Computing logarithms and other special functions

Mike Giles

University of Oxford
Mathematical Institute

Napier 400 – NAIS Symposium

April 2, 2014

Motivation & outline

Why is this not a solved problem? What's new?

- computer hardware continues to evolve:
  - new IEEE fused multiply-add operation has improved accuracy
  - vector computing is becoming important for both GPUs and CPUs
- there are "new" special functions to approximate

It's also interesting sometimes to see what goes on "under the bonnet".

Outline:

- reciprocal
- exponential
- logarithm
- inverse error function
- inverse Poisson CDF

Reciprocal

First task: computing x^{-1}. Standard floating point representation is

$$x = \pm r \times 2^n, \qquad n \in \mathbb{Z}, \quad 0.5 \le r < 1,$$

and hence reciprocal is

$$x^{-1} = \pm r^{-1} \times 2^{-n}.$$

How then to compute r^{-1}?

Method 1 (expansion): putting $r = 1 - \varepsilon$, then

$$\begin{aligned} r^{-1} &= 1 + \varepsilon + \varepsilon^2 + \varepsilon^3 + \varepsilon^4 + \varepsilon^5 + \dots \\ &= (1+\varepsilon)(1+\varepsilon^2)(1+\varepsilon^4)\dots \end{aligned}$$

Each term doubles the number of accurate bits, but it fails to give full machine accuracy.

Reciprocal

Method 2 (refinement): if $y \approx x^{-1}$, let $x\,y = 1-\varepsilon, \quad |\varepsilon| \ll 1$ and then

$$\begin{aligned} x^{-1} &= y\,(1-\varepsilon)^{-1} \\ &= y\,(1+\varepsilon+\varepsilon^2+\ldots) \end{aligned}$$

NVIDIA uses this approach for double precision reciprocals.

y is a low accuracy (20-bit) polynomial approximation computed by a special function unit (SFU), then full accuracy is achieved in 3 FMAs:

$$\begin{aligned} \varepsilon &:= 1 - x\,y \\ \varepsilon &:= \varepsilon + \varepsilon^2 \\ y &:= y + \varepsilon\,y \end{aligned}$$

Exponential

To compute $y = \exp(x)$, let

$$x = n \log 2 + r, \qquad n \in \mathbb{Z}, \quad |r| \leq \tfrac{1}{2} \log 2,$$

Then

$$y = \exp(r) \times 2^n$$

$\exp(r)$ can be computed using a standard expansion

$$\begin{aligned} \exp(r) &= \sum_{n=0}^{\infty} a_n r^n \\ &= 1 + a_1 \left(r + a_2 \left(r + a_3 \left(r + a_4 \left(r + a_5 \left(r + a_6 \left(\right.\right.\right.\right.\right.\right. \end{aligned}$$

Instead of $a_n = 1/n!$ can do slightly better using near-minimax polynomial approximation, but main point is that rapid decay in a_n means only 12 terms are needed for double precision.

Exponential

Mathematical libraries usually contain a second function:

$$\mathrm{expm1}(x) \equiv \exp(x) - 1.$$

When $|x| \ll 1$

- double precision error in $\exp(x)$ is $O(10^{-16})$
- double precision error in $\mathrm{expm1}(x)$ is $O(10^{-16}x)$

Logarithm

Similarly, to compute $y = \log(x)$, let

$$x = r \times 2^n, \qquad n \in \mathbb{Z}, \quad \frac{2}{3} < r \leq \frac{4}{3}$$

Then

$$y = n \log 2 + \log r$$

How to compute $\log r$?

Standard expansion gives:

$$\log(1+\varepsilon) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{1}{n} \varepsilon^n$$

but this converges slowly – needs 30 terms for double precision accuracy.

Logarithm

Since

$$\tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1}$$

putting $r = e^{2z}$ gives

$$z = \tanh^{-1}\left(\frac{r-1}{r+1}\right) = \tfrac{1}{2}\log r$$

so

$$\log r = 2\,\tanh^{-1}\left(\frac{r-1}{r+1}\right)$$

The advantage of this transformation is that $\tanh^{-1}(z)$ has a faster converging expansion around $z=0$, with just 12 odd powers of z required for double precision accuracy.

Logarithm

That is the algorithm currently used by NVIDIA, but an alternative idea is to use refinement.

If $y \approx \log(x)$, then let

$$x \, \exp(-y) = 1 + \varepsilon, \quad |\varepsilon| \ll 1$$

so that

$$\log x \;=\; y + \log(1+\varepsilon) \;\approx\; y + \varepsilon - \tfrac{1}{2}\,\varepsilon^2 + \ldots$$

A 20-bit approximation to $\log(x)$ in the SFU could lead to an efficient implementation, relying on the accuracy of $\mathrm{expm1}(x)$ when $x \approx 1$.

$$\varepsilon \;=\; x \, \exp(-y) - 1 \;=\; x \, \mathrm{expm1}(-y) + (x-1)$$

Inverse error function

Most mathematical libraries have an implementation of the error function

$$\mathrm{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t$$

and its inverse, $\mathrm{erf}^{-1}(x)$.

