A browser interface to the Giac/Xcas CAS

Bernard Parisse
(Université de Grenoble I)

bernard.parisse@univ-grenoble-alpes.fr

Introduction

Giac is a general purpose computer algebra system library written in C++. Xcas is a user interface to Giac, it is popular in French education but Giac/Xcas is not very well known outside France. Giac covers most symbolic computations from high-school to university levels, and has good performance for fast multivariate polynomial arithmetic (including Groebner basis computations) and linear algebra.

Many CAS have a small kernel and a large math library written in a user language. Giac implements a user language (with syntatic sugar for people used to Maple, Python, or TI CAS calculators), but all built-in commands of Giac are implemented in C++ and can be interfaced to all languages interfacing to C++ (or even C, interacting with the CAS kernel using C strings `char *`). This makes interfacing with other projects easy, and Giac is already interfaced with some codes (free and commercial):

- Xcas (GPL): it is the user interface I developed for Giac,
- Geogebra (GPL): Giac is the math kernel for the CAS window,
- the HP Prime calculator is running Giac as CAS,
- some applications, like PocketCAS on iOS (commercial), are using Giac for their math kernel.

I will discuss here a new Javascript interface to Giac that can be used as an alternative to a classic CAS. It is the math kernel of Xcas for Firefox, a CAS GUI for desktops and mobile devices, and may also be used in LaTeX documents to produce interactive math-enabled documents. The main difference with many web-based CAS user interfaces is that the computations are done *locally* by the web browser instead of requiring net access for each computation (sending the computation to a server and getting back the answer). It is therefore possible to install Xcas for Firefox on a mobile device and run it in airplane mode – this might be an alternative to calculators in education.

Xcas for Firefox

You can test Xcas for Firefox at
`www-fourier.ujf-grenoble.fr/~parisse/xcasen.html`
The Math button opens wizards for common CAS operations. For programming use Prog. Click on Doc, then Examples to see some examples of sessions.

Differences with native application

The feature set of Xcas for Firefox is not as complete as the Xcas native application feature set, for example it is not possible to speed up parallelizable algorithms (like modular determinant, modular Groebner basis, ...) on multi-core architectures. This does not impact educational use.

On the other hand, Xcas for Firefox is much more transparent to the user, because it does not require installation. Exchanging a worksheet is as easy as writing an email, because a worksheet can be encapsulated in an HTML link along with metadata like the e-mail address of the sender. Clicking on the Mail link of Xcas for Firefox will therefore automatically open the email client of the user with the e-mail address filled, either with the sender e-mail or with the default from Settings.

Technical details and performances

The C++ CAS code is compiled to Javascript (instead of machine language) using emscripten, then the user-interface code written in Javascript interacts with a Javascript entry point to the C++ code using strings. More precisely, the C-like Giac function

```
const char * caseval(const char *s)
```

is exported with a compile flag

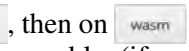
```
-s EXPORTED_FUNCTIONS="['_caseval']"
```

and is made callable from Javascript with

```
var docaseval = Module.cwrap(
  'caseval','string',['string']);
```

We will see that the performance of floating-point operations (with `double`) is not far from a natively compiled single-threaded application, thanks to asmjs, a subset of Javascript that the browser can translate to native code on the fly. Unfortunately, Javascript does not provide a type for 64-bit integers, which is a much bigger performance hit for exact computations, especially for operations like Groebner-basis computations (for example, cyclic7 takes more than 15 seconds instead of about 2 s natively). Recent versions of desktop browsers implement a new standard named web-assembly, and there the performance loss is not as severe, Groebner-basis computations like cyclic7 are 2 to 3 times slower than single-threaded native programs, and this is acceptable unless you run really large computations.

Following are a few benchmarks with Xcas 1.4.9-53 and Firefox 58 on a Mac (Core i5) using the interface at `www-fourier.ujf-grenoble.fr/~parisse/xcasen.html`

Click on the Settings button  , then on  to switch between asmjs and web-assembly (if your browser supports it).

