Flexible even densities

William McCausland

2018-11-24

Related files: `robust_HESSIAN.pdf`, 2017-06-17 and `new_skew_draw.pdf`, 2017-07-28.

Statement of problem

We are given a function $\varphi(x)$ with $\varphi(0) = 0$ such that $f(x) = c\phi(x)e^{\varphi(x)}$ is a good approximation of a target density that has a mode at zero. Here, $\phi(x)$ is the density of a $N(0, 1)$; we also let $\Phi(x)$ and $Q(x) = 1 - \Phi(x)$ be the corresponding cumulative distribution and survival functions. The normalizing constant c is not known.

Let $\varphi_e(x)$ and $\varphi_o(x)$ be the even and odd parts of $\varphi(x)$. Let $f_e(x)$ and $f_o(x)$ be the even and odd parts of $f(x)$. Let $F_e(x)$ be the cdf corresponding to $f_e(x)$, which is itself a proper and fully normalized density. Also let $f_{e+}(x) = 2f_e(x)1_{[0,\infty)}(x)$, $\phi_+(x) = 2\phi(x)1_{[0,\infty)}(x)$, which are also proper and fully normalized densities. We see that $F_{e+}(x) \equiv 2F_e(x) - 1$ is the cdf corresponding to f_{e+}. We also see that $\Phi_+(x) \equiv 2\Phi(x) - 1$ and $Q_+(x) \equiv 2Q(x)$ are the cumulative distribution and survival functions corresponding to $\phi_+(x)$.

We can always write

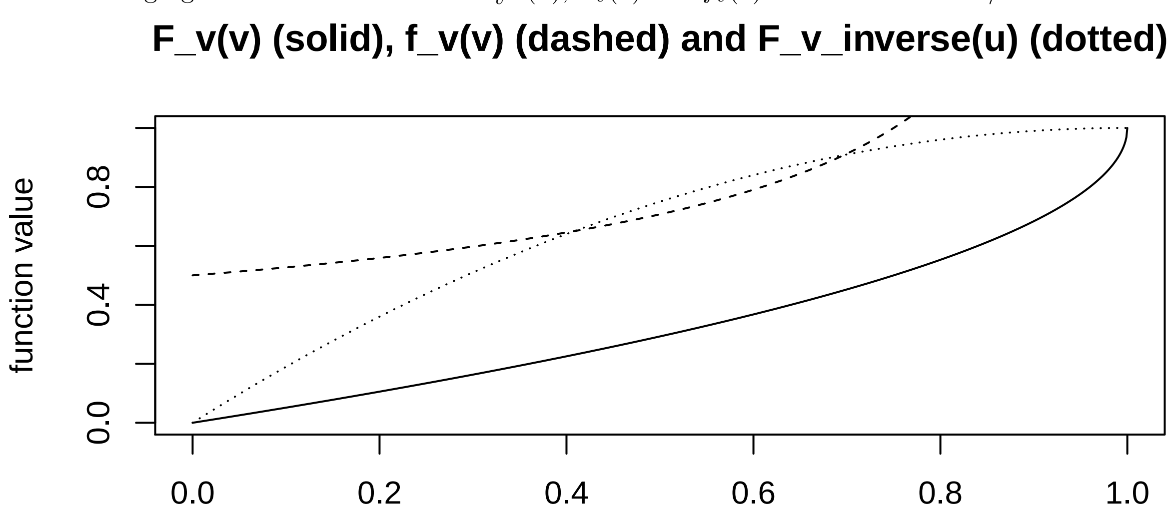
$$F_{e+}(x) = F_u(F_v(\Phi_+(x))), \tag{1}$$

where $F_v(v) \equiv 1 - (1 - v)^{1-\delta}$ and $F_u(u)$ is defined by $F(u) \equiv F_{e+}(F_v^{-1}(\Phi_+(u)))$. The point of the $F_v(v)$ function is to map a grid of evenly spaced points $u_i \in [0, 1]$ to a grid of points $v_i \in [0, 1]$ that are shifted towards one, which in turn correspond to points x_i that are well in the tails of the distribution with cdf $F_{e+}(x)$. For now, the values $\delta = 1/2$ (operations $(1 - v)^\delta$ and $(1 - u)^{1/(1-\delta)}$ are particularly efficient) and $\delta = 0$ (no transformation) make most sense.

Let $v = \Phi_+(x)$, $u = F_v(v) = F_v(\Phi_+(x))$. Then $x = \Phi_+^{-1}(v)$ and $v = F_v^{-1}(u)$ and $x = \Phi_+^{-1}(F_v^{-1}(u))$.

$$F_v(v) = 1 - (1 - v)^{1-\delta}, \quad f_v(v) = (1 - \delta)(1 - v)^{-\delta}, \quad f_v'(v) = \delta(1 - \delta)(1 - v)^{-\delta-1}.$$
$$F_v^{-1}(u) = 1 - (1 - u)^{1/(1-\delta)}$$

The following figure shows the functions $F_v^{-1}(u)$, $F_v(v)$ and $f_v(v)$ for the value $\delta = 1/2$.



Now taking the derivative of $F_{e+}(x)$ in (1) gives $f_{e+}(x) = f_u(F_v(\Phi_+(x)))f_v(\Phi_+(x))\phi_+(x)$. We approximate $F_{e+}(x)$ and $f_{e+}(x)$ by the functions $G(x)$ and $g(x)$ defined by

$$G(x) = G_u(F_v(\Phi_+(x))), \quad g(x) = g_u(F_v(\Phi_+(x)))f_v(\Phi_+(x))\phi_+(x),$$

where the $g_u(u)$ on $[0,1]$ is a fully normalized polynomial cubic spline density on $[0,1]$ and $G_u(u)$ is the corresponding cumulative distribution function. The value and first derivative of $g_u(u)$ agree with those of $f_u(u)$, up to a common multiplicative normalization constant, at the spline knots.

We can write (r for ratio)

$$r(x) \equiv \frac{f_{e+}(x)}{\phi_+(x)} = ce^{\varphi_e(x)} \cosh \varphi_o(x) = f_u(F_v(\Phi_+(x)))f_v(\Phi_+(x)),$$

$$\begin{aligned} r'(x) = \frac{d}{dx}\frac{f_{e+}(x)}{\phi_+(x)} &= ce^{\varphi_e(x)}\left[\cosh\varphi_o(x)\varphi_e'(x) + \sinh\varphi_o(x)\varphi_o'(x)\right] \\ &= f_u'(F_v(\Phi_+(x)))f_v^2(\Phi_+(x))\phi_+(x) + f_u(F_v(\Phi_+(x)))f_v'(\Phi_+(x))\phi_+(x), \end{aligned}$$

and then

$$\begin{aligned} f_u(F_v(\Phi_+(x))) &= \frac{r(x)}{f_v(\Phi_+(x))} \\ f_u'(F_v(\Phi_+(x))) &= \left[\frac{r'(x)}{\phi_+(x)} - f_u(F_v(\Phi_+(x)))f_v'(\Phi_+(x))\right] / f_v^2(\Phi_+(x)). \end{aligned}$$

Now we can evaluate $g_u(u_i) = f_u(u_i)$ at the spline knots $u_i = i/n$, $i = 0, 1, \ldots, n-1$. We first generate transformed knots $v_i = F_v^{-1}(u_i)$ then $x_i = \Phi_+^{-1}(v_i)$.

Then for $i = 0, 1, \ldots, n-1$,

$$\begin{aligned} g_u(u_i) &= \frac{e^{\varphi_e(x_i)}\cosh\varphi_o(x_i)}{f_v(v_i)}, \\ g_u'(u_i) &= \left[\frac{1}{\phi(x_i)}e^{\varphi_e(x_i)}[\varphi_e'(x_i)\cosh\varphi_o(x_i) + \varphi_o'(x_i)\sinh\varphi_o(x_i)] - g_u(u_i)f_v'(v_i)\right] / f_v^2(v_i). \end{aligned}$$

For $i = n$, $u_i = v_i = 1$ and $x_i = \infty$, so we need to take limits to find $f_u(1)$ and $f_u'(1)$. We can write $F_v(v) = 1 - Q_+(x)^{1-\delta}$, $f_v(v) = (1-\delta)Q_+(x)^{-\delta}$ and $f_v'(v) = \delta(1-\delta)Q_+(x)^{-\delta-1}$.

