Estendere ConTEXt MkIV con PARI-GP

- Lua è un linguaggio di scripting usato per *estendere* un programma
- LuaTEX è un programma (un interprete del linguaggio TEX) che è *estendibile* con Lua
- ConTEXt `MKIV` è un formato per il motore LuaTEX. Le macro sono scritte sia in TEX che in Lua

- È possibile usare librerie (moduli) scritte in Lua oppure librerie scritte in C. In questo caso è necessario scrivere e compilare uno specifico programma in C (il *binding*) che funge da interfaccia tra la libreria C e il linguaggio Lua
- in generale scrivere un binding non è semplice, ma Lua è stato progettato per essere un linguaggio estensibile con librerie C

- SWIG: **S**implified **W**rapper and **I**nterface **G**enerator.

- SWIG è un programma che aiuta lo sviluppatore a costruire il binding di una libreria C/C++. Per alcune librerie è sufficiente avere a disposizione i file *header* (ma è meglio avere tutto il codice sorgente)

- lo sviluppatore elenca in un file *interfaccia* `*.i` i simboli della libreria (costanti, strutture, funzioni...) che si vogliono usare
- `SWIG` legge il file di interfaccia e produce il binding (o wrapper)
- lo sviluppatore compila il codice ed ottiene il modulo Lua

- PARI/GP è un *Computer Algebra System* (CAS)
- è in grado di compiere calcoli numerici e simbolici
- è una libreria C (`libpari`) ed anche un linguaggio interpretato (GP) ~ Lua
- è stabile, ben scritto e documentato, disponibile per diversi OS

Questo è il file di interfaccia pari.i

```
%module pari
%{
#include "pari.h"
ulong overflow;
%}

%ignore gp_variable(char *s);
%ignore setseriesprecision(long n);
%ignore  killfile(pariFILE *f);

%include "pari/paritype.h";    %include "pari/paristio.h";
%include "pari/parisys.h";     %include "pari/paricom.h";
%include "pari/parigen.h";     %include "pari/parierr.h";
%include "pari/paricast.h";    %include "pari/paridecl.h";
%include "pari/paristio.h";    %include "pari/paritune.h";
%include "pari/paricom.h";     %include "pari/pariinl.h";

%inline %{
GEN uti_mael2(GEN m,long x1,long x2)
{return mael2(m,x1,x2);}
```

- per generare binding:

```
swig  -lua pari.i
```

- per compilare il binding e generare il modulo:

```
gcc -ansi \
    -I./pari -I/opt/swig-2.0.2/include \
    -c pari_wrap.c  -o pari_wrap.o

gcc -Wall -ansi -shared -I./pari \
    -I/opt/swig-2.0.2/include -L./ \
    -L/opt/swig-2.0.2/lib pari_wrap.o \
    -lpari -lm -o pari.so
```

- il modulo è caricabile in Lua con `require("pari")` (table `pari`)

Cominciamo con $\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1}$: il codice Lua è

```
\startluacode
require("pari")
pari.pari_init(4000000,500000)
document = document or {}
document.lscarso= document.lscarso or {}
local function sum(X,a,b,expr,start)
   local avma = pari.avma
   local start = start or '0.'
   local res = pari.gp_read_str(string.format(
     "sum(%s=%s,%s,%s,%s)",X,a,b,expr,start))
   res = pari.GENtoTeXstr(res)
   pari.avma = avma
   return res
end
document.lscarso.sum = sum
\stopluacode
```

Il codice TEX è più semplice:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,30,"4*(-1)^k/(2*k+1)","0"))}
\stopformula
\stopTEXpage
\stoptext
```

Il risultato *esatto*:

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = \frac{5863013579100197316985228 4}{1847292006410659792986502 5}$$

Possiamo calcolare un valore approssimato:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,30,"4*(-1)^k/(2*k+1)","0."))}
\stopformula
\stopTEXpage
\stoptext
```

Con una precisione di 28 cifre abbiamo:

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = 3.17384233719074940869022414 0$$

e possiamo anche calcolare la somma simbolica:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{3}\frac{1}{x^2+k}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,3,"1/(x^2+k)","0"))}
\stopformula
\stopTEXpage
\stoptext
```

$$\sum_{k=0}^{3}\frac{1}{x^2+k}=\frac{4x^6+18x^4+22x^2+6}{x^8+6x^6+11x^4+6x^2}$$

Alcuni vantaggi del binding:

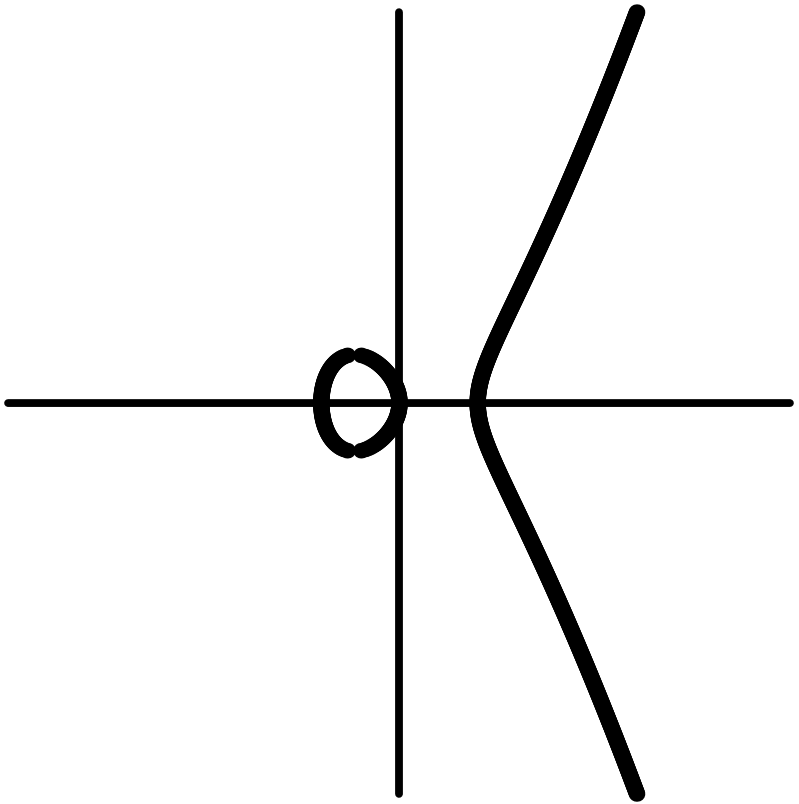
- separazione netta tra codice Lua (logica) e codice ConTEXt MKIV (presentazione) perché ConTEXt MKIV ha un avanzato Lua layer
- reutilizzo codice Lua (in altri contesti)
- reutilizzo di scripts GP (da altri contesti)
- accesso semplice alle funzioni della libreria

Un altro vantaggio di ConTEXt MKIV: integrazione con METAPOST.

Nel prossimo esempio vediamo come tracciare le radici reali di $x^3 - x - y^2$ per $y \in [-5,5]$ con step 2^{-6}.

Nell'articolo è spiegato come implementare la funzione `polroots` (in sostanza è un wrapper della funzione `roots` di `libpari`): il codice mostra un esempio di integrazione di codice Lua e METAPOST in ConTEXt MKIV.

