various functions.

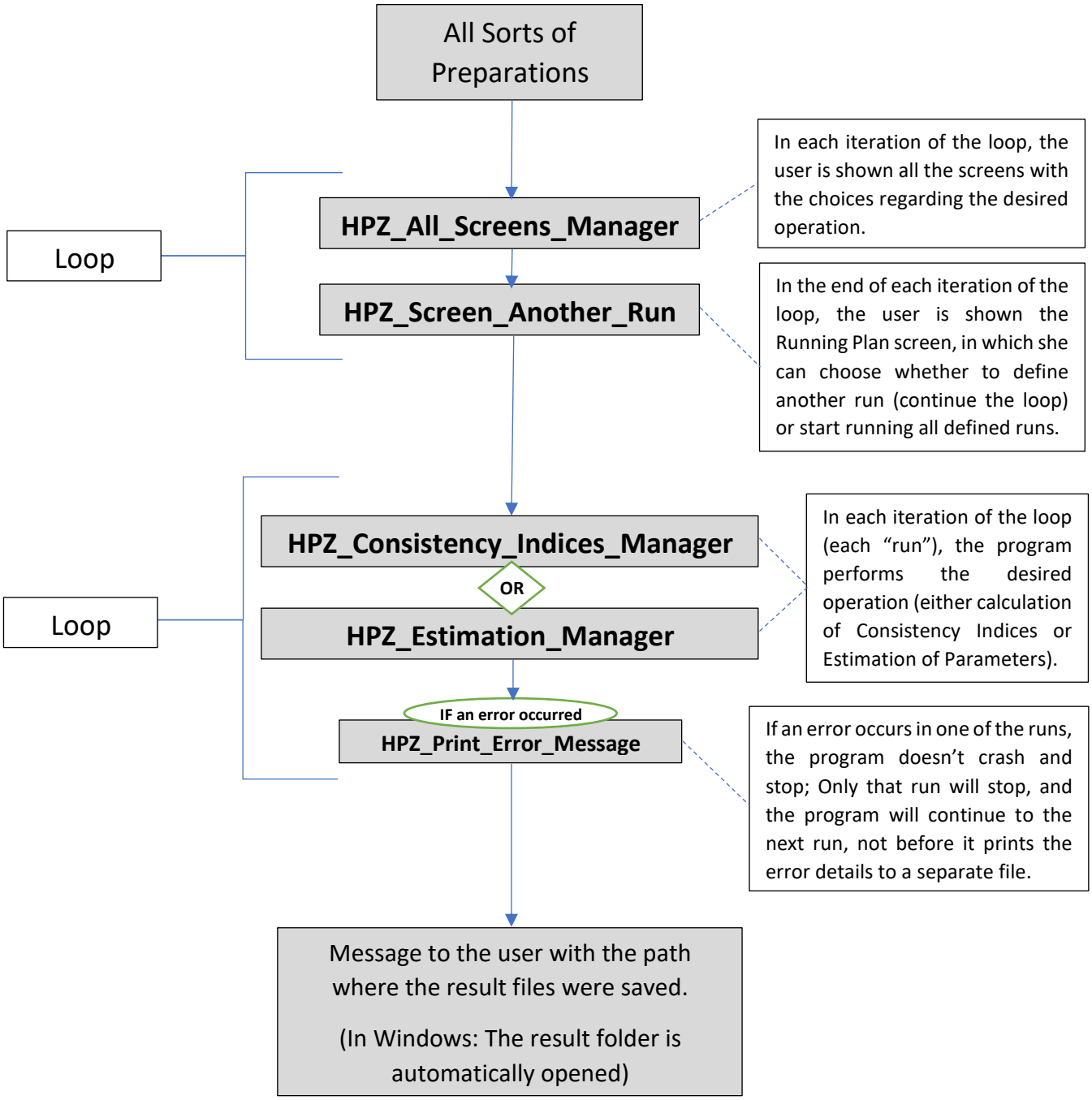
The Third Part of the guide gives step-by-step guidance regarding how to make specific changes to the code, e.g. - how to add a new preferences class or functional form, how to add a new method (in addition to NLLS, MMI and BI), and more.

Table of Contents

Part I : Code Map

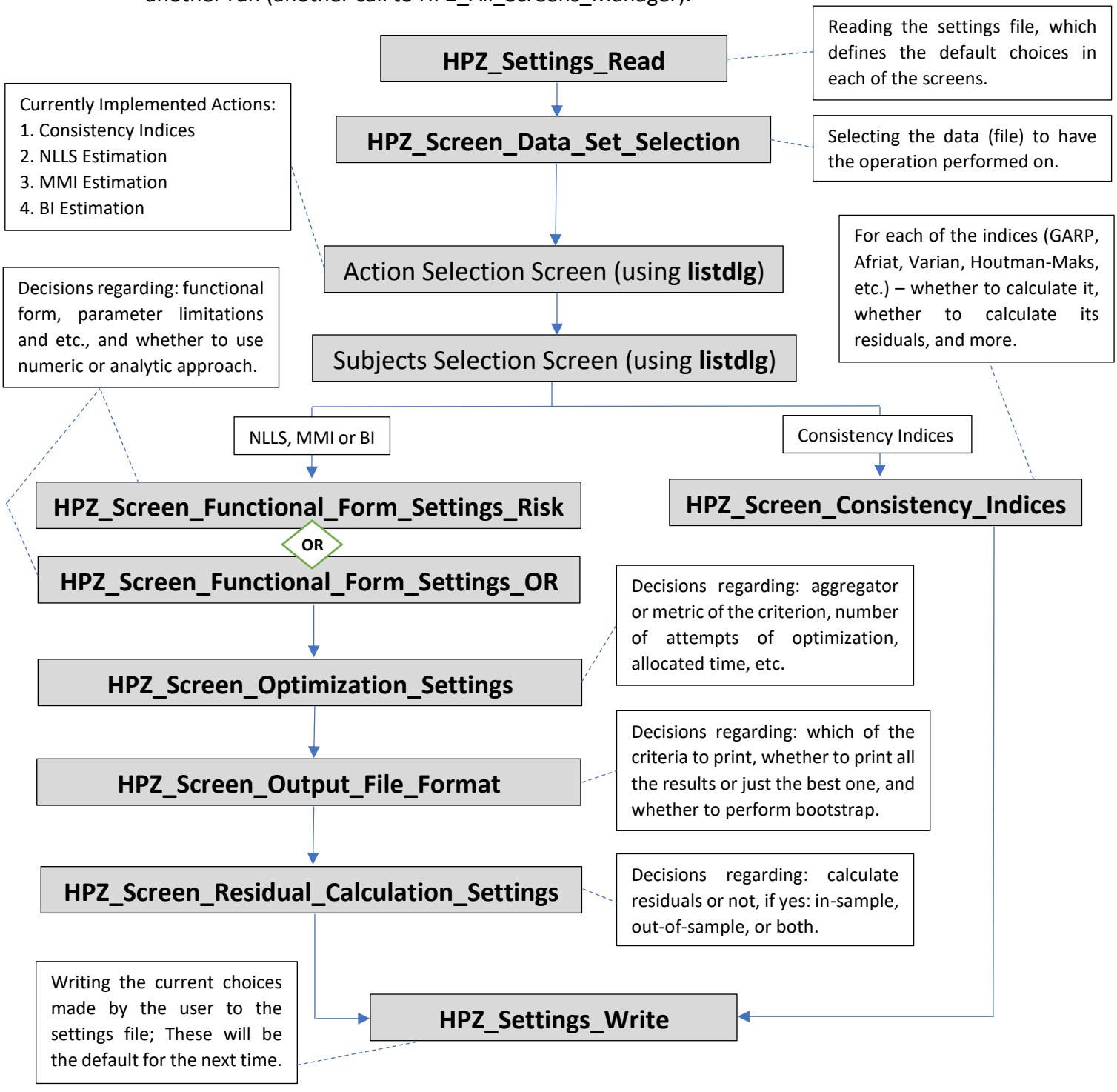
HPZ_Interface (Map)

HPZ_Interface is the main function of the program. When the user runs it, the program begins by showing the user a set of screens that opt him to choose the dataset, the subjects, the desired operation, and more specifications.



HPZ_All_Screens_Manager (Map)

HPZ_All_Screens_Manager is the main function that is responsible for the user interface screens. It shows the user the screens one after the other, in each screen the user makes her choices and moves on. The only screen that is not handled within this function is the Running Plan screen, in which the user chooses whether to define another run (another call to HPZ_All_Screens_Manager).

