



Application of clouds for modeling uncertainties in robust space system design

Final Report

Authors: Arnold Neumaier*, Martin Fuchs*, Erich Dolejsi*, Tibor Csendes^,
Jozsef Dombi^, Balázs Bánhelyi^, Zsolt Gera^

Affiliation: *Fakultät für Mathematik, Universität Wien, ^ Institute of
Informatics, University of Szeged

ESA Research Fellow/Technical Officer: Daniela Girimonte

Contacts:

Arnold Neumaier

Tel: +431 4277 50661

Fax: +431 4277 50670

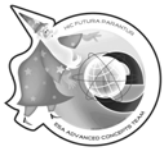
e-mail: Arnold.Neumaier@univie.ac.at

Daniela Girimonte

Tel: +31(0)71 5658888

Fax: +31(0)715658018

e-mail: act@esa.int



Available on the ACT website
<http://www.esa.int/act>

Ariadna ID: 05/5201
Study Duration: 4 months
Contract Number: 19703//06/NL/HE

Application of clouds for modeling uncertainties in robust space system design

Arnold Neumaier

Martin Fuchs

Erich Dolejsi

*Fakultät für Mathematik, Universität Wien
Nordbergstr. 15, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: <http://www.mat.univie.ac.at/~neum/>*

Tibor Csendes

Jozsef Dombi

Balázs Bánhelyi

Zsolt Gera

*Institute of Informatics, University of Szeged
H-6701 Szeged, P.O. Box 652, Hungary
email: csendes@inf.u-szeged.hu
WWW: <http://www.inf.u-szeged.hu/~csendes/>*

Final report

April 3, 2007

Contents

1	Study objectives	5
2	Material provided by ESA	5
3	Analysis of the material	7
3.1	Hidden features discovered during the analysis	15
4	Manual modifications	16
4.1	General modifications	17
4.2	Special preparations for DAG generation	17
5	Preprocessing, DAG generation and conversion	19
5.1	Preprocessing	19
5.2	DAG generation	20
5.3	Conversion to AMPL and LINGO	22
6	Multidimensional potential function based clouds	28
6.1	Calculation of a potential function based cloud from data . . .	30
6.2	Upper bound for density functions in continuous clouds	33
6.3	Illustrative examples	34
6.4	Examination of generated clouds	38
7	Interactive uncertainty elicitation	42
7.1	Using the interactive uncertainty elicitation software	45
7.1.1	The variable selection window	46
7.1.2	The probabilistic information panels	53

7.1.3	The output of the uncertainty elicitation interface . . .	57
8	The cloud demonstration applet	60
8.1	A real life example	68
9	Optimization problem formulation	70
9.1	Difficulties	71
10	Optimization	72
10.1	Symbolic solvers	72
10.1.1	Without Uncertainty	72
10.1.2	With Uncertainty: Sensitivity approximation in the smooth case	73
10.2	Heuristic approach	75
10.2.1	DAKOTA and other software packages	79
10.2.2	SNOBFIT-based heuristics	79
10.2.3	Matlab programs	83
11	Results	84
12	Robustness	95
12.1	Consequences of assuming different probability distributions .	99
13	Conclusions and future directions	100

Abstract. This report discusses a case study for modeling uncertainties in a Matlab model for space system design, with the goal of determining robust feasible designs for the model.

An initial analysis of the original model (provided without uncertainties) revealed a number of anomalies that were resolved. In addition, the problem structure revealed the presence of all difficulties an optimization problem can have: discrete variables, strong nonlinearities, discontinuities due to branching decisions, multiple objectives, multiple local minima.

To aid in providing reliable uncertainty information, an uncertainty elicitation interface (implemented in Matlab) is described. The uncertainty information actually provided by ESA consisted in independent worst case bounds (intervals) for a number of variables.

To be able to automatically insert the uncertainty information into the model, semiautomatic conversion tools into the modeling languages AMPL (cf. [6]) and LINGO (cf. [13]) are discussed.

The formal modeling of uncertainty by means of clouds, and the robust optimization of a resulting uncertain model was considered from a theoretical point of view, with a toy example demonstration. The sensitivity approach to the robust optimization requires tools from automatic differentiation which turned out not to be realizable for the model problem under discussion.

Instead, a heuristic approach using surrogate function modeling, corner searches, and discrete line searches was used to solve the robust optimization problem in case of interval uncertainties. This approach works satisfactorily for the model problem for the case of interval uncertainty.

Solving the model problem with uncertainty revealed significant robustness advantages of the approach using uncertainty. The influence on assumed knowledge about additional uncertainty information was illustrated by for a few model choices.

1 Study objectives

According to the study description 05/5201 in the call for proposals, the task of the project is to:

- Formulate sample space design applications in the design modelling frame of NEUMAIER [2];
- Investigate the possibility of modelling the random (uncertain) design variables by clouds;
- Quantify the kinds of available uncertainty information for the sample applications and study their reliability;
- Verify the robustness of the possible existing solutions of the sample design cases (e.g. by determining a proper failure probability value to the outcome of the design), and propose 'safe' designs (for given failure probabilities) as substitution solutions, by using global optimization and constraint satisfaction techniques.

2 Material provided by ESA

Apart from general background information and the answers to some questions, we obtained from ESA:

- An Excel document `COMPLETE_v02.xls` identifying variable names in the programs and input, output, control variables as used in the 'Instructions';
- A Word document `instructions.doc` with tables for the allowed values of the control parameters (design variables);
- Two Excel documents:
`initial_values.xls` and `initial_values_xeus.xls` giving two sets of initial values for the input and output variables. All the global input parameters are the result of a mission analysis study and will be considered given and fixed, apart from the uncertainties specified in the Excel document `Uncertainty_v01.xls`. It provides deterministic interval uncertainties for the global inputs.

- Eight m-files with Matlab functions for various subsystems, and the mass budget; complemented later by 3 more m-files with auxiliary programs, and finally (since more files were missing) with the whole (undocumented) Matlab implementation of the Manager system. Upon request of ESA we neglected the AOCS subsystem, and put to 0 the variables `mass_aocs` and `pow_aocs`.
- An m-file `TTC_cloud.m` with a small real life problem, from the area of satellite design.

The 'Instructions' mentioned that "in order to achieve compatibility with the existing tools, it would be very important to use the above mentioned subsystem model files and build the new tools around this core. If changes to these files can not be avoided, the issue should be brought to our attention."

Our first approach (using symbolic solvers) required changes to the program for two reasons:

1. The equations as given do not account for uncertainty, except in a rough form by means of safety factors. To take into account uncertainty, all uncertain equations will get an additional error terms.
2. To interface the programs to our optimization tools, we need to have the problem specified in a declarative rather than operational format. This means that we rewrote all control structures (if, while) and replaced them by mathematically equivalent equations involving binary variables.

We reverse engineered the programs to rediscover the relevant part of the conceptual level underlying the program, and generated semi-automatically streamlined versions of the programs that suit our purposes. A problem description on the conceptual level (which we had expected to get when writing our proposal), e.g., by detailed cross-references to LARSON & WERTZ et al. [1]) might have saved us some work.

All changes made are documented, and it will be ensured that they do not change the mathematical meaning of the equations. We also use the programs as they are to verify that our solutions are correct.

3 Analysis of the material

The modified version of the programs consists of nine m-files defining Matlab functions: six for the various subsystems, one for the mass budget, and two auxiliary functions.

The Attitude and Orbit Control System in file `AOCS_ss.m` was ignored by agreement.

id letter	file name	use
D	<code>DH_ss.m</code>	Data Handling System
T	<code>TTC_ss.m</code>	Tracking, Telemetry and Command
P	<code>pow_ss.m</code>	Power System
S	<code>str_ss.m</code>	Structure System
L	<code>str_stabil.m</code>	aux. for Structure System
G	<code>target_planet_func.m</code>	aux. for Thermal Protection System
H	<code>thermal_ss.m</code>	Thermal Protection System
R	<code>prop_ss.m</code>	Propulsion System
M	<code>mass_budget.m</code>	Mass Budget

In order to see what is going on and to check the correctness of our modifications, we wrote a program `testss.m` which encodes all initial values and the tables for the allowed choices of the design parameters, and evaluates these functions at randomly chosen global inputs within the specified uncertainty interval, and random choices for the design parameters.

The dependence of the m-files is shown in the following Figure 1; an entry is occupied by the letter of the corresponding column if the file whose id letter labels the row depends on results computed in the file whose id letter labels the column.

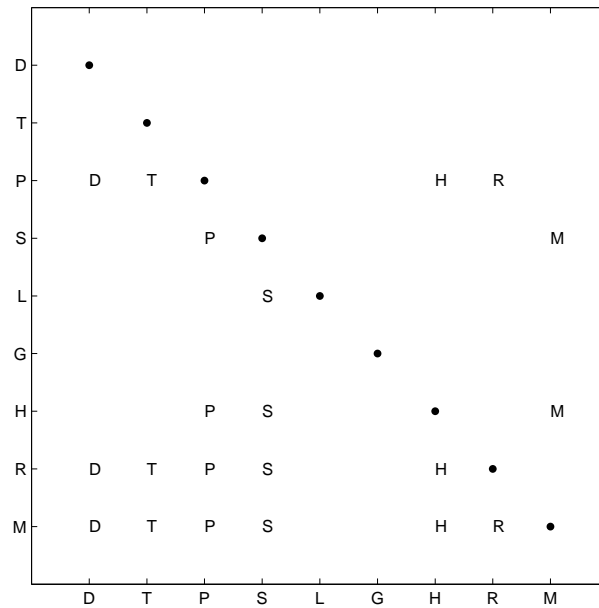


Figure 1: File dependence.

The problem defined by these files and the test program has 107 variables which are related by the equations defined in the above m-files. The 107 variables fall into five different classes:

- (F) 24 fixed variables,
- (G) 24 global input variables,
- (D) 24 design variables,
- (I) 19 intermediate result variables, and
- (O) 16 only result variables.

The fixed variables are fixed and certain for a given mission. Among them are a number of variables which have the value zero, which arise since the problem under discussion is in fact part of a bigger model in which these variables would get nontrivial values. There are 5 other fixed variables, 4 of which take discrete values only:

fixed variable	structure	meaning
target_planet	nonneg. integer	Target Planet id
count	nonneg. integer	Counter
elev	angle	Elevation
body_mount_SA	binary	Body Mounted Solar Arrays
primary	binary	Only Primary Source

The values of the fixed variables are fixed as follows:

a_ther	0
aff_DH	0
aff_TTC	0
aff_batt	0
aff_pow	0
aff_prop	0
aff_str	0
aff_tot	0
body_mount_SA	1
c_ther	0
cost_DH	0
cost_SA	0
cost_TTC	0
cost_batt	0
cost_pow	0
cost_prop	0
cost_str	0
cost_tot	0
count	1
elev	90
mass_aocs	0
pow_aocs	0
primary	0
target_planet	3

The remaining global input variables are fixed but uncertain for a given mission. We call a fixation of them a *scenario* for the global inputs.

The 24 design variables (23 original ones and one added by us, see Section 3.1) fall into 10 categories, each of which can be chosen independently and determines one or more design variables.

choice variable	used in	structure
θ_1	D	4 choices from \mathbb{R}^3
θ_2	T	14 choices from \mathbb{R}
θ_3	T	6 choices from \mathbb{R}
θ_4	T	8 choices from \mathbb{R}
θ_5	P	5 choices from \mathbb{R}^4
θ_6	P	20 choices from \mathbb{R}^3
θ_7	S	9 choices from \mathbb{R}^4 (cf. Section 3.1)
θ_8	H	44 choices from \mathbb{R}^2
θ_9	R	30 choices from \mathbb{R}^4
θ_{10}	S,P	continuous from $[0.5, 8.0]$

As described in Section 10.2.2 we have rearranged the tables as follows

mem	mem_mass	mem_pow	θ_1
8	2	6	1
64	2.5	7	2
64	3	8	3
128	3	14	4

θ_2	1	2	3	4	5	6	7	8	9	10	11	12	13	14
f	0.2	0.45	1.54	1.56	2.2	3	4	8	8.5	14	15.35	30	31.8	32.3

θ_3	1	2	3	4	5	6
D	0.2	0.7	1.1	1.56	1.7	2.44

θ_4	1	2	3	4	5	6	7	8
Eb	2.7	4	4.4	5.8	9.6	10.3	13.3	13.8

eta	d	ro	alfa	θ_5
0.14	2.27	2	0.65	1
0.19	1.9	2.84	0.89	2
0.22	0.87	2.84	0.92	3
0.25	1	3.14	0.92	4
0.27	1.61	3.14	0.92	5

spec	dens	eff	θ_6
30.1	84.12	0.72	1
32.9	107.01	0.72	2
34.3	98.26	0.72	3
35.2	103.98	0.72	4
37.3	107.17	0.72	5
38.6	115.03	0.72	6
42	61	0.7	7
43.39	76.89	0.7	8
50	69.3	0.7	9
50.4	74.65	0.7	10
50.9	88	0.7	11
52	49.8	0.7	12
52.8	78.8	0.7	13
52.9	82.8	0.7	14
53.4	83.7	0.7	15
90	175	0.96	16
120	259	0.96	17
124	283	0.96	18
129	303	0.96	19
145	327	0.96	20

El	rho	ult_str	yie_str	θ_7
45	1770	270	165	1
290	1856	320	290	2
68	2710	290	240	3
71	2800	460	380	4
72	2850	420	320	5
110	4430	900	855	6
196	7860	860	620	7
201	7940	970	660	8
203	8220	1280	1080	9

alfa_sup	eps_sup	θ_8
0.96	0.88	1
0.96	0.91	2
0.94	0.9	3
0.92	0.86	4
0.91	0.94	5
0.88	0.88	6
0.77	0.81	7
0.73	0.86	8
0.67	0.87	9
0.66	0.88	10
0.57	0.88	11
0.48	0.82	12
0.47	0.87	13
0.44	0.88	14
0.42	0.87	15
0.39	0.82	16
0.35	0.9	17
0.35	0.84	18
...		

alfa_sup	eps_sup	θ_8
0.28	0.87	19
0.2	0.9	20
0.17	0.92	21
0.09	0.9	22
0.32	0.02	23
0.23	0.03	24
0.31	0.03	25
0.44	0.03	26
0.09	0.03	27
0.26	0.04	28
0.33	0.04	29
0.4	0.05	30
0.39	0.07	31
0.52	0.1	32
0.7	0.13	33
0.29	0.3	34
0.12	0.45	35
0.55	0.46	36
0.43	0.49	37
0.44	0.56	38
0.26	0.58	39
0.15	0.59	40
0.38	0.67	41
0.92	0.72	42
0.97	0.75	43
0.17	0.76	44

I	m_eng	T_eng	P_eng	θ_9
3200	9	0.01	9	1
3600	10.3	0.01	10.3	2
3200	13	0.03	13	3
1500	9.9	0.08	9.9	4
1650	11.5	0.09	11.5	5
1700	15	0.12	15	6
4500	27	0.15	27	7
3470	26.7	0.15	26.7	8
227	0.19	0.75	0.19	9
226	0.33	1	0.33	10
227	0.2	2	0.2	11
230	0.38	4.5	0.38	12
228	0.22	6	0.22	13
295	0.41	10	0.41	14
230	0.24	10	0.24	15
295	0.45	22	0.45	16
235	0.52	22	0.52	17
234	0.36	24	0.36	18
235	1	90	1	19
280	2	110	2	20
234	1.8	350	1.8	21
306	2.5	404	2.5	22
235	1.6	445	1.6	23
315	4.8	450	4.8	24
320	4.7	460	4.7	25
315.5	3.6	490	3.6	26
328	6	530	6	27
303	4.5	890	4.5	28
239	5.8	2500	5.8	29
315	6.8	4000	6.8	30

θ_{10}	continuous
slimfactor	[0.5,8.0]

We treat θ_{10} (the slim factor `punt_h_vs_r`) as continuous parameter, although in the 'Instructions' it is restricted to 20 grid points in the specified range. We call a fixed θ a *choice* for the design variables.

The 'only result' variables only occur as result variables, hence are immaterial for the difficulty of the problem (except for `pow_tot`, which may influence the objective function).

All other variables are intermediate result variables, and occur both as input of some file and as output of some file.

Fixed point structure. Some of those are inputs before they are computed, that yields a recursive structure and leads to a fixed point problem. The sequence in which the subsystems are called determines the number of recursive variables and thus the dimension of the fixed point problem; the sequence should be chosen so that the dimension is minimal. With the manually chosen sequence as in Figure 1 we have a 3-dimensional fixed point problem size with the recursive variables `m_tot`, `pow_prop` and `pow_ther`.

3.1 Hidden features discovered during the analysis

During the analysis we discovered and resolved a number of peculiarities in the programs.

- Extra column in structure controls (in `hidden_data.m`) (cf. p.466 of [1])

E1	rho	ult_str	yie_str
72	2850	420	320
68	2710	290	240
71	2800	460	380
196	7860	860	620
201	7940	970	660
203	8220	1280	1080
45	1770	270	165
110	4430	900	855
290	1856	320	290

- Discontinuity and zero finding subproblem (in `str_ss.m`) resolved by rewriting `str_ss.m` and `str_stabil.m`; see Section 4.

- Restructured `target_planet_func.m`: separated constant and nonconstant parts. We also repaired a bug in cold case data of Venus: In place of `H_min_IR = H_target` it should read `H_min_IR = H_terra`.
- There was a typing error in `pow_ss`, lines 43 and 45: `teta0` has been replaced by `pi*(teta0)/180`.
- Undocumented pointer structure
The design variables are represented in the program as entries of a vector `x`. For example, `mem_mass` is recorded in `x(punt_mem_mass)`, with `punt_mem_mass` being a pointer containing the relevant index; similarly for the other design variables. Unfortunately, the indices used are nowhere documented. From close inspection of the Manager program we reconstructed the following probable pointer values:

```

punt_f           = 1;
punt_D           = 2;
punt_Eb          = 3;
punt_mem         = 4;
punt_mem_mass    = 5;
punt_mem_pow     = 6;
punt_eta         = 7;
punt_d           = 8;
punt_ro          = 9;
punt_alfa        = 10;
punt_spec        = 11;
punt_dens        = 12;
punt_eff         = 13;
punt_El          = 14;
punt_rho         = 15;
punt_ult_str     = 16;
punt_h_vs_r      = 17;
punt_alfa_sup    = 18;
punt_eps_sup     = 19;
punt_I           = 20;
punt_m_eng       = 21;
punt_P_eng       = 22;
punt_T_eng       = 23;

```

4 Manual modifications

We spent considerable time in writing support routines to help us transform the given programs reliably (and checkable) into a form that we can use,

without having to understand every detail of the programs.

4.1 General modifications

At first we did some generally important modifications:

- The design vector \mathbf{x} is no longer input for the files, but an identity function for the pointer variables. As a consequence the pointer variables now represent the value of the former component of \mathbf{x} they pointed at.
- In the file `prop_ss.m` the occurrence of `transfer_time` seems to be not necessary, so `transfer_time` has been removed as input, output and within the code of the file.
- As mentioned in Section 3.1 `str_ss.m` and `str_stabil.m` have to be rewritten. In `str_ss.m` `yie_str_ss` becomes `punt_yie_str_ss` and a new input variable for the file, thus the call of `hidden_data.m` can be omitted. `str_stabil` has been reshaped to a single line expression, the input variable `h` has been removed as it is never used by `str_stabil`.
- In `thermal_ss.m` two global values are commented out. As indicated above (cf. Section 3.1) `target_planet_func` is separated in a constant part `target_planet_func1` and a nonconstant part `target_planet_func2`. The calls to `radiaz_solare.m` are replaced by pasting the contents of `radiaz_solare.m` inline.

4.2 Special preparations for DAG generation

The goal here is to make the subsystem m-files interpretable by the DAG generator needed exclusively for the approach using symbolic solvers via the modeling system AMPL. But before the files can be passed to a preprocessor to modify them automatically, some changes were made manually:

- Control structures like "if" are removed in case they concern uncertain variables as AMPL lacks an implementation for them by now; then they are substituted by equivalent explicit expressions. E.g.,

```

if x<=1
    a=-27.5;
elseif x>1 & x<=2
    a=-28.4;
else
    a=-57.4;
end

```

is replaced by

```

controlsubs_a(1)=-57.4;

controlsubs_a(2)=-28.4;

controlsubs_a(3)=-27.5;

a=controlsubs_a(stepnum(x,1,2));

```

Thus the if-structures of the s subsystem m-files are handled by step functions `stepnum.m` (cf. Figure 2), `stepnum2.m` or `casenum.m`, which can be overloaded for the DAG.

- We have inserted newlines in the else-case because the preprocessor is not able to detect equals signs in lines starting with else (that would require an if-parser).
- The function calls to `str_stabil` in `str_ss` are replaced inline by the new single-line-expression. The function call to `fzero` has changed to a call to `fzero_str_stabil`, a new function that is overloaded for DAG inputs to create suitable constraints.
- We made some modifications to avoid semantic conflicts with the solver languages. In Matlab the equals-sign '=' is an assignment operator, while solver languages interpret '=' similar to the '==' in Matlab, left and right hand side are interchangeable. So in Matlab it is possible to overwrite variables like `x=10*x`, but the same expression as solver code constraint would implicate `x=0`. One has to find, rename and/or replace these variables if necessary, e.g. rename the overwritten `x` to `x2=10*x` and replace `x` in the following by `x2`. With regard to this problem we have modified `thermal_ss` for the variables `coseno`, `T_minr`, `pow_ther` and `str_ss` for the variables `punt_yie_str`, `punt_El`, `punt_ult_str` appropriately.

