16bit Simulation with GNU *Octave*

Andreas Stahel
Bern University of Applied Sciences

OctConf 2017, March 20–22, 2017, CERN, Geneva

Personal I

Andreas Stahel
Mathematics

- Teaching:
  - Math at Bachelor level to mechanical and electrical engineers
  - Numerical Methods for the Master Program of Biomedical Engineering
  - A class on how to use *Octave* to solve engineering problems
- As member of the Institute for Human Centered Engineering (HuCE): many industry projects in mathematical modeling
- Web: `https://web.ti.bfh.ch/sha1/`
- E-mail: `Andreas.Stahel@bfh.ch`

Personal II

Concerning *Octave*:

- *Octave* is used regularly for teaching, project work and research.
- I teach a class on how to use *Octave* for engineering problems.
- I started using *Octave* in 1993/94 and am addicted to it since.
- *Octave* replaces `MATLAB` for many reasons: open source, great community support, platform independent, (legally) free.
- My professional life would be different without *Octave*!

Thank you guys

Why computing on a μC?

- Some μC are very affordable and thus used in many devices, not visible by the user.
- Functions can useful to calibrate sensors, or one might do a first step of the data treatment on the μC based sensor.
- Developing on a true μC can be tedious. Using a desktop and the power of *Octave* is convenient.

Facts for Computing on μC I

- On most affordable μC only integer arithmetic is implemented in hardware, i.e. no FPU.
- Floating point libraries are large, slow and the results are often overly accurate.
- If you use a 12bit AD converter, there is no need for a 32bit accuracy of the subsequent calculations.
- Since $+$, $-$ and $*$ are available one can aim to implement polynomial functions.

Facts for Computing on μC II

Different approaches are possible to implement the evaluation of a given function. It is often a compromise between the amount or required storage and the computations needed.

more storage fewer computations	$\longleftrightarrow$ $\longleftrightarrow$	less storage more computations
look up table	piecewise interpolation linear quadratic	one global polynomial

Facts for Computing on μC III

- On a typical 16bit μC the arithmetic operations for integers are implemented in harware.

16bit	$\pm$	16bit	$\rightarrow$	16bit
16bit	$*$	16bit	$\rightarrow$	32bit

- Division by 2^{16} is free (high double byte), division by 2^k is cheap (shift).
- Use full the ranges available for the data types int16 or int32 to obtain optimal accuracy.
- For multiplications we aim to use the full range of 32bit results, but will only use the high double byte for further computations.

Approximation by Polynomials

To approximate a given function on a bounded interval by a polynomial, different mathematical tools might be useful:

- Linear regression, i.e. least square approximation
- Chebyshev approximation
- Optimization by using `fminsearch()`, based on maximum norm, or L_2-norm, or ...
- A combination of the above.

For the problem at hand I worked with Chebyshev approximations.

Chebyshev Approximation I

The Chebyshev polynomials on the interval $[-1\,,\,+1]$ are given by

$$
\begin{aligned}
T_n(x) &= \cos(n \arccos(x)) \\
T_0(x) &= \cos(0) = 1 \\
T_1(x) &= \cos(\arccos(x)) = x \\
T_2(x) &= 2\,x^2 - 1 \\
T_3(x) &= 4\,x^2 - 3\,x \\
T_4(x) &= 8\,x^4 - 8\,x^2 + 1 \\
&\vdots
\end{aligned}
$$

A recursive identity allows to determine the polynomials efficiently.

$$T_{n+1}(x) = 2\,x \cdot T_n(x) - T_{n-1}(x)$$

Chebyshev Approximation II

A function defined on $[-1\,,+1]$ is approximated by a polynomial $p_N(x)$ of degree N.

$$\begin{aligned} c_n &= \frac{2}{\pi}\int_{-1}^{1} f(x)\;T_n(x)\;\frac{1}{\sqrt{1-x^2}}\;dx \\ f(x)\approx p_N(x) &= \frac{c_0}{2}+\sum_{n=1}^{N} c_n\;T_n(x) \end{aligned}$$

This is easily implemented in *Octave*.

