USING LAPACK SOLVERS FOR STRUCTURED MATRICES WITHIN MATLAB

Radek Frízel, Martin Hromčík**, Zdeněk Hurák***, Michael Šebek****

*Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

**Centre for Applied Cybernetics, Czech Technical University in Prague,
Czech Republic

***Institute of Information Theory and Automation, Czech Academy of Sciences,
Czech Republic

Abstract

Linear polynomial equations, also called Diophantine in the systems and control literature, are often needed to design a dynamic feedback controller or compensator. Solving these equations numerically leads to special Toeplitz type sets of linear equations. To solve them, one can naturally call the general purpose solvers build in the standard MATLAB distribution. Another, and potentially more effective way, is to employ dedicated routines of the well known LAPACK library that exploit the structure of the problem to save computational time and memory requirements. This paper describes the experience of the authors with this approach.

Introduction

We compute the solution to a real system of the polynomial Diophantine equation $ax + by = c$ in Matlab, through the Sylvester matrix method using external solver. We construct band matrix A with sylvestr structure and compute the solution AX=C using extern solver, Lapack's functions. Matrix A is special band matrix identified with two columns, join of two Toeplitz matrices, B is matrix of right side and X is solution contains x and y from the polynomial Diophantine equation.

For using lapack's functons in Matlab we need MEX-file. This file can be written in the programming language C or Fortran. MEX-file is gate between Matlab and functions from Lapack library. In this gate arguments to be solved are passed from Matlab to and from Lapack's functions; they must be passed by reference. Calling the well known Lapack library (also called external solver) is well described for example in Matlab Help or in [2].

We use lapack's functions to solve a system of linear equations AX=B or A'X=B with a general band square matrix A using the LU factorization. Lapack library doesn't contain functions which solve general band rectangle matrix. For wolving these matrices we use QR factorization from Lapack library. But this factorization is for general matrices and that is not what we really wanted, exploit the structure of matrix.

Implementation

We create MEX-files in the programming language C under Windows. We use the lapack library, which is part of Matlab. This Lapack library isn't optimalized but for first steps it's enought. Later is better work under Linux and instal optimalization Lapack library.

How to create a MEX-file we show on example of calling lapack function which solve a system of linear equations AX=B or A'X=B with a general band square matrix A using the LU factorization. MEX-file contains one function mexFunction, which is interface between Matlab and lapack functions. This mex function has four parameters: number of input parameters, array of pointers to input parameters, numbers of output parameters and array of pointers to output parameters. To retrieve the values of input parameters and store them into local variables mexFunction uses appropriate Matlab fuctions. When it's done we call the lapack function. The pointers to output are stored to array of the mexFuction and the local arrays are destroyed.

Code of MEX file GenBand:

```
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
double *AB, *B, *IPIV, *temp, *temp1, *temp2, *temp3, zero=0.0;
int LDAB, KL ,KU, M, N, Info, k, LDB, NRHS;
char msg[100], *TRANS="N";

// Check the imput parameters:
if (nrhs != 4)
  {
      mexErrMsgTxt("Four input arguments required.");
  } else if (nlhs > 2)
  {
      mexErrMsgTxt("Too many output arguments.");
  };

// Retrieve the values of input parameters:
M = mxGetM(prhs[0]);
N = mxGetN(prhs[0]);
AB = mxCalloc (M*N, sizeof(double));
temp = mxGetPr(prhs[0]);
for (k=0;k<N*M;k++)
    {
    AB[k] = temp[k];
    };
LDAB = M;

if (!mxIsDouble(prhs[2]) || mxIsComplex(prhs[2]) ||
      mxGetN(prhs[2])*mxGetM(prhs[2]) != 1) {
    mexErrMsgTxt("Input KL must be a scalar.");
  }
  KL = mxGetScalar(prhs[2]);

if (!mxIsDouble(prhs[3]) || mxIsComplex(prhs[3]) ||
      mxGetN(prhs[3])*mxGetM(prhs[3]) != 1) {
    mexErrMsgTxt("Input KU must be a scalar.");
  }
  KU = mxGetScalar(prhs[3]);

IPIV = mxCalloc (N,sizeof(double));

// Call the first lapack fuction:
dgbtrf(&M,&N,&KL,&KU,AB,&LDAB,IPIV,&Info);
```

```
// Check errors:
if (Info < zero)
  {
      sprintf(msg,"Input %d to DGBTRF had an illegal value",-Info);
      mexErrMsgTxt(msg);
  };

// Retrieve the values of input parameters to second lapack functin:
 LDB = mxGetM(prhs[1]);
 NRHS = mxGetN(prhs[1]);
 B = mxCalloc(LDB*NRHS, sizeof(double));
 temp2 = mxGetPr(prhs[1]);
 for (k = 0; k < LDB*NRHS; k++) {
    B[k] = temp2[k];
    }
 Info = 0;

// Call the second lapack fuction:
dgbtrs(TRANS, &N, &KL, &KU, &NRHS, AB, &LDAB, IPIV, B, &LDB, &Info);

// Check errors:
  if (Info < zero) {
       sprintf(msg,"Input %d to DGBTRS had an illegal value",-Info);
       mexErrMsgTxt(msg);}

// Create matrix for the return arguments:
  plhs[0] = mxCreateDoubleMatrix(M,N,mxREAL);
  temp1 = mxCalloc(M*N, sizeof(double));
  for (k = 0; k < M*N; k++)
  {
     temp1[k] = AB[k];
  }
  mxSetPr(plhs[0],temp1);

  plhs[1] = mxCreateDoubleMatrix(LDB,NRHS,mxREAL);
  temp3 = mxCalloc(LDB*NRHS, sizeof(double));
  for (k = 0; k < LDB*NRHS; k++) {
     temp3[k] = B[k];
    }
  mxSetPr(plhs[1],temp3);

// Free arrays:
  mxFree(AB);
  mxFree(B);
  mxFree(IPIV);
 return;
}
```

