ISO/IEC JTC1/SC29/WG1
(ITU-T SG21)

Coding of Still Pictures

JBIG
Joint Bi-level Image Experts Group

JPEG
Joint Photographic Experts Group

TITLE:	Overview of the JPEG AIC-3 Data Analysis Software
SOURCE:	WG1
EDITORS:	*Dietmar Saupe (dietmar.saupe@uni-konstanz.de)*
PROJECT:	ISO/IEC 29170 (JPEG AIC)
STATUS:	Final
REQUESTED ACTION:	Publication

Contact:
ISO/IEC JTC 1/SC 29/WG 1 Convener – Prof. Touradj Ebrahimi
EPFL/STI/IEL/GR-EB, Station 11, CH-1015 Lausanne, Switzerland
Tel: +41 21 693 2606, Fax: +41 21 693 7600, E-mail: Touradj.Ebrahimi@epfl.ch

Overview of the JPEG AIC-3 Data Analysis Software

ISO/IEC JTC 1/SC 29/WG1N101393

Dietmar Saupe

January 15, 2026

Abstract

ISO/IEC plans to publish the standard 29170-3 in 2026 with the title *Information technology — JPEG AIC Assessment of image coding — Part 3: Subjective quality assessment of high-fidelity images*. This report presents a draft of the documentation for the corresponding *Reference Implementation* of the AIC-3 data analysis. It introduces the conceptual principles for the composition of the software which is split between several main programs that the user may needs to modify and run in sequence and a set of functions. All relevant data from input and from the computations are gathered in a few Matlab tables that are also explained here. Finally, all functions of the JPEG AIC-3 Reference Implementation are presented in this report as shown by the `help` function from the Matlab console. The online repository of the code is located at `gitlab.com/wg1/jpeg-aic/aic-3-data-analysis`.

Contents

1 Introduction

ISO/IEC plans to publish the standard 29170-3 in 2026 with the title *Information technology — JPEG AIC Assessment of image coding — Part 3: Subjective quality assessment of high-fidelity images*. This report presents a draft of the documentation for the corresponding *Reference Implementation* of the AIC-3 data analysis. The assessment of image coding proceeds by collecting responses to triplet comparisons of compressed images. The user interfaces and actual data collection by crowdsourcing are not part of the topics covered in this document. The appendix contains a rendering of the `readme` file. An example of an application of the software is included in the `gitlab` repository.

The data analysis has two main steps, followed by the presentation of the achieved solutions:

Dataset cleansing. First comes the dataset cleansing (`mainFilter.m`), producing a reliable subset of the collected data by identifying and removing assignments of subjects that contain responses that are not generated by the perceptual and cognitive processes under investigation.

Scale reconstruction. Second, for each source image with its compressed versions, the cleansed dataset of triplet responses is fitted with a unified functional model given by distortion-rate functions for plain and boosted triplet comparisons (`mainReconstruct.m`). The functional model implies probabilities for all possible responses to the triplet comparisons. The fitting procedure is based on minimization of the negative log-likelihood. In addition, the classic pointwise reconstruction and another functional model that minimizes the differences to the pointwise reconstructions in the least-squares sense is computed.

Reporting. The Matlab program `mainReport.m` reads the results of the reconstruction process and shows for each source image a figure with the distortion-rate curves from the plain and boosted triplet comparisons as well as the pointwise model.

This document assumes that the reader is already familiar with the JPEG AIC-3 standard. The descriptions of the programs and functions will be useful for understanding how to prepare an input data set for the analysis. Moreover, this information shall help researchers who need to modify and extended the implemented algorithms.

2 Core data structures

There are just a few data structures that comprise the essential information required for the data analysis. For convenience, they are implemented in the form of the Matlab tables `T, I, Q, A, X`, generated by the data input and data cleansing carried out by `mainFilter.m`. Internally, a subset of columns of `X` is converted to arrays for efficient computation of the negative log-likelihoods during optimization which consumes the greatest part of the CPU time.

2.1 T (table) – Raw response data read from csv files

The raw data from the assessment of perceived image distortion shall be contained in the two csv files `ptc.csv` and `btc.csv`, having one row for each response and the following column names. The function `csvImport` reads the files and joins the rows in the table `T`. The starred columns are required, the others optional.

`assignment`* (string) – Unique identifier for each assignment given to a worker

`worker` (integer) – Unique anonymized identifier for each worker who participated in this study

`mode`* (string) – Method used for image comparisons (BTC or PTC)

`batch` (integer) – Unique identifier for each batch within the study

`question_id`* (integer) – Unique identifier for each question presented to the workers

`source`* (string) – Unique acronym for the source image

`codec_left`* (string) – Unique acronym for the codec used for compression of the left image in the triplet

`codec_right`* (string) – Unique acronym for the codec used for compression of the right image in the triplet

`dlevel_left`* (integer) – Distortion/compression level of the left image

`dlevel_right`* (integer) – Distortion/compression level of the right image

`img_left`* (string) – Filename of the left image

`img_pivot`* (string) – Filename of the pivot image

`img_right`* (string) – Filename of the right image

`is_same`* (logical, 0 or 1) – Boolean value indicating if the left and right image in the triplet comparison are compressed with the same codec (same-codec comparison)

`is_cross`* (logical, 0 or 1) – Boolean value indicating if the triplet is a cross-codec comparison. So either `is_same` or `is_cross` is true.