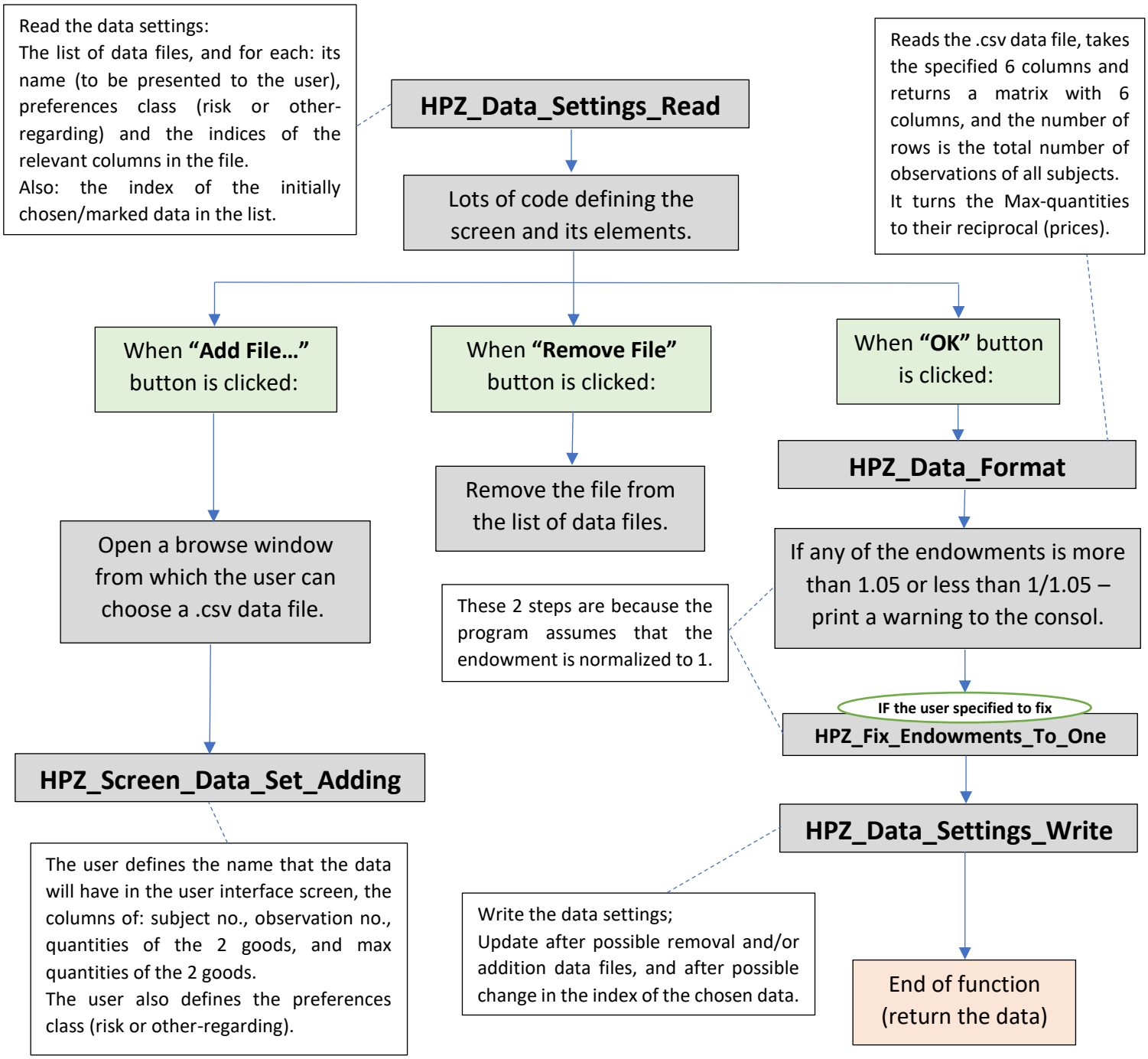


HPZ_Screen_Data_Set_Selection (Map)

HPZ_Screen_Data_Set_Selection creates and manages the first screen that is presented to the user, the screen in which the user chooses the data to have the operation performed on.

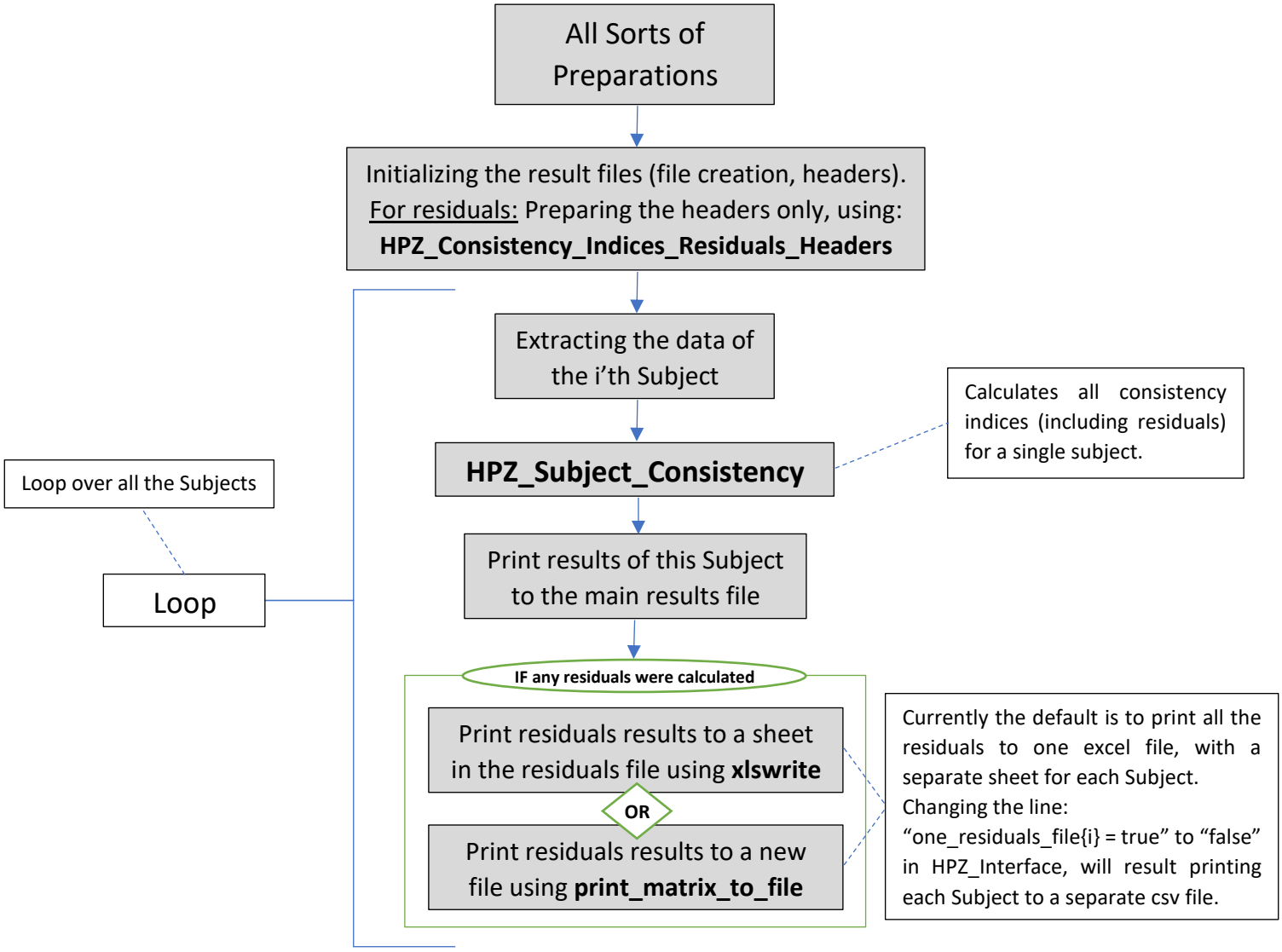
In addition to choosing a data file out of the existing list, the user can also remove a data from the list, or add a new data to the list. When adding new data, the user is allowed to browse for the data file, then the user is asked for information about the data (which columns contain the quantities, prices (their reciprocal) etc., and whether the data is of Risk Preferences or of Other-Regarding (OR) Preferences).

This screen also contains an option whether to automatically fix endowments to be precisely 1, or not.