Then

$$f_u(u) = r(x)\frac{1}{1-\delta}Q_+(x)^{\delta}, \qquad f_u'(u) = r'(x)\frac{1}{(1-\delta)^2}\frac{Q_+(x)^{2\delta}}{\phi_+(x)} - r(x)\frac{\delta}{(1-\delta)^2}Q_+(x)^{2\delta-1}.$$

$Q(x)$ is bounded as follows (see Wikipedia page "Q-function"):

$$\frac{x\phi(x)}{1+x^2} < Q(x) < \frac{\phi(x)}{x},$$

with the same expression true for $\phi(x)$ and $Q(x)$ replaced by $\phi_+(x)$ and $Q_+(x)$. Therefore, for $\delta \geq 2$,

$$\lim_{x\to\infty}\frac{Q_+^{2\delta}(x)}{\phi_+(x)} = 0, \quad \lim_{x\to\infty} Q_+^{2\delta-1}(x) = 0.$$

If in addition, $r(x)$ and $r'(x)$ vanish at ∞, then $g_u(1) = f_u(1) = 0$ and $g_u'(1) = f_u'(1) = 0$. Note that if $\delta = 0$, so that $F(v) = v$ is "inert", $f_u(1) = 0$, but $f_u'(1) = \infty$ is quite possible: $\lim_{x\to\infty} f_{e+}(x)/\phi_+(x)^2 < \infty$ would be required to avoid this.

The values at the knot $i = 0$ are constant: $Q_+(0) = 1$, $r(0) = c$ and $r'(0) = 0$ and thus

$$f_u(0) = \frac{c}{1-\delta}, \quad f_u'(0) = \frac{-c\delta}{(1-\delta)^2}.$$

Note that the derivative of $\log f_u(u)$ at $u = 0$ is then $-\delta/(1-\delta) < 0$.

To draw a variate x from the distribution with cdf $G(x)$, you can draw $U \sim U(0,1)$, then compute $u = G_u^{-1}(U)$, then $v = F_v^{-1}(u)$ then $x = \Phi_+^{-1}(v) = \Phi^{-1}(F_v^{-1}(u))$.

Spline review

The cubic polynomial $p(t)$ on $[0,1]$ that satisfies $p(0) = p_0$, $p(1) = p_1$, $p'(0) = m_0$ and $p'(1) = m_1$ is $p(t) = p_0 h_{00}(t) + m_0 h_{10}(t) + p_1 h_{01}(t) + m_1 h_{11}(t)$, where

$$h_{00}(t) = 2t^3 - 3t^2 + 1 = (1+2t)(1-t)^2 = B_{0,3}(t) + B_{1,3}(t),$$

$$h_{10}(t) = t^3 - 2t^2 + t = t(1-t)^2 = \frac{1}{3}B_{1,3}(t),$$

$$h_{01}(t) = -2t^3 + 3t^2 = t^2(3-2t) = B_{2,3}(t) + B_{3,3}(t),$$

$$h_{11}(t) = t^3 - t^2 = t^2(t-1) = -\frac{1}{3}B_{2,3}(t).$$

Here $B_{k,3}(t) = \binom{3}{k} t^k (1-t)^{3-k}$, one of the four third degree Bernstein polynomials. Note that $B_{k,3}(t)$ is also the normalized Beta density with paramaters $\alpha = k+1$ and $\beta = 4-k$.

We can also write this cubic in standard form as $c_0 + c_1 t + c_2 t^2 + c_3 t^3$, where

$$c_0 = p_0, \quad c_1 = m_0, \quad c_2 = 3(p_1 - p_0) - (2m_0 + m_1), \quad c_3 = 2(p_0 - p_1) + (m_0 + m_1).$$

For the purposes of normalizing densities, we have

$$\int_0^1 p(t)\, dt = c_0 + c_1/2 + c_2/3 + c_3/4 = (p_0 + p_1)/2 + (m_0 - m_1)/12.$$

Second derivative:

$$p''(t) = p_0 h''_{00}(t) + m_0 h''_{10}(t) + p_1 h''_{01}(t) + m_1 h''_{11}(t),$$

where

$$h''_{00}(t) = 12t - 6, \quad h''_{10}(t) = 6t - 4, \quad h''_{01}(t) = -12t + 6, \quad h''_{11}(t) = 6t - 2.$$

So

$$p''(t) = [12(p_0 - p_1) + 6(m_0 + m_1)]t + [6(p_1 - p_0) - 4m_0 - 2m_1]$$

In particular, $p''(0) = 6(p_1 - p_0) - 4m_0 - 2m_1$, $p''(1) = 6(p_0 - p_1) + 2m_0 + 4m_1$, $p''(1/2) = 3m_0 + 3m_1 - 4m_0 - 2m_1 = m_1 - m_0$.

A flexible spline density on $[0,1]$

In this section we consider the problems of evaluating and simulating from a density $f_u(u)$ on $[0,1]$, where we are given the values $g_u(i/n)$ and derivatives $g'_u(i/n)$, $i = 0, \ldots, n$ of an unnormalized density $g_u(u) \propto f_u(u)$ at the knots $0, 1/n, 2/n, \ldots, (n-1)/n, 1$. The unnormalized density $g_u(u)$ is a cubic spline on $[0,1]$.

We need to map back and forth between a value $u \in [0,1]$ and a pair (i,t), where $i \in \{0, 1, \ldots, n-1\}$ is the index of the subinterval $[i/n, (i+1)/n]$ and $t \in [0,1]$ is the position in subinterval i, as a fraction of the distance between i/n and $(i+1)/n$. Thus

$$i = \lfloor nu \rfloor, \quad t = nu - \lfloor nu \rfloor.$$

The inverse mapping gives $u = (i+t)/n$.

For $i = 0, 1, \ldots, n$, $p_i = g_u(i/n)$ and $m_i = \frac{1}{n} g'_u(1/n)$.

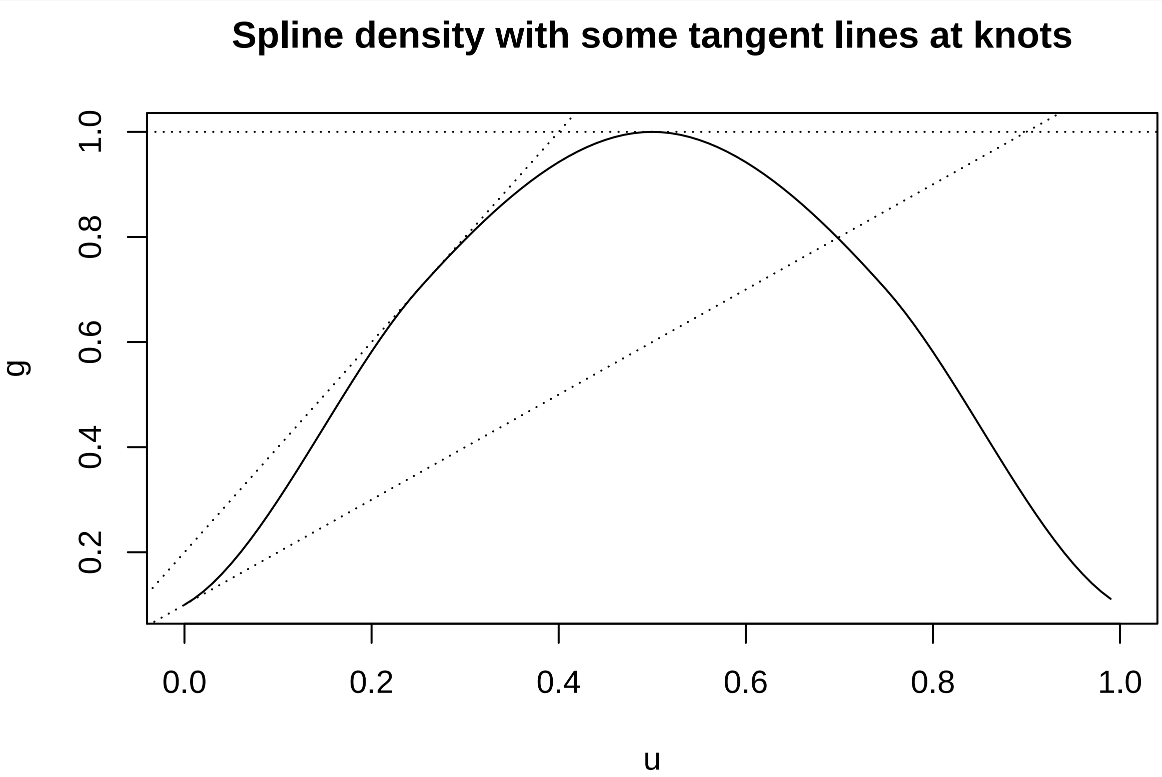
To evaluate $g_u(u)$, we compute i and t and then (direct evaluation of the second polynomial expression is more efficient than direct evaluation of the first)

$$g_u(u) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 = ((c_3 t + c_2)t + c_t)t + c_0,$$

where $c_0 = p_i$, $c_1 = m_i$, $c_2 = -3p_i - 2m_i + 3p_{i+1} - m_{i+1}$ and $c_3 = 2p_i + m_i - 2p_{i+1} + m_{i+1}$.