`is_trap`* (logical, 0 or 1) – Boolean value indicating if the triplet is a trap question

`question_order` (integer) – Order of the question in a batch presented to the worker

`response`* (string) – Worker's response to the question (either 'left', 'notsure', 'right', or 'skip')

`submission_time` (string) – Timestamp when the worker submitted the response (datetime format)

`response_time` (double) – Time taken by the worker to respond, in seconds

`reload_count` (integer) – Number of times the worker reloaded the page during the task

`resolution` (string) – Screen resolution of the worker's device during the task

`toggle_count` (integer) – Only for PTC: Number of toggles between the source and test images

2.2 I (table) – Image names and properties

`name` (string) – Filename of compressed image

`mode` (categorical) – Mode PTC or BTC

`questions` (integer) – Number of questions (and responses) that include the compressed image as a stimulus

`source` (categorical) – Acronym of the source image for the compressed image

`codec` (categorical) – Acronym of the codec used for the compressed image

`dlevel` (integer) – Distortion level of the compressed image

`bitrate` (double) – Bitrate in bpp for the compressed image

`cvvdp` (double) – CVVDP objective metric value for the compressed image

`rmse` (double) – RMSE metric value for the compressed image

`psnr` (double) – PSNR objective metric value for the compressed image

`ssim` (double) – SSIM objective metric value for the compressed image

2.3 Q (table) – Questions defining the triplet comparisons

`question_id` (integer) – Unique identifier for each question presented to the workers

`mode` (categorical) – Method used for image comparisons (BTC or PTC)

`type` (categorical) – Type of triplet comparisons (SAME or CROSS)

`source` (categorical) – Unique acronym for each source image used in the study

`codec_left` (categorical) – Unique acronym for the codec used for compression of the left image in the triplet

`codec_right` (categorical) – Unique acronym for the codec used for compression of the right image in the triplet

`dlevel_left` (integer) – Distortion/compression level of the left image

`dlevel_right` (integer) – Distortion/compression level of the right image

`img_left` (string) – Filename of the left image

`img_pivot` (string) – Filename of the pivot image (a source image

`img_right` (string) – Filename of the right image

2.4 A (table) – Assignments of study participants

`assignment` (categorical) – Unique identifier for each assignment of a batch of triplet comparisons

`mode` (categorical) – Method used for image comparisons (BTC or PTC)

`questions` (integer) – Number of comparisons in the assignment

`consistency` (double) – Consistency in [0,1] of responses in the assignment

`accuracy` (double) – Accuracy in [0,1] of responses in the assignment

`lr_bias` (double) – Order bias in [-1,1] of responses in the assignment

`ns_bias` (integer) – Bias for responses 'notsure' in the assignment

`meanRT` (double) – Mean response time in seconds

`skipRatio` (double) – Ratio of skipped reponses

`otsu` (logical, 0 or 1) – Flag raised (1) for unreliable assignments according to thresholding the mean of accuracy and consistency

`biased` (logical, 0 or 1) – Flag raised (1) for assignments having large order bias

`subset` (logical, 0 or 1) – Flag (1) indicating an accepted, reliable assignment

`nll` (double) – Negative log-likelihood of responses in the iterative outlier removal

2.5 X (table) – Aggregated response data

`group` (double) – Number of group of equivalent questions (i.e., equal and symmetrically equal questions). Group numbers start with 1 and are consecutive.

`mode` (categorical) – Either PTC or BTC for plain and boosted triplet comparisons.

`type` (categorical) – Either SAME or CROSS for same-codec and cross-codec comparisons.

`source` (categorical) – Acronym for the unique source image in the group of questions.

`codec_left` (categorical) – Acronym for the unique codec for the unique left image in the group of questions. When the left image is the source image, then the value is `codec_left = "none"`.

`bitrate_left` (double) – Unique bitrate for the unique left image image in the group of questions. When the left image is the source image, then the value is `bitrate_left = inf`.

`dlevel_left` (integer) – Unique distortion level for the unique left image in the group of questions. When the left image is the source image, then the value is `dlevel_left = 0`.

`codec_right` (categorical) – Acronym for the unique codec for the unique right image in the group of questions. When the right image is the source image, then the value is `codec_right = "none"`.

`bitrate_right` (double) – Unique bitrate for the unique right image image in the group of questions. When the right image is the source image, then the value is `bitrate_right = inf`.

`dlevel_right` (integer) – Unique distortion level for the unique right image in the group of questions. When the right image is the source image, then the value is `dlevel_right = 0`.

`votes` (double) – Total number of responses for this group of questions.

`left` (double) – Number of responses `left` for this group of questions.

`notsure` (double) – Number of responses `notsure` for this group of questions.

`right` (double) – Number of responses `right` for this group of questions.

`img_left` (string) – Filename of the unique left image.

`img_right` (string) – Filename of the unique right image.

2.6 Result (struct) - Container for reconstruction results

The structure has the following fields:

`date` (string) – Date

`runtime` (double) – Runtime in seconds

`processor` (string) – Kind of machine processor

`config` (struct) – Parameters for the data resampling, i.e., the fractions of PTC data and of cross-codec comparisons, the budget of responses per source image, a flag that declares that responses `notsure` are split half and half between `left` and `right`, and the seed value for the random number generator

`function` (string) – Short name of functional model, e.g., 'exp'

`boosting` (integer) – 1,2,3 (degree of polynomial boosting function)

`optPar` (struct) – Structure of 12 fields showing chosen parameters of the optimizer for three kinds of models

`source` (string) – Acronym of source image

`codecs` (categorical) – Ordered list of K codecs (acronyms)

`pt_model` (table) – Table showing the pointwise model. i.e., for each compared image its file name, mode, codec, bitrate and native scale value

`pt_nllptc` (double) – Negative log-likelihood of the pointwise model for PTC

`pt_nllbtc` (double) – Negative log-likelihood of the pointwise model for BTC

`ls_model` (struct) – Functional model using least-squares

`ls_nll` (double) – Negative log-likelihood of the functional least-squares model

`ls_rmsd` (double) – RMS difference to pointwise reconstruction for least-squares functional model

`mle_model` (struct) – Functional model using MLE

`mle_nll` (double) – Negative log-likelihood of the functional MLE model

`mle_rmsd` (double) – RMS difference to pointwise reconstruction for MLE functional model

`p_nll` (double) – p-value based on G-test statistic

`p_l1` (double) – p-value based on l1-test statistic

`p_l2` (double) – p-value based on l2-test statistic

2.7 model (struct) – Structure defining a functional model

The functional models for all codecs applied to a given source image are represented together in a Matlab `struct`. They share the same type of DR-function and the same degree of polynomial boosting function. Other (informative) fields may later be added to indicate a name, group name, datetime, a unique id, data from which it was generated and the corresponding dataset. The fields of the `struct` are as follows.