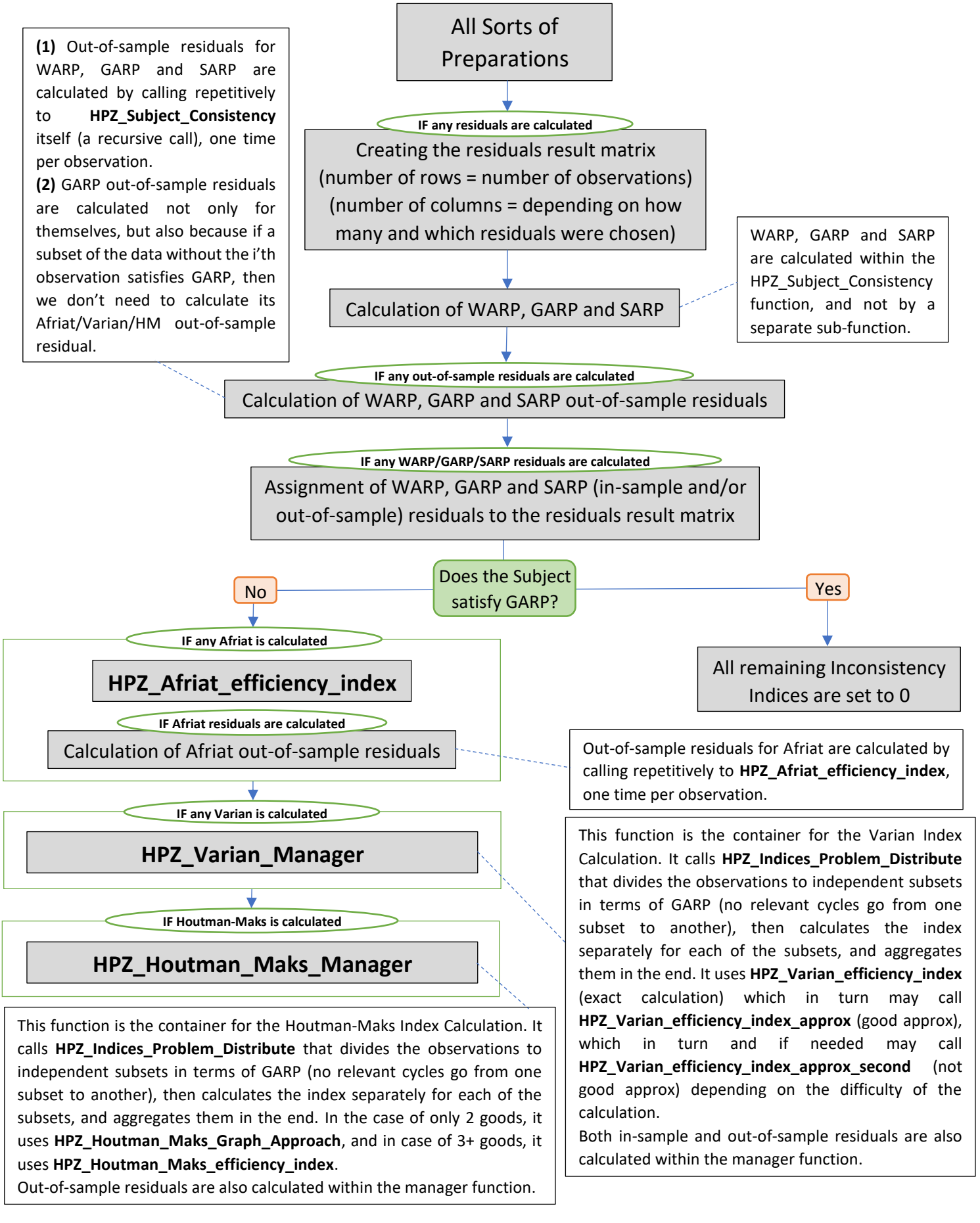
HPZ_Consistency_Indices_Manager (Map)

HPZ_Consistency_Indices_Manager is the head function that manages everything that has to do with the calculation of consistency and inconsistency indices, and their print to the results files.



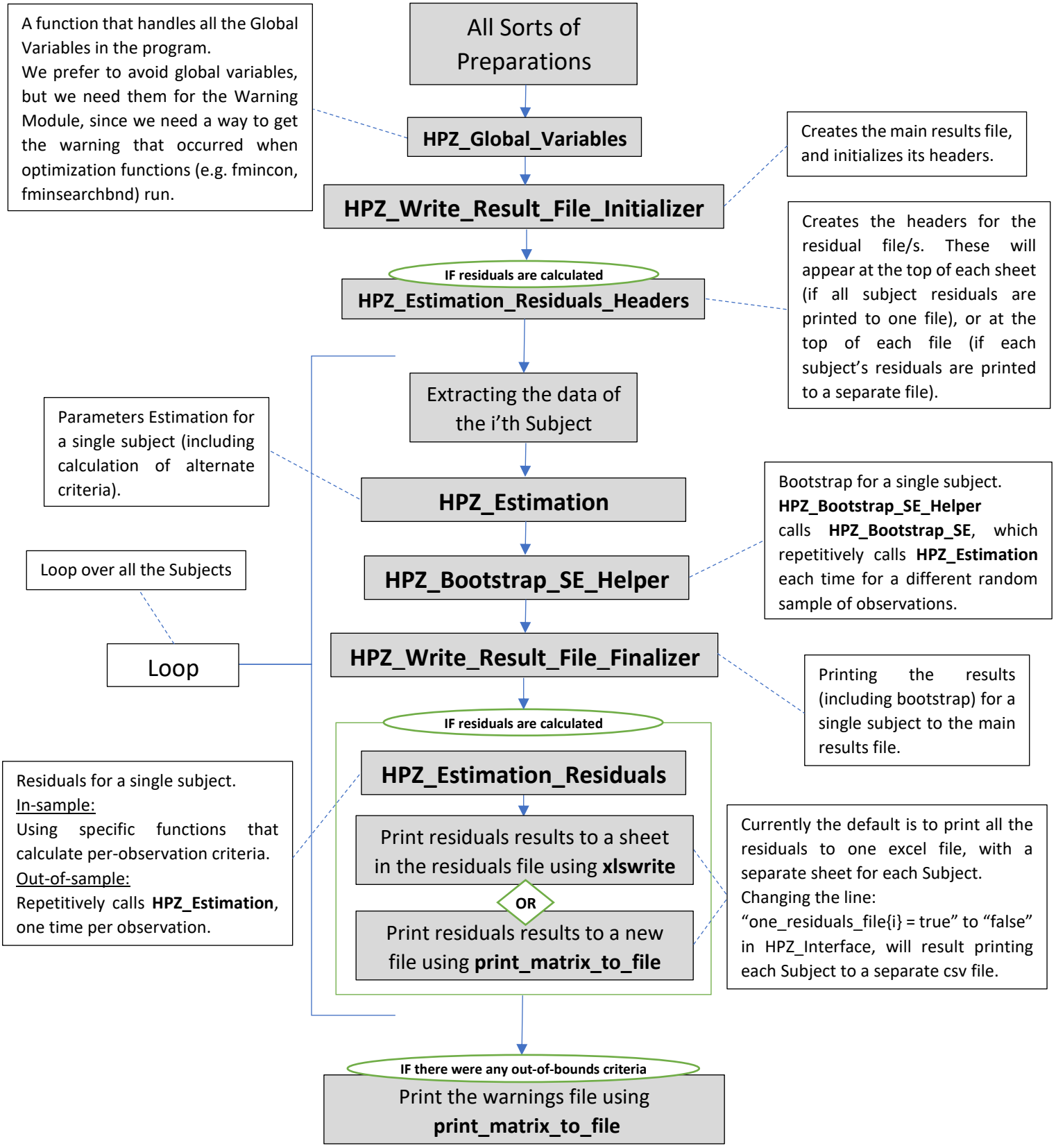
HPZ_Subject_Consistency (Map)

HPZ_Subject_Consistency manages the calculation of all the Consistency and Inconsistency Indices (including residuals) for a single, specific Subject.



HPZ_Estimation_Manager (Map)

HPZ_Estimation_Manager is the head function that manages everything that has to do with the estimation of parameters (both in Risk preferences and Other-Regarding preferences, by either the NLLS, MMI or BI criterions, with their various options), and printing the results (the parameters and the criteria) to the results files.