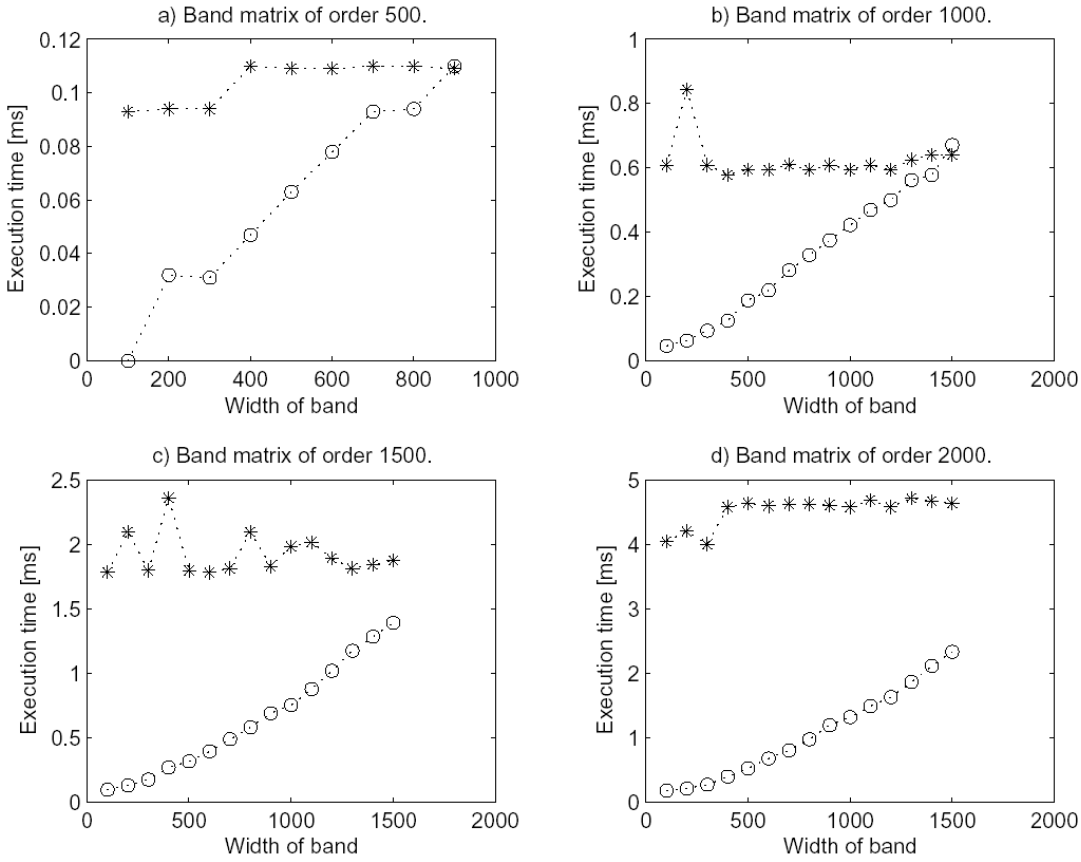
To compile and link the MEX file at the Matlab prompt, type:
`mex GenBand.c`.

Matlab produces file GenBand.dll, which can be called in the same manner as a standard Matlab m-file.

There are some possible problems you could meet if you want to call lapack functions from your mex file. The variables which are used as input parameters to lapack functions must be of the correct types (double, int etc.). All local arrays must be destroy (free) at the end.

Experimental testing

We compare standard function of Matlab, left matrix division, and external solver, Lapack function, dgbtrs, in solving of linear equations AX=B, A is general square band matrix. Measured values are shown in the graph 1.