`source` (categorical) – the source (acronym)

`codecs` (categorical) – Ordered list of K codecs (acronyms)

`fclass` (integer) – 1,2,...,6 (denoting the type of DR-function like exponential)

`bclass` (integer) – 1,2,3 (degree of polynomial boosting function)

`npar` (array) – 1x3 array, `npar(1)` is the number of parameters per DR-function, `npar(2) = bclass` is the number of parameters per boosting polynomial, `npar(3) = npar(1)+npar(2)` is the total number of parameters per codec

`fctpar` (array) – (K, L)-array with $L = 1, 2, 3$. It gives 1 to 3 parameters in one row for each of the K functional codec models without boosting.

`polpar` (array) – (K, L)-array with $L = 1, 2, 3$. It gives 1 to 3 parameters in one row for each of the K boosting polynomials.

3 Main programs

3.1 `mainFilter.m`

The program reads the input data from the triplet comparison assessments. Three files are expected: `ptc.csv, btc.csv, images.csv`. The data import generates the core data structures `T, I, Q` for the raw PTC and BTC response data (`T`), the image names and properties (`I`), and the questions defining the triplet responses (`Q`), respectively. See Section 2 for details, especially which columns are required in the input csv files.

The filtering proceeds by applying the JPEG AIC-3 dateset cleansing procedures at the assignment level. Assignments are classified as valid by Otsu thresholding the mean of accuracy and consistency, as well as requiring order bias being less or equal to 0.5. The table `A` is generated that shows the results and details of the thresholding. Finally, in the master table `T`, a columm `is_valid` is introduced that indicates the responses from those assignments that have passed the filtering. Note that up to this point, the filtering does not remove any data, it only flags assignments in `A` and responses in `T` that have successfully passed the filtering procedure.

Finally, the filtering is executed by removing the responses from rejected assignments. Moreover, responses to equivalent questions are grouped together. Equal questions and symmetrically reversed questions (questions with the same stimuli but reversed in order) are regarded equivalent. This process generates the table `X`, see Subsection 2.5. The program terminates by writing the output file `TAXIQ.mat`, which contains the tables `T, A, X, I, Q`.

3.2 `mainReconstruction.m`

The reconstruction program reads the output file `TTAXIQIQA.mat` of the filtering with the group summaries of all accepted responses for the triplet comparisons. AIC-3 recommends the collection of response data with ratios of 1:4 between plain and boosted triplet comparisons and also 1:4 between cross-codec and same-codec triplet comparisons. However, after the filtering step the cleansed dataset may have different ratios of these kinds. Therefore, the reconstructions are based on stratified resamplings that guarantee the desired ratios.

The user chooses the type of functional model in the code (`fclass = 1,...,6, bclass = 1,2,3`). The outer main loop in the program goes through the source images. For each source image the inner main loop carries out one or many reconstructions per source image, each with a different stratified data resampling. Having many bootstrapped resamplings allows the computation of confidence intervals (in another program).

The results of the reconstructions are stored in the struct array `Result` and contain for each execution of the inner loop the source and codec names, the functional model type, boosting polynomial degree, the pointwise, the least-squares and the MLE reconstructions, together with negative log-likelihoods and root-mean-square differences between functional

and pointwise models, parameters of the optimizers, and finally the runtime and the date. This list of output items can easily be extended.

3.3 `mainReport.m`

Users may have different goals in mind what parts of the results should be reported in what way. This program `mainReport.m` serves as an example that can be modified an extended to meet the user's intentions. It generates a page of figures per source, each showing the distortion-rate functions for all corresponding codecs in one subfigure per codec. A subfigure presents the graphs for the plain images as well as for the boosted images. In addition, the pointwise reconstructions are shown as circles.

4 Functions

4.1 Data: Importing, filtering and resampling

4.1.1 `csvImport.m`

```
function [T, I, Q] = csvImport (csvPTC, csvBTC, csvImages, fraction)
```

The function reads BTC and PTC data, and image meta data (bitrates etc). Optionally, it selects a random subset of assignments, useful for faster code development. Set `fraction < 1.0` for this purpose. This file follows the Interchange Format in Annex G of the JPEG AIC-Standard (informative). In addition, some checks are carried out: (1) Verify and correct `is_same/is_cross` flags, (2) verify responses are 'left', 'right', 'notsure' or 'skip', (3) verify names of compressed images are different between PTC and BTC, (4) verify and ensure question ids are consecutive from 1 to the max, (5) verify and ensure data rows for each question id are consistent, and (6) verify that the compressed image names have the same PTC/BTC mode flag, the same codec and distortion level in all rows.

Input: `(csvPTC, csvBTC, csvImages, fraction)`

`csvPTC` (string) – Filename of the csv input file for plain triplet comparisons (PTC), columns as in Subsection 2.1

`csvBTC` (string) – Filename of the csv input file for boosted triplet comparisons (BTC), columns as in Subsection 2.1

`csvImages` (string) – Filename of the csv input file for all images used in PTC and BTC, columns as in Subsection 2.2 (except for the number of questions which is computed in the function)

`fraction` (double) – Fraction of randomly chosen assignments that are selected from the available PTC and BTC assignments. This allows to scale down a large dataset to a small one for testing purposes. Default value of `fraction` is 1.0, i.e., all assignments are included. Note: This is not the AIC-3 type filtering.

