Extending ConTeXt MkIV with PARI/GP

Luigi Scarso

Abstract

This paper shows how to build a binding to PARI/GP, a well known computer algebra system, for ConTeXt MkIV, showing also some examples on how to solve some basic algebraic problems.

Sommario

Questo articolo mostra come costruire un binding a PARI/GP, un noto sistema di computer algebra, per ConTeXt MkIV; mostra inoltre alcuni esempi di come risolvere alcuni problemi algebrici basilari.

1 Introduction

PARI/GP is a relatively small computer algebra system that comes as C library (libpari) and an interpreter (gp) for its own language (GP) built on upon the same library. Although it has respectable capabilities on symbolic manipulations, it also has an extensive algebraic number theory module, hence it can perform, due to the highly optimised C library, complex numeric calculations very quickly and accurately. PARI stands for “Pascal ARithmétique” (the very first choice was the Pascal language, dropped soon for C), while GP originally was GPC for “Great Programmable Calculator, but also the C was dropped for unknown reasons. The current stable version is 2.3.4.

ConTeXt MkIV is today the most advanced macro format that uses LuaTEX. It’s still under active development: a milestone release is planned for spring 2012, when LuaTeX will reach the 1.0 version (currently is 0.70). ConTeXt MkIV has an integrated support for Lua and MetaPost and offers a superb (as of today) management of OpenType fonts. Its philosophy is different from ConTEX: ConTeXt MkIV is monolithic, while ConTEX is more modular.

In this paper we will show a way to “extend” ConTeXt MkIV with PARI/GP and some examples on how to use this powerful library — hopefully it should be simple to adapt them for luaInteX. We will see how to evaluate summations like \[ \sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} \], calculating the exact value or an approximation; but also we will able to evaluate series like \[ \sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1} \] or symbolic sums as \[ \sum_{k=0}^{\infty} \frac{1}{x^2 + k} \]. PARI/GP is also able to solve equations like \[ x^3 + \arctan (x)x^3 + 2x^2 + x + 1 = 0 \] and we will see how to use it together MetaPost to plot the roots of \( P[X, Y] = X^3 - X - Y^2 \). Finally, the last section is dedicated to show how to solve a simple problem of algebraic geometry using a few PARI/GP functions: the reader is encouraged to implement by himself the code.

2 Build the Lua binding

2.1 What is a binding?

It’s well known that ConTeXt MkIV uses LuaTeX as a typesetting engine, but maybe it’s little known to the TeX-user that Lua by itself is a complete high level programming language. It’s a scripting language, with a syntax quite similar to Pascal, and it’s used either as embedded language to enhance an application (e.g. to build plug-ins) or as glue language to “connect” together several object libraries — exactly the same as in Sage where the glue language is Python. In the latter case Lua is extended with the new libraries that become Lua modules: they can be built in into the Lua interpreter at compile time (as in the GSL Shell program) or loaded at runtime — which is the case of the extension shown in this paper.

Nowadays it’s quite common that a scripting or dynamic language can load at runtime an object library, written in C or C++ and then compiled. Usually it’s necessary to write some C code that act as an interface between the library and the language: this process is called “build the binding for the library”. Here, the developer decides which symbols of the library (i.e. functions, classes, variables and constants) export and how they are seen from the language (under which name, for example). This is a delicate phase, because one must know the conventions of the language, its interface code (the “Application Program Interface” or API of the language), the API of the target library and write the appropriate C code for each symbol to export: for libraries written in C these APIs are usually organised in the header files (with suffix “.h”) that contain the declarations of each function, variable or constant defined in the library — but not always all of them can be exported: the developer must also know the set of useful symbols to export the set of admissible.

It’s clear that each language has its own peculiarities, but many high level languages are implemented in C of C++ and hence shared a common

...
substrate; also, an object library must rigorously follows the conventions of the host Operating System, hence its structure cannot depend to the high level language.

Here is where SWIG enters to play. **SWIG (Simplified Wrapper and Interface Generator, so** [SWIG] is a program that helps the developer to build bindings and, for some libraries, practically can build automatically a binding only by reading all the headers files. SWIG reads a driver file, the so-called **interface file**: it executes its instructions and ends with the binding: most of the time it’s C code that must be compiled into an object module. The power of SWIG is that the user can select the target high level language so that the same interface file can be used for different languages (i.e. Lua, Python, Ruby and so on); on the other side, the code of the binding is more complicated than an binding manually built by a developer.

### 2.2 Build the Lua binding for PARI/GP

The Lua API are completely listed in the Lua book[PL1] and they describe a simple and robust mechanism: basically every C function that wants to interact with the Lua interpreter uses a stack to exchange data. The stack is modified by a relatively small set of functions that act on the Lua state, a global data structure that also keeps track of unused objects and calls the garbage collector when necessary. Hence every C function of the binding must only take care of calling the function of the target library with the right arguments and use the stack to exchange the in (input to the target function) and/or out (output to the Lua interpreter) values.

