QuantMinds International

Intercontinential O2 London, November 18, 2024

Rough volatility workshop

Lecture 4: Computation

Jim Gatheral
Department of Mathematics

Baruch College
The City University of New York

Outline of Lecture 4

- Rational approximation of rough Heston
- Smile plotting and parameter sensitivities
- The HQE scheme

The rough Heston model with $\lambda \geq 0$

- As shown in Lecture 2, in the case $\lambda \geq 0$, the rough Heston model may be written in forward variance form as

$$
\begin{aligned}
\frac{dS_t}{S_t} &= \sqrt{V_t}\left\{\rho\, dW_t + \sqrt{1-\rho^2}\, dW_t^{\perp}\right\}\\
d\xi_t(u) &= \sqrt{V_t}\,\kappa(u-t)\, dW_t, \quad u \geq t
\end{aligned}
$$

  where $\xi_t(u) = \mathbb{E}_t\left[V_u\right], u > t$ is the forward variance curve, $\frac{1}{2} < \alpha = H + \frac{1}{2} \leq 1$, and the kernel κ is given by

$$
\kappa(x) = \nu\, x^{\alpha-1}\, E_{\alpha,\alpha}(-\lambda\, x^{\alpha}),
$$

  where $E_{\alpha,\alpha}(\cdot)$ denotes the generalized Mittag-Leffler function.

The convolution Riccati equation

- Let $X = \log S$ and $X_{t,T} := X_T - X_t$.
- In Lecture 3, we showed that that affine forward variance (AFV) models have a cumulant generating function (CGF) of the form

$$\varphi_t\,(T;a) := \log \mathbb{E}_t\left[e^{\mathrm{i}a\,X_{t,T}}\right] = \int_t^T \xi_t(s)\,g(T-s;a)\,ds.$$

- $g(t;a)$ satisfies the convolution Riccati equation

$$g = -\tfrac{1}{2}\,a\,(a+\mathrm{i}) + \rho\,a\,\mathrm{i}\,(\kappa \star g) + \tfrac{1}{2}\,(\kappa \star g)^2,$$

where $(\kappa \star g)(t;a) := \int_0^t \kappa(t-s)\,g(s;a)\,ds$.

The rough Heston fractional ODE

- Let D^α and $I^{1-\alpha}$ represent respectively fractional differential and integral operators.
- In the rough Heston case, the convolution Riccati equation may be re-expressed as a fractional ODE.
- As originally proved in [Gatheral and Radoičić][6][7], we have:

> **Lemma 1.1 of [Gatheral and Radoičić][7]**
>
> Let $\kappa(\tau) = \nu\,\tau^{\alpha-1}\,E_{\alpha,\alpha}(-\lambda\,\tau^\alpha)$ and $h(t;a) = \frac{1}{\nu}\,(\kappa \star g)(t;a)$.
> Then h satisfies the fractional ODE
>
> $$\begin{aligned} D^\alpha h(t;a) &= -\frac{1}{2}\,a\,(a+\mathrm{i}) + (\mathrm{i}\,\rho\,\nu\,a - \lambda)\,h(t;a) + \frac{1}{2}\,\nu^2\,h^2(t;a);\\ I^{1-\alpha}h(t;a) &= 0. \end{aligned}$$

Solving the fractional ODE

- There exist a number of standard numerical techniques, such as the Adams scheme, for solving fractional differential equations such as the rough Heston fractional Riccati equation.
  - These techniques are all slow!

- [Gatheral and Radoičić][6][7] showed how to approximate the solution of the fractional ODE using a rational (Padé approximation).

- The idea is to paste together short- and long-time expansions of the solution.
- This approximation solution is just as fast as the classical Heston solution and appears to be more accurate than the Adams scheme for any reasonable number of time steps!

- As pointed out in [Baschetti et al.][3] for example, such rational approximations are extremely fast to compute relative to the alternatives, enabling efficient calibration of the rough Heston model.

The Lewis formula

- Given an approximate solution to the convolution Riccati Equation , an accurate approximation to the CGF may be easily computed.

- European option prices may then be obtained using the Lewis formula[Lewis][9]:

$$C(S,K,T) = S - \sqrt{SK}\,\frac{1}{\pi}\int_0^\infty \frac{du}{a^2+\frac{1}{4}}\,\mathrm{Re}\left[e^{-iak}\varphi_t\left(T;a-i/2\right)\right], \tag{1}$$

where S is the current stock price, K the strike price and T expiration.

- Implied volatilities may be computed by numerical inversion of the Black-Scholes formula.

- For option pricing with the Lewis formula, we need only find a good approximation for $a \in \mathcal{A}$ with

$$\mathcal{A} = \{z \in \mathbb{C} : \Re(z) \geq 0, -1 \leq \Im(z) \leq 0\} \tag{2}$$

where $\Re$ and $\Im$ denote real and imaginary parts respectively.

Solving the rough Heston Riccati equation for short times

- First, we derive a short-time expansion of the solution $h(t;a)$ of the fractional ODE.

- Consider the small t ansatz

$$h(t;a) = \sum_{j=1}^\infty b_j\, t^{j\,\alpha}. \tag{3}$$

- Then,

$$
\begin{aligned}
D^\alpha h &= \sum_{j=1}^{\infty} b_j \, \frac{\Gamma(1+j\,\alpha)}{\Gamma(1+(j-1)\,\alpha)} \, t^{(j-1)\alpha} \\
&= \sum_{j=0}^{\infty} b_{j+1} \, \frac{\Gamma(1+(j+1)\,\alpha)}{\Gamma(1+j\,\alpha)} \, t^{j\,\alpha}.
\end{aligned}
$$

- Substituting into the fractional IDE and matching coefficients of t^0 gives

$$
b_1 = -\frac{1}{\Gamma(1+\alpha)} \, \frac{1}{2} \, a(a+\mathrm{i}).
$$

- Doing the same with the coefficient of t^α gives

$$
b_2 = \frac{\Gamma(1+\alpha)}{\Gamma(1+2\alpha)} \, (\mathrm{i}\,\rho\,a - \lambda') \, \nu \, b_1,
$$

where as before, $\lambda' = \lambda/\nu$.

- This generalizes to the recursion

$$
\begin{aligned}
b_1 &= -\frac{1}{\Gamma(1+\alpha)} \, \frac{1}{2} \, a(a+\mathrm{i}) \\
b_k &= \frac{\Gamma(1+(k-1)\,\alpha)}{\Gamma(1+k\,\alpha)} \left\{ -\tilde{\lambda}\,\nu\, b_{k-1} + \frac{1}{2}\,\nu^2 \sum_{i,j=1}^{k-1} 1_{i+j=k-1} \, b_i \, b_j \right\},
\end{aligned}
$$

where $\tilde{\lambda} = \lambda' - \mathrm{i}\,\rho\,a$.

Solving the rough Heston Riccati equation for long times

- The fractional Riccati equation ODE may be re-expressed as

$$
D^\alpha h(t;a) = \frac{1}{2} \, (\nu\, h(t;a) - r_-) \, (\nu\, h(t;a) - r_+) \,, \tag{4}
$$

with $A = \sqrt{a\,(a+i) + (\lambda' - \mathrm{i}\,\rho\,a)^2}$; $\quad r_\pm = \{\lambda' - \mathrm{i}\,\rho\,a \pm A\}$; $\lambda' = \lambda/\nu$.

- Let $\nu\, h_\infty(t;a) = r_- \, [1 - E_\alpha(-A\,\nu\,t^\alpha)]$ where E_α is the Mittag-Leffler function.
- Then, for $t \in \mathbb{R}_{\geq 0}$ and $a \in \mathcal{A}$ where $\mathcal{A}$ us suitably defined, $h_\infty(t;a)$ satisfies

$$
\nu\, h_\infty(t;a) - r_- = -\frac{r_-}{A\nu} \, \frac{t^{-\alpha}}{\Gamma(1-\alpha)} + \mathcal{O}\left(\left| A\,\nu\,t^\alpha \right|^{-2} \right). \tag{5}
$$

and thus solves the rough Heston Riccati equationup to an error term of $\mathcal{O}\left(\left|A\,\nu\,t^\alpha\right|^{-2}\right)$, as $t \to \infty$.

- The form of the asymptotic expansion of $E_\alpha(-A\,\nu\,t^\alpha)$ motivates the following ansatz for $h(t;a)$ as $t \to \infty$:

$$h(t;a) = \sum_{k=0}^\infty g_k\, t^{-k\alpha}. \tag{6}$$

- Then

$$D^\alpha h(t;a) = \sum_{k=1}^\infty g_{k-1}\,\frac{\Gamma(1-(k-1)\alpha)}{\Gamma(1-k\,\alpha)}\,t^{-k\alpha}.$$

- Note that, from the asymptotic solution,

$$g_0 = \frac{r_-}{\nu}; \quad g_1 = -\frac{r_-}{A\nu^2}\,\frac{1}{\Gamma(1-\alpha)}.$$

- Also, from the fractional ODE, using that $g_0 = r_-/\nu$,

$$\begin{aligned} D^\alpha h(a,x) &= \frac{1}{2}\left(\nu\,h(t;a) - r_-\right)\left(\nu\,h(t;a) - r_+\right) \\ &= \nu\sum_{k=1}^\infty g_k\,t^{-k\alpha}\left(-A + \frac{1}{2}\,\nu\sum_{k=1}^\infty g_k\,t^{-k\alpha}\right). \end{aligned}$$

- We obtain

$$\begin{aligned} &\sum_{k=1}^\infty g_{k-1}\,\frac{\Gamma(1-(k-1)\alpha)}{\Gamma(1-k\,\alpha)}\,t^{-k\alpha} \\ &= \nu\sum_{k=1}^\infty g_k\,t^{-k\alpha}\left(-A + \frac{1}{2}\,\nu\sum_{k=1}^\infty g_k\,t^{-k\alpha}\right). \end{aligned}$$

- Matching coefficients of $t^{-\alpha}$ gives

$$g_1 = -\frac{1}{A\nu}\,\frac{1}{\Gamma(1-\alpha)}\,g_0.$$

- Similarly, matching coefficients of $t^{-2\alpha}$ gives

$$g_2 = -\frac{1}{A\,\nu}\left\{\frac{\Gamma(1-\alpha)}{\Gamma(1-2\,\alpha)}\,g_1 - \frac{1}{2}\,\nu^2\,g_1^2\right\}.$$

- The general recursion for $k > 2$ is given by

$$g_k = -\frac{1}{A\,\nu}\left\{\frac{\Gamma(1-(k-1)\alpha)}{\Gamma(1-k\alpha)}\,g_{k-1} - \frac{1}{2}\,\nu^2\sum_{i,j=1}^{\infty}1_{i+j=k}\,g_i\,g_j\right\}.$$

Rational approximations of h

- Now that we have short-time and long-time asymptotics of h, we can construct rational approximations that natch the short- and long-term to a given order.

- The only admissible global rational approximations of h are of the diagonal form

$$h^{(n,n)}(t;a) = \frac{\sum_{i=1}^n p_{n,i}y^n}{\sum_{j=0}^n q_{n,j}y^n} \tag{7}$$

with $y = \nu\,t^\alpha$.

- Explicit expressions for the coefficients $p_{n,i}$ and $q_{n,j}$ are provided in `roughHestonPadeLambda.R`.

- `roughHestonPadeLambda.R` is made openly accessible at https://github.com/jgatheral/RationalRoughHeston, together with Jupyter notebooks illustrating the usage of the $h^{(n,n)}$.

Some R-code

In [1]:
```
setwd("./QRV")
```

In [2]:
```
source("BlackScholes.R")
source("Heston.R")
source("HQE.R")
source("Lewis.R")
source("roughHestonPadeLambda.R")
source("gammaKernel.R")
source("plotIvols.R")
```

In [3]:
```
library(repr)
library(colorspace)
library(MittagLeffleR)
library(stinepack)
options(repr.plot.height=7,repr.plot.width=10,rep.plot.res=200)
```

Set up nice colors

In [4]:
```
my.col <- sequential_hcl(5, palette="Batlow")
bl <- "royalblue"
rd <- "red2"
pk <- "hotpink1"
gr <- "green4"
br <-"brown"
pu <- "purple"
or <- "orange"
```

R implementation of the rational approximation

- The complicated algebra to get the coefficients coefficients $p_{n,i}$ and $q_{n,j}$ from the b_k and the g_k need only be done once.
    - Wuth Mathematica in my case!
- `h.Pade22` is easy enough to be computed by hand.
- `h.Pade66` is too complicated to print!
- Let's look at some examples:

In [5]:
```
h.Pade22
```

```
function (params)
function(a, tau) {
    H <- params$H
    rho <- params$rho
    nu <- params$nu
    al <- H + 1/2
    lam <- params$lam
    lamp <- lam/nu
    lamTilde <- lamp - (0 + (0 + (0+1i))) * rho * a
    aa <- sqrt(a * (a + (0 + (0 + (0+1i)))) + lamTilde^2)
    rm <- lamTilde - aa
    rp <- lamTilde + aa
    b1 <- -a * (a + (0 + (0+1i)))/2 * 1/gamma(1 + al)
    b2 <- -b1 * lamTilde * nu * gamma(1 + al)/gamma(1 + 2 * al)
    g0 <- rm/nu
    g1 <- ifelse(al == 1, 0, -1/aa * 1/gamma(1 - al) * g0/nu)
    den <- g0^2 + b1 * g1
    q1 <- (b1 * g0 - b2 * g1)/den
    q2 <- (b1^2 + b2 * g0)/den
    p1 <- b1
    p2 <- b2 + b1 * q1
    y <- tau^al
    h.pade <- (p1 * y + p2 * y^2)/(1 + q1 * y + q2 * y^2)
    return(h.pade)
}
```

In [6]:
```
h.Pade33
```

```
function (params)
function(a, tau) {
    H <- params$H
    rho <- params$rho
    nu <- params$nu
    al <- H + 1/2
    lam <- params$lam
    lamp <- lam/nu
    lamTilde <- lamp - (0 + (0+1i)) * rho * a
    aa <- sqrt(a * (a + (0 + (0+1i))) + lamTilde^2)
    rm <- lamTilde - aa
    rp <- lamTilde + aa
    b1 <- -a * (a + (0+1i))/2 * 1/gamma(1 + al)
    b2 <- -b1 * lamTilde * nu * gamma(1 + al)/gamma(1 + 2 * al)
    b3 <- (-b2 * lamTilde * nu + nu^2 * b1^2/2) * gamma(1 + 2 *
        al)/gamma(1 + 3 * al)
    b4 <- (-b3 * lamTilde * nu + nu^2 * b1 * b2) * gamma(1 +
        3 * al)/gamma(1 + 4 * al)
    g0 <- rm/nu
    g1 <- -1/(aa * nu) * 1/gamma(1 - al) * g0
    g2 <- -1/(aa * nu) * (gamma(1 - al)/gamma(1 - 2 * al) * g1 -
        1/2 * nu^2 * g1 * g1)
    g3 <- -1/(aa * nu) * (gamma(1 - 2 * al)/gamma(1 - 3 * al) *
        g2 - nu^2 * g1 * g2)
    den <- g0^3 + 2 * b1 * g0 * g1 - b2 * g1^2 + b1^2 * g2 +
        b2 * g0 * g2
    p1 <- b1
    p2 <- (b1^2 * g0^2 + b2 * g0^3 + b1^3 * g1 + b1 * b2 * g0 *
        g1 - b2^2 * g1^2 + b1 * b3 * g1^2 + b2^2 * g0 * g2 -
        b1 * b3 * g0 * g2)/den
    q1 <- (b1 * g0^2 + b1^2 * g1 - b2 * g0 * g1 + b3 * g1^2 -
        b1 * b2 * g2 - b3 * g0 * g2)/den
    q2 <- (b1^2 * g0 + b2 * g0^2 - b1 * b2 * g1 - b3 * g0 * g1 +
        b2^2 * g2 - b1 * b3 * g2)/den
    q3 <- (b1^3 + 2 * b1 * b2 * g0 + b3 * g0^2 - b2^2 * g1 +
        b1 * b3 * g1)/den
    p3 <- g0 * q3
    y <- tau^al
    h.pade <- (p1 * y + p2 * y^2 + p3 * y^3)/(1 + q1 * y + q2 *
        y^2 + q3 * y^3)
    return(h.pade)
}
```