Output: `[T, I, Q]`

`T` (table) – Contains all parameters, measurements and the responses for triplet comparisons with one row per response. See Subsection 2.1 for the table columns.

`I` (table) – Table of compressed images with metadata. See Subsection 2.2 for details.

`Q` (table) – Table of unique triplet comparisons (questions). See Subsection 2.3 for details.

4.1.2 `filterAssignments.m`

`function [A] = filterAssignments (A, T)`

This function filters assignments based on mean of accuracy and consistency, and on order bias. Otsu thresholding is applied separately for PTC and BTC assignments. The threshold may be manually overriden by uncommenting some lines of code. Scatter plots of accuracy versus consistency are presented.

Input: `(A, T)`

`A` (table) – Table of assignments, see Subsection 2.4

`T` (table) – Table of input data, see Subsection 2.1

Output: `[A]`

`A` (table) – Updated table of assignments including columns computed in the function, see Subsection 2.4

4.1.3 `overviewData.m`

`function [] = overviewData (T)`

Informative overview and proportions correct. The following information is extracted and displayed on the console: Ratios of PTC/BTC, same-codec, cross-codec and trap questions and separately for PTC and BTC data the histograms and tables of responses, figures of response times, tables with proportions correct in answers (l/r/ns) for same-codec questions. Note: This function could also be applied for the filtered master table showing only the data from reliable assignments, for example at the end of `mainFilter.m`.

Input: `(T)`

`T` (table) – Master table containing the input from csv files.

4.1.4 `preprocessData.m`

`function [A] = preprocessData (T, I, informative)`

Preprocessing of the PTC and BTC response data (separately). For each assignment the following features are computed: skip ratio from all triplets responses, weighted consistency, weighted accuracy from same-codec questions, order bias from all triplet comparisons, mean response time. The weights are computed in two ways. The weights according to ISO/IEC 29170-3 are based of differences of distortion levels between the two test stimuli in a triplet comparison. The (preferred) alternative is to compute the weights proportional to the difference of the CVVDP objective assessment of fidelity. In the code, one or the other is chosen.

Input: `(T, I, informative)`

`T` (table) – Master table containing the input from csv files.

`I` (table) – Table of compressed images with metadata. See Subsection 2.2 for details.

`informative` (logical, 0 or 1) – If set to 1, optional features are computed for all assignments, namely the ratio of skipped questions, the proportion of responses 'notsure', and the mean response time per question. The function `overviewData` is called. Set `informative = 0` to compute only the required data for the following steps of the pipeline, namely accuracy, consistency, LR_bias.

Output: `[A]`

`A` (table) – Table of assignments, see Subsection 2.4

4.1.5 `getModeTypeDistribution.m`

`function [] = getModeTypeDistribution (X, T, option)`

In AIC-3 the recommended fractions of plain triplet comparisons and of cross-codec triplet comparisons are 20% each. This amounts to fractions 0.64 for same-codec BTC, 0.16 for each of same-codec PTC and cross-code BTC, and 0.04 for cross-codec PTC. This function computes the actual ratios for three cases, the complete raw data as given by table T, the cleansed dataset given by table X, and for resampled datasets that usually target the above recommended ratios.

Input: `(X, T, option)`

`X` (table) – Table of cleansed dataset with grouped responses per question, see Subsection 2.5

`T` (table) – Master table containing the input from csv files.

`option` (string) – The expected strings are 'raw_data', 'cleansed_data' and 'resampled_data'.

4.1.6 `resampleData.m`

`function [Xdata] = resampleData (X, config)`

This function takes as input the table `X` of summarized responses per unique question and resamples the responses according to specifications given in the configuration structure `config`. These include the total number of sampled responses, `config.budget`, the ratio of PTC responses, `config.ratioPTC`, and the ratio of cross-codec responses, `config.ratioCROSS`. The resampling proceeds as follows. Firstly, the total budget is divided up between the four combinations of modes PTC/BTC and types of same-codec/cross-codec triplet comparisons. Secondly, for each of these four categories their budgets are distributed uniformly between the corresponding questions (triplet comparisons). For each question the required number of responses is randomly sampled with replacement from the available responses.

Input: `(X, config)`

`X` (table) – Table of cleansed dataset with grouped responses per question, see Subsection 2.5

`config` (struct) – Prescription of parameters for the resampling of the response data. The required fields are `ratioPTC, ratioCROSS, budget, seed`. The seed value for generating random numbers is controlled here to allow for exact replication of the resampled datasets later on.

Output: `[Xdata]`

`Xdata` (table) – Table of cleansed and resampled dataset with grouped responses per question, see Subsection 2.5

4.1.7 `setupData.m`

`function [X] = setupData (T, I)`

Setup of the master table for efficient reconstruction of scale values. The master table `T` is already augmented by the question ids (last column). For the scale reconstruction we only need to know how many votes came for the three response options left/right/notsure for each question. Moreover, we can join responses for each pair of symmetric questions. The result is the group summary table `X`.

Input: `(T, I)`

`T` (table) – Master table containing the input from csv files.

`I` (table) – Table of compressed images with metadata. See Subsection 2.2 for details.

Output: `[X]`

`X` (table) – Table of cleansed dataset with grouped responses per question, see Subsection 2.5

4.1.8 `setupJobs.m`

`function [J] = setupJobs (X, I)`

This function prepares the reconstructions that will be carried out separately per source but for all codecs together that were applied to the source image. The function collects and orders these codecs as well as all images involved per source in the table `J`.

Input: `(X, I)`

`X` (table) – Table of cleansed dataset with grouped responses per question, see Subsection 2.5

`I` (table) – Table of compressed images with metadata. See Subsection 2.2 for details.

Output: `[J]`

`J` (struct) – Structure with fields for the source image, the lists of codecs per source, and the lists of images derived from each source image.