Here is R code providing a function evaluating $g_u(u)$, followed by code that sets up an example.

```
# Evaluate unnormalized density g(u), where u is the point of evaluation,
# and p and m are vectors of values and derivatives at knots.
# Derivatives should already be transformed to suit the interval [0,1].
g.eval = function(u, p, m) {
  n = length(p) - 1     # Number of knots
  i = floor(u*n)        # Subinterval index in {0,1,...,n-1}
  t = u*n - i           # index of u in subinterval [i/n, (i+1)/n]
  c0 = p[i+1]           # Subinterval spline
  c1 = m[i+1]
  c2 = -3*p[i+1] - 2*m[i+1] + 3*p[i+2] - m[i+2]
  c3 = 2*p[i+1] + m[i+1] - 2*p[i+2] + m[i+2]
  g = (((c3*t+c2)*t+c1)*t+c0)
}

# Set up values and [0,1]-normalized derivatives at knots.
n = 4
p = c(0.1, 0.7, 1.0, 0.7, 0.1)
m = c(1/n, 2/n, 0/n, -2/n, -1/n)

# Plot unnormalized density
u = seq(0,1,by=0.01)
g = lapply(u, g.eval, p, m)
plot(u, g, type='l', main='Spline density with some tangent lines at knots')
abline(a=0.1, b=1, lt='dotted')
abline(a=0.2, b=2, lt='dotted')
abline(h=1, lt='dotted')
```



We now move on to drawing a variate from the spline density and evaluating the fully normalized density. The total area A under the cubic spline $g_u(u)$ is

$$A = \frac{1}{n}\left[\frac{1}{2}p_0 + p_1 + \ldots + p_{n-1} + \frac{1}{2}p_n + \frac{1}{12}(m_0 - m_n)\right]$$

First draw a knot $k^* \in \{0, \ldots, n\}$ with probabilities $\pi_0 = p_0/2 + m_0/12$, $\pi_n = p_n/2 + m_n/12$, and $\pi_i = p_i$ for $i = 1, \ldots, n-1$.

If $k^* = 0$, draw

$$t \sim \frac{3p_0}{6p_0 + m_0}\mathrm{Be}(1,4) + \frac{3p_0 + m_0}{6p_0 + m_0}\mathrm{Be}(2,3).$$

If $k^* = n$, draw

$$t \sim \frac{3p_n}{6p_n - m_n}\mathrm{Be}(4,1) + \frac{3p_n - m_n}{6p_n - m_n}\mathrm{Be}(3,2).$$

If $0 < k^* < n$, draw

$$t \sim \frac{1}{2}\mathrm{Be}(1,4) + \frac{1}{2}\mathrm{Be}(2,3),$$

and then with probability

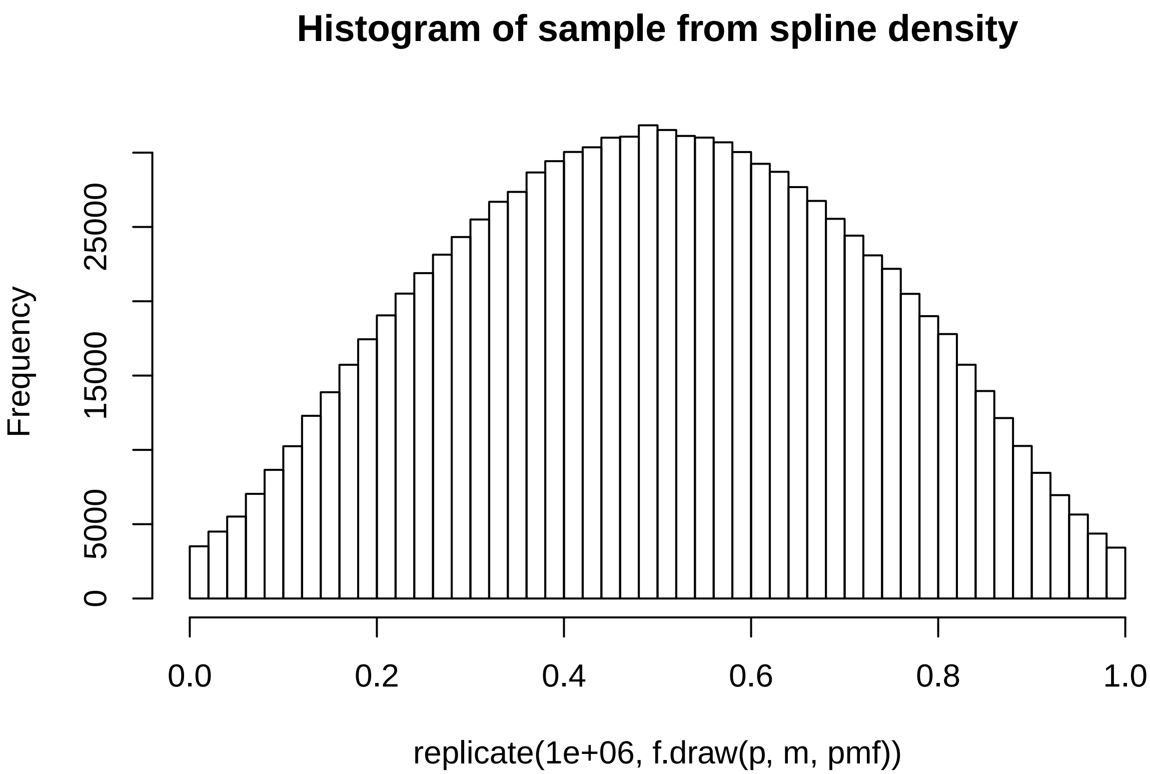
$$\frac{p_i(1+2t) + m_i t}{2p_i(1+2t)} = \frac{p_i + (2p_i + m_i)t}{2p_i + 4p_i t},$$

set $i = k^*$; and with complementary probability set $i = k^* - 1$ and $t = 1 - t$. WJM Aside: postive t weight is $p_i(1+2t) + m_i t$ and negative weight is $p_i(1+2t) - m_i t$.

Then set $u = (i + t)/n$.

```
f.draw = function(p, m, pmf) {
  n = length(p) - 1
  k = sample.int(n+1, 1, prob=pmf)
  if (k==1) {
    i = 1
    if (runif(1) < 3*p[1]/(6*p[1]+m[1]))
      t = rbeta(1, 1, 4)
    else
      t = rbeta(1, 2, 3)
  }
  else if (k==n+1) {
    i = n
    if (runif(1) < 3*p[n+1]/(6*p[n+1]-m[n+1]))
      t = rbeta(1, 4, 1)
    else
      t = rbeta(1, 3, 2)
  }
  else { # WJM: Combine two if clauses to save one runif() call?
    i = k
    t = runif(1)
    if (runif(1) < t*t*(3-2*t))
      t = 1-t
    if (runif(1) > (p[i]*(1+2*t) + m[i]*t) / (2*p[i]*(1+2*t))) {
      i = k-1
      t = 1-t
    }
  }
  u = (i-1+t)/n
}
```

```r
# Set up vectors p, m and F defining spline on $[0,1]$
pmf = p
pmf[1] = p[1]/2 + m[1]/12
pmf[5] = p[5]/2 - m[5]/12

hist(replicate(1000000, f.draw(p, m, pmf)), 40, main='Histogram of sample from spline density')
```