In [7]:
```
h.Pade55
```

```
function (params)
function(a, tau) {
    H <- params$H
    rho <- params$rho
    nu <- params$nu
    al <- H + 1/2
    lam <- params$lam
    lamp <- lam/nu
    lamTilde <- lamp - (0 + (0 + (0+1i))) * rho * a
    aa <- sqrt(a * (a + (0 + (0 + (0+1i)))) + lamTilde^2)
    rm <- lamTilde - aa
    rp <- lamTilde + aa
    b1 <- -a * (a + (0 + (0+1i)))/2 * 1/gamma(1 + al)
    b2 <- -b1 * lamTilde * nu * gamma(1 + al)/gamma(1 + 2 * al)
    b3 <- (-b2 * lamTilde * nu + nu^2 * b1^2/2) * gamma(1 + 2 *
        al)/gamma(1 + 3 * al)
    b4 <- (-b3 * lamTilde * nu + nu^2 * b1 * b2) * gamma(1 +
        3 * al)/gamma(1 + 4 * al)
    b5 <- (-b4 * lamTilde * nu + nu^2 * (1/2 * b2 * b2 + b1 *
        b3)) * gamma(1 + 4 * al)/gamma(1 + 5 * al)
    g0 <- rm/nu
    g1 <- ifelse(al == 1, 0, -1/(aa * nu) * 1/gamma(1 - al) *
        g0)
    g2 <- ifelse(al == 1, 0, -1/(aa * nu) * (gamma(1 - al)/gamma(1 -
        2 * al) * g1 - 1/2 * nu^2 * g1 * g1))
    g3 <- ifelse(al == 1, 0, -1/(aa * nu) * (gamma(1 - 2 * al)/gamma(1
-
        3 * al) * g2 - nu^2 * g1 * g2))
    g4 <- ifelse(al == 1, 0, -1/(aa * nu) * (gamma(1 - 3 * al)/gamma(1
-
        4 * al) * g3 - nu^2 * (1/2 * g2 * g2 + g1 * g3)))
    den <- (-g0^5 - 4 * b1 * g0^3 * g1 - 3 * b1^2 * g0 * g1^2 +
        3 * b2 * g0^2 * g1^2 + 2 * b1 * b2 * g1^3 - 2 * b3 *
        g0 * g1^3 + b4 * g1^4 - 3 * b1^2 * g0^2 * g2 - 3 * b2 *
        g0^3 * g2 - 2 * b1^3 * g1 * g2 + 2 * b1 * b2 * g0 * g1 *
        g2 + 4 * b3 * g0^2 * g1 * g2 - b2^2 * g1^2 * g2 - 2 *
        b1 * b3 * g1^2 * g2 - 3 * b4 * g0 * g1^2 * g2 + b1^2 *
        b2 * g2^2 - 2 * b2^2 * g0 * g2^2 + 4 * b1 * b3 * g0 *
        g2^2 + b4 * g0^2 * g2^2 + 2 * b2 * b3 * g1 * g2^2 - 2 *
        b1 * b4 * g1 * g2^2 - b3^2 * g2^3 + b2 * b4 * g2^3 -
        2 * b1^3 * g0 * g3 - 4 * b1 * b2 * g0^2 * g3 - 2 * b3 *
        g0^3 * g3 + 2 * b1^2 * b2 * g1 * g3 + 4 * b2^2 * g0 *
        g1 * g3 + 2 * b4 * g0^2 * g1 * g3 - 2 * b2 * b3 * g1^2 *
        g3 + 2 * b1 * b4 * g1^2 * g3 - 2 * b1 * b2^2 * g2 * g3 +
        2 * b1^2 * b3 * g2 * g3 - 2 * b2 * b3 * g0 * g2 * g3 +
        2 * b1 * b4 * g0 * g2 * g3 + 2 * b3^2 * g1 * g2 * g3 -
        2 * b2 * b4 * g1 * g2 * g3 + b2^3 * g3^2 - 2 * b1 * b2 *
        b3 * g3^2 + b1^2 * b4 * g3^2 - b3^2 * g0 * g3^2 + b2 *
        b4 * g0 * g3^2 - b1^4 * g4 - 3 * b1^2 * b2 * g0 * g4 -
        b2^2 * g0^2 * g4 - 2 * b1 * b3 * g0^2 * g4 - b4 * g0^3 *
        g4 + 2 * b1 * b2^2 * g1 * g4 - 2 * b1^2 * b3 * g1 * g4 +
        2 * b2 * b3 * g0 * g1 * g4 - 2 * b1 * b4 * g0 * g1 *
        g4 - b3^2 * g1^2 * g4 + b2 * b4 * g1^2 * g4 - b2^3 *
        g2 * g4 + 2 * b1 * b2 * b3 * g2 * g4 - b1^2 * b4 * g2 *
        g4 + b3^2 * g0 * g2 * g4 - b2 * b4 * g0 * g2 * g4)
    q1 <- (-(b1 * g0^4) - 3 * b1^2 * g0^2 * g1 + b2 * g0^3 *
        g1 - b1^3 * g1^2 + 4 * b1 * b2 * g0 * g1^2 - b3 * g0^2 *
        g1^2 - b2^2 * g1^3 - 2 * b1 * b3 * g1^3 + b4 * g0 * g1^3 -
        b5 * g1^4 - 2 * b1^3 * g0 * g2 - b1 * b2 * g0^2 * g2 +
        b3 * g0^3 * g2 + 4 * b1^2 * b2 * g1 * g2 + 2 * b1 * b3 *
        g0 * g1 * g2 - 2 * b4 * g0^2 * g1 * g2 + 2 * b2 * b3 *
        g1^2 * g2 + b1 * b4 * g1^2 * g2 + 3 * b5 * g0 * g1^2 *
        g2 - 2 * b1 * b2^2 * g2^2 + b1^2 * b3 * g2^2 - 2 * b1 *
        b4 * g0 * g2^2 - b5 * g0^2 * g2^2 - b3^2 * g1 * g2^2 -
        b2 * b4 * g1 * g2^2 + 2 * b1 * b5 * g1 * g2^2 + b3 *
        b4 * g2^3 - b2 * b5 * g2^3 - b1^4 * g3 - b1^2 * b2 *
        g0 * g3 + b2^2 * g0^2 * g3 + b4 * g0^3 * g3 - 2 * b1^2 *
        b3 * g1 * g3 - 4 * b2 * b3 * g0 * g1 * g3 - 2 * b5 *
        g0^2 * g1 * g3 + b3^2 * g1^2 * g3 + b2 * b4 * g1^2 *
        g3 - 2 * b1 * b5 * g1^2 * g3 + b2^3 * g2 * g3 - b1^2 *
        b4 * g2 * g3 + b3^2 * g0 * g2 * g3 + b2 * b4 * g0 * g2 *
        g3 - 2 * b1 * b5 * g0 * g2 * g3 - 2 * b3 * b4 * g1 *
        g2 * g3 + 2 * b2 * b5 * g1 * g2 * g3 - b2^2 * b3 * g3^2 +
        b1 * b3^2 * g3^2 + b1 * b2 * b4 * g3^2 - b1^2 * b5 *
        g3^2 + b3 * b4 * g0 * g3^2 - b2 * b5 * g0 * g3^2 + b1^3 *
        b2 * g4 + 2 * b1 * b2^2 * g0 * g4 + b1^2 * b3 * g0 *
        g4 + 2 * b2 * b3 * g0^2 * g4 + b1 * b4 * g0^2 * g4 +
        b5 * g0^3 * g4 - b2^3 * g1 * g4 + b1^2 * b4 * g1 * g4 -
        b3^2 * g0 * g1 * g4 - b2 * b4 * g0 * g1 * g4 + 2 * b1 *
        b5 * g0 * g1 * g4 + b3 * b4 * g1^2 * g4 - b2 * b5 * g1^2 *
        g4 + b2^2 * b3 * g2 * g4 - b1 * b3^2 * g2 * g4 - b1 *
        b2 * b4 * g2 * g4 + b1^2 * b5 * g2 * g4 - b3 * b4 * g0 *
        g2 * g4 + b2 * b5 * g0 * g2 * g4)/den
    q2 <- (-(b1^2 * g0^3) - b2 * g0^4 - 2 * b1^3 * g0 * g1 -
        b1 * b2 * g0^2 * g1 + b3 * g0^3 * g1 + 2 * b1^2 * b2 *
        g1^2 + b2^2 * g0 * g1^2 - b4 * g0^2 * g1^2 + b1 * b4 *
        g1^3 + b5 * g0 * g1^3 - b1^4 * g2 - b1^2 * b2 * g0 *
        g2 - 2 * b2^2 * g0^2 * g2 + 3 * b1 * b3 * g0^2 * g2 +
        b4 * g0^3 * g2 - 2 * b1 * b2^2 * g1 * g2 - 4 * b1 * b4 *
        g0 * g1 * g2 - 2 * b5 * g0^2 * g1 * g2 - b2 * b4 * g1^2 *
        g2 + b1 * b5 * g1^2 * g2 + 2 * b1 * b2 * b3 * g2^2 -
        2 * b1^2 * b4 * g2^2 - b3^2 * g0 * g2^2 + 3 * b2 * b4 *
        g0 * g2^2 - 2 * b1 * b5 * g0 * g2^2 + b3 * b4 * g1 *
        g2^2 - b2 * b5 * g1 * g2^2 - b4^2 * g2^3 + b3 * b5 *
        g2^3 + b1^3 * b2 * g3 + 3 * b1^2 * b3 * g0 * g3 + 3 *
        b1 * b4 * g0^2 * g3 + b5 * g0^3 * g3 + b2^3 * g1 * g3 -
        2 * b1 * b2 * b3 * g1 * g3 + b1^2 * b4 * g1 * g3 + b3^2 *
        g0 * g1 * g3 - b2 * b4 * g0 * g1 * g3 - b3 * b4 * g1^2 *
        g3 + b2 * b5 * g1^2 * g3 - b2^2 * b3 * g2 * g3 - b1 *
        b3^2 * g2 * g3 + 3 * b1 * b2 * b4 * g2 * g3 - b1^2 *
        b5 * g2 * g3 - b3 * b4 * g0 * g2 * g3 + b2 * b5 * g0 *
        g2 * g3 + 2 * b4^2 * g1 * g2 * g3 - 2 * b3 * b5 * g1 *
        g2 * g3 + b2 * b3^2 * g3^2 - b2^2 * b4 * g3^2 - b1 *
        b3 * b4 * g3^2 + b1 * b2 * b5 * g3^2 - b4^2 * g0 * g3^2 +
        b3 * b5 * g0 * g3^2 - b1^2 * b2^2 * g4 + b1^3 * b3 *
        g4 - b2^3 * g0 * g4 + b1^2 * b4 * g0 * g4 - b2 * b4 *
        g0^2 * g4 + b1 * b5 * g0^2 * g4 + b2^2 * b3 * g1 * g4 +
        b1 * b3^2 * g1 * g4 - 3 * b1 * b2 * b4 * g1 * g4 + b1^2 *
        b5 * g1 * g4 + b3 * b4 * g0 * g1 * g4 - b2 * b5 * g0 *
        g1 * g4 - b4^2 * g1^2 * g4 + b3 * b5 * g1^2 * g4 - b2 *
        b3^2 * g2 * g4 + b2^2 * b4 * g2 * g4 + b1 * b3 * b4 *
        g2 * g4 - b1 * b2 * b5 * g2 * g4 + b4^2 * g0 * g2 * g4 -
        b3 * b5 * g0 * g2 * g4)/den
    q3 <- (-(b1^3 * g0^2) - 2 * b1 * b2 * g0^3 - b3 * g0^4 -
        b1^4 * g1 - b1^2 * b2 * g0 * g1 + 2 * b2^2 * g0^2 * g1 -
        b1 * b3 * g0^2 * g1 + b4 * g0^3 * g1 + b1 * b2^2 * g1^2 -
        2 * b1^2 * b3 * g1^2 - 2 * b2 * b3 * g0 * g1^2 - b5 *
        g0^2 * g1^2 + b2 * b4 * g1^3 - b1 * b5 * g1^3 + b1^3 *
        b2 * g2 + 3 * b1^2 * b3 * g0 * g2 + 3 * b1 * b4 * g0^2 *
        g2 + b5 * g0^3 * g2 + 2 * b3^2 * g0 * g1 * g2 - 2 * b2 *
        b4 * g0 * g1 * g2 - b3 * b4 * g1^2 * g2 + b2 * b5 * g1^2 *
        g2 - b1 * b3^2 * g2^2 + b1 * b2 * b4 * g2^2 - b3 * b4 *
        g0 * g2^2 + b2 * b5 * g0 * g2^2 + b4^2 * g1 * g2^2 -
        b3 * b5 * g1 * g2^2 - b1^2 * b2^2 * g3 + b1^3 * b3 *
        g3 + b2^3 * g0 * g3 - 4 * b1 * b2 * b3 * g0 * g3 + 3 *
        b1^2 * b4 * g0 * g3 - 2 * b3^2 * g0^2 * g3 + b2 * b4 *
        g0^2 * g3 + b1 * b5 * g0^2 * g3 - b2^2 * b3 * g1 * g3 +
        3 * b1 * b3^2 * g1 * g3 - b1 * b2 * b4 * g1 * g3 - b1^2 *
        b5 * g1 * g3 + 3 * b3 * b4 * g0 * g1 * g3 - 3 * b2 *
        b5 * g0 * g1 * g3 - b4^2 * g1^2 * g3 + b3 * b5 * g1^2 *
        g3 + b2 * b3^2 * g2 * g3 - b2^2 * b4 * g2 * g3 - b1 *
        b3 * b4 * g2 * g3 + b1 * b2 * b5 * g2 * g3 - b4^2 * g0 *
        g2 * g3 + b3 * b5 * g0 * g2 * g3 - b3^3 * g3^2 + 2 *
        b2 * b3 * b4 * g3^2 - b1 * b4^2 * g3^2 - b2^2 * b5 *
        g3^2 + b1 * b3 * b5 * g3^2 + b1 * b2^3 * g4 - 2 * b1^2 *
        b2 * b3 * g4 + b1^3 * b4 * g4 + b2^2 * b3 * g0 * g4 -
        2 * b1 * b3^2 * g0 * g4 + b1^2 * b5 * g0 * g4 - b3 *
        b4 * g0^2 * g4 + b2 * b5 * g0^2 * g4 - b2 * b3^2 * g1 *
        g4 + b2^2 * b4 * g1 * g4 + b1 * b3 * b4 * g1 * g4 - b1 *
        b2 * b5 * g1 * g4 + b4^2 * g0 * g1 * g4 - b3 * b5 * g0 *
        g1 * g4 + b3^3 * g2 * g4 - 2 * b2 * b3 * b4 * g2 * g4 +
        b1 * b4^2 * g2 * g4 + b2^2 * b5 * g2 * g4 - b1 * b3 *
        b5 * g2 * g4)/den
    q4 <- (-(b1^4 * g0) - 3 * b1^2 * b2 * g0^2 - b2^2 * g0^3 -
        2 * b1 * b3 * g0^3 - b4 * g0^4 + b1^3 * b2 * g1 + 4 *
        b1 * b2^2 * g0 * g1 - b1^2 * b3 * g0 * g1 + 4 * b2 *
        b3 * g0^2 * g1 - b1 * b4 * g0^2 * g1 + b5 * g0^3 * g1 -
        b2^3 * g1^2 + b1^2 * b4 * g1^2 - 2 * b3^2 * g0 * g1^2 +
        2 * b1 * b5 * g0 * g1^2 + b3 * b4 * g1^3 - b2 * b5 *
        g1^3 - b1^2 * b2^2 * g2 + b1^3 * b3 * g2 - 2 * b2^3 *
        g0 * g2 + 2 * b1 * b2 * b3 * g0 * g2 + b3^2 * g0^2 *
        g2 - 2 * b2 * b4 * g0^2 * g2 + b1 * b5 * g0^2 * g2 +
        2 * b2^2 * b3 * g1 * g2 - 4 * b1 * b2 * b4 * g1 * g2 +
        2 * b1^2 * b5 * g1 * g2 - b4^2 * g1^2 * g2 + b3 * b5 *
        g1^2 * g2 - b2 * b3^2 * g2^2 + b2^2 * b4 * g2^2 + b1 *
        b3 * b4 * g2^2 - b1 * b2 * b5 * g2^2 + b4^2 * g0 * g2^2 -
        b3 * b5 * g0 * g2^2 + b1 * b2^3 * g3 - 2 * b1^2 * b2 *
        b3 * g3 + b1^3 * b4 * g3 + b2^2 * b3 * g0 * g3 - 2 *
        b1 * b3^2 * g0 * g3 + b1^2 * b5 * g0 * g3 - b3 * b4 *
        g0^2 * g3 + b2 * b5 * g0^2 * g3 - b2 * b3^2 * g1 * g3 +
        b2^2 * b4 * g1 * g3 + b1 * b3 * b4 * g1 * g3 - b1 * b2 *
        b5 * g1 * g3 + b4^2 * g0 * g1 * g3 - b3 * b5 * g0 * g1 *
        g3 + b3^3 * g2 * g3 - 2 * b2 * b3 * b4 * g2 * g3 + b1 *
        b4^2 * g2 * g3 + b2^2 * b5 * g2 * g3 - b1 * b3 * b5 *
        g2 * g3 - b2^4 * g4 + 3 * b1 * b2^2 * b3 * g4 - b1^2 *
        b3^2 * g4 - 2 * b1^2 * b2 * b4 * g4 + b1^3 * b5 * g4 +
        2 * b2 * b3^2 * g0 * g4 - 2 * b2^2 * b4 * g0 * g4 - 2 *
        b1 * b3 * b4 * g0 * g4 + 2 * b1 * b2 * b5 * g0 * g4 -
```

```
        b4^2 * g0^2 * g4 + b3 * b5 * g0^2 * g4 - b3^3 * g1 *
        g4 + 2 * b2 * b3 * b4 * g1 * g4 - b1 * b4^2 * g1 * g4 -
        b2^2 * b5 * g1 * g4 + b1 * b3 * b5 * g1 * g4)/den
    q5 <- (-b1^5 - 4 * b1^3 * b2 * g0 - 3 * b1 * b2^2 * g0^2 -
        3 * b1^2 * b3 * g0^2 - 2 * b2 * b3 * g0^3 - 2 * b1 *
        b4 * g0^3 - b5 * g0^4 + 3 * b1^2 * b2^2 * g1 - 3 * b1^3 *
        b3 * g1 + 2 * b2^3 * g0 * g1 + 2 * b1 * b2 * b3 * g0 *
        g1 - 4 * b1^2 * b4 * g0 * g1 + b3^2 * g0^2 * g1 + 2 *
        b2 * b4 * g0^2 * g1 - 3 * b1 * b5 * g0^2 * g1 - b2^2 *
        b3 * g1^2 - 2 * b1 * b3^2 * g1^2 + 4 * b1 * b2 * b4 *
        g1^2 - b1^2 * b5 * g1^2 - 2 * b3 * b4 * g0 * g1^2 + 2 *
        b2 * b5 * g0 * g1^2 + b4^2 * g1^3 - b3 * b5 * g1^3 -
        2 * b1 * b2^3 * g2 + 4 * b1^2 * b2 * b3 * g2 - 2 * b1^3 *
        b4 * g2 - 2 * b2^2 * b3 * g0 * g2 + 4 * b1 * b3^2 * g0 *
        g2 - 2 * b1^2 * b5 * g0 * g2 + 2 * b3 * b4 * g0^2 * g2 -
        2 * b2 * b5 * g0^2 * g2 + 2 * b2 * b3^2 * g1 * g2 - 2 *
        b2^2 * b4 * g1 * g2 - 2 * b1 * b3 * b4 * g1 * g2 + 2 *
        b1 * b2 * b5 * g1 * g2 - 2 * b4^2 * g0 * g1 * g2 + 2 *
        b3 * b5 * g0 * g1 * g2 - b3^3 * g2^2 + 2 * b2 * b3 *
        b4 * g2^2 - b1 * b4^2 * g2^2 - b2^2 * b5 * g2^2 + b1 *
        b3 * b5 * g2^2 + b2^4 * g3 - 3 * b1 * b2^2 * b3 * g3 +
        b1^2 * b3^2 * g3 + 2 * b1^2 * b2 * b4 * g3 - b1^3 * b5 *
        g3 - 2 * b2 * b3^2 * g0 * g3 + 2 * b2^2 * b4 * g0 * g3 +
        2 * b1 * b3 * b4 * g0 * g3 - 2 * b1 * b2 * b5 * g0 *
        g3 + b4^2 * g0^2 * g3 - b3 * b5 * g0^2 * g3 + b3^3 *
        g1 * g3 - 2 * b2 * b3 * b4 * g1 * g3 + b1 * b4^2 * g1 *
        g3 + b2^2 * b5 * g1 * g3 - b1 * b3 * b5 * g1 * g3)/den
    p1 <- b1
    p2 <- b2 + b1 * q1
    p3 <- b3 + b1 * q2 + b2 * q1
    p4 <- b4 + b3 * q1 + b2 * q2 + b1 * q3
    p5 <- g0 * q5
    y <- tau^al
    h.pade <- (p1 * y + p2 * y^2 + p3 * y^3 + p4 * y^4 + p5 *
        y^5)/(1 + q1 * y + q2 * y^2 + q3 * y^3 + q4 * y^4 + q5 *
        y^5)
    return(h.pade)
}
```

