This worksheet shows, how to call Pari from Maple through pari_mpl.dll

In general only Pari commands can be input (described in the Pari manual user.pdf) while all 'low level' commands for the C programs (see the manual libpari.pdf) are not accepted.

AVt, Nov 2006

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

I have not tested, which Maple version is needed, however I compiled the DLL against files coming with Maple 10.

Choose some precision for Maple, default would be 10 (this is relative precision, not absolute one), Pari's default is 28:

```
Digits:=28;
```

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

Digits := 28

```
> 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**

Let's do some simple examples:

```
> pari("1/2 + 1/4");
  evalf_pari("1/2 + 1/4");
  value_pari("sqrt(2)");
```

"3/4"

0.750000000000000000000000000

1.41421356237309504880168872

Look at Pari's Pi:

```
> pari("139755218526789 / 44485467702853");
  evalf_pari(%);
  ``;
  'evalf_pari("Pi") - evalf_pari("139755218526789/44485467702853")': '%'=%;
```

"139755218526789/44485467702853"

3.14159265358979323846264338

$$\text{evalf\_pari}(\text{"Pi"}) - \text{evalf\_pari}(\text{"139755218526789/44485467702853"}) = 0.$$

Check, whether parsing Pari's floating point results work:

```
> tst:="10^31*exp(I*(sqrt(2)+Pi))";
  pari(tst);
  value_pari(tst);
  ``;
  tst:="exp(-100)*0.";
  pari(tst);
  value_pari(tst);
```

tst := "10^31*exp(I*(sqrt(2)+Pi))"

"-1.5594369476537447345464797 89 E30 - 9.87765945992735527069134072 1 E30*I"

$-0.15594369476537447345464797\,89 \cdot 10^{31} - 0.98776594599273552706913407\,21 \cdot 10^{31}\,I$

tst := "exp(-100)*0."

"0.E-72"

0.

It does. But I am not sure whether that covers all needs, so adjust if needed to have real and complex numerics of 'arbitrary' precision and nice speed.

Note the principle for calling Pari: store the Pari command in a string and call to execute it.

This also works for a sequence of commands (in Pari they are to be separated by semicolons):

```
> tstStr:="default(realprecision,18); simplify(exp(log(2^3) + I*Pi))";
  value_pari(tstStr);
```

$$\text{tstStr} := \text{"default(realprecision,18); simplify(exp(log(2\^3) + I*Pi))"}$$

$$-8.00000000000000000 - 0.1734723477\ 10^{-17}\ I$$

Note that Pari is now set to 18 as precision

```
> evalf_pari("1248285321/1111111111");
  pari("precision(0.0)");
  pari("precision(0.)");
```

$$1.12345678901234568$$

$$\text{"19"}$$

$$\text{"19"}$$

If the command string causes an error in Pari, then the error message is handed over to Maple (Maple says **ln** while Pari wants **log** for the natural logarithm):

```
> pari("ln(1.0)");
error processing: PARI 2.3.0 closed!
Error, (in ModuleApply)  [PARI error message]:
  unknown function or error in formal parameters
```

Most in Pari's notation is quite common for a Maple user, but one should download the documentation from `http://pari.math.u-bordeaux.fr` and have a look into tutorial.pdf and users.pdf (note that one does not need the interface GP here).

It is somewhat technical and not really for teaching oneself the easy way - but I am not aware of lecture notes (or a book) which teaches the system step by step.

The main purpose of Pari is number theory while I am interested 'only' in its numerics and I try to give some hacks how to do that.

Let us restart Pari (it would be done automatically at the first call).

```
> start_pari();
PARI 2.3.0 initialized.
```

$$1$$

Since Pari is running in the background (if it was not stopped) and thus remembers commands already given
one can use that to execute series of commands.

A simple thing is to define a function (see the manual) and evaluate it later:

```
> str1:="f(z) = {z^2}";
  str2:="f(3)";
  pari(str1):
  pari(str2);
```

str1 := "f(z) = {z^2}"

str2 := "f(3)"

"9"

```
> "f(x) = {atanh(x)}";
  pari(%):
  pari("f(5.5)");
  value_pari("f(5.5)");
```

"f(x) = {atanh(x)}"

"0.183862390062658677 + 1.57079632679489662*I"

0.183862390062658677 + 1.57079632679489662 I

As already said not all commands are possible: no C library commands are accepted:

