A linear system solver for Mathematica that uses the SACLIB

Version 1.0[1]

Erhard Aichinger

November 7, 1996

Abstract

We describe two functions that replace Mathematica's function `NullSpace`. Both functions take as input a matrix A and compute a basis of the vector-space of all solutions of the linear system $A \cdot x = 0$.

One of them, `ENullSpace`, is a plain Mathematica program, whereas the other function, `CNullSpace`, uses MathLink to call a C-program that computes the nullspace of a matrix using the SACLIB.

Especially if the matrix contains parameters, the presented algorithms outdo Mathematica's in-built function `NullSpace` in efficiency.

Contents

Chapter 1

Specification of the implemented functions

1.1 Specification of `ENullSpace`

`ENullSpace` works as the mathematica function `NullSpace`.
Restrictions on the input: No optional arguments, such as `Modulus -> 5`, can be given.

As an example, we compute the nullspace of the 1×2-matrix $(a + 2b, c)$.

```
In[17]:= ENullSpace [{{a+2b, c}}]

                -a - 2 b
Out[17]= {{1, --------}}
                   c

In[18]:= NullSpace [{{a+2b, c}}]

                  c
Out[18]= {{-(-------), 1}}
              a + 2 b
```

1.2 Specification of `CNullSpace`

`CNullSpace` computes the the nullspace of the matrix A of (possibly multivariate) *polynomial expressions*[1] over the rational numbers.

A polynomial expression is any term built up from rational numbers and variables using the operations `+`, `-`, `*` and $\hat{}n$, where n is a nonnegative integer.

Let D be the ring of all polynomials with rational coefficients in the variables V, where V is the set of all variables occuring in the input matrix A. Then these polynomial expressions are interpreted as elements of D.

[1] The implementation can easily be modified that it takes as an input also a matrix of rational functions instead of polynomials.

The output of `CNullSpace` is a basis of the vector space of the solutions of the linear system $A \cdot x = 0$ over the field $Q(D)$, where $Q(D)$ is the quotient field of D.

```
In[24]:= CNullSpace [{{(a+b)*a^3, b}}]

                  4     3
                -a   - a  b
Out[24]= {{1, ----------}}
                    b

In[25]:= CNullSpace [{{-3, 2}, {-6, 4}}]

           2
Out[25]= {{-, 1}}
           3
```

Chapter 2

Motivation

The Mathematica function `NullSpace` has two main disadvantages when it is applied to a matrix of polynomial expressions:

1. The output of `NullSpace` is hardly usable unless it gets simplified. We illustrate this by the following example.

```
In[42]:= NullSpace [{{b,a,a},{a,b,a}}]

                            2               2
                a     a (-a   + a b)      -a   + a b
Out[42]= {{-(-) + -------------, -(---------), 1}}
                b           2    2           2    2
                      b (-a   + b )       -a   + b

In[43]:= ENullSpace [{{b,a,a},{a,b,a}}]

                  a           a
Out[43]= {{-(-----), -(-----), 1}}
                a + b       a + b

In[44]:= CNullSpace[{{b,a,a},{a,b,a}}]

                a        a
Out[44]= {{------, ------, 1}}
              -a - b   -a - b
```

2. `NullSpace` is slow. In the following table, the computation time of the three algorithms for some matrices is printed. The matrix M_n can be found in the file `TestMatrices/matrix`n`.m`, the matrix A_3 in the file `TestMatrices/apery3.m`.

   Computation times are taken on the DEC-station `jade.risc.uni-linz.ac.at` using Mathematica Version DEC RISC 2.2.

   In the columns, you find the following CPU-times in seconds:

   C CPU-time used for computing the nullspace with `CNullSpace`. In this case, both the time used in Mathematica (for writing the input to the link and reading the output from the link) and the time used in the C-program are added.

E CPU-time used for computing the nullspace with `ENullSpace`.

N CPU-time used for computing the nullspace with `NullSpace`. Since `NullSpace` does not give a simplified, but a hardly usable form of the result, the "real" imformation is contained in the column labelled **SN**.