R implementation of the Lewis formula

In [8]:
```
option.OTM.raw
```

```
function (phi, k, tau)
{
    integrand <- function(u) {
        Re(exp(-(0 + (0+1i)) * u * k) * phi(u - (0 + (0+1i))/2,
            tau)/(u^2 + 1/4))
    }
    k.minus <- (k < 0) * k
    res <- exp(k.minus) - exp(k/2)/pi * integrate(integrand,
        lower = 0, upper = Inf, rel.tol = 1e-10, subdivisions = 1000)
$value
    return(ifelse(res < 0, NA, res))
}
```

The rough Heston smile

In [9]:
```
params.rHeston <- list(H=0.05,nu=0.4,rho=-.65,lam=0)
xiCurve <- function(t){.16^2+0*t}
```

In [10]:
```
phi <- phiRoughHestonRational(params.rHeston, xiCurve, h.approx= h.Pade44, r
```

In [11]:
```
vol <- function(k){
    sapply(k,function(x){impvol.phi(phi)(x,1)})}
system.time(curve(vol(x),from=-.4,to=.4,col=rd,lwd=2,xlab="Log-strike k",yla
```

```
  user  system elapsed 
 3.026   0.043   3.069 
```

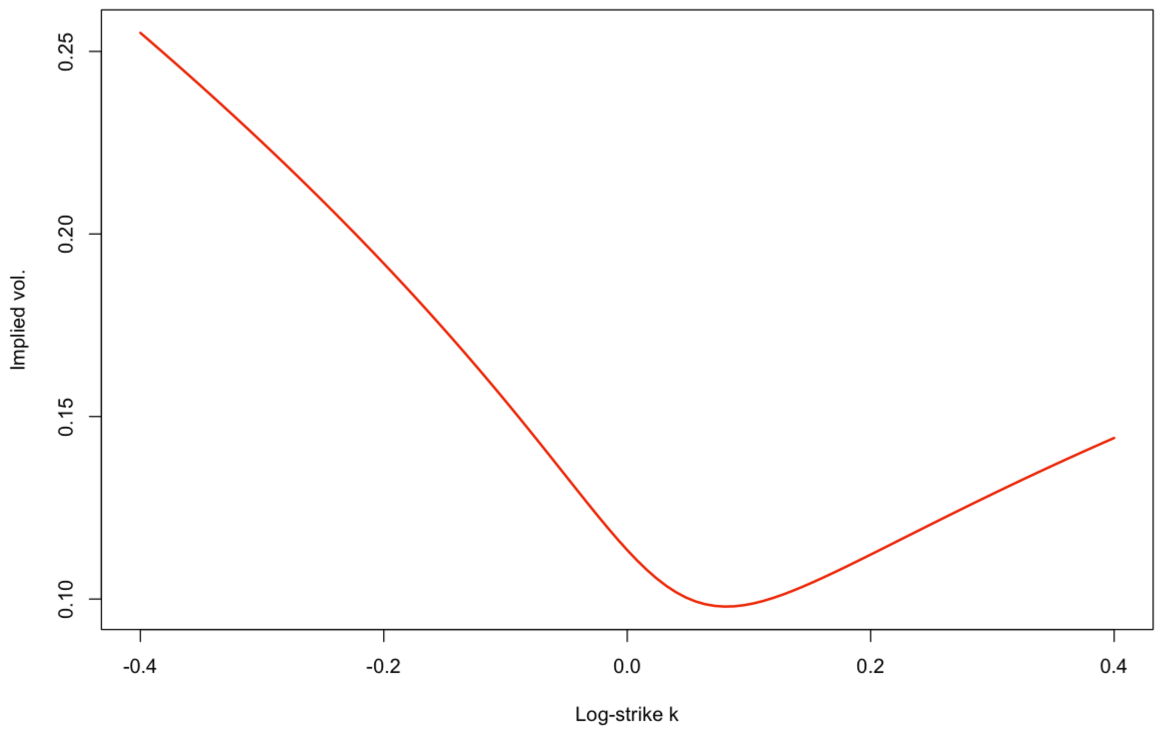

Figure 1: The 1-year rough Heston smile using the approximation $h^{(3,3)}$.

On generating the smile

- In our code, we compute the Lewis formula for each strike and expiration.
- There are much more efficient methods that take advantage of the structure of the characteristic fuction.
  - For example the COS method or the more recent SINC method of [Baschetti et al.][3].
    - Their code is available at https://github.com/fabioBaschetti/SINC-method!

How does $h^{(3,3)}$ compare with $h^{(2,2)}$ and $h^{(4,4)}$?

In [12]:
```
phi2 <- phiRoughHestonRational(params.rHeston, xiCurve, h.approx=h.Pade22, r
phi4 <- phiRoughHestonRational(params.rHeston, xiCurve, h.approx=h.Pade44, r
```

In [13]:
```
vol2 <- function(k){sapply(k,function(x){impvol.phi(phi2)(x,1)})}
vol4 <- function(k){sapply(k,function(x){impvol.phi(phi4)(x,1)})}
```

In [14]:
```
curve(vol(x),from=-.4,to=.4,col=rd,lwd=2,xlab="Log-strike k",ylab="Implied v
curve(vol2(x),from=-.4,to=.4,col=bl,lwd=2,add=T,lty=2)
curve(vol4(x),from=-.4,to=.4,col=gr, lwd=2,,add=T,lty=2)
```

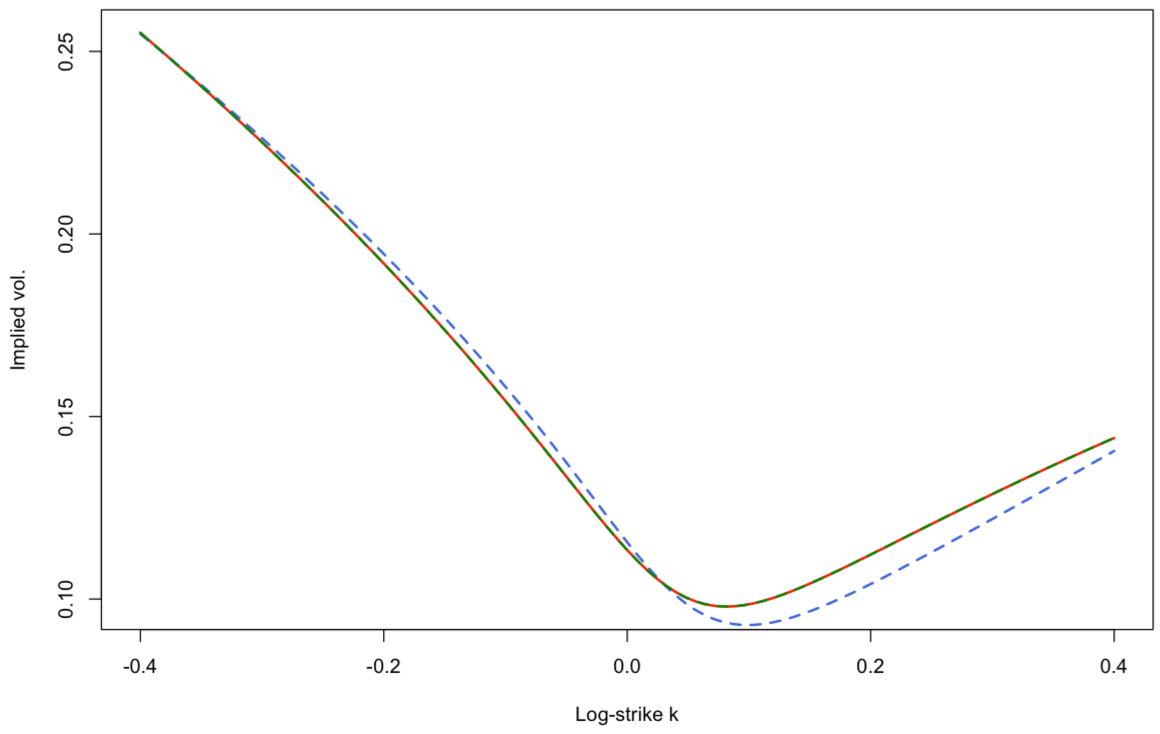

Figure 2: The 1-year rough Heston smile in red with approximation $h^{(3,3)}$. The blue dashed line is $h^{(2,2)}$, and the green dotted line $h^{(4,4)}$.

