Interface between INVERSE and Mathematica

(FOR VERSION 3.11)

Igor Grešovnik, Stanislav Hartman, Domen Cukjati

Ljubljana, 25 April, 2006

Contents:

6. INTERFACE BETWEEN MATHEMATICA AND INVERSE

6.1 Running INVERSE

6.1.1 As stand alone program

Inverse can be run as stand alone program. In this case entire funcionalitity of the program is provided, only functions which uses Mathematica (through MathLink) are not available.
Program is run from command prompt:
»*programname filename*«
Programname is name of inverse and filename is name of command file.

6.1.2 As a program linked to Mathematica

On the other hand, Inverse can also be run as a program linked to Mathematica (Wolfram Research), which enables that both functionalities (of Inverse and Mathematica) can be used. In this case Inverse is run from the notebook by the command:
»*link=Install["programname filename"]*«
Programname is name of inverse and filename is name of command file.
Afterthat Inverse can be used in two ways:

- **Modul of functions**. User can use several functions (InvNewVar, InvSetVar, InvGetVar, InvDelVar, InvInverse, ...) as they are functions defined in Mathematica. By this way Inverse apears as a modul of several functions.
- **Interpreter**. User can set up whole inverse/optimisation procedure in Inverse command file (in the same way as for stand alone version), where also functions which are defined in Mathematica can be used through a small interface. To run Inverse's interpreter from Mathematica's notebook command »*InvInterpret[]*« should be executed in the notebook.

6.2 Test cases

Test cases are shown and described in directory
»*...\ighome\ex\intfc\mathematica*«. Manuals for test cases are in teh file
»`manual.doc`«.

6.3 Interpreter Functions for accessing Mathematica Functionality

6.3.1 **mathsetvariable0 *{*** *mathvarspec*, *subspec* ***},*** **mathsetvariable0 *{*** *command* ***}***

The function can take one or two arguments. When called with two arguments, it assigns a value to a variable defined in Mathematica. The first argument *mathvarspec* is a variable specification argument that specifies the name of the Mathematica variable that is set, eventually with indices of a sub-part of that variable. The second argument *subspec* is a specification of an interpreter variable or a sub-table of its elements. Function works for *string*, *counter*, *scalar*, *vector* and *matrix* variables. It *subspec* refers to a multi-dimensional table of elements then a nested list of values of these elements is assigned to the variable. String values are represented in Mathematica as strings, counter values as integer numbers, scalars values as numbers, vector values as lists of numbers, and matrix values as two dimensional lists of numbers. More precisely, matrices are represented as lists of depth 3 whose elements are lists of numbers, all with the same dimension. Elements of the list are rows of the matrix.

In the case that there is only one argument, the function executes a command in Mathematica. In this case, the only argument *command* must be a string that is interpreted as a statement (command) that is executed in Mathematica.

Example:

```
*{ Create a vector variable (of rank 2) containing 2*3 vector
elements: }
newvector{v [3 2]}
*{ Assign individual elements (note different formats for listing
vector elements): }
setvector { v[1 1]  4 { 11.1 11.2 11.3 11.4 } }
setvector { v[1 2]  2 { 12.1 12.2 } }
setvector { v[2 1]  3 { 1 : 21.11 } { 2 : 21.22 } { 3 : 21.33 } }
setvector { v[2 2]  1 { 22.001 } }
setvector { v[3 1]  2 { 31.1  31.2 } }
setvector { v[3 2]  4 { 3 : 3.3 } { 4 : 4.4 } }
*{ Assign a list of 3 vectors contained in sub-table v[1] to
Mathematica variable veclist1: }
```

```
mathsetvariable0{ "veclist1" , v[1]}
*{ Execute a command (represented by the string argument) in
Mathematica – printing of the newly set variable }
mathsetvariable0{"Print\3\d veclist1= \d,vec1\4"}
*{ Assign a nested list (of dimensions 3x2) to Mathematica variable
veclist2: }
mathsetvariable0{ "veclist2" , v }
*{ Assign a (single) vector v[3,1] to Mathematica variable vec: }
mathsetvariable0{ "vec1" , v[3,1] }
*{ Execute a command (represented by the string argument) in
Mathematica – printing of the newly set variable }
mathsetvariable0{"Print\3\d vec1= \d,vec1\4"}
```