Approx. linear algebra benchmarks

LU decomposition:

```
n:=500;
m:=ranm(n,n,uniformd,-1,1):;
time(p,l,u:=lu(m));
maxnorm(l*u-permu2mat(p)*m)
```

native: 0.05s, wasm: 0.045s, asmjs: 0.055s

Giac built-in QR decomposition:

```
n:=500;
m:=ranm(n,n,uniformd,-1,1):;
time(q,r:=qr(m,-1));
maxnorm(m-q*r);
```

native: 0.08s, wasm: 0.09s, asmjs: 0.1s

Schur decomposition:

```
n:=500;
m:=ranm(n,n,uniformd,-1,1):;
time(p,q:=schur(m));
maxnorm(m-p*q*trn(p));}
```

native: 0.45s, wasm: 0.5s , asmjs: 0.5s

Exact benchmarks from the Nemo test suite

Series expansion:

```
n:=200; series("t");
u:= t + O(t^n);
time(r:=(u/(exp(u)-1))*exp(x*u));
```

native: 0.6s, wasm: 3.6s, asmjs: 2.9s.

Power of a polynomial with coefficient in a field extension of $\mathbb{Q}$:

```
x:=rootof(cyclotomic(20));
f:=(3x^7 + x^4 - 3x + 1)*y^3 +
   (2x^6-x^5+4x^4-x^3+x^2-1)*y +
   (-3x^7+2x^6-x^5+3x^3-2x^2+x);
p:=symb2poly(f,y,[]);
time(s:=p^800);
```

native: 0.3s, wasm: 1.9s, asmjs: 1.7s

Determinant of a matrix with coefficients in a field extension of $\mathbb{Q}$:

```
purge(x);
n:=80;
a:=rootof(x^3+3x+1);
m:=ranm(n,n,a):;
time(d:=det(m));
```

native: 6s, wasm: 24s, asmjs: 24s

Determinant of a matrix with coefficients in a field extension of $\mathbb{Z}/p\mathbb{Z}$:

```
p := 2003*1009; n:=40;
f(j,k):={
  k:=rand(6);
  return randpoly(x,k) mod p;
};
m:=matrix(n,n,f):;
time(det(m));
```

native: 0.45s, wasm: 0.4s, asmjs: 0.7s

Characteristic polynomial of a random matrix with integer coefficients:

```
n:=100; m:=ranm(n,n,-21):;
time(charpoly(m));
```

native: 0.05s, wasm: 0.11s, asmjs: 0.9s

Minimal and characteristic polynomial of a random matrix with coefficients in a non-prime finite field. A large random matrix is constructed by blocks from a small one and similarity transforms are applied:

```
restart(1):;
n1:=30; n:=2*n1; GF(103,2,g);
m:=ranm(n1,n1,g):;
M:=blockmatrix(2,2,[m,0*m,0*m,m]):;
for j from 1 to 10 do
  p:=idn(n);
  q:=p;
  r:=rand(n);
  d:=ranv(1,-4)[0];
  p[r]:=seq(d,n);
  p[r,r]:=1;
  q[r]:=seq(-d,n);
  q[r,r]:=1;
  M:=q*M*p;
od:;
time(p:=pmin(M)); time(q:=pcar(m));
p-q;
```

native: 0.38 and 0.06, wasm: 0.55 and 0.08, asmjs: 0.63 and 0.09

Groebner-basis benchmark (available in Doc, Examples in Xcas for Firefox, remove `mod p` for computation on $\mathbb{Q}$)

cyclic7 modulo `prevprime(2^24)`:

native: 0.3s, wasm: 0.4s, asmjs: 1.9s

cyclic7 on $\mathbb{Q}$:

native: 1.7s, wasm:5.3s, asmjs: 16s

Interactive documents written in LaTeX

Combining LaTeX rendering quality and CAS computing is not new:

1. many math softwares provide converters to export data to a LaTeX file,
2. some programs handle both LaTeX-like rendering and computation, e.g. `texmacs` or `lyx`.

In the first case, however, if the writer changes some inputs in his computation, he must export again the result and include it in his LaTeX file, and in the second case the data format is not standard LaTeX and requires additional software to be installed on the reader device or a net access to a server to run the computations.

The solution presented here is new in that the writer edits a standard LaTeX file, adds a few simple commands like `\giacinputmath{factor(x^10-1)}` or `\giacinput{plot(sin(x))}` and compiles it to produce an HTML5+MathML document. The reader can see the document in any browser (it is optimized for Firefox), and he can modify computation commandlines and run them.

The writer can also compile the LaTeX file to PDF for printing purposes by running `giac file.tex`, where `giac` will precompute the Giac commands in an intermediate tex file and call `pdflatex` on it.