The interface file pari.i, the object library of PARI/GP, is shown below and its instructions are quite simple: they basically say “read all the headers and produce the binding”. This is for example the role of the `%include "pari/paritype.h"` instruction, that just says “read the header paritype.h which is in the pari folder and write the binding code”, but we can also map some libpari functions into something else, as in

```
GEN uti_mael2(GEN m,long x1,long x2) {return mael2(m,x1,x2);} %
GEN uti_mael5(GEN m,long x1,long x2,long x3,long x4,long x5) {return mael2(m,x1,x2,x3,x4,x5);} %
```

The binding is then built with

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

This is the complete interface file pari.i used under Linux 32 bit: the header files are in the subfolder pari of the folder that contains the build script.

```bash
%module pari
%
#include "pari.h"
ulong overflow;
```
Once compiled, the pari.so is suitable to be loaded as a Lua module with require("pari").

As a final note for this section, the same steps can be followed under Windows using MinGW [MINGW] or with GNU [GUB] to cross-compile the library in a host system (Linux) for a target system (Windows) — as is the case of this paper, where the examples use a cross-compiled dll libpari.

3 Examples

3.1 Summations

As we said briefly in the introduction, PARI/GP has its own language GP, with more than 450 functions, and its interpreter, the gp program. Most of the time these functions are in a one-to-one correspondence with the functions exported by the library libpari, but sometimes there are some “sugar syntactic” constructs for the sake of simplicity. In any case, libpari has the gp_read_str(char *) function that evaluates a GP sentence and returns the result, so that on the Lua side it’s possible to use both the library and the GP language. The library is usually faster than GP and it has a finer control — which usually also means that it’s necessary to write more code.

In this first example, we will see how to calculate exactly a summation. The GP function is sum(X,a,b,expr,start) that stands for

$$\sum_{X=a}^{b} expr(X,\cdot)$$,

where start is the initial value of

$$\sum_{X=a}^{b} expr(X,\cdot)$$:

\begin{verbatim}
\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(X=%s,%s,%s,%s,%s),X,a,b,expr,start())
res = pari.GENtoTexstr(res)
parsi.avma = avma
return res
end
document.lscarso.sum = sum
\stopluacodegen
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = 3.173842337190749408690224140
\endformula
\stoptext
\stop TEXnica N° 11, Aprile 2011 Extending Con\TeX\textsuperscript{e}X MkIV with PARI/GP

that gives

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = 5863013579100197316985228418472920064106597929865025$$

Let’s explain step by step the code. First we need to load the module with require("pari") — assuming that the library is in the standard path or in the current folder (see CLUAINPUTS in \texttt{LUATEX} for more details).

Next, we must avoid conflicts with others Lua functions. A common solution is to define a namespace (document.lscarso in this case), and a local function (sum(X,a,b,expr,start)) to be stated within the namespace (document.lscarso.sum = sum). This is a general issue when one defines its own module, not only for PARI/GP — it’s the same problem of redefining \TeX\textsuperscript{e}X macros.

There is another issue with PARI/GP. Like Lua, PARI/GP also uses a stack but it has not a garbage collector, and every time it makes a calculation the result is not deleted; after a while the stack is full and the process aborts. Fortunately, it’s easy to clear the stack: at the beginning of every function it’s sufficient to record the initial position on the stack with local avma=pari.avma and then reset the stack with pari.avma=avma just before the return statement of the function. This is an issue with libpari, because most of GP functions manage the stack correctly.

After these notes, calling the GP sum function is a matter of calling gp_read_str(char *) with the right formatted string, which is trivial thanks to string.format, a standard Lua\TeX\textsuperscript{e}X function. Last but not least is pari.GENtoTexstr(GEN), a libpari function that translates a pari object (e.g. a fraction) into a \TeX\textsuperscript{e}X expression. It’s important to note that the result is exact because we have imposed with start=0 that all the values are in \texttt{Q}: if we want an approximated value we just use start=0. (0. means “zero” as floating point value) and the result is

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = 3.173842337190749408690224140$$

But we can do things a bit better. First, we want to control the precision of the result, i.e. how many digits to show. This is quite simple: the GP default(\ldots) function can be used to get/set some internal constants and realprecision is what we need:

local function set_precision(prec)
local avma = pari.avma
local prec = math.floor(prec+0.5) or 28
local res = pari.gp_read_str(
    string.format("default(realprecision,%s")
prec)
pari.avma = avma
return res
end

local function get_precision(prec)
local avma = pari.avma
local res = pari.GENtostr(pari.gp_read_str(
    "default(realprecision)"))
pari.avma = avma
return res
end

Once we define the precision, we can extend the summation to “infinity”, i.e. until the partial sums are stable within the precision. Of course this depends on the character of the series — in our case it’s an alternating series. For this kind of series GP has the \texttt{sumalt(X=a,expr)} function that does the job:

local function sum_alternate(X,a,expr,prec)
local avma = pari.avma
local gp = document.lscarso.get_precision
local oldprec = gp(prec)
document.lscarso.set_precision(prec)
local res=pari.GENtostr(pari.gp_read_str(
    string.format("sumalt(%s=%s,%s")
    X,a,expr))
document.lscarso.set_precision(oldprec)
pari.avma = avma
res=string.gsub(res,"(%d)\","%1\hskip0sp")
return res
end

We can hence try to calculate the series with a precision of 800 digits:

\[
\sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1} \approx \frac{\pi}{4}.
\]

Given that the result is quite long (see fig.\ref{fig:evaluation}) we insert an invisible skip to help \TeX to break the expression.