SN CPU-time used for computing the nullspace with `NullSpace` and simplifying the result using `Simplify`.

Matrix	**C**	**E**	**N**	**SN**
M_0	8	31	28	30
M_1	7	84	147	254
M_2	3	58	6	56
M_3	46	6882	> 60000	> 60000
A_3	129	206	> 12000	> 12000

It is our general experience that Mathematica's in-built function `NullSpace` is only good for small examples or for matrices that contain no variables.

Chapter 3

How to use `CNullSpace`

The program is installed on the DEC-stations at RISC. This means that you can use C-NullSpace from a Mathematica session at any environment, but the C-program will run at one of the DEC-stations. The current location of the installation is

`~comb/mathematica/CNullspace.`

All pathnames refer to this directory.

If you want to use this program, copy the file `Sac/sacLink.m` into the directory from which you start Mathematica. You can then load `CNullSpace` into Mathematica performing the following steps.

1. Start your Mathematica session.
2. Load the file `sacLink.m` into Mathematica by `<< sacLink.m`.
3. Type `Setup`.

Usually, a message appears that informs you about successful installation. However, the following complications may occur:

- You get the error message `Illegal instruction (core dumped)`. This means that there is currently not enough memory at the host where you wanted to run `CNullSpace`. By default, this C-host is `lion.risc.uni-linz.ac.at`, but it can be any of the DEC-stations at RISC. So, try:

  ```
  In[3]:= CHost = "tiger.risc.uni-linz.ac.at"
  In[4]:= Setup
  ```

- There may be problems with the port on which the connection was to be set up. Try `In[5]:= CurrentPort = 5609`, `In[6]:= Setup`.

`CurrentPort` will usually take a value between 5000 and 9000. If you just want to use `CNullSpace`, you now know all you need to know. The rest is for those that want to know how it all works.

Chapter 4

How to use SACLIB-programs from Mathematica – a concrete example

4.1 Motivation

Using SACLIB-programs ([C+93]) from Mathematica combines the efficiency of C and the SACLIB-algorithms with the user friendly interface of Mathematica. This is made possible by MathLink ([Wol93]) and is surprisingly easy to do.

4.2 A concrete example

In this section, we show a concrete example:we compute the product of two integer polynomials using the SACLIB. This example can be found in the directory `Example`.

For those who want to write their own SACLIB functions that can be called from Mathematica, we suggest that they **just edit the sources of this example**.

4.3 The SACLIB/C-part

First of all, we need to prepare a SACLIB-function that performs the required operation. For making this easier, we have written a library of utitilities for the SACLIB that performs two tasks:

- It provides a data-type "rational function" for the SACLIB.
- Since the SACLIB can only read from files and the keyboard, new input functions for the SACLIB were written that read expressions from strings. For this purpose, the abstract datatype `CString` (read: cursor-string) was implemented.

The specification of this addition to the SACLIB can be found in the file

`Sac/SacUtils/CS_RF_SAC.h.`

It is described in the appendix. The library can be found in `Sac/SacUtils/sacutils.a`.

We first need a C-file that multiplies two polynomials. The function `PProduct` takes as an input a string containg a list of the two polynomials in Mathematica-InputForm, computes their product, and writes this output to the link between C and Mathematica. Note that we suggest that the input and output of all C-functions called via MathLink be represented as Strings. Actually, there is no need for other data-types. This file can be found in `Example/product.c` and we report it here.