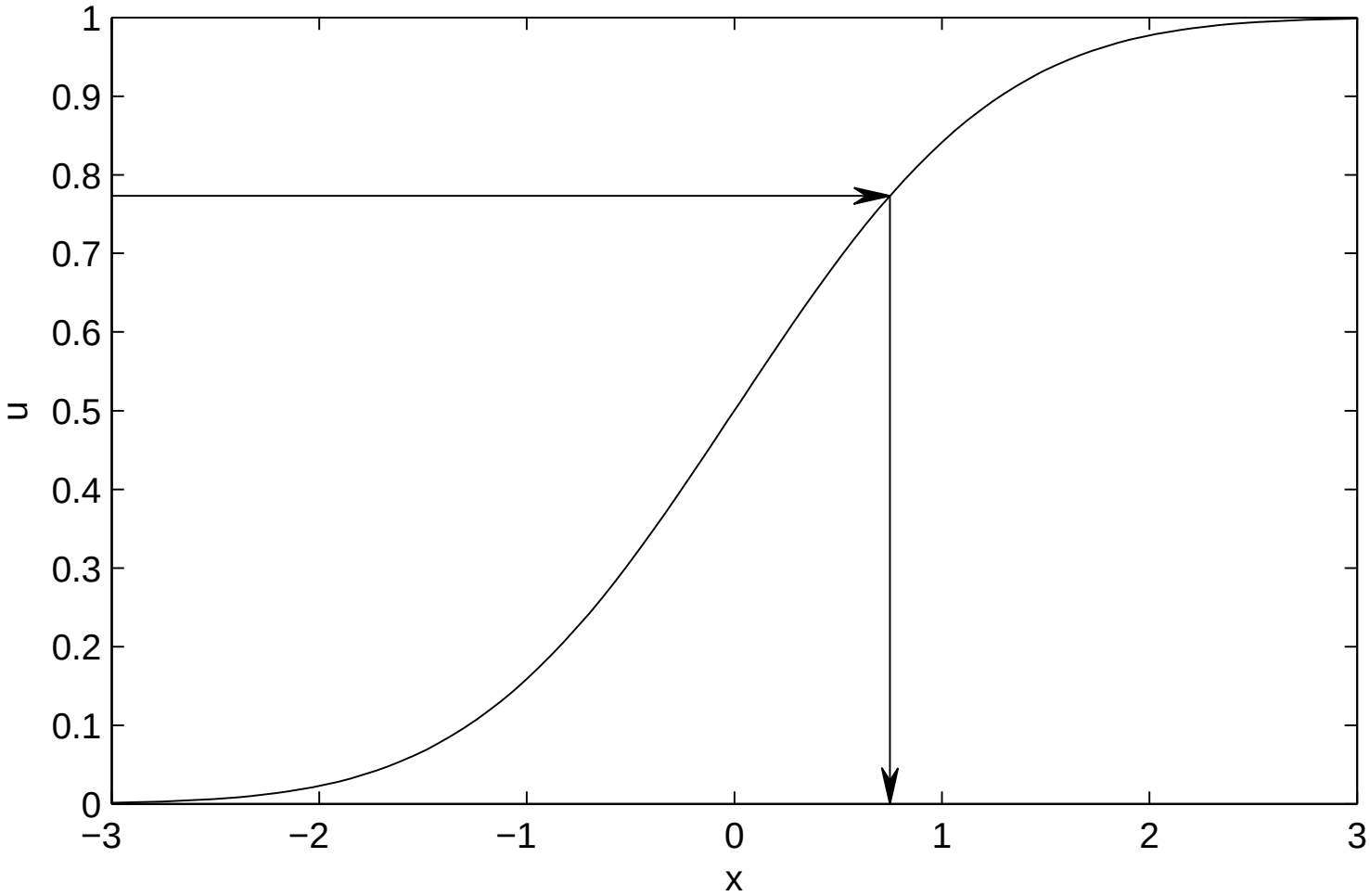
My interest in it is because of the relationship with $\Phi(x)$, the Normal cumulative distribution function (CDF). For a standard Normal random variable X,

$$\mathbb{P}(X<x) \equiv \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} \, \mathrm{d}t = \frac{1}{2} + \frac{1}{2} \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)$$

$$\Longrightarrow \qquad \Phi^{-1}(x) = \sqrt{2} \, \mathrm{erf}^{-1}(2x-1)$$

Inverse error function

One use is for converting uniformly distributed random variables on $(0, 1)$ to unit Normal random variables:



Inverse error function

Key properties of $\mathrm{erf}^{-1}(x)$:

- odd function of x
- $\sqrt{\log(1-|x|)}$ singularity near $x = \pm 1$

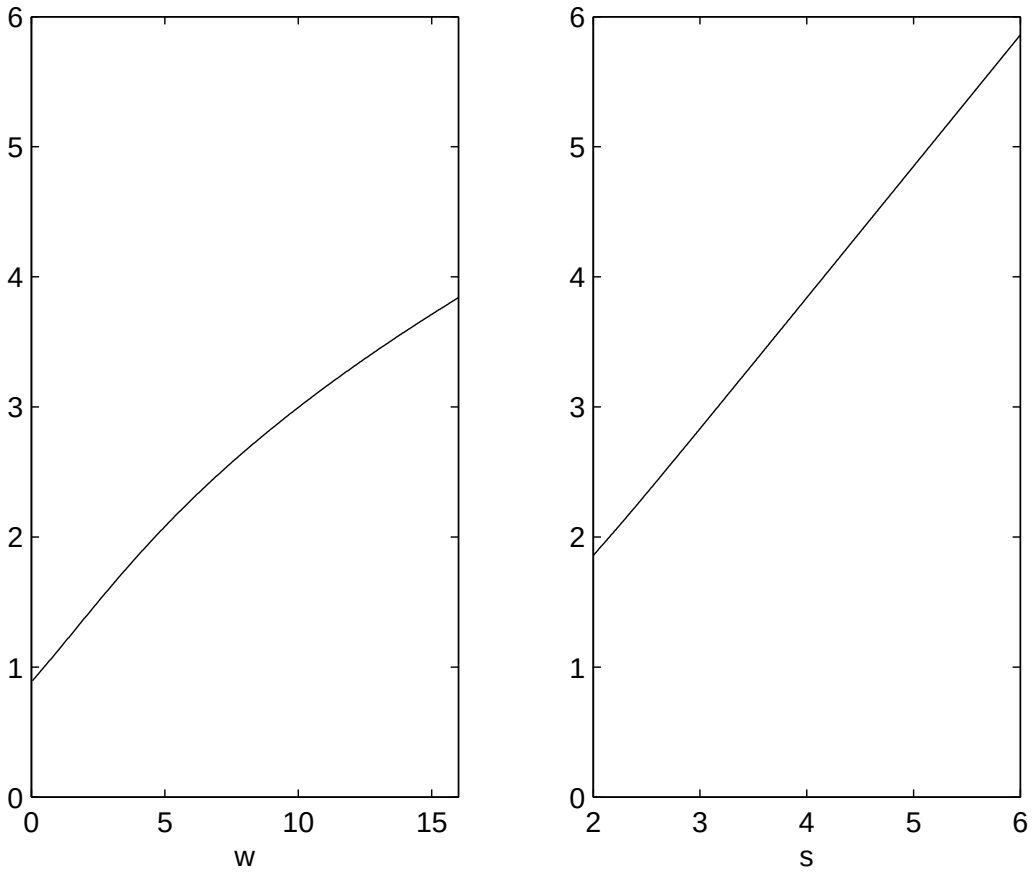
Standard CPU approximations use

- polynomial approximation in odd powers of x in a central region
- polynomial function of $\sqrt{\log(1-|x|)}$ near either end

Great for CPU, but expensive for GPU because vector execution means it effectively executes both pieces of code and just keeps the result of the relevant one.

Inverse error function

Transformation: let $\mathrm{erf}^{-1}(x) = x\, f(w), \quad w = -\log(1-x^2)$



$\mathrm{erf}^{-1}(x)\,/\,x$ plotted versus w and $s \equiv \sqrt{w}$.

Inverse error function

New single precision approximation is

$$\mathrm{erf}^{-1}(x) = \begin{cases} x\, p_1(w), & w \le w_1 \quad \text{central region} \\ x\, p_2(\sqrt{w}\,), & w_1 < w \quad \text{tail region} \end{cases}$$

Double precision approximation is similar but breaks tail region into two parts.

The central region is chosen to cover 99.9% of the $(-1, 1)$ x-interval, so each 32-element vector only uses central region in most cases.

p_1, p_2 are generated to be near-minimax polynomials of degree 8 in single precision, 20 in double precision.

The implementations in NVIDIA's maths library are now based on this.

Poisson CDF and inverse

A discrete Poisson random variable N with rate λ takes integer value n with probability