Of course this is a well known series: from \[\arctan(1) = \frac{\pi}{4}\] one can calculate the Taylor expansion of \[\arctan(x)\] around \[x = 0\] with \texttt{taylor(expr,x)}:

local function taylor(expr,x)
local avma = pari.avma
local res = pari.gp_read_str(
    string.format("taylor(%s,%s")
    expr,x))
res = pari.GENtoTeXstr(res)
pari.avma = avma
return res
end

\[\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \frac{1}{9}x^9 - \frac{1}{11}x^{11} + \frac{1}{13}x^{13} - \frac{1}{15}x^{15} + O(x^{16})\]

The series is convergent in \[x = 1\] (there are several proofs about this, e.g. see \cite{Leibniz}), and the result is exactly \[\arctan(1)\], hence

\[
4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1} = 4 \frac{\pi}{4} = \pi .
\]

It’s important to note that theoretically this series has a slow convergence to \(\pi\) (it’s hence a bad choice to calculate \(\pi\)) but in practice it can be used with \texttt{PARI/GP} to give quickly an high precision result — this is the power of the library.

Before going further, let’s consider again this summation:

\[
\sum_{k=0}^{\infty} \frac{(-1)^k}{x^2+k} = \sum_{k=0}^{\infty} (\arctan(1)x^{2k})
\]

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Evaluation of an alternating series with 800 digit precision.}
\end{figure}

\cite{Leibniz}
that gives
\[ \sum_{k=0}^{3} \frac{1}{x^2 + k} = \frac{4x^6 + 18x^4 + 22x^2 + 6}{x^3 + 6x^6 + 11x^4 + 6x^2} \]

PARI/GP is also capable of some symbolic calculations — it’s not only a numeric library.

3.2 Continued fractions

A simple finite (canonical) continued fraction is a rational number \( q \) given by
\[ q = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \ddots + \frac{1}{a_n}}} \]

where \( a_0 \) is an integer and \( a_j, j > 0 \) are strictly positive integers. Every rational number can be expressed with a finite continued fraction; if we consider a succession of finite continued fractions, for \( n \to \infty \) we have an infinite (canonical) continued fraction, and every irrational number has an unique infinite continued fraction. For a finite c.f. \([a_0, a_1, a_2, \ldots, a_n]\) the rational number given by calculating all the intermediate fractions is usually termed as \( p_n/q_n \), i.e.
\[ [a_0, a_1, a_2, \ldots, a_n] = \frac{p_n}{q_n} \]

For example \([0, 3] = 1/3\) and it’s possible to show that \( p_n/q_n = [a_0, a_1, a_2, \ldots, a_n] \) is the fraction in lowest terms. The GF \texttt{contfrac} function calculate (the vector of) the continued fraction of a rational number, while \texttt{contfracpnq}, given a (finite vector of) continued fraction, return \( p_n, q_n \). But the interesting point here is to use, given a real number with a fixed precision, the continued fraction to find its best rational approximation. The \texttt{libpari\_bestappr(x,A)} function calculates exactly what we need:

```lua
local function bestappr(x,A)
    local avma = pari.avma
    local x = tostring(x) or nil
    local A = math.floor(A+0.5)
    local res, bestx
    if x == nil then return nil,nil end
    bestx = pari.bestappr(pari.geval(pari.strtoGENstr(x)),
        pari.geval(pari.strtoGENstr(tostring(A))))
    res = {}
    res[1] = pari.GENtostr(bestx)
    res[2] = pari.GENtostr(pari.uti_gel(bestx,1))
    res[3] = pari.GENtostr(pari.uti_gel(bestx,2))
    pari.avma = avma
    return res[1],res[2],res[3]
end
```

Note that the return value is an array with 3 components, namely \( p_n/q_n \). We also use \texttt{pari.uti\_gel}, the \textit{wrapped} version of \texttt{libpari\_gel} function, to access an array by components.