The first argument *mathvarspec* can refer to element of a list variable, but in this case variable must already be defined in Mathematica with list of sufficient dimensions assigned to it. For example, in *Mathematica* ve can have a statement (that has already been executed)

```
listv={1,3,4}
```

and execute the following code in *Inverse*:

```
setvector { v 3 {1.1, 1.2, 1.3} }
mathsetvariable0 { listv[2], v }
```

Execution of this code will change the contents of the variable listv in Mathematica to

listv = { 1, {1.1, 1.2, 1.3}, 4 }

6.3.2 **tostringlist** ***{*** *subspec*, *strelspec* ***}***

This function stores an interpreter variable (defined in *Inverse*) in a string form understandable to *Mathematica*. The first argument *subspec* is a specification of table of variable elements to be stored in a string. Interpreter variables of types *string*, *counter*, *scalar*, *vector* abd *matrix* are supported. The second argument *strelspec* is the specification of a string element where the string representation will be stored. If *strelspec* does not contain any indices and the variable specified by this argument does not exist, then the function creates a new string variable (of rank 0 and with specified name) and stores the string representation into the only element of this variable[1].

[1] If the argument *strelspec* contains any indices, then the specified variable must be an existent string variable of such dimensions that the specified indices are in the range.

This function is especially useful for composition of strings that represent commands to be executed in *Mathematica* (e.g. by the *mathsetvariable0* function).

Example:

```
*{ Create a new vector variable and assign its elements: }
newvector{v [3 2]}
setvector { v[1 1]  4 { 11.1 11.2 11.3 11.4 } }
setvector { v[1 2]  2 { 12.1 12.2 } }
setvector { v[2 1]  3 { 1 : 21.11 } { 2 : 21.22 } { 3 : 21.33 } }
setvector { v[2 2]  1 { 22.001 } }
setvector { v[3 1]  2 { 31.1  31.2 } }
setvector { v[3 2]  4 { 3 : 3.3 } { 4 : 4.4 } }

newvector {v [2 3]}
setvector { v[1 1]  4 { } }
setvector { v[1 2]  2{ 1.1 2.2 } }
setvector { v[1 3]  4{ 3 : 3.3 } { 4 : 4.4 } }
setvector { v[2 1]  4 { } }
setvector { v[2 2]  2{ 1.1 2.2 } }
setvector { v[2 3]  4{ 3 : 3.3 } { 4 : 4.4 } }
*{ Store a string representation of a sub-table of variable
elements to a zero-rank interpreter variable (which is created anew
since it does not exist yet): }
tostringlist { v[2], nizgv }
printstring { nizgv }
*{ Create a new string variable (with 3*4 string elements) and
assign string representation of a single vector element to element
[2,3] of that variable: }
newstring { str1[3,4] }
tostringlist { v[3,1], str1[2,3] }
printstring { str1[2,3] }
```

6.3.3 **mathfunc** ***{*** *mathfunc*, *numargin*, *numargout*, *argin1*, *argin2*, …, *argout1*, *argout2*,…***}***

Executes a *Mathematica* function from *Inverse*. The string argument *mathfunc* is the name of the function to be called in Mathematica, the value argument *numargin* is the number of function input arguments and the value argument *numargout* is the number of output argument of the called function. The rest are two groups of arguments: In the first group (*argin1*, *argin2*, …) are input arguments of the function and in the second group (*argout1*, *argout2*, …) are output arguments of the function.

6.3.4 mathexecute *{ expression <notify> }*

Evaluates (executes) *expression* (which is a string argument) in Mathematica. If a counter argument *notify* is specified and it is not zero then a notice about the action is printed to the standard output and the program output file (if open). Results of the evaluation are not used by this function.

Example

mathexecute { “a=Pi/2” }

Warning:

When stating string arguments directly in double quotes, one must be careful about stating brackets and quotes. All brackets must be closed, otherwise they should be replaced by the appropriate escape sequences. Escape sequences used in *Inverse* are described in the manual about flow control.

6.3.5 mathgetscalar *{ expression elspec }*

This function evaluates *expression* (specified as string argument) in Mathematica and assigns its value to the element of a scalar variable specified by *elspec*.

Expression must evaluate to a real or integer number, otherwise the function call reports an error.

Argument *elspec* must either specify an existent element of a scalar variable, or eventually a non-existent zero-rank variable (in which case a scalar variable with a given name is created anew).

