Definition of Data Formats for Optimization Software

Revision 0.2, June 2011.
(Revision 0: Jan. 2011)

Igor Grešovnik

Contents:

1 Introduction 1

1.1 How Standards are Specified 1

2 Formats 1

2.1.1 Comments on the JSON format 2

2.2 Neural Networks 2

2.2.1 Usage scheme for Neural Networks in Response Approximation and Optimization 2

2.2.2 Formats for Neural Networks used in Optimization 3

2.2.2.1 Neural network data definition file 4

2.2.2.2 Neural network training data set 5

2.2.2.3 Neural network input and output data (parameters at which input is evaluated) 7

2.2.2.4 Neural network direct analysis files ..... **Error! Bookmark not defined.**

2.3 Direct Analysis for Optimization Procedures 8

2.3.1 General scheme: optimization algorithm – direct analysis 8

2.3.2 Formats for Data Exchange Between Direct Analysis and Optimization Code 9

2.3.2.1 Standart analysis request format (analysis input file) 9

2.3.2.2 Standard analysis results format (analysis output file) 10

2.3.2.3 Standard analysis results format (analysis output file) for multiobjective case (currently not in use) 11

2.3.2.4 Possible alternative: XML format for standard analysis results (analysis output file) for single objective case (currently not in use) 12

Legend of graphic symbols:

- this section / paragraph / text is not yet complete. Since nothing can be considered definite or complete at a library like "**Error! Reference source not found.**", this sign will denote portions of this manual where more content is intended at this very moment, but there was no time to add it.

- Developers' section – these contents will be of more interest for developers of the library than its users. To define the terms – users of the library will usually be developers of some other software. In this document, term "developers" is used for those who contribute to the library itself, i.e. people who add functionality and make it publicly available, who suggest conceptual changes or who contribute free additional documentation for the benefit of other users and developers.

- Consideration.

- Warning.

1 INTRODUCTION

This document defines various data formats that are used for storage and exchange of data in various software modules.

Primary aim of this is to define a set of standard formats that will enable exchange of data between different parts of software, which are sometimes implemented as separate applications. An example of this includes exchange of optimization parameters and response functions between optimization software and software that implements a direct analysis (e.g. a neural network approximation tool or mesh-less simulation software).

Another possible aim is to define formats for storage of input and output data (results) for numerical simulations, which would enable linking of various simulations for simulation of complete chains of processes, or development of common pre-processing or post-processing software for different simulation programs. For this aim things are not yet clear because it is unclear how software development will actually be organized within the group. It would also make much more sense to put development of simulation software on common grounds (i.e. to adopt the idea of a unified simulation framework), because in this way there would be much more development potential available for development of standardized pre- and post- processing components and input/output modules for numerical simulations. Further extension of these ideas would be modules for programmable manipulation with simulation software, but implementation of these ideas is not feasible in the current stage.

1.1 How Standards are Specified

This documents aims at providing an overview of standard data formats used. However, this is not authoritative standard itself. In many cases formats are specified through examples and are not necessarily up to date. Instead, code implementation within or based on the IGLib library will be used as de facto standard for all data formats mentioned hereby.

2 FORMATS

Most of the information exchange formats used are JSON formats[1]. JSON (JavaScript Object Notation) is a well established standard for transferring objects between various applications and platforms, it is a human readable text format, very simple and less redundant as most other standard text formats used for similar purposes. JSON is also extensible to a certain degree. If something is added in later versions of data definitions without changing the current field names,

data written in according to old definitions can still be read in when using updated object definitions.

2.1.1 Comments on the JSON format

This format can represent general object that include other objects, plain data such as numbers and strings, and arrays of objects or plain data.

Order of fields within objects is not prescribed, therefore an arbitrary order can be used.

The null keyword can be used for any object that is not specified.

Fields that are not specified can be simply omitted, if the data represented by the file permits these fields to be unspecified.

Spaces do not matter. One can omit all spaces in order to make files shorter. In the examples presented in the current document, spacing and indentation is used such that the code is more readable.

