This worksheet shows, how to use Pari from Maple for numerical linear algebra.
For details please check the Pari manuals, at least Chap 3.8 in users.pdf.

There are certainly better libraries for numerical linear algebra, it is for completeness
and my main interest in numerics using Pari is around integration - as an additional
tool, not as a replacement (it is not too handy ...).

AVt, Nov 2006

```
> restart; kernelopts(version);
```

Maple 10.06, IBM INTEL NT, Oct 2 2006 Build ID 255401

Settings to use Pari

```
> currentdir(): theDLL:=cat(%,`\\pari_mpl.dll`);
```

This stores the location of the DLL in a string, here it is supposed exist in the directory of this worksheet.
One can use an explicite path otherwise (care for inputting because of back slashes)

theDLL := "C:_Work\DevCpp\pari_240_mpl\release\pari_mpl.dll"

```
> pari := module()
    local ModuleLoad, ModuleUnload;
    global theDLL;
    export ModuleApply;
    ModuleLoad:=   define_external(`start_Pari`,MAPLE,LIB=theDLL);
    ModuleUnload:= define_external(`stop_Pari`, MAPLE,LIB=theDLL);
    ModuleApply:=  define_external(`eval_Pari`, MAPLE,LIB=theDLL);
  end module;
```

The reason for accessing the DLL through a module with the above syntax is, that Pari is to be initialized and closed, the above should guarantee it, even if Maple is exited, please check Maple's help for 'ModuleUnload'.

For doing this explicitely within a session one defines:

```
start_pari:=    define_external(`start_Pari`,MAPLE,LIB=theDLL):
stop_pari:=     define_external(`stop_Pari`,MAPLE,LIB=theDLL):
restart_pari := define_external(`restart_Pari`, MAPLE,LIB=theDLL):
```

The last command is convenient as I do not know of an easy way to clean up variables or do a save garbage collection.

Communication between Maple and Pari is through strings (even if this means different things in C and Maple).

If an error message is given concerning the string size for input ot output, then use the following command (`buffersize_pari(100000)` allows up to 100000 characters):

```
buffersize_pari := define_external(`buffersize_Pari`, MAPLE,LIB=theDLL):
```

If an error from Pari is shown complaining the stack size, then `pari("allocatemem(x=0)");` can
be used to double the stack size.

In case of 'numerical' results from Pari one has to modify the answers to numerics in Maple, the following is a short-hand for parsing Pari's result (and easier than coding a 'string replace' function in C):

```
pp:=proc(str)                 # parse pari result
   StringTools:-SubstituteAll(str," E","E");   # remove white space
   StringTools:-SubstituteAll(%,".E",".0E"); # to cover 0
   parse(%);                  # convert string to numeric value
end proc:
```

In case one knows the answer is numeric one use the following combined command:

```
value_pari:= pp@pari:
```

For rational numbers Pari returns the fraction, either use that in Maple or send it back to Pari to evaluate the expression to a floating point number:

```
evalf_pari := proc(str)
  value_pari(cat("1.0 * ", str)); # value_pari(cat(str, " + 0."));
end proc:
```

As a variant one could have used `value_pari(cat(str, " + 0."))` as well.

In case of any error by executing Pari its kernel is stopped, an error message is printed.

```
pari := module() local ModuleLoad, ModuleUnload; export ModuleApply; global theDLL;   end module
```

>

>

First Steps

Try some basic commands and try to find out the formatings used

```
> restart_pari():

  Digits:=16;
  cat("default(realprecision,",  convert(Digits,string), ")"); pari(%):
  ``;
  theDim:=2;
  cat("theDim =",  convert(%,string)); pari(%):
PARI 2.3.0 restarted and initialized.
                                        Digits := 16
                                    "default(realprecision,16)"


                                        theDim := 2
                                      "theDim =2"
> pari("X = mathilbert(theDim)");
"
    [1 1/2]

    [1/2 1/3]
    "
> pari("X[2,]");                # 2nd column
                                    "[1/2, 1/3]"
> pari("Y = 1/X");              # Y = inverse matrix for X
"
    [4 -6]

    [-6 12]
    "
> pari("Y*X");                  # multiplication
"
    [1 0]
```

```
[0 1]
"
```

So it looks quite similar to Maple.

And for floating point numbers?

```
> L:=[
  "X = 1.0 * X",                # let it have floating point entries, 16
  decimals
  "default(realprecision,60)",  # increase precision
  "Y = 1/X"                     # Y = inverse matrix for X
  ]:
  map(pari,L):                  # execute that commands
  ``;
  pari("Y");
  ``;
  value_pari("Y[1,1]"); length(%);
  ``;
  pari("precision(0.0)");


"
   [3.999999999999999999 -5.999999999999999999]

   [-5.999999999999999999 11.999999999999999999]
   "

                                3.999999999999999999
                                          24

                                        "67"
```

So Pari first works with 24 digits internally, has a rounding error and does not compute with 60 Digits here,
but uses 67 internally. Maple behaves different:

```
> LinearAlgebra[HilbertMatrix](2); X:=evalf(%): Digits:=60:
  Y:= 1/X; X:='X': Y:='Y': Digits:=16:
```

$$\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix}$$

$$Y := \begin{bmatrix} 4.000000000000001200000000000000480000000000000192000000000000 & \\ -6.000000000000002400000000000000960000000000000384000000000000 & \\ -6.000000000000002400000000000000960000000000000384000000000000 & \\ 12.000000000000004800000000000001920000000000000768000000000000 & \end{bmatrix}$$

NB: now decreasing Digits in Maple would still give above Y through Maple's remember table. Not sure about Pari.