$$e^{-\lambda} \frac{\lambda^n}{n!}$$

Hence, the cumulative distribution function is

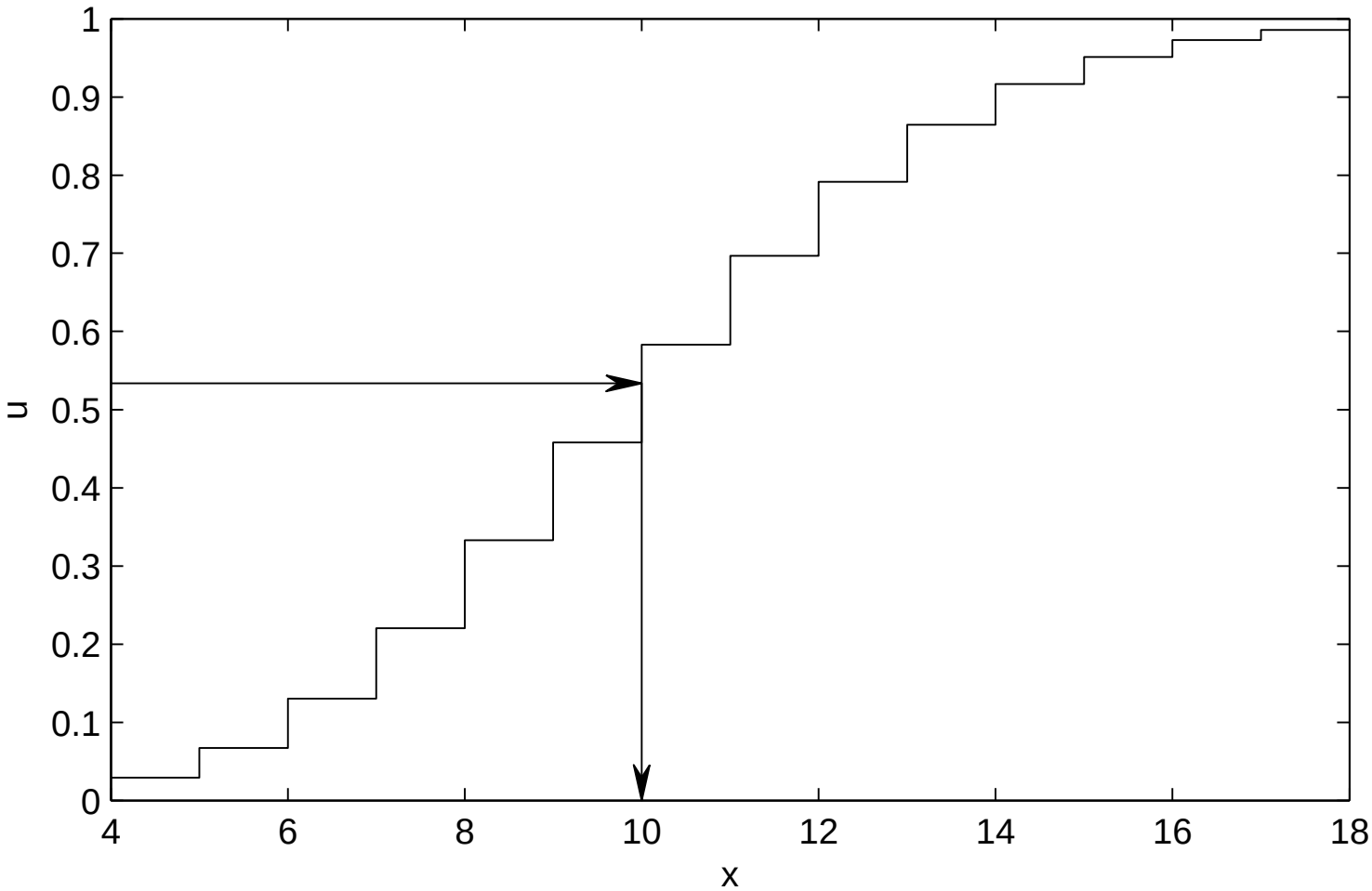
$$\overline{C}(n) \equiv \mathbb{P}(N \le n) = e^{-\lambda} \sum_{m=0}^{n} \frac{\lambda^m}{m!}.$$

To generate N, can take a uniform $(0,1)$ random variable U and then compute $N = \overline{C}^{-1}(U)$, where N is the smallest integer such that

$$U \le \overline{C}(N)$$

Poisson CDF and inverse

Illustration of the inversion process



Poisson CDF and inverse

When λ is fixed and not too large ($\lambda < 10^4$?) can pre-compute $\overline{C}(n)$ and perform a table lookup.

When λ is variable but small ($\lambda < 10$?) can use bottom-up/top-down summation.