2.2 Neural Networks

2.2.1 Usage scheme for Neural Networks in Response Approximation and Optimization

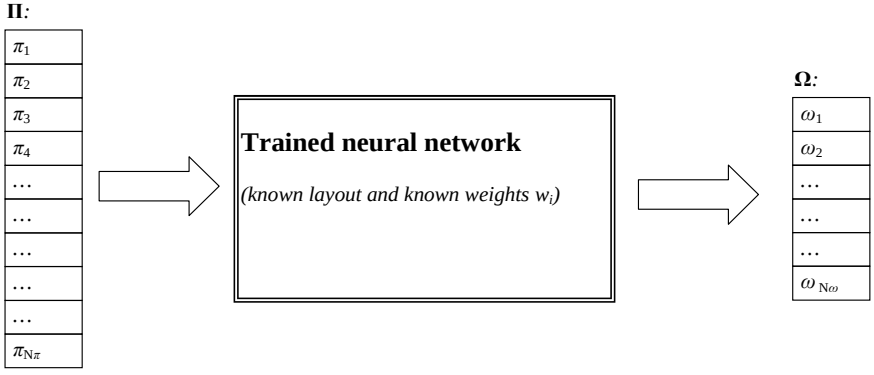


Figure 1: Neural network serving as an approximate model of a real-life process.

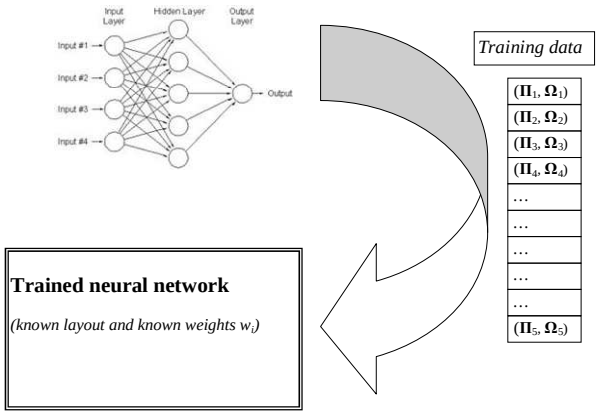
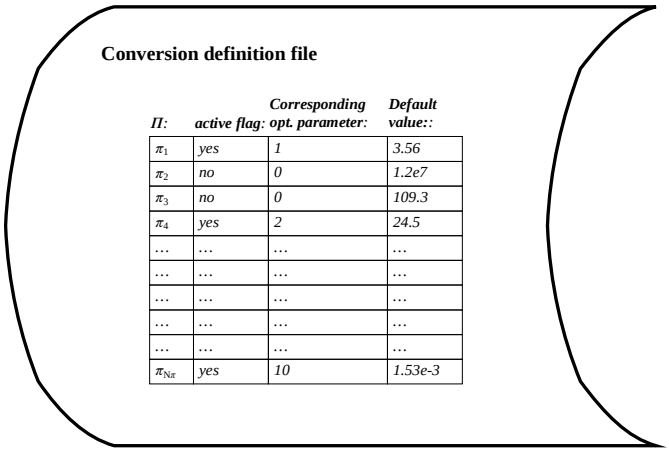


Figure 2: Training of a neural network.



Conversion definition file

Π:	*active flag:*	*Corresponding opt. parameter:*	*Default value::*
π_1	*yes*	*1*	*3.56*
π_2	*no*	*0*	*1.2e7*
π_3	*no*	*0*	*109.3*
π_4	*yes*	*2*	*24.5*
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
$\pi_{Nπ}$	*yes*	*10*	*1.53e-3*

Figure 3: Example of primitive parameterization: sample conversion file that defines conversion from optimization parameters to process parameters used to utilize the trained neural network as direct analysis used for neural network-based optimization (test purposes only).

2.2.2 Formats for Neural Networks used in Optimization

When neural network is used as approximation of arbitrary response or as approximation of direct analysis response (used in optimization), all data exchange files must be put in a single

directory that is used for file-based data exchange and other communication. Standard names are used for data exchange and signaling files.

