Fast evaluation of the inverse Poisson CDF

Mike Giles

University of Oxford
Mathematical Institute
Oxford e-Research Centre

GPU Technology Conference 2014

March 26, 2014

Outline

- problem specification
- incomplete Gamma function
- CPUs versus GPUs
- inverse approximation based on Temme expansion
- Temme asymptotic evaluation
- complete algorithm
- results

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

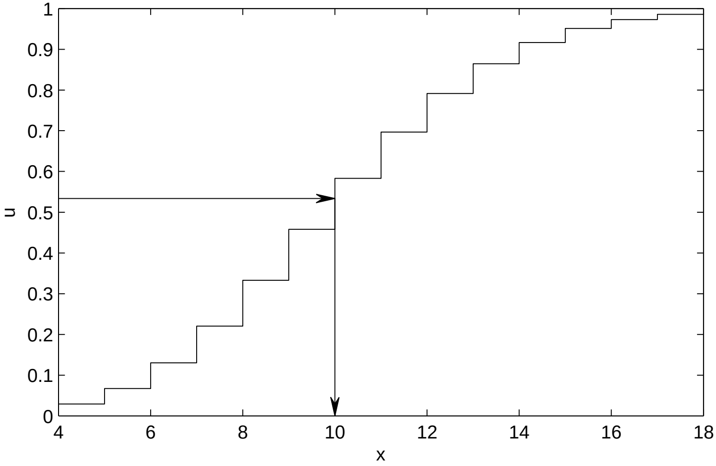
$$\overline{C}(n) \;\equiv\; \mathbb{P}(N \leq n) \;=\; e^{-\lambda} \sum_{m=0}^{n} \frac{\lambda^m}{m!}.$$

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

$$U \;\leq\; \overline{C}(N)$$

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.

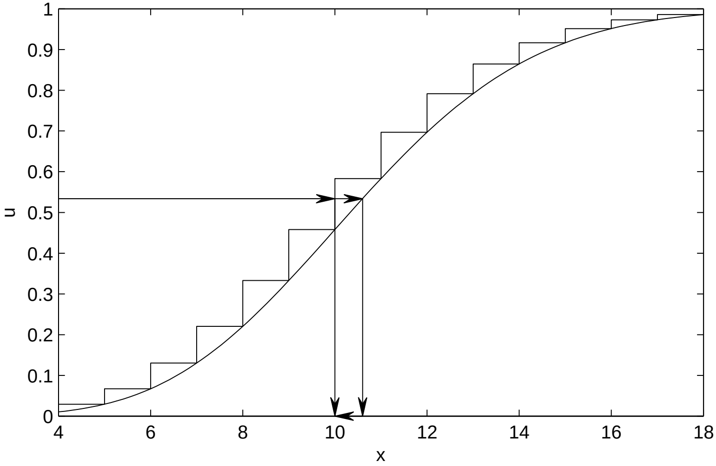
When λ is variable and large, then rejection methods can be used to generate Poisson r.v.'s, but the inverse CDF is sometimes helpful:

- 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.

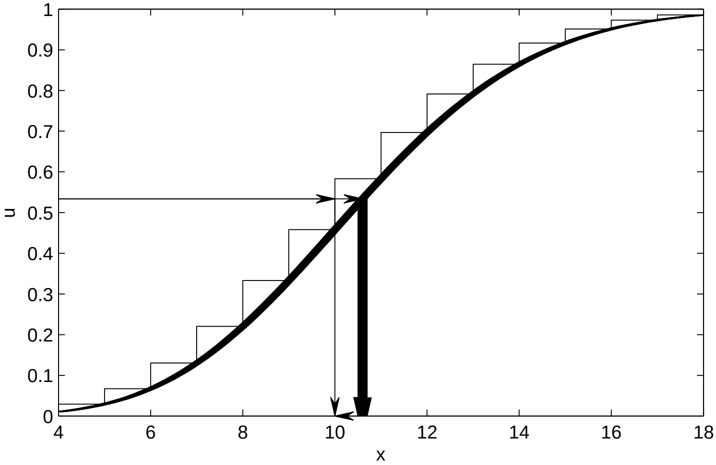
Poisson CDF and inverse

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

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

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

We will 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 $\overline{C}(m)$

CPUs and GPUs

On a CPU, if the costs of $\widetilde{Q}(u)$ and $\overline{C}(m)$ are C_Q and C_C, the average cost is approximately

$$C_Q + 2\,\delta\, C_C.$$

However, on a GPU with a warp length of 32, the C_C penalty is incurred if any thread in the warp needs it, so the average cost is

$$C_Q + \left(1 - (1{-}2\,\delta)^{32}\right)\, C_C \;\approx\; C_Q + 64\,\delta\, C_C \quad \text{if } \delta \ll 1.$$

This pushes us to more accurate approximations for GPUs.