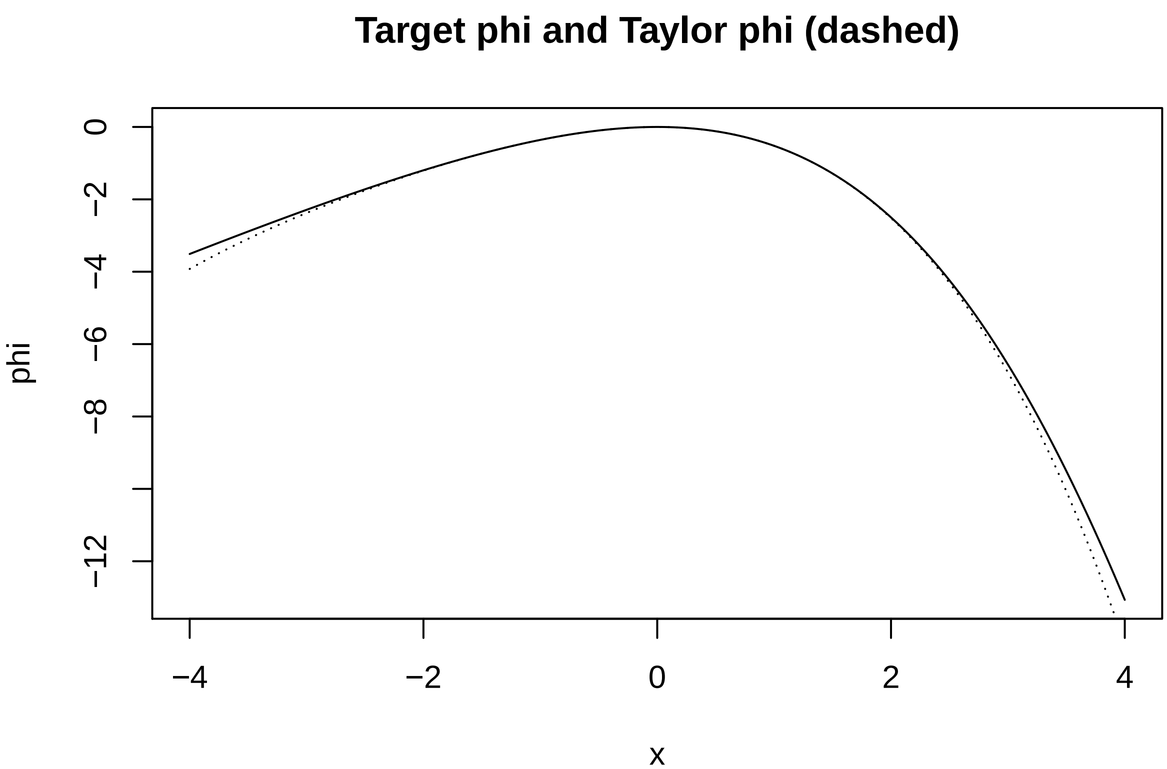



```r
h.eval.05 = function(u, p, m, p05, m05) {
  n = length(p) - 1     # Number of knots
  i = floor(u*n)        # Subinterval index in {0,1,...,n-1}
  t = u*n - i           # index of u in subinterval [i/2, (i+1)/n]
  c0 = p[i+1]           # Subinterval spline
  c1 = m[i+1]
  c2 = -3*p[i+1] - 2*m[i+1] + 3*p[i+2] - m[i+2]
  c3 = 2*p[i+1] + m[i+1] - 2*p[i+2] + m[i+2]
  if (i==n-1) {
    p.res = max(p05 - 0.5*p[i+1] - 0.125*m[i+1], 0)
    m.res = m05 + 1.5*p[i+1] + 0.25*m[i+1]
    a = p.res + 0.5 * m.res
    b = p.res - 0.5 * m.res
    new.term = 16*t^2*(1-t)^2 * (a*t + b*(1-t))
  } else { new.term = 0 }
  f = (((c3*t+c2)*t+c1)*t+c0) + new.term
}
```

Approximation testing ground

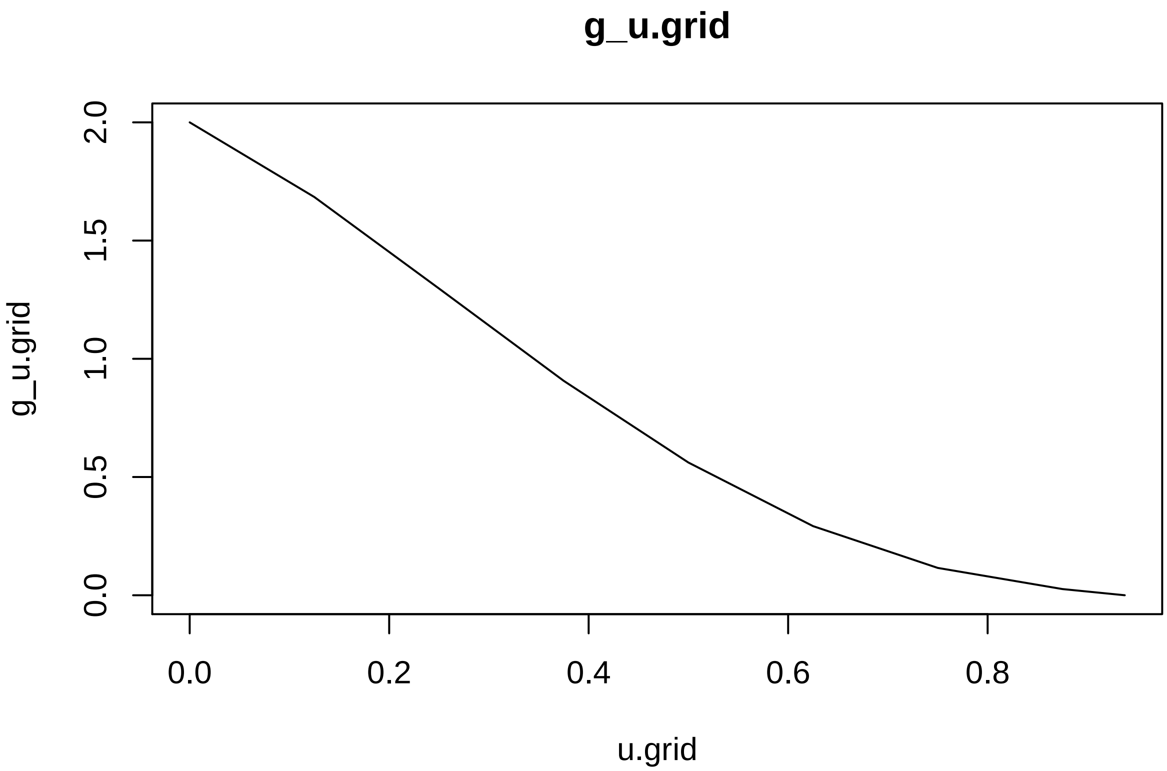
```
library(knitr)
# Step one: set up a true posterior distribution (of x) as a test case.
#
# The prior is x ~ N(mu, sigma^2). Let omega = 1/sigma^2
# The sample is Y=(Y_1,...,Y_n), with, for i=1,...,n,
#   Y_i ~ iid Po(theta e^x)
#   or Y_i ~ iid GaPo(r, (theta/r) e^x),
# Note that the Poisson case is the limit of the Gamma Poisson case as r->infty.
#
# In both cases, the sufficient statistic is (n, y_bar), where y_bar is the
# sample mean.
# The parameter values omega, theta and r, and the sufficient stastistic
# (n, y_bar) are specified here below.
# The value of mu is set to make the posterior mode of x equal to zero.
n = 1
y_bar = 1
r = 19 # Inf for Poisson, finite for Gamma-Poisson
theta = 2
omega = 2; sigma = omega^-0.5

# Step two: Compute derivatives of phi = log f(x|y) - log f_N(0, omega^{-1}),
# normalized so that phi(0) = 0
if (is.infinite(r)) {
  # Value of mu parameter giving a mode at x=0
  mu = n*(theta-y_bar)/omega
  # Derivatives of phi at zero, from 2nd to 5th
  h_2 = -n*theta
  h_3 = -n*theta
  h_4 = -n*theta
  h_5 = -n*theta
} else {
  # Value of theta parameter giving a mode at x=0
  mu = n*r*(theta-y_bar)/(omega*(r+theta))
  th_by_r = theta/r
  # Derivatives of phi at zero, from 2nd to 5th
  h_2 = -n*(r+y_bar)*th_by_r / (1+th_by_r)^2
  h_3 = -n*(r+y_bar)*(th_by_r - th_by_r^2) / (1+th_by_r)^3
  h_4 = -n*(r+y_bar)*(th_by_r - 4*th_by_r^2 + th_by_r^3) / (1+th_by_r)^4
  h_5 = -n*(r+y_bar)*(th_by_r - 11*th_by_r^2 + 11*th_by_r^3 - th_by_r^4) / (1+th_by_r)^5
}

# Normalized values of h_2
a_2 = h_2/omega; a_3 = h_3/omega^{1.5}; a_4 = h_4/omega^2; a_5 = h_5/omega^{2.5}

# Step three: Compute true phi and its odd and even parts on a fine grid, as an
# illustration.
c = n*y_bar + omega*mu
x = seq(-4, 4, length.out=2001)
x.plus = x[x>=0]
z = x*sigma
if (is.infinite(r)) {
```

```r
  phi = c*z - n*theta*(exp(z)-1)
  phi_m = -c*z - n*theta*(exp(-z)-1) # phi(-x)
  phi_o = c*z - n*theta*sinh(z)      # odd part of phi
  phi_e = n*theta*(1-cosh(z))        # even part of phi
} else {
  phi = c*z - n*(r+y_bar)*log((1 + th_by_r * exp(z))/(1 + th_by_r))
  phi_m = -c*z - n*(r+y_bar)*log((1 + th_by_r * exp(-z))/(1 + th_by_r))
  phi_o = c*z - 0.5*n*(r+y_bar)*log((1 + th_by_r * exp(z))/(1 + th_by_r * exp(-z)))
  phi_e = -0.5*n*(r+y_bar)*(log(1 + th_by_r^2 + 2*th_by_r*cosh(z)) - 2*log(1 + th_by_r))
}

# Plot phi (target) and 5th order Taylor expansion phi_h around x=0
plot(x, phi, type='l', main='Target phi and Taylor phi (dashed)', xlab='x', ylab='phi')
phi_h = a_2*x^2/2 + a_3*x^3/6 + a_4*x^4/24 + a_5*x^5/120
phi_h_e = a_2*x^2/2 + a_4*x^4/24
phi_h_o = a_3*x^3/6 + a_5*x^5/120
lines(x, phi_h, lt='dotted')
```