On the writer side

The writer must install `hevea` (`hevea.inria.fr`) or `hevea-mathjax`, `Giac/Xcas` for computing-enabled output and `heveatomml` for MathML output. The files `giac.tex`, `hevea.sty`, `mathjax.sty`, and `giac.js` must be copied to the LaTeX working directory.

The writer opens a LaTeX file with his usual editor. In the preamble he adds the following lines:

```
\makeindex
\input{giac.tex}
\giacmathjax
```

For support of interactive CAS LaTeX commands, the writer should add

```
\begin{giacjshere}
\tableofcontents
\printindex
```

just after `\begin{document}` and

```
\end{giacjshere}
```

just before `\end{document}`. Printing the table of contents and index before the first LaTeX section command is recommended, otherwise the HTML output `Table` and `Index` buttons will not link correctly.

The rest of the source file is standard LaTeX except that new commands are available for interactive CAS support, for example:

- `\giacinputmath{`*cmdline*`}` and `\giaccmdmath{`*cmd*`}{`*args*`}` embed an interactive command (in the second form only *args* is interactive), whose MathML or plot output is typeset in LaTeX's inline math style (`$...$`). Both take an optional HTML 'style' argument for detailed formatting control.

  Example:

  ```
  \giacinputmath{factor(x^10-1)}
  \giaccmdmath{factor}{x^4-1}
  ```

- `\giacinputbigmath`, `\giaccmdbigmath` the same but typeset in LaTeX's display math style (`$$...$$`).
- `\begin{giacprog}...\end{giacprog}` holds a program or multi-line command. The program is run whenever the user presses the `ok` button. To have the program executed already at load time, replace `giacprog` with `giaconload`.
- `\giacslider{`v`}{`$v_{\min}$`}{`$v_{\max}$`}{`Δv`}{`v_0`}{`*cmd*`}` adds a slider. When the user modifies the slider interactively, the new value is stored in variable v and *cmd* is executed.

  Example:

  ```
  \giacslider{a}{-5}{5}{0.1}{0.5}
    {plot(sin(a*x))}
  ```

Once the source file is written, it is compiled to HTML5 with the command

```
hevea2mml sourcefile.tex
```

The HTML output and the `giac.js` files should be available to the Web server in the same directory.

For a more precise description, please refer to `www-fourier.ujf-grenoble.fr/~parisse/giac/castex.html`

On the reader side

The reader's browser opens an HTML5+MathML file (linking to the `giac.js` Javascript). The MathML is rendered natively on Firefox or Safari, while Chrome or Internet Explorer will automatically load MathJax to render MathML – this is of course noticeably slower if the document is large. Computations are run by the reader's browser (the CAS is Javascript code). This is

```
\newcommand{\giacinput}[2][style="width:400px;height:20px;font-size:large"]{
  \ifhevea
    \@print{<textarea onkeypress="UI.ckenter(event,this,1)"}
    \@getprint{#1>#2}
    \@print{</textarea>
      <button onclick="
        previousSibling.style.display='inherit';
        var tmp=UI.caseval(previousSibling.value);
        tmp=UI.rmquote(tmp);
        nextSibling.innerHTML=' '+tmp;
        UI.render_canvas(nextSibling);
      ">ok</button>
      <span></span>
      <br>}
  \else
    \lstinline@#2@
  \fi
}
```

Figure 1: *The definition of* `giacinput` *with HTML code highlighted in blue and Javascript in red.*

slower than native code but faster than network access to a server and it does not require setting up a specific server for computations.

How this is done

All `\giac...` commands are defined in `giac.tex`. An example definition is shown in Fig. 1.

In general, if `hevea` compiles the file, the `\ifhevea` part is active, and the command will output an HTML5 `<textarea>` element and a OK `<button>`, with a callback to Javascript code that will evaluate the CAS command through `UI.caseval` and fill the next HTML5 `<span>` field with the result.

The CAS evaluation is performed by the same method as inside Xcas for Firefox.

References

[1] L. Maranget, Hevea: LaTeX to HTML5 compiler, `http://hevea.inria.fr`, 2017.

[2] B. Parisse and R. D. Graeve, Giac/Xcas Computer Algebra System, `http://www-fourier.ujf-grenoble.fr/~parisse/giac.html`, 2017.

[3] A. Zakai, Emscripten: C/C++-to-Javascript compiler, `http://kripken.github.io/emscripten-site`, 2017.