Temme expansion

Temme (1979) derived a uniformly convergent asymptotic expansion for $C(x)$ of the form

$$C(x) = \Phi\left(\lambda^{\frac{1}{2}} f(r)\right) + \lambda^{-\frac{1}{2}}\, \phi\left(\lambda^{\frac{1}{2}} f(r)\right) \sum_{n=0}^{\infty} \lambda^{-n}\, a_n(r)$$

where $r = x/\lambda$ and

$$f(r) \equiv \sqrt{2\ (1 - r + r \log r)},$$

with the sign of the square root matching the sign of $r-1$.

Temme expansion

Based on this, can prove that the quantile function is

$$Q(u) \approx \lambda\, r + c_0(r)$$

where

$$r = f^{-1}(w/\sqrt{\lambda}), \qquad w = \Phi^{-1}(u)$$

and

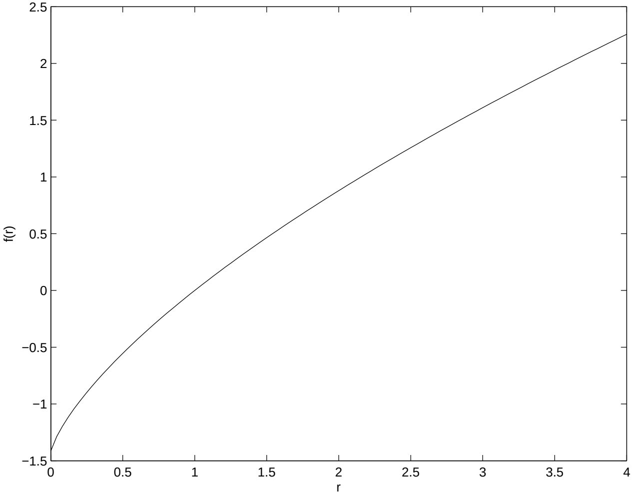
$$c_0(r) = \frac{\log\left(f(r)\sqrt{r}/(r-1)\right)}{\log r}$$

Both $f^{-1}(s)$ and $c_0(r)$ can be approximated very accurately (over a central range) by polynomials, and an additional *ad hoc* correction gives

$$\widetilde{Q}_T(u) = \lambda\ r + p_2(r) + p_3(r)/\lambda$$

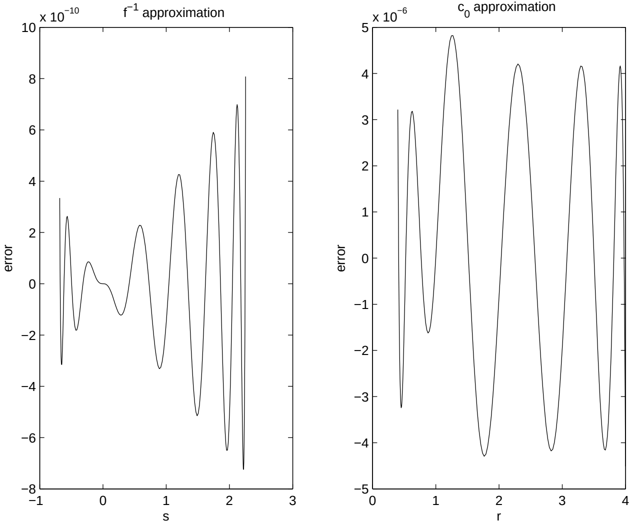
Temme approximation

The function $f(r)$



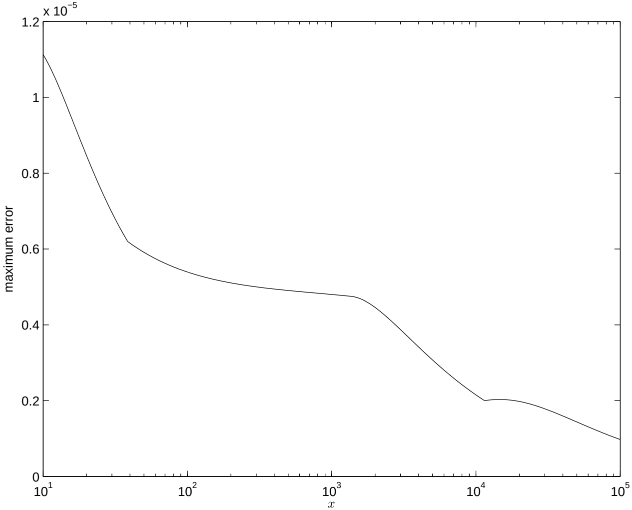
Temme approximation

Errors in $f^{-1}(s)$ and $c_0(r)$ approximations:



Temme approximation

Maximum error in $\widetilde{Q}_T$ approximation:



$C(m)$ evaluation

In double precision, when $\widetilde{Q}(u)$ is too close to an integer $m+1$, we need to evaluate $C(m)$ to choose between m and $m+1$.