Increasing precision seems to need elementwise operation

```
> pari("X[1,1] = precision( X[1,1], 60)"):
  pari("X[1,2]=precision(X[1,2],60)"): pari("X[2,1]=precision(X[2,1],60)"):
  pari("X[2,2]=precision(X[2,2],60)"):
  pari("X");
  pari("Y = 1/X");
"
```

```
[1.0000000000000000000000000000000000000000000000000000000000000000000 0.5000000000000000000000\
0000000000000000000000000000000000000000]

[0.5000000000000000000000000000000000000000000000000000000000000000000 0.3333333333333333333342\
368351437379203616728773340582847595215]
"
"
[3.9999999999999999999674739348254348669833028990263686111807 50 -5.99999999999999999349\
4786965086973396660579805273722236150 1]

[-5.99999999999999999934947869650869733966605798052737222361501 11.99999999999999998698\
957393017394679332115961054744447230 0]
"
[ >
```

Performance, memory issues

```
> restart_pari();
PARI 2.3.0 restarted and initialized.
```

1

```
> Digits:=40;
  cat("default(realprecision,",  convert(Digits,string), ")"); pari(%):
```

Digits := 40

"default(realprecision,40)"

```
> theDim:=100;
  cat("theDim =",  convert(%,string));: pari(%):
```

theDim := 100

"theDim =100"

```
> L:=[
  "setrand(1)",                                                  # seed for
  random generator
  "A = 1.0*matrix(theDim,theDim, i,j, random(1000)/1000)",  # random(n)
  gives a random integer between 0 and n-1
  "X = A * mattranspose(A)"                                      # make it
  symmetric
  ]:
```

To avoid error messages for I/O strings either add some trivial command as a last one (only that is returned).

Or increase space: we need 100*100*digits + some overhead, so 10^6 will be enough (note, that restarting Pari will not change the new buffer size, it is a global variable in the DLL for the maximal length of a string for communication between the 2 systems).

```
> buffersize_pari(1000000); # an administrative command, default is 100000
```

```
  for I/O strings
```

$$1000000$$

In depends on your PC whether you now may get some Pari error message "not enough memory" (and Pari is closed), but if so, then increase memory and run the script again.

To avoid that one may allocate lots of memory, even if working with medium sized matrices (I am not quite sure whether all memory for Pari is cleaned through the close command, may be I did not understand the manual completely for that point).

However repeated use will grap all your PC has ... so take care ... or use this:

```
> pari("allocatemem(32000000)"); # increase memory to 32 MB, default in the
  DLL is 16 MB, a pure Pari command
```

$$"0"$$

```
> 'map(pari,L)'; %:  # execute the script
```

$$\mathrm{map}(\mathrm{pari}, \mathrm{L})$$

Now you can invert:

```
> st:=time():
  pari("Y = 1/X"):
  `seconds needed` =time() - st;
```

$$\mathrm{seconds\ needed} = 0.531$$

This is quite fast (and includes returning the result to Maple ...), so check at least the first entries of the product:

```
> pari("Z=Y*X"): # should be the unity matrix
  'value_pari("Z[1,1]")': '%' = %;
  'value_pari("Z[1,2]")': '%' = %;
  'value_pari("Z[2,1]")': '%' = %;
  'value_pari("Z[2,2]")': '%' = %;
```

$$\mathrm{value\_pari}(\text{"Z[1,1]"}) = 1.000000000000000000000000000000000000000$$

$$\mathrm{value\_pari}(\text{"Z[1,2]"}) = -0.761956040\ 10^{-44}$$

$$\mathrm{value\_pari}(\text{"Z[2,1]"}) = 0.624015722\ 10^{-44}$$

$$\mathrm{value\_pari}(\text{"Z[2,2]"}) = 1.000000000000000000000000000000000000000$$

Maple is slower, relative exactness is comparable (well, it is better I guess):

```
> randomize(1):
  X:=evalf(LinearAlgebra[RandomMatrix](100,generator=rand(0..1000-1)/1000)):

  st:=time():
  Y:=1/X:
  `seconds needed` =time() - st;
```

$$\mathrm{seconds\ needed} = 5.421$$

```
> Z:=Y.X:
  Matrix(2, (i,j) -> Z[i,j]);
```

$$\begin{bmatrix} 1.000000000000000000000000000000000000002 & -0.315\ 10^{-39} \\ 0.3\ 10^{-39} & 1.000000000000000000000000000000000000001 \end{bmatrix}$$

```
> unassign('X','Y','Z'):
```

```
  stop_pari();
PARI 2.3.0 closed.
```

0

```
> 
```

Communication, exactness

To make use of Pari one needs to send and receive data for matrices (and vectors) and in the following I suggest a way for that - we already have seen that the structure seems to be quite similar. Since I am not used to work with regular expressions & grep commands I will do it the basic way through row or column operations.