4.2 Functional models

4.2.1 `makeModel.m`

`function [model] = makeModel (source, codecs, fclass, bclass)`

Creates a full model structure, initialized with random model parameters for the functions and boosting polynomial coefficients. The core information of a model consists of the source, the codecs applied to it, and the numerical parameters of the distortion rate function and polynomial boosting transfer function.

Input: `(source, codecs, fclass, bclass)`

`source` (categorical) – Source acronym

`codecs` (array of categoricals) – Ordered list of the codecs, the order determines the sequence of variables in the optimization procedure.

`fclass` (integer) – An integer 1,2,... determining the type of distortion-rate finction. For example, fclass = 1 denotes the exponential function.

`bclass` (integer) – An integer 1,2, or 3 determining the number of parameters in the polynomial boosting transfer function. It is equal to the degree of the polynomial.

Output: `[model]`

`model` (struct) – A struct with fields 'source', 'codecs', 'fctpar' and 'polpar', 'fclass', 'bclass', see Subsection 2.7.

4.2.2 `rollout.m`

`function [x] = rollout (model)`

The functional model in the structure `model` contains (among other things) the parameters of the DR-function and of the polynomial boosting function for each codec in two 2D-arrays with a row per codec. The optimizers expect these parameters altogether in the form a column vector `x`. When there are K codecs in the model, `x` has the following structure: K blocks of size `nfct` for the functional models of the K codecs, followed by K blocks of size `npol` for the boosting functions of the K codecs. The function `rollup(x, model)` is the complementary function that takes such a vector `x` and stores it in a structure `model`, see below.

Input: `(model)`

`model` (struct) – A struct with fields 'source', 'codecs', 'fctpar' and 'polpar', 'fclass', 'bclass'. The functional models for codecs are stored in the rows of `model.fctpar`. The boosting parameters are stored in the rows of `model.polpar`. Row $k = 1, ..., K$ refers to the k-th codec in the list `model.codecs`.

Output: `[x]`

`x` (array) – Column vector of $K\times$ `nfct` + $K\times$ `npol` parameters of the functional model in the input.

4.2.3 `rollup.m`

`function [model] = rollup (x, model)`

The complementary function to `rollout`. It copies the input vector into the model structure.

Input: `(x, model)`

`x` (array) – Linear array containing the parameters of the DR-functions and the boosting transformation.

`model` (struct) – Container for storing the parameters of `x`. On input it must provide matching metainformation, i.e., the corresponding numbers of parameters per codec.

Output: `[model]`

`model` (struct) – A struct with fields 'source', 'codecs', 'fctpar' and 'polpar', 'fclass', 'bclass'. The functional models for codecs are stored in the rows of `model.fctpar`. The boosting parameters are stored in the rows of `model.polpar`. Row $k = 1, ..., K$ refers to the k-th codec in the list `model.codecs`.

4.2.4 `DRfunc.m`

`function [d] = DRfunc (rate, idx, fclass, fctpar)`

This function evaluates the DR-function for one or multiple bitrates and different corresponding codecs. The function is specified by the class `fclass` (e.g., exponential) and the rows `idx` of the 2D-parameter array `fctpar` (stored in the model structures). This function gives native units for NLL, for JND units divide by 0.6745.

Input: `(rate, idx, fclass, fctpar)`

`rate` (array) – Array of bitrates (source images have bitrate `inf`)

`idx` (array) – Corresponding array of integer indices 0,1,2,... of N codecs, $N \leq K$ is the number of codecs used in the functional model that is evaluated. The indices give the row numbers for the codecs in the 2D-array `fctpar` (for source images the row number is 0).

`fclass` (integer) – Integer 1, ..., 6 determining the type of DR-function.

`fctpar` (array) – 2D-array of model parameters. Each row corresponds to one codec.

Output: `[d]`

`d` (array) – Array of distortion values for the given bitrates.

4.2.5 `boost.m`

`function [v] = boost (v, idx, bclass, polpar)`

Applies the boosting transformation to the input array `v` for a codec that is specified by `idx` and a polynomial of degree `bclass` (1, 2 or 3) with coefficients in a row of the array `polpar`.

Input: `(v, idx, bclass, polpar)`

`v` (array) – Array of values of perceived distortion of compressed images without boosting.

`idx` (array) – Corresponding array of integer indices 0,1,2,... of codecs as in the function `DRfunc`. Eaxh index indentifies a row number in the 2D-array `polpar`, identifying the codec.

`bclass` (integer) – Polynomial degree of the boosting transformation.

`polpar` (array) – 2D-array of polynomial coefficients. Each row corresponds to one codec.

Output: `[v]`

`v` (array) – Array of transformed plain compression distortions, representing perceived distortion of compressed images when boosted.

4.2.6 `fct.m`

`function [v] = fct (model, data)`

This function evaluates a functional model for a list of N images (plain and boosted). It is at the core of the objective functions to be minimized in the optimizer for the negative log-likelyhood. It calls the functions `DRfunc` and `boost`.

Input: `(model, data)`

`model` (struct) – Holds the models for K codecs used for one fixed source. See Subsection 2.7 for details.

`data` (struct) – The structure contains the required information for the N images in three column vectors of length N. These fields are as follows. The array `data.btc` is a logical vector indicating the boosted images (BTC). The array `data.bitrate` contains the corresponding bitrates. The array `data.idx` contains vector of N codec indices for the N images. The K codecs listed in model.codecs are enumerated as $1, ..., K$. For source images, the bitrate is `inf` and the codec index is 0.

Output: `[v]`

`v` (array) – Array of distortion values for the given images.

4.2.7 `fctScalar.m`

```
function [v] = fctScalar (M, mode, codec, bitrate)
```

This is a scalar version of the function `fct`, computing the distortion value for just one image. However, this scalar version differs from the vector version: Input is a codec acronym, not a codec index. The function is less efficient as `fct` but more practical in less time-critical contexts such as reporting and production of figures.