Sensitivity of the rough Heston smile to ν

First, a function to compute the 1-year smile:

In [15]:
```r
vol <- function(params)function(k){ # A function to compute the 1-year smile
    phi <- phiRoughHestonRational(params, xiCurve, h.approx=h.Pade33, n=20)
    sapply(k,function(x){impvol.phi(phi)(x,1)})}


sub.nu <- function(nu.in){
    tmp <- params.rHeston
    tmp$nu <- nu.in
    return(tmp)
}
```

In [16]:
```r
yrange <- c(0.07,.3)
curve(vol(params.rHeston)(x),from=-.5,to=.5,col=my.col[1],ylim=yrange,lwd=2,
nu.vec <- params.rHeston$nu + c(0.1,0.2,0.3,0.4,0.5)
for (j in 1:5)
    {
    curve(vol(sub.nu(nu.vec[j]))(x),from=-.5,to=.5,col=my.col[j+1],lty=1,lwd
    }
```

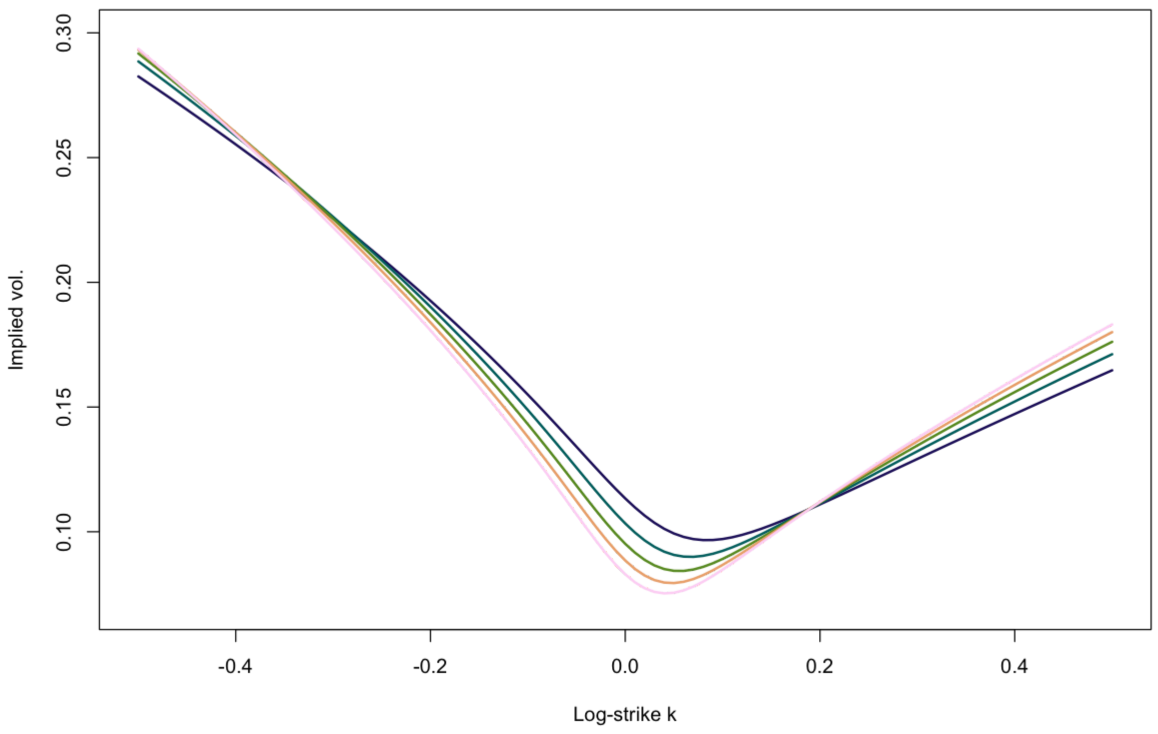

Figure 3: The dotted lines are smiles with $\eta \mapsto= \eta + \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

Sensitivity of the rough Heston smile to ρ

In [17]:
```
sub.rho <- function(rho.in){
    tmp <- params.rHeston
    tmp$rho <- rho.in
    return(tmp)
}
```

In [18]:
```
yrange <- c(0.07,.3)
curve(vol(params.rHeston)(x),from=-.5,to=.5,col=my.col[1],ylim=yrange,lwd=2,
rho.vec <- params.rHeston$rho - c(0.05,0.10,0.15,0.20,0.25)
for (j in 1:5)
    {

    curve(vol(sub.rho(rho.vec[j]))(x),from=-.5,to=.5,col=my.col[j+1],lwd=2,a
    }
```

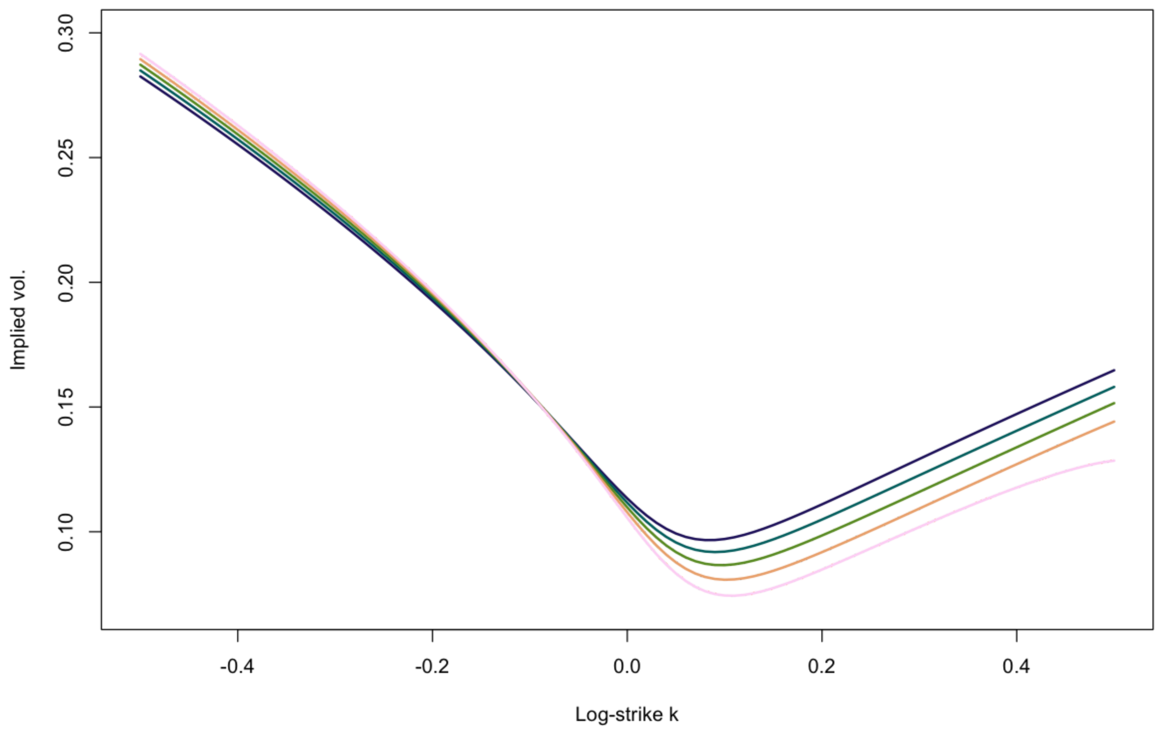

Figure 4: The dotted lines are smiles with $\rho \mapsto \rho - \{0.05, 0.10, 0.15, 0.2, 0.25\}$.

Sensitivity of the rough Heston 1 year smile to H

In [19]:
```
sub.H <- function(H.in){
    tmp <- params.rHeston
    tmp$H <- H.in
    return(tmp)
}
```

In [20]:
```
yrange <- c(0.07,.3)
curve(vol(params.rHeston)(x),from=-.5,to=.5,col=my.col[1],ylim=yrange,lwd=2,
H.vec <- params.rHeston$H + seq(0.1,0.4,0.1)
for (j in 1:4)
    {
    curve(vol(sub.H(H.vec[j]))(x),from=-.5,to=.5,col=my.col[j+1],lty=1,add=T
    }
```

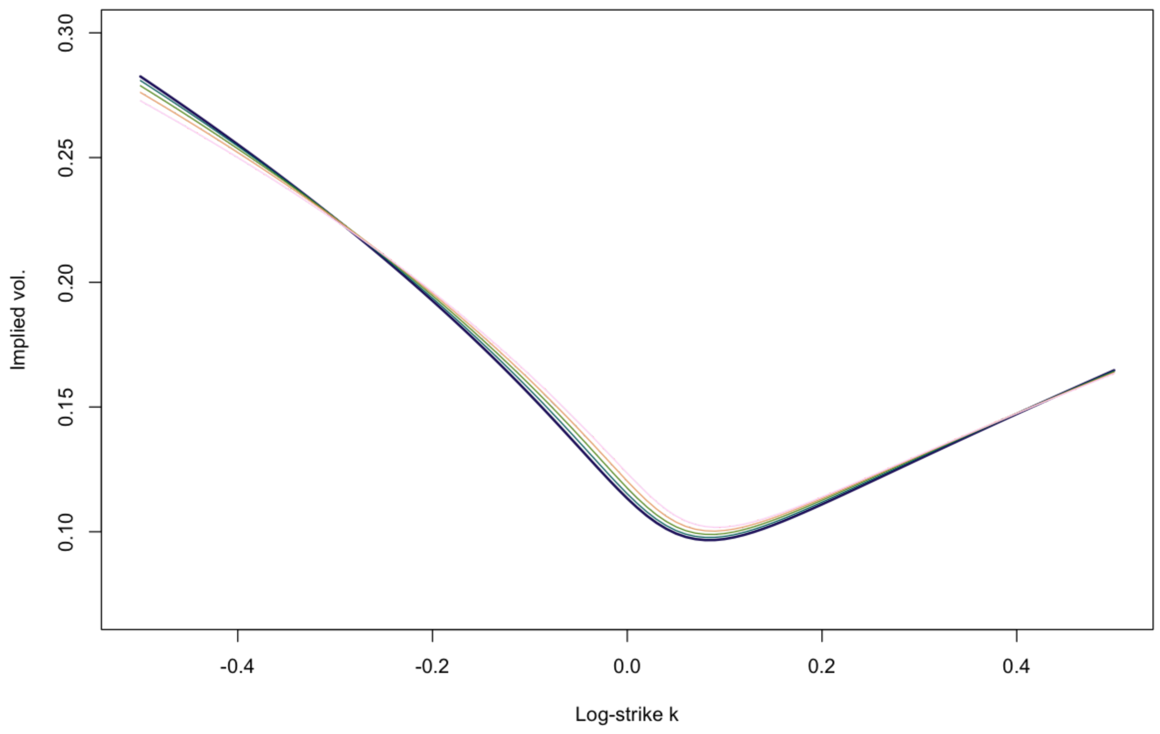

Figure 5: The dotted lines are 1 year smiles with $H \mapsto H + \{0.1, 0.2, 0.3, 0.4\}$.

Sensitivity of the rough Heston 1 week smile to H

A function to draw the 1-week smile:

```
In [21]: vol <- function(params)function(k){
             phi <- phiRoughHestonRational(params, xiCurve, h.approx=h.Pade33, n=20)
             sapply(k,function(x){impvol.phi(phi)(x,1/52)})}
```

```
In [22]: yrange <- c(0.05,.4)
         curve(vol(params.rHeston)(x),from=-.15,to=.15,col=my.col[1],ylim=yrange,lwd=
         H.vec <- params.rHeston$H + seq(0.1,0.4,0.1)
         for (j in 1:4)
             {
             curve(vol(sub.H(H.vec[j]))(x),from=-.15,to=.15,col=my.col[j+1],lty=1,lwc
             }
```

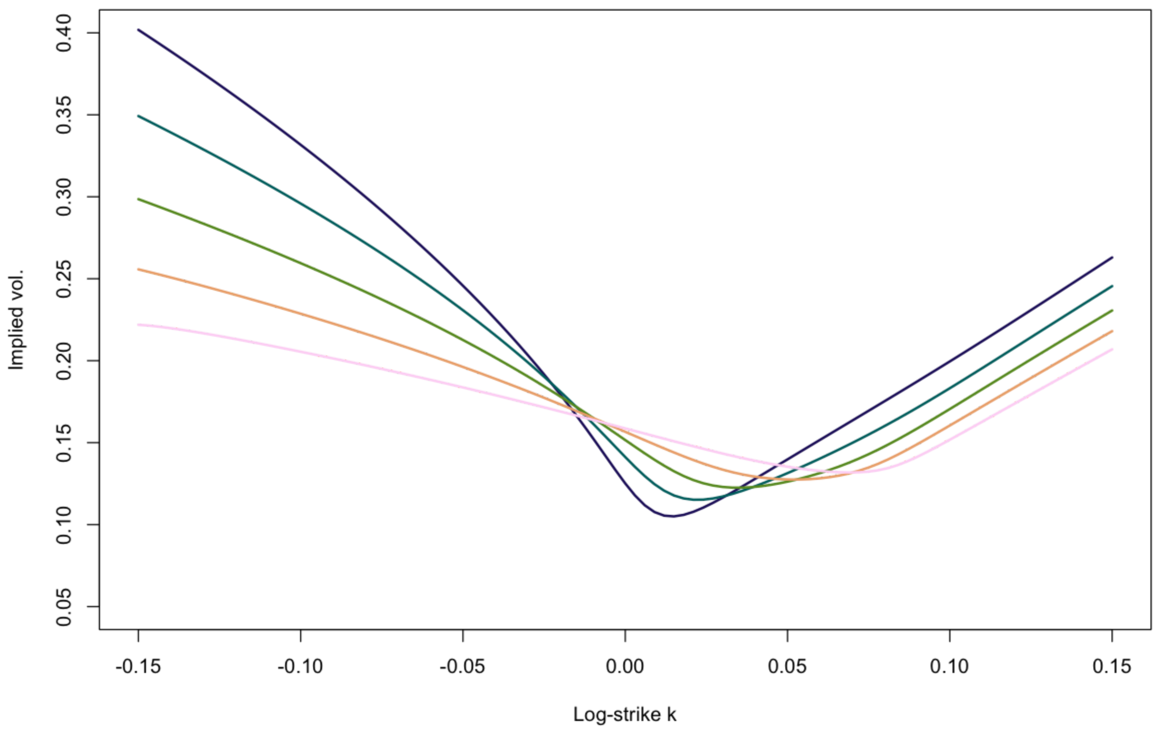

Figure 6: The dotted lines are 1 week smiles with $H \mapsto H + \{0.1, 0.2, 0.3, 0.4\}$. The smile flattens as we increase H.

Ease of calibration of rough volatility models

- Rough volatility models are typically very parsimonious.
- Moreover, from the above sensitivity analyses, the effect of changing each parameter is clear:
  - ν controls curvature
  - ρ controls slope/orientation
  - H controls explosivity
- Contrast this with the classical Heston model where volatility of volatility and mean reversion are competing effects.