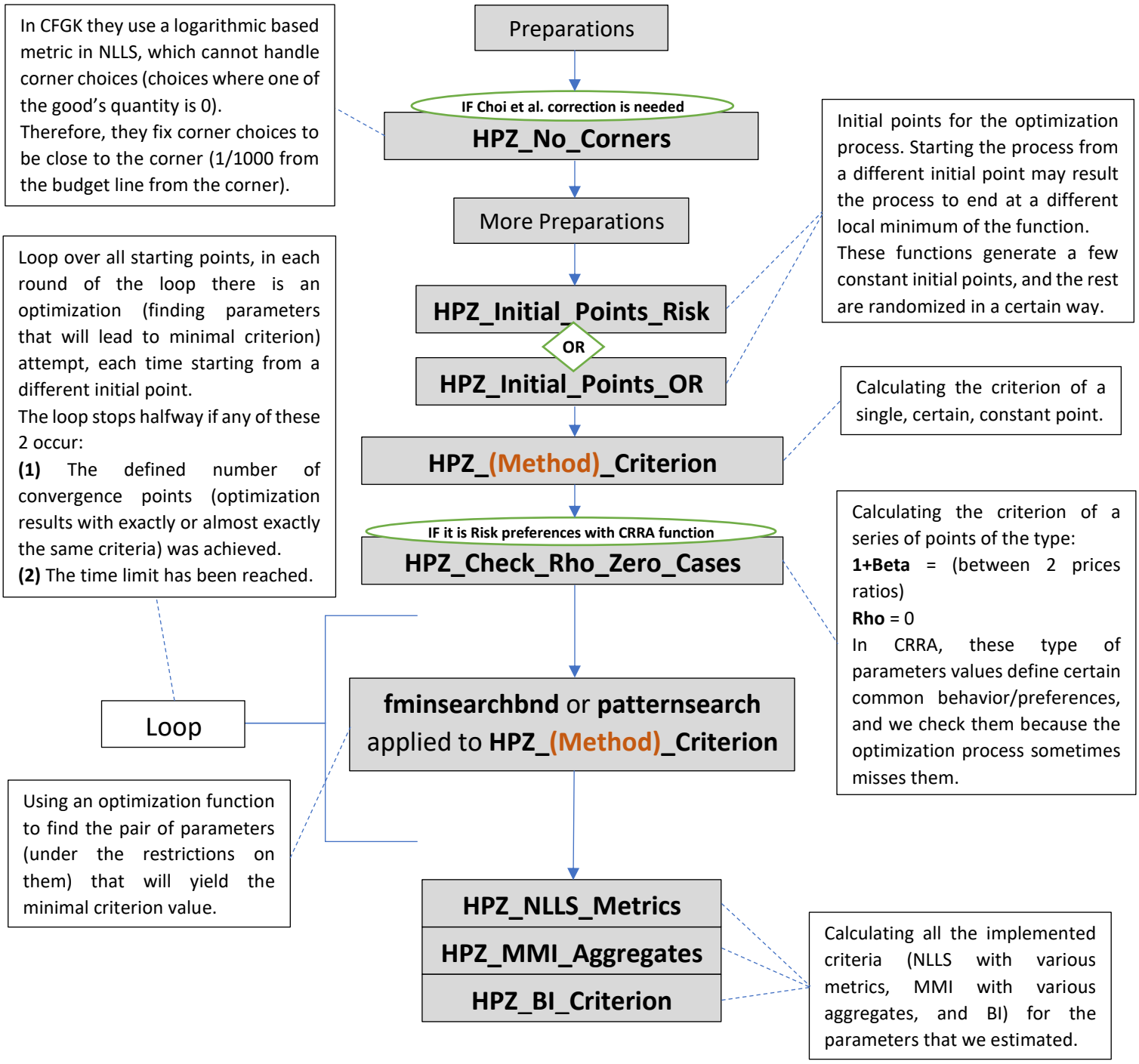


HPZ_Estimation (Map)

HPZ_Estimation manages the estimation of parameters and calculation of all criteria (not including bootstrap and residuals) for a single, specific Subject.

It works by calling an optimization function (such as fminsearchbnd or patternsearch) with a criterion calculating function as an argument, and the optimization function attempts to find the parameters that will minimize the criterion.

Note: **(Method)** is one of these:
NLLS – **Nonlinear Least Squares**. A distance-based (distance between bundles of goods) approach.
MMI – **Money Metric Index**. Finding the minimal expenditure required to get the same level of utility.
BI – **Binary Index**. Finding the number of observations in which the utility of the chosen bundle is strictly less than that of the optimal bundle.



Criterion Calculation

This part is an extension explanation of the functions of the type: **HPZ_(Method)_Criterion**.

The following is a summary description of the functions involved in the calculation of the criteria, for each one of the currently implemented methods (NLLS, MMI, BI).

NLLS

HPZ_NLLS_Criterion (returns the criterion with the desired metric)

Calls:

HPZ_NLLS_Metrics (returns the criterion based on all metrics (Euclidean, CFGK))

First Calls:

HPZ_NLLS_Choices_Numeric (uses: HPZ_Risk_Utility_Helper, HPZ_OR_Utility_Helper)

OR

HPZ_NLLS_Choices_Analytic

Later Calls:

HPZ_NLLS_Criterion_Euclid (returns a per-observation Euclidean criterion)

OR

HPZ_NLLS_Criterion_Ldr (returns a per-observation CFGK criterion)

MMI

HPZ_MMI_Criterion (returns the criterion with the desired aggregate)

Calls:

HPZ_MMI_Aggregates (returns all aggregates (Max, Mean, AVGSSQ))

Calls:

HPZ_MMI_Criterion_Per_Observation (returns a per-observation criterion, pre-aggregation)

Calls:

HPZ_MMI_Numeric (uses: HPZ_MMI_Grid_Search, HPZ_Risk_Utility, HPZ_Risk_Log_Utility, HPZ_Risk_Utility_Constraint, HPZ_OR_Utility, HPZ_OR_Utility_Constraint)

OR

HPZ_MMI_Analytic (uses: HPZ_Risk_Utility, HPZ_Risk_Log_Utility, HPZ_OR_Utility)

OR

HPZ_MMI_Semi_Numeric (performs a binary search using: HPZ_NLLS_Choices_Analytic)

BI

HPZ_BI_Criterion (returns the BI criterion with mean aggregate)

Calls:

HPZ_MMI_Criterion_Per_Observation (See above)

It then checks which observations have a criterion of 0 – those will be set to 0, and which observations have a criterion > 0 – those will be set to 1.

OR

HPZ_BI_Semi_Numeric (uses: HPZ_NLLS_Choices_Analytic)

After calculating the optimal choice, it uses either HPZ_Risk_Log_Utility or HPZ_OR_Utility to calculate both the utility of the chosen bundle and that of the optimal choice.

It then checks in which observations the utility of the chosen bundle is not less than that of the optimal bundle – those will be set to 0, and the rest will be set to 1.

Part II – General Modules

Constants

In order to avoid cases that a programmer will change the value of a certain variable (a variable that is not set by the user through a user interface screen) in one place in the code but forgets to do so in another - resulting many possible problems - we do our best to use constants instead of numerical values.

All the constants are defined in the HPZ_Constants classdef in the "Others" folder.

The constants can be divided into 3 types:

1. **Arbitrary Numbers** that are used as flags. E.g. instead of writing:
   if (action_flag == 3)
   We will write:
   if (action_flag == HPZ_Constants.MMI_action)
   Sticking to this approach will make sure that when a new action is added, and say it is not added in the end but in the middle - no harm will be done.
2. **String Constants**. Most of these are used to communicate with the user (Through prints to the consol, warnings, error messages, waitbars, message boxes, user interface screens).
   Note that some of these strings are used in the Data Settings Module. such strings should not be changed (if changed, you may need to change accordingly the values in the data settings file, or its name).
3. **Numeric Constants** that are used in various parts of the code, e.g. the number of digits after the point to be printed, the required significance level when performing bootstrap, thresholds to be used in different circumstances, etc.

Variables Documentation

The code contains many variables that are passed and received in multiple functions. Those variables hold the exact same name and meaning in all those multiple functions, therefore documenting an explanation to these variables in each of these functions would be a bad approach - both because it would be tedious for the reader, and also because it will be hard to maintenance (programmers will change the documentation in some functions and will forget to do so in others).

For exactly this purpose we have the "HPZ_Variables_Documentation" file. This file contains the documentation for (almost) each of the multiple-appearing variables in the code, while in the beginning of the functions where this variable is present, there is only a succinct reference to this file.