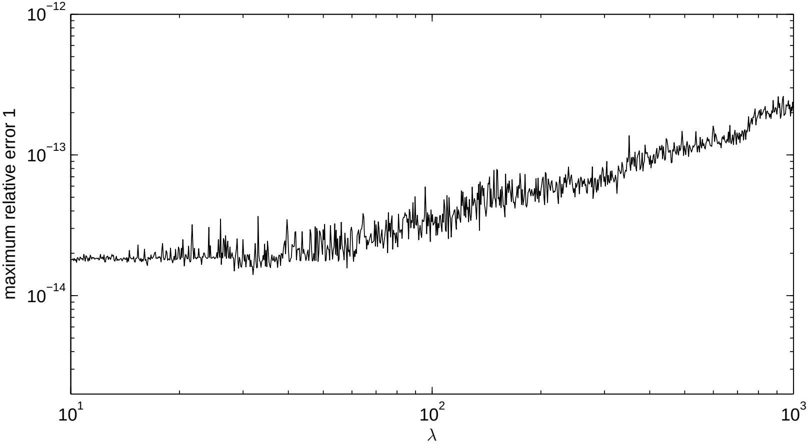
When $\frac{1}{2}\lambda \leq m \leq 2\lambda$, this can be done very accurately using another approximation due to Temme (1987).

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

In single precision this "correction" procedure does not improve the accuracy.

Temme approximation

Maximum relative error in Temme approximation for $C(m)$



The GPU algorithm (single precision)

given inputs: λ, u

if $\lambda > 4$
$\quad w := \Phi^{-1}(u)$
$\quad s := w/\sqrt{\lambda}$

$\quad$ if $s_{min} < s < s_{max}$ $\qquad$ main branch
$\qquad r := p_1(s)$
$\qquad x := \lambda\, r + p_2(r) + p_3(r)/\lambda$
$\quad$ else
$\qquad r := f^{-1}(w/\sqrt{\lambda})$ $\qquad$ Newton iteration
$\qquad x := \lambda\, r + c_0(r)$
$\qquad x := x - 0.0218/(x+0.065\lambda)$
$\quad$ end

$\quad n := \lfloor x \rfloor$

The GPU algorithm (single precision)

```
    if x > 10
        return n
    end
end

use bottom-up summation to determine n

if u > 0.5 and not accurate enough
    use top-down summation to determine n
end
```

Top-down summation finds smallest n such that

$$1-u \quad \geq \quad e^{-\lambda} \sum_{m=n+1}^{\infty} \frac{\lambda^m}{m!}$$

The GPU algorithm (double precision)

given inputs: λ, u

if $\lambda > 4$

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

$s := w/\sqrt{\lambda}$

if $s_{min} < s < s_{max}$

$r := p_1(s)$

$x := \lambda\, r + p_2(r) + p_3(r)/\lambda$

$\delta := 2\times 10^{-5}$

else

$r := f^{-1}(w/\sqrt{\lambda})$

$x := \lambda\, r + c_0(r)$

$x := x - 0.0218/(x+0.065\lambda)$

$\delta := 0.01/\lambda$

end

$n := \lfloor x+\delta \rfloor$

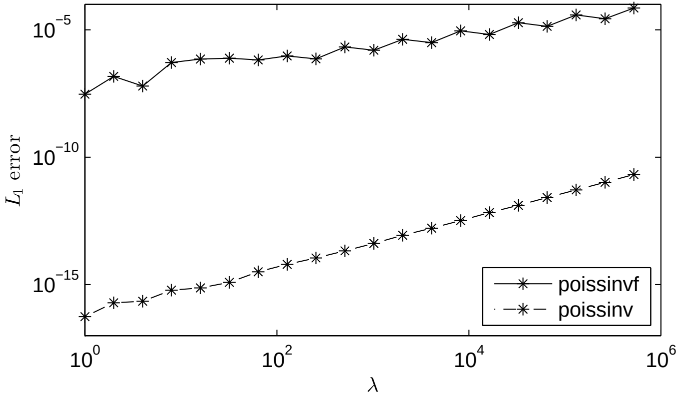
The GPU algorithm (double precision)

```
if x > 10
    if x−n > δ
        return n
    else if C(n) < u          "correction" test
        return n
    else
        return n−1
    end
end
end
```

use bottom-up summation to determine n

if $u > 0.5$ and not accurate enough
    use top-down summation to determine n
end

Accuracy



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.)

Performance

Samples/sec for `poissinvf` and `poissinv` using CUDA 5.0

	GTX670		K20	
λ	`poissinvf`	`poissinv`	`poissinvf`	`poissinv`
2	1.25e10	1.03e09	1.94e10	5.29e09
8	5.66e09	3.70e08	8.77e09	2.07e09
32	8.07e09	6.98e08	1.25e10	4.20e09
128	8.38e09	6.98e08	1.25e10	4.20e09
mixed	4.91e09	3.00e08	6.83e09	1.64e09
`normcdfinvf`	1.96e10		2.70e10	
`normcdfinv`		9.61e08		7.15e09

Conclusions

- By approximating the inverse incomplete Gamma function, have developed an approach for inverting the Poisson CDF for $\lambda > 4$
- Computational cost is roughly cost of inverse Normal CDF function plus three polynomials of degree 8–12
- Report and open source CUDA implementation available now: `http://people.maths.ox.ac.uk/gilesm/poissinv`
- Report also describes a second approximation which is faster for CPUs, but has more branching so is worse for GPUs