Input: `(M, mode, codec, bitrat)`

`M` (struct) – Holds the models for K codecs used for one fixed source. See Subsection 2.7 for details. Note that only one of these will be evaluated by this function.

`mode` (categorical) – `PTC` or `BTC`.

`codec` (categorical) – Acronym of codec used for the image.

`bitrate` (double) – Bitrate of codec applied for the image.

Output: `[variables]`

`v` (double) – Distortion value for the given image.

4.2.8 `getBounds.m`

```
function [lb, ub] = getBounds (model)
```

Sets the lower and upper bounds for all variables in the optimization. These bounds are heuristic.

Input: `(model)`

`model` (struct) – Holds the models for K codecs used for one fixed source. See Subsection 2.7 for details.

Output: `[variables]`

`lb` (array) – Lower bounds for all variables under optimization.

`ub` (array) – Upper bounds for all variables under optimization.

4.2.9 `getNpars.m`

`function [nfct, npol, name] = getNpars (model)`

The function supplies for the given model the number of parameters in a DR-function, in a boosting polynomial, and an acronym of the type of DR.function.

Input: `(model)`

`model` (struct) – Holds the models for K codecs used for one fixed source. See Subsection 2.7 for details.

Output: `[nfct, npol, name]`

`nfct` (integer) – Number of parameters in a DR-function

`npol` (integer) – Number of parameters in a boosting polynomial (the degree)

`name` (string) – Acronym of the type of DR-function.

4.2.10 `printModel.m`

`function [] = printModel (model, str)`

Prints to console the source image acronym of the model, the given input string and the parameters of the DR-functions and boosting functions for all the codecs covered in the model.

Input: `(model, string)`

`model` (struct) – Holds the models for K codecs used for one fixed source.

`str` (string) – String to be displayed.

4.2.11 `printModelDiff.m`

`function [] = printModelDiff (M1, M2)`

Prints to console the difference between parameters in two compatible models (i.e., with the same DR- and boosting function types).

Input: `(variables)`

`M1, M2` (structs) – Hold the two models used for one fixed source.

4.2.12 `printModelFunctions.m`

`function [] = printModelFunctions (fclass, bclass)`

Prints to console one line informing about DR- and boosting function types.

Input: `(fclass, bclass)`

`fclass` (integer) – Integer 1, ..., 6 determining the type of DR-function.

`bclass` (integer) – Polynomial degree of the boosting transformation.

4.3 Optimizers

4.3.1 `fitAllModels.m`

`function [result] = fitAllModels (Job, XS, fclass, bclass, result)`

For the reconstruction of scale values of perceived distortion magnitude, different models can be applied: the classic pointwise model, the functional model fitted to the pointwise reconstuction using the least-squares method, and the functional model from maximum likelihood estimation. In this function all three models are computed for a given source image and an associated list of codecs. Corresponding data from triplet comparisons must be provided. The result of the least-squares fit is used as the initial guess for the optimizer in the MLE approach.

Input: `(Job, XS, fclass, bclass, result)`

`Job` (struct) – Structure with fields for the source image, the corresponding list of codecs for the source image, and the list of images derived from the source image. The function `setupJobs` generates a structure array for all source images, and typically, the function `fitAllModels` will be called with `Job` being one of the entries of this struct array. See Subsection 4.1.8.

`XS` (table) – Contains the input data from the cleansed dataset with grouped responses per question and filtered for the source image.

`fclass` (integer) – Integer 1, ..., 6 determining the type of DR-function.

`bclass` (integer) – Polynomial degree of the boosting transformation.

`result` (struct) – A structure which collects results from all computed reconstructions along with useful side information for later analysis and reporting. For efficiency, the fields of the structure are defined in the calling program `mainReconstruct` and must be adhered to. Although the input structure is not actively used in this function, it is necessary to include here to ensure that all required fields are passed to the calling program.

Output: `[result]`

`result` (struct) – See above.

4.3.2 `fitPTS.m`

`function [ImgList, nll] = fitPTS (ImgList, XSM)`

This function provides the classic pointwise reconstruction by MLE, i.e., one distortion scale value per compressed image, in native Thurstonian units.

Input: `(ImgList, XSM)`

`ImgList` (table) – Table of images, filtered for the source image and the mode (PTC or BTC).

`XSM` (table) – Contains the input data from the cleansed dataset with grouped responses per question and filtered for the source image and the mode (PTC or BTC) and with columns restricted to `img_left, img_right, left, right`.

Output: `[ImgList, nll]`

`ImgList` (table) – Same as the input table, with one additional column for the computed scale value for each image.

`nll` (double) – Value of the overall negative likelihood of the computed pointwise model w.r.t. the input data.

4.3.3 `fitLS.m`

`function [model, data, exitflag] = fitLS (model, ImgList)`

Least-squares fit of a functional model to the pointwise reconstruction.

Input: `(model, ImgList)`

`model` (struct) – Template structure for the functional models. See Subsection 2.7 for details.

`ImgList` (table) – Table of images, filtered for the source image.

Output: `[model, data, exitflag]`

`model` (struct) – Structure for the functional models for K codecs used for one fixed source.

`data` (structure) – The minimal data to extract the scale values for the images compressed from the source image (bitrate, logical indicator for BTC images, codec index), compare 4.2.6.

`exitflag` (integer) – Flag passed through from the optimizer.

4.3.4 `fitMLE.m`

`function [MLE_model, MLE_nll, exitflag] = fitMLE (LS_model, DL, DR)`

Maximum likelihood estimation of a functional model for triplet comparisons with initialization by a least-squares model.

Input: `(LS_model, DL, DR)`

`LS_model` (struct) – The functional model from the least-squares fit to the pointwise reconstruction. It provides the initialization for the MLE of a functional model and serves as a template for the MLE model (only the numerical parameters are replaced).

