MathLink Connection to C−XSC Function ParLinSolve

E. Popova, IMI at BAS
epopova@bio.bas.bg

In this notebook we demonstrate *MathLink* connection between *Mathematica* and an external C−XSC function ParLinSolve() [1,2]. *MathLink* is an open interprocess communication protokol allowing high−level communication between *Mathematica* and external programs [4].

ParLinSys is a *MathLink*−compatible external program [3] which allows integration of a C−XSC function ParLin−Solve() into *Mathematica*. For more details of the implementation see the template file ParLinSys.tm and the source file ParLinSys.cpp.

Here we do not discuss the *MathLink* technology (see [3, 4]) but demonstrate the execution of C−XSC functions from within a *Mathematica* session.

Usage

The following installs an executable *MathLink*−compatible external program.

```
In[1]:= lnk = Install["ParLinSys"]
```

```
Out[1]= LinkObject[./ParLinSys, 2, 2]
```

The **Install** function establishes the connection and makes all definitions necessary for the execution of the external function visible for the *Mathematica* session.

Let us see the newly available functions and the patterns that can be evaluated on the particular link

```
In[2]:= LinkPatterns[lnk]
```

```
Out[2]= {ParLinSolveTB[p_Integer, flag_Integer,
    ap_?MatrixQ, bp_?MatrixQ, ip_?MatrixQ]}
```

and the corresponding usage message

In[3]:=
```
?? ParLinSolveTB
```

```
ParLinSolveTB[p, SharpC, Ap, bp, pval] copmutes
  verified    enclosure for the solution set of a square
  parametric linear system by a C-XSC module ParLinSys.
 'p' is the number of the parameters;
'SharpC' integer flag: 1 specifying sharp
  enclosure for the iteration matrix, 0 otherwise;
 'Ap' is the coefficient matrix in factoirized
  representation (A_0, A_1, . . ., A_p);
 'bp' right-hand side vector in the numerical
  matrix representation (b_0,...,b_p);
 'pval' interval values  for the parameters
  specified by their end-points in a px2 matrix.

ParLinSolveTB[p_Integer, flag_Integer,
  ap_?MatrixQ, bp_?MatrixQ, ip_?MatrixQ] := ExternalCall[
  LinkObject[./ParLinSys, 2, 2], CallPacket[0, {p, flag, ap, bp, ip}]]
```

For the end−users, it is usually more convenient to define parametric matrices and parametric vectors symbolically.

In[4]:=
```
mat = {{3, p1}, {p1, 3}}; vec = {p2, p2};
```

Below we define a *Mathematica* function parToNumMLData, which transforms a parametric matrix, or a parametric vector, whose elements depend affine−linearly on given parameters into a numeric matrix suitable for input for the C−XSC function ParLinSolve, respectively the external program ParLinSolveTB.

In[5]:=
```
parToNumMLData::usage =
  "parToNumMLData[data, pars] transforms a parametric matrix of
    dimension (n1, n2), whose elements depend affine-linearly
    on the parameters contained in the list pars of dimension p,
    into a numeric matrix of dimension (p+1, n1) for n2=1 and
    dimension (n1(p+1), n2) otherwise. The output numeric matrix
    is intended to be an input for the C-XSC module ParLinSys.";

parToNumMLData[m_?MatrixQ, pars_List] :=
  Join[ReplaceAll[m, # → 0 & /@ pars],
   Flatten[Function[p, Map[Coefficient[#, p, 1] &, m, {2}]] /@ pars, 1]];

parToNumMLData[b_?VectorQ, pars_List] :=
  Join[{ReplaceAll[b, # → 0 & /@ pars]},
   Function[p, Coefficient[#, p, 1] & /@ b] /@ pars];
```

Transforming our symbolic data we get the numerical input form of the parametric matrix/vector required by ParLinSolveTB.

In[8]:=
```
Ap = parToNumMLData[mat, {p1, p2}]
```

Out[8]=
```
{{3, 0}, {0, 3}, {0, 1}, {1, 0}, {0, 0}, {0, 0}}
```

In[9]:=
```
bp = parToNumMLData[vec, {p1, p2}]
```

Out[9]=
```
{{0, 0}, {0, 0}, {1, 1}}
```

For the parameter interval values, we just define a list of interval end−points in the same parameter order {p1, p2} as specified by the second argument of parToNumMLData. This is because the interval constructors in C−XSC provide

directed outward rounding for the interval end−points and the *Mathematica* function Interval, if applied, would introduce extra rounding.