---

product.c

```
/* file: product.c  */

#include <saclib.h>         /* SACLIB */
#include <stdio.h>
#include <stdlib.h>
#include <mathlink.h>       /* MathLink */
#include <CS_RF_SAC.h>      /* utilities for using the SACLIB */



                            /* prototype */
 void PPRODUCT (char *polylist);
/**************************************************************
   PRODUCT takes a list of two polynomials {p_1, p_2}
   and returns their product.
 **************************************************************/



                            /* implementation */
 void PPRODUCT (char *polylist)
/**************************************************************
   PPRODUCT takes a list of two polynomials {p_1, p_2}
   and returns their product.
 **************************************************************/
{
   CString cs, cs1;              /* input string */
   CString cs_result;       /* output string */
   Word V_list,             /* the list of variable names */
        P_list,             /* the list of polynomials */
        factor1,            /* the two polynomials to be multiplied */
        factor2,
        product,            /* the result */
        ok;                 /* ok == 1 iff no error */



                            /* the function is used via Mathlink */
   Set_link_mode ('l');
```

```
                            /* write input string to a CString */
   cs_new (&cs);
   string_to_cs (polylist, &cs);

                            /* parse string */
   CS_RFL_READ (&V_list, &P_list, &ok, &cs);

                            /* check input */
   if ( (! ISLIST (P_list)) ||
        LENGTH (P_list) != 2 )
   {
       Fatal_error ("incorrect input\n");
   }

                            /* compute the product */
   factor1 = LELTI (P_list, 1);
   factor2 = LELTI (P_list, 2);
   product = RF_PROD ( LENGTH (V_list),      /* number of variables */
                       factor1,
                       factor2 );

                            /* initialize CString that contains the result */

   cs_new (&cs_result);
                            /* write product to cs_result */

   CS_RF_WRITE (V_list, product, &ok, &cs_result);

                           /* put result back to the link */
   MLPutFunction (stdlink, "List", 2);
   MLPutString   (stdlink, cs_result.str);

                           /* deallocate memory */
   cs_del (&cs_result);
   cs_del (&cs);

                          /* put last part of the result
                             to the link */
   MLPutInteger (stdlink, 0);

   return;

}



int main(int argc, char *argv[])
{
   Word stack;
   int  a;
                      /* memory allocated for saclib */
   NU = 8000000;
   fprintf(stderr, "NU=%d\n", NU);

                      /* start SACLIB */
   BEGINSACLIB (&stack);
   GCM = 1;
   fprintf(stderr, "Started c-program!\n");
```

```
                    /* wait for input from Mathematica */
   a = MLMain(argc, argv);

                    /* end SACLIB */
   ENDSACLIB (SAC_FREEMEM);
   fprintf(stderr, "Ended c-program!\n");
   return a;
}
```

4.4 The link between Mathematica and C

The function `PProduct` shall be called from Mathematica. For this purpose, we provide a file that gives more information of the function to be called. This is the file `Example/productLinker.tm`. We give its source here.

productLinker.tm

```
:Begin:
:Function: PPRODUCT
:Pattern:  CProduct [ coeffMatrix_String ]
:Arguments: { coeffMatrix }
:ArgumentTypes: { String }
:ReturnType: Manual
:End:
```

This file is then compiled to a C-Program by the program `mprep` that comes with MathLink. All these compilations are done by the makefile `Example/Makefile`. We also give here the source of the makefile:

Makefile

```
# file: Makefile
#  define the directories.

SACDIR = /usr/local/saclib/saclib2.1
                                               # location of SACLIB
MATHDIR = /usr/local/packages/mathematica
                                               # location of Mathematica
MATHLIB = $(MATHDIR)/Bin/MathLink
                                               # location of Mathlink
MPREP = $(MATHLIB)/mprep
                                               # location of mprep
MATHINC = $(MATHDIR)/Source/Includes
                                               # location of mathlink.h
```

```
SACUTILS = /home/eaiching/SacMath/Sac/SacUtils
                                              # location of CS_RF_SAC.h


INCLUDES = -I$(SACUTILS) -I$(SACDIR)/include -I$(MATHINC)
LIBS  =  $(SACUTILS)/sacutils.a $(SACDIR)/lib/saclibo.a $(MATHLIB)/libML.a

CFLAGS = -g
CC = gcc

#  define the objects (test and source objects)

SRCS = productLinker.c product.c
OBJS = $(SRCS:.c=.o)

product: $(OBJS) $(LIBS)
 $(CC) $(CFLAGS) $(OBJS) $(LIBS) -lm -o $@

.c.o:
$(CC) -c $(CFLAGS) $(INCLUDES) $*.c  -o $@

productLinker.c: productLinker.tm
$(MPREP) productLinker.tm -o $@

clean:
rm -f productLinker.c *.o product

depend: productLinker.c
makedepend -- $(CFLAGS) $(INCLUDES) -- $(SRCS)
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

4.5 The Mathematica-part

Now we need a Mathematica-program that calls the C-Program, lets it run in the background and links it to the Mathematica-session. This is done in the following file `Example/product.m`. We report this file here:

---

product.m