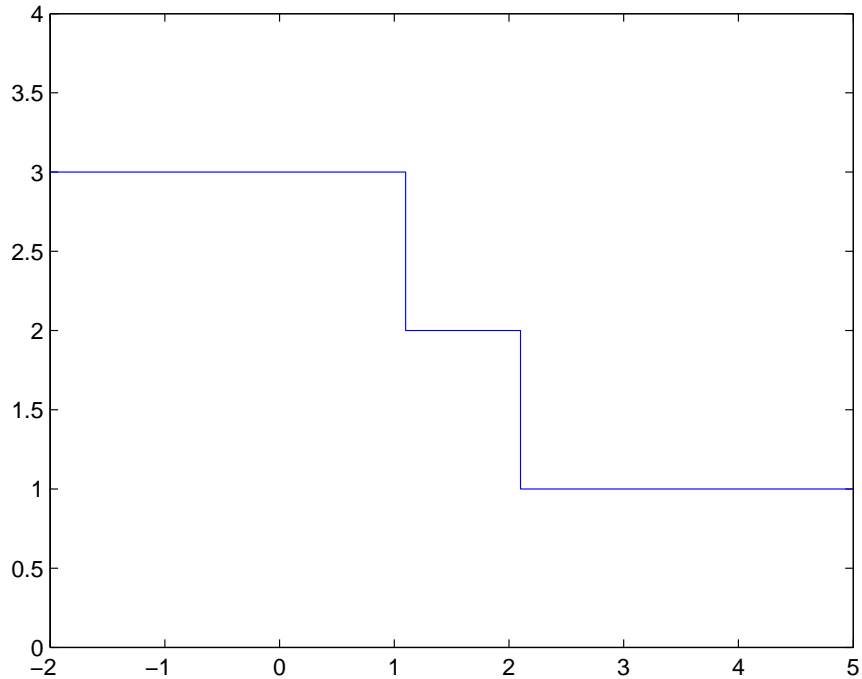


Figure 2: The step function $\text{stepnum}(x,1,2)$ with steps at 1 and 2.

5 Preprocessing, DAG generation and conversion

The conversion of the given Matlab model to an AMPL model enables the application of symbolic solver software on the optimization problem (cf. Section 9). After the changes in Sections 4.1 and 4.2 the subsystems should be ready for this conversion. We create an AMPL model to be flexible in our choice of a solver as we have several converters from AMPL to different solver languages. Moreover the syntax in AMPL is quite intuitively comprehensible, which is a significant advantage concerning readability.

5.1 Preprocessing

After the changes in Sections 4.1 and 4.2 the program `preprocessor.m` reads the modified files. It removes all comments, triple dots and print commands

from each file. If any control structure is still found the preprocessor prints a warning to make it easier to remove it manually.

Apart from that the preprocessor tracks equals signs and replaces the right hand side by a function call of `equation`. Example:

```
mass_pow=(mass_SA+mass_batt)*1.2;
```

reads in the preprocessed version

```
mass_pow=equation('mass_pow', 'Linenumber 91', 'pow_ss',...  
(mass_SA+mass_batt)*1.2);
```

`equation` will be overloaded for the DAG, and should later on deal with the uncertainty information.

`stage1.m` includes all the results of both manual modification and preprocessing. In addition to that it provides detailed information about the structure of the files and their variables (cf. Section 3).

5.2 DAG generation

To represent the mathematical structure of the complete model we use a directed acyclic graph (DAG) representation of the computational flow. This representation allows one to perform automatically a number of important problem transformations (including writing equivalent Matlab or AMPL code, and generating code for partial derivatives and interval enclosures). For details, see, e.g., SCHICHL & NEUMAIER [4].

The current version of the DAG generator (parser) is able to generate a DAG from the preprocessed files in absence of `subsasn` and `subsindex`.

The DAG is both a class in the MATLAB sense and a cell array which defines a tree (a special form of a DAG).

Let's have a look at the following simple Matlab function:

```
function z=held(x,y)  
    z=x^y
```

This function takes 2 inputs x, y and puts a variable z to the caller's workspace.

The input variables form two groups: precise and imprecise variables. The precise ones are simply arrays of doubles, while the imprecise ones are variables of class **DAG**.

Some equations like

$$z=x^z$$

could include uncertainty information in the form of safety factors

$$z=[0.8, 1.2]*x^z$$

or additive constants

$$z=x^z+-0.003.$$

Therefore it is necessary to overload equations to find possible sources of uncertainty.

The preprocessed files will be called using DAG and double variables. When calculations are performed solely with doubles then the results are again precise data (at least in this version of the DAG). When a function will be called with doubles and DAGs then the double information will be copied to a special field called `value`

The classical Matlab datatypes cannot store information about uncertainty. Therefore a new datatype, the 'DAG' class, is created to propagate these information. Normally, integer variables are precise and floats are imprecise. When imprecise equations occur, an additional source of imprecise data (from the DAG class) is created.

A typical **DAG** node has the form

```

DAG{157} =
    base: '0'  base is 1 when a newnode will be created
    child: []  not set
    filename: ''  for equation to use in the gui
    left: 153  index of the left part of the binary operator
    linenum: ''  the linenum of the equation in filename only for
                equation
    name: ''  name of the variable empty when intermediate
              variable only for equation
    op: 25  type of operation
    pos: 157  position in the DAG
    right: 156  index of the right part of the binary operator
    value: NaN  value of variable NaN when no value is available

```

In general, the `value` can be a general array.

`y=newnode(x,yadd)` creates a DAG variable and enlarges the DAG (cell array).

`yadd` overwrites the standard settings of `newnode`.

`overload` overloads the operators.

5.3 Conversion to AMPL and LINGO

To create the AMPL model we make use of our generated DAG.

The conversion to AMPL will be done automatically.

Some conversions from Matlab to AMPL are very complex because AMPL lacks the support of most of the Matlab functions, e.g. `subsref`, `ceil`, `floor` and `log10`.

The language conversion is difficult because the semantics of Matlab differs from AMPL.

In most overloaded functions the AMPL code will be created directly, some files like `ceil2ampl`, `casenum2ampl`, `stepnum2ampl` are auxiliary functions for this purpose.

The following tabular illustrates the overloaded functions for the DAG. In some cases overloading leads to a non-trivial problem formulation for the solver constraints. On the other hand some overloaded operators are not in use yet.

DAG object	non-trivial	Description
and.m		
asin.m		
casenum.m	yes	aux. fct. casenum2ampl
ceil2ampl.m		obsolete function
ceil.m	yes	aux. fct. ceil2ampl
colon.m		
controlsubstant.m		obsolete function
cos.m		
ctranspose.m		
dag.m		
display.m		special output for Matlab internally, this is a special version for developers
disp.m		the same like display, but function differs
double.m		special, has no impact on the AMPL file
eq.m		
equation.m	yes	the most important function, both double and DAG objects are implemented in a special way
exp.m		
fzero_str_stabil.m	yes	cf. Section 4.2
ge.m		
generatesubrefampl.m	yes	obsolete function
get.m	yes	Matlab only
gt.m		
horzcat.m	yes	special, creates an AMPL vector, written by equation
...		

DAG object	non-trivial	Description
ldivide.m	yes	not yet needed
le.m		
log10.m	yes	similar to Matlab log10 but not identical for AMPL
logical.m	yes	not yet needed
log.m		
lt.m		
max.m	yes	max only for 2 doubles or integer, 2 input variables version
min.m	yes	min only for 2 doubles or integer, 2 input variables version
minus.m		
mldivide.m	yes	not yet needed
mplus.m		
mpower.m		
mrdivide.m		
mtimes.m		
ne.m		
not.m		
or.m		
overload.txt		
plus.m		fundamental function for overloading
polyval.m	yes	polynomials will be written by this function verbatim to the AMPL file
power.m		
rdivide.m	yes	not yet needed
...		

DAG object	non-trivial	Description
round.m	yes	this is a dag2gams problem, so it needs to be overloaded
sin.m		
sqrt.m		
stepnum2.m	yes	aux. fct. stepnum22ampl, special function to resolve ifs
stepnum.m	yes	aux. fct. stepnum2ampl, special function to resolve ifs
subsasgn.m	yes	not yet needed
subsindex.m	yes	very special but simple, concerns only integer or doubles, no vectors
subsref.m	yes	very difficult for AMPL, but no problem for LINGO (different rounding : always truncate)
times.m		
transpose.m	yes	not yet needed
uminus.m		
uplus.m		
vertcat.m	yes	not yet needed
x.m		

The following function are essential for the DAG generation:

ceil2ampl.m	yes	for ceil
equation.m	yes	equation for only double equations
horzcatextra	yes	needed by equation
makeampl.m		
newnode.m		this is the constructor for the class DAG
overload.m		important to make the files in the @dag directory
subsys.m		creates the AMPL code
subtest.m		creates the AMPL code for the test cases
update.m		not yet needed for creation of the DAG

ceil generates for $y = \text{ceil}(x)$:

```

var y integer, >=-10^50,<=10^50;
s.t. ans_ceil_1 :(y - x)>=0;
s.t. ans_ceil_2 :(y - x) <=1-eps);

```

round generates constraints in an analogous manner.

horzcat (double, for DAG analogically) generates for $m = [4.5, 4, 2]$:

```

subject to doublehorzcatvar_73_1: m[1]= 4.5 ;
subject to doublehorzcatvar_73_2: m[2]= 4 ;
subject to doublehorzcatvar_73_3: m[3]= 2 ;

```

Subreferencing in AMPL is created like this:

```

var y >=-10^50,<=10^50;
var b {i in 1 .. 3} binary;
var x integer, >=1, <=10^50;
subject to const1 :
sum{i in 1 .. 3} b[i]=1;
subject to const2 {i in 1 .. 3}:
b[i]*(x -i)=0;
subject to const3 {i in 1 .. 3}:
b[i]*(y-m[i])=0;

```

means $y = m[x]$, the entry of the vector m at x .

The if-structures, as mentioned in Section 4.2, are handled via step functions to be overloaded.

As an example, to model the stepfunction from Figure 2 we create the constraints:

```

subject to stepfct_1_1: (b[1]-1)*(x-2)>=0;
subject to stepfct_1_2: b[1]*(x-2)>=0;
subject to stepfct_2_1: abs(b[1]-1)*(b[2]-1)*(x-1)>=0;
subject to stepfct_2_2: b[2]*(x-1)>=0;
subject to stepfct_3_1: (b[3]-1)*(x-1)<=0;
subject to stepfct_3_2: b[3]*(x-1)<=0;
subject to stepfct_1: sum {i in 1 .. 3} b[i]=1;
subject to stepfct_2: stepfct=sum {i in 1 .. 3} b[i]*i;

```

for the variables

```
var x;  
var b {1 .. 3} binary;  
var stepfct integer, >=1;
```

`stepfct` is the value of the step function at `x`.

In addition to the DAG evaluation we take the results of our structural analysis to initialize sets, parameters, the design tables, variables, design constraints and boxes for variables. Boxes have to be declared for all variables of the model as they are required by one of the converters and some solvers.

The generated AMPL code is initially written to various files:

```
ampl.equation  
ampl.param  
ampl.var  
sets  
params  
paramsdata  
vars  
st  
dispvars.
```

All the constraints, initializations, declarations, assignments are finally put together and aligned to three files `s1.mod`, `s1.dat` and `s1.run`. `s1.run` can be run in AMPL, `s1.mod` satisfies the requirements of the converters `ampl2dag` and `dag2gams` taking the output DAG of `ampl2dag` (these converters are provided within the Coconut environment). Thus we are able to create a GAMS file from our AMPL model.

The AMPL syntax does not allow multiple declarations - but needs at least one - for sets, parameters, variables and constraints, so the model is checked after its generation for missing or multiple declarations and appropriately modified afterwards.

Besides we generate a modified DAG for the questionnaire gui (cf. Section 7.1) with the option to choose the set of variables the questionnaire will have to deal with. We implemented line intervals for the if structures to improve the reference to the original files shown by the gui textviewer.

As mentioned above we are able to create a GAMS file from our AMPL model. Unfortunately initializations of the variables are lost during the conversion, so we have written a program to re-set them automatically and additionally prepare the GAMS file for a possible conversion to LINGO (this converter is built-in for GAMS) - at the current stage LINGO10 is our preferred solver.

6 Multidimensional potential function based clouds

The concept of clouds was introduced by NEUMAIER in [2]. It is a new notion for handling uncertainty.

Clouds allow the representation of incomplete stochastic information in a clearly understandable and computationally attractive way.

They describe the rough shapes of typical samples of various size, without fixing the details of the distribution.

The use of clouds permits a worst case analysis without losing track of important probabilistic information.

All computed probabilities – and hence the resulting designs – are safeguarded against perturbations due to unmodelled (and unavailable) information.

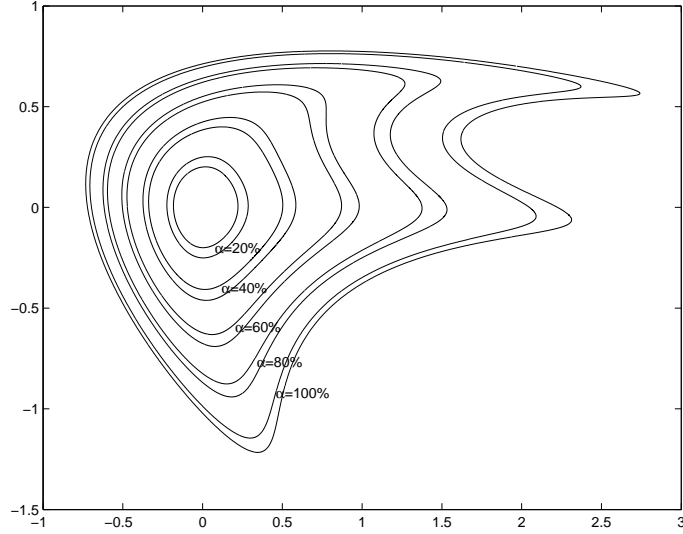


Figure 3:

The special case of interest for large-scale models is a **confocal cloud** defined by a continuous **potential** V which assigns to each scenario x a potential function $V(x)$ defining the shape of the cloud, and a **lower probability** $\underline{\alpha}(U)$ and an **upper probability** $\bar{\alpha}(U)$ defining the fuzzy boundary of the cloud, such that, for all U ,

$$\underline{\alpha}(U) \leq \Pr(V(x) < U) \leq \Pr(V(x) \leq U) \leq \bar{\alpha}(U).$$

$\underline{\alpha}$ and $\bar{\alpha}$ must be strictly increasing continuous functions of U mapping the range of V to $[0, 1]$. This corresponds to the level function $\mathbf{x}(\xi)$ defined by $\underline{x}(\xi) = \underline{\alpha}(V(\xi))$, $\bar{x}(\xi) = \bar{\alpha}(V(\xi))$.

For a given failure probability ε and $\alpha = 1 - \varepsilon$, the so-called **α -cut** describes an inner region \underline{C}_α of **α -relevant** scenarios with $V(x) < \underline{U}_\varepsilon$ and a (generally larger) region \bar{C}_α of **α -reasonable** scenarios with $V(x) < \bar{U}_\varepsilon$, where

$$\bar{\alpha}(\underline{U}_\varepsilon) = 1 - \varepsilon, \quad \underline{\alpha}(\bar{U}_\varepsilon) = 1 - \varepsilon.$$

The conditions defining the cloud guarantee that $\underline{U}_\varepsilon \leq \bar{U}_\varepsilon$, and that there is a region C with $\underline{C}_\alpha \subseteq C \subseteq \bar{C}_\alpha$ containing a fraction of α of all scenarios considered possible.

The potential determines the shape of the cloud. In particular,

$$V(x) = \max_k |x_k - \mu_k| / r_k$$

defines **rectangular clouds** (a sort of fuzzy boxes),

$$V(x) = \|Ax - b\|_2^2$$

defines **elliptical clouds** (a sort of fuzzy ellipsoids) The worst case bounds provided by ESA correspond to rectangular clouds. Elliptical clouds are appropriate in many situations, too.

This Section deals with clouds based on potential functions, specifically with their computation, properties, and their illustration through examples.

6.1 Calculation of a potential function based cloud from data

The direct calculation of a cloud, based on a general multidimensional potential, is very difficult. To this end, this Section presents a method which reduces computational complexity by representing a cloud by an ellipsoidal potential function and an interval-valued potential function.

Suppose we have n sample points $\mathbf{x}_1, \dots, \mathbf{x}_n$. Each sample point represents a possible setting. Let μ be the sample mean and let C be the covariance matrix of the sample. Let R be the upper triangular Cholesky factor of C^{-1} , i.e.

$$R^T R = C^{-1}.$$

Assume we have a quadratic potential function

$$V(\xi) = \|R(\xi - \mu)\|^2.$$

Theorem 4.3 in [2] defines the potential based cloud as

$$\mathbf{x}(\xi) := [\underline{p}(V(\xi)), \bar{p}(V(\xi))],$$

where (Pr denotes "probability of")

$$\underline{p}(u) = Pr(V(x) > u), \quad \bar{p}(u) = Pr(V(x) \geq u)$$

and $\underline{p}, \bar{p} : \mathbb{R} \rightarrow [0, 1]$ are the potential level maps. These are approximated using the Kolmogorov-Smirnov distribution in the following way.

Let $\alpha \in (0, 1)$ be a confidence level (e.g. 0.05 to allow 5% error), let F be the hypothetical cumulative distribution function and let F_n be the empirical

distribution function. The Kolmogorov-Smirnov test decides whether F_n and F can be considered the same. This decision is made according to a predefined threshold level $c = H^{-1}(1 - \alpha)$, where H is the Kolmogorov-Smirnov function

$$H(t) = 1 - 2 \sum_{i=1}^{\infty} (-1)^{i-1} e^{-2i^2 t^2}.$$

The hypothesis $F = F_n$ is accepted if

$$D_n = \sup_{x \in \mathbb{R}} |F_n(x) - F(x)| \leq c/\sqrt{n}.$$

Remark. Note, that there exists a better upper bound than c/\sqrt{n} , but it has the same order of magnitude.

The worst case is when for all x , $|F_n(x) - F(x)| = c/\sqrt{n}$. Hence the two "worst" distributions which are accepted are

$$\begin{aligned} \bar{p}(V(\xi)) &:= \min(1, 1 - (\bar{F}_n(V(\xi)) - c/\sqrt{n})), \text{ and} \\ \underline{p}(V(\xi)) &:= \max(0, 1 - (\underline{F}_n(V(\xi)) + c/\sqrt{n})), \end{aligned}$$

where

$$\begin{aligned} \bar{F}_n(t) &= Pr(V(\mathbf{x}) < t) = \frac{1}{n} \sum_{i=1}^n I(V(\mathbf{x}_i) < t), \\ \underline{F}_n(t) &= Pr(V(\mathbf{x}) \leq t) = \frac{1}{n} \sum_{i=1}^n I(V(\mathbf{x}_i) \leq t). \end{aligned}$$

The Matlab functions realizing the above described algorithm are as follows.

```
function out = V(xi)
% the value of the potential function at xi
% input: R, M (mean) describing the potential, xi
% output: potential of xi, "distance from M"
    out = R*(xi-repmat(M,1,size(xi,2)));
    out = sum(out.^2,1)';
end

function out = FnUpper(x) % 1x1 -> 1x1
% the distribution of the potential values
```

```

% calculated from the potential values of the sample points
% the number of strictly smaller values / number of values
    out = 0;
    for k0 = 1:nSamples
        if (VSamples(k0) < x)
            out = out + 1;
        end
    end
    out = out / nSamples;
end

function out = FnLower(x) % 1x1 -> 1x1
% the distribution of the potential values
% calculated from the potential values of the sample points
% the number of less or equal values / number of values
    out = 0;
    for k0 = 1:nSamples
        if (VSamples(k0) <= x)
            out = out + 1;
        end
    end
    out = out / nSamples;
end

function out = pUpper(vxi)
    out = max(0, min(1, 1-(FnUpper(vxi) - c/sqrt(nSamples))));
end

function out = pLower(vxi)
    out = max(0, min(1, 1-(FnLower(vxi) + c/sqrt(nSamples))));
end

function [l u] = cloudi(xi)
    l = pLower(V(xi));
    u = pUpper(V(xi));
end

```

These functions are utilized by a Matlab script, which calculates the cloud itself.

```

nSamples = 1000;
ALPHA = 0.005;
c = kolminv(1-ALPHA); % Kolmogorov-Smirnov constant

samples = generateSamples(nSamples, 2); % n x nSamples matrix
C = cov(samples'); % n x n, symmetrical
M = mean(samples')'; % n x 1 vector
R = chol(inv(C)); % n x n, upper triangular matrix

% the potentials of the sample points
VSamples = V(samples);

% the value of the cloud at (u,v)
u = .5; v = -.3;
x = cloudi([u; v]);

```

To sum up, a cloud is represented by two functions. The importance of the potential function is that it reduces the dimensionality to 1, so that the Kolmogorov-Smirnov test can be applied. The selection of the potential function is arbitrary, it affects basically the shape of the cloud. Here, machine learning algorithms could be applied in order to determine the best potential function for a given problem. In view of the lack of sample information in the data provided by ESA we did not pursue this possibility, and consider only ellipsoidal clouds. The interval-valued distribution functions determine the "thickness" of the cloud, which in fact depends on the number of sample points n and the confidence value α selected.

