TomSym

TomSym

TOMSYM is a symbolic manipulation package which is part of the TOMLAB optimization software suite. It simplifies the work of defining optimization problems by automatically generating derivatives, m-code and other things needed for the numeric solvers to work as efficiently as possible.

TOMSYM is also the foundation on which PROPT is built. We recommend that PROPT users read through this guide first, before continuing on to the PROPT manual.

Solving optimization problems

An optimization problem is defined by an objective function and a set of constraints. Problems are usually classified depending on the properties of the objective and constraints, and different types of numeric techniques are used for different types of problems. For example, solvers usually handle quadratic objectives and linear constraints more efficiently than general nonlinear ones.

TOMSYM provides the `ezsolve` function for solving any type of optimization problem. It examines the objective and constraints to determine the problem type, and then automatically selects the numeric solver which it deems most apropriate.

For example, here we find a minimum to a second degree polynomial in two variables

```
>> toms a b
>> ezsolve(a^2 + 4*b^2 + 3*a - 2*b - a*b)
Problem type appears to be: qp
Time for symbolic processing: 0.040062 seconds
Starting numeric solver
=====* * * ====================================================================
TOMLAB - Tomlab Optimization Inc. Development license  999001. Valid to 2013-02
==============================================================================
Problem:  1:                                    f_k      -2.266666666666666607
                                             f(x_0)       0.000000000000000000

Solver: CPLEX.  EXIT=0.  INFORM=1.
CPLEX Barrier QP solver
Optimal solution found

FuncEv    2 GradEv    2 ConstrEv    2 Iter    2
CPU time: 0.010000 sec. Elapsed time: 0.002605 sec.

ans =

    a: -1.4667
    b: 0.0667
```

Looking at the output from the solver, we see that `ezsolve` determined the problem type to be QP (quadratic programming) and selected CPLEX at the solver. The problem was then solved numerically and the solution is returned as a matlab structure with fields corresponding to the symbol names.

Most solvers handle the linear constraints separately from the nonlinear ones. With TOMSYM, this happens transparently. The linear constraints are collected and passed to the solver in the form of a (sparse) matrix, while the

nonlinear ones are passed as m-code, together with m-code for their derivatives.

Symbolic matrices

With TOMSYM, a single symbol can be a scalar, vector or matrix. Expressions that involve matrix symbols are stored only once. This behaviour is defferent from many symbolic algebra packages, where symbolic arrays are treated as arrays of symbols (i.e. one symbolic expression is stored for each position in the array).

Symbolic matrices can be manipulated just like normal MATLAB matrices. For example, `*` gives the matrix product, and `.*` gives the element-wise product of two matrices.

```
>> A = tom('A',2,2)

A = tomSym(2x2):

   A

>> b = tom('b',2,1)

b = tomSym(2x1):

   b

>> y = A*b

y = tomSym(2x1):

   A*b
```

In this example `A` is 2-by-2 matrix symbol, `b` is a 2-by-1 column vector symbol, and the matrix expression `y` is also a 2-by-1 column vector.

Taking the derivative of a vector expression with respect to a vector symbol gives the so-called Jacobian matrix.

```
>> derivative(y,b)

ans = tomSym(2x2):

   A
```

Derivatives involving matrices are computed as if the matrices were vectors, with elements taken in column-first order.

```
>> derivative(y,A)

ans = tomSym(2x4):

   kron(b',eye(2))
```

A consequence of this definition is that the derivative of a matrix with respect to itself is the identity matrix.

```
>> full(derivative(A,A))
```

```
ans =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Vectorized code

In languages such as C or FORTRAN, "loops" are very common occurences. Matlab, on the other hand, uses a syntax where loops can usually be replaced by *vectorized* expressions. An example is the matrix product, which can be coded in C using three nested for loops, but in Matlab is accomplished by simply using the operator `*`.

tomSym is intended for use with vectorized code. It supports all the functions that are typically needed to generated vectorized expressions, such as `sum`, `diff`, `repmat`, `sparse`, etc.

Article Sources and Contributors

TomSym *Source*: http://tomwiki.com/index.php?oldid=879 *Contributors*: Elias, Per