For this let us take a ugly matrix of usual double floating numbers to be inverted (which can not be done in 16 digits precision the obvious way.

```
> Digits:=16:
```

some ugly data matrix, dim = 30, numerical singular

```
> originalData:=
  [
  [1424013813,  3932.813098,  724.1195293,  1564.476858,  13.62933267,
   12.80511561,  11.20914716,  7.857998376,  11.3823492,  13.93437693,
   12.58013573,  12.64639054,  6.121170734,  15.02906173,  29289629.14,
   48677.37199,  20887.16355,  30701.50155,  2865.133463,  2777.202947,
   2598.332247,  2175.220492,  2618.14654,  2897.04628,  2752.510332,
   2759.889884,  1919.869422,  2821.500771,  2430.813653,
   602492.2009],
  [3932.813098,  0.01086537,  0.002000546,  0.004322229,  3.76533E-05,
   3.53764E-05,  3.0967E-05,  2.1709E-05,  3.14457E-05,  3.84959E-05,
   3.47548E-05,  3.49375E-05,  1.69107E-05,  4.15059E-05,  80.9055647,
   0.134483094,  0.057705683,  0.084820125,  0.007915516,  0.007672601,
   0.007178406,  0.006009478,  0.007233166,  0.008003659,  0.007604373,
   0.007624722,  0.005304015,  0.007793387,  0.006713834,
   1.664528997],
  [724.1195293,  0.002000546,  0.000368354,  0.000795818,  6.93283E-06,
   6.5136E-06,  5.70173E-06,  3.99713E-06,  5.78988E-06,  7.08797E-06,
   6.39915E-06,  6.43279E-06,  3.11365E-06,  7.64305E-06,  14.89651925,
   0.024761255,  0.010625023,  0.015617262,  0.001457424,  0.001412698,
   0.001321706,  0.001106482,  0.00133179,  0.001473653,  0.001400136,
   0.001403883,  0.00097659,  0.001435031,  0.001236223,  0.306476535],
  [1564.476858,  0.004322229,  0.000795818,  0.001719405,  1.49786E-05,
   1.40728E-05,  1.23187E-05,  8.63585E-06,  1.25091E-05,  1.53137E-05,
   1.38255E-05,  1.38982E-05,  6.72708E-06,  1.65115E-05,  32.18428483,
   0.053497302,  0.022955344,  0.033741674,  0.0031488,  0.003052166,
   0.002855574,  0.002390575,  0.002877358,  0.003183862,  0.003025025,
   0.003033121,  0.002109941,  0.00310026,  0.002670911,  0.66215013],
  [13.62933267,  3.76533E-05,  6.93283E-06,  1.49786E-05,  1.30649E-07,
   1.22601E-07,  1.0732E-07,  7.52356E-08,  1.08981E-07,  1.33411E-07,
   1.20447E-07,  1.21081E-07,  5.86069E-08,  1.4396E-07,  0.280377882,
   0.000466044,  0.000199977,  0.000293941,  2.7448E-05,  2.65897E-05,
   2.4877E-05,  2.08261E-05,  2.5067E-05,  2.77369E-05,  2.63532E-05,
   2.64238E-05,  1.83814E-05,  2.70211E-05,  2.32785E-05,
   0.005768339],
  [12.80511561,  3.53764E-05,  6.5136E-06,  1.40728E-05,  1.22601E-07,
   1.15308E-07,  1.0083E-07,  7.0687E-08,  1.02391E-07,  1.25344E-07,
   1.13165E-07,  1.13758E-07,  5.50622E-08,  1.35257E-07,  0.263422863,
   0.000437862,  0.000187884,  0.000276166,  2.57727E-05,  2.49949E-05,
   2.33727E-05,  1.95669E-05,  2.35512E-05,  2.60596E-05,  2.47598E-05,
```

```
        2.48258E-05,  1.72697E-05,  2.53873E-05,  2.18717E-05,
        0.005419526],
      [11.20914716,  3.0967E-05,  5.70173E-06,  1.23187E-05,  1.0732E-07,
        1.0083E-07,  8.83673E-08,  6.18761E-08,  8.96281E-08,  1.09721E-07,
        9.90595E-08,  9.95784E-08,  4.82003E-08,  1.18375E-07,  0.230590129,
        0.000383286,  0.000164466,  0.000241744,  2.25603E-05,  2.1868E-05,
        2.04716E-05,  1.7128E-05,  2.06157E-05,  2.28115E-05,  2.16736E-05,
        2.17314E-05,  1.51174E-05,  2.22206E-05,  1.91471E-05,
        0.004744022],
      [7.857998376,  2.1709E-05,  3.99713E-06,  8.63585E-06,  7.52356E-08,