6.2 Upper bound for density functions in continuous clouds

Let \mathbf{x} be a continuous cloud. According to Proposition 5.1 in [2], the bounds for the density function of a random variable x in \mathbf{x} can be calculated as follows. Suppose A is an infinitely small subset of M . Let $\mathbf{x}(A) = [a_l, a_u]$. Then by Equation (22) in [2],

$$Pr(x \in A) \in [0, \min(a_u, 1 - a_l)].$$

6.3 Illustrative examples

In this subsection we provide several examples which show the use of the above described algorithm. For the sake of simplicity we assume two dimensional data. Samples are generated according to either uniform or a normal distribution with fixed mean on $[-5, 5]^2$. Figures 4-6 show the steps of the construction method with an initial sample set size 100, normal distribution, and 99.5% confidence. Similarly, Figures 7 and 8 shows the cloud construction method from sample points with uniform distribution.

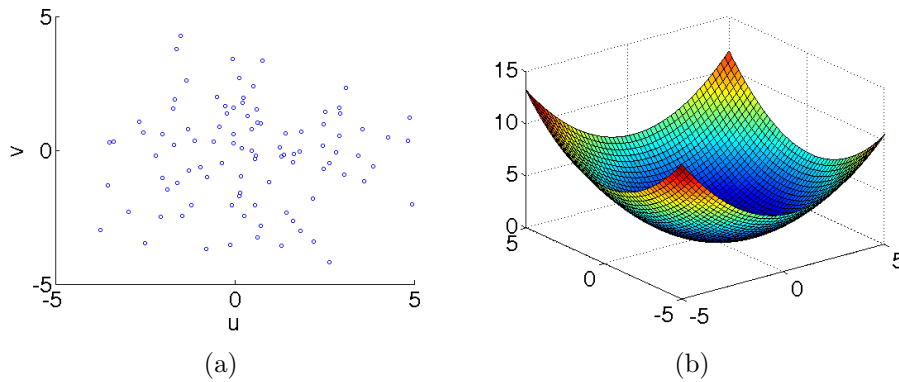


Figure 4: (a) 100 sample points with normal distribution. (b) Quadratic potential fitted on the sample.

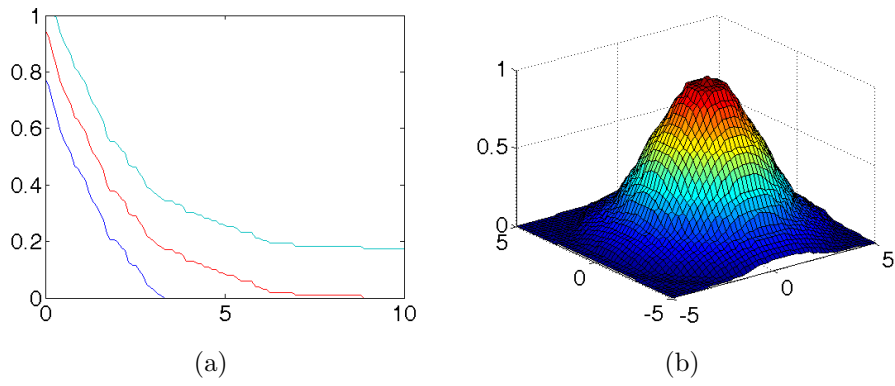


Figure 5: (a) in red: the transformed distribution function of the potential values; in blue and cyan: the approximated \underline{p} and \bar{p} functions with 99.5% confidence level. (b) the distribution function according to the original sample points.

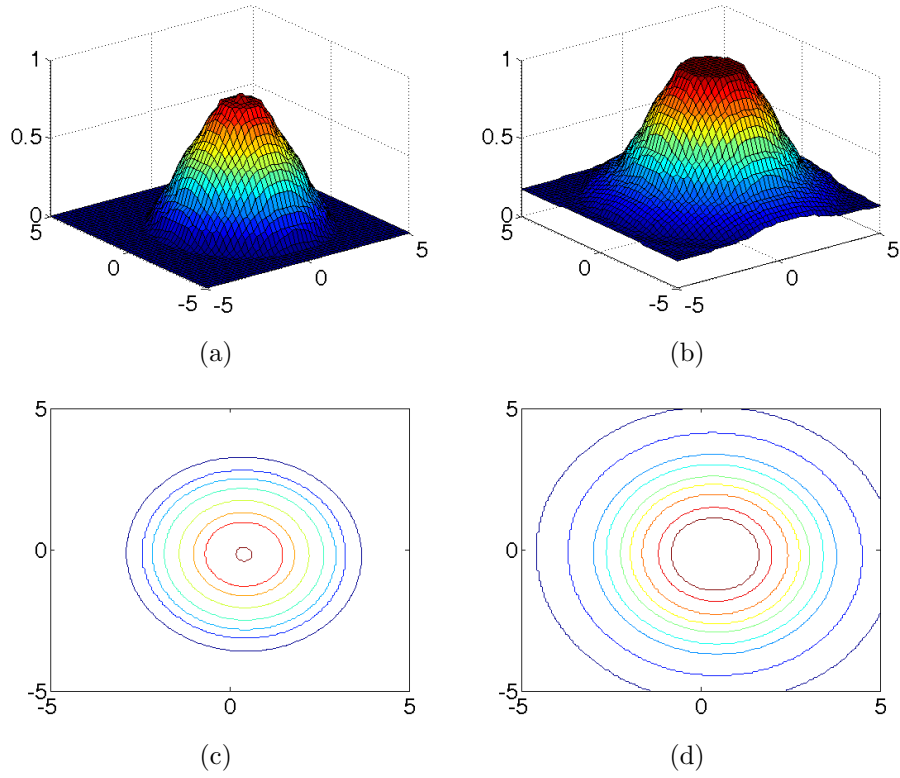
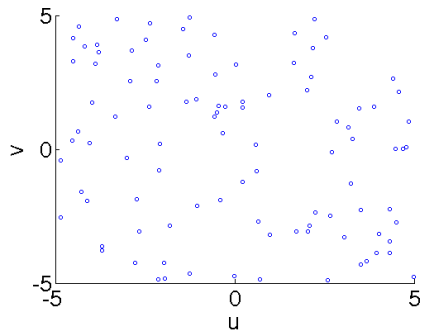
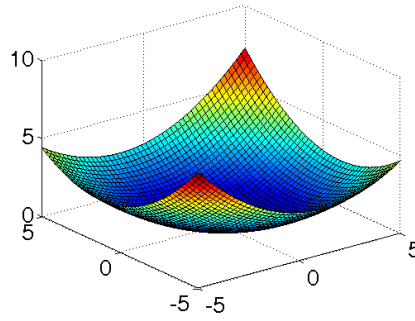


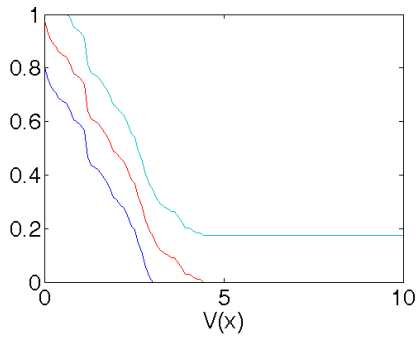
Figure 6: Subfigures (a) and (b): the functions $\underline{p}(V(\xi))$ and $\bar{p}(V(\xi))$, respectively. Subfigures (c) and (d) show the corresponding level sets of (a) and (b).



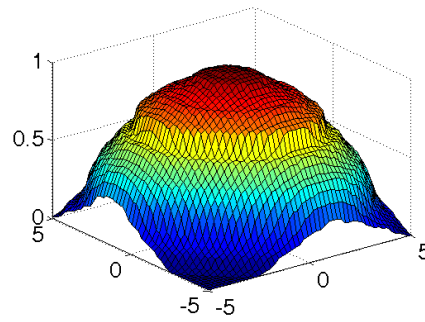
(a) sample points with uniform distribution



(b) quadratic potential fitted to the sample points



(c) transformed empirical and approximated potential distributions



(d) distribution of the original sample

Figure 7: Steps of the cloud construction from data with uniform distribution. Part I.

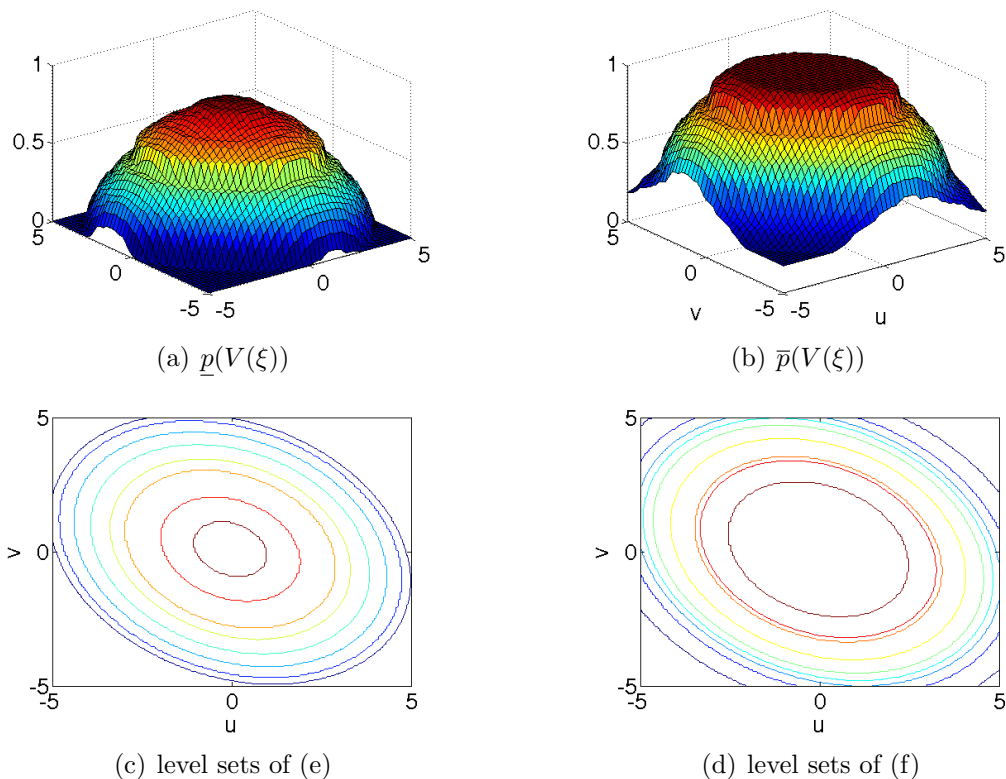


Figure 8: Steps of the cloud construction from data with uniform distribution. Part II.

6.4 Examination of generated clouds

The number of sample points and the confidence level heavily affect the shape and thickness of the constructed cloud. In fact, the thickness of the cloud is determined only by the number of sample points and the confidence level. As a consequence of the Kolmogorov-Smirnov method, this thickness is between c/\sqrt{n} and $2c/\sqrt{n}$.

The present subsection shows the influence of these parameters. Figures 9-12 illustrate the cloud construction method with various sample set sizes and confidence levels.

Note, that as the number of sample points increases, the cloud gets thinner. Three examples are shown. If the number of sample points is low (in our example $n = 10$, see Figure 9), then the cloud is thick, and the mean of

the sample cannot be determined with a good confidence. An example with $n = 100$ was illustrated on Figures 4-6. If the number of sample points is increased to $n = 1000$ (as on Figure 10), then the cloud is apparently more "smooth". It still has tiny jumps, of course.

The change in the confidence level has less effect. As it increases, the cloud gets thicker. The alterations induced by increasing the confidence value, can be followed on Figure 11, Figures 4-6, and Figure 12, respectively.

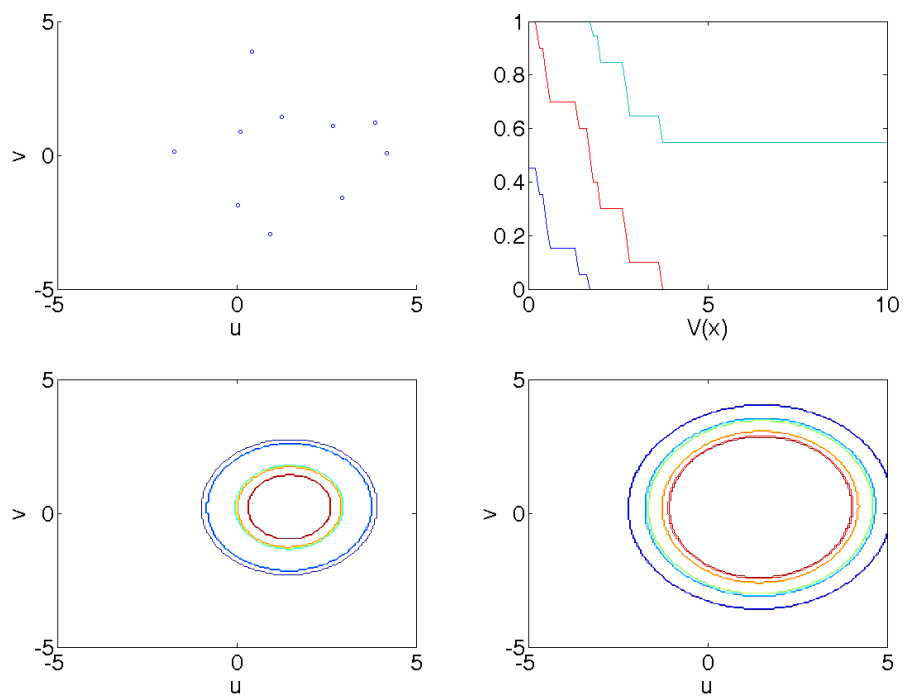


Figure 9: Cloud construction from 10 sample points with normal distribution and 99.5% confidence level.

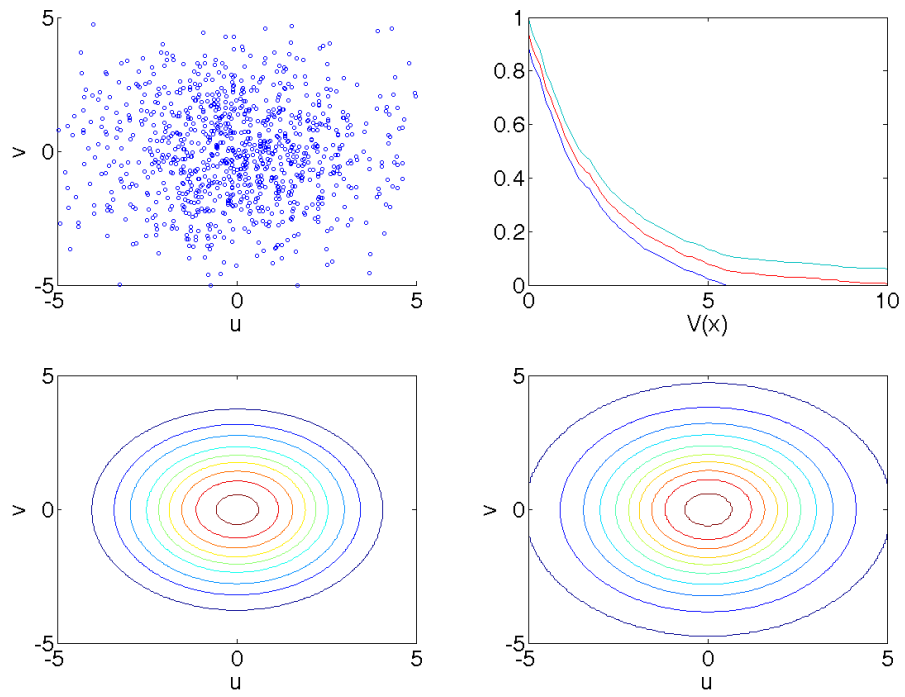


Figure 10: Cloud construction from 1000 sample points with normal distribution and 99.5% confidence.

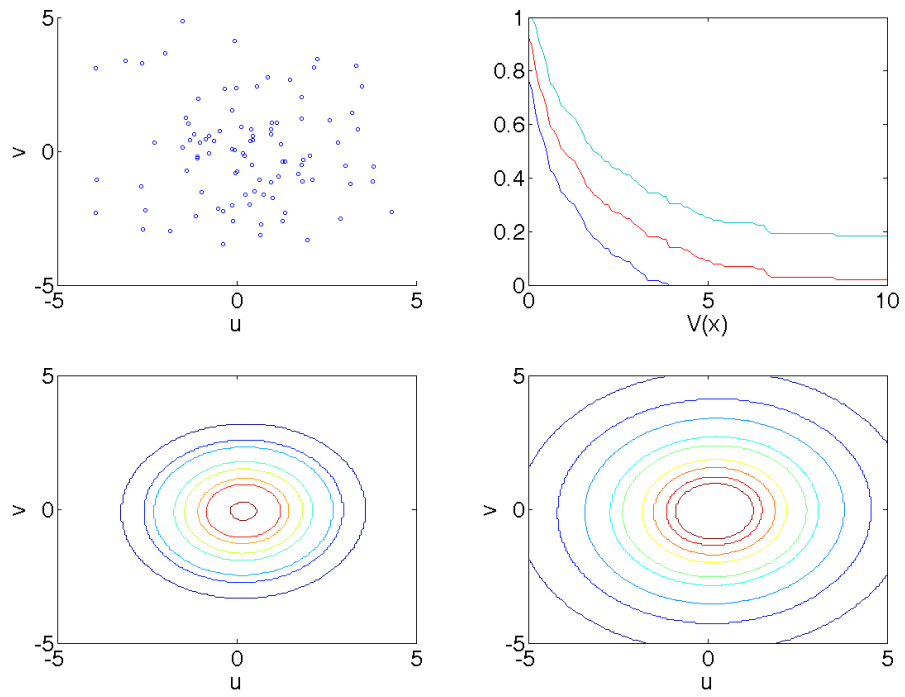


Figure 11: Cloud construction from 100 sample points with normal distribution and 99% confidence.

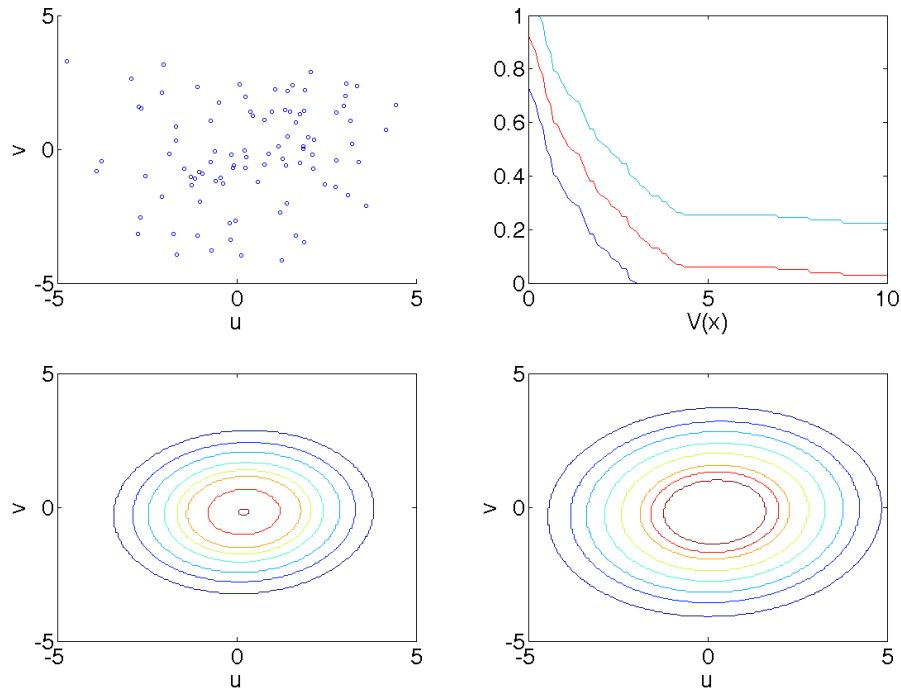


Figure 12: Cloud construction from 100 sample points with normal distribution and 99.9% confidence.

7 Interactive uncertainty elicitation

As possible sources of uncertainty we detected the following:

- Inputs
- Equations
- Safety margins
- Table entries (for design variables)
- Regression coefficients (in the context of `polyval`)
- Other constants

For a reliable uncertainty analysis and associated robustness measures, a complete specification of the extent of uncertainty is mandatory. This Section is intended to help specify completely and realistically uncertainties in a mathematical model. An interactive (and enhanced) version of this questionnaire is available in Matlab.

It is important to realize that any analysis based on assumed knowledge which is in fact not available results in spurious predictions.

Thus it is important to be reasonably sure of the uncertainty specifications made.

It is better to specify less with confidence than to specify unknown details based on pure guesswork.

The strength of the cloud approach is that it can utilize such partial information, in contrast to worst case analysis, which ignores probabilistic knowledge, and to traditional fully probabilistic methods, which require to know exactly the probability distribution.

Uncertainty comes in various forms, among which the following are typical:

1. **Worst case bounds on uncertainties.** E.g., a design parameter may be known to be realizable within 10% relative accuracy of the nominal design value, and no knowledge about relative frequencies is available >from past experience.
2. **Probability distribution.** E.g., a parameter has been frequently measured in the past and is known to take random values with a Gaussian distribution with given mean and standard deviation, or with another known distribution.
3. **Families of probability distributions.** E.g., a parameter is known to be typically Gaussian distributed but its mean and/or standard deviation may not be known reliably.
4. A **sample of typical values, intervals** representing expert knowledge, or various other incomplete ways of describing available information about the uncertainty.
5. **Qualitative information** only, such as expert opinions expressed in ordinary language.

6. **Simulation techniques** which provide a number of plausible scenarios, i.e., values for certain groups of parameters, under conditions that can be controlled by the simulation program.

In addition, information about the **dependence** or independence of different uncertain variables may be available in the form of

- bounds on correlations,
- qualitative statements (such as "x and y vary at a similar rate"),
- joint probability distributions, or
- structural independence of groups of variables.