```
> "f(x) = {gtofp(x, 28)}";
  pari(%);
  pari("f(1/2)");
```

"f(x) = {gtofp(x, 28)}"

"0"

```
error processing: PARI 2.3.0 closed!
Error, (in ModuleApply)  [PARI error message]:
  unknown function or error in formal parameters
```

To have an easy way for calling Pari functions one can use the following syntax:

```
> p_fct:=proc(fctName, z::complex)
  # call Pari by fctName(z) with an appropriate string fctName(z)
    pari(cat( convert(fctName,string), "(", convert(z,string), ")" ));
  end proc; #maplemint(%);
```

p_fct := **proc**(fctName, z::complex) pari(cat(convert(fctName, string), "(", convert(z, string), ")")) **end proc**

```
> p_fct("exp", sqrt(2.0)); parse(%); ``;
  p_fct("gamma", sqrt(2.0)); parse(%); ``;
  p_fct(gamma, sqrt(2.0)); parse(%);
```

"4.11325037878292752"

4.11325037878292752

"0.886581428719259125"

0.886581428719259125

"0.886581428719259125"

0.886581428719259125

```
> p_fct(gamma, sqrt(-2.0)); pp(%);
```

"-0.0451511912606807269 - 0.224110916771629123*I"

-0.0451511912606807269 – 0.224110916771629123 I

Note that the function name must be kown in Pari.

Using a Maple function would need a callback - I have not (yet?) implemented that.

A systematic way to work with command string is to put them into a list and to use Maple's mapping command
to let Pari execute each:

```
> L:=[
  "default(realprecision,60)",
  "zet(s) ={ local(n); sumalt(n=1, (-1)^(n-1)*n^(-s)) / (1 - 2^(1-s))}"
  ];

  map(pari, L);
```

$$L := [\text{"default(realprecision,60)"}, \text{"zet(s) ={ local(n); sumalt(n=1, (-1)^(n-1)*n^(-s)) / (1 - 2^(1-s))}"}]$$

$$[\text{"60"}, \text{"0"}]$$

With this example from the manual let us check Maple's value for Zeta(2):

```
> value_pari( "zet(2)" );
  evalf[60](Zeta(2));
```

$$1.64493406684822643647241516664602518921894990120679843773556$$

$$1.64493406684822643647241516664602518921894990120679843773556$$

This is convenient, if one wants to re-run the 'script' with some different setting.

Finding errors or debugging however is messy, if one (say: me) is not used to Pari.

Anyway: if the stuff is not too long or complicated it can be done, existing scripts can be incorporated into Maple.
Hope you enjoy it ...

Let us make some performance checks for medium precision with 60 Digits (I will not care to test exactness here,
that really is ok).

Restart Pari to have a clean stack (hm ... not sure about that ...).

```
> restart_pari();
  pari("default(realprecision,60)");
  Digits:=60;
PARI 2.3.0 restarted and initialized.
```

$$1$$

$$\text{"0"}$$

$$Digits := 60$$

The following have the same names in Maple and Pari:

```
> timingList:= [sqrt, exp, log, sin, tan];
  for fct in timingList do
  forget(evalf); gc();
  print(``);
  fctName:=convert(fct,string): print(fctName);
```

```
imax:= 1000;
i:='i':
myZ:= (10.0+i/imax) + I * (-2.0+i/imax);

  forget(evalf); gc(); st:=time():
  for i from 1 to imax do
    pp(p_fct(fctName,myZ));
  end do:
  tPari:=time()-st:
  print('Pari'=tPari);

  forget(evalf); gc(); st:=time():
  for i from 1 to imax do
    evalf(fct(myZ));
  end do:
  tMaple:=time()-st:
  print('Maple'=tMaple);

  print('factor'=evalf[2](tMaple/tPari));

end do:
```

timingList := [sqrt, exp, log, sin, tan]

"sqrt"
Pari = 0.422
Maple = 0.329
factor = 0.79

"exp"
Pari = 0.485
Maple = 0.969
factor = 2.0

"log"
Pari = 0.531
Maple = 0.891
factor = 1.7

"sin"
Pari = 0.485
Maple = 0.860
factor = 1.8

"tan"
Pari = 0.500
Maple = 0.875
factor = 1.8