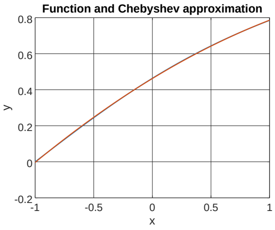
If a function $g(z)$ is defined on $[a\,,b]$ then move it to the standard interval $[-1\,,+1]$ by $f(x)=g(a+(x+1)\cdot\frac{b-a}{2})$

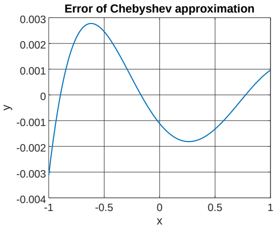
Approximate arctan(x) by a Parabola

- The above Chebyshev approximation can be used to approximate the function $f(x) = \arctan((1+x)/2)$ by a parabola.

$$\begin{aligned} f(x) \approx p_2(x) &= -0.0709107 \cdot x^2 + 0.394737 \cdot x + 0.4625339 \\ &= (-0.0709107 \cdot x + 0.394737) \cdot x + 0.4625339 \end{aligned}$$

- The relative error of this approximation $p_2(x)$ can be determined in bits, use `log2()`, leading to $7.97 \approx 8$ correct bits.





Setup for the 16bit Computation

To determine the 16bit values of

$$p_2(x) = (-0.0709107 \cdot x + 0.394737) \cdot x + 0.4625339$$

aim for vector `y16` and a factor `yscale` such that

$$\text{yscale} \cdot \text{y16} \approx p_2(x)$$

The goal is to implement this evaluation with a 16bit arithmetic, while keeping the result as accurate as possible and avoiding overflow, i.e. results exceeding $\pm 2^{15}$.

Start with a fine grid of x-values, e.g. `x=linspace(-1,1,100000);` Since $-1 \leq x \leq 1$ we multiply x by `xscale` and convert to `int16` with $\text{x16} \approx \text{xscale} \cdot x$, such that

$$-\text{MaxVal} \leq \text{x16} \leq +\text{MaxVal} = 2^{15} - 1 = 32767$$

With this we use the full accuracy available on a 16bit arithmetic.

Step 1: res1 $= -0.070910677 \cdot x + 0.3947365$ I

To perform the first Horner step proceed as follows:

$y = -0.070910677 \cdot x + 0.3947365$	
y16 = -32767 (use full scale)	$\text{yscale} = \frac{32767}{0.070910677}$
prod16 $=$ y16 * x16/2^{16}	$\text{yscale} = \frac{\text{yscale}\cdot\text{xscale}}{2^{16}}$
if $\lvert\text{prod16} + \text{yscale} \cdot 0.395\rvert > 32767$	rescale, divide by 2^k
add16 $=$ int16(yscale * 0.3947365)	integer to be added
y16 $=$ prod16 $+$ add16	

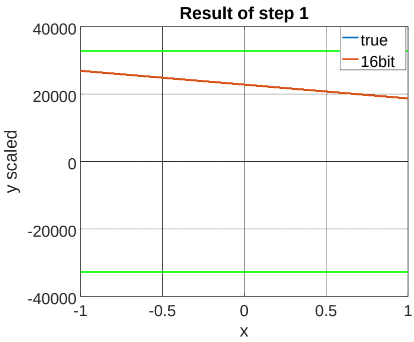
With the above numbers rescaling by $1/4$ is required and leads to add16 $= 22800$.

The result y16 satisfies

$$\text{yscale} \cdot y16 \approx -0.070910677 * x + 0.3947365$$

Step 1: $\text{res1} = -0.070910677 \cdot x + 0.3947365$ II

The result can be visualized, using exact (double precision) and approximate (16bit) computations.



Step 2: $\text{res2} = \text{res1} \cdot x + 0.4625339$ I

To perform the second Horner step proceed as follows:

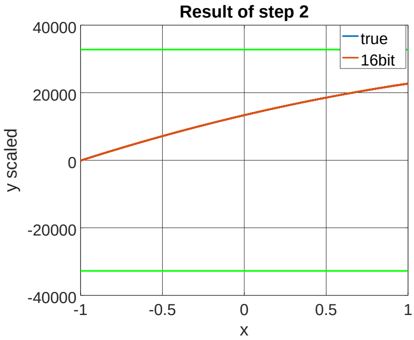
$$
\begin{array}{ll}
y = \text{res1} \cdot x + 0.4625339 & \\
\text{prod16} = \text{y16} * \text{x16}/2^{16} & \text{yscale} = \frac{\text{yscale} \cdot \text{xscale}}{2^{16}} \\
\text{if } |\text{prod16} + \text{yscale} \cdot 0.4625| > 32767 & \text{rescale, divide by } 2^k \\
\text{add16} = \text{int16}(\text{yscale} * 0.4625339) & \text{integer to be added} \\
\text{y16} = \text{prod16} + \text{add16} &
\end{array}
$$

With the above numbers no rescaling is required, thus $\text{add16} = 13357$
The result y16 satisfies

$$\text{yscale} \cdot y16 \approx p_2(x)$$

Step 2: $\text{res2} = \text{res1} \cdot x + 0.4625339$ II

The result can be visualized, exact (double precision) and approximate (16bit) computations.



