Something Old, Something New...

Nic Schraudolph
Telluride 2007

NICTA

PASCAL
Pattern Analysis, Statistical Modelling and Computational Learning

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
ANU THE AUSTRALIAN NATIONAL UNIVERSITY
UNSW
NSW Department of State and Regional Development
BusinessACT ACT GOVERNMENT

Victoria The Place To Be
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT Queensland University of Technology
The University of Queensland Australia
NICTA Partners

Overview

Three half-hour lectures:

- **Floating-Point Bit Twiddling**
  Fast approximate exponential, logarithm, power, logistic functions (for GPUs & microcontrollers)
- **Gradient-Based Optimization**
  Review of standard methods, background for:
- **Stochastic Quasi-Newton Methods**
  My latest & greatest algorithms for fast online adaptation, and learning from large sets of data.

Twiddling the Bits of IEEE-754 Floating-Point Numbers for Fun ~~& Profit~~

just

Exponentials: Ubiquitous but Slow

Exponentials ubiquitous in scientific computing:

- **physics:** translates energy into probability (thermodynamics, electronics, quantum anything)
- **statistics:** exponential family of distributions ⇒ maximum likelihood estimation (machine learning!)

They are not cheap to compute:

- typically involves 10^{th} order Chebyshev polynomial
- used to be slow on general-purpose CPUs (embedded systems didn't have any floating-point)
- now hardware-accelerated on general-purpose CPUs but still slow on embedded systems: GPUs, µCs, ...

IEEE-754 Floating-Point Format

IEEE-754 value: $y = (-1)^s (1 + m)\ 2^{(x - 1023)}$

- s: *sign* bit
- x: 11-bit *exponent* (shifted by const. *bias* $x_0 = 1023$)
- m: 52-bit *mantissa*, binary fraction in the range [0,1)
- stored in 8 bytes of memory as:

sxxx xxxx | xxxx mmmm | mmmm mmmm | mmmm mmmm | mm...

1 2 3 4

Simple idea: to exponentiate a number, write it into the IEEE-754 exponent (duh).

Fast Approximate Exponentiation

NICTA

Specifically: to get EXP(x),

- multiply x by $2^{52} \cdot \ln 2$, cast result to integer
- add bias: $2^{52} \cdot 1023$, reinterpret as IEEE-754

Done! Okay, some more details:

- Use C *union* or C^{++} *reinterpret_cast* with 64-bit integer to directly access IEEE-754 components
- Can also use 2 32-bit integers (multiplier becomes $2^{20} \cdot \ln 2$; beware of big-endian vs. little-endian h/w)
- Especially fast for quantized arguments (uses only integer arithmetic!)
- no seatbelts (beware of overflow into sign bit!)

Exponentials for Nothing, and the Interpolation's for free!

NICTA

What happens to the "tail end" of that large integer we write into the IEEE-754 exponent?

- it overflows into the mantissa. Oh dear?
- actually, this performs linear interpolation for us!

We can use a trick to improve accuracy:

$$EXP_2(x) := EXP(x/2) / EXP(-x/2)$$

- at the cost of a single floating-point division, we now have piecewise *rational* interpolation
- even higher accuracy is possible, but gets increasingly expensive ⇒ usually not worth it