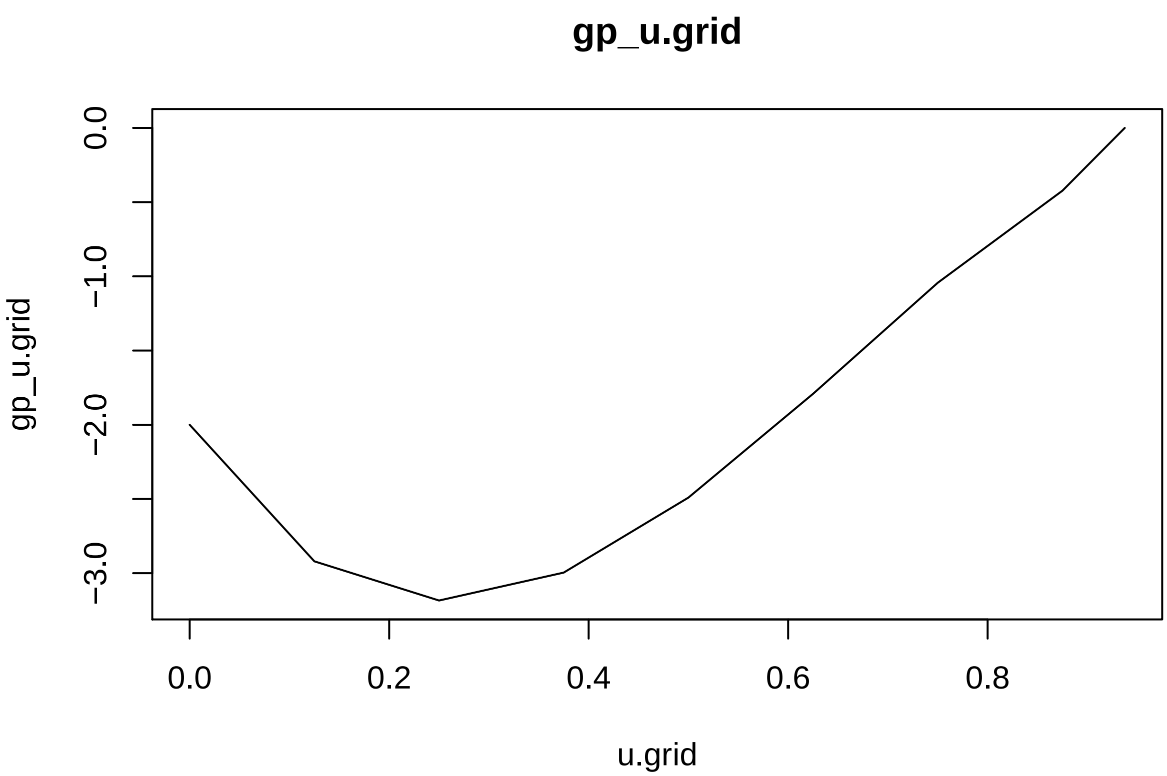



```r
# Step four: Select knots in u space at which to evaluate f_e and f_e',
# compute v knots and x knots
knot.n = 8
delta = 0.5 # Needs to be in [0,1] (delta=0 has no effect)
u.grid = seq(0, 1, length.out=knot.n+1)
u.grid[knot.n+1] = 1 - 0.5/knot.n
v.grid = 1-(1-u.grid)^(1/(1-delta))
x.grid = qnorm(0.5+0.5*v.grid)
z.grid = sigma*x.grid

# Step five: compute true target and Taylor approximation at knots
```

```r
if (is.infinite(r)) {
  phi.grid = c*z.grid - n*theta*(exp(z.grid)-1)
  phi_m.grid = -c*z.grid - n*theta*(exp(-z.grid)-1) # phi(-x)
  phi_o.grid = c*z.grid - n*theta*sinh(z.grid)      # odd part of phi
  phi_e.grid = n*theta*(1-cosh(z.grid))             # even part of phi
} else {
  phi.grid = c*z.grid - n*(r+y_bar)*log((1 + th_by_r * exp(z.grid))/(1 + th_by_r))
  phi_m.grid = -c*z.grid - n*(r+y_bar)*log((1 + th_by_r * exp(-z.grid))/(1 + th_by_r))
  phi_o.grid = c*z.grid - 0.5*n*(r+y_bar)*log((1 + th_by_r * exp(z.grid))/(1 + th_by_r * exp(-z.grid)))
  phi_e.grid = -0.5*n*(r+y_bar)*(log(1 + th_by_r^2 + 2*th_by_r*cosh(z.grid)) - 2*log(1 + th_by_r))
}
true.f_u.grid = exp(phi_e.grid) * cosh(phi_o.grid)

phi_h.e.grid = a_2*x.grid^2/2 + a_4*x.grid^4/24
phi_h.o.grid = a_3*x.grid^3/6 + a_5*x.grid^5/120
phi_h.e.p.grid = a_2*x.grid + a_4*x.grid^3/6
phi_h.o.p.grid = a_3*x.grid^2/2 + a_5*x.grid^4/24

# Evaluate f_e(x)/phi(x), its derivative and f_v(v) and f_v'(v) at knots
r.grid = exp(phi_h.e.grid) * cosh(phi_h.o.grid)
rp.grid = r.grid * phi_h.e.p.grid + exp(phi_h.e.grid) * sinh(phi_h.o.grid) * phi_h.o.p.grid
rp.grid = rp.grid / (2*dnorm(x.grid))
f_v.grid = (1-delta)*(1-v.grid)^-delta
f_vp.grid = delta*(1-delta)*(1-v.grid)^(-delta-1)
g_u.grid = r.grid / f_v.grid
gp_u.grid = (rp.grid - g_u.grid * f_vp.grid)/(f_v.grid)^2

p05 = g_u.grid[knot.n+1]; m05 = gp_u.grid[knot.n+1]
g_u.grid[knot.n+1] = 0
gp_u.grid[knot.n+1] = 0
plot(u.grid, g_u.grid, type='l', main='g_u.grid')
```




```r
plot(u.grid, gp_u.grid, type='l', main='gp_u.grid')
```

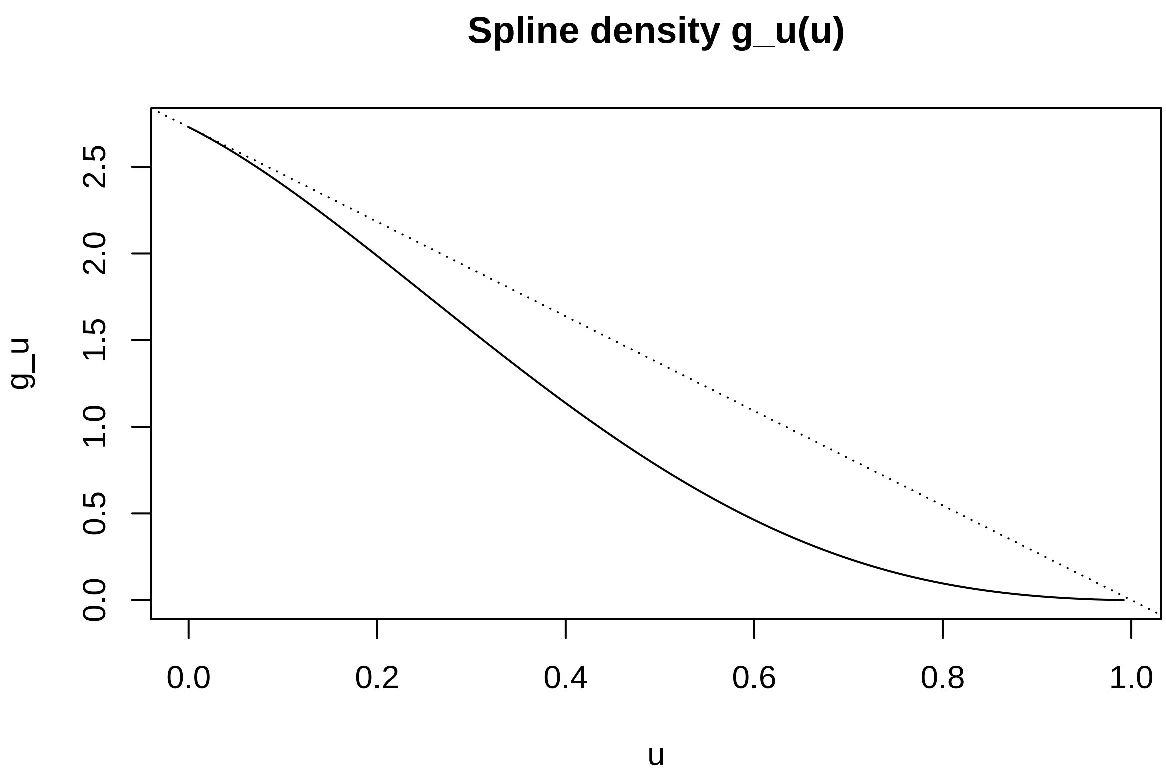



```r
m.grid = gp_u.grid / knot.n; m05 = m05 / knot.n
table = data.frame(u=u.grid, g_u=g_u.grid, gp_u=gp_u.grid, m=m.grid)
kable(table, caption='Note that gp_u is the correct derivative, m is the scaled derivative for spline e
```