```
(************************************************************
     file: product.m
     File loaded into mathematica for establishing
     the connection to  "product".
************************************************************)


                             (* global variables *)

      (* Directory in which the executable
                                 'nullspace' is.*)
CDir = "/home/eaiching/SacMath/Example";
```

```
      (* host, where the C - program is running *)
CHost = "lion.risc.uni-linz.ac.at";

                              (* if WARNINGS == 1 then warnings are printed *)
WARNINGS = 1;



             (* internal global variables *)

CProduct::Error = "`1` "
On [CProduct::Error]


WaitingTime = 10;

CurrentPort = 7578;

ProvokeExitOfC = "exit"


SetUpC [n_] :=
                         (* creates C - process in the background *)
    Module [{cmd},
            cmd = "rsh " <>
                  CHost <> " " <> CDir <>
                  "/product -linkmode Listen -linkname " <>
                  ToString [n] <> "&";
            Run [cmd]
    ]



SetUpM [n_] :=
                        (* links Mathematica - process to running C-process *)
   Module[{},
          CurrentLink = Install [ToString [n],LinkMode -> Connect,
                                             LinkHost -> CHost];
          CurrentPort = n;
         ]


SetUp [n_] :=
       Module[{},
       SetUpC [n];
       Pause  [WaitingTime];
       SetUpM [n];
       If [CurrentLink === $Failed,
            WarningMessage ["connection could not be established"];
                         (* try to kill C - process *)
            CProduct [ProvokeExitOfC],
                                           (* else *)
          Print ["--- Usage : CProduct [{p1, p2}] ---"]
       ];
       ]
```

```
ReSetUp [n_] := Module[{},
                        (* it is supposed that the C - program has exited *)
                  Uninstall[CurrentLink];
                  Pause [1];
                  SetUpC[n];
                  Pause [WaitingTime];
                  SetUpM [n];
                  If [CurrentLink == $Failed || CENullSpace [{{1}}] =!= {},
                    WarningMessage ["connection could not be re-established"]
                     ];
                 ]


UnSetup :=
      Module[{},
             CProduct[ProvokeExitOfC];
             Uninstall [CurrentLink];
             CurrentPort = CurrentPort + 1
            ]


Setup := SetUp [CurrentPort]


CProduct [{p1_, p2_}] := CProduct [ ToString [ InputForm [{p1,p2}] ] ];
CProduct [p1_, p2_] := CProduct [{p1, p2}]
```

Now try the following Mathematica-session.

```
In[1]:= << product.m

In[2]:= Setup
```

If you get error messages, then there is not enough memory on the host where the C-program is to run. Try to change this host by

```
In[3]:= CHost = "tiger.risc.uni-linz.ac.at"

In[4]:= Setup
```

If you get the message `--- Usage :  CProduct [p1, p2] ---`, then the connection is properly set up. Try

```
In[5]:= CProduct[a^2, a^2+b]