Instead of an arbitrary real number, we choose \( \pi \) because the \texttt{libpari\_mppi(long)} function gives \( \pi \) with the required precision.

```lua
\startluacode
local collect = {}
local avma = pari.avma
local prec = 800
document.lscarso.set_precision(prec)
local pi = pari.mppi(prec)
local pi_str = pari.GENtostr(pi)
pari.avma = avma
--print("=====>pi:",pi_str)
for d = 4,50000,1 do
    res,num,den = document.lscarso.bestappr(pi_str,d)
    collect[res] = {num,den,d}
end
context("\starttabulate[|l|l|]"
context("\HL"
context(string.format("\NC %s \NC %s \NC\NR",
    "fraction","approx. value"))
context("\HL")
context(string.format("\NC %s \NC %s \NC\NR",
    "\text\WC %s \text\WC %s \text\WC\NR",
    "fraction","approx. value"))
context("\H")
for k,v in pairs(collect) do
    print(k, v[1]/v[2],v[3])
    context(k, v[1]/v[2])
end
context("\stoptabulate")
\stopluacode
```

Note that we use \( p_n, q_n \) as a key for the dictionary \texttt{collect}, so we have just the set of results — i.e. we drop the same best approximations for different denominators. For a precision of 800 digits and a range of denominators between 4 and 5000 we have hence:

<table>
<thead>
<tr>
<th>fraction</th>
<th>approx. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>333/106</td>
<td>3.1415094339623</td>
</tr>
<tr>
<td>104348/33215</td>
<td>3.1415926539214</td>
</tr>
<tr>
<td>16/5</td>
<td>3.2</td>
</tr>
<tr>
<td>13/4</td>
<td>3.25</td>
</tr>
<tr>
<td>22/7</td>
<td>3.1428571428571</td>
</tr>
<tr>
<td>355/113</td>
<td>3.141592920354</td>
</tr>
<tr>
<td>19/6</td>
<td>3.1666666666667</td>
</tr>
<tr>
<td>103993/33102</td>
<td>3.1415926530119</td>
</tr>
</tbody>
</table>

where the approximate values are due to the Lua floating point math.
3.3 Equations

Solving numeric equations in PARI/GP require more attention than other packages. The GP solve(X=a,b,expr) functions implements a very good algorithm but it works with one variable only and it fails if expr is not defined in [a,b] and it hasn’t a variation in [a,b]. This Lua wrapper solve tries to ensure that at in [a,b] there is a variation, by evaluating the sign of expr(a)*expr(b):

```latex
function solve(expr,X,a,b,prec)
local av = pari.avma
pari.gp_read_str(string.format("default(realprecision,%s)",prec))
local tr,res
pari.gp_read_str(string.format("f(%s)=%s",X,expr))
tr = pari.gp_read_str(string.format("if(f(%s)*f(%s)<0,1,0)",a,b))
tr = tonumber(tr)
res = nil
if (tr==1) then
  local expr=string.format("solve(%s=%s,%s,%s)",X,a,b,expr)
  res = pari.gp_read_str(expr)
  res = pari.GENtostr(res)
end
return res,
pari.GENtoTeXstr(pari.strtoGENstr(expr))
end
```

The following code tries to solve

\[ x^5 + x^3 \arctan(x) + 2 \cdot x^2 + x + 1 \approx 0 \]

for \( x \in [-100,100] \) with a precision of 12 digits:

```latex
\startluacode
local solve = document.lscarso.solve
for a=-100,99,1 do
  local res,TeX,aa,bb =
    solve('x^5+atan(x)*x^3+2*x^2+x+1','x',a,(a+1),12)
  if res ~= nil then
    context(string.format("$%s\approx 0$ \crlf for $x\approx %s$\par","TeX",res))
  end
  pari.avma = avma
  return res
end
\stopluacode
```

We have hence:

\[ x^5 + \arctan(x) \cdot x^3 + 2 \cdot x^2 + x + 1 \approx 0 \]

for \( x \approx -1.47704735548 \)

PARI/GP has a rich set of functions for polynomials, and solve is not necessarily the best choice to find the roots of multivariate polynomials. The next and last example will show how to draw the real roots of \( P[X,Y] \) with a given precision in a square region \( [a,b] \times [a,b] \). First of all, we need to understand that with a fixed precision there is also an associated zero: with precision=12 then zero=1E-96. Next, PARI/GP finds the complex roots of a univariate polynomial, so we need a get_value wrapper to evaluate \( P(x,y) \) for \( y \in [a,b] \) (with a given precision), so we have an expression in the \( x \) indeterminate that we will consider as a polynomial \( P[X] \):

```latex
\startluacode
local poly = "x^3-x-y^2"
local step= 1/2^6
local results = {}
local limit = 5
local zero = '0.E-96'
local prec = 12
get_value = document.lscarso.get_value
polroots = document.lscarso.polroots
\startMPpage...
\stopMPpage
```

We have hence:

\[ x^3 - x - y^2 \approx 0 \]

for \( x \approx -1.47704735548 \)