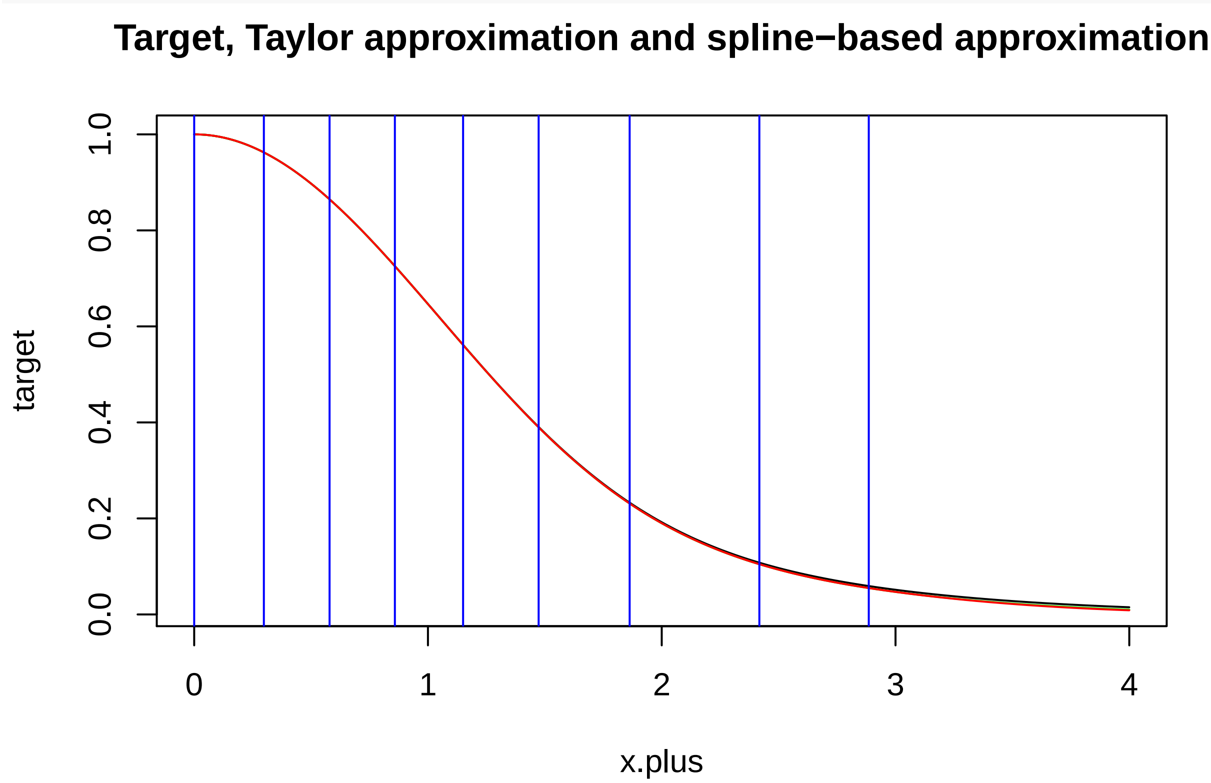
Table 1: Note that gp__u is the correct derivative, m is the scaled derivative for spline evaluation

u	g_u	gp__u	m
0.0000	2.0000000	-2.0000000	-0.2500000
0.1250	1.6841537	-2.9200544	-0.3650068
0.2500	1.2970810	-3.1840953	-0.3980119
0.3750	0.9068828	-2.9959841	-0.3744980
0.5000	0.5612876	-2.4906906	-0.3113363
0.6250	0.2923261	-1.7914069	-0.2239259
0.7500	0.1155287	-1.0432279	-0.1304035
0.8750	0.0261627	-0.4228145	-0.0528518
0.9375	0.0000000	0.0000000	0.0000000

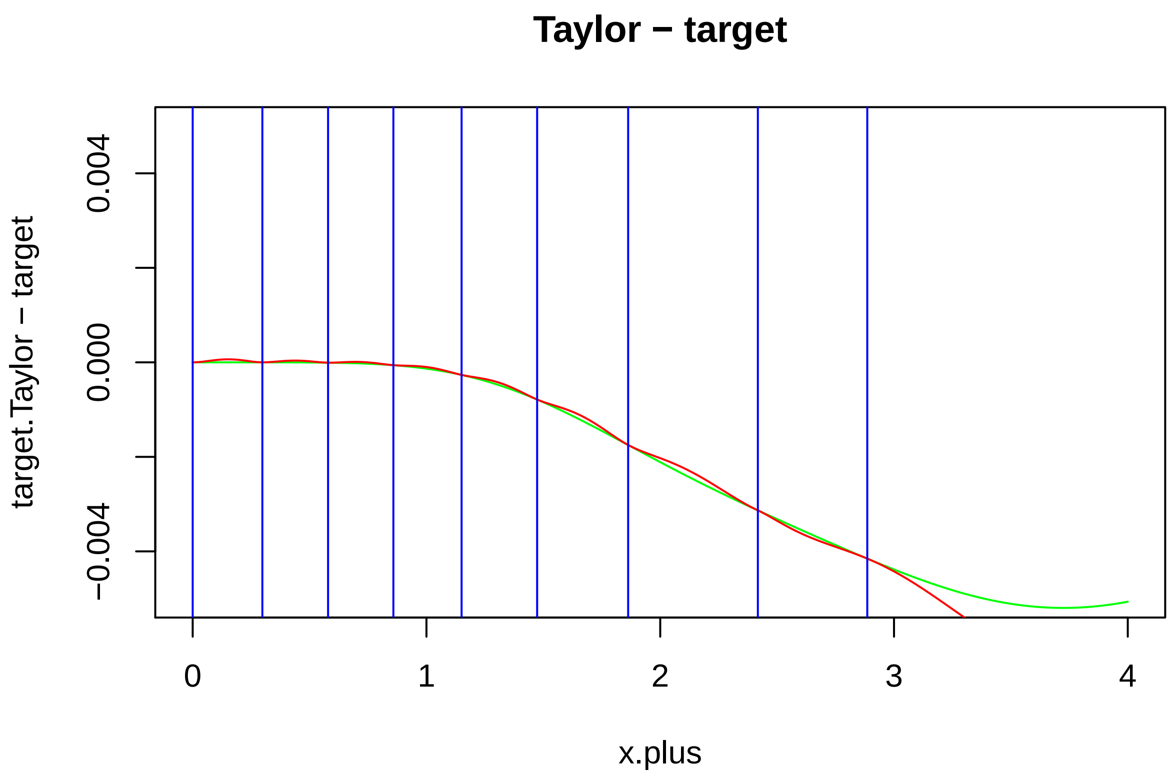
```
# Normalize g_u values so that cubic spline integrates to one.
c = sum(g_u.grid) - 0.5*(g_u.grid[1] + g_u.grid[knot.n+1]) + (m.grid[1]-m.grid[knot.n+1])/12
c = c/knot.n
g_u.grid = g_u.grid/c
m.grid = m.grid/c
p05 = p05/c
m05 = m05/c

# Compute on fine grid
v = 2*pnorm(x.plus)-1
u = 1-(1-v)^(1-delta)
g_u = sapply(u, h.eval.05, g_u.grid, m.grid, p05, m05)
f_v = (1-delta)*(1-v)^-delta
target = exp(phi_e[x>=0]) * cosh(phi_o[x>=0])
target.Taylor = exp(phi_h_e[x>=0]) * cosh(phi_h_o[x>=0])
approx = g_u * f_v / (g_u[1] * (1-delta))

plot(u, g_u, type='l', main='Spline density g_u(u)')
abline(a=g_u[1], b=-g_u[1], lt='dotted')
```