In[10]:=

```
pVals = {{1, 2}, {10, 10.5}};
```

Now, we are ready to call our external function. *Mathematica* function **InputForm** is used to show all the digits of the result.

In[11]:=

```
res = ParLinSolveTB[2, 1, Ap, bp, pVals]
res // InputForm
```

Out[11]=

```
{{1.79264, 2.76292}, {1.80188, 2.75367}}
```

Out[12]//InputForm=

```
{{1.792638317329675, 2.762917238225881}, {1.8018848752730778,
2.753670680282478}}
```

The result of ParLinSolveTB function is not a list of *Mathematica* interval objects but a list involving just the interval end−points, corresponding to the interval vector generated by the C−XSC function. The goal is the same as for the input intervals: to avoid an extra outward rounding introduced by the *Mathematica* function **Interval**.

In[13]:=

```
(Interval[#] & /@ res) // InputForm
```

Out[13]//InputForm=

```
{Interval[{1.7926383173296747, 2.7629172382258815}],
Interval[{1.8018848752730776, 2.7536706802824784}]}
```

Triggering error messages

Specifying the first argument as a real number will not match the function pattern, see the output from Link−Patterns[lnk] or the :Pattern: line in the template file.

Thus the function remains unevaluated.

In[14]:=

```
ParLinSolveTB[2., 1, Ap, bp, pVals]
```

Out[14]=

```
ParLinSolveTB[2., 1, {{3, 0}, {0, 3}, {0, 1}, {1, 0}, {0, 0}, {0, 0}},
 {{0, 0}, {0, 0}, {1, 1}}, {{1, 2}, {10, 10.5}}]
```

Now,we demonstrate triggering *MathLink* communication error messages, by calling the function with a very big value for the first integer argument.

In[15]:=

```
ParLinSolveTB[2^100, 1, Ap, bp, pVals]
```

```
ParLinSolveTB::mlink :
 Low-level MathLink error: machine number overflow.
```

Out[15]=

```
$Failed
```

Once the link is closed no one external program can be executed even with appropriate data

In[16]:=

```
ParLinSolveTB[2, 1, Ap, bp, pVals]
```

```
ParLinSolveTB::mlink :
 Low-level MathLink error: MLGet out of sequence.
```

Out[16]=

```
$Failed
```

until the external program be installed again.

In[17]:=

```
lnk = Install["ParLinSys"]
```

Out[17]=

```
LinkObject[./ParLinSys, 5, 3]
```

Input data are checked for consistency within the external program. Specifying a wrong number of parameters, we get

In[18]:=

```
ParLinSolveTB[3, 1, Ap, bp, pVals]
```

```
ParLinSolveTB::data : Incompatible input data.
```

Out[18]=

```
$Failed
```

In[19]:=

```
lnk = Install["ParLinSys"]
```

Out[19]=

```
LinkObject[./ParLinSys, 6, 4]
```

In order to demonstrate the computational error messages in action, we call the function ParLinSolveTB with a large interval for the first interval parameter.

In[20]:=

```
ParLinSolveTB[2, 1, Ap, bp, {{1, 20}, {10, 10.5}}]
```

```
ParLinSolveTB::cond :
 Verification failed, system is probably ill-conditioned.
```

Out[20]=

```
$Failed
```

In[21]:=

```
Uninstall[lnk]
```

Out[21]=

```
./ParLinSys
```

References

[1] Hofschuster, W., Kraemer, W.: C−XSC 2.0: A C++ Library for Extended Scientific Computing. In Alt, R., Frommer, A., Kearfott, B., Luther, W. (eds.): Numerical Software with Result Verification, LNCS 2991, Springer−Verlag, Heidelberg, pp. 15−35, 2004.

[2] Popova, E.D., Kraemer, W.: Parametric Fixed−Point Iteration Implemented in C−XSC. Preprint BUW−WRSWT 2003/3, WRSWT, Bergische Univ., Wuppertal, 2003.

[3] Popova, E.: On the Interoperability between Interval Software, presented at Dagstuhl Seminar 08021 "Numerical Validation in Current Hardware Architectures", Schloss Dagstuhl, Germany, January 6−11, 2008.

[4] Wolfram Research Inc.: *MathLink* for UNIX Developer Guide, Version 4, Revision 14, Wolfram Research Inc., Champaign, IL, December 15, 2004

February, 2008.