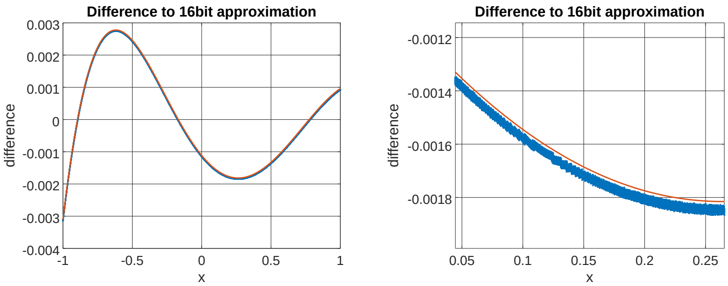
This is an approximation of the function $\arctan(\frac{1+x}{2})$.

The Result

To examine the quality graph the difference of true function and its 16bit approximation. The accuracy is given by

7.96 bit	for difference to the arctan-function
13.6 bit	for the difference to the polynomial $p_2(x)$

The error is dominated by the Chebyshev approximation.



The Resulting C++ Code

```
#include <octave/oct.h>
DEFUN_DLD (atan32,args,nargout,...
           "atan␣with␣int16␣arithmetic")
// for x = z*32767 and -1 <= z <= 1 the value of
// y = 28878.761*arctan((z+1)/2) will be computed
{
  static int i0 = -32767;
  static int i1 = +22800;
  static int i2 = +13357;
  int x = args(0).int_value();  int r ;
  r = i1+((i0*x)>>18);
  r = i2+((x*r)>>16);
  return octave_value_list (octave_value(r));
}
```

In General I

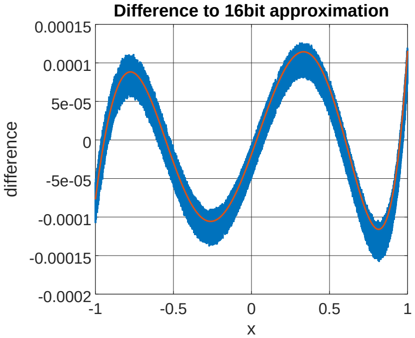
- The above Horner steps can be implemented in an *Octave* function. Examine the code `HornerStep.m`.
- The Chebyshev approximation can be of higher order, leading to more accuracy and more computational effort.
- There is no need for manual intervention. One can pack all of the above in an *Octave* script. Examine the code `atan16.m`.
- Using the code in `atan16.m` for a Chebyshev approximation of order 4 leads to smaller errors.

In General II

For the approximation by $p_4(x)$ of degree 4 we obtain an accuracy of

12.3 bit	for difference to the arctan-function
14.1 bit	for the difference to the polynomial $p_4(x)$

The error contributions by the Chebyshev approximation and the 16bit arithmetic are of the same magnitude.



A Fast Division in Hardware

The above technique was used to develop a fast hardware algorithm to divide numbers.

- Title: An Efficient Hardware Implementation for a Reciprocal Unit
- Authors: A. Habegger, A. Stahel, J Götte, M. Jacomet all Bern University of Applied Sciences
- DELTA 2010: 5th IEEE Symposium on Electronics Design, Test & Applications

What can I give to the community?

All of the above would not be possible without the help of the great *Octave* community.

Thank you guys

It is only fair that I try to contribute too.

- Find the lecture notes, codes and data on my web page web.ti.bfh.ch/˜sha1 in the frame `Octave`, search for the file `OctaveAtBFH.pdf`. Or use the direct link web.ti.bfh.ch/˜sha1/Labs/PWF/Documentation/OctaveAtBFH.pdf
- For a class on statistics I put together a collection of commands in web.ti.bfh.ch/˜sha1/StatisticsWithMatlabOctave.pdf .
- On a few occasions I reported bugs or contributed some code to *Octave* and its packages[1].

[1] The help and support you get from the community is amazing and beats any tech support from commercial companies I deal with!

That's all folks

Thank you for your attention

Slides and codes are available at
web.ti.bfh.ch/~sha1/Octave/OctConf2017/