A System for Interfacing MATLAB with External Software Geared Toward Automatic Differentiation

02. Sept. 2006 - ICMS 2006 - Castro-Urdiales

H. Martin Bücker, Andre Vehreschild

RWTH Aachen University, Institute for Scientific Computing, Aachen, Germany

Atya Elsheikh

University of Siegen, Department of Simulation, Siegen, Germany

Introduction

1. Introduction
2. Linking simulation code and MATLAB
3. AMG
4. Automatic differentiation concepts
5. Linking of differentiated codes
6. A real world example
7. Conclusions

What is MATLAB?

- Matrix Laboratory – The Mathworks, Inc.
- Multipurpose, multidisciplinary, extensible, mathematical software package
- Every object is a matrix.
- Mathematical syntax – easy to learn.
- Professional plotting capabilities

An (artificial) simulation

MATLAB

+

Simulation code

```
double foo(double x) {
  return sin(x)+ x*2;
}
```

- Adding non-MATLAB codes possible
- MATLAB External Interface – MEX
- Enables the use of e.g. C/Fortran codes like MATLAB functions

Interfacing with MATLAB

```
#include <mex.h>
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]) {
   double *x, *y;
   /* Get input argument */
   x = mxGetPr(prhs[0]);
   /* Allocate storage for output argument */
   plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
   y = mxGetPr(plhs[0]);
   y[0]= foo_impl(x[0]);
}
```

- 8 lines of code (w/o comments) for handling one scalar argument and one scalar result.
- Even more lines of code, when handling matrices and more than one argument or result.

Automated MEX-file generation

- **Changing the simulation code may need changes of the MEX-interface.**
- **Updating the MEX-interface manually is error-prone and tedious.**
- **The Automatic MEX-file Generator (AMG) uses a small set of directives to generate sophisticated MEX-interfaces, e.g.:**

```
y=foo(x)
-kernel {
        y[0]=foo_impl(x[0]);
}
```

AMG output

```
y=foo(x)
-kernel {
        y[0]=foo_impl(x[0]);
}
```

3 lines of code

AMG

68 lines of code (comments partially stripped)

```
double foo_impl(double);
#define  x_par   arg[0]

void  mexFunction(int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[]) {

  MatObj** arg;

  arg = (MatObj **) mxMalloc( sizeof(MatObj *) * nrhs) ;

   { int i; for(i=0;i< nrhs;++i){
       arg[i] = (MatObj *) mxMalloc( sizeof(MatObj) );
       extract_all(prhs[i],arg[i]);
    }
  }

  double* x =  arg[0]->data;
  double* y;
  y = create_matrix_fast(&plhs[0],1,1,mxDOUBLE_CLASS,
                         mxREAL);

  y[0]=foo_impl(x[0]);

  /* Free temporary buffers. */
  { int i; for(i=0; i< nrhs; ++i)
     mxFree(arg[i]);
  }
  mxFree(arg);
}
```

Automatic Differentiation (AD)

- A computer program implements a mathematical function by concatenating basic operations like: +, -, *, /, sin(), tan(), ...
- AD applies the chain rule to the concatenation of the basic operations.
- AD-Tools to transform, e.g., Fortran, C or MATLAB exist.

y= f(x,c) → AD → [g_y,y]= g_f(g_x,x,c)

- More information on: http://www.autodiff.org

AMG supported differentiation

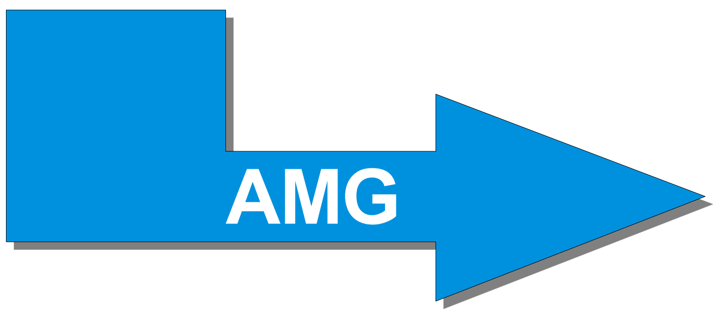
- Currently support for interfacing with two AD-tools is available:
  - ADIFOR (for Fortran77, default mode of AMG) and
  - ADiC (for C, directive `-adic` needed)
- AMG needs to know for which variables derivatives are sought (specified by the `-active()` directive).
- Conversion of data structure for Jacobian matrix:

$$\text{MATLAB} \longrightarrow \begin{cases} \text{ADiC} \\ \text{ADIFOR} \end{cases}$$