Finally, we need to iterate \( y \) over \([a,b]\) and find the roots of \( P[X] \). Instead of producing a table, we plot the values with the help of a MetaPost
\startMPpage...\stopMPpage environment:

```latex
\startluacode
local poly = "x^3-x-y^2"
local step= 1/2^6
local results = {}
local limit = 5
local zero = '0.E-96'
local prec = 12
get_value = document.lscarso.get_value
polroots = document.lscarso.polroots
\stopluacode
```
With a precision of 12 digits and a square region of \([-5,5]\) we have then:

\[
(x, y) \subset k \times k \text{ where }
\]

\[
x = (1 - t)^3 P_x + 3(1 - t)^2 t C_{1x} + 3(1 - t) t^2 C_{2x} + t^3 Q_x
\]

\[
y = (1 - t)^3 P_y + 3(1 - t)^2 t C_{1y} + 3(1 - t) t^2 C_{2y} + t^3 Q_y
\]

and \(t \in [0,1] \subset k\). The points \(P, C_1, C_2, Q\) are called control points; \(P\) is the starting point \((t = 0)\), \(Q\) is the ending point \((t = 1)\). MetaPost has a built-in macro to draw cubic Bezier curves: for example this code

```plaintext
\starttext
\startMPpage
pair P,C[],Q; P:=(0,0); C[1]:=(20,1); C[2]:(30,15); Q:=(50,7);
draw P..controls C[1] and C[2]..Q;
\stopMPpage
\stopetext
```

draws:

![Bezier curve](image)

A nice property is that the derivative of the curve at \(P\) and \(Q\) are \(3(C_1 - P)\) and \(3(Q - C_2)\), as we can qualitatively see in the previous picture.

The next definition is the ring of polynomials in the two indeterminates \(k[X,Y]\). Informally, it’s the set of the finite expressions \(P[X,Y]\) where

\[
P[X,Y] = \sum_{i,j} a_{ij} X^i Y^j
\]

and the total degree of \(P[X,Y]\) is \(\max(i+j)\), \(a_{ij} \neq 0\) and if all \(a_{ij}\) are zero then we have the zero polynomial \(0\) (the zero of the ring \(k[X,Y]\)) which has degree undefined (but many authors set it to \(-1\) or \(-\infty\)). Two polynomials \(P_1[X,Y]\) and \(P_2[X,Y]\) are identical if and only if \(P_1[X,Y] - P_2[X,Y] = 0\); two polynomials with different degree are never identical.

Given \((x, y) \in k \times k\) and a polynomial \(P[X,Y]\), with an abuse of notation we can consider the function \(P(x, y) : k \times k \rightarrow k\) where we “replace” \(X\) with \(x\) and \(Y\) with \(y\) in \(P[X,Y]\) and then calculate the value \((P(x,y)\) it’s an expression of \(k\).

Now, a root of \(P[X,Y]\) is a point \((x, y) \in k \times k\) for which \(P(x, y) = 0\); given a \(P[X,Y]\), the set of its roots is the curve of the polynomial and describe the properties of a curve is the main scope of the algebraic geometry (of course not only with two indeterminates but not only with \(k = \mathbb{Q}\) or \(k = \mathbb{R}\)). In general it’s a difficult task, but anyway we can already say something:

1. \(P[X,Y] = X^2 + 1\) has no roots in \(k \times k\) (remember that \(k = \mathbb{Q}\) or \(k = \mathbb{R}\));
2. \(P[X,Y] = X\) has infinite roots because for every \(y \in k\) \(P(0,y) = 0\);
3. \( P[X, Y] = 0 \) (the zero polynomial) has infinite roots: for every point of \((x, y) \in k \times k \) \( P(x, y) = 0 \), so in this case the set of roots is \( k \times k \).

Hence the curve of \( P[X, Y] \) can be empty, infinite but properly included in \( k \times k \), or all \( k \times k \). Of great interest are also the set of intersections of two curves, which can be empty (i.e. \( P_1[X, Y] = Y - 2 \) and \( P_2[X, Y] = Y \)), finite (\( P_1[X, Y] = X - Y \) and \( P_2[X, Y] = X + Y \)) and infinite (\( P_1[X, Y] = 0 \) and \( P_2[X, Y] = Y \)).

Let’s consider \( P[X, Y] = X^2 + Y^2 - 1 \). The “brute force” approach to have an idea of its curve \( C \) is, given a tolerance \( \epsilon \) and a limit \( R \), verify whenever \( [x^2 + y^2 - 1] < \epsilon \) for \((x, y) \in [-R, R] \times [-R, R] \); a bit less primitive way is to solve \( x^2 + y^2 - 1 = 0 \) for \( y \), with a fixed tolerance \( \epsilon \) and the parameter \( x_0 \in [-R, R] \). Both methods was shown in the previous sections, so here we take another way. Let’s consider the set \( P_1[X, Y] = Y - tX \) indexed by \( t \); it’s trivial to show that its curves are the lines \( y = tx \). The intersection between \( P(x, y) = 0 \) and \( P_1(x,y) = 0 \) is also easy to calculate: just replace \( y^2 \) with \((tx)^2 \). We have hence \( x^2 + (tx)^2 - 1 = 0 \) and

\[
C = \begin{cases} 
  x^2 = \frac{1}{1 + t^2} \\
  y^2 = \frac{t^2}{1 + t^2}
\end{cases}
\]