The Settings Module

All the choices that the user makes in the user-interface screens are automatically saved for the next time.

The Settings Module is composed of 2 segments:

1. The Data Settings Module
2. The User Interface Settings Module

The Data Settings Module

This module handles saving the list of datasets that will appear in the first screen shown to the user (function HPZ_Screen_Data_Set_Selection). It saves and reloads for each dataset: its name on the list, the file's path (file name included), the corresponding preferences class, and the numbers of columns in the file that hold the required information (subject no., observation no., the 2 chosen quantities and the maximum quantities (the reciprocal of the prices), and also the index of the dataset selected by default.

This module is completely handled within the HPZ_Screen_Data_Set_Selection function, and is handled by the 3 functions inside the "Data Files Settings" folder.

The changes to the list are saved immediately after the user presses the "OK" button of the data selection screen.

The User Interface Settings Module

This module handles saving all the other choices of the user, both in the first screen and in the rest of them (almost all functions with the prefix HPZ_Screen). For example, it saves the user's choice to perform MMI estimation based on the Mean aggregator, so the next time these choices will be the default ones.

This module is handled mainly within the HPZ_All_Screens_Manager function, and is handled by the 3 functions inside the "Screen Settings" folder. But the module requires the "cooperation" of the other HPZ_Screen functions to operate, that is - these functions must be programmed to accept their own settings as input, and finally return them as output to be saved later.

All changes to the default settings are saved only after the user finishes all the regular screens and reaches the "Running Plan" screen (HPZ_Screen_Another_Run - the screen where the user chooses whether to have the program start running or whether she wants to define another run).

The Warnings Module

Warnings File

When performing parameter estimation, the program checks whether the resulting criteria (per observation) make sense, that is: not -Inf, Inf or NaN, and in the case of MMI – also not greater than 1 and not smaller than 0. This check is performed every time a criterion is calculated, and not only for the final resulting criterion (also, it is checked for every per-observation criterion, before aggregation).

The program sums the number of each of these 5 options (-Inf, Inf, NaN, <0, >1) for each subject, and when the running ends, it prints a warning file next to the results file, in which each line represents a subject, each column one of these 5 options, and the value in each square is the number of times such occurrence happened during the estimation of this subject. If there were no problems at all, the warnings file will not be printed.

Debugger Mode

In the beginning of the HPZ_Constants function, there is the statement: "debugger_mode = false". Changing it to "true" will not affect the printing of the warnings file, but it will result comprehensive and detailed warnings to be printed to the consol while running, whenever the criterion of an observation is off bounds.

The advantages of the debugger mode are:

(1) It prints the warnings immediately, so if there is a general problem in the code or in the data file, you will find out immediately rather than only when running will end.
(2) The warnings are very detailed: you get to see all the variables that were used in the calculation that resulted in an out-of-bounds criterion as result, which makes it easier to understand the source of the problem.

The Waitbar Module

While the program runs, a waitbar is presented to the user for her convenience.

There are several options for the waitbars. Which of these options will be used is determined by the variables "**general_waitbar_options**", "**bootstrap_waitbar**" and "**residuals_waitbar**" in the beginning of the **HPZ_Interface** function, if you wish to change it.

When attempting to make changes to the way the waitbars work or presented, it is recommended to make the changes in all mentioned below functions, in order to keep the program consistent. Also, some changes can be made by simply changing some values in HPZ_Constants.

Option 1 : smart waitbar (default)

If number of subjects is 10 or more – use a single waitbar. If there are 2–10 subjects, and the run contains either residuals or bootstrap that will have its own waitbar, also use a single waitbar. Otherwise, use a per subject waitbar.

Option 2 : per subject waitbar

For every subject a waitbar appears with the subject number, that shows the progress of the calculation or estimation for this subject. When a process with a repetitive nature of calculation or estimation occurs (Varian out-of-sample residuals, Estimation out-of-sample residuals and Estimation bootstrap) – another waitbar appears.

The waitbar is being invoked in the following functions:

HPZ_Estimation (1 waitbar), **HPZ_Estimation_Residuals** (1 waitbar) and **HPZ_Bootstrap_SE** (1 waitbar). In addition, constants and strings that relate to the waitbars are held in the **HPZ_Constants** classdef.

Option 3 : a single waitbar

A single waitbar appears for the all subjects together. The advancement of the process is determined as: (number of subjects calculated) / (total number of subjects)

The waitbar is being invoked in the following functions:

HPZ_Consistency_Indices_Manager (1 waitbar),

HPZ_Estimation_Manager (1 waitbar).

The values of **bootstrap_waitbar** and of **residuals_waitbar** are only relevant when a single waitbar is used; depending on their values (true/false), another waitbar may appear to show the progress of bootstrap or residuals per subject.

Option 4 : no waitbar

No waitbar is presented. This option might be convenient if you add your own code with your own waitbars or other type of sign to the advancement of the process, and do not need the currently defined waitbars.

Part III – How to make certain additions to the code

General Considerations when making changes and additions

Whenever you make a change to the code or adding things, you should consider whether to make changes or update the following:

- **HPZ_Constants** – if you need new constants for the change, or to change the values of constants, it is a better and a more recommended approach to do it through this file.
- **HPZ_Variables_Documentation** – if you created new variables, or added new possible values to existing variables, you should update the documentation of these variables in this file.
- **The Settings Module** – if you created a new screen with options to the user, and you want the user's choices to be saved for the next time, you should add it to the settings module, which would require some changes in the files: HPZ_Settings_Read, HPZ_Settings_Reset, HPZ_Settings_Write and HPZ_All_Screens_Manager.
  Some changes to the existing screens may also require updating the settings module as needed.
  Also, some other changes in the code may also require changes in the settings module.
- **The Waitbar Module** – if you add a new function or a new function segment, that have or may have a long running time, you should consider adding it a waitbar (that will be activated only if the "per-subject waitbar" option is activated).
- **Possible Calculation Problems** – whenever you make changes to calculations or add new calculations, especially ones with powers (^) included, you should try to see whether the order of calculation is such that may not lead to Inf/Inf or to 0/0 and such for some values of the variables (matlab rounds ~10^{300} to Inf and ~10^{-300} to 0).
  You should also check for which values of the variables the resulting variables will be biased or Inf and such, and may want to limit the variables' values by some thresholds.

A list of very simple changes that can be done from the code

Requires a change in HPZ_Constants:

(1) **Debugger Mode**
Change the value of "debugger_mode" from "false" to "true", to activate the debugger mode, which will cause printing very specific and detailed warnings while the run, mainly regarding unreasonable criterion values.

(2) **Precision of printing of results**
Change the value of "print_precision" (currently set to 20) to control number of significant digits that are printed in the results files (not recommended).

(3) **BI threshold**
Change the value of "BI_threshold" (currently set to 10^{-5}) to control the MMI criterion result that will be considered as equal to 0 for the BI criterion.

(4) **MMI threshold (for the warning module)**
Change the value of "MMI_threshold" (currently set to 10^{-5}) to control the MMI criterion deviation from [0,1] that will result a warning.

(5) **Maximal starting points**
Change the value of "max_starting_points_numeric/analytic" to control the maximal number of initial points = maximal number of optimization attempts for numeric approach and for analytic approach.

(6) **Possible values of number of convergence points**
Add values to the cell array "min_counter_values" for more number of convergence options, or delete values (make sure NOT to delete the current value saved by the Setting Module, will cause the program to crash).

(7) **Sample size (number of repetitions) of Bootstrap**
Change the values of "bootstrap_sample_size_numeric/analytic" to control the sample size of bootstrap, in numeric approach and in analytic approach.

(8) **Significance level of Bootstrap**
Change the value of "significance_level" to control the significance level of the confidence intervals calculated by the Bootstrap.

Requires a change in HPZ_Interface:

(1) **One residuals file or separate residuals files**
Change the value of "one_residuals_file{i}" from "true" to "false" to have the residuals for each subject to be printed to a separate CSV file.

(2) **Smart waitbar, Per-subject waitbar, single waitbar, or without any waitbar**
Choose which of the 4 lines that assign value to "waitbar_options" to keep working, and which 3 will be set as documentation ("%").

Add a new inconsistency index (in addition to Afriat, Varian, HM)

Note: These steps are normally what needs to be done, but depending on the index, you might need to do more things, or do some things somewhat differently.

For convenience we will refer the new inconsistency index's name as "**INDEX**".