There may also be uncertainty in the goals, which would have to be specified similarly.

Examples of informal uncertainty .

The interface discussed below contains text fields in which verbal specifications of uncertainty can be given. These are not automatically translated but give hints on which information to automatize next. Here we give some examples of informal specifications of uncertainty.

- In the equation $y = \cos(7.0x) * 1.1$, the constant 7.0 is accurate to 10% (to ± 0.01 , etc.) in the worst worst case; 1.1 is a safety factor which represents a reasonable upper bound for a number t which may be regarded as lognormal with $\text{var}(\log t) = 0.03$.
- The input variable z is 17 ± 3 , but smaller values are more likely than larger ones.
- Three typical values for z are 15, 16, 17; rare values would be 14, 18, 20, and values outside [12,25] are impossible.
- Three experts estimated the possible range of x as [17, 20], [15, 20], [16, 18]. The first expert has twice as much experience than the other two.
- A sample of 20 values for (x, y) is given by [...]
- Based on a sample of 500 values, the distribution of x is compatible with $N(22.5, 1.7)$.

- Variables x and y are positively correlated (are independent).
- Variables x and y are at most weakly correlated, with a correlation coefficient in $[-0.2, 0.2]$.
- The components of x have mean $\mu = \dots$ and covariance matrix $\Sigma = \dots$

If different experts differ in their assessment, conflicting information may be worth recording.

Invent (and characterize) further categories as needed, if similar situations occur over and over again. If the assignment is itself uncertain, put the type in parentheses, e.g., (NA) for uncertainty which is probabilistic but not quite normally distributed, or (BM) for uncertainty which is nonprobabilistic but where the bounds are uncertain. Explain if possible.

In the irregular case (e.g., other distributions, or fuzzy information, or subjective judgments), some explanatory text describing the form the uncertainty takes should be given if possible. (We shall later query the information available in these cases by asking for typical alternative values, extreme values, etc.)

7.1 Using the interactive uncertainty elicitation software

The crucial part of collecting information on an unknown problem is the questioning of the area experts on uncertainties of the variables taken into consideration. This is not an easy task, for the following reasons. Usually the experts have very specific knowledge of their areas. More importantly, the structure of this knowledge can be very different from the applied model. Furthermore, the experts usually do not and need not have explicit knowledge of the complete model.

Independent to the modeling framework, a good questionnaire asks such questions that the experts can answer. It shall not refer to the model of the problem nor shall use its notions, but preferably use the language of the experts, or at least stick to a common framework. In our case it means that the concept of clouds is not referred to explicitly in the questionnaire. We have chosen probability theory notion as the intermediate language used between the experts and the model. Experts are requested to answer questions

using probabilistic notions such as mean, variance or correlation. Probability theory is a good choice to be a mediator, since it is widely known and understood, and has a strong relationship to the concept of clouds.

7.1.1 The variable selection window

The uncertainty elicitation uses two windows. The variable selection window and the window of probabilistic information panels. In this section the variable selection window is presented. Figure 13 shows the initial state of the window. The set of variables can be loaded with the "Load all" button. This loads all probabilistic information contained in a given mat-file. Analogously, the "Save all" button saves all probabilistic information in a mat-file.

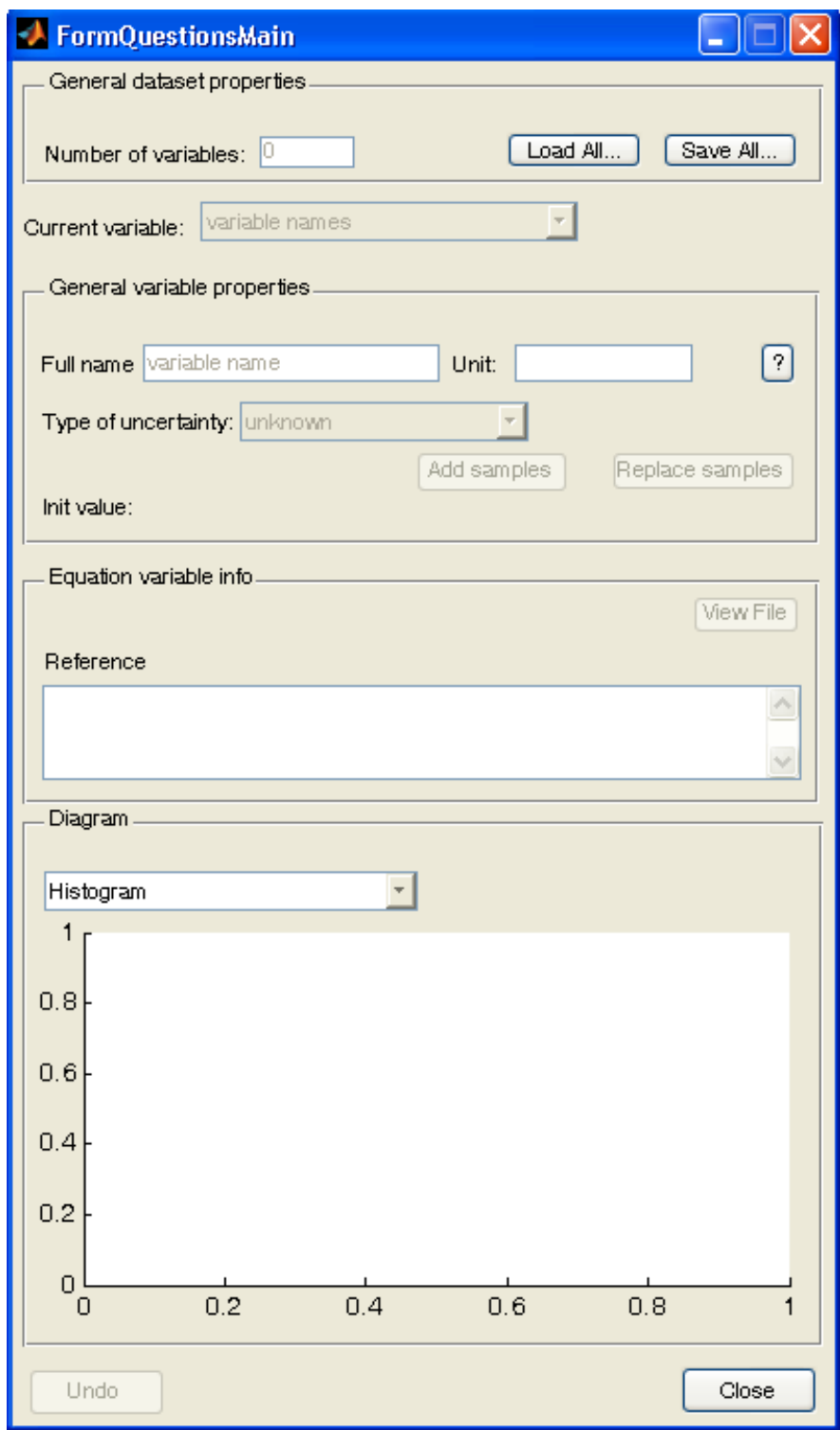


Figure 13: The initial variable selection window.

Figure 14 shows the state of the window after the loading the mat-file. Note, that the list of variables are enabled and filled in, and the full name, unit, type and initial value fields are also filled in with the first variable's data.

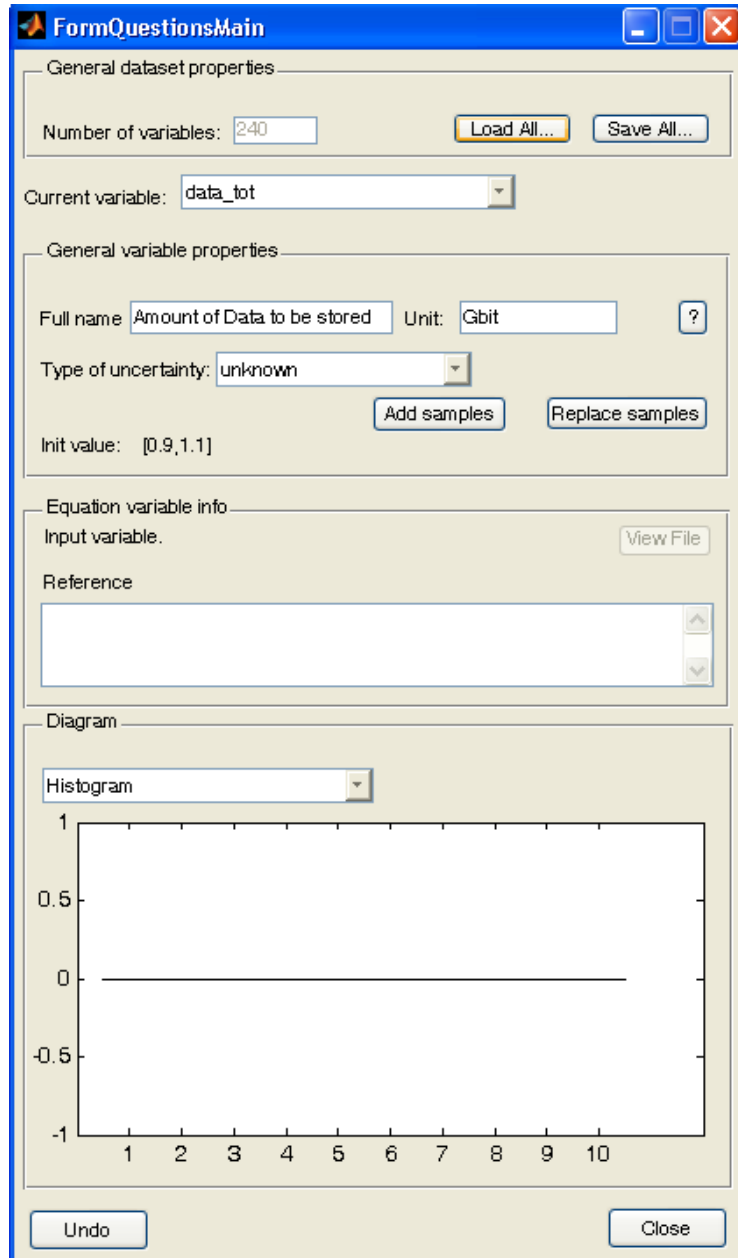


Figure 14: The filled in variable selection window.

Variables can be selected from the drop-down list, as illustrated on Figure 15.

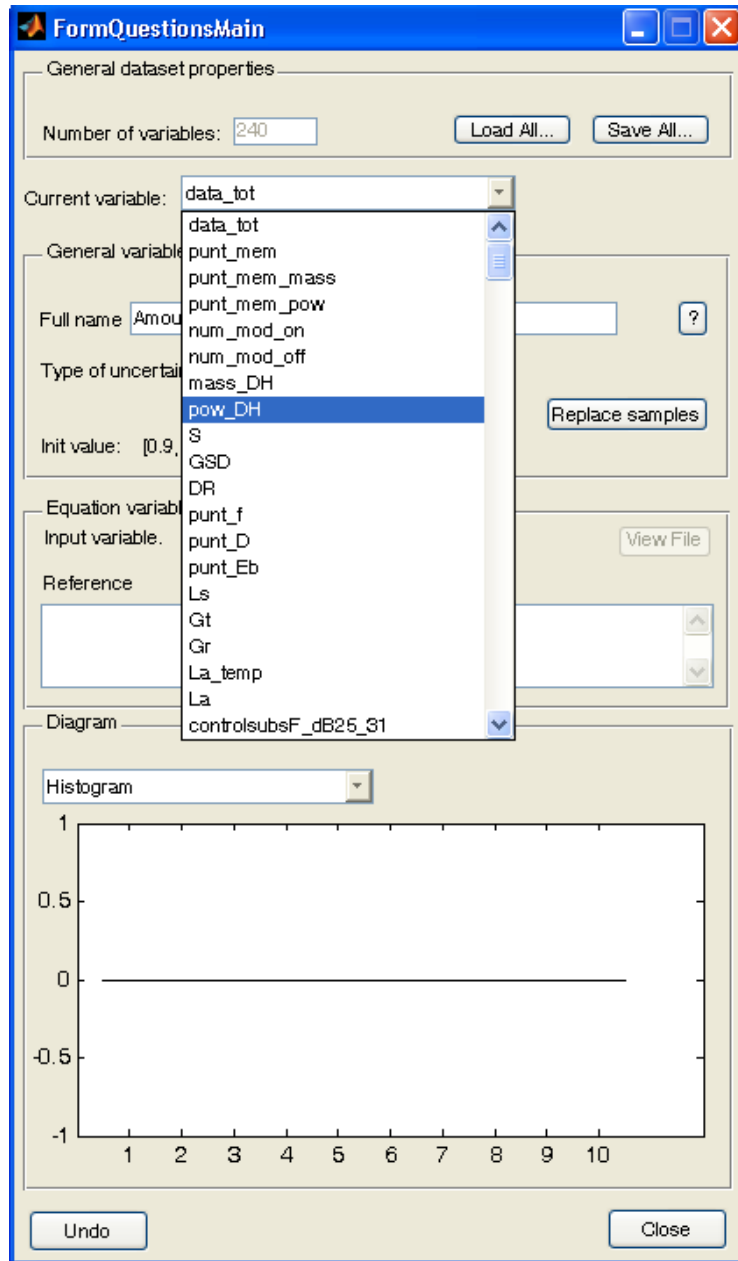


Figure 15: Variable selection.

After changing the current variable the fields are refreshed automatically

according to the new selection. If the variable is determined by other variables then the "equation variable info" panel shows the file and the line number where its equation is at. The reference textbox may contain any reference to the origin of the formula.

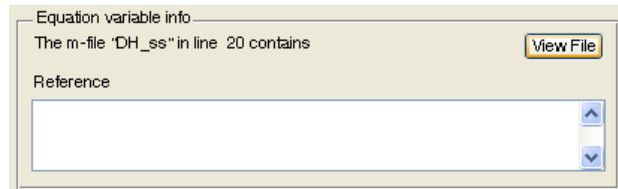


Figure 16: The "Equation variable info" panel.

The "View File" button opens the Matlab source file and highlights its corresponding line(s) where the current variable is determined. See Figure 17 for an example.

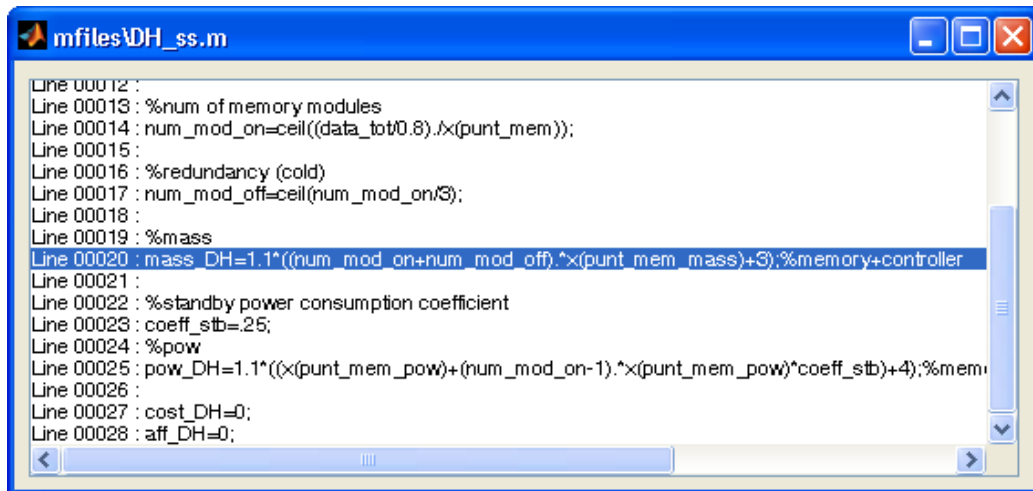


Figure 17: The mat-file viewer window.

Samples can be assigned to the variable by the "Add samples" and the "Replace samples" buttons. When assigned these sample points are showed in the bottom part of the window. It is possible to have multiple graphs illustrating variable information, for example a histogram of the sample points (Figure 18).

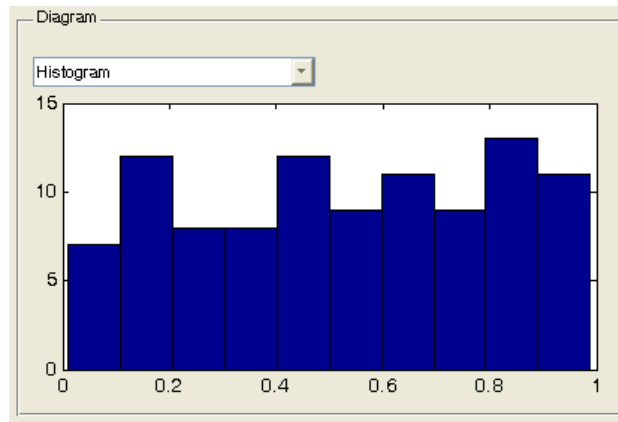


Figure 18: The diagram panel with a histogram.

The random behaviour of each variable is modeled by assigning a specific type of uncertainty to them. The current variable's uncertainty type is shown and can be modified with a drop-down list (Figure 19).

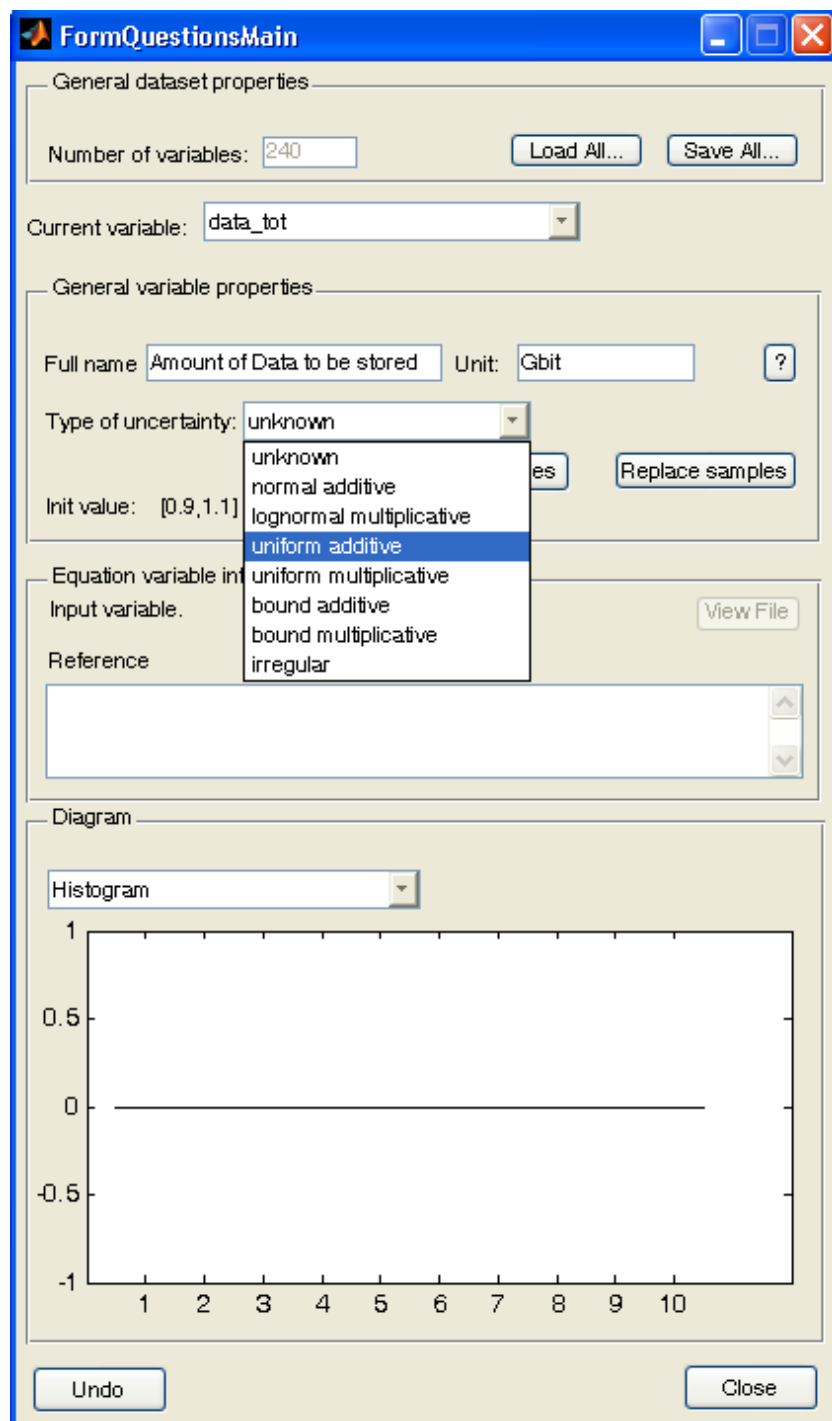


Figure 19: Selecting the type of the current variable.

7.1.2 The probabilistic information panels

After selecting a variable in the main window (or after creating a new one), a child window opens. Here the user can specify various probabilistic statistics values of the currently selected variable. This window is divided into several probabilistic information panels, each responsible for a specific property. The number and kind of the panels are determined by the uncertainty type of the current variable.