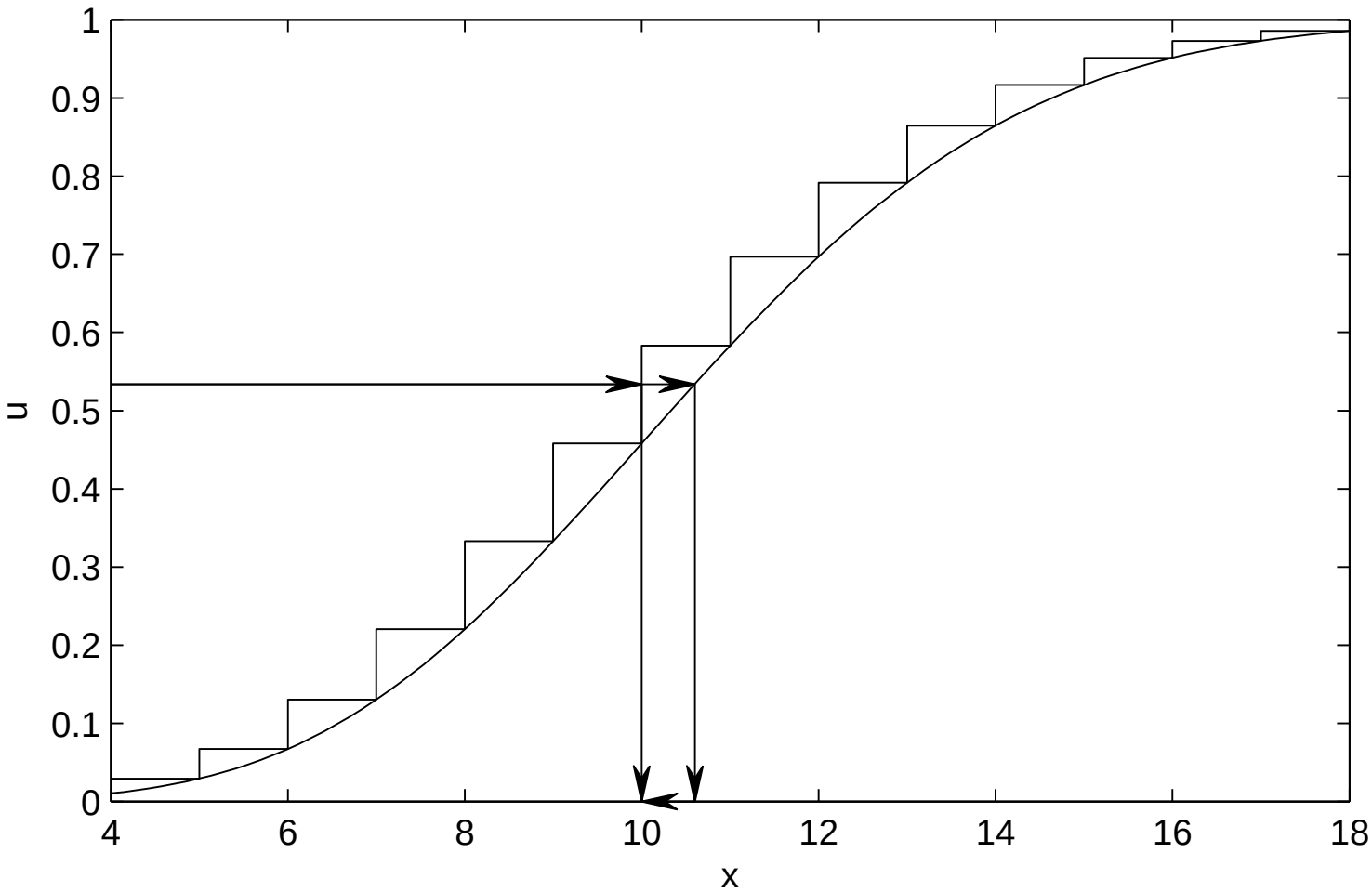
When λ is variable and large, then rejection methods can be used to generate Poisson r.v.'s, but the inverse CDF is sometimes helpful:

- stratified sampling
- Latin hypercube
- QMC

This is the problem I am concerned with, approximating $\overline{C}^{-1}(u)$ at a cost similar to the inverse Normal CDF, or inverse error function.