`DL` (struct) – The data for the list of left images in the triplet comparisons (bitrate, logical indicator for BTC images, codec index, number of votes).

`DR` (struct) – The corresonding data for the list of right images in the triplet comparisons.

Output: `[MLE_model, MLE_nll, exitflag]`

`MLE_model` (struct) – The functional model from the MLE for the triplet comparisons for one source and all involved codecs.

`MLE_nll` (double) – The minimal negative log-likelihood that could be achieved in the optimization.

`exitflag` (integer) – Flag passed through from the optimizer.

4.3.5 nllfct.m

`function [nll, gtest, pval, Perf] = nllfct (x, M, DL, DR, nboot)`

The main purpose of this function is the efficient computation of the negative log-likelihood of the functional model, given by `x` and `M`, w.r.t. the input data `DL` and `DR`. Optionally, the function may compute the corresponding G-test statistic, and even a p-value for the null hypothesis given by the functional model. The p-value is estimated by parametric bootstrapping, i.e., generating many samples of triplet comparisons corresponding to those in `DL` and `DR`, then computing the resulting statistics. Several statistics are implemented: G-test, l1, and l2.

Input: `(x, M, DL, DR, nboot)`

`x` (array) – 1D-array of variables (parameters) of a functional model for which the NLL is computed. Usually, the function `rollout` generates such a vector of variables from a fully defined model structure. To ensure that the input model `M` properly reflects the input model parameters in `x`, the function first copies `x` into `M`.

`M` (struct) – Holds the models for K codecs used with one fixed source image.

`DL` (struct) – The data for the list of left images in the triplet comparisons (bitrate, logical indicator for BTC images, codec index, number of votes).

`DR` (struct) – The corresponding data for the list of right images in the triplet comparisons.

`nboot` (integer) – Number of bootstrap samples used to compute the p-value based on the G-test statistic (optional, i.e., when `narg > 2`).

Output: `[nll, gtest, pval, Perf]`

`nll` (double) – Value of the overall negative likelihood of the functional model, given by `x` and `M`, w.r.t. the input data `DL` and `DR`.

`gtest` (double) – Value of the G-test statistic. Provided, when more than one output variable is specified in the call, i.e., `nargout > 1`.

`pval` (struct) – P-values based on bootstrapped evaluation of the distribution of several statistics under the null hypothesis of the input model. Provided, when more than two output variables are specified in the call, i.e., `nargout > 2`. The field names are `gtest, l1, l2`, corresponding to the used statistics.

`Perf` (table) – Contains columns of data useful for analysis of the G-test statistic under the null hypothesis. Not used in the reconstruction. Provided, when more than two output variables are specified in the call, i.e., `nargout > 2`.

4.3.6 rmsDiff.m

`function [rmsd] = rmsDiff (x, model, data, scalePT)`

Computes the root-mean-square difference of the functional models w.r.t. given scale values in `scalePT`, which contains the pointwise scale reconstruction. The models are applied for one source image using several codecs with and without boosting, yielding a list of plain and boosted compressed images at given bitrates. This root-mean-square difference is minimized in the function `fitLS`, and it is also reported for the results of `fitMLE`.

Input: `(x, model, data, scalePT)`

`x` (array) – 1D-array of variables (parameters) of a functional model for which the RMSE is computed.

`model` (struct) – Structure for the functional models. The numerical parameters of the functional model are ignored and copied from `x`, instead.

`data` (struct) – The data for the list of images consisting of bitrate, logical indicator for BTC images and codec index.

`scalePT` (array) – The array of pointwise reconstructions of perceived distortion for the list of images.

Output: `[rmsd]`

`rmsd` (double) – The root-mean-square difference between the functional model, applied to the list of images w.r.t. the given scale values in `scalePT`.

4.3.7 `optPar.m`

`function [Output] = optPar (type)`

Returns the optimization parameters for the general nonlinear Matlab optimizer `fmincon` for the three types of optimization (PT, LS, MLE). These parameters include the maximal number of iterations, maximal number of function evaluations and several tolerances.

Input: `(type)`

`type` (string) – One of the following strings: `"PT", "LS", "MLE", "All"`.

Output: `[optPar]`

`optPar` (struct) – The optimizer parameters. In case of `type = "All"`, the parameters for the three main types `"PT", "LS", "MLE"` are reported altogether.

4.4 Utilties

4.4.1 `calculateAllBinomialPValues.m`

`function [pValueL, pValueR, pValueB, pValueS] = calculateAllBinomialPValues (n, k, p0, alpha, verbose)`

The function computes p-values for the null hypothesis of a binomial probability distribution with parameter `p0`, the probability of success in a Bernoulli experiment, and w.r.t. the data given by `k` successes out of `n` observations. Different p-values are extracted for the left-, right- and two-sided cases. The p-value for the two-sided case is given in two variations, once as two times of the minimum of the left- and right-sided p-value (`pValueB`), and once as the sum all probabilities less than or equal to the given one for `k` successes out of `n` trials (`pValueS`, also called the 'sum of small values').

Input: `(n, k, p0, alpha, verbose)`

`n` (integer) – Total number of trials

`k` (integer) – Observed number of successes

`p0` (double) – Probability of success

`alpha` (double) – Significance level (e.g., 0.05)

`verbose` (logical) – Setting it to `true (1)` will print out p-values and interpretation.

Output: `[pValueL, pValueR, pValueB, pValueS]`

`pValueL` (double) – p-value for the left-tailed test

`pValueR` (double) – p-value for the right-tailed test

`pValueB` (double) – p-value for the two-tailed test, classic method

`pValueS` (double) – p-value for the two-tailed test, sum of small values

4.4.2 `mydisp.m`

`function [[]]] = mydisp (fid, varargin)`

Prints formatted text if global variable `verbose` is `true`. Usage: `mydisp(fid, formatSpec, ...)`. Variables are defined as `global` in some of the main programs.