or, after few simplifications,

\[
C = \begin{cases} 
  x = \pm \frac{1}{\sqrt{1 + t^2}} \\
  y = \pm \frac{t}{\sqrt{1 + t^2}}
\end{cases}
\]

where \( t \) runs over \( k \). This is called a parametric form of \( C \), while \( x^2 + y^2 - 1 = 0 \) is called the implicit form of the curve (the curve defined implicitly by \( X^2 + Y^2 - 1 \)). Although it was easy to obtain, it’s not a nice formula, due to the presence of the square root factor; maybe we cannot avoid the fractions, but we would like to have integer exponents. In fact, in \( k = \mathbb{Q}, \sqrt{1 + t^2} \) not always is rational: for example for \( t = 1/3 \) we have the \( \sqrt{10} \) factor which is not rational.

We can now come back to our problem, and make some assumptions. First, we will consider a quarter of a circumference \( C \) with \( x \geq 0 \) and \( y \geq 0 \) and radius 1. Next, we will assume that MetaPost is able to draw a circumference, that we will use as a reference. In this case, the best candidates seem to be the cubics with control points \( \{P, C_1, C_2, Q\} = \{(0, 0), (a, 0), (0, 0), (0, 1)\} \) where \( a \in [0,1] \) and hence we can draw these cubics for some values of \( a \), just to see how they look like.

After few tries we can discover that a good approximation can be reach with \( a = 0.55 \), so we have some hints for a solution; but to better understand the problem we need to find the implicit form of a Bezier curve, at least for those ones that we are studying. Let’s rewrite the parametric form:

\[
C_a = \begin{cases} 
  x = (1 - t)^3 + 3(1 - t)^2t + 3(1 - t)t^2 a \\
  y = 3(1 - t)^2ta + 3(1 - t)t^2 + t^3
\end{cases}
\]

First let’s note that for \( t = 1/2 \) we have

\[
\text{subst}((1-t)^3 + 3(1-t)^2t + 3(1-t)t^2, 3(1-t)^2ta + 3(1-t)t^2 + t^3, t, 1/2) = \{3/8a + 1/2, 3/8a + 1/2\}
\]

i.e. the middle points of the curves are along the line \( y = x \).

Next let’s introduce the hybrid form

\[
\begin{cases} 
  -X + (1 - t)^3 + 3(1 - t)^2t + 3(1 - t)t^2 a \\
  -Y + 3(1 - t)^2ta + 3(1 - t)t^2 + t^3
\end{cases}
\]

We would like, if possible, to eliminate the parameter \( t \) from the hybrid form and obtain a polynomial \( P_a[X, Y] \) indexed by \( a \). The PARI-GP function \texttt{polresultant} is what we need:

\[
\text{PARY=polresultant(}\
  -X + (1-t)^3+3*(1-t)^2*t+3*(1-t)*t^2*a,\
  -Y + 3*(1-t)^2*t*a+3*(1-t)*t^2+3*t^3, t; 1/2)\
\]

give us

\[
P_a[X, Y] = (27a^3 - 54a^2 + 36a - 8)X^3 + ((81Y - 108)a^3 + (-162Y + 207)a^2 + (108Y - 126)a + (-24Y + 24))X^2 + ((-81Y + 81)a^3 + (-81Y^2 - 162Y + 81)a^3 + (-162Y^2 + 414Y - 252)a^2 + (108Y^2 - 252Y + 144)a + (-24Y^2 + 48Y - 24))X + ((81Y - 81)a^4 + (27Y^3 - 108Y^2 + 81Y)a^3 + ((-54Y^3 + 207Y^2 - 252Y + 99)a^2 + (36Y^3 - 126Y^2 + 144Y - 54)a + (-8Y^3 + 24Y^2 - 24Y + 8))
\]

Now, if \( P_a[X, Y] \) has degree 3 then it cannot be identical to \( X^2 + Y^2 - 1 \), and each approximation
will never be perfect (always remember that $k = \mathbb{Q}$ or $k = \mathbb{R}$ and that they have infinite elements). If we calculate the roots of the coefficient of $X^3$ with

\[
\text{factor}(27*\text{a}^3 - 54*\text{a}^2 + 36*\text{a} - 8)
\]

we have

\[
[3\text{a} - 2 \ 3]
\]

i.e $a_0 = 2/3$ is a root of order 3. So, maybe for $a_0 = 2/3$ we have a chance that $P_{a_0}[X,Y]$ has degree 2. Let’s substitute $a$ with 2/3 in $P_a[X,Y]$:

\[
\text{subst}(\text{PaXY},a,2/3)
\]

The result is:

\[
0
\]

This suggests that the previous parametrization is not valid if $a = 2/3$: in fact if we calculate $\text{polresultant}$ with $a = 2/3$ we have

\[
P_{23XY}=\text{polresultant}(-X + (1-t)^2*3*(1-t)*t^2*(2/3),
-\ Y + 3*(1-t)^2+t^2*(2/3)*3*(1-t)*t^2+3,t) = X^2 + (-2t + 2)*X + (Y^2 + 2t - 3)
\]