2.2.2.1 Neural network data definition file

This file contains definitions of input and output parameters of the neural network. This enables one to check whether correct data are transferred, and to define new direct analysis modules based on custom definitions of optimization parameters and optimization response functions.

The outer-most object contains two data elements, *Input* and *Output* (separate definitions – descriptions of input and output parameters) and their corresponding dimensions, *InputLength* and *OutputLength*. *Input* and *Output* contain array of objects that describe each individual data element of either input parameters or output values. Both of them contain fields *IsInput* (specifying whether the specific element is a part of input or output data), *Name* (unique name of the element, must comply with standards for variable names in programming languages like C, Java or C#), *Description* (optional), *ElementIndex* (index of the element in the corresponding data vector, either input or output), and *ElementIndexSpecified* (a flag telling whether *ElementIndex* is specified – because it does not need to be). *Input* contains the following additional fields: *DefaultValue* (the defalult value of the corresponding input parameter), *MinimalValue* and *MaximalValue* (prescribed bounds within which the corresponding input parameter should lie) and *BoundsDefined* (a flag specified whether bounds are defined; if false then *MinimalValue* and *MaximalValue* will be ignored by all operations).

Data definition file name: *neuraldatadefinition.json*

Example:

```
{
  "Input": [
    {
      "Description": "This is description of input parameter No. 0.",
      "ElementIndex": 0,
      "ElementIndexSpecified": true,
      "IsInput": true,
      "Name": "param_0",
      "BoundsDefined": false,
      "DefaultValue": 0,
      "MaximalValue": 0,
      "MinimalValue": 0
    },
    {
      "Description": "This is description of input parameter No. 1.",
      "ElementIndex": 1,
      "ElementIndexSpecified": true,
      "IsInput": true,
      "Name": "param_1",
      "BoundsDefined": false,
      "DefaultValue": 0,
      "MaximalValue": 0,
      "MinimalValue": 0
    }
  ],
  "InputLength": 2,
  "Output": [
    {
      "Description": "This is description of output parameter No. 0.",
      "ElementIndex": 0,
      "ElementIndexSpecified": true,
      "IsInput": false,
      "Name": "param_0"
    }
  ],
  "OutputLength": 1
}
```


Figure 4: Example of neural network data definition file. This example is for two input parameters and one output value, bounds on input parameters are not defined.

2.2.2.2 Neural network training data set

This data is used to train a network. Usually, trained network can be saved into a file (i.e. entire network topology and values of weights), but formats of these files can vary and can be dependent on neural network software used. Our internal format of training data is standardized, however. It contains pairs of neural network inputs and corresponding outputs (measurements, either actually measured or artificially generated) in the fiels *Elements* of the outer-most object. Beside these pairs, the outer-most object contains the field *InputLength* (number of input parameters

of the neural network), *OutputLength* (number of output values generated by the neural network) and *Length* (the number of input/output pairs – training elements - in the training set).

Each training element contains two fields, *InputParameters* and *OutputValues* (names are self-descriptive). Both are vectors, containing the fields *Length* (dimension of the vector) and *Elements* (array of component values).

Training data file name: *neuraltrainingset.json*

Example:


```
{
  "Elements": [
    {
      "InputParameters": {
        "Elements": [
          1,
          1.001
        ],
        "Length": 2
      },
      "OutputValues": {
        "Elements": [
          10
        ],
        "Length": 1
      }
    },
    {
      "InputParameters": {
        "Elements": [
          2,
          2.001
        ],
        "Length": 2
      },
      "OutputValues": {
        "Elements": [
          20
        ],
        "Length": 1
      }
    }
  ],
  "InputLength": 2,
  "Length": 2,
  "OutputLength": 1
}
```


Figure 5: Example of neural network training set.