        7.0687E-08,  6.18761E-08,  4.34768E-08,  6.28323E-08,  7.69208E-08,
        6.94456E-08,  6.98108E-08,  3.37907E-08,  8.30157E-08,  0.161651516,
        0.000268696,  0.000115296,  0.000169471,  1.58156E-05,  1.53304E-05,
        1.43429E-05,  1.20211E-05,  1.44523E-05,  1.59919E-05,  1.51941E-05,
        1.52348E-05,  1.05979E-05,  1.55808E-05,  1.3424E-05,  0.003325719],
      [11.3823492,  3.14457E-05,  5.78988E-06,  1.25091E-05,  1.08981E-07,
        1.02391E-07,  8.96281E-08,  6.28323E-08,  9.11711E-08,  1.11418E-07,
        1.00591E-07,  1.01119E-07,  4.89453E-08,  1.20276E-07,  0.234153733,
        0.00038921,  0.000167008,  0.00024548,  2.29092E-05,  2.22062E-05,
        2.07759E-05,  1.73927E-05,  2.09525E-05,  2.31642E-05,  2.20087E-05,
        2.20675E-05,  1.53511E-05,  2.25716E-05,  1.94535E-05,
        0.004817349],
      [13.93437693,  3.84959E-05,  7.08797E-06,  1.53137E-05,  1.33411E-07,
        1.25344E-07,  1.09721E-07,  7.69208E-08,  1.11418E-07,  1.36551E-07,
        1.23143E-07,  1.23789E-07,  5.9918E-08,  1.47214E-07,  0.286652414,
        0.000476472,  0.000204451,  0.000300518,  2.80452E-05,  2.71845E-05,
        2.54336E-05,  2.12924E-05,  2.56278E-05,  2.83736E-05,  2.6943E-05,
        2.70149E-05,  1.87927E-05,  2.76291E-05,  2.38E-05,  0.005897413],
      [12.58013573,  3.47548E-05,  6.39915E-06,  1.38255E-05,  1.20447E-07,
        1.13165E-07,  9.90595E-08,  6.94456E-08,  1.00591E-07,  1.23143E-07,
        1.11352E-07,  1.11761E-07,  5.4096E-08,  1.32925E-07,  0.258794118,
        0.000430167,  0.000184582,  0.000271313,  2.53198E-05,  2.45429E-05,
        2.29621E-05,  1.92232E-05,  2.31373E-05,  2.56018E-05,  2.43439E-05,
        2.43898E-05,  1.69665E-05,  2.4946E-05,  2.14897E-05,  0.005324286],
      [12.64639054,  3.49375E-05,  6.43279E-06,  1.38982E-05,  1.21081E-07,
        1.13758E-07,  9.95784E-08,  6.98108E-08,  1.01119E-07,  1.23789E-07,
        1.11761E-07,  1.1248E-07,  5.43802E-08,  1.33568E-07,  0.260155994,
        0.000432429,  0.000185553,  0.000272739,  2.5453E-05,  2.46718E-05,
        2.30826E-05,  1.93242E-05,  2.32588E-05,  2.57363E-05,  2.44526E-05,
        2.45324E-05,  1.70557E-05,  2.50712E-05,  2.16032E-05,
        0.005352282],
      [6.121170734,  1.69107E-05,  3.11365E-06,  6.72708E-06,  5.86069E-08,
        5.50622E-08,  4.82003E-08,  3.37907E-08,  4.89453E-08,  5.9918E-08,
        5.4096E-08,  5.43802E-08,  2.63781E-08,  6.46807E-08,  0.125922103,
        0.000209307,  8.98126E-05,  0.000132013,  1.232E-05,  1.19418E-05,
        1.11728E-05,  9.35349E-06,  1.1258E-05,  1.24571E-05,  1.18358E-05,
        1.18674E-05,  8.26427E-06,  1.21384E-05,  1.04589E-05,
        0.002590641],
      [15.02906173,  4.15059E-05,  7.64305E-06,  1.65115E-05,  1.4396E-07,
        1.35257E-07,  1.18375E-07,  8.30157E-08,  1.20276E-07,  1.47214E-07,
        1.32925E-07,  1.33568E-07,  6.46807E-08,  2.44439E-07,  0.309110243,
        0.000513712,  0.000220443,  0.000324009,  3.02498E-05,  2.93217E-05,
        2.74305E-05,  2.29682E-05,  2.76478E-05,  3.05899E-05,  2.90659E-05,
        2.91375E-05,  2.0274E-05,  3.69647E-05,  2.59052E-05,  0.006358139],
      [29289629.14,  80.9055647,  14.89651925,  32.18428483,  0.280377882,
        0.263422863,  0.230590129,  0.161651516,  0.234153733,  0.286652414,
        0.258794118,  0.260155994,  0.125922103,  0.309110243,  602492.2009,
        1001.388,  429.6892162,  631.5891963,  58.94095231,  57.13211627,