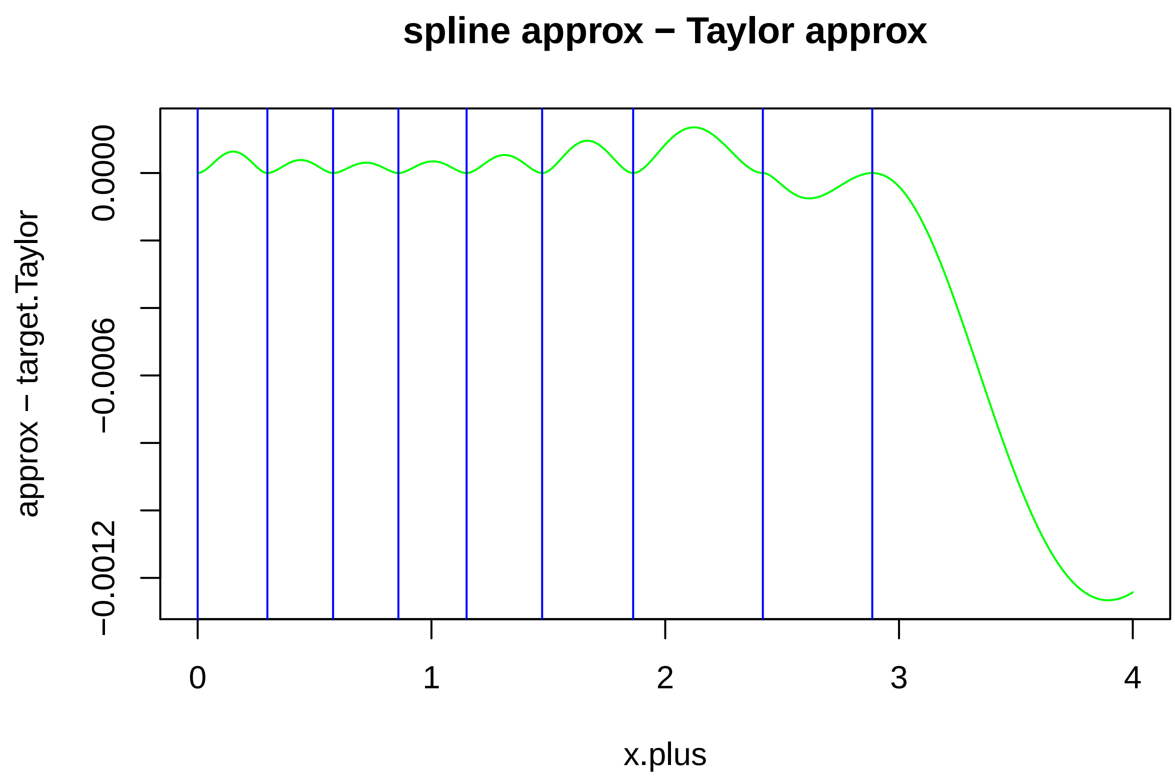

```r
plot(x.plus, target, type='l', main='Target, Taylor approximation and spline-based approximation')
lines(x.plus, target.Taylor, col='green')
lines(x.plus, approx, col='red')
abline(v=x.grid, col='blue')
```



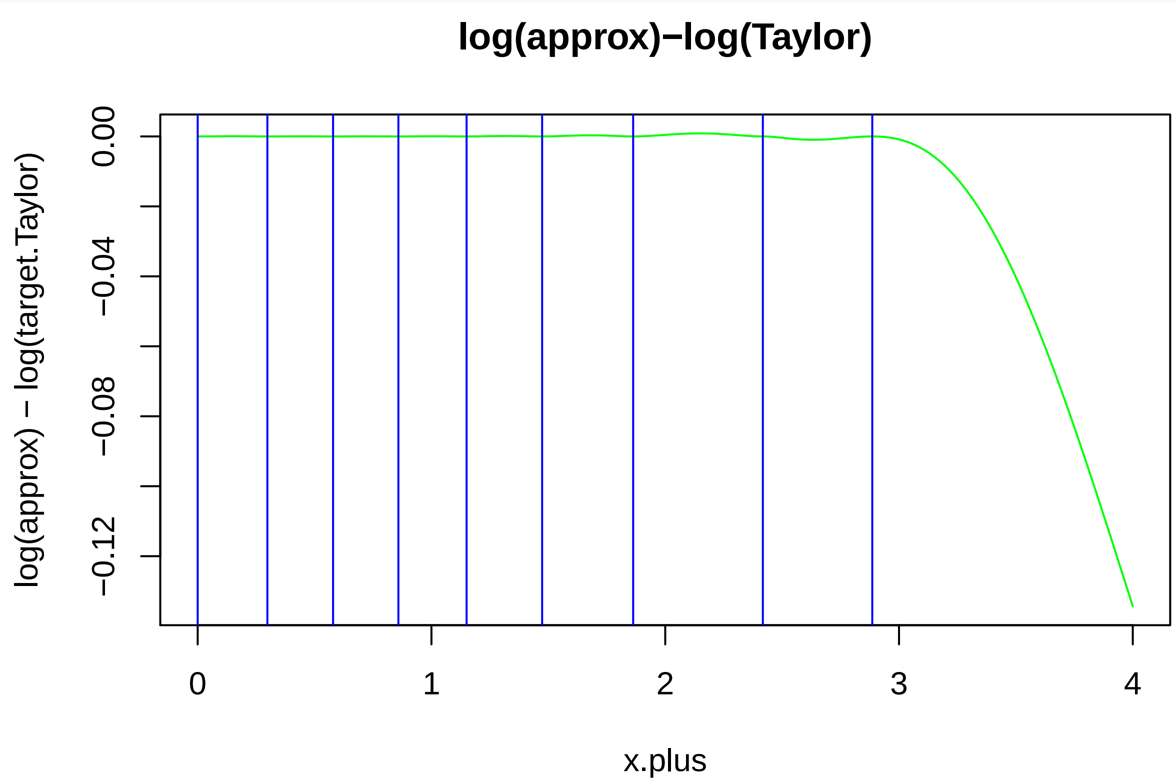
```r
plot(x.plus, target.Taylor - target, type='l', col='green', ylim=c(-0.005,0.005), main='Taylor - target
lines(x.plus, approx - target, col='red')
abline(v=x.grid, col='blue')
```



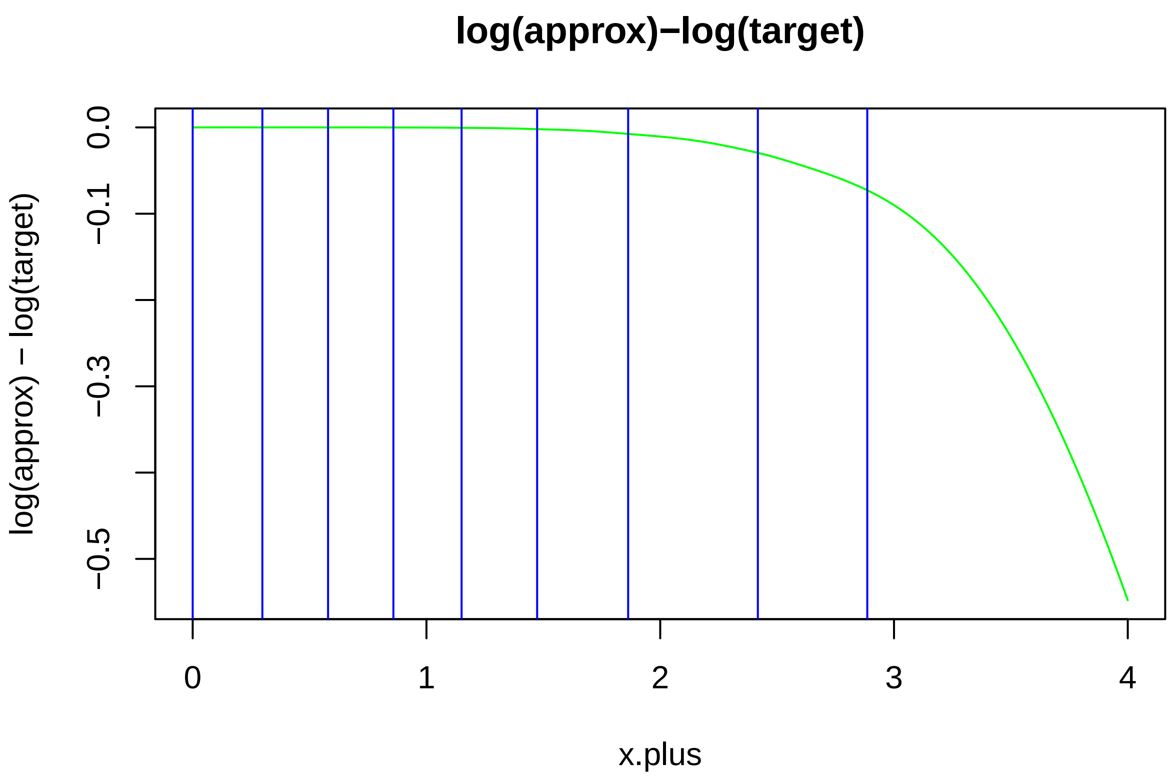

```r
plot(x.plus, approx - target.Taylor, type='l', col='green', main='spline approx - Taylor approx')
abline(v=x.grid, col='blue')
```




```r
plot(x.plus, log(approx) - log(target.Taylor), type='l', col='green', main='log(approx)-log(Taylor)')
abline(v=x.grid, col='blue')
```




```r
plot(x.plus, log(approx) - log(target), type='l', col='green', main='log(approx)-log(target)')
abline(v=x.grid, col='blue')
```




```
phi01 = dnorm(x.plus)
Ew = sum(phi01 * target^2/approx)/sum(phi01 * target)
Ew2 = sum(phi01 * target^3/approx^2)/sum(phi01 * target)
var_w = Ew2 - Ew^2
sd_w = sqrt(var_w)

Ew.approx = sum(phi01 * target.Taylor^2/approx)/sum(phi01 * target.Taylor)
Ew2.approx = sum(phi01 * target.Taylor^3/approx^2)/sum(phi01 * target.Taylor)
var_w.approx = Ew2.approx - Ew.approx^2
sd_w.approx = sqrt(var_w.approx)
```