2.2.2.3 Neural network input and output data (parameters at which input is evaluated)

Input data is used read by neural network module to produce the output data. Neural network module usually includes a wrapper class (which reads input data, applies it to the neural network software, gets the calculated output and writes this output to the output data file). Neural network module is coded in our software libraries, while neural network software can be external software (either defined as a library or a stand alone application). The neural network module written for specific kind of neural network software must therefore implement reading/writing of input and output data in standard format, and must transfer this data to/from neural network (either through files or directly through API calls).

Both files are in the same format and simply contain a vector of values.

Input data file name: *neuralin.json*

```
{
  "Elements": [
    2,
    2.001,
    2.002,
    2.003,
    2.004,
    2.005
  ],
  "Length": 6
}
```

Figure 6: Example of input parameters for neural network.

Output data file name: *neuralout.json*

Example:

```
{
  "Elements": [
    20,
    20.001,
    20.002
  ],
  "Length": 3
}
```

Figure 7: Example of neural network training set.

2.3 Direct Analysis for Optimization Procedures

2.3.1 General scheme: optimization algorithm – direct analysis

minimise	$f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n$	(1) a)
subject to	$c_i(\mathbf{x}) \le 0, \ i \in I$	b)
and	$c_j(\mathbf{x}) = 0, \ j \in E,$	c)
where	$l_k \le x_k \le u_k, \ k = 1, 2, ..., n$.	d)

Figure 8: Example: statement of an optimization problem.

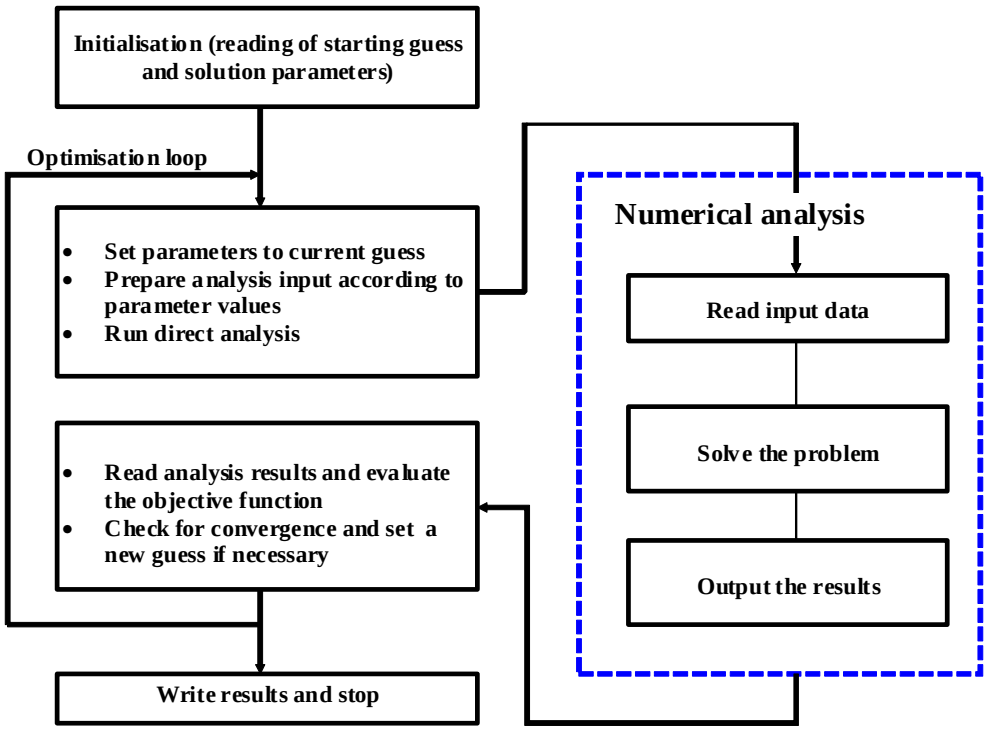


Figure 9: Solution scheme for optimization problems.