```
\startluacode
local poly = "x^3-x-y^2";local step= 1/2^6
local results = {}      ;local L = 5
local zero = '0.E-96'   ;local prec = 12
get_value = document.lscarso.get_value
polroots  = document.lscarso.polroots
context("\\startMPpage")
context("pickup pencircle scaled 0.1pt;")
context(string.format("draw (-%s,0)--(%s,0);", L, L))
context(string.format("draw (0,-%s)--(0,%s);", L, L))
context("pickup pencircle scaled 0.2pt;")
for y=-L,L,step do
 local poly_x = get_value(poly,'y',y,prec)
 local roots = polroots(poly_x,prec)
 for _,root in pairs(roots) do
  local real,imag = root[1],root[2]
  if imag == zero then
     if real ==  zero then real = '0' end
     context(string.format("draw (%s,%s);",real,y))
  end  end end
context("\\stopMPpage")
\stopluacode
```



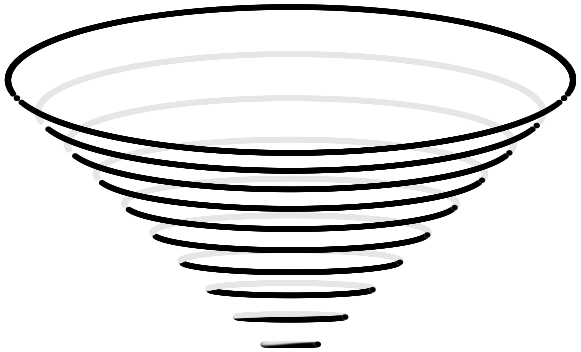
Un altro esempio: utilizziamo una semplice macro METAPOST per proiettare un punto (x,y,z) su un piano:

```
%% http://users.encs.concordia.ca/
%% ~grogono/Writings/mpref.pdf
%%
vardef perspective(expr c) =
  t := dist/(bluepart c + dist);
  (t * redpart c, t * greenpart c)
  enddef;
```

Poi consideriamo $x^2 + y^2 - z^2$ e risolviamo $x^2 + y^2 - z^2 = 0$ dove x e z sono ``parametri''.

I risultati sono in results:

```
\startluacode context("\\startMPpage")
context("vardef perspective(expr c) = t := dist/(bluepart
c + dist); (t * redpart c, t * greenpart c)enddef;")
context("pickup pencircle scaled 0.1pt;")
context("dist:=20;color XYZ;pair XY;")
for i,v in ipairs(results) do
 x,y,z=v[1],v[2],v[3]
 if math.abs(y)>4092 then if y<0 then y=-4091 else y=4091
end end
 if math.abs(y) < 7 then
  context("XYZ := (%.6f,%.6f,%.6f);",x,z,y)
  context("XY := perspective(XYZ+(0,0,0));")
  if y<0 and z<5 then
   context("drawoptions(withcolor 0.9white);")
  else
   context("drawoptions(withcolor 0.0white);")
  end
  context("draw XY;")
 end end
context("\\stopMPpage")\stopluacode
```



Ultimo esempio: risolviamo $(x^2-1)(y^2-1)-(z^2-1)^2=0$, ma questa volta produciamo una pagina METAPOST per ciascun valore dello step z_i

```
\startluacode
for _,table_results_z in ipairs(results_t) do
results,z = table_results_z[1],table_results_z[2]
context("\\startMPpage")
context("pickup pencircle scaled 0.1pt;")
context('label("$z=%.3f$",(10,100)) scaled 0.1;',z)
for i,v in ipairs(results) do
 x,y=v[1],v[2]
 if math.abs(y)>4092 then if y<0 then y=-4091 else y=4091
end end
 if math.abs(y) < 7 then
   context("draw (%.6f,%.6f);",x,y)
 end end
context("\\stopMPpage")
\stopluacode
```

Il pdf ottenuto avrà qualche decina di pagine: se il non contiene troppi ``oggetti'' possiamo trasformarlo in swf con `pdf2swf` (`www.swftools.org`) ed utilizzarlo come se fosse una figura:

```
\externalfigure[pari-polroots-swf.swf][height=0.8\textheight]
```

- PARI/GP per *sperimentare* idee (matematiche) con effettivi algoritmi
- ConTEXt MKIV per *presentare* le idee (formule e risultati)
- METAPOST per *disegnare* relazioni

Conclusione:

- PARI/GP è una libreria ben scritta; costruire il binding è facile — *ma in generale è un compito difficile, anche con SWIG*
- il codice Lua non dipende dalla piattaforma e neanche da ConTEXt MKIV— *ma il binding è specifico per ogni piattaforma*
- è facile da usare per calcoli relativamente semplici — *ma un uso specializzato richiede una adeguata preparazione di teoria dei numeri*

That’s all folks

Thank you !