Examples:

Assume that the following code has been evaluated in Mathematica:

```
a={1, 1.1, 2.2};
b=3;
```

The following code in Inverse then makes sense:

```
*{ Create a scalar variable of rank 1 and dimension 3: }
newscalar { svar [3] }
```

```
*{ Assign values of valid Mathematica expressions that evaluate
either to integer or real numbers, to elements of the created
scalar variable svar: }
mathgetscalar { "a[[1]]", svar[1] }
mathgetscalar { "a[[2]]+b*a[[3]]", svar[2] }
*{ Assign expression value to a newly created scalar variables
(which must be of rank 0, i.e. no indices specified): }
mathgetscalar { "Cos[b]*a[[1]]" , x[] }
mathgetscalar { "3*Pi/2", y }  *{ square brackets can be omitted
when element specification contains no indices }
```

The following code would be invalid:

```
mathgetscalar { "a[[4]]", z[] }
```

because Mathematica variable a has only 3 elements.

Expressions to be evaluated can be composed by using string operations provided in Inverse, for example:

```
newstring { varname [2] }
setstring { varname[1], "mvar" }
setstring { varname[2], "mvar" }
={ i:24 }
stringappend {varname[1] $i  }
={ i:i+1 }
stringappend { varname[2] $i }
newstring { expr[] }
stringwrite { expr[], #varname[1] "+" #varname[2] }
mathgetscalar { #expr[] result[] }
```

In this case, a zero rank scalar variable result is created and the value of Mathematica expression "*mvar24+mvar25*" is assigned to this variable. See the manual on file interpreter variables for more details about the string operations used in the above example. Of course, if *Mathematica* variables mvar24 and mvar25 do not exist then 0 is assigned to these variables.

6.3.6 mathgetcounter *{ expression elspec }*

6.3.7 mathgetstring *{ expression elspec }*

6.3.8 mathgetvector *{ expression elspec }*

6.3.9 mathgetmatrix *{ expression elspec }*

These functions are similar to *mathgetscalar{}*, but they assign result of evaluation of expression to different types of variables. Type and structure of the result of evaluation must be compatible with the function. For example, for *mathgetmatrix{}* , expression must evaluate to a list of depth 2 whose elements have numerical values.

Example:

```
setstring{expr "{{1/3,2/3},{2.1,2.2},{3.1,3.2}}"}
mathgetmatrix{ #expr vmat}
dwrite{"Value of expression \d" #expr "\d:\n " }
printmatrix{ vmat }
```

Execution of the above code will cause the following output:

```
Value of expression "{{1/3,2/3},{2.1,2.2},{3.1,3.2}}":


Matrix vmat:
Dimension: 3*2
Elements:

m[1,1]= 0.33333333
m[1,2]= 0.66666667

m[2,1]= 2.1
m[2,2]= 2.2

m[3,1]= 3.1
m[3,2]= 3.2
```

6.3.10 mathdebugmode *{ <setmode> }*

It a counter argument *setmode* is specified and is 0 then debug mode for interface functions is switched off. Otherwise debug mode is switched on (i.e. if function is called without arguments or the argument is non-zero).

This affect some function of the interface with Mathematica (*mathexecute*, *mathgetscalar* and alike). In debug mode, these functions generate some additional terminal output describing the execution steps, which enables easier identification of reasons for unexpected behavior.

Example:

```
*{ Set the debugging mode: }
mathdebugmode{1}
mathdebugmode{}
setscalar{ doset 1 }   mathdebugmode{ #doset }
={ doset: 1 }   mathdebugmode{ $doset }
*{ Unset the debugging mode: }
mathdebugmode{ 0 }
={ doset: 0 }   mathdebugmode{ $doset }
```

6.3.11 mathanalyse *{ }*

This function is specialized version of previous one.
Function calculates all necessary values for optimization (objective, constraints, gradients of objective, gradients of constraints) and writes proper values into proper variables (objectivemom, constraintmom, gradobjectivemom, gradconstraintmom).
Function executes block "`mathanfunc`", which must be previously defined in Mathematica (kernel).
Format of output in Mathematica must be as follows:
{ calcobj, obj, calcconstr, constr, calcgradobj, gradobj, calcgradconstr, gradconstr, clientdata}

Variable	Type (in Mathematica)	Description
Calcobj	Integer	1 if obj is calculated and 0 if it is not

obj	Real	objective value
calcconstr	Integer	1 if constr is calculated and 0 if it is not
constr	List of Reals	constraints
calcgradobj	Integer	1 if gradobj is calculated and 0 if it is not
gradobj	List of Reals	gradients of objective
calcgradconstr	Integer	1 if gradconstr is calculated and 0 if it is not
gradconstr	List of Lists of Reals	gradients of every constraint
clientdata	Integer	anything (at the moment)

Function works ONLY if link with Mathematica is established.