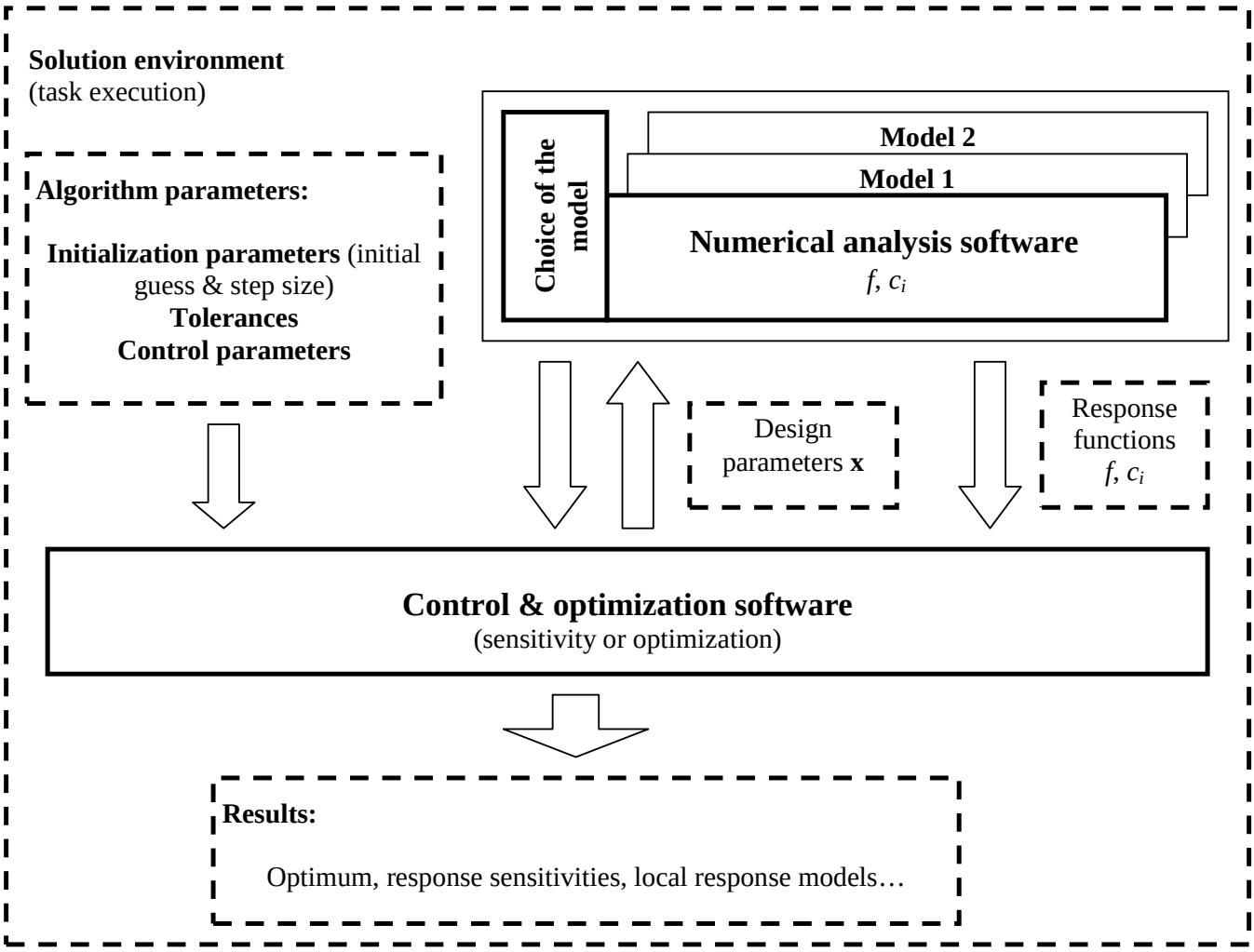


Figure 10: Solution environment scheme.