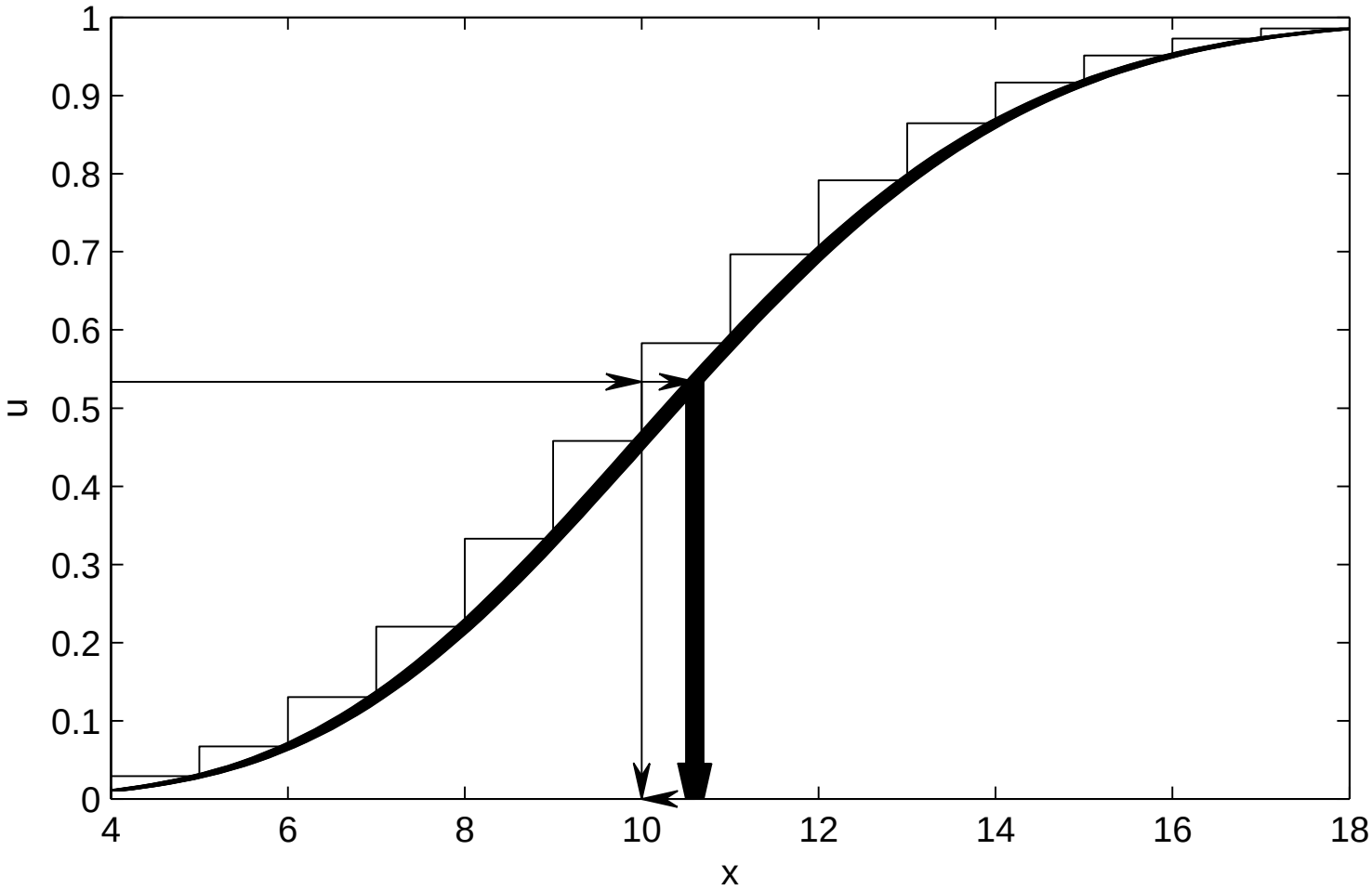
Poisson CDF and inverse

Illustration of the inversion process through rounding down of some $Q(u) \equiv C^{-1}(u)$ to give $\overline{C}^{-1}(u)$



Poisson CDF and inverse

Errors in approximating $Q(u)$ can only lead to errors in rounding down if near an integer



Incomplete Gamma function

If X is a positive random variable with CDF

$$C(x) \equiv \mathbb{P}(X < x) = \frac{1}{\Gamma(x)} \int_\lambda^\infty e^{-t}\, t^{x-1}\, \mathrm{d}t.$$

then integration by parts gives

$$\mathbb{P}(\lfloor X \rfloor \le n) = \frac{1}{n!} \int_\lambda^\infty e^{-t}\, t^n\, \mathrm{d}t = e^{-\lambda} \sum_{m=0}^{n} \frac{\lambda^m}{m!}$$

$$\Longrightarrow \quad \overline{C}^{-1}(u) = \lfloor C^{-1}(u) \rfloor$$

We approximate $Q(u) \equiv C^{-1}(u)$ so that $|\widetilde{Q}(u) - Q(u)| < \delta \ll 1$

This will round down correctly except when $Q(u)$ is within δ of an integer – then we need to check some $C(m)$

Normal approximation

It is well known from the Central Limit Theorem that

$$C(x) \;\approx\; \Phi\left(\frac{x-\lambda}{\sqrt{\lambda}}\right)$$

which motivates the following change of variables

$$x = \lambda + \sqrt{\lambda}\, y, \quad t = \lambda + \sqrt{\lambda}\,(y-z)$$

giving

$$C(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} I(y,z)\; \mathrm{d}z$$

where

$$\log I = \tfrac{1}{2}\log(2\pi) - \log\Gamma(x) - t + (x-1)\log t - \tfrac{1}{2}\log\lambda$$

Normal approximation

An asymptotic expansion in powers of $\varepsilon \equiv \lambda^{-1/2}$ yields

$$I(y,z) = \exp(-\tfrac{1}{2}z^2)\left(1 + \sum_{n=1}^{\infty} \varepsilon^n p_n(y,z)\right)$$

where $p_n(y,z)$ are polynomial in y and z. Integrating by parts gives

$$C(x) \approx \Phi(y) + \phi(y)\Big(\varepsilon(-\tfrac{1}{3} - \tfrac{1}{6}y^2) + \varepsilon^2(\tfrac{1}{12}y + \tfrac{1}{72}y^3 - \tfrac{1}{72}y^5) + \varepsilon^3(-\tfrac{1}{540} - \tfrac{23}{540}y^2 + \tfrac{7}{2160}y^4 + \tfrac{5}{648}y^6 - \tfrac{1}{1296}y^8)\Big)$$

and inverting this gives the asymptotic expansion

$$Q(u) = \lambda + \sqrt{\lambda}\,w + (\tfrac{1}{3} + \tfrac{1}{6}w^2) + \lambda^{-1/2}(-\tfrac{1}{36}w - \tfrac{1}{72}w^3) + \lambda^{-1}(-\tfrac{8}{405} + \tfrac{7}{810}w^2 + \tfrac{1}{270}w^4) + O(\lambda^{-3/2})$$

where $w = \Phi^{-1}(u)$.

Normal approximation

All asymptotic expansions were performed using MATLAB's Symbolic Toolbox.