Step I

Write a function named **HPZ_(INDEX)_efficiency_index** that its input is:

- Matrix of expenditures (cell i,j has the expenditure required to buy the bundle chosen in observation j given the prices in observation i),
- Matrix of identical choices (cell j,k has 1 if choices are identical, 0 otherwise),
- index_threshold (an arbitrary threshold we use).

It could be that one of these inputs will not be needed, or that another input is needed.

The function should calculate the new index and return it. If you want to implement in-sample residuals, you should implement it inside this function (you can look at HPZ_Varian_efficiency_index as reference).

If approximation is sometimes needed:

If calculation of the index may take too long, and you want the function to "give up" on exact calculation at some point and turn to an approximation:

Write a function named **HPZ_(INDEX)_efficiency_index_approx** that calculates the approximation, and have it called from the non-approx function when it decides to "give up".

In some rare cases (such as in Varian), there are 2 level of approximations: the first approximation is less time consuming than the exact calculation but, in some cases, still might take too much time, and the second approximation that always runs fast. In this case, the second approximation name should be **HPZ_(INDEX)_efficiency_index_approx_second**, and the (first) approx function should call it when it also fails.

Step II

Open **HPZ_Constants**, Ctrl-F and search for "Varian".

Where there is "waitbar_finish_VARIAN", you need to add "waitbar_finish_**(INDEX)**". If the order of calculation of the indices is such that INDEX is calculated last, then have it hold the value "1.0", and change the value of ""waitbar_finish_HOUTMAN" and maybe of some of the other values as well, to lower values.

The other places where "Varian" appears are in relation to residuals calculation. Varian out-of-sample residuals can take very long, so it has its own waitbar. If you plan on implementing out-of-sample residuals for the new index, and it may take long to compute, you should implement waitbar constants in a similar way as Varian is implemented.

Step III

Open **HPZ_Screen_Consistency_Indices**, there:

Add a parameter named **(INDEX)_flags** both as an input to the function and as an output (preferably after the other flags).

Add graphical elements for the new index. You can do it by copying the elements of another index, then changing their names and making adjustments. If necessary, increase the height of the screen so the new index will fit in.

Create functions for these new elements, so for example when the user will choose not to calculate the index at all, the option to calculate residuals will be disabled, and when the user chooses to calculate, the residuals option will be enabled, and the same with residuals and in-sample and out-of-sample.

In the function that is called when the "OK" button is clicked (pb_call), implement the assignment to the (INDEX)_flags in a similar way to the way it is implemented in the existing indices.

Step IV

Open **HPZ_All_Screens_Manager**, Ctrl-F and search for "VARIAN_flags". In each place where it appears, add (INDEX)_flags as an input and/or as output, as needed.

Open **HPZ_Interface**, Ctrl-F and search for "VARIAN_flags". In each place where it appears, add (INDEX)_flags as an input and/or as output, as needed.

Step V

Open **HPZ_Settings_Write**, Ctrl-F and search for "VARIAN_flags". In each place where it appears, implement a similar code using (INDEX)_flags. Don't forget to update the numbers of the lines in the documentation.

Open **HPZ_Settings_Reset**, Ctrl-F and search for "VARIAN". In each place where it appears, implement a similar code using for the new (INDEX). Don't forget to update the numbers of the lines in the documentation.

Open **HPZ_Settings_Read**, Ctrl-F and search for "VARIAN_flags". In each place where it appears, implement a similar code using (INDEX)_flags. Don't forget to update the numbers of the lines in the documentation.

Step VI

Open **HPZ_Variables_Documentation**, Ctrl-F and search for "Varian" (or "Houtman").

Add there a documentation similar to the documentation for the other indices.

Step VII

Open **HPZ_Consistency_Indices_Manager**, Ctrl-F and search for "VARIAN_flags".

In each place where it appears, implement a similar code using (INDEX)_flags.

Open **HPZ_Consistency_Indices_Residuals_Headers**, implement there the headers that will be printed to the residual file/s in case this index was chosen, in a similar way that the existing indices are implemented.

Step VIII

Open **HPZ_Subject_Consistency**, there:

Implement the new (INDEX) in a similar way to Afriat, Varian and Houtman-Maks.

If you have 2 approximations, you can look at Varian as reference;

If when calculation is exact it is possible to know the out-of-sample residuals without actually performing out-of-sample, you can look at Houtman-Maks as reference.

You should call **HPZ_(INDEX)_efficiency_index** to calculate the index and when calculating the out-of-sample residuals; but if the full index was calculated using an approximation, it is recommended that the out-of-sample will call immediately to that approximation, and won't try to perform exact calculation (See Varian as reference).

Add a new method (in addition to NLLS, MMI, BI)

Note 1: These steps are normally what needs to be done, but depending on the method, you might need to do more things, or do some things somewhat differently.

Note 2: When there are multiple aggregates or metrics, we instructed to create a separate new variable. You may also use the currently existing variables aggregation_flag and metric_flag, but it may (or may not) cause problems when you will want to make certain changes in the program in the future. **Also**, even if you decide to rely on the existing flags, you will need to create separate flags for the settings module (in the functions: HPZ_Settings_(Read/Reset/Write) and HPZ_All_Screens_Manager), as was done with "param1_restrictions" for example.

Step I

Create 2 functions that calculate the per-observation criterion:

(i) In numeric approach
(ii) In analytic approach

The functions' names should be, respectively:

HPZ_(Method)_Numeric and **HPZ_(Method)_Analytic**

You can use the fucntions HPZ_MMI_Numeric and HPZ_MMI_Analytic as reference and/or as a base for these functions.

If the calculation requires calculating the optimal choices gives the prices and the examined parameters, you can simply use the existing HPZ_NLLS_Choices_Numeric and HPZ_NLLS_Choices_Analytic functions.

Numeric approach

In the numeric approach, the function simply uses **fmincon** to find the optimal value. Since the numeric approach does not use an analytic solution to the problem, it only uses the utility, therefore it will call functions that calculate the utility (such as in HPZ_Risk_Utility), or functions that calculate a preserving transformation of the utility (such as in HPZ_Risk_Log_Utility)

Analytic approach

It could be that you will give up on the analytic approach, either because the analytic problem is too difficult to solve, or because you don't want to waste time on it.

If you decide to implement the analytic approach, you need to find the analytic solution for each case of the parameters. You will normally need 5-10 different cases to cover all the options.

Step II

Create a function that has "**numeric_flag**" as one of the input variables, and uses the 2 functions from **Step I** to calculate the per-observation criterion with the desired approach (numeric or analytic).

The function should also implement the **warnings module**, that is: check how many of the per-observation criteria are -Inf, Inf, NaN, or outside the range they should be in.

The function name should be: **HPZ_(Method)_Criterion_Per_Observation**

You can use the function HPZ_MMI_Criterion_Per_Observation as reference.

Step III

If there are a few aggregators or a few metrics:

Create a function named:

HPZ_(Method)_Aggregates or **HPZ_(Method)_Metrics**

That will use the function from **Step II**, and will calculate each of the aggregates based on the per-observation criteria.

Then create a function named:

HPZ_(Method)_Criterion

That will have "(method)_aggregation_flag" or "(method)_metric_flag" as one of the input variables, that will call the previous function and will return only the criterion with the desired aggregation / metric.

If there is only one aggregator or metric:

Create a function named:

HPZ_(Method)_Criterion

That will use the function from **Step II**, and will calculate the criterion based on the per-observation criteria.

Step IV

Open the **HPZ_Constants** file, there:

Look for the NLLS_action, MMI_action, etc. constants. Add next to them a new constant (with a different value) called: **(Method)_action**.

Look for the NLLS_action_name, MMI_action_name, etc. constants. Add next to them a new constant called: **(Method)_action_name**, with the text you want the user will see in the action selection screen. Also, add "HPZ_Constants.(Method)_action_name" to the **all_actions_names** constant.

Look for the NLLS_action_file_name, MMI_action_file_name, etc. constants. Add next to them a new constant called: **(Method)_action_file_name**, with the text you want to be printed as part of the file name when performing this method.

If there are a few aggregates / metrics, look for "euclidean_metric", and define their constants like the corresponding constants for the NLLS metrics and the MMI aggregates.

Step V

Open the **HPZ_Estimation** file, there:

Use ctrl-F to find all places in the code with "NLLS" (or with "MMI").