Example:

```
mathanalyse{}
```

6.4 Mathematica Functions for accessing Interpreter Functionality

6.5 InvExecute [code]

Executes the *code*, which must be a string argument, by *Inverse* interpreter.

Example:

```
(* Assign 3D Vector of Mathematica (list of 3 numbers): *)
v={1,2*10^(-33),3};
(* Compose commands that copies the above vector to variable vec1:
*)
command=StringJoin[
      "setvector{ vec1[], ",
      ToString[Length[v]],
      " { "
      ];

For[i=1,i≤Length[v],i=i+1,
    command=StringJoin[command,
          ToString[CForm[v[[i]]]],
          " "
          ];
    ];
command=StringJoin[command,"} } \n\n"];
Print["Command ",command, "is being executed in Inverse.\n"];
(* Execute the command: *)
InvExecute[command]
(* Check whether vector variable has been set correctly: *)
checkv=InvGetVar["vector","vec1",{}]
```

6.5.1 InvNewVar *[type, name, indexes]*

Function creates variable of type *type* with name *name* and indexes *indexes*, which is then defined in Interpreter.

Currently available *types* are: counter, scalar, vector, matrix, string and vfile.

Indexes must be a list of integer numbers, which represent indexes of the variable. They can be also empty list "*{}*" (setting variable with rank 0

Function works ONLY if MathLink is established.

Example:

```
InvNewVar["vector","vec0",{2,2,2}]
```

6.5.2 InvSetVar *[type, name, indexes, value]*

Function sets *value* to variable of type *type* with name *name* and indexes *indexes*, which must be previously defined in Interpreter.

Currently available *types* are: counter, scalar, vector, matrix, string and vfile.

Indexes must be a list of integer numbers, which represent indexes of the variable. They can be also empty list "*{}*" (setting variable with rank 0). If *indexes* are only zero "*{0}*", function returns number and range of indexes.

If variable with name *name* and type *type* does not exists yet AND *indexes* are empty list "*{}*" (setting variable with rank 0), then variable is automatically created and values set to the variable.

Values must be of following format:

Type of variable	counter	scalar	vector	matrix	string	vfile
Format of *value*	number	number	list of numbers	list of list of numbers	string	not yet implemented!

Function works ONLY if MathLink is established.

Example:

```
InvSetVar["vector","vec0",{2,2,2}, {11,12,13,14,15,16}]
```

6.5.3 InvGetVar *[type, name, indexes]*

Function gets variable defined in Inverse.

Currently available *types* are: counter, scalar, vector, matrix, string and vfile. *Name* is name of variable as defined in Interpreter. *Indexes* must be a list of integer numbers, which represent indexes of the variable.

If *indexes* is a list containing only one zero (*{0}*) then function returns dimension of the variable. In this case, *type* can be "" or "*any*", in which case dimensions of the variable are returned (if existent) regardless of its type. Dimensions are returned as a list of indices (which is empty if variable range is 0). If the variable does not exist then a string is returned rather than a list.

If *indexes* is a list containing a single negative integer (e.g. *{-1}*) then the function returns a string representing a *variable type* (long form).

Function returns values of these formats:

Type of variable	counter	scalar	vector	matrix	string	vfile
Format of value	number	number	list of numbers	list of list of numbers	string	not yet implemented!

Function works ONLY if MathLink is established.

Examples:

```
InvGetVar["vector","vec0",{2,2,2}]
```

The following code defines a Mathematica function that evaluates to **`True`** if a variable with a given name and type (type can be omitted) exists and to **`False`** if it does not:

```
InvVarExists[type_,name_]:=ListQ[ InvGetVar[type,name,{0}] ];
InvVarExists[name_]:=ListQ[ InvGetVar["any",name,{0}] ];
```

""

After evaluation of the above code in Mathematica, we can use the function in expressions like

```
If [  InvVarExists [”vector”, ”parammom”],
  Print[”Vector variable parammom exists. ”];
]
If [  InvVarExists [”any”, ”str”],
  Print[”Variable str exists. ”];
]
If [  InvVarExists [”a”],
  Print[”Variable a exists. ”];
]
```

6.5.4 InvDelVar *[name]*

Function deletes variable *name*, which is defined in Inverse.
Function works ONLY if MathLink is established.

Example:

```
InvDelVar["v"]
```