4.4.3 `myfprintf.m`

`function [[]]] = myfprintf (fid, varargin)`

Prints formatted text if global variable `verbose` is `true`. Usage: `myfprintf(fid, formatSpec, ...)`. Variables are defined as `global` in some of the main programs.

A The `readme` file

The appendix contains a rendering of the `readme` file, explaining how to use the AIC-3 Data Analysis software.

Appendix (readme.md)

JPEG AIC-3 Data Analysis

This code implements the data analysis of responses to triplet comparisons collected in an image quality assessment campaign organized according to the JPEG AIC-3 standard, to be published as ISO/IEC 29170-3 in 2026.

- An extended documentation is contained in the PDF `Overview of the JPEG AIC-3 Data Analysis Software.pdf` (ISO/IEC JTC 1/SC 29/WG1N101393) on the JPEG AIC webpage `https://jpeg.org/aic/documentation.html`. In the following this is referred to as *Documentation*.
- An example application with listings and results is collected in this repository.

The data analysis has two main steps, followed by a third step for the presentation of the computed solutions.

1. Dataset cleansing (`mainFilter.m`). The dataset cleansing produces a reliable subset of the collected data by identifying and removing assignments of subjects that are considered to contain responses that are not generated by the perceptual and cognitive processes under investigation.

2. Scale reconstruction (`mainReconstruct.m`). For each source image with its compressed versions, the cleansed dataset of responses to plain and boosted triplet comparisons is fitted with a unified functional model given by distortion-rate functions. The functional model implies probabilities for all possible responses to the triplet comparisons. The fitting procedure is based on minimization of the negative log-likelihood. In addition, the classic pointwise reconstruction and another functional model that minimizes the differences to the pointwise reconstructions in the least-squares sense is computed.

3. Presentation (`mainReport.m`). This program reads the results of the reconstruction process and shows for each source image a figure with the distortion-rate curves from the plain and boosted triplet comparisons as well as the pointwise model.

Input files

The code contains the three main programs as `.m` files, four subdirectories with functions (`data` , `models` , `optimizer` , `utils`) and one subdirectory `input` for the input files. To run the program, three files are expected in the input subdirectory:

- `ptc.csv` : A csv file containing the response data for plain triplet comparisons (PTC). Each

row has the data for one response.

- `btc.csv` : A similar csv file containing the response data for boosted triplet comparisons (BTC).
- `images.csv` : A csv file containing metadata for the compressed images.

The columns in these csv files are listed in the following sections. The columns with a starred names are mandatory, the others optional.

Column Descriptions for `PTC.csv` and `BTC.csv`

Column Name	Type	Description
`assignment*`	string	Unique identifier for each assignment given to a worker
`worker`	integer	Unique anonymized identifier for each worker who participated in the study
`mode*`	string	Method used for image comparisons (BTC or PTC)
`batch`	integer	Unique identifier for each batch within the study
`question_id*`	integer	Unique identifier for each question presented to the workers
`source*`	string	Unique acronym for the source image
`codec_eft*`	string	Unique acronym for the codec used for compression of the left image in the triplet
`codec_right*`	string	Unique acronym for the codec used for compression of the right image in the triplet
`dlevel_left*`	integer	Distortion/compression level of the left image
`dlevel_right*`	integer	Distortion/compression level of the right image
`img_left*`	string	Filename of the left image
`img_pivot*`	string	Filename of the pivot image
`img_right*`	string	Filename of the right image
`is_same*`	logical	Boolean value indicating if the left and right image in the triplet comparison are compressed with the same codec (same-codec comparison)
`is_cross*`	logical	Boolean value indicating if the triplet is a cross-codec comparison. So either is same or is cross is true
`is_trap*`	logical	Boolean value indicating if the triplet is a trap question
`question_order`	integer	Order of the question in a batch presented to the worker

`response*`	string	Worker's response to the question (either `left`, `notsure`, `right`, or `skip`)
`submission_time`	string	Timestamp when the worker submitted the response (datetime format)
`response_time`	double	Time taken by the worker to respond, in seconds
`reload_count`	integer	Number of times the worker reloaded the page during the task
`resolution`	string	Screen resolution of the worker's device during the task
`toggle_count`	integer	Only for PTC: Number of toggles between the source and test images

Column Descriptions for `image.csv`

Column Name	Type	Description
`name*`	string	Filename of compressed image
`source*`	categorical	Acronym of the source image for the compressed image
`codec*`	categorical	Acronym of the codec used for the compressed image
`dlevel*`	integer	Distortion level of the compressed image
`bitrate*`	double	Bitrate in bpp for the compressed image
`cvvdp*`	double	CVVDP objective metric value for the compressed image. IF CVVDP cannot be provided a substitute metric must be given (do not change the column name).
`rmse`	double	RMSE metric value for the compressed image
`psnr`	double	PSNR objective metric value for the compressed image
`ssim`	double	SSIM objective metric value for the compressed image

Running the programs

Execute the three main programs sequentially:

- `mainFilter.m`: The input files are read and processed. The results are written to the outputfile `TAXIQ.mat`, containing five tables `T`, `A`, `X,` `I`, `Q`. See the *Documentation* for the details of the data contained in these tables.
- `mainReconstruct.m`: Reads `TAXIQ.mat`, does the reconstructions and writes the output file `TAXIQResult.mat`. This file contains the (augmented) tables from the input file

`TAXIQ.mat` and the structure `Result` that lists all pointwise models together with the least-squares and MLE functional models.

- `mainReport.m` : Reads `TAXIQResult.mat` and produces a figure per source image that shows reconstructions for all corresponding codecs. This function is set up to handle up to six codecs for any given source image. Edit the code in `mainReport.m` to accommodate more than six codecs for a source, or to generate other figures and tables.

Communicating author

Dietmar Saupe, `dietmar.saupe@uni-konstanz.de`