        53.45230862,  44.74813272,  53.85998785,  59.59738048,  56.62408605,
        56.77577774,  39.49509266,  58.03642325,  49.99876155,
```

```
    12394.43865],
   [48677.37199,  0.134483094,  0.024761255,  0.053497302,  0.000466044,
    0.000437862,  0.000383286,  0.000268696,  0.00038921,  0.000476472,
    0.000430167,  0.000432429,  0.000209307,  0.000513712,  1001.388,
    1.664528997,  0.714237606,  1.049839806,  0.097972241,  0.094965642,
    0.088848853,  0.074380713,  0.089526618,  0.099063222,  0.094121128,
    0.094373037,  0.065649034,  0.096458913,  0.083097392,  20.6022915],
   [20887.16355,  0.057705683,  0.010625023,  0.022955344,  0.000199977,
    0.000187884,  0.000164466,  0.000115296,  0.000167008,  0.000204451,
    0.000184582,  0.000185553,  8.98126E-05,  0.000220443,  429.6892162,
    0.714237606,  0.306476535,  0.450479355,  0.042039275,  0.040749165,
    0.038124493,  0.031916333,  0.038415344,  0.042507415,  0.040386794,
    0.040494891,  0.028169622,  0.041391273,  0.03565737,  8.840306595],
   [30701.50155,  0.084820125,  0.015617262,  0.033741674,  0.000293941,
    0.000276166,  0.000241744,  0.000169471,  0.00024548,  0.000300518,
    0.000271313,  0.000272739,  0.000132013,  0.000324009,  631.5891963,
    1.049839806,  0.450479355,  0.66215013,  0.061792462,  0.059896137,
    0.056038188,  0.046912948,  0.056465667,  0.06248055,  0.059363496,
    0.059522394,  0.041405753,  0.060838412,  0.052412077,
    12.99414338],
   [2865.133463,  0.007915516,  0.001457424,  0.0031488,  2.7448E-05,
    2.57727E-05,  2.25603E-05,  1.58156E-05,  2.29092E-05,  2.80452E-05,
    2.53198E-05,  2.5453E-05,  1.232E-05,  3.02498E-05,  58.94095231,
    0.097972241,  0.042039275,  0.061792462,  0.005768339,  0.005589639,
    0.005229605,  0.004378026,  0.005269522,  0.005830813,  0.005539931,
    0.005554768,  0.003864091,  0.005678886,  0.004892283,
    1.212627255],
   [2777.202947,  0.007672601,  0.001412698,  0.003052166,  2.65897E-05,
    2.49949E-05,  2.1868E-05,  1.53304E-05,  2.22062E-05,  2.71845E-05,
    2.45429E-05,  2.46718E-05,  1.19418E-05,  2.93217E-05,  57.13211627,
    0.094965642,  0.040749165,  0.059896137,  0.005589639,  0.005419526,
    0.005069118,  0.004243688,  0.005107815,  0.005651873,  0.005369939,
    0.005384288,  0.003745496,  0.00550463,  0.004742246,  1.175414085],
   [2598.332247,  0.007178406,  0.001321706,  0.002855574,  2.4877E-05,
    2.33727E-05,  2.04716E-05,  1.43429E-05,  2.07759E-05,  2.54336E-05,
    2.29621E-05,  2.30826E-05,  1.11728E-05,  2.74305E-05,  53.45230862,
    0.088848853,  0.038124493,  0.056038188,  0.005229605,  0.005069118,
    0.004744022,  0.003970343,  0.004778811,  0.00528784,  0.005024054,
    0.005037478,  0.003504269,  0.005149796,  0.004436971,  1.09970484],
   [2175.220492,  0.006009478,  0.001106482,  0.002390575,  2.08261E-05,
    1.95669E-05,  1.7128E-05,  1.20211E-05,  1.73927E-05,  2.12924E-05,
    1.92232E-05,  1.93242E-05,  9.35349E-06,  2.29682E-05,  44.74813272,
    0.074380713,  0.031916333,  0.046912948,  0.004378026,  0.004243688,
    0.003970343,  0.003325719,  0.004000628,  0.004426804,  0.00420596,
    0.004217219,  0.002933647,  0.00431168,  0.003714627,  0.92062854],
   [2618.14654,  0.007233166,  0.00133179,  0.002877358,  2.5067E-05,
    2.35512E-05,  2.06157E-05,  1.44523E-05,  2.09525E-05,  2.56278E-05,
    2.31373E-05,  2.32588E-05,  1.1258E-05,  2.76478E-05,  53.85998785,