More Fast Functions

NICTA

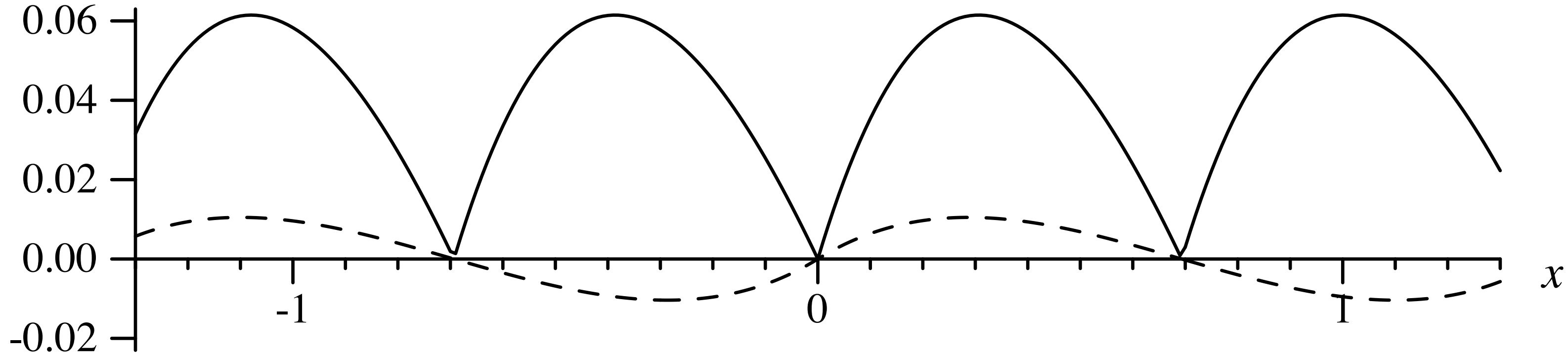
- logistic function: $y = 1/(1 + e^{-x})$ (tanh is similar)
  quite important for neuromorphs...!
  implement this as EXP(x/2)/[EXP(x/2) + EXP(-x/2)]
  ⇒ accuracy like EXP_2, but no extra division
- logarithms: just use EXP or EXP_2 in reverse
- power functions: use $x^y = 2^{y \ln_2 x}$
  (base 2: multiplier becomes bit shift ⇒ yet faster)
- square root: adjust for bias, shift 1 bit right
  (use this to initialize Newton-Raphson iterations)

How Inaccurate is it?

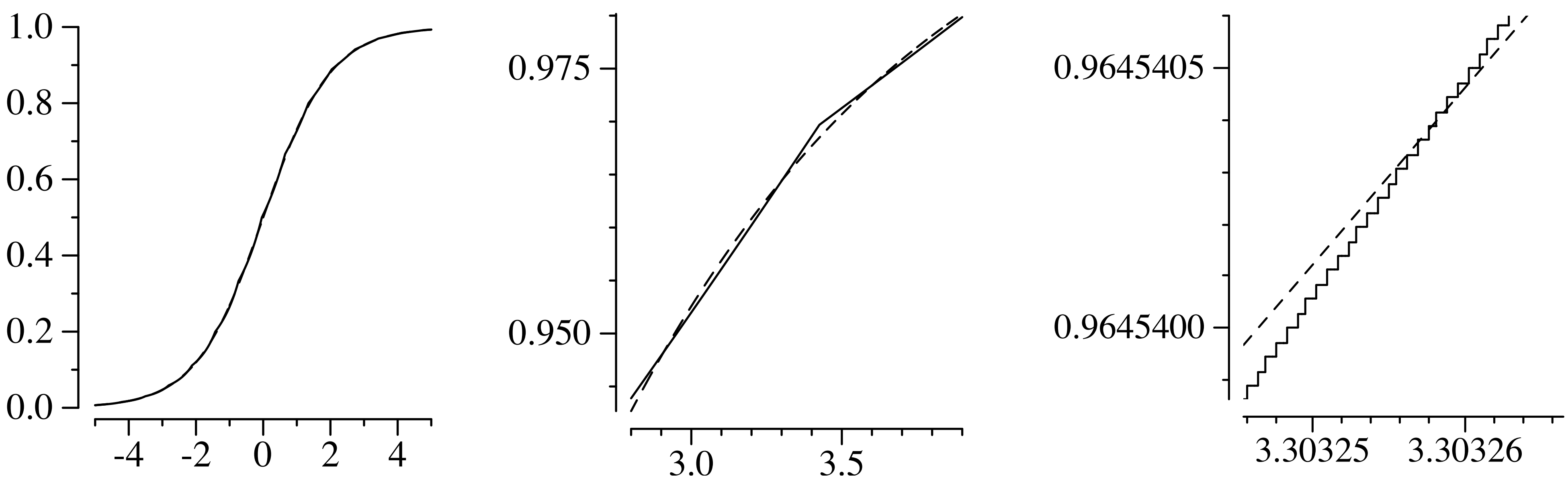
rel. error:

—— EXP

- - - EXP_2



Logistic via 1/(1 + EXP(-x)), 32-bit integers:



Conclusion

NICTA

IEEE-754 bit twiddling

- yields fast, approximate exp, log, pow, tanh, sqrt, ...
- saves memory (no look-up table to store)
- preserves cache (no memory access)
- zero-cost interpolation (overflow into mantissa)

For more information:

- basic EXP (with full error analysis) published in Neural Computation (1998)
- everything else not yet published (but I have code if you ask nicely :-)