2.3.2 Formats for Data Exchange Between Direct Analysis and Optimization Code

2.3.2.1 Standart analysis request format (analysis input file)

Format for this file is as follows:

```
{ { p1, p2, … }, { reqcalcobj, reqcalcconstr, reqcalcgradobj,
            reqcalcgradconstr }, cd }
```

Legend:

p1, *p1*, *p3* – optimization parameters at which analysis was performed
Flags that tell whether something has actually been calculated (0 – yes, 1- no):

- *reqcalcobj* – flag for the objective function
- *reqcalcconstr* – flag for constraint functions
- *reqcalcgradobj* – gradient of the objective function
- *reqcalcgradconstr* – gradients of constraint functions

cd – a free parameter that can be used to transfer additional information to the direct analysis. In principle *cd* can be anything embedded in curly brackets (*{..}*) If only the eventual embedded curly brackets are properly closed. Most commonly it will not be used at all and therefore empty brackets ("*{}*") will be put in place of *cd*. Otherwise, interpretation of what stands in curly bracket is entirely in the domain of the analysis program, therefore the documentation of the analysis program should provide information on how to compose *cd*.

```
{{0.1111, 0.2222, 0.3333}, {1, 1, 1, 1}, {}}
```

Figure 11: Example of analysis request file (3 parameters, calculate objective function, constraints and their gradients).

2.3.2.2 Standard analysis results format (analysis output file)

```
{
  { p1, p2 ... },
  {
    calcobj, obj,
    calcconstr, { constr1, constr2, ... },
    calcgradobj, { dobjdp1, dobjdp2, ... },
    calcgradconstr,
    {
      { dconstr1dp1, dconstr1dp2, ... },
      { dconstr2dp1, dconstr2dp2, ... },
      ...
    },
    errorcode
  },
  { reqcalcobj, reqcalcconstr, reqcalcgradobj, reqcalcgradconstr }
  < , { ind1, ind2, ... }, { coef1, coef2, ... }, defdata >
}
```

Legend:

- *calcobj* – flag for the objective function
- *calcconstr* – flag for constraint functions
- *calcgradobj* – gradient of the objective function
- *calcgradconstr* – gradients of constraint functions

obj – value of the objective functions
constr1, *constr2*, … - values of the constraint functions
dobjdp1, *dobjdp2*, ... – derivatives of the objective function with respect to individual parameters (components of the objective function gradient)

dconstr1dp1, …, *dconstr2dp1*, *dconstr2dp2* – derivatives of individual constraint functions with respect to individual optimization parameters – components of gradients of the constraint functions (e.g. *dconstr2dp3* is the derivative of the second constraint function with respect to the third parameter)

errorcode – integer error code of analysis – 0 for no error, usually a negative number for errors, values are function specific

reqcalcob , *reqcalcconstr*, *reqcalcgradobj* and *reqcalcgradconstr* are request flags for calculation of the various values, as have been passed to the analysis function. The same as with parameter values, these flags are requested only for verification. In vast majority of cases these flags will not be used by the optimization program, and they can simply be set to 1.

```
{
  {0.1111, 0.2222, 0.3333},
  {
    1, 1.1111,
    1, {200.1, 200.2},
    1, {10, 10.002, 10.004},
    1,
    {
      {10.101, 10.102, 10.103},
      {20.201, 20.202, 20.203}
    },
    0
  },
  {1, 1, 1, 1}
}
```

Figure 12: Example of analysis result file (3 parameters, 2 constraints, calculate objective function, constraints and their gradients).

2.3.2.3 Standard analysis results format (analysis output file) for multiobjective case (currently not in use)

```
{
  { p1, p2 ... },
  {
    calcobj, {obj1, obj2, ... },
    calcconstr, { constr1, constr2, ... },
    calcgradobj,
    {
      { dobj1dp1, dobj1dp2, ... },
      { dobj2dp1, dobj2dp2, ... },
      ...
    },
    calcgradconstr,
    {
      { dconstr1dp1, dconstr1dp2, ... },
      { dconstr2dp1, dconstr2dp2, ... },
      ...
    },
    errorcode
  },
```

```
  { reqcalcobj, reqcalcconstr, reqcalcgradobj, reqcalcgradconstr }
  < , { ind1, ind2, ... }, { coef1, coef2, ... }, defdata >
}
```