    0.089526618,  0.038415344,  0.056465667,  0.005269522,  0.005107815,
    0.004778811,  0.004000628,  0.004817349,  0.005328188,  0.005062381,
    0.005075924,  0.003531001,  0.005189947,  0.00447202,  1.108093575],
   [2897.04628,  0.008003659,  0.001473653,  0.003183862,  2.77369E-05,
    2.60596E-05,  2.28115E-05,  1.59919E-05,  2.31642E-05,  2.83736E-05,
    2.56018E-05,  2.57363E-05,  1.24571E-05,  3.05899E-05,  59.59738048,
    0.099063222,  0.042507415,  0.06248055,  0.005830813,  0.005651873,
    0.00528784,  0.004426804,  0.005328188,  0.005897413,  0.00560163,
    0.005616606,  0.003907115,  0.005742469,  0.004946826,  1.22613078],
   [2752.510332,  0.007604373,  0.001400136,  0.003025025,  2.63532E-05,
    2.47598E-05,  2.16736E-05,  1.51941E-05,  2.20087E-05,  2.6943E-05,
    2.43439E-05,  2.44526E-05,  1.18358E-05,  2.90659E-05,  56.62408605,
```

```
    0.094121128,  0.040386794,  0.059363496,  0.005539931,  0.005369939,
    0.005024054,  0.00420596,  0.005062381,  0.00560163,  0.005324286,
    0.005336429,  0.003712212,  0.005456205,  0.004700332,  1.16496087],
   [2759.889884,  0.007624722,  0.001403883,  0.003033121,  2.64238E-05,
    2.48258E-05,  2.17314E-05,  1.52348E-05,  2.20675E-05,  2.70149E-05,
    2.43898E-05,  2.45324E-05,  1.18674E-05,  2.91375E-05,  56.77577774,
    0.094373037,  0.040494891,  0.059522394,  0.005554768,  0.005384288,
    0.005037478,  0.004217219,  0.005075924,  0.005616606,  0.005336429,
    0.005352282,  0.003722144,  0.005470155,  0.004712966,
    1.168079265],
   [1919.869422,  0.005304015,  0.00097659,  0.002109941,  1.83814E-05,
    1.72697E-05,  1.51174E-05,  1.05979E-05,  1.53511E-05,  1.87927E-05,
    1.69665E-05,  1.70557E-05,  8.26427E-06,  2.0274E-05,  39.49509266,
    0.065649034,  0.028169622,  0.041405753,  0.003864091,  0.003745496,
    0.003504269,  0.002933647,  0.003531001,  0.003907115,  0.003712212,
    0.003722144,  0.002590641,  0.003805745,  0.00327887,  0.81255444],
   [2821.500771,  0.007793387,  0.001435031,  0.00310026,  2.70211E-05,
    2.53873E-05,  2.22206E-05,  1.55808E-05,  2.25716E-05,  2.76291E-05,
    2.4946E-05,  2.50712E-05,  1.21384E-05,  3.69647E-05,  58.03642325,
    0.096458913,  0.041391273,  0.060838412,  0.005678886,  0.00550463,
    0.005149796,  0.00431168,  0.005189947,  0.005742469,  0.005456205,
    0.005470155,  0.003805745,  0.006358139,  0.004843146,  1.19387532],
   [2430.813653,  0.006713834,  0.001236223,  0.002670911,  2.32785E-05,
    2.18717E-05,  1.91471E-05,  1.3424E-05,  1.94535E-05,  2.38E-05,
    2.14897E-05,  2.16032E-05,  1.04589E-05,  2.59052E-05,  49.99876155,
    0.083097392,  0.03565737,  0.052412077,  0.004892283,  0.004742246,
    0.004436971,  0.003714627,  0.00447202,  0.004946826,  0.004700332,
    0.004712966,  0.00327887,  0.004843146,  0.004998033,  1.028500935],
   [602492.2009,  1.664528997,  0.306476535,  0.66215013,  0.005768339,
    0.005419526,  0.004744022,  0.003325719,  0.004817349,  0.005897413,
    0.005324286,  0.005352282,  0.002590641,  0.006358139,  12394.43865,
    20.6022915,  8.840306595,  12.99414338,  1.212627255,  1.175414085,
    1.09970484,  0.92062854,  1.108093575,  1.22613078,  1.16496087,
    1.168079265,  0.81255444,  1.19387532,  1.028500935,  255]
   ]:
> originalMatrix:=convert(originalData,Matrix):
  m:=LinearAlgebra[RowDimension](originalMatrix);
  LinearAlgebra[Determinant](originalMatrix);
```