AMG-supported differentiation (2)

```
[g_y,y]=g_foo(g_x,x)
-adic
-active(x,y)
-kernel {
   ad_foo(grad(x)[0],
          grad(y)[0]);
}
```

6 lines of code

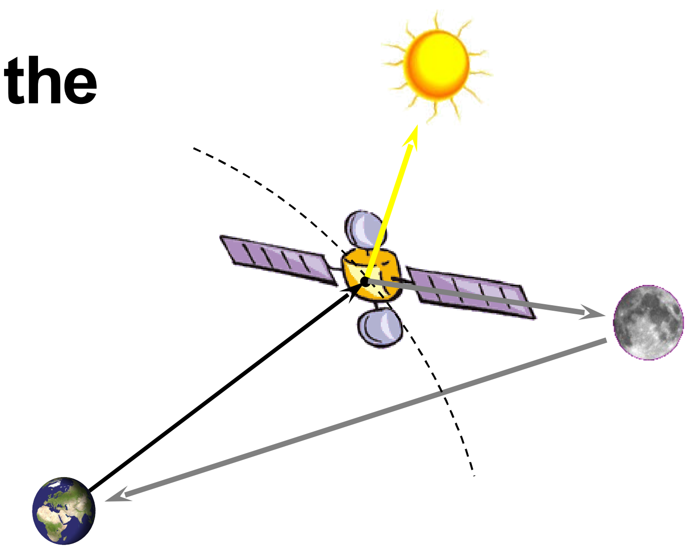


115 lines of code (comments partially stripped)

```
#define g_x_par  arg[0]
#define x_par  arg[1]

int amg_num_independents;
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]) {

  MatObj** arg;
  arg = (MatObj **) mxMalloc( sizeof(MatObj *) * nrhs) ;

  { int i; for(i=0;i< nrhs;++i){
      arg[i] = (MatObj *) mxMalloc( sizeof(MatObj) );
      extract_all(prhs[i],arg[i]);
    }
  }

  amg_num_independents = numel(g_x) / numel(x);
  double* g_x = arg[0]->data;
  double* x = arg[1]->data;
  int* amg_itmp;
  amg_itmp = (int *) malloc(sizeof(int) * 2);
  amg_itmp[0] = numel(x) * amg_num_independents;
  amg_itmp[1] = 1;
  if(!has_equal_size(g_x_par,amg_itmp,2))
    printf("??? Error: argument 1 is not of admissible size\n");

  double* g_y;
  g_y = create_matrix_fast(&plhs[0],1 * 1 * amg_num_independents,1,mxDOUBLE_CLASS,mxREAL);
  double* y;
  y = create_matrix_fast(&plhs[1],1,1,mxDOUBLE_CLASS,mxREAL);

  ad_AD_Init(amg_num_independents);

  DERIV_TYPE* grad(x) = deriv(x,g_x);
  DERIV_TYPE* grad(y) = (DERIV_TYPE *) mxMalloc( sizeof(DERIV_TYPE) * 1 * 1);

  ad_foo(grad(x)[0], grad(y)[0]);

  get_value(grad(y),1 * 1,y);
  get_grad(grad(y),1 * 1,amg_num_independents,g_y);

  ad_AD_Final();

  /* Free temporary buffers. */
  { int i; for(i=0; i< nrhs; ++i)
    mxFree(arg[i]);
  }
  mxFree(arg);
}
```

A real life example: MSIS-86

- **MSIS-86 is a atmospheric chemistry code written by NASA in Fortran77**
- **Computes the concentrations of certain species in the atmosphere**
- **Used in a model to compute the trajectory of a satellite implemented in MATLAB**



M. Kalkuhl, W. Wiechert
University of Siegen

MSIS-86 + MATLAB

- **Called from MATLAB:**
```
[D,T] = msis86(iday, ut, alt, xlat, xlong,
                  xlst, f107a, f107, ap, mass)
```
- **A Kalman filter is applied to the trajectory of the satellite.**
- **The Kalman filter needs the derivative of `msis86()`'s `D` with respect to four inputs.**
- **AMG is used to semi-automatically generate the interface to link the two differentiated codes.**