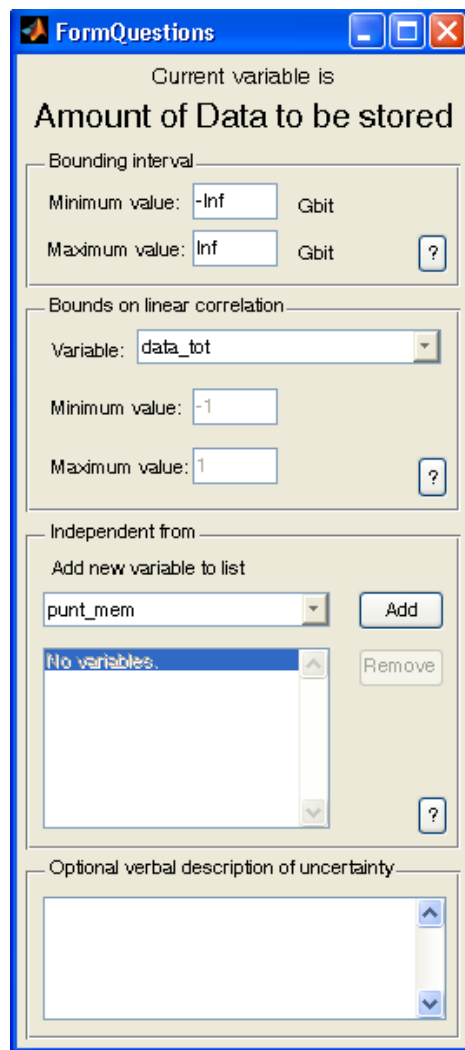
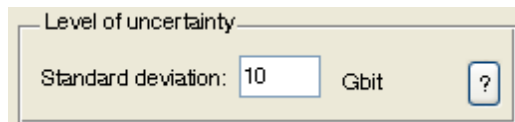


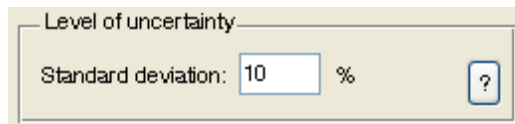
Figure 20: The window of the probabilistic information panels.

In the case when the current variable has a normal distribution, its standard deviation can be set via the panels seen on Figures 21 and 22. The standard deviation can either be set as a nominal value or relative to the variable's mean. This is determined by the type of uncertainty of the current variable. If it is "normal additive", then the nominal value, if it is "lognormal multiplicative", then its value relative to the mean can be set.



A rectangular panel with a light beige background and a thin border. At the top, it is titled "Level of uncertainty". Below the title, the text "Standard deviation:" is followed by a text input field containing the number "10". To the right of the input field is the unit "Gbit". Further to the right is a small square button with a question mark inside.

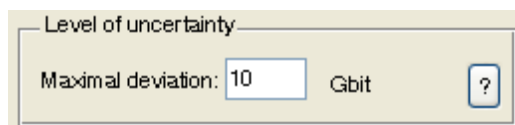
Figure 21: Panel for setting the nominal value of the standard deviation.



A rectangular panel with a light beige background and a thin border. At the top, it is titled "Level of uncertainty". Below the title, the text "Standard deviation:" is followed by a text input field containing the number "10". To the right of the input field is the unit "%". Further to the right is a small square button with a question mark inside.

Figure 22: Panel for setting the standard deviation relative to the mean.

If the current variable has a uniform additive or a bound additive distribution, then the user can set its maximal deviation (nominally), using the panel shown on Figure 23.



A rectangular panel with a light beige background and a thin border. At the top, it is titled "Level of uncertainty". Below the title, the text "Maximal deviation:" is followed by a text input field containing the number "10". To the right of the input field is the unit "Gbit". Further to the right is a small square button with a question mark inside.

Figure 23: Panel for setting the nominal value of the maximal deviation.

If the current variable has a uniform multiplicative or a bound multiplicative distribution, then the user can set its maximal deviation (in relation to the mean), using the panel on Figure 23.

Level of uncertainty

Maximal deviation: %

Figure 24: Panel for setting the maximal deviation relative to the mean.

The panel for setting a bounding interval for the current variable (see Figure 25) is enabled for all types of uncertainties. Using this panel, one can set the extremal values for the current variable. For example, in case the current variable is wind speed in km/h, the minimum value shall be set to 0. It is not necessary to fill in both fields, if one is left empty it is considered unknown.

Bounding interval

Minimum value: Gbit

Maximum value: Gbit

Figure 25: Panel for the setting the bounding interval.

For variables with normal or lognormal distribution, their expected value or mean is one of the most informative property. However, the expected value of a random variable may not be known exactly. This is reason why its corresponding panel (Figure 26) contains two fields. This way, the user can specify an interval in which a real (but unknown) mean belongs.

Expected value

Minimum value: Gbit

Maximum value: Gbit

Figure 26: Panel for setting the expected value.

In many cases correlations between variables is easy to assess. Especially in the two extremal cases, when a variable completely determines the value of another one, or when two variables are fully linear independent. Apart from

these cases, it is usually easy to predict at least whether two variables are positively correlated or not. The purpose of the correlations and independence panels (see Figures 27 and 28) is to assess this information. The information contained in these panels are symmetrical. For example, if variable A is set to be independent of variable B, then this means that B is also independent of A, and this information appears automatically in the panel of variable B, too. This mechanism also applies to the correlation panels too.

Correlations can be specified as a subinterval of $[-1, 1]$. The set of independent variables of the current variable is represented as a list. This list can be modified in two ways. The "Add" button adds the selected variable to the list of independent ones, and the "Remove" button deletes the highlighted element from the list.

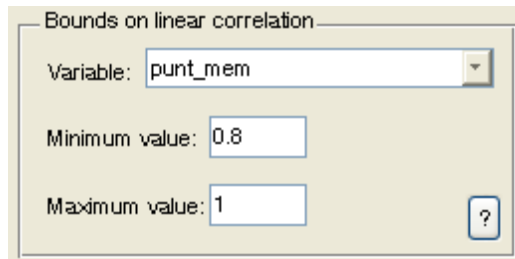


Figure 27: Panel for setting the correlations.

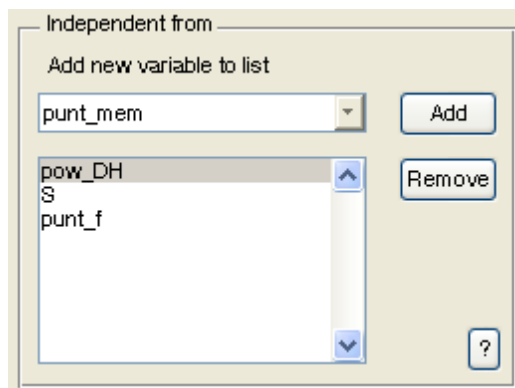


Figure 28: Panel for setting the property of independency.

Additional remarks on the uncertainty of the current variable can be given in the panel shown on Figure 29. For example, if the uncertainty categories

in the panels don't apply, the user can specify here what is known instead, e.g. giving two interval estimates by two different experts. This information is retrieved manually to improve the elicitation interface in later versions, adapting it to the knowledge actually available.

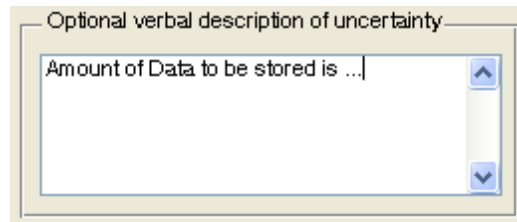


Figure 29: Panel for setting additional remarks.

Table 1 summarizes the corresponding panels for each type of uncertainty. The assignment of the panels is stored in a table, which can simply be modified or augmented, according to demands.

7.1.3 The output of the uncertainty elicitation interface

The purpose of the interactive uncertainty elicitation interface is to provide the assessed uncertainty information in a structured form for further elaboration. This structure is then the input for example of an optimization software which concerns also the random nature of the variables.

The output is actually a Matlab one dimensional cell array. Each element of this cell array contains information on one variable. Each element is a structure with the following fields.

field	description
name	short name
fullname	long name
isequation	true, if the variable is determined by an equation in some file
infile	the file name where the variable occurs
linenumber	the position of the variable in the file
unit	the unit of the variable
description	short description of the variable
type	the uncertainty type of the variable.
reference	any reference to the meaning of the variable
samples	sample points assigned to the variable
typeproperties	is a structure with the following fields
minvalue	lower bound for the possible values
maxvalue	upper bound for the possible values
mincorrelation	vector of lower bounds for the linear correlations with other variables
maxcorrelation	vector of upper bounds for the linear correlation with other variables
qualitativestatement	optional verbal description of the uncertainty of the variable
independence	list of variables which are independent from the current one
deviation	the standard / maximum deviation of the normal distribution measured in units
deviationpercent	the standard / maximum deviation in percentages of the nominal value
expectvaluemin	lower bound for the expected value
expectvaluemax	upper bound for the expected value

- normal additive
 1. LevelStdDev
 2. BoundingBox
 3. ExpectedValue
 4. Correlations
 5. Independency
 6. Qualitative
- uniform additive
 1. LevelMaxDev
 2. BoundingBox
 3. Correlations
 4. Independency
 5. Qualitative
- bound additive
 1. LevelMaxDev
 2. BoundingBox
 3. Correlations
 4. Independency
 5. Qualitative
- irregular
 1. BoundingBox
 2. Correlations
 3. Independency
 4. Qualitative
- lognormal multiplicative
 1. LevelStdDevPercent
 2. BoundingBox
 3. ExpectedValue
 4. Correlations
 5. Independency
 6. Qualitative
- uniform multiplicative
 1. LevelMaxDevPercent
 2. BoundingBox
 3. Correlations
 4. Independency
 5. Qualitative
- bound multiplicative
 1. LevelMaxDevPercent
 2. BoundingBox
 3. Correlations
 4. Independency
 5. Qualitative
- unknown
 1. BoundingBox
 2. Correlations
 3. Independency
 4. Qualitative

Table 1: Summary of the available panels for each type of uncertainty.

8 The cloud demonstration applet

A cloud demonstration applet is available online at

<http://www.inf.u-szeged.hu/~banhelyi/CloudDemonstrationApplet/>

a screenshot of the applet (in the initial state) is shown on Figure 30.

The applet visualizes the following optimization problem. Suppose there are two external variables u and v . Every pair (u, v) represents an event. These are external in the sense that their values are determined independently of the posed problem. Usually these values are not known, thus we consider them to be random. There are 100 sample points of these, which are illustrated on the left figure in the applet by blue dots. In reality these sample points originate from observations. For example u and v may be the temperature and humidity of air. For demonstrational purposes these sample points are randomly generated and the user can choose among three sample distributions:

1. independent normal – u and v are independent and have normal distribution
2. uniform – u and v , when both have uniform distribution
3. dependent normal – u and v are correlated and have a bimodal joint distribution

Also on the left figure, a level cut of the calculated cloud is shown by two red curves. The actual level cut can be set with the "safe" slider. The cloud is determined by the sample, according to the algorithm described above. In all cases a level cut of the cloud is given by two ellipses, since a quadratic potential function is fitted on the sample. Events outside the outer red curve are considered to be rare. These events have less probability than what is set by the "safe" threshold with the confidence value of "confidence level". Events inside the inner curve are considered to be frequent.

On the right figure, the design variables x and y are shown. These are internal, design variables in the sense, that their values can be altered. We also consider a technical constraint $g(u, v, x, y) \leq 0$, which depends only on

the external variables u and v , and the design variables x and y . In our case it is a fixed inequality:

$$g(u, v, x, y) = ux^2 + vy^2 - 30 \leq 0.$$

This technical constraint represents the relationship between the external and the design variables.

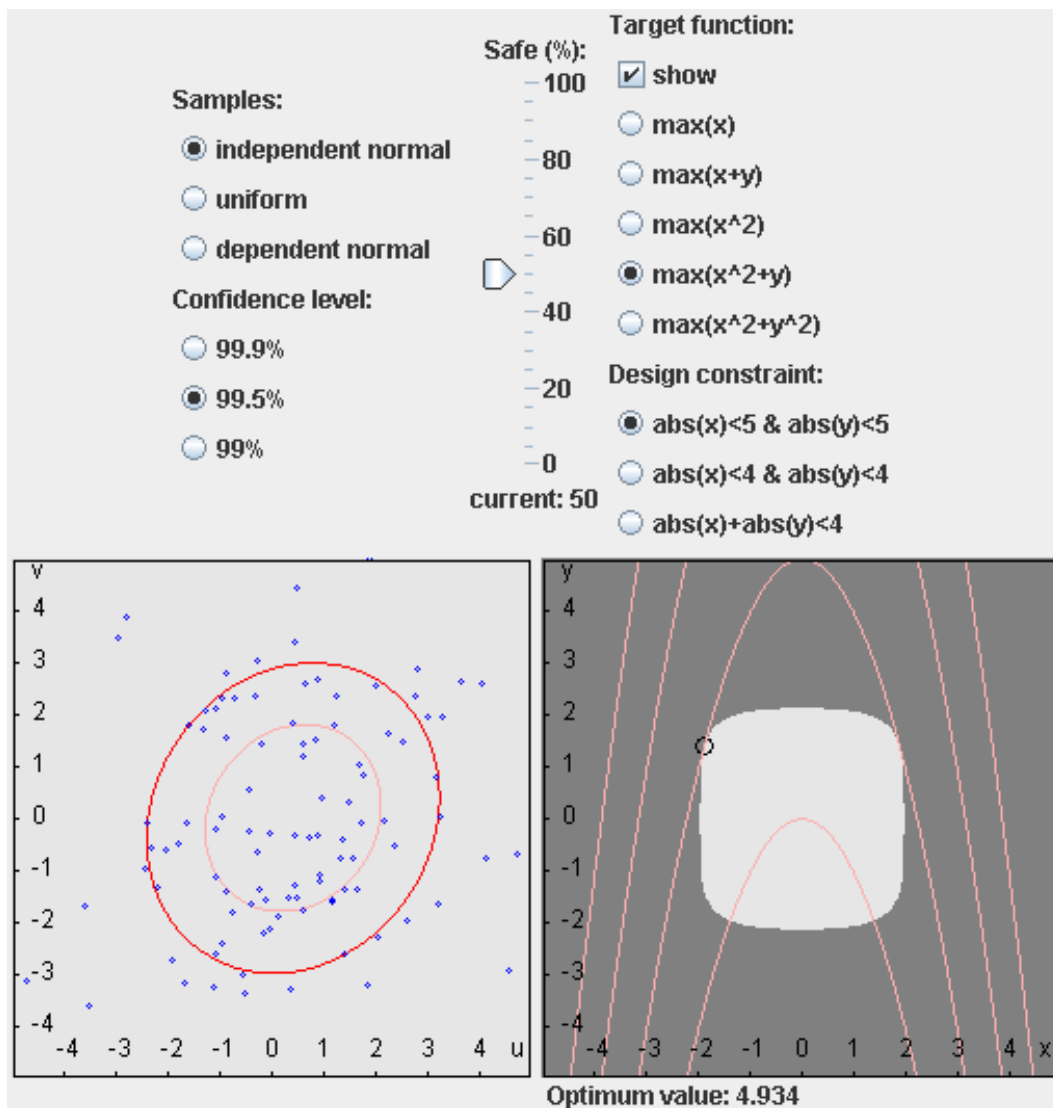
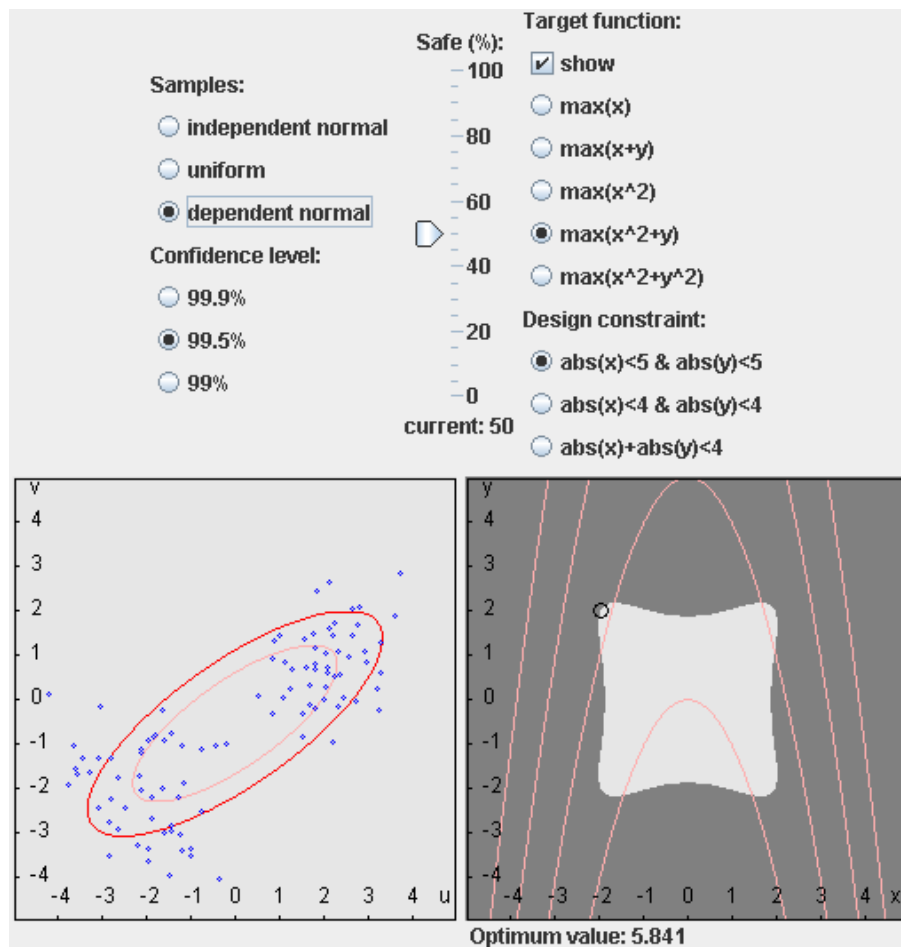


Figure 30: A screenshot of the cloud demonstration applet with default settings.

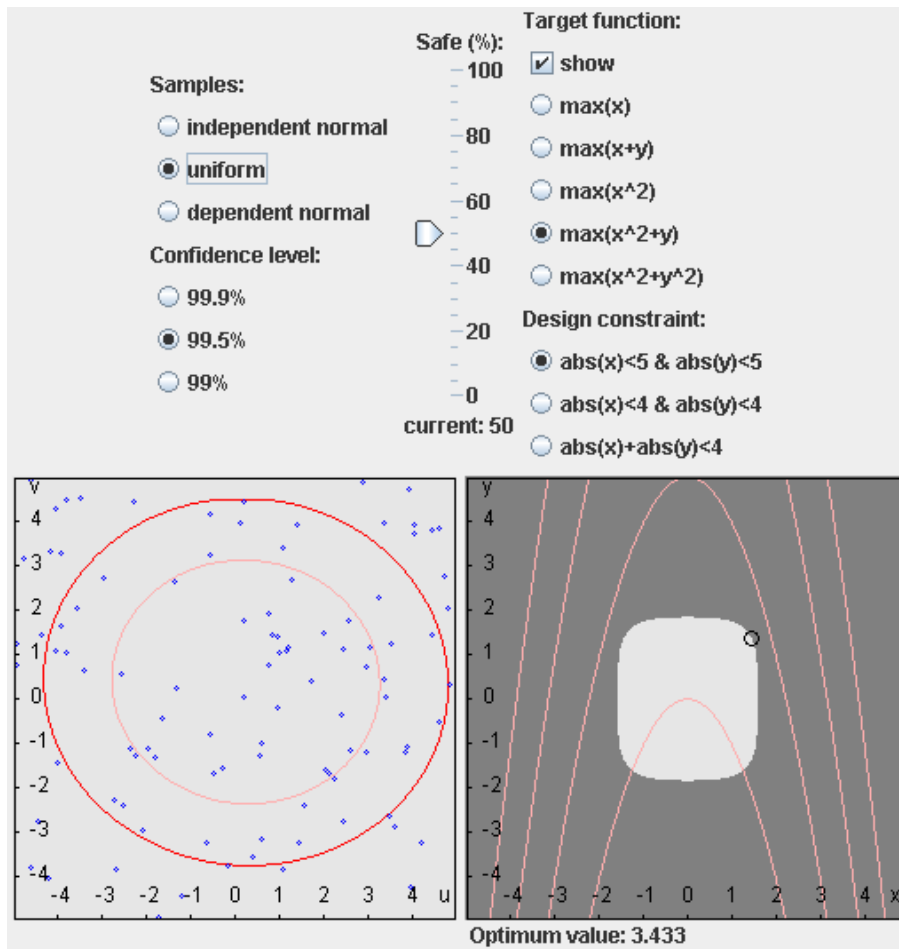
The dark gray region on the right figure is the set of all (x, y) points not fulfilling the technical constraint $g(u, v, x, y) < 0$. Moreover, the design variables x and y can be further restricted by "design constraints". This can also be chosen by the user, and it is represented by an orange curve on the right. The optimization is done under these constraints regarding the objective function set by the user. The optimum value is attained at a point shown by a black circle. The value of the objective function in the found optimizer point is also shown at the bottom of the figure.

Figures 31-35 show the applet with various settings. On each figure only one parameter is changed. Figures 31 and 32 show the effect of different sample distributions. Figure 33 illustrates the change in the level cut (and also the feasible set) caused by setting "Safe". Figure 34 shows how the target function affects the optimal place and value. Also, see Figure 35 for the effect of design constraints.