We should not be surprised: after all the curve with $a = 2/3$ is showed in the picture so we already know that it is a curve; only it has order 2:

\[
[(1-t)^2*3*(1-t)*t^2*(2/3),
3*(1-t)^2+3*(1-t)*t^2+3]
\]

\[
= [-t^2 + 1, -t^2 + 2t]
\]

i.e. $x = 1 - t^2$, $y = 2t - t^2$, $t \in [0,1]$ (actually, the picture shows that a cubic Bezier curve can be a segment ($a = 0$), a curve of second order ($a = 2/3$) or a curve of third order). But anyway again $X^2 + (-2t + 2)*X + (Y^2 + 2t - 3) - X^2 - Y^2 + 1$ is not the zero polynomial.

The conclusion then seems to be: $P_a[X,Y]$, $a \in [0,1]$ can never be equal to $X^2 + Y^2 - 1$, or with a Bezier curve we can never draw a circumference.

But we must consider that it depends from the result of the parametrization, and we have not proved that this is unique — or that one from $\text{polresultant}$ is exact.

But let’s go on. We have already seen that for $t = 1/2$ each curve has $x = y$; another idea from the picture comes if we consider the intersection of the circumference with the line $y = x$, i.e. $(x,y) = (1/\sqrt{2},1/\sqrt{2})$, and impose that our Bezier curves pass through $(1/\sqrt{2},1/\sqrt{2})$. This is because the curves in $P$ and $Q$ are already a good approximation of the circumference (they have similar derivatives) and they evolve smoothly, so if we choose the middle point as another constrain things maybe are better. Theoretically we should go from $P_a[X,Y]$ to $P_a(x,y) = 0$, substitute $y$ with $x$, substitute $x$ with $1/\sqrt{2}$ and calculate the roots of $P_a(1/\sqrt{2},1/\sqrt{2}) = 0$, but we can collect all these steps with

\[
\text{factor}(\text{subst}(\text{PaXY},Y,X,1/\sqrt{2}))
\]

that gives

\[
[a + 0.471404520791048616383139538 \ 1]
[a + 0.4714046207910149042294785297 \ 1]
\]

\[
[a - 0.5522847498307933984022516309 \ 1]
[a - 0.6666666666666666666666666666685 \ 1]
\]

and we find that, for $a \in [0,1]$ and $a \neq 2/3$,

\[
a_0 = 0.5522847498307933984022516309
\]

is our only solution.

Now, for $t = 1/2$ the curves $P_{a_0}$ must pass in $x = y = 1/\sqrt{2}$, hence, from the parametric form,

\[
3\ a + 2 \ 1
\]

which have a unique solution $a_0$

\[
a_0 = 4\ \sqrt{2} - 1 \ 3
\]

It has a nice look (sometimes it is called the “1-2-3-4” formula): its value is

\[
4*(\sqrt{2})-1/3
= 0.5522847498307933984022516323
\]

and it is a root of $P_a(x,y)$:

\[
\text{subst}(\text{polresultant}(\text{PaXY},Y,X,1/\sqrt{2}), a, 4*(\sqrt{2})-1/3)
= 1.186446801E-27
\]

Hence, whether we follow the implicit way or the parametric way the conclusion is the same: there is a unique Bezier curve that is an optimal approximation of a circumference. The parametric way give us a nice formula for the curve, the implicit way tell us that there are no Bezier curves that are circumferences.

Now, the last step. We would like to have an estimation of the error, i.e. how much our optimal cubic differs from the circumference. We can consider the distance between two points $P_1 = \{y = tx \cap C \} \cap P_2 = \{y = tx \cap P_{a_0}(x,y) \} \cap C$ and take the max of this distance as an indicator of the error. Recalling that $C$ has radius 1, it’s easy to show that $P_1P_2 = P_2 - P_1 = P_2 - 1$ and the relative percentage error is

\[
e_r = \frac{P_1P_2}{P_2} = 100 = 100 \left( \frac{P_2 - 1}{} \right)
\]

The following code print the error $e_r$ vs. the angle $\phi$, where $y = x\tan(\phi)$, $0^\circ \leq \phi < 90^\circ$ , and the max($e_r$):

\[
B=\text{subst}(\text{polresultant}(\text{PaXY},a, 4*(\sqrt{2})-1/3),Y,m*X);
\]

\[
\text{MAX}=100;
\]

\[
\text{forstep}(f=0.09,1,\text{m}=\text{atan}(Pf/180*\text{f}));
\]

\[
\text{roots}=\text{polroots}(\text{B},m,\text{m}*)
\]

\[
\text{forstep}(i=1,3,1,
\text{xf}=\text{real}(\text{roots}[i]);
\text{yf}=\text{imag}(\text{roots}[i]);
\text{if}(\text{xf}<1.0 \&\& \text{xf}>-0.0 \&\& \text{iy}<1e-12,
\text{yf}=\text{mf}+\text{Pf};
\text{Pf}=\text{sqrt}(\text{xf}^2+\text{yf}^2);)
\]

\[
\text{print}(f,,(\text{Pf}-1)*100,,\text{MAX}))
\]