This gives three approximations:

$$\begin{aligned}
\widetilde{Q}_{N1}(u) &= \lambda + \sqrt{\lambda}\, w + (\tfrac{1}{3} + \tfrac{1}{6}\, w^2) \\
\widetilde{Q}_{N2}(u) &= \widetilde{Q}_{N1}(u) + \lambda^{-1/2}(-\tfrac{1}{36}\, w - \tfrac{1}{72}\, w^3) \\
\widetilde{Q}_{N3}(u) &= \widetilde{Q}_{N2}(u) + \lambda^{-1}(-\tfrac{8}{405} + \tfrac{7}{810}\, w^2 + \tfrac{1}{270}\, w^4)
\end{aligned}$$

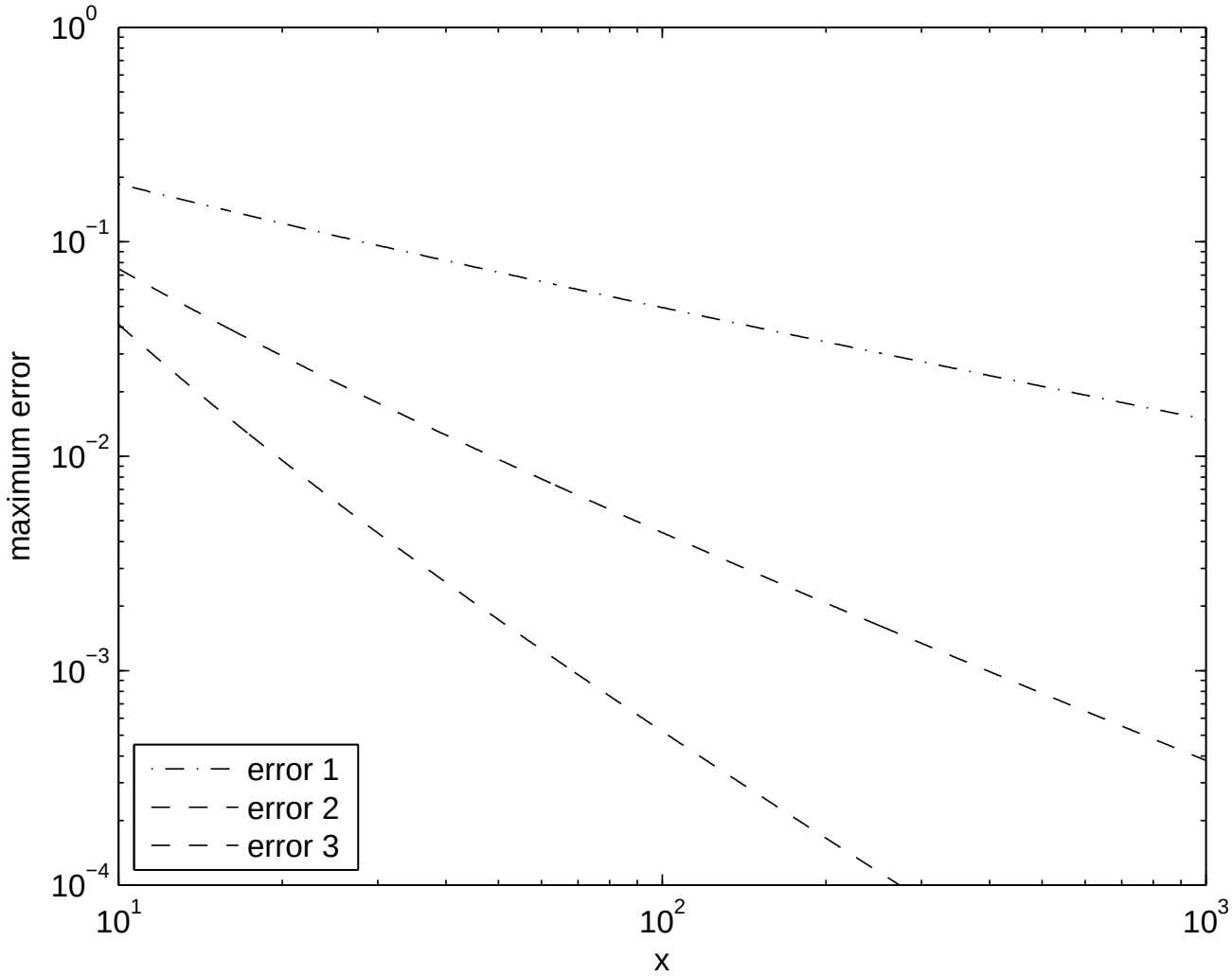
and suggests an error bound for $\widetilde{Q}_{N2}$:

$$\delta = \lambda^{-1}(\tfrac{1}{40} + \tfrac{1}{80}\, w^2 + \tfrac{1}{160}\, w^4)$$

with $\mathbb{E}[\delta] = \frac{9}{160}\, \lambda^{-1}$.

Normal approximation

Maximum error over range $|w| \leq 3$:



Alternative Temme approximation

The Normal approximation is quite cheap, but not sufficiently accurate when $|w|$ is large.