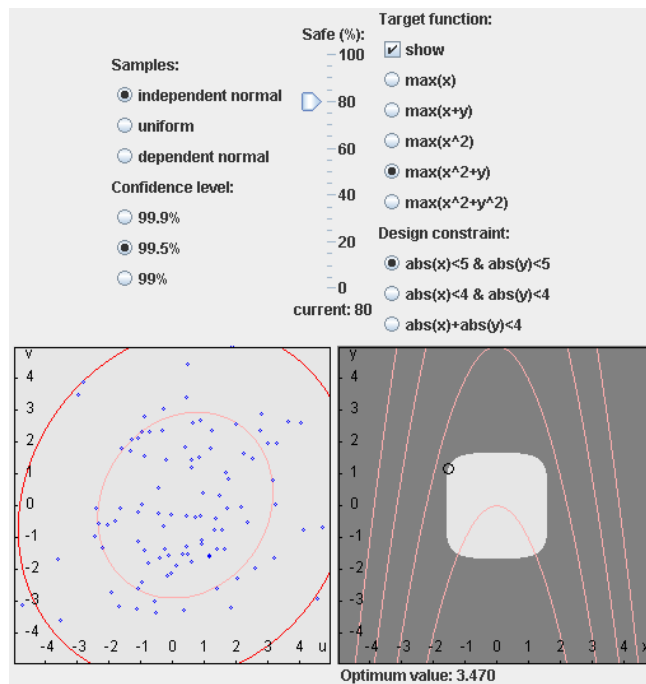
(a)

Figure 31: Screenshot of the applet with dependent normal sample points.

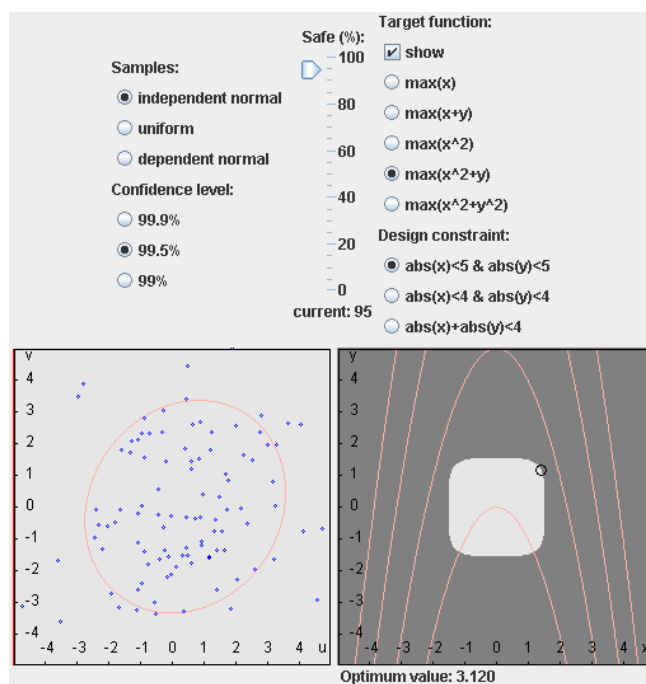


(a)

Figure 32: Screenshot of the applet with uniform sample points.

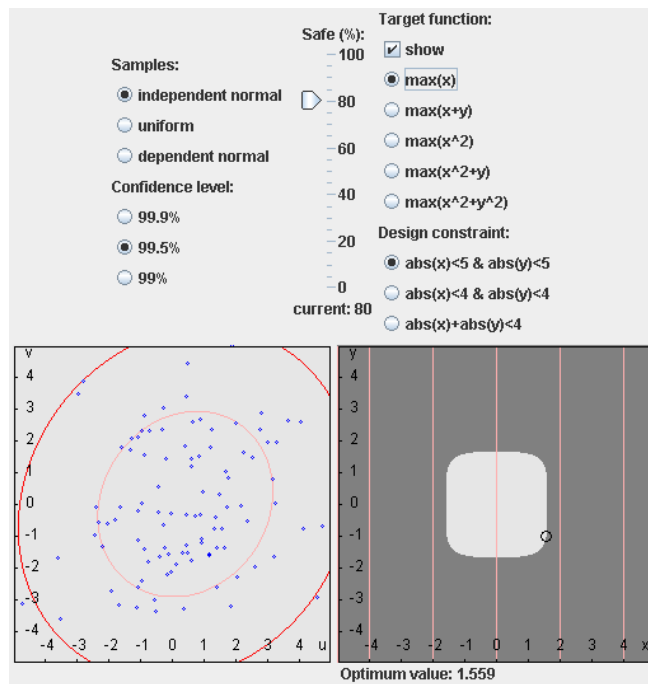


(a)

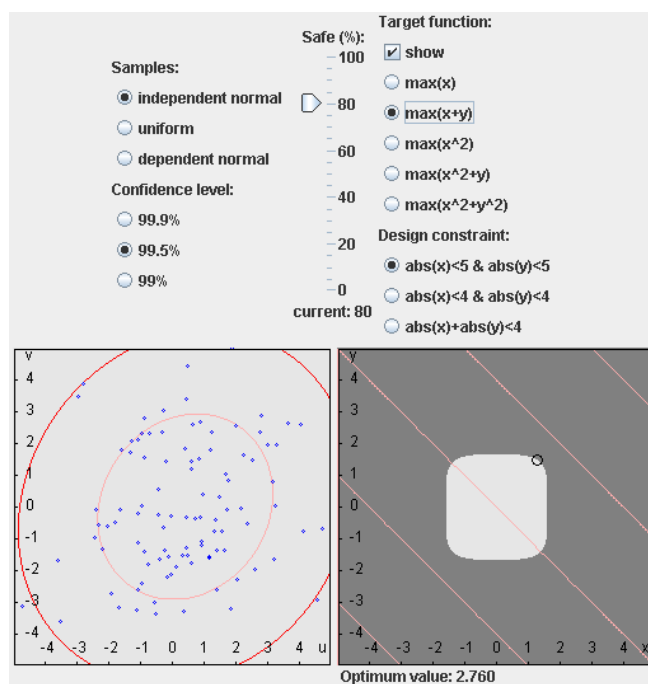


(b)

Figure 33: Screenshots of the applet with independent normal samples and different "Safe" values.

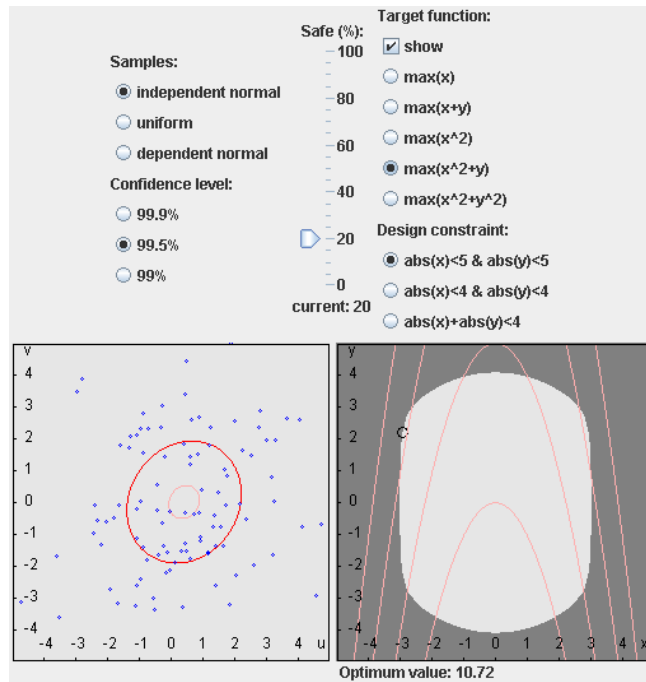


(a)

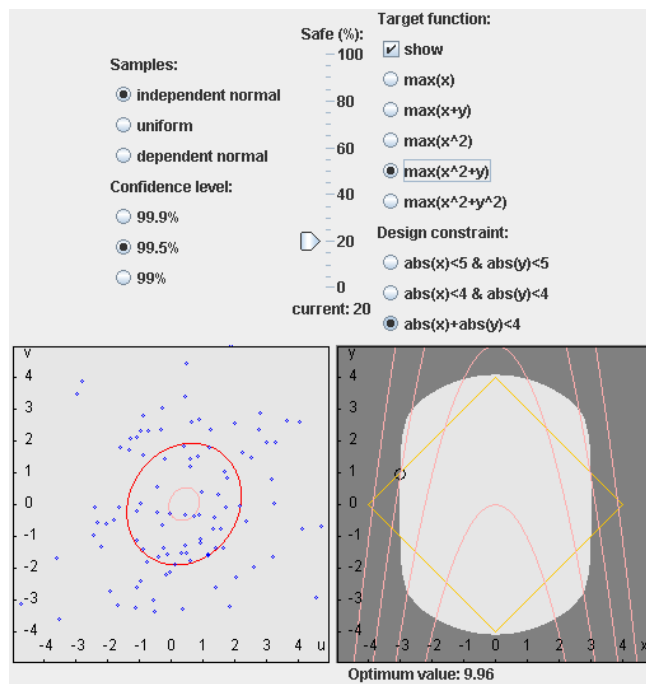


(b)

Figure 34: Screenshots of the applet with independent normal samples and different target functions.



(a)



(b)

Figure 35: Screenshots of the applet with independent normal samples and different design constraints.

8.1 A real life example

ESA has provided a real problem, from the area of satellite design. The problem has four variables, two (the temperature of the antenna and the path length) are external and cannot be controlled, two are internal and refer to design parameters of a satellite. Furthermore, there are three constants which affect the constraints and the objective function. The input parameters are shown in Table 2, and the following Java code provides the target function. There is also a constraint just specifying an upper bound on the objective function; it has no effect on the optimization.

parameter	meaning [unit]	interval
$x = R$	data rate, [bps]	from $2.5e3$ to $10e6$
$y = SAD$	space antenna diameter, [m]	from 0.2 to 2
$u = Ts$	space antenna temperature, [K]	from 250 to 1000
$v = S$	path length, [m]	from $300e6$ to $300e9$

Table 2: External variables and design variables of a real life optimization problem

```

static double TargetFunction(double u, double v,
                             double x, double y,
                             int band,
                             int modulation,
                             double elev) {

double k=1.380e-23;    //Boltzmann constant
double eta_S=0.55;    //spacecraft antenna efficiency
double eta_G=0.65;    //ground antenna efficiency
double Ll=-0.5;       //[dB] -1 if not cooled receiver,
                       //      -0.5 if cooled receiver
double c=3e8;         //[m/s] light speed

double[] freq={2e9,8e9,37e9}; //[Hz]
double[] ebn0_mod={9.6,10.3,9.6,13.3,9.2,4.4,
                  2.7,4.0,9.6,13.8}; //[dB]

double f=freq[band];
double GSD=5; // [m]
double ebn0=ebn0_mod[modulation]+2; //margin

```

```

// Elevation angle

double Gt=20.0*Math.log10(Math.PI)+20*Math.log10(y)
        +20*Math.log10(f)+10*Math.log10(eta_S)
        -20*Math.log10(c)-12*0.84*0.84;

// Antenna pointing offset 84% beamwidth

double Gr=20.0*Math.log10(Math.PI)+20*Math.log10(GSD)
        +20*Math.log10(f)+10*Math.log10(eta_G)
        -20*Math.log10(c)-12*0.1*0.1;

// Antenna pointing offset 10% beamwidth

double Ls=20.0*Math.log10(c)-20.0*Math.log10(4*Math.PI)
        -20.0*Math.log10(v)-20.0*Math.log10(f);
double[] coeff_La={4.657e-7,-1.304e-5,2.556e-4,7.979e-4,0.041};

double La_temp=((coeff_La[0]*f*1e-9+coeff_La[1])*f*1e-9
        +coeff_La[2])*f*1e-9+coeff_La[3])*f*1e-9
        +coeff_La[4];

double La=-La_temp/Math.sin(elev*Math.PI/180)-1.3;

double pow_dB=ebn0+10*Math.log10(k)+10*Math.log10(u)
        +10*Math.log10(x)-Ll-Ls-La-Gt-Gr;
double pow_TTC=Math.pow(10,(pow_dB/10)); //[watt]

return pow_TTC;
}

```

Although the solution of the optimization problem is not too time-consuming, its visualization takes usually much more resources. The reason for this is that a lot of cases have to be calculated separately in order to get a fine image of the set of feasibility. This is the reason why a Java applet could not visualize this problem quickly, in real time.

The above simple demonstration problem could do this demonstration in real time since the objective function there involved just a few operations. The present antenna design problem has a much larger computational complexity.

On the top of that, we have found that the original set problem has almost exclusively feasible points in the given variable rangess. Once we decrease the allowed level of the objective function, we can find the level sets to be line segments.

The total computational time for the 256 x 256 points is about half an hour on our standard strength PC; but the acceptable amount of waiting time should be under 10 seconds. To decide whether a point on the right window is feasible requires the calculation with all the points of the left window. Monotonicity could possibly give a kind of remedy, but it is not clear at present to which extent it can be automatically exploited.

A possible way out could be to decrease the resolution from the present 125 x 125 to a lower level – although the total computation time can only be decreased proportionally with the number of illustration points. Additional constraints could also help, since they can be utilized to reduce the amount of computational burden.

9 Optimization problem formulation

According to our understanding, the problem can be naturally posed as a particular case of the following general form:

$$\begin{aligned}
 \min_{\theta} \max_{x,z,\varepsilon} f(x) & && \text{(objective functions)} \\
 \text{s.t.} \quad z = Z(\theta) + D\varepsilon & && \text{(table constraints)} \\
 F(x, z) = E\varepsilon & && \text{(functional constraints)} \\
 V(\varepsilon) \leq V_{\alpha} & && \text{(cloud constraint)} \\
 \theta \in C & && \text{(selection constraints)}
 \end{aligned} \tag{1}$$

Here z is the vector consisting of all global input variables and all design variables, x is the vector consisting of all output variables (including intermediate outputs if they have an associated uncertainty), θ is the vector of choice variables (with one variable for each independent table of choices), and ε is the vector of uncertainties. Uncertain global inputs are treated as design variables with a single choice only.

The table constraints assign to each choice θ a design vector z whose value is the nominal table entry $Z(\theta)$ plus its (unknown) error $D\varepsilon$ with specified uncertainty.

The functional constraints express the functional relationships defined in the Matlab functions. It is assumed that the number of equations and the number of output variables is the same (i.e., $\dim F = \dim x$), and that the equations are (at least locally) uniquely solvable for x . This is likely to be the case, and was confirmed for the model problem in some scenarios, cf. Section 11.

The cloud constraint expresses the knowledge about the uncertainty in the form of a confocal cloud, with a potential to be derived from the uncertainty specification. At present, all uncertainties are deterministic intervals, so that the cloud is box-shaped.

The selection constraints specify how many choices are allowed for each choice variable, and the range for the variables with continuous choices (at present only the slim factor).

9.1 Difficulties

- The problem structure is – apart from the dimension – already in the form without uncertainty as difficult as the most complex problem category. It is a fullblown mixed integer nonlinear program (MINLP), with multiple (two) objectives and internal branching, which cannot be handled directly with standard optimization software. MINLP is still a recent research direction which has not yet matured.
- In addition, there is the bilevel structure imposed by the uncertainties, which is already a nontrivial complication in the traditional situation where all variables are continuous. The current methods for handling such problems require at least that the objective and the functional constraints are continuously differentiable.
- This lack of C^1 structure of the problem without uncertainty makes the bilevel problem significantly more difficult than the smooth problems we have experience with. To use current optimization technology for bilevel problems, the partially discrete nature of the current problem would have to be eliminated in one way or another.
- The uncertainty structure may become slightly more complex as additional uncertainty information becomes known.
- In view of these difficulties, which seem central to a good attack on the problem, we decided to simplify the problem in another complicating aspect; We restrict our discussions to the single objective case,

using some form of aggregation or constraints to reduce the multiple objectives (`mass_tot` and `pow_tot`) to a single objective. The initial objective suggested by ESA is to minimize the total mass `mass_tot` with respect to the input variables (the mission requirements).

10 Optimization

Now we discuss how to solve the optimization problem stated in Section 9. On the one hand we will cope with the full bilevel problem (1), on the other hand – as to the difficulties concerning the bilevel structure mentioned in 9.1 – we investigate the simplified problem:

$$\begin{aligned}
 \min_{\theta} \quad & f(x) \\
 \text{s.t.} \quad & z = Z(\theta) \\
 & F(x, z) = 0 \\
 & \theta \in C
 \end{aligned} \tag{2}$$

with the uncertain global input variables fixed in the middle of their given boxes.

Problem (1) will be denoted by OUU (**O**ptimization **U**nder **U**ncertainty) and problem (2) by OWU (**O**ptimization **W**ithout **U**ncertainty).

10.1 Symbolic solvers

10.1.1 Without Uncertainty

After the conversion to LINGO (cf. Section 5.3) we have been trying to make the solver find the optimal solution of problem (2) for the objective function total mass `m_tot`, but experienced problems even to find a feasible solution for the model.

It is all but obvious what causes the infeasibility. We started to examine the LINGO solver and some other solvers and concluded that the solvers themselves, at least LINGO, seem to work properly. We checked the converters and fixed some problems that are mentioned in Section 5.3 (like box declaration): the converters should not create any errors now. We have found

several mistakes in the generation of the AMPL model file, but the current version does not appear to differ from the original Matlab model anymore (it was finally checked line by line). Therefore we went back to the subsystems and tried to check them for possible errors (we found some and corrected them, cf. Section 3.1).

The reason why the solver is still unable to find a feasible solution is probably the translation of if-structures into explicit expressions (cf. Section 4.2) because doing so requires every branch of the structure to be evaluable in any case. LINGO – unlike AMPL – does have support for if-structures, so a direct DAG to LINGO conversion should be helpful with respect to this problem. A Matlab if-parser would be needed for that conversion. Due to lack of time, this was not yet implemented.

10.1.2 With Uncertainty: Sensitivity approximation in the smooth case

For convex optimization, robust optimization is a well-developed subject; see, e.g. [5]. In the nonconvex case or when discrete structures are present, virtually nothing has been done.

For attacking the bilevel problem we devised the following plan.

For a design optimization problem with smooth (at least continuously differentiable) functional constraints and no logical constraints, we shall approximate the bilevel problem by a related ordinary optimization problem, using methods from sensitivity analysis. This solves the bilevel problem up to higher order terms in the uncertainties, and hence is appropriate provided the latter are reasonably small.

In our case, the cloud is presently box-shaped, so that the potential can be taken (after proper scaling) as the maximum norm, $V(\varepsilon) = \|\varepsilon\|$. For a problem given in the form (1), we may then proceed as follows:

For a fixed choice θ , we consider the problem

$$\begin{aligned} \hat{f}(\theta, \alpha) := & \max_{x, z, \varepsilon} f(x) \\ \text{s.t.} & z = Z(\theta) + D\varepsilon, \\ & F(x, z) = E\varepsilon, \\ & \|\varepsilon\| \leq V_\alpha \end{aligned} \tag{3}$$

Let the solution of the functional constraints be $x = x(z, \varepsilon)$, and write $\hat{z} = Z(\theta)$, $\hat{x} = x(\hat{z}, 0)$. Then the problem (3) is to maximize

$$\begin{aligned}\hat{f}(\varepsilon) &= f(x(\hat{z} + D\varepsilon, \varepsilon), \hat{z} + D\varepsilon) \\ &= f(\hat{x} + x_z D\varepsilon + x_\varepsilon \varepsilon, \hat{z} + D\varepsilon) + O(\varepsilon^2) \\ &= f(\hat{x}, \hat{z}) + [f_x(\hat{x}, \hat{z})(x_z D + x_\varepsilon) + f_z(\hat{x}, \hat{z})D]\varepsilon + O(\varepsilon^2),\end{aligned}$$

with index notation for partial derivatives. Because of the simple form of the potential, the worst case is given by

$$\hat{f}(\theta, \alpha) = f(\hat{x}, \hat{z}) + V_\alpha \|f_x(\hat{x}, \hat{z})(x_z D + x_\varepsilon) + f_z(\hat{x}, \hat{z})D\|^* + O(\varepsilon^2),$$

where $\|\cdot\|^*$ denotes the dual norm, in our case the row sum norm. To get the required partial derivatives of x , we differentiate the functional constraint with respect to z and ε and obtain

$$F_x x_z + F_z = 0, \quad F_x x_\varepsilon = E.$$

Assuming that F_x (which under our assumptions is a square matrix) is non-singular, we find

$$x_z = -F_x^{-1}F_z, \quad x_\varepsilon = F_x^{-1}E,$$

hence

$$f_x(x_z D + x_\varepsilon) = f_x F_x^{-1}(E - F_z D).$$

Introducing the multiplier $y^T := f_x F_x^{-1}$ and neglecting the higher order term, we find (upon deleting the carets) that the original bilevel problem approximately reduces to solving the ordinary nonlinear program

$$\begin{aligned}\min_{\theta, x, z} & f(\hat{x}, \hat{z}) + V_\alpha \|y^T(E - F_z D) +_z D\|^* \\ \text{s.t.} & z = Z(\theta) \\ & F(x, z) = 0 \\ & F_x(x, z)^T y = f_x(x, z)^T\end{aligned}\tag{4}$$

Essentially, this problem can be solved by any program that is able to solve the original problem without uncertainties. (In our case, this still is a mixed integer nonlinear program of considerable difficulty, but the bilevel structure disappeared.)

The robustness of the solution of (4) found, considered as an approximate solution of (3), can be checked independently by a rigorous interval analysis (again assuming smoothness), as described in the material in [3] referred to in our proposal.

To which extent the same approximations will work in the absence of smoothness (with partial derivatives replaced by subgradients), is an issue that will have to be investigated.

As the symbolic solver does not yet provide useful results without uncertainty exclusive of additional focus on improving the converter methods and model generation we did not implement the case with uncertainty.

10.2 Heuristic approach

In view of the difficulties mentioned we approach the solution of (1) and (2) by means of heuristics. For this purpose one only needs the modifications specified in Section 4.1. In Section 3 the recursive structure within the subsystems is mentioned which leads to a fixed point problem $G(x, \theta, \varepsilon) = x$ (where x is the vector containing the three recursive variables `m_tot`, `pow_prop` and `pow_ther`, all choice variables and global inputs fixed, i.e. fixed choice θ and scenario with deviation ε).