Dynamics of the rough Heston volatility surface

- All rough stochastic volatility models have essentially the same implications for the shape of the volatility surface.
- Recall from Lecture 2 that we can differentiate between models by examining how ATM skew depends on ATM volatility keeping model parameters fixed.

- In Figure 7, we that rough Heston dynamics are not consistent with empirical dynamics, in contract to rough Bergomi.

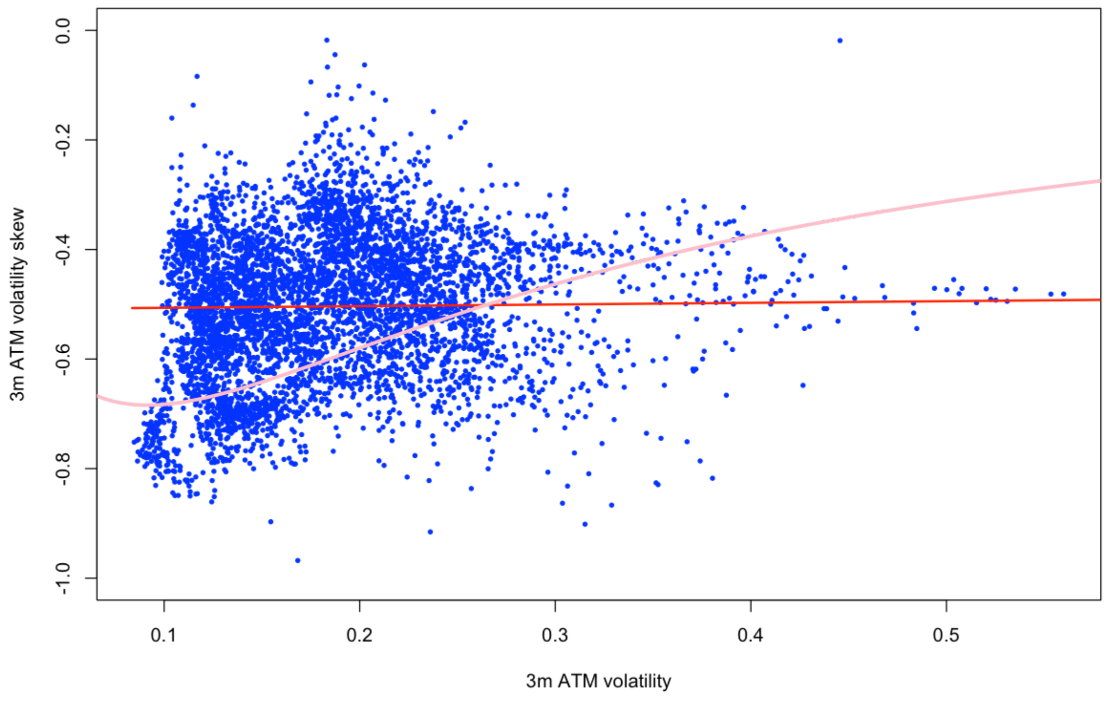

Figure 7: Blue points are empirical 3-month ATM volatilities and skews (from Jan-1996 to today); the red line is the rough Bergomi computation with the above parameters; the pink curve is the rough Heston computation.

Fit rough Heston on February 15, 2023

- Recall that in Lecture 3, we estimated the (strange-looking) parameters:

$H 0.511599077350975$nu 1.04560609788258 $rho - 0.971373372481705$lambda 2.23552496279593

- Not surprisingly, these parameters generate pretty bad-looking smiles.
- However, surprisingly, fitting to just 5 points of each of the six slices in our earlier subset of smiles, we get rather similar parameters:

Load the implied volatility data

In [23]:
```
load("spxIvols20230215.rData")

ivolData <- spxIvols20230215
ivolData <- ivolData[!is.na(ivolData$Bid),]
head(ivolData)
```

A data.frame: 6 × 7

	Expiry	Texp	Strike	Bid	Ask	Fwd	CallMid
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
29	20230216	0.002737851	3725	0.6790964	0.7226482	4146.742	421.8169
30	20230216	0.002737851	3730	0.6712863	0.7144126	4146.742	416.8169
31	20230216	0.002737851	3740	0.6556784	0.6979523	4146.742	406.8169
32	20230216	0.002737851	3750	0.6400859	0.6815060	4146.742	396.8169
33	20230216	0.002737851	3760	0.6245079	0.6650726	4146.742	386.8169
34	20230216	0.002737851	3770	0.6089435	0.6486510	4146.742	376.8169

Load the forward variance curve

In [24]:
```r
load(file="xi20230215.rData")

xi <- xiCurveObj$getForwardVarCurve()
```

Extract six slices

In [25]:
```r
expiries <- unique(ivolData$Texp)

ive <- ivolData[ivolData$Texp %in% expiries[c(2,10,21,28,34,42)],]
ive$kk <- log(ive$Strike/ive$Fwd)
ive$tt <- ive$Texp

head(ive)
```

A data.frame: 6 × 9

	Expiry	Texp	Strike	Bid	Ask	Fwd	CallMid	
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<
270	20230217	0.005475702	3555	0.6345745	0.7107853	4146.459	591.5214	-0.1538
271	20230217	0.005475702	3560	0.6291774	0.7048296	4146.459	586.5214	-0.1524
272	20230217	0.005475702	3565	0.6237852	0.6988788	4146.459	581.5214	-0.151
273	20230217	0.005475702	3570	0.6183979	0.6929328	4146.459	576.5214	-0.149
274	20230217	0.005475702	3575	0.6130153	0.6869916	4146.459	571.5214	-0.1482
275	20230217	0.005475702	3580	0.6076374	0.6810552	4146.459	566.5214	-0.146

Compute `modelVol`

In [26]:
```r
fit.5 <- list(H=0.53,rho=-.64,nu=1.11,lambda=1.28)
```

```
phi3 <- phiRoughHestonRational(fit.5, xi, h.approx=h.Pade33, n=20)
vol3 <- Vectorize(function(k,tau){impvol.phi(phi3)(k,tau)})
```

In [27]:

```
system.time(ive$modelVol <- vol3(ive$kk,ive$tt))
```

```
  user  system elapsed 
65.926   0.958  67.203 
```

Plot the smiles

In [28]:

```
res.plot6 <- plotIvols(ive,modelVol=T)
```

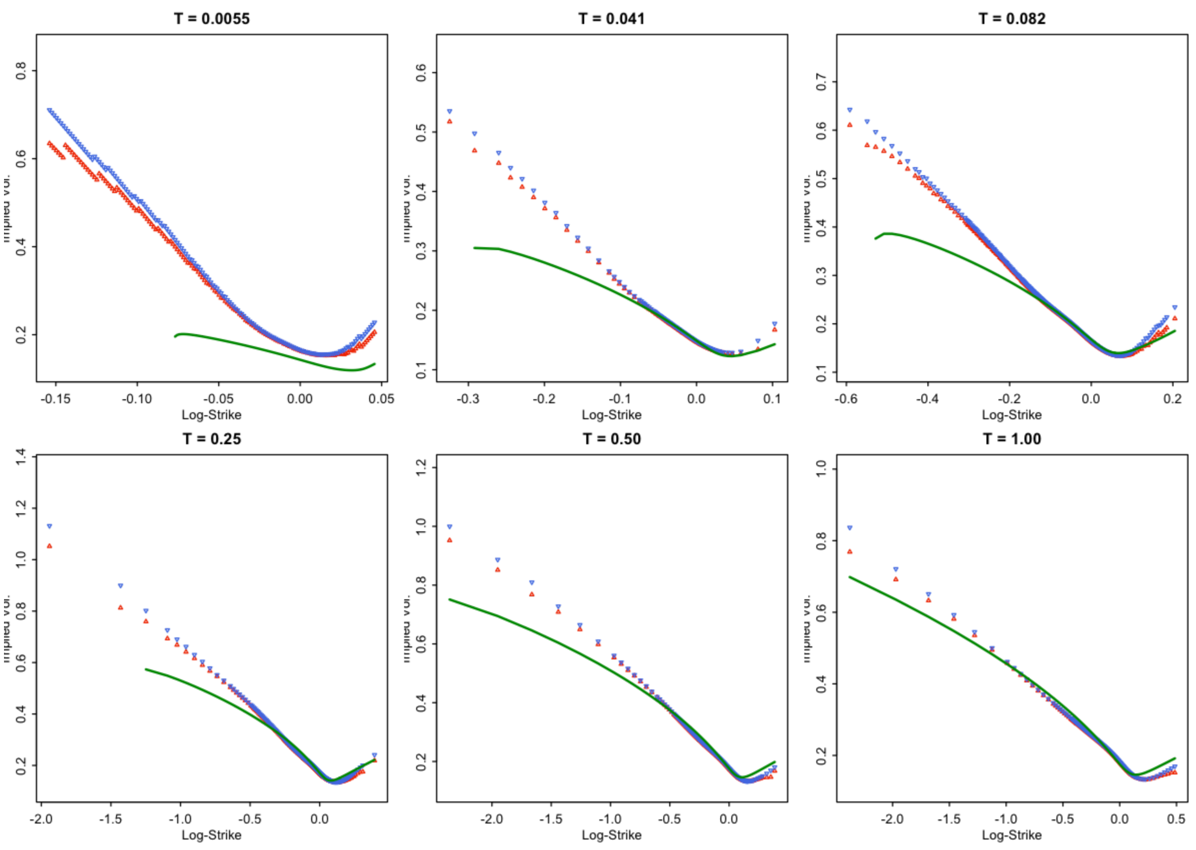

Figure 8: Six rough Heston smiles (green) with `fit.5` parameters superimposed on February 15, 2023 SPX smiles.

Comments on Figure 8

- With just one computation for 6 slices taking 70 seconds, calibration with this code is not practical.
  - We would need, at the least, to use something like the SINC method of [Baschetti et al.][3].
- The parameters look crazy - very close to classical Heston.
  - And inconsistent with the scaling of VIX futures seen in Lecture 2.

    - But at least they are consistent with the leverage contract estimates of Lecture 3.

- With just one computation for 6 slices taking 60 seconds, calibration with this code is not practical.

Why Monte Carlo?

- The rational approximation allows us to value European options only.
- We may be (are) interested in valuing other kinds of option. We need a Monte Carlo scheme.
    - Also, we have a rational approximation for rough Heston only.
        - The Monte Carlo scheme can have any kernel.

Andersen's Quadratic Exponential (QE) scheme

- [Andersen][1] came up with the following clever scheme for simulating the Heston model that guarantees non-negativity of the simulated V process while matching mean and variance at each step.
- Define

$$\psi = \frac{\mathrm{var}_t\left[V_\Delta\right]}{\mathbb{E}_t[V_\Delta]^2}.$$

- Expectation and variance are wrt $\mathcal{F}_t$.

Algorithm ψ^-

If $\psi \leq 2$, simulate V_Δ as

$$V_\Delta = \alpha\left(\beta + Z\right)^2$$

with $Z \sim N(0,1)$ and

$$\beta^2 = \frac{2}{\psi} - 1 + \sqrt{\frac{2}{\psi}}\sqrt{\frac{2}{\psi} - 1}; \quad \alpha = \frac{\mathbb{E}\left[V_\Delta\right]}{1+\beta^2}.$$

Algorithm ψ^+

On the other hand, if $\psi \geq 1$, simulate v_Δ as

$$v_\Delta = -1_{U<p}\,\gamma\,\log\frac{U}{p}$$

with $U_n \sim \mathcal{U}(0,1)$ and

$$p = \frac{2}{1+\psi}; \quad \gamma = \frac{1}{2}\,\mathbb{E}\left[v_\Delta\right]\,(1+\psi)\,.$$

- It is straightforward to check that means and variances are correctly matched in both cases.
    - The quadratic and exponential distributions are chosen because they have similar shapes to the true distribution in their respective regions of applicability.
- Since the two regions of applicability overlap, Andersen suggests to use algorithm ψ^- if $\psi < 3/2$ and algorithm ψ^+ if $\psi \geq 3/2$.
- Note that the algorithms $\psi^\pm$ depend only on expectation and variance so this scheme should work whenever these can be computed or approximated.
    - In particular in the case of affine forward variance models.

Function to compute ψ

In [29]:
```
psi <- function(params,dt)function(v){

    eta <- params$eta
    lam <- params$lambda
    vbar <- params$vbar

    eldt <- exp(-lam*dt)

    ev <- (v-vbar)*eldt+vbar
    varv <- eta^2/lam*(eldt*(1-eldt)*(v-vbar)+vbar/2*(1-eldt^2))

    return(varv/ev^2)

}
```

Code to implement ψ^- and ψ^+

In [30]:
```
psiM
```

```
function (psi, ev, w)
{
    beta2 <- 2/psi - 1 + sqrt(2/psi) * sqrt(abs(2/psi - 1))
    alpha <- ev/(1 + beta2)
    vf <- alpha * (sqrt(abs(beta2)) + w)^2
    return(vf)
}
```

In [31]:
```
psiP
```

```
function (psi, ev, u)
{
    p <- 2/(1 + psi)
    gam <- ev/2 * (1 + psi)
    vf <- -(u < p) * gam * log(u/p)
    return(vf)
}
```

Affine forward variance (AFV) models

- Now, followng [Efficient Simulation][5], we explain how to simulate affine forward variance (AFV) models in general.
    - In particular, rough affine models.
- In order to do this, we extend Andersen's QE scheme to get the mean and variance correct at each step.
- And we adapt the hybrid scheme of [Bennedsen et al.][2].

Discretization of the spot and variance processes

From the AFV dynamics,

$$d\xi_t(u) = \kappa(u-t)\,\sqrt{V_t}\,dW_t,$$

it follows that

$$\begin{aligned} V_T = \xi_T(T) = \xi_0(T) + \int_0^T\, d\xi_s(T) \\ = \xi_0(T) + \int_0^T\, \kappa(T-s)\,\sqrt{V_s}\,dW_s. \end{aligned}$$

Formal representation of the V process

- Wlog, let $t = 0$ and $\xi(u) = \xi_0(u)$. Let the time step $\Delta = T/n$ where n is the number of steps.

- As in [Bennedsen et al.][2], we have the following exact decomposition:

$$V_{n\Delta} = \xi(n\Delta) + \sum_{k=1}^{n} \int_{(k-1)\Delta}^{k\Delta} \kappa(n\Delta - s)\, \sqrt{V_s}\, dW_s.$$

Discretization of the V-process

- With simpler notation,

$$V_n = \xi_n + \sum_{k=1}^{n} \int_{(k-1)\Delta}^{k\Delta} \kappa(n\Delta - s)\, \sqrt{V_s}\, dW_s =: \hat{\xi}_n + u_n,$$

  where the $\mathcal{F}_{n-1}$-adapted variable $\hat{\xi}_n$ is given by

$$\hat{\xi}_n = \mathbb{E}\left[V_n \middle| \mathcal{F}_{n-1} \right] = \xi_n + \sum_{k=1}^{n-1} \int_{(k-1)\Delta}^{k\Delta} \kappa(n\Delta - s)\, \sqrt{V_s}\, dW_s,$$

  and the martingale increment u_n by

$$u_n = \int_{(n-1)\Delta}^{n\Delta} \kappa(n\Delta - s)\, \sqrt{V_s}\, dW_s.$$

The X-process

- We also need to simulate the nth increment of the component of the log-stock price process $X = \log S$ parallel to the volatility process,

$$\chi_n = \int_{(n-1)\Delta}^{n\Delta} \sqrt{V_s}\, dW_s.$$

  - We write the increments as χ_n to emphasize that they should be approximately χ^2 distributed random variables.