Standard deviation of Metropolis-Hastings numerator using target is 0.0032853. Same using Taylor approximation of target is 2.6924076×10^{-4}.

Coding steps

Precomputation of quantities only depending on K and δ

1. $u_k = k/K$, $k = 0, 1, \ldots, K-1$, $u_K = 1 - k/(2K)$.
2. $v_k = F_v^{-1}(u_k) = 1 - (1-u_k)^2$, $k = 0, 1, \ldots, K$.
3. $z_k = \Phi_+^{-1}(v_k)$, $k = 0, 1, \ldots, K$.
4. $f_v(v_k) = (1-v_k)^{-1/2}/2$, $k = 0, 1, \ldots, K$.
5. $f_v'(v_k) = (1-v_k)^{-3/2}/4$, $k = 0, 1, \ldots, K$.
6. $z_k^p/p!$ and $(-z_k)^p/p!$, $p = 1, \ldots, 5$ and $k = 0, 1, \ldots, K$.

Initial computation, whether drawing or just evaluating

These quantities depend on ω, $h_2, \ldots, h_5$ and pre-computed quantities.

1. $\sigma = \omega^{-1/2}$, σ^p for $p = 1, 2, 3, 4, 5$.
2. $a_p = h_p \sigma^p$ for $p = 2, 3, 4, 5$.
3. $\varphi(z_k) = \sum_{p=2}^5 z_k^p/p!$, $\varphi(-z_i) = \sum_{p=2}^5 z_k^p/p!$, $k = 0, \ldots, K$.
4. $p_0, \ldots, p_K$, m_0 and m_K

$$p_0 = 2, \quad p_K = 0, \quad m_0 = -2/K, \quad m_K = 0,$$

$$p_k = \frac{\exp(\varphi(z_k)) + \exp(\varphi(-z_k))}{2 f_v(v_k)}, \quad k = 1, \ldots, K-1.$$

5.

Computation if drawing

1. Draw k^* from discrete distribution.
2. m_{k^*}

$$m_{k^*} = \frac{1}{K f_v^2(v_{k^*})} \left[\frac{\exp(\varphi(z_{k^\star}))\varphi'(z_{k^\star}) + \exp(\varphi(-z_{k^\star}))\varphi'(-z_{k^\star})}{2} - p_{k^*} f_v'(v_{k^*}) \right]. \tag{2}$$

3. Draw t and adjust k and t if necessary.
4. $u = (k+t)/K$.
5. $v = F_v^{-1}(u)$.
6. $z = \Phi_+^{-1}(v)$ or $z = \Phi_+^{-1}(u)$.

Computation if only evaluating

1. $v = \Phi_+(z)$.
2. $u = F_v(v) = 1 - (1-v)^{1-\delta}$.
3. $k = \lfloor K \cdot u \rfloor$, $t = K \cdot u - \lfloor K \cdot u \rfloor$.
4. Evaluate m_k and m_{k+1} as in (2).

Final computation

1. Evaluate $f_u(u)$.
2. Evaluate $f_v(v)$.
3. Evaluate $\varphi_+(z)$.
4. Reflection sampling
5. Scaling

Decisions to make, based on a_2, a_3, a_4:

1. Whether to do polynomial in $e^{-ax^2/2}$ times $e^{-bx^2/2}$ or spline.
2. $\delta = 1/2$ or $\delta = 0$.
3. Evaluate at 2nd and perhaps 3rd point.
4. Number of knots.
5. What to do about $\dot{\mu}$ part.

Brainstorm evaluation

1. Special attention to functions of e^{-x} in derivatives of $\log f(y_t|x_t)$.
2. Faa di Bruno with 5 derivatives of f but many more for $g = e^x$.
3. Keep values and derivatives from previous evaluations - even if just x_t°.
4. Return limits of ψ, ψ' or other tail information.
5. Figure out $f''(u)$ and how it changes discontinuously at knots (diagnostic of insufficient number of bins)
6. Do three evaluations x_t°, $x_t^\circ \pm 1$ as a matter of course.
7. Endogenous number of function evaluations, number of knots, based on
    a. relative size of higher order terms compared with lower order terms
    b. When 1st or second derivative becomes positive.
    c. precomputed thresholds
    d. h_2 relative to ω is important
8. Evaluate t and $(1-t)$ powers at the same time.

Illustration using Gaussian SV model

From that model, $\psi''(x) = \psi^{(4)}(x) = -e^{-x}y_t^2 \equiv -c$, $\phi'''(x) = \psi^{(5)}(x) = c$. Then $h_2 = h_4 = -c$, $h_3 = h_4 = c$. Then $a_2 = -c/\omega = c\sigma^2$, $a_3 = c/\omega^{3/2} = c\sigma^3$, $a_4 = -c/\omega^2 = c\sigma^4$ and $a_5 = c/\omega^{5/2} = c\sigma^5$.

There are two dimensions on which to make a decision about what approximation to use: the level of $a_2, \ldots, a_5$ and how quickly they decay.

Brainstorm rapid quick draws

1. Scale mixture of normals or 3-component mixture of normals with symmetry around zero.
2. Single polynomial $1 + ax^2 + bx^4 + cx^6$ on $[0, 1]$ for f_u.
3. $f_u = e^{-ax^2}$ on $[0, 1]$.
4. Simple draws I already have in HESSIAN method paper.
5. Simple draws I did a year ago.

Brainstorm tail information in $\log f(y_t|x_t)$

1. Even part of $\log f(y_t|x_t)$ is not that useful, even part of $f(y_t|x_t)$ more realistic.
2. Finding the largest c such that $c|x| + \log f(y_t|x_t)$ has limits at right and left that are not infinite seems promising.
3. Two functions $\psi_t(x_t) = \log f(y_t|x_t)$:

$$\psi_1(x) = -\frac{1}{2}[\log 2\pi + x + e^{-x}y_t^2], \quad \psi_1'(x) = -\frac{1}{2}(1 - e^{-x}y_t^2).$$

$$\lim_{x\to\infty} \psi_1'(x) = -1/2, \quad \lim_{x\to\infty} \psi_1(x) - \psi_1'(x) = ?, \quad \lim_{x\to-\infty} \psi_1'(x) = -\infty.$$

$$\psi_2(x) = c - \frac{1}{2}[x_t + (\nu+1)\log(1+z)], \quad \psi_2'(x) = -\frac{1}{2} + \frac{\nu+1}{2}\frac{z}{1+z},$$

where $z = e^{-x}y_t^2/\nu$.

$$\lim_{x\to\infty} \psi_2'(x) = \nu/2, \quad \lim_{x\to-\infty} \psi_2'(x) = -1/2.$$

Probability of negative sign

Probability of changing sign is

$$\frac{e^{\varphi(-x)}}{e^{\varphi(-x)} + e^{\varphi(x)}} = \frac{e^{\varphi_o(-x)}}{e^{\varphi_o(-x)} + e^{\varphi_o(x)}} = \frac{1}{1 + e^{2\varphi_o(x)}}$$