$$m := 30$$

$$0.1005136328082889\ 10^{-211}$$

The usual way is: convert to rationals and invert (but that would not give something useful for usual numerical C programs, I know ...)

```
> Digits:=1000:
  A:=convert( convert(originalData,rational) ,Matrix):
  Digits:=16:

  'det(A)': '%'= map(evalhf, LinearAlgebra[Determinant](A)); # hardware
  floatings = double precision

  B:= 'A^(-1)';
  B:= LinearAlgebra[MatrixInverse](A):

  E := Matrix(m,m,shape=identity):
  `norm(A*B - identity)` = LinearAlgebra[MatrixNorm](A.B - E);
```

$$\det(A) = 0.100513630287470666\ 10^{-211}$$

$$B := \frac{1}{A}$$

$$\text{norm}(A*B - \text{identity}) = 0$$

The answer is almost immediate (here) and B is an exact inverse, as the norm shows (Maple uses GMP for that?).

Sending a Maple matrix to Pari

```
> restart_pari():
  pari("allocatemem(32000000)");
  buffersize_pari(1000000); # actually already done ...
PARI 2.3.0 restarted and initialized.
```

"0"

1000000

```
> m:=LinearAlgebra[RowDimension](originalMatrix);
  cat("m =",  convert(%,string)); pari(%):
```

m := 30

"m =30"

```
> LA:= convert(A, listlist): # list of the rows of A
> pari("A = matrix(m,m)"): # Pari matrix, initialized with an integer 0

  for i from 1 to m do
    iStr := convert(i, string);
    convert(LA[i], string);
    cat("A[", iStr, ",] = ", %);
    pari(%);
  end do:
> pari("B = 1/A"): # slow as no GMP is available
> value_pari("B[1,1]");
  B[1,1];
```

3847236593124687185596986767911625984267428943092908771119206620268503278348400876\

0000 / 3242375170563570222279580758662031810153357964228224051589123339445557749\

206610842378641

3847236593124687185596986767911625984267428943092908771119206620268503278348400876\

0000 / 3242375170563570222279580758662031810153357964228224051589123339445557749\

206610842378641

Slow, yes (lots of Euclidean Algorithm and no GMP for the DLL) ... but exact. And Pari is using 'symbolic' rational numbers.

Increase precision to 100 digits to see, what happens for floating point numbers:

```
> Digits:=100;
  cat("default(realprecision,",  convert(Digits,string), ")");: pari(%):
```

Digits := 100

"default(realprecision,100)"

Conversion to floating point numbers can either be done as in Maple ...

```
> pari("A_num = 1.0 * A"):
  pari("B_num = 1/A_num"):
```

```
> value_pari("B_num[1,1]");
  value_pari("1.0*B[1,1]");
  evalf[100](B[1,1]);
```

```
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704377
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704366
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704366
```

... or sending data as floats

```
> pari("A_num = matrix(m,m)"): # reset the matrix

  for i from 1 to m do
    iStr := convert(i, string);
    convert(LA[i], string); evalf[100](%):
    cat("A_num[", iStr, ",] = 1.0*", %);
    pari(%);
  end do:

  pari("B_num = 1/A_num"):
> value_pari("B_num[1,1]");
  value_pari("1.0*B[1,1]");
  evalf[100](B[1,1]);
```

```
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704377
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704366
0.011865488694992638916204448217972035546821487959721296607187677828008866305069231\
    315419301114476704366
```

In any case the floating point inversion is fast (and slightly less exact than the one over the Rationals).

```
[ >
```

Receiving a matrix from Pari

Assign by looping through the rows (we already know the matrix size, otherwise determine it by matsize)

```
> 'pari("matsize(B_num)")': '%'=%;

  tmp:='tmp': tmp:=NULL:
  for i from 1 to m do
    convert(i,string); cat("i =", %); pari(%); # set row index for Pari
    tmp:= tmp, value_pari("B_num[i,]");        # build sequence of rows
  end do: i:='i': pari("i=0"):                 # clean up index counter

  B_mpl := convert([tmp], Matrix):             # convert list of lists
```

```
   to a Matrix of floats
```

$$\text{pari("matsize(B\_num)")} = \text{"[30, 30]"}$$

```
> B_mpl[1,1];
```

0.011865488694992638916204448217972035546821487959721296607187677828008866305069235\
3154193011144767043 77

```
> `norm(B - B_mpl)` = LinearAlgebra[MatrixNorm](B - B_mpl);
  `norm(evalf(B) - B_mpl)` = LinearAlgebra[MatrixNorm](evalf(B) -
  B_mpl);
```

$$\text{norm(B - B\_mpl)} = 0.11110194682\ 10^{-83}$$

$$\text{norm(evalf(B) - B\_mpl)} = 0.11110194682\ 10^{-83}$$

Not that bad ... but a lot of digits are lost against an exact solution, Maple would do it much better and is exact within the chosen working precision:

```
> B_float:='evalf(1/A)';
  `norm(B - B_float)` = LinearAlgebra[MatrixNorm](B - B_float);
```

$$\text{B\_float} := \text{evalf}\left(\frac{1}{A}\right)$$

$$\text{norm(B - B\_float)} = 0.$$

```
[ >
```

A short lesson about long floats ...

A large error occurs in double precision for the 11th unit vector, **e11 – (A . B . e11)** should 0 of course:

```
> e11:= Vector(m): e11[11]:=1: 'e11' = convert(e11,list);
```

$$\text{e11} = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> oldDigits:=Digits: Digits:= 16;
  e11 - originalMatrix.evalf(B).e11: #convert(%,list);
  'norm(e11-(A.(B.e11)))' = LinearAlgebra[VectorNorm](%);
  Digits:= oldDigits:
```

$$\text{Digits} := 16$$

$$\text{norm(e11 – (A . (B . e11)))} = 0.5$$