An evaluation of $G(x, \theta, \varepsilon)$ amounts to running once the following Matlab code:

```
[mass_DH,pow_DH,cost_DH,aff_DH]=...
    DH_ss_preprocessed(data_tot,punt_mem,punt_mem_mass,...
    punt_mem_pow)

[beam,mass_TTC,pow_TTC,cost_TTC,aff_TTC]=...
    TTC_ss_preprocessed(punt_f,punt_D,punt_Eb,elev,S,GSD,DR)

[diam_SA,P_SA,A_SA,T_SA,mass_SA,cost_SA,aff_SA,capacita,...
volume_batt,mass_batt,cost_batt,aff_batt,pow_tot,...
mass_pow,cost_pow,aff_pow]=...
    pow_ss_preprocessed(punt_eta,punt_alfa,punt_ro,punt_d,...
    punt_spec,punt_eff,punt_dens,max_sun_dist,min_sun_dist,...
    Tecl,teta0,Tday,y,cicli,body_mount_SA,primary,...
    target_planet,punt_h_vs_r,powzero,pow_TTC,pow_DH,...
    pow_aocs,pow_prop,pow_ther)

[diam,S_Thickness,height,area_tot,mass_str,cost_str,aff_str]=...
    str_ss_preprocessed(punt_El,punt_rho,punt_ult_str,...
    punt_h_vs_r,punt_yie_str,m_tot,diam_SA,body_mount_SA,...
    ax_g,lat_g,ax_freq,lat_freq)
```

```

[H_planet,Gs_hot_2,H_min_IR]=...
    target_planet_func2_preprocessed(target_planet,...
    H_terra,H_target)

[rad_area,mass_ther,pow_ther,c_ther,a_ther]=...
    thermal_ss_preprocessed(Tup,Tdown,target_planet,...
    min_sun_dist,max_sun_dist,punt_alfa_sup,punt_eps_sup,...
    area_tot,A_SA,P_SA,dens_dep_rad,Tecl,body_mount_SA,...
    m_tot,H_planet,Gs_hot_2,H_min_IR)

[m_fuel,m_tot_prop,mass_prop,pow_prop,N_prop,cost_prop,...
aff_prop,cost_tot,aff_tot]=...
    prop_ss_preprocessed(deltaV,T_PP,punt_I,punt_m_eng,...
    punt_P_eng,punt_T_eng,mzero,mass_TTC,mass_DH,...
    mass_aocs,mass_pow,mass_ther,mass_str)

[mass_harness,m_tot,m_tot_mb]=...
    mass_budget_preprocessed(mzero,mass_TTC,mass_pow,...
    mass_prop,mass_str,mass_DH,mass_aocs,mass_ther,m_fuel,...
    count)

```

The preprocessed files and the function `target_planet_func2` are used with the general modification in Section 4.1. The recursive variables are those intermediate results variables (cf. Section 3) which occur as input before they are calculated: `m_tot`, `pow_prop` and `pow_ther`. (As mentioned before, it depends on the sequence of the subsystem function calls which variables are recursive.)

In the following we write $G(x) = G(x, \theta, \varepsilon)$ if θ and ε are fixed. Then the wanted solution is a **fixed point** of G , i.e., the equation $G(x) = x$ must hold.

We implemented three methods to solve the fixed point problem.

1. **Iteration**
2. **Zero finding**
3. **Global residual nonzero minimizing**

The first one is simple **fixed point iteration**, (as apparently in the Manager, provided by ESA). This produces a sequence of points $x_{k+1} = G(x_k)$; if it converges to x then x is a fixed point. For our second method, we reformulated

the problem to a **zero finding problem** $F(x) = 0$ with $F(x) = G(x) - x$, that is solved by the program `nleq.m` (cf. [7]). As a third method we minimize $\|G(x) - x\|$ with the global optimization program **MCS** (cf. [8]).

	Iteration	NLEQ	MCS
average time for solving one fixed point problem / s	0.4	0.5	1.8
converging on an average of / %	50	60	60

The iteration method is the fastest one, but fails converging in too many cases. The convergence of the two other methods with `nleq` and **MCS** respectively is far better and their results match each other. The performance of `nleq` is not much slower than the iteration, while **MCS** is more robust, but takes significantly more time to solve the fixed point problem and requires input boxes for the range of the recursive variables. So we choose `nleq` to solve the fixed point problem.

An evaluation of the problem corresponds to obtaining a value for the objective function f (cf. f in (2) \rightarrow OWU, $\max f$ in (1) \rightarrow OUU, in our case $f = m_{tot}$) for a fixed θ , satisfying the functional constraints.

In OUU the objective function is $\max f$ among all possible scenarios for the input variables, so we obtain a value of the objective function for a fixed θ by finding the worst case for `m_tot` in the region defined by the cloud constraints (in our case boxes for each uncertain input variable as the cloud is box shaped).

An evaluation of the problem for a fixed choice in OWU means solving the fixed point problem; in OUU it is the result of the **corner search** which is explained below.

To find the worst case we do a **corner search**: A corner is a point in \mathbb{R}^{24} (since we have 24 uncertain global inputs) where each variable takes the maximum or minimum value of the associated uncertainty interval. The corner search starts with solving the fixed point problem for the scenario of all uncertain variables fixed at the maximum of their uncertainty intervals. Then we start with the first variable and change the scenario by taking the opposite end of the uncertainty interval for this variable, and we solve the fixed point problem for this new corner scenario. We go on varying corners in the same manner for each variable. If a corner worse than the starting

corner (i.e. a higher value of the objective `m_tot`) is found, this corner will be taken as starting point for the further search. Each coordinate will be varied this way at most once and the corner search returns the maximum value for `m_tot` found. If no solution was found for some scenario during the search, the worst case for `m_tot` is set NaN (not a number) and the corner search ends returning NaN. The corner search thus requires solving (at most) 24 fixed point problems, since there are 24 uncertain variables. To be sure to find the correct worst case by corner search one has to assume monotony for `m_tot` within the boxes (cf. Section 12). Steep peaks (not yet observed) or any hidden constraints (observed, cf. Section 12) decrease the reliability of the corner search, but cannot be handled properly with heuristics anyway.

For a given θ the algorithm (simplified) for the corner search looks like:

```
function m_tot=cornersearch(theta)

Let  $u_i \in [\underline{u}_i, \bar{u}_i]$ ,  $i=1, \dots, 24$ , be the uncertain variables in the
uncertainty intervals.
Let  $x1=[m\_tot, pow\_prop, pow\_ther]$  be the vector containing the
recursive variables.
Solve  $G(x1)=x1$  for fixed  $u1=(\bar{u}_1, \bar{u}_2, \dots, \bar{u}_{24})$ 

for i=1:24
    u2=u1;
    u2i =  $\underline{u}_i$ ;    % opposite end of the uncertainty interval
    Solve  $G(x2)=x2$  for the new u2
    if infeasible
        return NaN;
    end
    if x2(1)>x1(1)          % m_tot worse
        u1=u2;x1=x2;      % new worst case saved
    end
end

return x1(1);
```


For one corner search	time
average	3.9 sec
fastest	0.5 sec
slowest	11.0 sec

As a simple check of our assumptions at the end of each corner search the fixed point problem for the opposite corner (i.e. the opposite end of uncertainty interval in each variable is taken) is solved and should yield a smaller or equal value for `m_tot` than the corner search.

We regard a choice as infeasible if the evaluation of the problem for this choice does not provide a real-valued, nonnegative solution (strictly speaking we cannot guarantee the infeasibility because `nleq.m` cannot guarantee convergence in any case where a solution exists). A choice is considered feasible if the evaluation does provide such a solution.

Now we need a strategy to sample choices and heuristically finding an optimal choice.

10.2.1 DAKOTA and other software packages

At first we tested DAKOTA (cf. [9]). DAKOTA takes the evaluation described above as a black box and creates a surrogate model by sampling. It should be able to optimize the surrogate model and later on to solve the whole OUU problem replacing the corner search. Unfortunately after detailed studies on the program functionalities it turned out DAKOTA wasn't even able to solve simple MINLPs as some public license subroutines were not working and a substitution by appropriate commercial subroutines was not available.

Moreover, we spent some time on further software experiments with `nomadm` (cf. [10]) and `condor` (cf. [11]) that failed because of problems with software libraries, path problems and gui problems.

10.2.2 SNOBFIT-based heuristics

Finally we based a sampling strategy on SNOBFIT (cf. [12]). Similar to DAKOTA we take our evaluation of the problem as a black box function

that provides values of `m_tot` for given choices. For a given set of starting points and function values SNOBFIT fits a quadratic model, minimizes this and suggests new sample points to be evaluated to improve the model. The number of the suggested points, boxes and mesh size for the suggested points can be adjusted. The integrality of the choice variables is ignored by SNOBFIT, the integer choice variables are simply initialized on a grid with mesh size 1 and the suggested points are those points on the grid nearest to the points in \mathbb{R}^{10} (10 choice variables) that SNOBFIT has requested. Suggested points are classified by SNOBFIT into 5 types: type 1 best prediction, type 2 putative local minimizer, type 3 alternative good point, type 4 to explore the empty region, type 5 to fill up the required number of function values if too little points of the other classes are found. We proceed as follows:

To prepare the subsystem model for the quadratic fit we first permute the tables for the choice variables manually to achieve an order as smooth as possible for each of them.

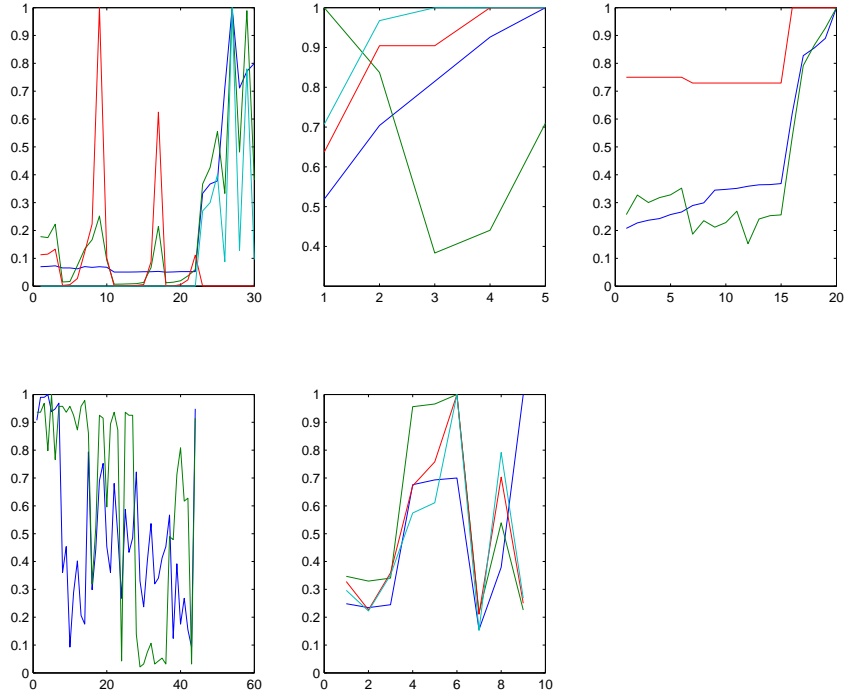


Figure 36: For each multidimensional table for the choice variables the value of the i th table entry is plotted against i . It looks very rough, and is unsuitable for local quadratic models as used by SNOBFIT.

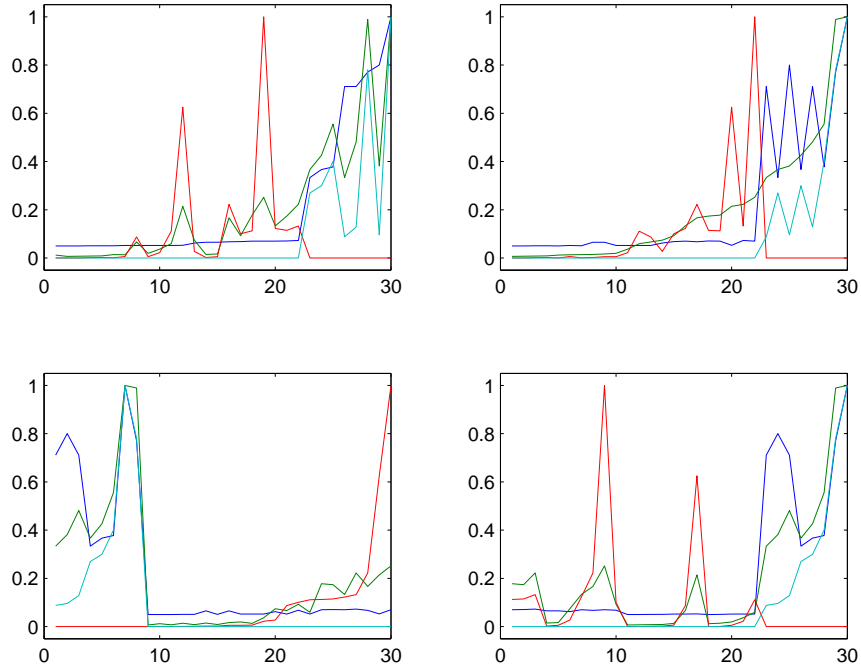


Figure 37: Four permuted plots of the first table as an example, the permutations result from sorting one of the four coordinates of the table by increasing values: first `punt_I`, second `punt_m_eng`, third `punt_T_eng`, fourth `punt_P_eng`. We decided to use the third permutation, which is visually the least rough one.

Then we create 20 independent sets of 80 starting points (randomized space-filling design suggested by SNOBFIT run without input) and evaluate them; about 17% of the starting points were found feasible. For each set of starting points we do an independent search for the global optimum.

The search starts with SNOBFIT which suggests 5 points from the starting points model. We evaluate them and afterwards take a point of type 1 (if it exists) and a point of type 4 or 5 (if it exists) to start a line search (see the next paragraph) from both 2 points to find a local minimum. The line search from the point of type 1 leads to a local minimum close to the best prediction of SNOBFIT, the line search from the type 4 or 5 point is involved to find further local minima to achieve a more global search. All evaluations during the line search will be input for the next SNOBFIT calculation which

provides the current optimal solution, again requesting 5 points to evaluate, and we continue in the same manner. If SNOBFIT provides the same optimal solution 5 times in a row this solution is returned and the search ends. The fivefold confirmation of the optimum has been found necessary to make the heuristics reliable.

For the line search all choice variables but one are fixed, this one is varied through all possible choices or varied by MCS in the case of continuous choice variables (slimfactor). The minimal choice is stored and another variable will be varied, all intermediate functions values are saved for the SNOBFIT search. The line search ends if it identifies the choice to be varied as one that has already been stored within the intermediate data of the SNOBFIT search because in this case the line search will fall into the same local minimum as before and will not provide any new problem evaluations. The line search serves the purpose to accelerate the convergence of the whole SNOBFIT search by approx. factor 5, as SNOBFIT lacks integer implementation and is thus not efficient at finding local minima.

To improve the reliability of the optimum we do this search independently with the 20 sets of starting points.

For one search with SNOBFIT	time
OWU average	0:40:38 hours
fastest	0:33:49 hours
slowest	0:49:26 hours
OOU average	6:23:29 hours
fastest	3:28:56 hours
slowest	11:57:49 hours

10.2.3 Matlab programs

The central Matlab programs for the heuristic algorithms are the following.

To generate the 20 sets of starting points we use a Matlab script `gensfinput.m`. The 20 sets are stored in the mat-files `sfinput_1.mat` to `sfinput_20.mat`.

We then do the 20 SNOBFIT runs

```

for i=1:20
    tic
    [request,xbest,fbest,fbestv]=...
        snobfitbb('sf',[ 'sfinput_' num2str(i)]);
    t=toc;
    save(['fbestv_' num2str(i)],'fbestv','t');
end;

```

The program

```
function [request,xbest,fbest,fbestv]=snobfitbb(filename,sfinput)
```

takes as inputs the file name of the workspace file required by SNOBFIT and the name of the file containing the starting points. The outputs are the last request of SNOBFIT, the optimal choice found, the function value at the optimum and a vector **fbestv** with the optimal function value after each iteration of the search. The results are stored in the mat-files **sf_ver1.mat** to **sf_ver20.mat**.

We save **fbestv** and the time separately for the performance analysis.

11 Results

The first set of initial values, provided by ESA (**initial_values.xls**) has lead to no results. We traced back the possible error to **TTC_ss** where the variable **pow_ttc** gets the order of magnitude 10^9 W during the calculation of **mass_ttc**, it results in **mass_ttc** to be about 10^{24} kg. Thus no fixed point could be found.

So only the second set of initial values provided (`initial_values_xeus.xls`) was used. The certain input values were:

variable	initial value
a_ther	0
aff_DH	0
aff_TTC	0
aff_batt	0
aff_pow	0
aff_prop	0
aff_str	0
aff_tot	0
body_mount_SA	1
c_ther	0
cost_DH	0
cost_SA	0
cost_TTC	0
cost_batt	0
cost_pow	0
cost_prop	0
cost_str	0
cost_tot	0
count	1
elev	90
mass_aocs	0
pow_aocs	0
primary	0
target_planet	3

The 20 runs of SNOBFIT resulted in the following objective function values, giving an indication of the relative efficiency of the starting phase (space-filling design) and the optimization.

SNOBFIT run #	m_tot (starting point)	m_tot (best)
1	1849.0879	1730.6923
2	1790.1089	1732.7921
3	1790.6906	1731.7220
4	1846.9259	1734.7001
5	1839.7273	1744.5038
6	1851.4180	1731.2303
7	1875.4363	1734.5897
8	1810.0846	1732.4699
9	1829.1427	1736.1494
10	1778.7890	1733.9263
11	1775.2053	1773.0134
12	1837.6448	1739.3202
13	1775.5364	1736.1511
14	1854.6080	1730.6923
15	1775.2645	1730.6923
16	1845.6264	1732.0388
17	1750.1638	1732.4699
18	1801.9826	1734.6118
19	1867.5913	1732.4699
20	1864.5641	1732.4699

Figure 38 shows the convergence behavior of SNOBFIT after the first iteration, for a few of the runs.

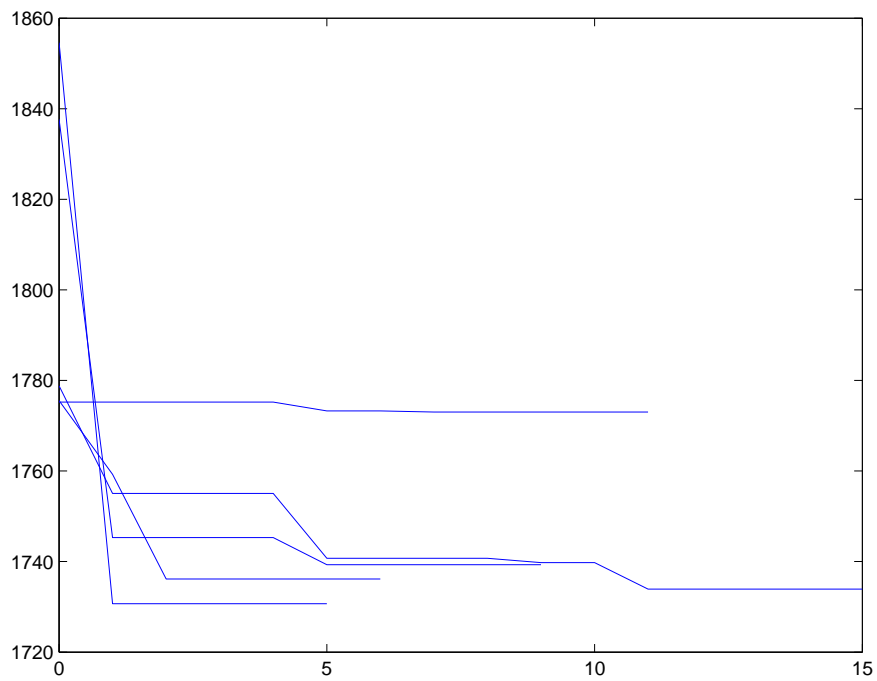


Figure 38: Progress by number of iterations in 5 different SNOBFIT searches.