Out[5]= {(a^2 b + a^4)/(1), 0}
```

The first part of the output corresponds to the result, the second part is an error-code.

Chapter 5

A nullspace-algorithm using the SACLIB

The NullSpace algorithm we have implemented is just Gauß's elimination procedure.

Arithmetic is done in the field of rational functions over the rationals. An element in this field is represented by a pair of polynomials with integral coefficients. The result of every arithmetical operation is immediately shortened.

The algorithm is just Gauß's elimination procedure. The pivot element is chosen as the "smallest" possible nonzero element in the matrix. The measure for the size of a rational function is the amount of memory it needs. When the pivot element is found, lines and columns of the matrix are exchanged such that the pivot element appears in the main diagonal.

The Mathematica-Version of the same algorithm can be found in `SacMath/ENullSpace.m`. Its C-implementation is in `Sac/CNullSpace/erhard.c`.

Chapter 6

Acknowledgments

This project was carried out as “Praktikum Programmierprojekt” under the direction of Peter Paule. I want to thank him and the Combinatorics group for the pleasant cooperation. Special thanks go to Markus Schorn for his advice.

Appendix A

New input/output functions for the SACLIB

A.1 A datatype replacing files: strings with cursor

The existing I/O-functions of the SACLIB read their input from a file and write it to a file. When calling a SACLIB-function from a C-Program, it is however desirable to write the output into a string rather than on a file.

For this purpose, we have implemented the abstract datatype of **strings with cursor** into C. These *strings with cursor* imitate the behaviour of *files*; however, instead of writig data to harddisks or other physical devices, the contents of the string is storted in the machines main memory.

In C, the strings with cursor are realized as the abstract datatype `CString`. They allow the following functions. A more concise description can be found in the file

`Sac/SacUtils/CS_RF_SAC.h` .

A.1.1 Constructors

```
int cs_new (CString *s);
   /* Creates the cursor string *s and allocates
      memory for it. */

void cs_del (CString *s);
   /* frees the memory allocated for * s */

void string_to_cs (char    *string, CString *csp);
   /* transforms the string a into the cursor string
      *csp. *csp must not be initialized.
      Destructive, i.e. string cannot be used
      afterwards.
      The cursor is set at the first element. */
```

A.1.2 Selectors

```
char cs_getc (CString *s);
   /* gets the next character of *s.
      Returns EOF at the end of the string. Moves the cursor. */

char cs_nextchar (CString s);
   /* Returns the first non-whitespace from
      the cursor position. Does not move the cursor. */


void cs_reset_cursor (CString *s);
   /* sets the cursor to the beginning of * s */

void cs_backspace (CString *csp);
   /* sets the cursor one place back,
      If cursor points to the first char
      of *csp, *csp remains unchanged. */
```

A.1.3 Writing to a `CString`

```
char cs_putc (char c, CString *s);
   /* Returns: the character that was put into the string,
      EOF if cs_put was not successful. Moves the cursor. */
```

A.2 Writing SACLIB-expressions to strings with cursor

`CS_AWRITE ( A, *csp )`

Cursor string Atom write. *Writes the atom* `A` *to the cursor string* `*csp`.

This is just the cursor-string version of the SACLIB-operation `AWRITE`. In the same way, most of the the SACLIB-output functions have a cursor-string version, which work precisely as their versions writing to the standard output. A pointer to the cursor string is always given additionally as the last parameter.

These functions are:

`CS_BKSP, CS_CREAD, CS_CREADB, CS_CWRITE, CS_IPDWRITE, CS_IPEXPREAD, CS_IPFACTREAD, CS_IPPOWREAD, CS_IPTERMREAD, CS_IREAD, CS_IUPWRITE, CS_IWRITE, CS_LWRITE, CS_VREAD, CS_VWRITE.`

Appendix B

A datatype for rational functions in the SACLIB

B.1 Introduction

B.1.1 Purpose

The `sacutils`-package provides operations for multivariate rational functions over the rationals. The following description assumes that you are familiar with [C+93].

B.1.2 Definition of terms

internal representation A rational function $\frac{p}{q}$ is represented as the SACLIB-list $(1, p, q)$, where p and q are polynomials **over the integers** in *sparse recursive representation.*

external representation The external representation of p is the string $(\hat{p})|(\hat{q})$, where $\hat{p}$ and $\hat{q}$ are string representations of p and q.

B.2 Operations for rational functions

B.2.1 Constructors

```
R <- RF_0 ( r )
```

Rational function Zero. *Builds a rational function in r variables equal to* $0/1$.

```
R <- RF_1 ( r )
```

Rational function One. *Builds a rational function in r variables equal to* $1/1$.