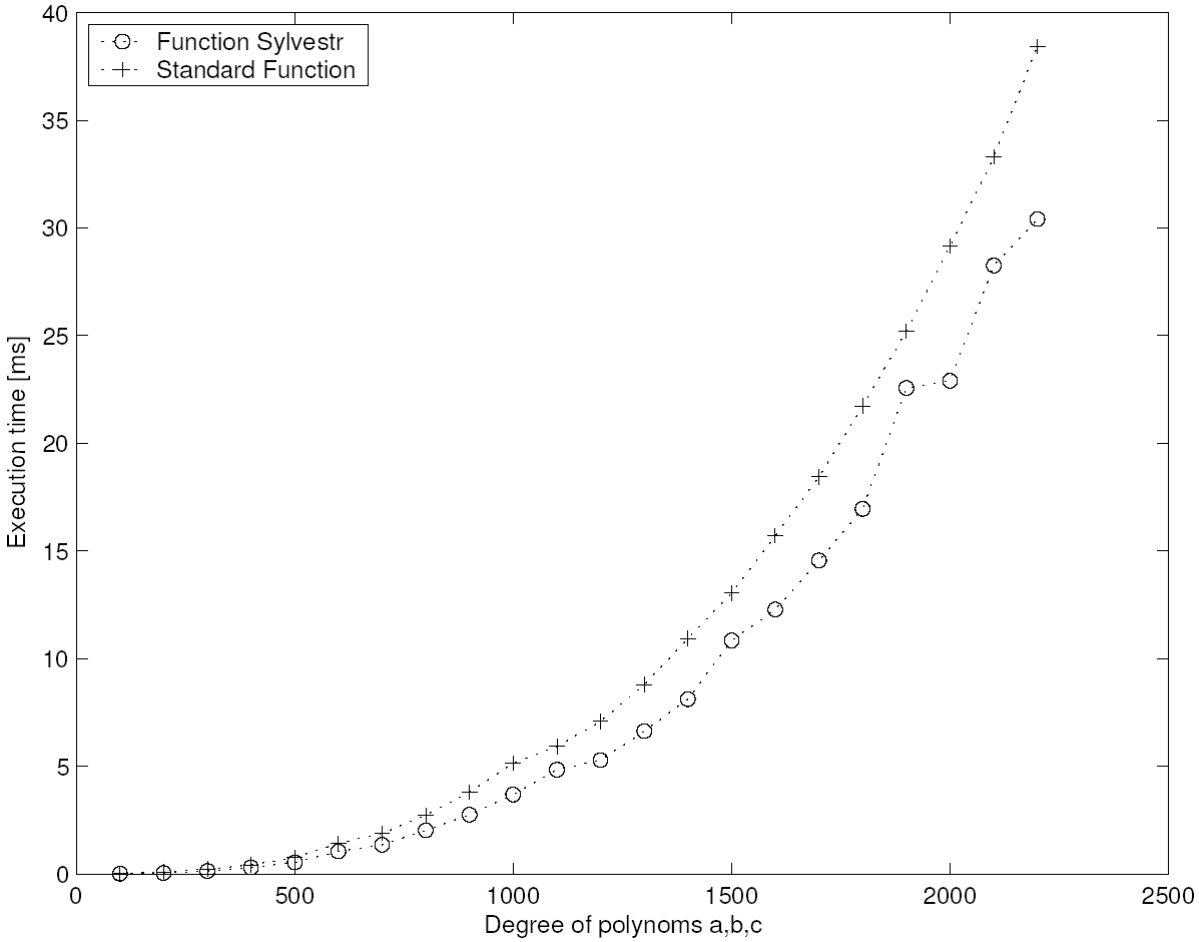


Graph 1: How depends execute time of both method on band width. ("*" representability Matlab's function and "o" representability external solver.) Order of matrix is: a) 500, b) 1000, c) 1500, d) 2000.

We compute the solution to a real system of the scalar polynomial Diophantine equation $ax + by = c$ in Matlab, through the Sylvester matrix method using external solver. Measured values are shown in the graph 2.

Conclusion

The external solver is faster than general purpose solver build in standard MATLAB distribution. We make sure about that with experimental testing. Now we need to solve AX=B, where A is general band rectangle matrix. We trying use another function from lapack library in which we can utilize known structure of matrix. For example Lapack function with decomposition of rectangle band matrix to bidiagonal square matrix multiplying with two orthogonal square matrices, A=Q*Ab*P'.



Graph 2: How depends execute time of both method on degree of polynoms. ("+" representability Matlab's function and "o" Lapack library function.)

Acknowledgment

The work of R.Frízel has been supported by the Ministry of Education of the Czech Republic under contract No. INGO LA156. The work of M. Hromčík has been supported by the Ministry of Education of the Czech Republic under contract No. LN00B096. The work of Z.Hurák and M.Šebek has been supported by the Ministry of Education of the Czech Republic under contract KONTAKT No. ME496.

Reference

[1] LAPACK – Linear Algebra PACKage [online]. http://www.netlib.org/lapack [cite 2004-04-29]

[2] Using LAPACK and BLAS Functions [online]. http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/matlab_external.html [cite 2004-04-29]

[3] HAVLENA, V. (1999). *Moderní teorie řízení - Doplňkové skriptum*. Vydavatelství ČVUT, Praha.