- We then have the following discretization of the X process:

$$X_n = X_{n-1} - \tfrac{1}{4}\,(V_n + V_{n-1})\,\Delta \\ + \sqrt{1-\rho^2}\,\sqrt{\bar{V}_n\,\Delta}\,Z_n^\perp + \rho\,\chi_n,$$

where $Z_n^\perp$ is standard normal, independent of χ_n and u_n.

Choices of kernel

Let $\tilde\eta = \eta\,\sqrt{2H}$. The code uses the gamma kernel $\kappa(\tau) = \tilde\eta\,\tau^{\alpha-1}\,e^{-\lambda\tau}$ which has the two special cases

- The *power-law* kernel (rough Heston with $\lambda = 0$)

$$\kappa(\tau) = \sqrt{2H}\,\eta\,\tau^{\alpha-1} =: \tilde\eta\,\tau^{\alpha-1},$$

- and the exponential kernel (classical Heaton)

$$\kappa(\tau) = \tilde\eta\,e^{-\lambda\tau}.$$

- The algorithm can deal with any kernel however.

Some definitions

- We define for $i, j \geq 0$,

$$\mathcal{K}_i(\Delta) = \int_0^\Delta \kappa(s + i\Delta)\,ds;$$
$$\mathcal{K}_{i,j}(\Delta) = \int_0^\Delta \kappa(s + i\Delta)\,\kappa(s + j\Delta)\,ds.$$

- The $\mathcal{K}_{i,j}(\Delta)$ with $i \neq j$ are not in general computable in closed-form but are easy to compute numerically.

Covariances and correlations

- It can be shown that

$$\mathrm{var}[u_n|\mathcal{F}_{n-1}] = \bar{V}_n\,\mathcal{K}_{0,0}(\Delta) + \mathcal{O}\left(\Delta^{1+2H}\right),$$

where

$$\bar{V}_n := \frac{1}{2H+1}\left[\hat{\xi}_n + 2H\,V_{n-1}\right].$$

- Similarly

$$\begin{aligned}\mathrm{var}[\tilde{\xi}_{n+1}|\mathcal{F}_{n-1}] &\approx \bar{V}_n\,\mathcal{K}_{1,1}(\Delta)\\ \mathrm{var}[\chi_n|\mathcal{F}_{n-1}] &\approx \bar{V}_n\,\Delta\\ \mathrm{cov}[u_n,\tilde{\xi}_{n+1}|\mathcal{F}_{n-1}] &\approx \bar{V}_n\,\mathcal{K}_{0,1}(\Delta)\\ \mathrm{cov}[u_n,\chi_n|\mathcal{F}_{n-1}] &\approx \bar{V}_n\,\mathcal{K}_0(\Delta)\\ \mathrm{cov}[\chi_n,\tilde{\xi}_{n+1}|\mathcal{F}_{n-1}] &\approx \bar{V}_n\,\mathcal{K}_1(\Delta).\end{aligned}$$

Given a suitable kernel, all of these may be easily computed.

The correlation matrix

- Because variances and covariances in an AFV model are linear in ξ, the correlation matrix takes the simple form.

$$R=\begin{pmatrix}1 & \rho_{u\chi} & \rho_{u\xi}\\ \rho_{u\chi} & 1 & \rho_{\xi\chi}\\ \rho_{u\xi} & \rho_{\xi\chi} & 1\end{pmatrix}.$$

where

$$\begin{aligned}\rho_{u\chi} &= \frac{\mathcal{K}_0(\Delta)}{\sqrt{\Delta}\sqrt{\mathcal{K}_{0,0}(\Delta)}}\\ \rho_{u\xi} &= \frac{\mathcal{K}_{0,1}(\Delta)}{\sqrt{\mathcal{K}_{0,0}(\Delta)}\sqrt{\mathcal{K}_{1,1}(\Delta)}}\\ \rho_{\xi\chi} &= \frac{\mathcal{K}_1(\Delta)}{\sqrt{\Delta}\sqrt{\mathcal{K}_{1,1}(\Delta)}}\end{aligned}$$

are all independent of n.

The power-law kernel

- In the case of the power-law kernel $\kappa(\tau)=\tilde{\eta}\,\tau^{\alpha-1}$, these correlations are functions of H only.

- Specifically

$$\rho_{u\chi}=\frac{\sqrt{2H}}{H+1/2},$$

and the other correlations may be easily computed numerically.

- In Figure 9, we plot these correlations as a function of H.

Code for the correlation functions

In [32]:
```
rho.uchi <- function(H){
    params <- list(al=H+1/2,lam=0,eta=0.8,rho=-0.65, H=H,lam=0)
    del <- 1/10
    k00 <- G00(params)(del)
    k0 <- G0(params)(del)
    k01 <- G01(params)(del)
    k11 <- G11(params)(del)
    k1 <- G1(params)(del)
    return(k0/sqrt(k00*del))
}
```

In [33]:
```
rho.uxi <- function(H){
    params <- list(al=H+1/2,lam=0,eta=0.8,rho=-0.65, H=H,lam=0)
    del <- 1/10
    k00 <- G00(params)(del)
    k0 <- G0(params)(del)
    k01 <- G01(params)(del)
    k11 <- G11(params)(del)
    k1 <- G1(params)(del)
    return(k01/sqrt(k00*k11))
}
```

In [34]:
```
rho.xichi <- function(H){
    params <- list(al=H+1/2,lam=0,eta=0.8,rho=-0.65, H=H,lam=0)
    del <- 1/10
    k00 <- G00(params)(del)
    k0 <- G0(params)(del)
    k01 <- G01(params)(del)
    k11 <- G11(params)(del)
    k1 <- G1(params)(del)
    return(k1/sqrt(k11*del))
}
```

Plot of the correlation matrix in the power-law kernel case

In [35]:
```
leg.txt <- c(expression(rho[mu*chi]),
             expression(rho[mu*xi]),
             expression(rho[chi*xi]))
leg.posn <- "bottomright"
leg.inset <- .05
```

In [36]:
```
curve(Vectorize(rho.uchi)(x),from=1e-12,to=0.5,
     col=my.col[4],xlab="H",ylab="",n=1000,lwd=2,cex.lab=1.5)
curve(Vectorize(rho.uxi)(x),from=1e-12,to=0.5,
     col=my.col[3],add=T,n=1000,lwd=2)
curve(Vectorize(rho.xichi)(x),from=1e-12,to=0.5,
    col=my.col[1],add=T,n=1000,lwd=2)
legend(leg.posn,leg.txt, cex=1.5, inset=.05, col=my.col[c(4,3,1)], lwd=2)
```

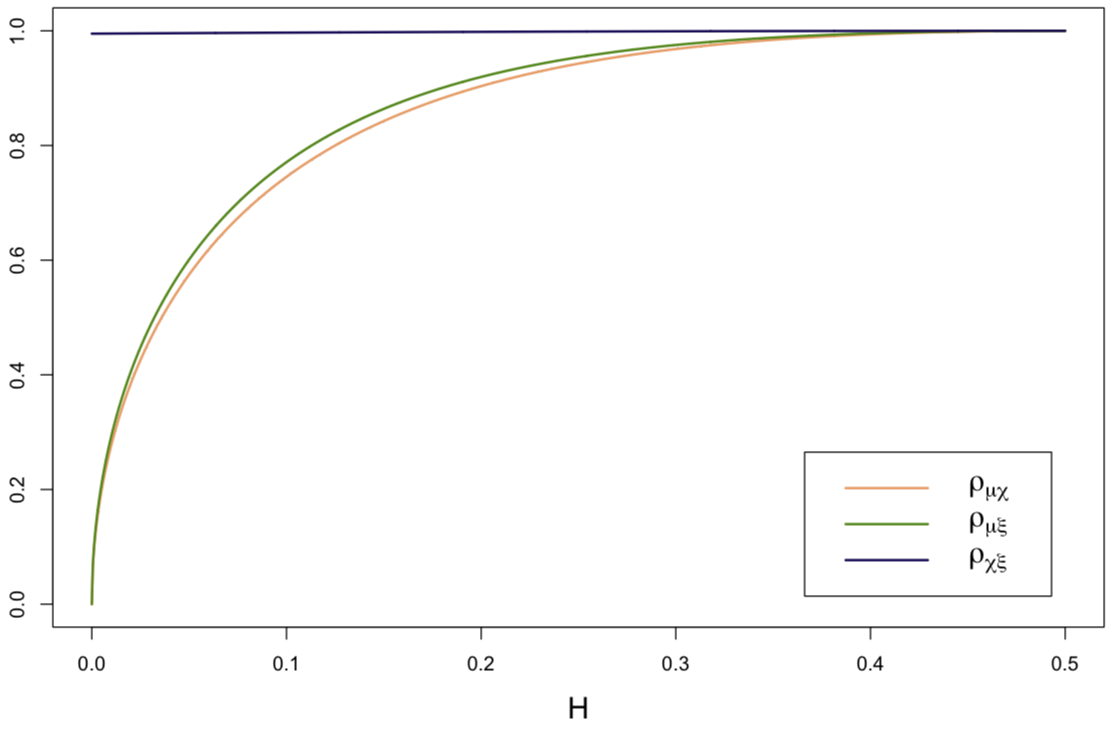

Figure 9: The correlations $\rho_{u\chi}$, $\rho_{u\xi}$, and $\rho_{\xi\chi}$ vs. H in the power-law kernel case.

A further approximation

- By assumption, the kernel behaves as a power-law kernel for Δ sufficiently small.
- Figure 9 thus suggests the following approximation whose motivation is easy to see by thinking of $\mathcal{K}_i(\Delta)$ as Δ times the average value of $\kappa(s+i\Delta)$ over the interval $(0,\Delta]$.

  For $i \geq 0$ and $j \geq 1$,

$$\mathcal{K}_{i,j}(\Delta)\,\Delta \approx \mathcal{K}_i(\Delta)\,\mathcal{K}_j(\Delta).$$

An approximate correlation matrix

With this last approximation,

$$\mathcal{K}_{0,1}(\Delta) \approx \frac{1}{\Delta}\,\mathcal{K}_1(\Delta)\,\mathcal{K}_0(\Delta); \quad \mathcal{K}_{1,1}(\Delta) \approx \frac{1}{\Delta}\,\mathcal{K}_1(\Delta)^2.$$

Substituting these expressions into our earlier expression for the correlation matrix gives

$$\bar{R} = \begin{pmatrix} 1 & \bar{\rho} & \bar{\rho} \\ \bar{\rho} & 1 & 1 \\ \bar{\rho} & 1 & 1 \end{pmatrix},$$

where

$$\bar{\rho} \approx \frac{\mathcal{K}_0(\Delta)}{\sqrt{\mathcal{K}_{0,0}(\Delta)\,\Delta}} = \rho_{u\chi}.$$

Consequences for simulation

- At each step, we need to generate (at least) three random variables: u_n, χ_n, and $\hat{\xi}_{n+1}$.

$$\begin{aligned} u_n &= \int_{(n-1)\Delta}^{n\Delta} \kappa(n\Delta - s)\,\sqrt{V_s}\,dW_s \\ \chi_n &= \int_{(n-1)\Delta}^{n\Delta} \sqrt{V_s}\,dW_s \\ \hat{\xi}_{n+1} &= \xi_{n+1} + \sum_{k=1}^{n} \int_{(k-1)\Delta}^{k\Delta} \kappa((n+1)\Delta - s)\,\sqrt{V_s}\,dW_s. \end{aligned}$$

- When the model is Markovian ($H = 1/2$), we need only generate u_n at the nth time step; χ_n and $\hat{\xi}_{n+1}$ are perfectly correlated with u_n.
    - In practice, in the non-Markovian case ($H < 1/2$), we need only generate one other random variable consistent with the correlation matrix $\bar{R}$.

Average values of the kernel

- Echoing the notation of [Bennedsen et al.][3], let

$$b_j^{\star\,2} = \frac{1}{\Delta}\,\mathcal{K}_{j-1,j-1}(\Delta).$$

- $b_j^{\star\,2}$ thus gives the RMS average of the kernel at the jth lag.

The evolution of the forward variance curve

- The approximation

$$\int_{(k-1)\Delta}^{k\Delta} \kappa((n+1)\Delta - s)\,\sqrt{V_s}\,dW_s \approx b^{\star}_{n+1-k}\,\chi_k$$

gives

$$\hat{\xi}_{n+1} \approx \xi_{n+1} + \sum_{k=1}^{n} b^\star_{n+1-k}\, \chi_k.$$

- Similarly (though not needed for the algorithm), for $m > n$,

$$\mathbb{E}\left[V_m \middle| \mathcal{F}_n\right] \approx \xi_m + \sum_{k=1}^{n} b^\star_{m-k}\, \chi_k.$$

- We see that the entire forward variance curve evolves according to the weighted historical path of the $X = \log S$ process.

A Riemann-sum QE scheme

- Inspired by the Riemann-sum scheme of [Bennedsen et al.][2] and the rough-Donsker scheme of [Horvath et al.][8], we simulate the u_n, $\hat{\xi}_{n+1}$ and χ_n as if all three were perfectly correlated, equivalent to setting $\bar{\rho} = 1$ in (7).
- From Figure 9 such an approximation may be justified if H is not too much less than $\frac{1}{2}$.

The RSQE scheme

1. Given χ_k, for $k < n$, with ϵ very small, compute $\hat{\xi}_n = \max\left[\epsilon,\, \xi_n + \sum_{k=1}^{n-1} b^\star_{n-k+1}\, \chi_k\right]$.
2. With $\mathrm{var}[V_n|\mathcal{F}_{n-1}] = {b^\star_1}^2\, \bar{V}_n\, \Delta$, simulate V_n using the QE scheme.
3. $u_n = V_n - \hat{\xi}_n$.
4. $\hat{\xi}_{n+1} = \xi_{n+1} + \sum_{k=1}^{n} \frac{b^\star_{n-k+1}}{b^\star_1}\, u_k$.
5. Finally, $X_n = X_{n-1} - \frac{1}{4}\left(V_n + V_{n-1}\right)\Delta + \sqrt{1-\rho^2}\,\sqrt{\bar{V}_n\,\Delta}\, Z^\perp_n + \rho\, \chi_n$.