The following two tables show the optimal choice found with OUU, and two corresponding scenarios, the scenario with the nominal values of the uncertain parameters and the scenario determined by the corner search, in which, for this choice, the worst case was attained.

variable	optimal robust choice
punt_I	328
punt_m_eng	6
punt_T_eng	530
punt_P_eng	0
punt_eta	0.2500
punt_d	1
punt_ro	3.1400
punt_alfa	0.9200
punt_spec	50
punt_dens	69.3000
punt_eff	0.7000
punt_alfa_sup	0.7000
punt_eps_sup	0.1300
punt_f	32.3000
punt_D	0.2000
punt_Eb	2.7000
punt_El	290
punt_rho	1856
punt_ult_str	320
punt_yie_str	290
punt_mem	8
punt_mem_mass	2
punt_mem_pow	6
punt_h_vs_r	0.6125

variable	nominal value	value for worst case
DR	22000	24200
GSD	15	14.25
H_target	1500000000	1575000000
H_terra	180000	189000
S	1500000000	1575000000
T_PP	10	11
Tday	15552000	16329600
Tdown	273.15	300.465
Tecl	0	0
Tup	313.15	344.465
ax_freq	15	15.75
ax_g	5	5.25
cicli	1	1.05
data_tot	0.33594	0.36953
deltaV	43	47.3
dens_dep_rad	2	2.2
lat_freq	10	10.5
lat_g	1.8	1.89
max_sun_dist	1.2	1.26
min_sun_dist	1	1.05
mzero	1188	1306.8
powzero	190	209
teta0	20	22
y	20	22

The next table contains for this choice the corresponding solution evaluated at the nominal scenario (i.e., without uncertainty), and at the scenario with worst case uncertainty. In particular, we can see the effect on the objective function.

variable	without corner search	worst case in OUU
m_tot	1.5661e+03	1.7307e+03
mass_DH	7.7000	7.7000
pow_DH	11	11
cost_DH	0	0
aff_DH	0	0
beam	3.2508	3.2508
mass_TTC	3.1449	3.2437
pow_TTC	2.1060	2.8300
cost_TTC	0	0
aff_TTC	0	0
diam_SA	2.0030	2.2401
P_SA	315.4116	346.0418
A_SA	3.2167	4.0234
T_SA	305.9935	297.6203
mass_SA	10.1005	12.6334
cost_SA	0	0
aff_SA	0	0
capacita	0	0
volume_batt	0	0
mass_batt	0	0
cost_batt	0	0
aff_batt	0	0
...		

variable	without corner search	worst case in OUU
pow_tot	223.4166	245.1129
mass_pow	12.1206	15.1601
cost_pow	0	0
aff_pow	0	0
diam	2.0030	2.2401
S_Thickness	7.4965e-04	8.1406e-04
height	0.6134	0.6860
area_tot	10.1622	12.7106
mass_str	12.8897	17.5073
cost_str	0	0
aff_str	0	0
H_planet	180000	189000
Gs_hot_2	1418	1418
H_min_IR	1.5000e+09	1.5750e+09
rad_area	12.9166	10.5030
mass_ther	62.6451	69.2277
pow_ther	0	0
c_ther	0	0
a_ther	0	0
m_fuel	19.5176	23.7112
m_tot_prop	1.6435e+03	1.8160e+03
mass_prop	2.3421	2.8453
pow_prop	0	0
N_prop	0	0
cost_prop	0	0
aff_prop	0	0
cost_tot	0	0
aff_tot	0	0
mass_harness	78.3064	86.5346
m_tot	1.5661e+03	1.7307e+03
m_tot_mb	1.5661e+03	1.7307e+03

The columns of the next table show the best point found with OWU (worst case found with corner search infeasible), another good point from OWU (worst case feasible) and an example choice where no fixed point has been found (only one for negative `pow_ther`):

variable	OWU best	OWU good	no feasible fixed point
<code>m_tot</code>	1.5660e+03	1.5757e+03	
<code>punt_I</code>	328	315	280
<code>punt_m_eng</code>	6	4.8000	2
<code>punt_T_eng</code>	530	450	110
<code>punt_P_eng</code>	0	0	0
<code>punt_eta</code>	0.2500	0.2500	0.1400
<code>punt_d</code>	1	1	2.2700
<code>punt_ro</code>	3.1400	3.1400	2
<code>punt_alfa</code>	0.9200	0.9200	0.6500
<code>punt_spec</code>	30.1000	43.3900	42
<code>punt_dens</code>	84.1200	76.8900	61
<code>punt_eff</code>	0.7200	0.7000	0.7000
<code>punt_alfa_sup</code>	0.2300	0.3500	0.2600
<code>punt_eps_sup</code>	0.0300	0.8400	0.5800
<code>punt_f</code>	32.3000	30	8.5000
<code>punt_D</code>	0.2000	0.7000	0.7000
<code>punt_Eb</code>	2.7000	10.3000	9.6000
<code>punt_El</code>	290	290	110
<code>punt_rho</code>	1856	1856	4430
<code>punt_ult_str</code>	320	320	900
<code>punt_yie_str</code>	290	290	855
<code>punt_mem</code>	8	64	128
<code>punt_mem_mass</code>	2	3	3
<code>punt_mem_pow</code>	6	8	14
<code>punt_h_vs_r</code>	1.0589	5.6000	4.9614
...			

variable	OWU best	OWU good
mass_DH	7.7000	9.9000
pow_DH	11	13.2000
cost_DH	0	0
aff_DH	0	0
beam	3.2508	1
mass_TTC	3.1449	5.6005
pow_TTC	2.1060	1.1256
cost_TTC	0	0
aff_TTC	0	0
diam_SA	1.5234	0.6644
P_SA	315.4116	317.3057
A_SA	3.2167	3.2360
T_SA	305.9935	305.9935
mass_SA	10.1005	10.1611
cost_SA	0	0
aff_SA	0	0
capacita	0	0
volume_batt	0	0
mass_batt	0	0
cost_batt	0	0
aff_batt	0	0
pow_tot	223.4166	224.7582
mass_pow	12.1206	12.1933
cost_pow	0	0
aff_pow	0	0
diam	1.5234	0.6644
S_Thickness	7.4158e-04	8.5763e-04
height	0.8066	1.8604
area_tot	7.5054	4.5767
...		

variable	OWU best	OWU good
mass_str	12.7509	14.8349
cost_str	0	0
aff_str	0	0
H_planet	180000	180000
Gs_hot_2	1418	1418
H_min_IR	1.5000e+09	1.5000e+09
rad_area	28.2001	0.9249
mass_ther	62.6380	63.0260
pow_ther	0	0
c_ther	0	0
a_ther	0	0
m_fuel	19.5154	20.4416
m_tot_prop	1.6433e+03	1.6534e+03
mass_prop	2.3418	2.4530
pow_prop	0	0
N_prop	0	0
cost_prop	0	0
aff_prop	0	0
cost_tot	0	0
aff_tot	0	0
mass_harness	78.2975	78.7825
m_tot	1.5660e+03	1.5757e+03
m_tot_mb	1.5660e+03	1.5757e+03

Our next figure illustrates the difference of the results computed with and without uncertainty.

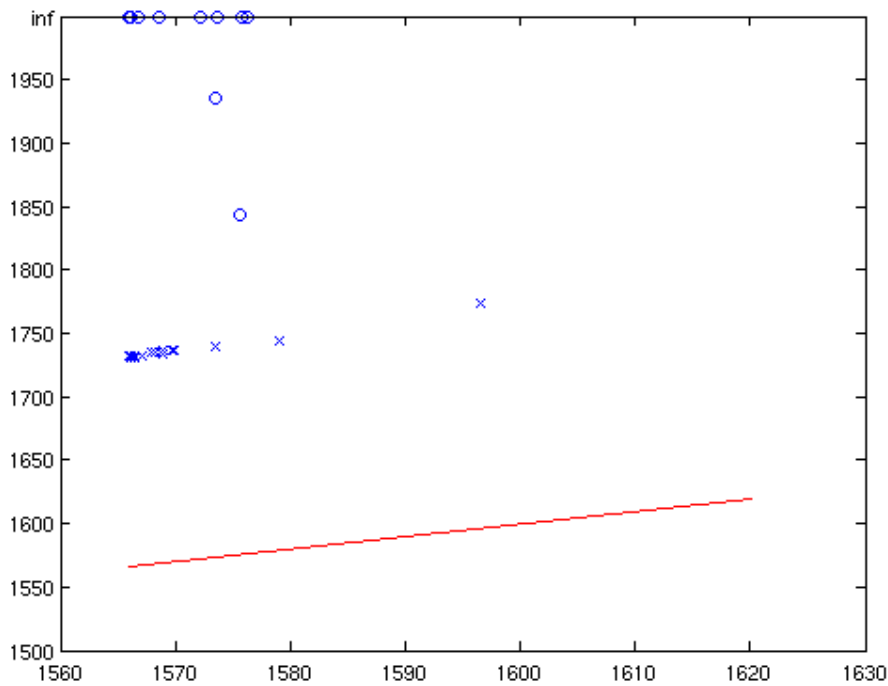


Figure 39: Circles denote designs found in different runs by optimization without uncertainty; crosses denote designs found by optimization with uncertainty. The horizontal axis displays $v_1 = m_{\text{tot}}(\text{design}, \text{nominal})$, the vertical axis $v_2 = \max_{\text{uncertainty}} m_{\text{tot}}(\text{design}, \text{uncertainty})$; Because of the different ranges of the coordinates, the points where $v_1 = v_2$ lie on the only slightly slanted red line drawn.

We chose 20 good choices found by minimizing the objective `m_tot` without taking uncertainty into account. For each of these choices we have the value of `m_tot` when all uncertain variables are fixed in the middle of their boxes v_1 , and additionally compute the worst case value v_2 and mark the point (v_1, v_2) . We also chose 20 good choices found by OOU. For each of those choices we again compute the value of `m_tot` with all uncertain variables fixed in the middle of their boxes v_1 and the worst case value v_2 and mark the point (v_1, v_2) .

By comparing with the red line, we see that for the robust solutions found (crosses running almost parallel), the difference between the worst case and the evaluation of m_tot with fixed uncertain variables appears to be almost constant, reflecting the effect of the variations in the domain of uncertainty.

On the other hand, only two of the choices found without uncertainty methods (circles) are feasible under all admissible uncertainties; and in these two cases, the worst case in the feasible range is much worse than that for the robust solutions found.

Moreover, by comparing the values for $v1$ for crosses and circles, we see that the best robust solutions (found by optimization under uncertainty) have at the nominal point values m_tot which are competitive with those of the best solutions computed without uncertainty.

We conclude that the quality of the optimal solutions does not differ significantly whether or not you take uncertainty into account, while the robustness is drastically improved.

12 Robustness

In the heuristic approach, the corner search is the means to do a robustness check. But as illustrated in Figure 40 the corner search may fail detecting hidden constraints, so we investigate the optimal choice with MCS which in contrast to the corner search also evaluates the interior of the boxes for the uncertain variables.

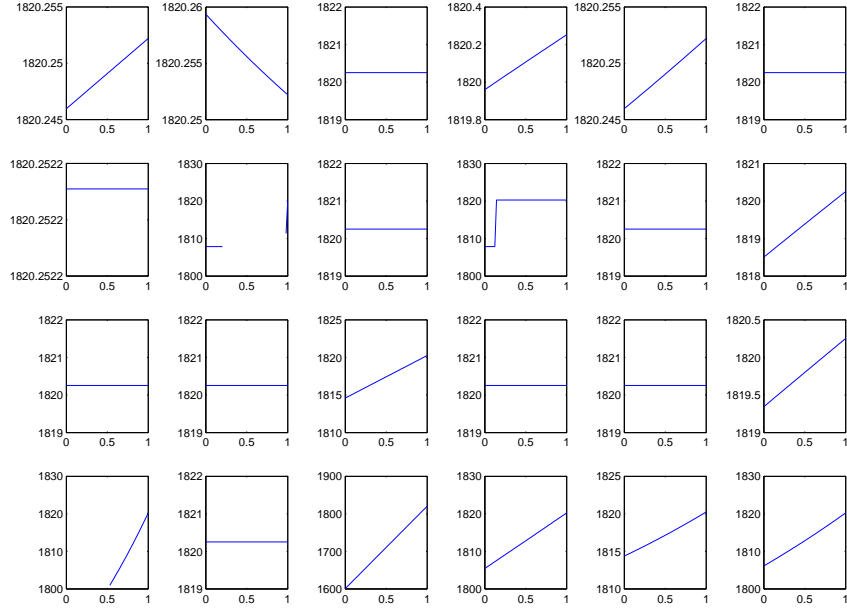


Figure 40: Unstable behavior at a choice displaying a jump discontinuity and two hidden constraints. For each uncertain variable $unc_i, i = 1..24$ (scaled) we fix all the other uncertain variables $unc_k, k \neq i$ at the midpoint of their boxes and plot $m_{tot}(unc_i)$.

From the corner search, one can also get an approximation to the gradient of m_{tot} with respect to the uncertain variables, which is exact in the limit of a linear dependence.

global input variables	partial derivatives of m_{tot}
DR	3.6251e-05
GSD	-0.12821
H_target	0
H_terra	0
S	1.2821e-09
T_PP	0
Tday	0
Tdown	0
Tecl	0
Tup	0
ax_freq	0
ax_g	1.416
cicli	0
data_tot	0
deltaV	0.61048
dens_dep_rad	0
lat_freq	0
lat_g	0.85064
max_sun_dist	66.8306
min_sun_dist	0
mzero	1.2886
powzero	0.19848
teta0	0.27904
y	0.43781

Figure 41 shows that the approximate linear dependence is a reasonable assumption in a typical robust case.

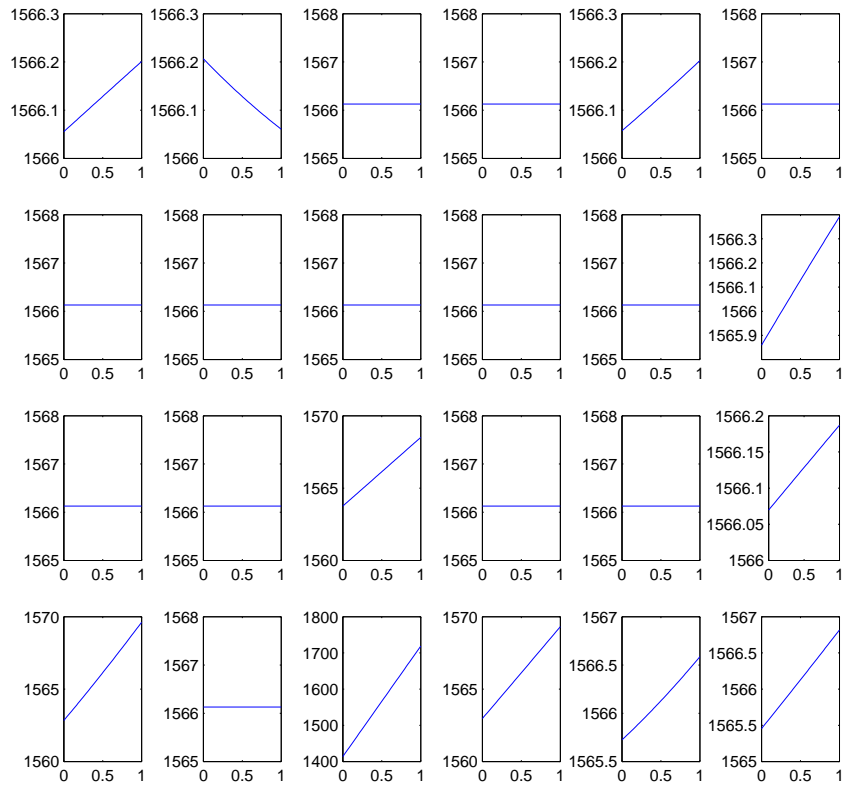


Figure 41: The same information as in Figure 40, but for a robust choice. Within the domain of uncertainty, the dependence on the uncertain parameters is essentially linear.

In the present application, the possible unreliability of the corner search is therefore entirely due to the lack of continuity introduced into the model by branching conditions whose consequences do not match at the switch-over point.

12.1 Consequences of assuming different probability distributions

By now the uncertainty information provided has yielded α -cuts for $\alpha = 100\%$ and a boxed shaped potential function. The consequences assuming different probability distributions will be discussed in the following example:

The boxes for the uncertain variables are shrunk to 70% of the original size and three different probability distributions are assumed. First we cut off 15% from each corner of a box, second we cut off 30% from one random corner of each box, third we cut off 30% from the opposite to the corner that we took as second distribution. The corner search is adapted to the new boxes and we evaluate our choices found with SNOBFIT for OWU and OOU and additionally do 5 SNOBFIT searches assuming the first probability distribution mentioned.

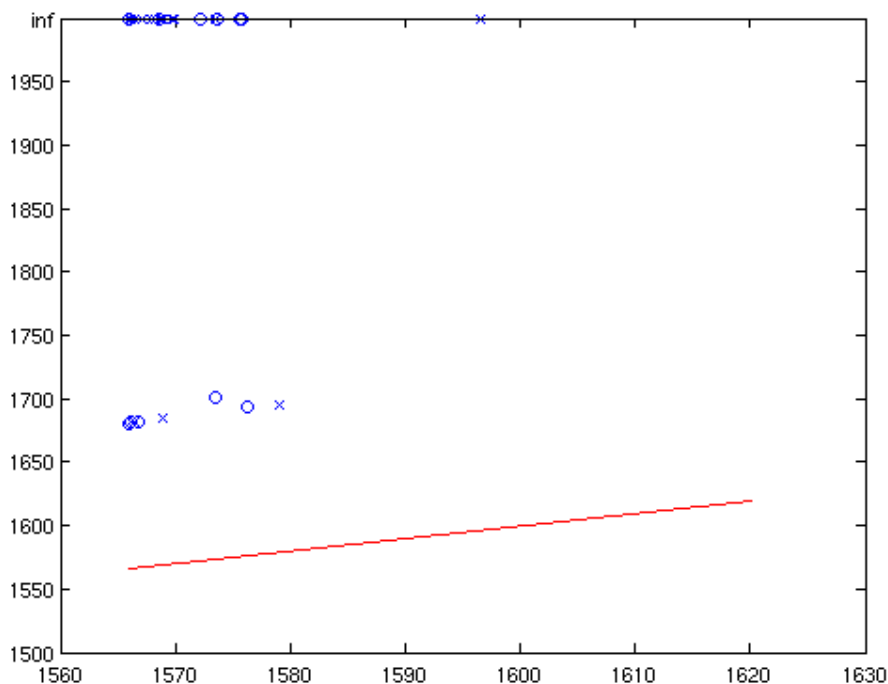


Figure 42: The same information as in Figure 39, but now the corner search is adapted to the new boxes of the first probability distribution assumption. Many OOU choices (crosses) are now infeasible.

It is consistent with intuitive expectations that a) some of the OWU choices that have been infeasible in OUU with the wider boxes now become feasible for the smaller boxes and b) the 5 brandnew points are infeasible for the original boxes.

It is on first sight counterintuitive and a matter of concern, however, that some choices formerly claimed feasible in OUU (where only some corners of the full box were inspected) are infeasible in OUU with the smaller boxes (cf. Figure 42). This should not happen with problems in which the structure of the model depends continuously on the uncertain parameters.

The infeasibility is caused by hidden constraints, cf. Figure 40, which invalidate the assumptions inherent in the corner search. The hidden constraints possibly arise from requirements like $\text{mass} \geq 0$: it has been observed that those choices suddenly turning infeasible often lead to fixed points with negative pow_ther . Possibly, this points to unrealistic simplifications in the models specified by by ESA.

In general, hidden constraints such as those shown in Figure 40 cannot be handled with heuristics:

Gaps in \mathbb{R}^{24} (24 uncertain variables) are hard to detect; already simple checks such as the computations leading to Figure 40 are quite expensive, and they explore by far not all possible uncertainty combinations in 24 dimensions. Even our check with MCS, whose global search is significantly more robust (and much more expensive than a corner search) cannot exclude them and may mistakenly claim a not everywhere in the uncertainty domain feasible solution to be feasible.

13 Conclusions and future directions

We studied symbolic and heuristic methods for handling uncertainty in space system design.

Semiautomatic conversion tools to input for the current generation of global solvers turned out not to be successful for the model problem under discussion. One reason is the heavy use of branching structures in the programs defining the model.

Heuristic techniques do handle those structures, but cannot cope with hidden

constraints, cannot guarantee global optimality, feasibility and infeasibility and lack a sophisticated integer implementation.

We were able to solve the robust optimization problem in case of interval uncertainties. This approach works satisfactorily for the model problem, except for the presence of hidden constraints.

Solving the model problem with uncertainty revealed significant robustness advantages of the approach using uncertainty, without significantly compromising the quality of the solutions at the nominal (certain) values of the parameters.

To improve the uncertainty treatment, more data are needed. The questionnaire gui is a tool to acquire these data, and turn it into quantitative uncertainty descriptions by means of clouds.

It is recommended to continue both paths, heuristics and symbolic solvers, to implement on the one hand better integer handling, on the other hand better control-structures models.

To avoid hidden constraints, whose presence is the major difficulty in the present heuristic approach, more care is needed in the formulation of the models to be solved. In particular, it is essential to present the models in such a way that all branches that result in discontinuities are avoided and replaced by explicit choices which can be controlled via inputs in the respective programs (rather than via decisions made within them). We believe that this can be done in each case by corresponding attention during the creation of the models.

References

- [1] W. Larson and I. Wertz, Space Mission Analysis and Design, Kluwer Academic Publishers, 3rd edition, 1999.
- [2] A. Neumaier, Clouds, fuzzy sets and probability intervals, *Reliable Computing* 10 (2004), 249–272.
- [3] A. Neumaier, Uncertainty modelling for robust verifiable design. Slides, 2004
<http://www.mat.univie.ac.at/~neum/papers.html#unc>

- [4] H. Schichl and A. Neumaier, Interval Analysis on Directed Acyclic Graphs for Global Optimization, *J. Global Optimization* 33 (2005), 541–562.
- [5] A. Ben-Tal and A. Nemirovski, Robust optimization—methodology and applications *Math. Programming* 92 (2002), 453-480.
- [6] AMPL, A Modeling Language for Mathematical Programming
<http://www.ampl.com/>
- [7] NLEQ,
http://www.zib.de/Numerik/numsoft/CodeLib/codes/nleq1_m/nleq1.m
- [8] MCS,
www.mat.univie.ac.at/~neum/software/mcs/
- [9] DAKOTA,
<http://www.cs.sandia.gov/DAKOTA/>
- [10] NOMADm,
<http://www.afit.edu/en/ENC/Faculty/MAbramson/NOMADm.html>
- [11] CONDOR,
<http://iridia.ulb.ac.be/~fvandenb/optimization/CONDORdownload.html>
- [12] SNOBFIT,
<http://www.mat.univie.ac.at/~neum/software/snobfit/>
- [13] LINGO,
<http://www.lindo.com/products/lingo/>