Also, even when $|w| < 3$, it is not accurate enough for GPU execution because most vectors of length 32 will have one element with $\widetilde{Q}(u)$ close to an integer.

Hence, I have derived a second approximation based on work by Temme (1979). He developed a very accurate uniform asymptotic expansion for $C(x)$; I obtained the corresponding expansion for its inverse $Q(u)$.

$C(m)$ evaluation

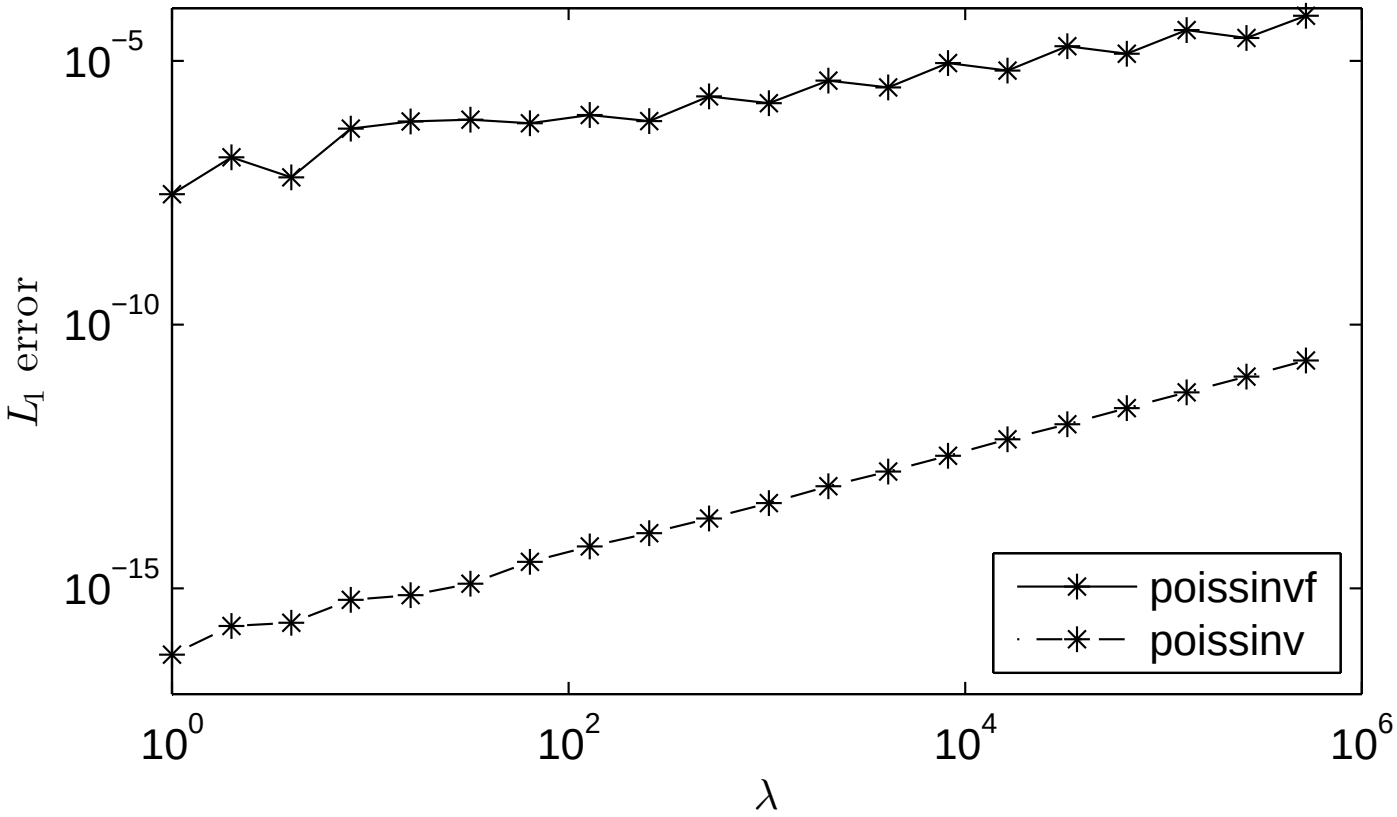
When $\widetilde{Q}(u)$ is too close to an integer $m+1$, (i.e. within δ) we need to evaluate $C(m)$ to choose between m and $m+1$.

When $\frac{1}{2}\,\lambda \le m \le 2\,\lambda$, this can be done very accurately using another approximation due to Temme (1987).

Outside this range, a modified version of bottom-up / top-down summation can be used, because successive terms decrease by factor 2 or more.

This correction determines the final accuracy of the inverse Poisson CDF function.

Accuracy of approximations



L_1 errors of `poissinvf` and `poissinv` functions written in CUDA.

(It measures the fraction of the $(0, 1)$ interval for which the error is ± 1.)

Conclusions

- hopefully an interesting insight into what goes on in mathematical libraries
- definitely not something that most people need to worry about, but we need one or two people to work on such things
- codes and papers are available from `http://people.maths.ox.ac.uk/gilesm/codes.html`