Examples of analysis ouptut files:

```
{ {1.11, 2.22}, { 1, 6.1605, 1, {-0.165, -2.44} , 1, {2.22, 4.44}, 1, { {-
            1.5, 0.}, {0., -2.} }, 0 }, { 1, 1, 1, 1}, {}, {}, "3" } }

{ {1.11, 2.22}, { 1, 6.1605, 1, {-0.165, -2.44} , 0, { }, 0, {  }, -1 }, { 1,
            1, 1, 1}, {33, 45}, {2.5, 3.33 38.1}, "3" } }
```

2.3.2.4 Possible alternative: XML format for standard analysis results (analysis output file) for single objective case (currently not in use)

Alternative format: XML (analysis output):

```
<!-- Analysis output file, created by analysis wrapper. -->
<data type="analysispoint" mode="analysis_output" ind="1">
  <ret type="counter">0</ret>
  <reqcalcobj type="counter">1</reqcalcobj>
  <reqcalcconstr type="counter">1</reqcalcconstr>
  <reqcalcgradobj type="counter">1</reqcalcgradobj>
  <reqcalcgradconstr type="counter">1</reqcalcgradconstr>
  <calcobj type="counter">1</calcobj>
  <calcconstr type="counter">1</calcconstr>
  <calcgradobj type="counter">1</calcgradobj>
  <calcgradconstr type="counter">1</calcgradconstr>
  <param type="vector" dim="2">
    <vector_el type="scalar" ind="1">1.6</vector_el>
    <vector_el type="scalar" ind="2">1</vector_el>
  </param>
  <obj type="scalar">0.20088905308774715</obj>
  <constr type="table" eltype="scalar" dim="2">
    <table_el type="scalar" ind="1">0.0</table_el>
    <table_el type="scalar" ind="2">0.0</table_el>
  </constr>
  <gradobj type="vector" dim="2">
    <vector_el type="scalar" ind="1">0.24138</vector_el>
    <vector_el type="scalar" ind="2">0.0172418</vector_el>
  </gradobj>
  <gradconstr type="table" eltype="vector" dim="2">
    <table_el type="vector" dim="2" ind="1">
      <vector_el type="scalar" ind="1">-1.1</vector_el>
      <vector_el type="scalar" ind="2">2.1</vector_el>
    </table_el>
    <table_el type="vector" dim="2" ind="2">
      <vector_el type="scalar" ind="1">0</vector_el>
      <vector_el type="scalar" ind="2">-1</vector_el>
    </table_el>
  </gradconstr>
  <!-- Optional definition data: -->
```

```
  <cd type="string">Definition data</cd>
</data>
```

Figure 13: Example of analysis output file in XML (currently not in use).

References:

[1] Introducing JSON. Electronic document at http://www.json.org/ .

[2] JSON on Wikipedia, http://en.wikipedia.org/wiki/JSON

[3] I. Grešovnik. IoptLib – Investigative Optimization Library, library homepage located at http://www2.arnes.si/~ljc3m2/igor/ioptlib/ .

[4] I. Grešovnik. IoptLib user's manual. Electronic document at http://www2.arnes.si/~ljc3m2/igor/ioptlib/doc/optlib.pdf.

[5] Igor Grešovnik: *Optimization program Inverse*, electronic document at http://www2.arnes.si/~ljc3m2/inverse/index.html.

[6] Igor Grešovnik: Coordination of software development in COBIK and Laboratory for Multiphase Processes. Treatiese, COBIK, 2011.

[7] Igor Grešovnik: *Programmers' guidelines for Development of Software within COBIK & Laboratory for Multiphase Processes*. Treatise, COBIK, 2012.

[8] Igor Grešovnik: *IGLib.NET*, electronic document at http://www2.arnes.si/~ljc3m2/igor/iglib/index.html.

[9] Igor Grešovnik: *IGLib Documentation*, electronic document at http://dl.dropbox.com/u/12702901/code_documentation/generated/iglib/index.html .

[10] Igor Grešovnik: *IGLib.NET Code Documentation*, electronic document at http://dl.dropbox.com/u/12702901/code_documentation/generated/iglib/html/index.html .