Make sure to add in each of these places the required addition:

- Adding an "elseif (action_flag == HPZ_Constants.(Method)_action)" in various places.
- Adding the function HPZ_(Method)_Criterion before the loop, and inside the loop (inside fminsearchbnd, patternsearch, or another optimization function).
- Adding the function HPZ_(Method)_Aggregates / HPZ_(Method)_Metrics / HPZ_(Method)_Criterion in the end of the function.
- And more.

The HPZ_Estimation (Map) above might be helpful at this step.

If there are a few aggregates / metrics – add it as a new input variable to the HPZ_Estimation function (named: **(Method)_aggregates** or something similar), then use it wherever you need. You should add a documentation of this variable to the **HPZ_Variables_Documentation** file.

Step VI

Open the **HPZ_Estimation_Residuals** file, there:

Use ctrl-F to find all places in the code with “NLLS” (or with “MMI”). In each of these places, make the required changes.

The in-sample residuals may require some thought, but the out-of-sample section should be almost completely copy-paste with minor changes.

If there are a few aggregates / metrics – add it as a new input variable to the HPZ_Estimation_Residuals function, then use it wherever you need. Since you already added this input variable to HPZ_Estimation (in Step V), you now need to add it to every place the HPZ_Estimation was called, even when the method is one of the old methods (you can simply pass “0”).

Open the **HPZ_Estimation_Residuals_Headers** file, there:

Where there is “if-elseif” statements regarding “action_flag”, add the option: “elseif (action_flag == HPZ_Constants.(Method)_action)”, and there add the header that will appear in the residual file for this method for each of its aggregates / metrics.

If there are a few aggregates / metrics – add it as a new input variable to the HPZ_Estimation_Residuals_Headers function.

Step VII

If there are a few aggregates / metrics – add it as a new input variable to the **HPZ_Bootstrap_SE** and to the **HPZ_Bootstrap_SE_Helper** functions.

Also make sure that when HPZ_Bootstrap_SE_Helper calls HPZ_Bootstrap_SE, it passes this variable, and that when HPZ_Bootstrap_SE calls HPZ_Estimation it passes this variable.

Step VIII

Open the **HPZ_Write_Result_File_Initializer** file, there:

Use ctrl-F to find all places in the code with “NLLS” (or with “MMI”).

In each of these places, make the required changes.

Step IX

Open the **HPZ_Screen_Output_File_Format** file, there:

In the "options" section, assuming you have n (n >= 1) aggregates / metrics, add those n values, and program them as the other values (NLLS euclidean, NLLS CFGK, MMI Max, etc.) are programmed. Note that the vectors: "output_file_config", "output_file_config_save" and "file_val_str", that are now of length 6, will become of length 6+n.

Using ctrl-F to find all places in the code with "NLLS" (or with "MMI"), may help you find all the places that require changes.

In order to easily insert the new elements to the screen, you should use the "current_bottom" variable, in the same way it is used already. You will also need to increase the screen's height.

If there are a few aggregates / metrics – add a new input variable, similar to the current "aggregation_flag" (for MMI) and "metric_flag" (for NLLS).

Open the **HPZ_Screen_Optimization_Settings** file, there:

If there are a few aggregators or a few metrics:

> Copy either the NLLS or the MMI design and code, and make the required changes.

If there is only one aggregator or metric:

> Copy the BI design and code, and make the required changes.

Using ctrl-F to find all places in the code with "NLLS" (or "MMI" or "BI"), may help you find all the places that require changes.

In order to easily insert the new elements to the screen, you should use the "current_bottom" variable, in the same way it is used already. You will also need to increase the screen's height.

If there are a few aggregates / metrics – add a new input and output variable, similar to the current "aggregation_flag" (for MMI) and "metric_flag" (for NLLS).

Step X

If there are a few aggregates / metrics – perform these changes in these 3 functions:

Open the **HPZ_Settings_Read** file, there:

Add a new output variable, similar to the current “aggregation_flag” (for MMI) and “metric_flag” (for NLLS).

Near the places where aggregation_flag (MMI) and metric_flag (NLLS) are being read, make a line to read the new flag. Make sure to update the documentation about the lines numbers.

Open the **HPZ_Settings_Reset** file, there:

Near the places where aggregation_flag (MMI) and metric_flag (NLLS) are being reset, make a line to reset the new flag. Make sure to update the documentation about the lines numbers.

Open the **User_Interface_Settings.csv** file, There: add a row in the corresponding place, and write “1” in its first column.

Open the **HPZ_Settings_Write** file, there:

Add a new input variable, similar to the current “aggregation_flag” (for MMI) and “metric_flag” (for NLLS).

Near the places where aggregation_flag (MMI) and metric_flag (NLLS) are being written, make a line to write the new flag. Make sure to update the documentation about the lines numbers.

Step XI

Open the **HPZ_All_Screens_Manager** file, there:

Use ctrl-F to find all places in the code with “NLLS” (or with “MMI”). In each of these places, make the required changes.

If there are a few aggregates / metrics – Use ctrl-F to find all places in the code with “aggregation_flag” (or with “metric_flag”), and add the new flag accordingly:

Add the new flag as a new input and/or output variable, as needed, to **HPZ_Settings_Read**, **HPZ_Settings_Write**, **HPZ_Screen_Output_File_Format,** and **HPZ_Screen_Optimization_Settings**.

Add this new aggregation/metric flag as output variable of HPZ_All_Screens_Manager itself.

Step XII

Open the **HPZ_Estimation_Manager** file, there:

In the very end of the function, where the warnings file is printed, choose whether in this method only -Inf, Inf and NaN values will cause warnings (like in NLLS), or also values greater than 1 and smaller than 0 (like in MMI).

If there are a few aggregates / metrics – Use ctrl-F to find all places in the code with "aggregation_flag" (or with "metric_flag"), and add the new flag accordingly:

Add the new flag as a new input variable, to **HPZ_Estimation_Residuals_Headers**, **HPZ_Bootstrap_SE_Helper**, **HPZ_Estimation** and **HPZ_Estimation_residuals**.

Step XIII

Open the **HPZ_Interface** file, there:

In the place where there is: "action_flag{i} == …" add the new action, in addition to NLLS, MMI and BI.

If there are a few aggregates / metrics – Use ctrl-F to find all places in the code with "aggregation_flag" (or with "metric_flag"), and add the new flag as needed, specifically in the **HPZ_All_Screens_Manager** and the **HPZ_Estimation_Manager** functions.

Add a new aggregate or metric to existing methods (NLLS, MMI)

Open the **HPZ _Constants** file, there:

Use ctrl-F to find all places in the code with “euclidean_metric”. Add the new NLLS metric or MMI aggregate as a new constant with a new different value.

Open the **HPZ_Variables_Documentation** file, there:

Use ctrl-F to find all places in the code with “metric_flag”. Add the new NLLS metric or MMI aggregate to the documentation.

Only in NLLS: Create a new function: **HPZ_NLLS_Criterion_(Metric),** which similarly to HPZ_NLLS_Criterion_Euclidean and HPZ_NLLS_Criterion_Ldr, will calculate the per-observation criterion according to the new metric.

Only in NLLS: Open the **HPZ_NLLS_Criterion_Per_Observation** file, there:

Where metric_flag is being used, add an “elseif” statement which will use the new previously defined function.

Open the **HPZ_NLLS_Metrics** (or **HPZ_MMI_Aggregates** for MMI) file, there:

Add the new metric/aggregate to the output of the function, and add the calculation and assignment to the new output variable in the end of the function.

Only in MMI: Open the **HPZ_MMI_Criterion** file, there:

Add the new aggregate to the output of the function, and add the calculation and assignment to the new output variable in the end of the function.

Add a new preferences class (in addition to Risk and OR)

Note: These steps are normally what needs to be done, but depending on the method, you might need to do more things, or do some things somewhat differently.

Step I

Open **HPZ_Constants**, use ctrl-F to find "pref_class". Add a constant named "**(Pref_Class)_pref**", and assign it a different value than the other preferences classes. Add one or more constants named "**(Functional_Form)_func**" for each of the functional form you wish to implement for this new preferences class.

Open **HPZ_Variables_Documentation**, update the documentation of the constants you just added.

Step II

Create a new function named **HPZ_(Pref_Class)_Utility**.

This function should have the following as input: x (a vector of length 2, representing the quantities of the 2 goods), the utility function's parameters (e.g. beta, alpha, rho), and function_flag (the type of utility function, out of the utility functions of this preferences class).