RSQE code

In [37]:
```
RSQE.sim
```

```
function (params, xi)
function(T, paths, steps) {
    library(gsl)
    eta <- params$eta
    lam <- params$lambda
    H <- params$al - 1/2
    rho <- params$rho
    rho2m1 <- sqrt(1 - rho * rho)
    eps.0 <- 1e-10
    W <- matrix(rnorm(steps * paths), nrow = steps, ncol = paths)
    Wperp <- matrix(rnorm(steps * paths), nrow = steps, ncol = paths)
    U <- matrix(runif(steps * paths), nrow = steps, ncol = paths)
    G00p <- Vectorize(G00(params))
    dt <- T/steps
    sqrt.dt <- sqrt(dt)
    tj <- (1:steps) * dt
    xij <- xi(tj)
    G00del <- G00(params)(dt)
    G00j <- c(0, G00p(tj))
    bstar <- sqrt(diff(G00j)/dt)
    bstar1 <- bstar[1]
    u <- array(0, dim = c(steps, paths))
    v <- rep(xi(0), paths)
    xihat <- rep(xij[1], paths)
    x <- numeric(paths)
    y <- numeric(paths)
    w <- numeric(paths)
    for (j in 1:steps) {
        varv <- eta^2 * (xihat + 2 * H * v)/(1 + 2 * H) * G00del
        psi <- varv/xihat^2
        vf <- ifelse(psi < 3/2, psiM(psi, xihat, W[j, ]), psiP(psi,
            xihat, U[j, ]))
        u[j, ] <- vf - xihat
        dw <- (v + vf)/2 * dt
        w <- w + dw
        dy <- as.numeric(u[j, ])/(eta * bstar1)
        y <- y + dy
        x <- x - dw/2 + sqrt(dw) * as.numeric(rho2m1 * Wperp[j,
            ]) + rho * dy
        btilde <- rev(bstar[2:(j + 1)])
        if (j < steps) {
            xihat <- xij[j + 1] + as.numeric(btilde %*% u[1:j,
                ])/bstar1
        }
        xihat <- ifelse(xihat > eps.0, xihat, eps.0)
        v <- vf
    }
    res <- list(x = x, v = v, w = w)
    return(res)
}
```

Classical Heston with RSQE

- Classical Heston has $H = \frac{1}{2}$ and the exponential kernel is a special case of the gamma kernel.
    - Let's apply the RSQE code to the classical Heston case.

In [38]:
```
params.cHeston <- list(al=1,eta=0.8,rho=-0.65, H=.5,lambda=1,v=0.04,vbar=0.0
xi0 <- function(s){0.04+0*s} # The forward variance curve
```

The following function computes classical Heston implied volatilities using the classical solution.

In [39]:
```
impvolHeston <- function(params)Vectorize(
                        function(k,t){impvol.phi(phiHeston(params))(k,t)},
                                                            vectorize.args = "
```

Run the RSQE Monte Carlo

In [40]:
```
system.time(res.128.RSQE <- RSQE.sim(params.cHeston,xi0)(T=1, paths=1e5, ste
```

```
  user  system elapsed 
 9.873   3.324  13.224 
```

In [41]:
```
S.128.RSQE <- exp(res.128.RSQE$x)
```

Why is RSQE slow compared to Andersen's QE scheme?

- The reason is the convolution step $\hat{\xi}_n = \max\left[\epsilon,\, \xi_n + \sum_{k=1}^{n-1} b^\star_{n-k+1}\,\chi_k\right]$.

- In the case of the exponential kernel,

$$b^{\star\,2}_j = \frac{1}{\Delta}\,\mathcal{K}_{j-1,j-1}(\Delta) = \eta^2\,\frac{1}{\Delta}\int_{(j-1)\Delta}^{j\Delta} e^{-2\lambda s}\,ds = e^{-2(j-1)\Delta}\,b^{\star\,2}_1.$$

  so rather than compute the convolution at each step, we need only keep track of the exponentially weighted moving average of the χ_j.

  - That would save a lot of time!
- If the kernel is not exponential, we are out of luck.

Compare RSQE with exact classical Heston smile

In [42]:
```
kk <- seq(-.8,.4,.02)
smile.128.RSQE <- ivS(S.128.RSQE, T=1, exp(kk))
exactHestonVols.cHeston.kk <- impvolHeston(params.cHeston)(kk,1)
options(repr.plot.width=10,repr.plot.height=7,repr.plot.res=150)
```

Plot the smiles

In [43]:
```
plot(kk,smile.128.RSQE,col=rd,lwd=2,type="l",
     xlab="Log-strike k", ylab = "Implied vol.",cex.lab=1.5)
lines(kk,exactHestonVols.cHeston.kk,col=bl,lwd=2,lty=2)
```

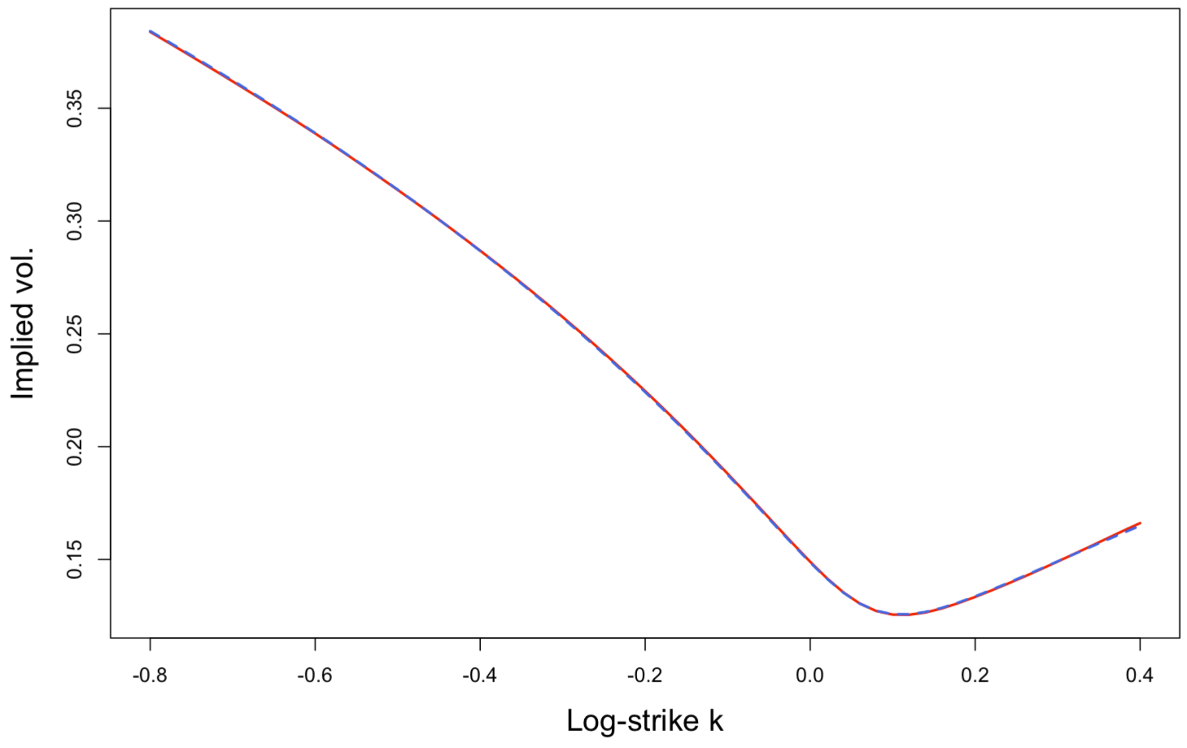

Figure 10: Exact and RSQE 1-year classical Heston smiles compared.

Plot the smile errors

In [44]:
```
plot(kk,smile.128.RSQE-exactHestonVols.cHeston.kk,col=rd,lwd=2,type="l",
     xlab="Log-strike k", ylab = "Implied vol. error",cex.lab=1.5,ylim=c(-.0
abline(h=.001,lty=2)
abline(h=-.001,lty=2)
```

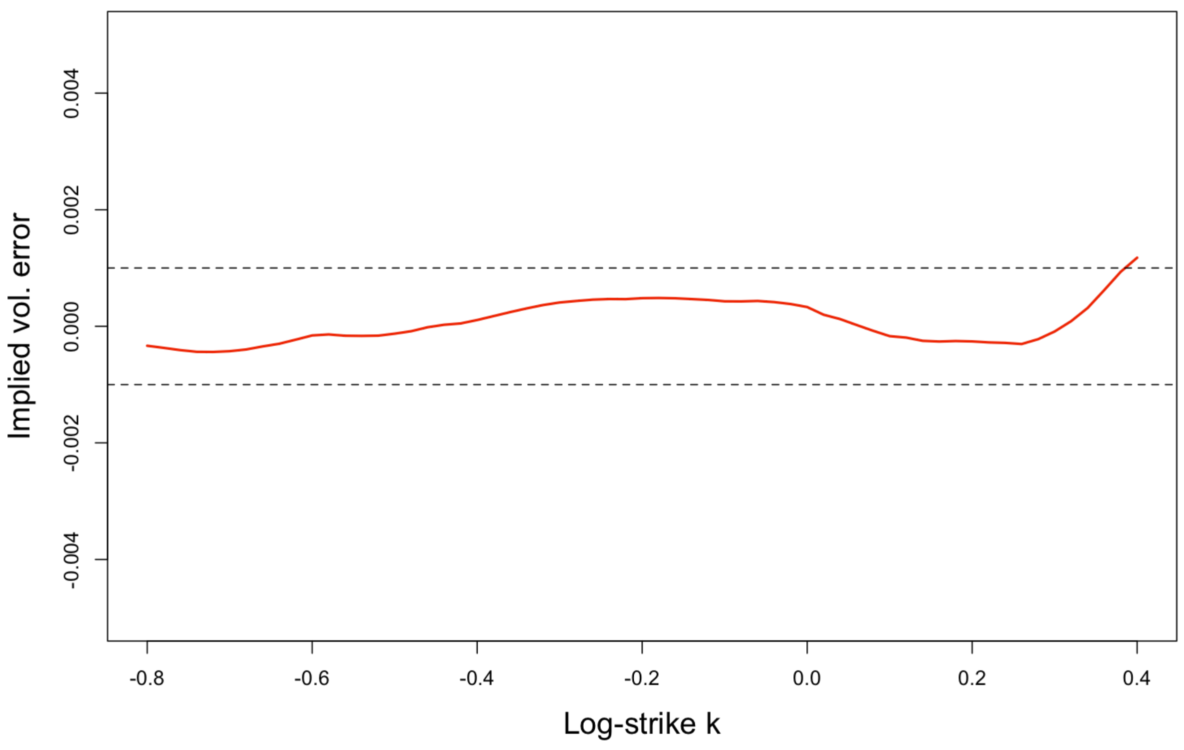

Figure 11: 1-year classical Heston smile errors with BCC2 parameters, using the RSQE scheme.

A hybrid QE scheme

- The RSQE scheme matches unconditional means and variances at each step but it does not match the covariance structure of the process.
- For example, consider the conditional covariance between u_n and χ_n which is given by

$$\mathrm{cov}[u_n, \chi_n | \mathcal{F}_{n-1}] = \int_{(n-1)\Delta}^{n\Delta} \kappa(n\Delta - s)\, \mathbb{E}\left[V_s | \mathcal{F}_{n-1}\right] ds \approx \bar{V}_n\, \mathcal{K}_0(\Delta).$$

- The RSQE scheme sets $u_n = b_1^\star\, \chi_n$ so that

$$\mathrm{cov}[u_n, \chi_n | \mathcal{F}_{n-1}] \approx b_1^\star\, \mathrm{var}[\chi_n | \mathcal{F}_{n-1}] = \bar{V}_n \sqrt{\mathcal{K}_{0,0}(\Delta)\, \Delta},$$

which is equivalent to the approximation

$$\mathcal{K}_0(\Delta) \approx \sqrt{\mathcal{K}_{0,0}(\Delta)\, \Delta}.$$

- This approximation, though accurate for small Δ when the kernel κ has no singularity at zero, is obviously very inaccurate when H is small.

- The essence of the hybrid scheme with $\kappa = 1$ of [Bennedsen et al.][2] is to correct the error in the approximation $\mathcal{K}_0(\Delta) \approx \sqrt{\mathcal{K}_{0,0}(\Delta)\,\Delta}.$ by simulating another random variable, uncorrelated with u_n, so as to match the covariance of u_n and χ_n.
  - For this, we need a bivariate version of Andersen's QE scheme.

A bivariate version of Andersen's QE scheme

- As before, let

$$\begin{aligned} u_n &= \int_{(n-1)\Delta}^{n\Delta} \kappa(n\Delta - s)\,\sqrt{V_s}\,dW_s \\ \chi_n &= \int_{(n-1)\Delta}^{n\Delta} \sqrt{V_s}\,dW_s. \end{aligned}$$

- Linear regression gives

$$u_n \approx \beta_{u\chi}\,\chi_n + \varepsilon_n,$$

  where $\beta_{u\chi} = \mathcal{K}_0(\Delta)/\Delta$, and ε_n and χ_n are uncorrelated.

- Since $V_n = \hat{\xi}_n + u_n \geq 0$, we must ensure that $\beta_{u\chi}\,\chi_n + \varepsilon_n + \hat{\xi}_n \geq 0$.

- We now present a bivariate QE scheme to achieve this.

A bivariate QE scheme

- Let χ_n and ε_n be generated independently using the QE scheme with the following conditional means and variances:

$$\begin{aligned}\mathbb{E}\left[\beta_{u\chi}\,\chi_n|\mathcal{F}_{n-1}\right] &= \tfrac{1}{2}\hat{\xi}_n; \quad \mathbb{E}\left[\varepsilon_n|\mathcal{F}_{n-1}\right] = \tfrac{1}{2}\hat{\xi}_n;\\ \operatorname{var}[\chi_n|\mathcal{F}_{n-1}] = \bar{V}_n\,\Delta; &\quad \operatorname{var}[\varepsilon_n|\mathcal{F}_{n-1}] = \bar{V}_n\,\left(\mathcal{K}_{0,0}(\Delta) - \tfrac{1}{\Delta}\mathcal{K}_0(\Delta)^2\right).\end{aligned}$$

- Then $V_n = \beta_{u\chi}\,\chi_n + \varepsilon_n + \hat{\xi}_n \geq 0$.
- Moreover, with $u_n = \beta_{u\chi}\,\chi_n + \varepsilon_n$,

$$\operatorname{var}[u_n|\mathcal{F}_{n-1}] = \bar{V}_n\,\mathcal{K}_{0,0}(\Delta); \quad \operatorname{cov}[u_n, \chi_n|\mathcal{F}_{n-1}] = \bar{V}_n\,\mathcal{K}_0(\Delta).$$

The hybrid QE (HQE) scheme

We summarize the resulting hybrid QE (HQE) scheme below:

1. Given χ_k, for $k < n$, with ϵ very small, compute $\hat{\xi}_n = \max\left[\epsilon, \xi_n + \sum_{k=1}^{n-1} b^\star_{n-k+1}\,\chi_k\right]$.
2. Simulate χ_n and ε_n using the bivariate QE scheme
3. $V_n = \hat{\xi}_n + \frac{1}{\Delta}\mathcal{K}_0(\Delta)\,\chi_n + \varepsilon_n$.
4. Finally, $X_n = X_{n-1} - \frac{1}{4}\,(V_n + V_{n-1})\,\Delta + \sqrt{1-\rho^2}\,\sqrt{\tilde{V}_n\,\Delta}\,Z_n^\perp + \rho\,\chi_n$, where $\tilde{V}_n = \frac{1}{2}(V_n + V_{n-1})$.

Rough Heston kernel parameterizations

- The gamma kernel with $\lambda = 0$ used by the HQE scheme has

$$\kappa(\tau) = \sqrt{2H}\,\eta\,\tau^{\alpha-1}$$

.