```
> pari("e11 = vectorv(m)"): # column vector with 0 entries
  pari("e11[11] = 1"):
  pari("e11 - A_num*(B_num*e11)");
```

"[0.E-91, -2.340838178 E-97, 8.77814316 E-98, 1.170419089 E-97, -9.14389913 E-100, 1.37158486\
9 E-99, 3.428962174 E-99, 1.142987391 E-99, -3.200364696 E-99, 9.14389913 E-100, 1.371584869 \
E-99, -4.57194956 E-100, 1.371584869 E-99, 1.828779826 E-99, 3.835229270 E-93, -9.36335271 E-\
97, 3.745341084 E-96, 3.745341084 E-96, -5.85209544 E-98, 9.36335271 E-97, -2.926047722 E-97,\
2.340838178 E-97, 5.85209544 E-97, 3.511257267 E-97, 2.340838178 E-97, 5.26688590 E-97, 1.17\
0419089 E-97, 0.E-97, 2.340838178 E-97, -5.99254573 E-95]~"

```
> pari("default(realprecision,16)");
  pari("e11 = vectorv(m)"): # column vector with 0 entries
  pari("e11[11] = 1"):
  pari("e11 - 1.0*A_num*(1.0*B_num*e11)");
```

"16"

"[-0.0000610351562, 0.0000000002910383046, 5.093170330 E-11, -5.82076609 E-11, 1.591615728 \
E-12, -2.273736755 E-13, 1.136868377 E-13, 3.410605132 E-13, 1.136868377 E-12, 6.82121026 E-\
13, -1.136868377 E-12, 0.E-12, -1.136868377 E-13, 6.82121026 E-13, -0.000001430511474, -0.000\
000006053596736, 0.E-9, -0.000000003259629011, -2.037268132 E-10, 0.0000000002910383046, -\

```
2.910383046 E-11, -2.910383046 E-11, 2.910383046 E-11, -1.455191523 E-10, 0.000000000232830\
6437, 5.82076609 E-11, -1.455191523 E-10, 2.910383046 E-11, -5.82076609 E-11, 0.000000044703\
48358]~"
```

Certainly bad news: working with 100 digit floats looses ~ 10 digits (internally Pari uses 105) and switching to 16 (internally 24) gives 4 digits exactness ...

Working over the Rationals is ok, as in Maple:

```
> pari("e11 = vectorv(m)"): # column vector with 0 entries
  pari("e11[11] = 1"):
  pari("e11 - A*(B*e11)");
        "[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]~"
> # clean up Maple ... hm: this is still stored in the sheet?
  unassign('e11','B_float','B_mpl','A','B');
```

```
>
```

Appendix: converting to Rationals in Pari

Above converting to rationals was used ...

```
> restart_pari():
PARI 2.3.0 restarted and initialized.
> Digits:=40;
  cat("default(realprecision,",  convert(Digits,string), ")");: pari(%):
                                        Digits := 40
                                "default(realprecision,40)"
> # a bit ugly to write and read code, avoids brackets ...
  # for real numbers only (but use real() and imag() in the complex case)
  "
  convertRational(x) =
  {
  local(X,i,cf,M);
  global(d);

  X = abs(x);
  i=0;
  while( abs(i)!=20,
    if (10^d < X && X!=0,
      X = X/10^d; i=i+1,
      if(X < 1/10^d,
        X = X*10^d; i=i-1,
        break
      );
    );
  );

  cf = contfrac(X);
  M = contfracpnqn(cf);
  return( sign(x)*10^(i*d)*M[1,1]/M[2,1] );
  }
  ";
  pari(%):
"
    convertRational(x) =
    {
    local(X,i,cf,M);
```

```
    global(d);

    X = abs(x);
    i=0;
    while( abs(i)!=20,
      if (10^d < X && X!=0,
        X = X/10^d; i=i+1,
        if(X < 1/10^d,
          X = X*10^d; i=i-1,
          break
        );
      );
    );

    cf = contfrac(X);
    M = contfracpnqn(cf);
    return( sign(x)*10^(i*d)*M[1,1]/M[2,1] );
    }
    "
> # set global needed in the conversion routine
  'pari("d=precision(0.0)")'; %;
                                  pari("d=precision(0.0)")
                                          "48"
> x:=value_pari("x = -Pi*10^(13)");
                  x := -0.31415926535897932384626433832795028841971 10^14
> # result from Pari
  value_pari("convertRational(x)");
  value_pari("1.0*convertRational(x)");
                          -2816211357252516227090327662031
                                  89642791659654709
                   -0.31415926535897932384626433832795028841971 10^14
> # Maple's representation
  convert(x , rational);
  1.0*%;
                          -16231648462978540442894701 5
                                  5166694174826
                   -0.31415926535897932384626433832795028842000 10^14
> x:=value_pari("x = -Pi*10^(-80.1)");
                  x := -0.24954557467487516695544419253314106625220 10^-79
> value_pari("convertRational(x)");
  value_pari("1.0*convertRational(x)");
-33876957 / 135754589293507561410711807777935071903500000000000000000000000000000000\
    0000000000000
                   -0.24954557467487516695544419253314106625220 10^-79
> convert(x , rational);
  1.0*%;
                                          -1
    40072840454208316706459735382311199671897270706088174310968935214339539186111043
```

-0.24954557467487516695544419253314106625 10^{-79}

```
> stop_pari();
PARI 2.3.0 closed.
```

0

```
>
>
>
>
```