The error is $e_r = 0.0272\%$, and confirms that $a_0$ is a good choice.
3.4.1 Some notes

A different parametrization of the circumference is given by the intersection of the line that join (0,1) and (r,0), \( y = -\frac{1}{r} x + 1 \), with \( x^2 + y^2 - 1 = 0 \):

\[
C = \begin{cases} 
  x = \frac{2r}{1 + r^2} \\
  y = \frac{r^2 - 1}{1 + r^2}
\end{cases}
\]

Both \( x \) and \( y \) are rational if \( r \) is rational, so this shows that there are infinite rational points on a circumference.

Another question is when \( \sqrt{1 + t^2} \) is rational if \( t \) is rational. Let \( t = \frac{t_1}{t_2}, t_1 \geq 0, t_2 \geq 0 \): then

\[
\sqrt{1 + \left(\frac{t_1}{t_2}\right)^2} = \frac{n_1}{n_2} \iff \sqrt{t_1^2 + t_2^2} = t_2 \frac{n_1}{n_2}
\]

i.e. the problem is to find \( t_1 \) and \( t_2 \) so that \( t_1^2 + t_2^2 = t_3^2 \) is an integer, or \( t_1^2 + t_2^2 = t_3^2 \) with \( t_3 \) integer. There is an easy “3-4-5” formula to start with: \( 3^2 + 4^2 = 5^2 \). If we rewrite it as \( 4^2 = 5^2 - 3^2 \) or \( 4 \cdot 4 = (5 - 3)(5 + 3) = 2 \cdot 8 \), we see that 4 divides \( 5 + 3 \), so \( t_1 t_1 = (t_3 - t_2)(t_3 + t_2) \) and \( t_1 \) divides \( t_3 + t_2 \) is true at least for the triple \( (t_1,t_2,t_3) = (3,4,5) \). Let’s then suppose that \( t_1 \) divides \( t_3 + t_2 \), i.e. \( t_3 + t_2 = k_1 t_1 \): with this condition we have

\[
\begin{cases} 
  t_1 = (t_3 - t_2) k_1 \\
  k_1 t_1 = (t_3 + t_2)
\end{cases}
\]

and hence

\[
t_3 = \frac{t_1 k_1^2 + 1}{2 k_1}
\]

that is an integer if \( t_1 = l_1 k_1 \) and \( l_1 (k_1^2 + 1) \) is even, with \( l_1 \) integer.

The last note is about the field \( k \). We have used \( k = \mathbb{Q} \) or \( k = \mathbb{R} \) were the distinction between \( P[X,Y] \) and \( p(x, y) \) seem to be artificial, but there are fields \( k \) where distinction matter. For example the classes of residues modulo \( p \) with \( p \) prime is a field \( \mathbb{F}_p \) with the usual operation “+, x” as in \( \mathbb{Q} \) or \( \mathbb{R} \). Let’s consider \( \mathbb{F}_5 \): denote each class of residue with \( \{(0, 1, 2, 3, 4) \} \) (hence \( 1 \cdot 3 = 22 = 2 \) and \( 4 + 3 = 7 = 2 \)).

The polynomial \( P[X] = X(X - 1)(X - 2)(X - 3)(X - 4) \) is \( 5^6 \) it’s not the zero polynomial of \( \mathbb{F}_5 \), but \( P(x) = 0 \) for every element in \( \mathbb{F}_5 \).

4 Conclusion

One of the main advantages of ConTeXt MkIV is the clear separation between Lua code and TeX code, and in this case it’s a very useful thing that we can import a pari-lua script into ConTeXt MkIV without too much work to adapt it to the ConTeXt MkIV machinery — i.e. we have an high degree of code reuse. PARi/GP has also a nice TeX formatter, even if in some situations things are a bit crude.

On the other side, solving numerical problems always requires some amount of theoretical analysis before doing the computation, as in the case of solve — in some circumstances PARi/GP abruptly aborts if it finds an error. Some computations can require a long time to finish, and given that ConTeXt MkIV is a multi pass system a caching mechanism should be provided to solve these situations.

Numeric results can (but they shouldn’t) depend on the compiler and/or platform, but it seems that from this point of view that PARi/GP is really platform-independent.

References

URL http://pari.math.u-bordeaux.fr

URL http://sagemath.org

URL http://www.nongnu.org/gsl-shell

URL http://www.inf.puc-rio.br/~roberto/pil2

URL http://swig.org

URL http://www.mingw.org

URL http://www.lilypond.org/gub


URL http://en.wikipedia.org/wiki/Leibniz_formula_for_pi

Luigi Scarso
luigi.scarso at gmail dot com