```
> myZ;
  pp(p_fct(fctName,myZ)); # Pari
  evalf(fct(myZ)); # Maple
```

```
11.001000000000000000000000000000000000000000000000000000000000 –
    0.999000000000000000000000000000000000000000000000000000000000 I
-0.003938721944142570862541568767772039692258429686280629195254 38 –
    1.313732223064146228970795239470516374012898207245097973851 30 I
-0.003938721944142570862541568767772039692258429686280629195254 38 –
    1.313732223064146228970795239470516374012898207245097973851 30 I
```

The last 3 commands are just to see we are not looking at empty commands ...

For the performance it must be said, that the DLL used here is by no means optimal: the sources are not completely the original ones, since I wanted to have a native Windows DLL (not depending on cygwin.dll). Thus they are for a generic processor and miss the assembler parts for the distributed version. Additionally the Pari group recommends to invoke GMP - which I have not done.

So this might look as if it not very useful - but let us look at some special functions (still with 60 Digits).

The spelling for the Gamma function is different in Pari and Maple:

```
> gc();
  imax:= 100;
  i:='i':
  myZ:= (10+i/imax) + I * (-2.0+i/imax);

    forget(evalf); gc(); st:=time():
    for i from 1 to imax do
      p_fct(gamma,myZ);
    end do:
    tPari:=time()-st:
    print('Pari'=tPari);

    forget(evalf); gc(); st:=time():
    for i from 1 to imax do
      evalf(GAMMA(myZ));
    end do:
    tMaple:=time()-st:
    print('Maple'=tMaple);

    print('factor'=evalf[2](tMaple/tPari));
```

$$\text{imax} := 100$$

$$\text{myZ} := 10 + \frac{\text{i}}{100} + \left(-2.0 + \frac{\text{i}}{100}\right) \text{I}$$

$$\text{Pari} = 0.078$$

$$\text{Maple} = 0.906$$

$$\text{factor} = 12.$$

Dito for Bessel functions:

```
> pari("besselK1(z) = {besselk(1,z)}");

  pp(pari("besselK1(-1 - 2*I)")); # simple check
  BesselK(1,-1 - 2*I): evalf(%);
```

"0"

-2.183454386204148296561785644538159814228230893355209793143 94 – 0.099929752591113738696765568814 I

-2.183454386204148296561785644538159814228230893355209793143 94 –

```
    0.09992975259111373869676556881380032802998874955080673975513 87 I
> gc();
  imax:= 100;
  i:='i':
  myZ:= (10+i/imax) + I * (-2.0+i/imax);

    forget(evalf); gc(); st:=time():
    for i from 1 to imax do
      pp(p_fct(besselK1,myZ));
    end do:
    tPari:=time()-st:
    print('Pari'=tPari);

    forget(evalf); gc(); st:=time():
    for i from 1 to imax do
      evalf(BesselK(1,myZ));
    end do:
    tMaple:=time()-st:
    print('Maple'=tMaple);

    print('factor'=evalf[2](tMaple/tPari));
```

$$\text{imax} := 100$$

$$\text{myZ} := 10 + \frac{\text{i}}{100} + \left(-2.0 + \frac{\text{i}}{100}\right) \text{I}$$

$$\text{Pari} = 0.141$$

$$\text{Maple} = 1.641$$

$$\text{factor} = 11.$$

Note however that Pari does not have the rich catalogue available in Maple, so it may be need to convert to those:

```
> convert(BesselK(nu,x), BesselI); # Pari only has Bessel in special cases
```

$$\frac{1}{2}\,\frac{\pi\,(-\text{BesselI}(-\nu, x) + \text{BesselI}(\nu, x))}{\sin(\nu\,\pi)}$$

```
> pp(pari("besseli(1/2,-1 - 2*I)")); BesselI(1/2,-1 - 2*I): evalf(%);
```

$$0.774048495527521777639472674700926328403979483295369690089685 - 0.171623805371532054092681015945273376821203505696119707297606\ \text{I}$$

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

$$0$$

```
>
>
>
>
>
```