```
R <- IPS_TO_RF ( P1, P2 )
```

Integral polynomials to rational function. *Builds the rational function* `P1/P2`. *Both polynomials must be polynomials in the same number of variables* `r`.

B.2.2 Predicates

```
R <- IS_RF_VALID ( r, F )
```

Is valid rational function. *Returns true iff* `F` *is a rational function with nonzero denominator. Does not check whether numerator and denominator are polynomials in the right number of variables.*

```
R <- IS_RF_ZERO ( r, F )
```

Is rational function zero. *Returns true iff* `F` *has numerator zero.*

B.2.3 Operations

```
RF_SHORTEN ( r, *F )
```

Rational function Shorten. *Shortens* `F`.

```
R <- RF_SUM ( r, A, B )
```

Rational function Sum. *Computes* `A` + `B`. *Shortens the result unless* `A` *and* `B` *have relatovely prime denominators.*

```
R <- RF_DIF ( r, A, B )
```

Rational function Difference. *Computes* `A - B`. *Shortens the result unless* `A` *and* `B` *have relatively prime denominators.*

```
R <- RF_NEG ( r, A )
```

Rational function Difference. *Computes* `-A`. *Does not shorten.*

```
R <- RF_PROD ( r, A, B )
```

Rational function Product. *Computes* `A` * `B`. *Returns a shortended result provided that* `A` *and* `B` *are shortended.*

```
R <- RF_DIV ( r, A, B )
```

Rational function Product. *Computes* `A` / `B`. *Returns a shortended result provided that* `A` *and* `B` *are shortended. Complains and stops if* `B` *is zero.*

B.2.4 Input/Output functions for rational functions

```
CS_RFL_WRITE (Word V, Word A, Word * ok, CString * csp);
/******************************************************************
   Outputs a list of rational functions.

   Input:
      V: list of variables that is used for writing functions.
```

```
      A: function to be written on csp.
   Output:
      *ok  1 iff no error
      *csp Cursorstring to which A is written.
******************************************************************/


void CS_RFL_READ (Word * V, Word * A, Word * ok, CString * csp);
/******************************************************************
   Reads a list of rational functions from the cursor string *csp.
   Reading starts at the beginning of the cursor string *csp.
   On leaving CS_RFL_READ the cursor is on the first position in *csp
   that has not been read.
   Output:
      V :  List of the variables,
      A :  SACLIB-representation of the list,
      ok:  1 iff no error,
      csp :CString from which the list is read.
******************************************************************/


void CS_RF_READ (Word r, Word V_list, Word * RF, Word * ok, CString * cs);
/*********************************************************************
   CS_RF_READ
   reads a rational function in the variables
   in V_list  from the CString *cs.

   The rational function can be given in the following form:
      ratFu ::=
 poly | poly "/" poly,
   where poly is a polynomial expression in the variables
   in V_list.
   Parameters :
     Input :
V_list : list of variables
     Output:
* RF   : rational function in internal canonical form.
* ok   : 1 iff a rational function could be read.
* cs   : String from which the rat.fu. is read.
 The cursor is set to the first character that does
 not belong to the rat.fu.
       Definition :
     The internal canonical form of a rational function
     is the SACLIB - List (1, num, denum), where
     num and denum are polynomials in SACLIB - internal
     canonical form.
*********************************************************************/


void CS_RF_WRITE (Word V,
  Word A,
  Word * ok,
  CString * cs);
/**********************************************************
```

```
   writes the rational function A to the cursorstring
   cs.
   Input:
      V : list of the variables
      A : rational function to be printed
   Output:
      *ok : 1 iff no error occured
      *cs : string to which rational function is written
**********************************************************/
```

Appendix C

Location of the installation

The current location of the installation at RISC is

`~comb/mathematica/CNullspace.`

All pathnames refer to this directory.

Bibliography

[C+93] G.E. Collins et al. Saclib 1.1 user's guide. Technical Report 93-19, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, March 1993.

[Wol93] Wolfram Research. *MathLink Reference Guide, Version 2.2*, 1993.

Author's address
Institut für Mathematik
Universität Linz
Tel. 2468 / 9154
email: `Erhard.Aichinger@risc.uni-linz.ac.at`