If needed: Create a function named **HPZ_(Pref_Class)_Log_Utility** or if you used another transformation that is not log: **HPZ_(Pref_Class)_(Transformation)_Utility**.

"If needed" means if you realize that the original utility function will go too fast to 0 or to Inf or -Inf, or to another value, and therefore you prefer to use an order preserving transformation of the utility function.

Create a new function named **HPZ_(Pref_Class)_Utility_Constraint**, and implement it in a similar manner to HPZ_Risk_Utility_Constraint / HPZ_OR_Utility_Constraint. You will need this function in the numeric calculation of MMI.

Create a new function named **HPZ_(Pref_Class)_Utility_Helper**, and implement it in a similar manner to HPZ_Risk_Utility_Helper / HPZ_OR_Utility_Helper. This function's main purpose is to turn utility to minus utility, so if the optimization process (numeric) finds its minimum, it actually finds the maximum utility (needed in order to find the optimal choice, for NLLS).

Step III

Open **HPZ_Screen_Data_Set_Adding**, add another new option to the object "S.pref_class_rd" in addition to the already implemented pref classes. Also, in the "ok_button_call" function, add an "elseif" for the new pref class.

Create a new function named **HPZ_Screen_Functional_Form_Settings_(Pref_Class)**, and implement it in a similar way as HPZ_Screen_Functional_Form_Settings_Risk and HPZ_Screen_Functional_Form_Settings_OR are implemented.

Step IV

Open **HPZ_Settings_Read**, **HPZ_Settings_Reset** and **HPZ_Settings_Write**.

In each of these functions, you need to add 2 new variables that will be read / reset / written: "(Pref_Class)_param1_restrictions" and "(Pref_Class)_param2_restrictions".

Add them in a similar way to the already existing "risk_param1_restrictions", "risk_param2_restrictions", "OR_param1_restrictions", "OR_param2_restrictions", and make sure they are on the same line in all these 3 files.

Open **User_Interface_Settings.csv** and add there 2 lines (rows) in the right place, with values in the first 2 columns in each. The first value should be the default minimal value for each parameter (-HPZ_Constants.infinity = -1000000000 for no restriction from below) and the second value should be the default maximal value for each parameter (HPZ_Constants.infinity = 1000000000 for no restriction from above).

Step V

Open **HPZ_All_Screens_Manager**, look for an "if" statement involving "pref_class", add an "elseif" that calls the new HPZ_Screen_Functional_Form_Settings_(Pref_Class) when the new pref class was chosen. Remember that the output of the functional form settings screen in the all screens manager should be named "(Pref_Class)_param1_restrictions" and not just "param1_restrictions", and only afterwards they are made equal, because it is essential for the settings module.

Also, make sure to add the new param restriction variables to the HPZ_Settings_Read and HPZ_Settings_Write functions, in their input or output, as needed.

Try ctrl-F with "risk_param1_restrictions" and make sure in each place it appears, the new "(Pref_Class)_param1_restrictions" also appears, to make sure you didn't forget anything.

Step VI – NLLS

If you wish to implement the analytic approach, you should add the new pref class to the **HPZ_NLLS_Choices_Analytic** function, in a similar way to the other pref classes implemented there.

If you wish to implement the numeric approach, you should add the new pref class to the **HPZ_NLLS_Choices_Numeric** function, in a similar way to the other pref classes implemented there.

Step VII – MMI and BI

If you wish to implement the analytic approach, you should add the new pref class to the **HPZ_MMI_Analytic** function, in a similar way to the other pref classes implemented there.

If you wish to implement the numeric approach, you should add the new pref class to the **HPZ_MMI_Numeric** and **HPZ_MMI_Grid_Search** functions, in a similar way to the other pref classes implemented there.

Note: in these 2 the BI is based on the MMI, so there is no need for any change there.

If you implemented an analytic approach for NLLS, and wish to implement the semi-numeric approach for MMI and BI, you should add the new pref class to the **HPZ_MMI_Semi_Numeric** and **HPZ_BI_Semi_Numeric** functions, in a similar way to the other pref classes implemented there.

Note: if you implemented an analytic approach for MMI, it is very recommended to implement the semi-numeric approach as well, and run both on the same data in order to find potential bugs and computational issues.

Step VIII

Open **HPZ_Write_Result_File_Initializer** and **HPZ_Estimation_Residuals_Headers**, in the places in those functions where pref_class and/or function_flag are used, make the required changes and additions (these are needed for the headers in the results files).

Add a new functional form to preferences classes (Risk or OR)

In this section, we will refer to "**Risk**" or "**OR**" (depending on which of these is the preference class you wish to add a functional form to) by "**(Pref_Class)**".

Step I

Open **HPZ_Constants**, use ctrl-F to find "pref_class". Add a constant named "**(Functional_Form)_func**" for the new functional form you wish to implement, near the existing functional forms for the relevant pref class.

Open **HPZ_Variables_Documentation**, update the documentation of the constant you just added.

Step II

Open **HPZ_(Pref_Class)_Utility**, add there an "elseif" statement with a fitting implementation for the new function_flag.

If needed: Open **HPZ_(Pref_Class)_Log_Utility** or if there is another transformation that is not log: **HPZ_(Pref_Class)_(Transformation)_Utility** ("If needed" means if those functions exist), and add there an "elseif" statement with a fitting implementation for the new function_flag.

Note: It could be that there is no transformation utility function, but the new functional form does require such a function (or requires a different transformation). If there is such need, create a new function as needed, and make the required changes in **HPZ_(Pref_Class)_Utility_Constraint** and/or in **HPZ_(Pref_Class)_Utility_Helper**.

Step III

Open **HPZ_Screen_Functional_Form_Settings_(Pref_Class)**, and add there the new functional form. You need to add a new radio element (S.functional_form_rd) for it, and to add an "elseif" statement in the "ok_button_call" function.

Step IV

If the restrictions to the parameters in the new functional form are different than the restrictions in the existing functional forms, perform the following steps:

Open **HPZ_Settings_Read**, **HPZ_Settings_Reset** and **HPZ_Settings_Write**.

In each of these functions, you need to add 2 new variables that will be read / reset / written: "(Pref_Class)_(Functional_Form)_param1_restrictions" and "(Pref_Class)_(Functional_Form)_param2_restrictions".

Add them in a similar way to the already existing "risk_param1_restrictions", "risk_param2_restrictions", "OR_param1_restrictions", "OR_param2_restrictions", and make sure they are on the same line in all these 3 files.

Open **User_Interface_Settings.csv** and add there 2 lines (rows) in the right place, with values in the first 2 columns in each. The first value should be the default minimal value for each parameter (-HPZ_Constants.infinity = -1000000000 for no restriction from below) and the second value should be the default maximal value for each parameter (HPZ_Constants.infinity = 1000000000 for no restriction from above).

Open **HPZ_All_Screens_Manager**, and make sure that when HPZ_Screen_Functional_Form_Settings_(Pref_Class) is called, it returns both "(Pref_Class)_(Functional_Form)_(param1/param2)_restrictions" and "(Pref_Class)_(param1/param2)_restrictions", and immediately later the variables "param1/param2)_restrictions", should be assigned the restrictions according to the chosen function_flag.

Also, make sure to add the new param restriction variables to the HPZ_Settings_Read and HPZ_Settings_Write functions, in their input or output, as needed.

Try ctrl-F with "(Pref_Class)_param1_restrictions"and make sure in each place it appears, the new "(Pref_Class)_(Functional_Form)_(param1_restrictions" also appears, to make sure you didn't forget anything.

Step V – NLLS

If you wish to implement the analytic approach, you should add the new functional form to the **HPZ_NLLS_Choices_Analytic** function, in a similar way to the other functional forms implemented there.

If you wish to implement the numeric approach, you should add the new functional form to the **HPZ_NLLS_Choices_Numeric** function, in a similar way to the other functional forms implemented there.

Step VI – MMI and BI

If you wish to implement the analytic approach, you should add the new functional form to the **HPZ_MMI_Analytic** function, in a similar way to the other functional forms implemented there.

If you wish to implement the numeric approach, you should add the new functional form to the **HPZ_MMI_Numeric** and **HPZ_MMI_Grid_Search** functions, in a similar way to the other functional forms implemented there.

Note: the BI is based on the MMI, so there is no need for any change there.

Step VII

Open **HPZ_Write_Result_File_Initializer** and **HPZ_Estimation_Residuals_Headers**, in the places in those functions where the relevat pref_class is used and function_flag is also used, make the required changes and additions (these are needed for the headers in the results files).