AMG directives: AD(msis86)

```
[g_D,D,g_T,T] = g_msis_f(iday, ut, g_alt, alt, g_xlat, xlat,
       g_xlong, xlong, xlst, f107a, f107, ap, mass)

-kernel {
   int tmass = (int) mass[0];
   int tiday = (int) iday[0];
   int g_p = numel(g_alt);

   g_msis_(&g_p, &tiday, ut, alt, g_alt, xlat, g_xlat,
       xlong, g_xlong, xlst, f107a, f107, ap, &tmass,
       D, g_D, T, g_T);
}
```

MEX-Interface (1)

```
/***************************************************************
 *  AMG version 0.0.4.1av, Copyright (C) 2005 Atya El-Shekh **
 *   patched by Andre Vehreschild 2006                      **
 *                                                          **
 *    AMG is distributed under GNU General Public License   **
 ***************************************************************/


#include <stdio.h>
#include <mex.h>
#include "amgmex.h"

/**
 * The input parameters. Given an input argument arg
 * The fields of the input arguments can be accessed as follows :
 * size(arg)    : an array of integers representing the dimensions
 * ndims(arg)   : the number of dimensions
 * nrows(arg)   : the number of rows
 * ncols(arg)   : the number of columns
 * numel(arg)   : the number of the elements
 * class(arg)   : the type of the matrix object
 *
 * for cell arrays, similar macros can be used to manipulate
 * the fields.
 * celldata(arg) : a pointer to the data field of the cell array arg.
 * similarly:
 *  celldims, cellndims, cellnrows, cellncols, cellnelem, cellcalss
 **/
```

MEX-Interface (2)

```
extern void g_msis_(int*, int*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*,
                    double*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*);

#define  iday_par   arg[0]
#define  ut_par   arg[1]
#define  g_alt_par   arg[2]
#define  alt_par   arg[3]
#define  g_xlat_par   arg[4]
#define  xlat_par   arg[5]
#define  g_xlong_par   arg[6]
#define  xlong_par   arg[7]
#define  xlst_par   arg[8]
#define  f107a_par   arg[9]
#define  f107_par   arg[10]
#define  ap_par   arg[11]
#define  mass_par   arg[12]

int* amg_tmp;
void  mexFunction(int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[]) {

          MatObj** arg;
          void** out;
```

MEX-Interface (3)

```
  arg = (MatObj **) mxMalloc( sizeof(MatObj *) * nrhs) ;

{ int i; for(i=0;i< nrhs;++i){
            arg[i] = (MatObj *) mxMalloc( sizeof(MatObj) );
            extract_all(prhs[i],arg[i]);
  }
}


  double*        iday =  arg[0]->data;
  double*          ut =  arg[1]->data;
  double*       g_alt =  arg[2]->data;
  double*         alt =  arg[3]->data;
  double*      g_xlat =  arg[4]->data;
  double*        xlat =  arg[5]->data;
  double*     g_xlong =  arg[6]->data;
  double*       xlong =  arg[7]->data;
  double*        xlst =  arg[8]->data;
  double*       f107a =  arg[9]->data;
  double*        f107 =  arg[10]->data;
  double*          ap =  arg[11]->data;
  double*        mass =  arg[12]->data;
  int* amg_itmp;

  double* g_D;
  g_D = create_matrix_fast(&plhs[0],numel(g_alt),8,mxDOUBLE_CLASS,mxREAL);
```

MEX-Interface (4)

```
        double* D;
        D = create_matrix_fast(&plhs[1],1,8,mxDOUBLE_CLASS,mxREAL);

        double* g_T;
        g_T = create_matrix_fast(&plhs[2],numel(g_alt),2,mxDOUBLE_CLASS,mxREAL);

        double* T;
        T = create_matrix_fast(&plhs[3],1,2,mxDOUBLE_CLASS,mxREAL);


   int tmass = (int) mass[0];
   int tiday = (int) iday[0];
   int g_p = numel(g_alt);

   g_msis_(&g_p, &tiday, ut, alt, g_alt, xlat,
                            g_xlat, xlong, g_xlong, xlst, f107a, f107, ap,
                            &tmass, D, g_D, T, g_T);

        /* Free temporary buffers. */
        {
                  int i; for(i=0; i< nrhs; ++i)
                  mxFree(arg[i]);
        }
        mxFree(arg);

}
```

Conclusion

- **Benefits of AMG:**
  - **simplifies generation of MEX-interfaces**
  - **tedious and error-prone job of manually writing a MEX-interface is eased by a simple but powerful macro language**
  - **support for linking of different AD-tools**

Thank you!

vehreschild@sc.rwth-aachen.de