- On the other hand, when $\lambda = 0$, the rough Heston kernel (used in the Padé approximation for example) takes the form

$$\kappa(\tau) = \nu \, \frac{\tau^{\alpha-1}}{\Gamma(\alpha)},$$

- So ν and η are related as

$$\eta = \frac{\nu}{\sqrt{2H}\,\Gamma(\alpha)}.$$

Rough Heston parameters

- We choose rough Heston parameters to give roughly the same 1-year smile as the classical Heston model, with $H = 0.05$:

In [45]:
```
params.rHeston <- list(nu=0.45, eta=.45/(sqrt(2*.05)*gamma(0.55)),rho=-0.65,
xi0 <- function(s){0.04+0*s} # The forward variance curve
```

Compute the rational approximation to the rough Heston smile

In [46]:
```
volPade <- function(h.approx)function(params,xi)function(k){
    phi <- phiRoughHestonRational(params, xi, h.approx, n=20)
    sapply(k,function(x){impvol.phi(phi)(x,1)})}

volPade.44 <- volPade(h.approx = h.Pade44)(params.rHeston,xi0)(kk)
volPade.55 <- volPade(h.approx = h.Pade55)(params.rHeston,xi0)(kk)
```

Plot the classical Heston and rough Heston smiles

In [47]:
```
plot(kk,volPade.44,col=bl,lwd=2, type="l",
     xlab="Log-strike k", ylab = "Implied vol.",cex.lab=1.5)
lines(kk,exactHestonVols.cHeston.kk,col=rd,lwd=2)
legend("topright",c("Rough Heston","Classical Heston"), cex=1.5, inset=.05,
       lty=1,col=c(bl,rd), lwd=2)
```

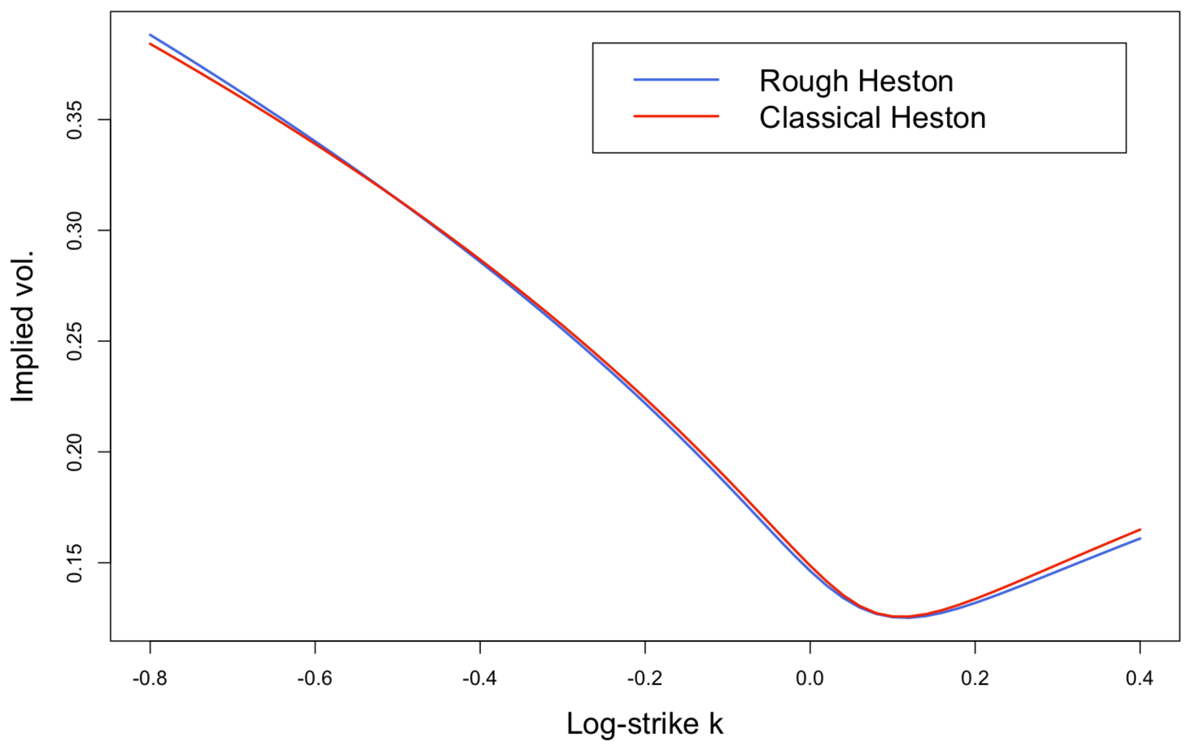

Figure 12: Classical Heston smile with `cHeston` parameters and rough Heston smile with `rHeston` parameters superimposed.

Code for gamma kernel used in the HQE code

In [48]:
```
gGamma
```

```
function (params)
function(tau) {
    al <- params$al
    H <- al - 1/2
    lam <- params$lam
    return(sqrt(2 * H) * tau^{
        al - 1
    } * exp(-lam * tau))
}
```

The HQE code

In [49]:
```
HQE.sim
```

```
function (params, xi)
function(T, paths, steps) {
    library(gsl)
    nu <- params$eta
    lam <- params$lambda
    H <- params$al - 1/2
    rho <- params$rho
    rho2m1 <- sqrt(1 - rho * rho)
    eps.0 <- 1e-10
    W <- matrix(rnorm(steps * paths), nrow = steps, ncol = paths)
    Wperp <- matrix(rnorm(steps * paths), nrow = steps, ncol = paths)
    Z <- matrix(rnorm(steps * paths), nrow = steps, ncol = paths)
    U <- matrix(runif(steps * paths), nrow = steps, ncol = paths)
    Uperp <- matrix(runif(steps * paths), nrow = steps, ncol = paths)
    dt <- T/steps
    sqrt.dt <- sqrt(dt)
    tj <- (1:steps) * dt
    xij <- xi(tj)
    G0del <- nu * G0(params)(dt)
    G1del <- nu * G1(params)(dt)
    G01del <- nu^2 * G01(params)(dt)
    Gjj <- nu^2 * (Gkk(params)(dt))((1:steps) - 1)
    G00del <- Gjj[1]
    G11del <- Gjj[2]
    bstar <- sqrt(Gjj/dt)
    bstar1 <- bstar[1]
    rho.vchi <- G0del/sqrt(G00del * dt)
    beta.vchi <- G0del/dt
    u <- array(0, dim = c(steps, paths))
    chi <- array(0, dim = c(steps, paths))
    v <- rep(xi(0), paths)
    xihat <- rep(xij[1], paths)
    x <- numeric(paths)
    y <- numeric(paths)
    w <- numeric(paths)
    for (j in 1:steps) {
        xibar <- (xihat + 2 * H * v)/(1 + 2 * H)
        var.eps <- xibar * G00del * (1 - rho.vchi^2)
        psi.chi <- 4 * G00del * rho.vchi^2 * xibar/xihat^2
        psi.eps <- 4 * G00del * (1 - rho.vchi^2) * xibar/xihat^2
        z.chi <- ifelse(psi.chi < 3/2, psiM(psi.chi, xihat/2,
            W[j, ]), psiP(psi.chi, xihat/2, U[j, ]))
        z.eps <- ifelse(psi.eps < 3/2, psiM(psi.eps, xihat/2,
            Wperp[j, ]), psiP(psi.eps, xihat/2, Uperp[j, ]))
        chi[j, ] <- (z.chi - xihat/2)/beta.vchi
        eps <- z.eps - xihat/2
        u[j, ] <- beta.vchi * chi[j, ] + eps
        vf <- xihat + u[j, ]
        vf <- ifelse(vf > eps.0, vf, eps.0)
        dw <- (v + vf)/2 * dt
        w <- w + dw
        y <- y + chi[j, ]
        x <- x - dw/2 + sqrt(dw) * as.numeric(rho2m1 * Z[j, ]) +
            rho * chi[j, ]
        btilde <- rev(bstar[2:(j + 1)])
        if (j < steps) {
            xihat <- xij[j + 1] + as.numeric(btilde %*% chi[1:j,
                ])
        }
        v <- vf
    }
    res <- list(x = x, v = v, w = w)
    return(res)
}
```

In [50]:
```
system.time(res.128.HQE <- HQE.sim(params.rHeston,xi0)(T=1, paths=1e5, steps
```

```
  user  system elapsed
12.585   4.305  16.904
```

In [51]:
```
S.128.HQE <- exp(res.128.HQE$x)
```

Plot the smile

In [52]:
```
kk <- seq(-.8,.4,.02)
smile.128.HQE <- ivS(S.128.HQE, T=1, exp(kk))
```

In [53]:
```
plot(kk,smile.128.HQE,col=rd,lwd=2,type="l",
     xlab="Log-strike k", ylab = "Implied vol.",cex.lab=1.5)
lines(kk,volPade.44,col=bl,lwd=2,lty=2)
# lines(kk,volPade.55,col=gr,lwd=4,lty=3)
```

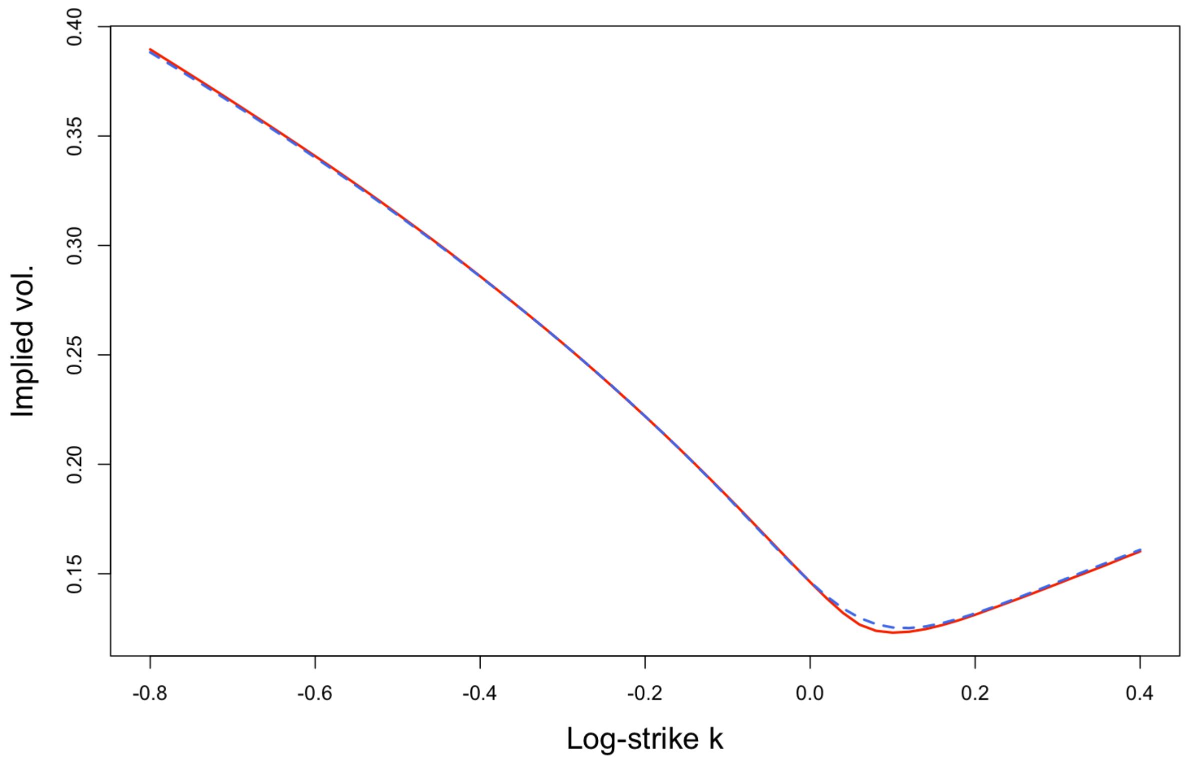

Figure 13: The rough Heston smile with parameters `paramsHQE` . The solid red line is from the Padé approximation; the dashed blue line is from the HQE scheme.

In [54]:
```
plot(kk,smile.128.HQE-volPade.44,col=rd,lwd=2,type="l",
     xlab="Log-strike k", ylab = "Implied vol. error",cex.lab=1.5,ylim=c(-.0
abline(h=.001,lty=2)
abline(h=-.001,lty=2)
```

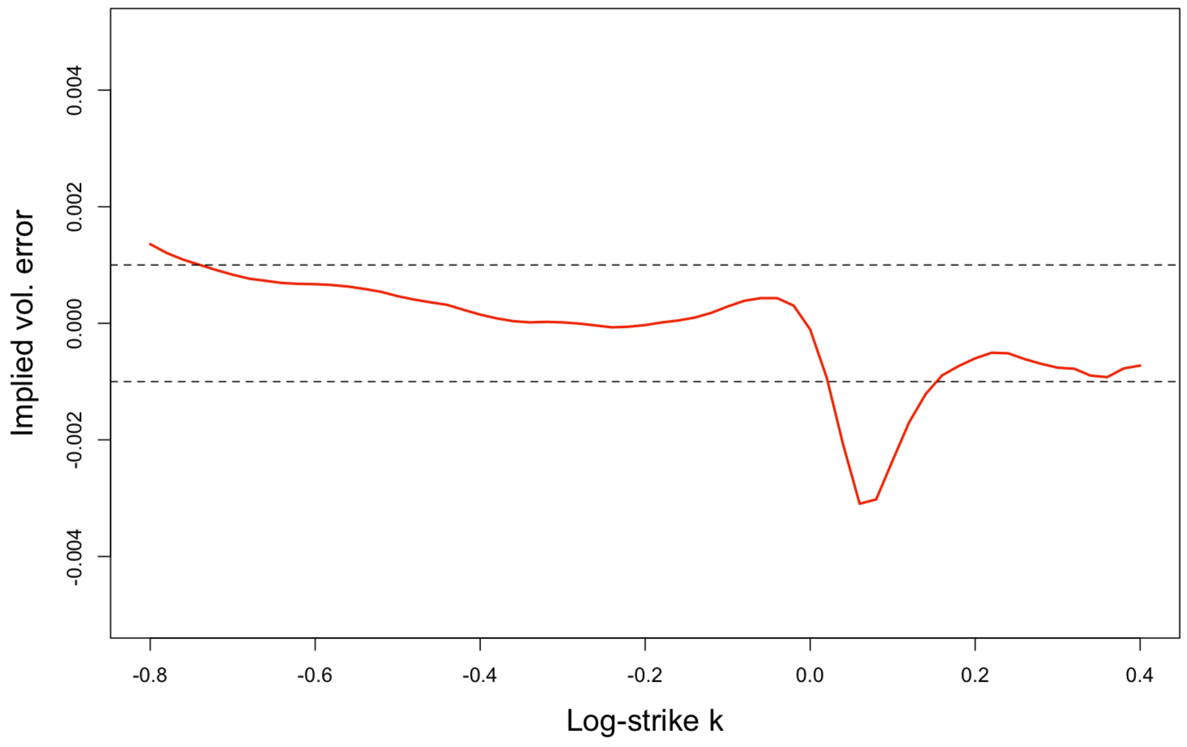

Figure 14: Rough Heston smile errors with `paramsHQE` parameters, using the HQE scheme.

Convergence of the RSQE and HQE schemes

- Surprisingly (in view of Figure 9), we can also use the RQSE scheme to compute Rough Heston smiles.
  - RSQE is slower to converge.
- The following figure from [Efficient Simulation][4] shows that it definitely makes sense to use HQE rather than RSQE for small H.

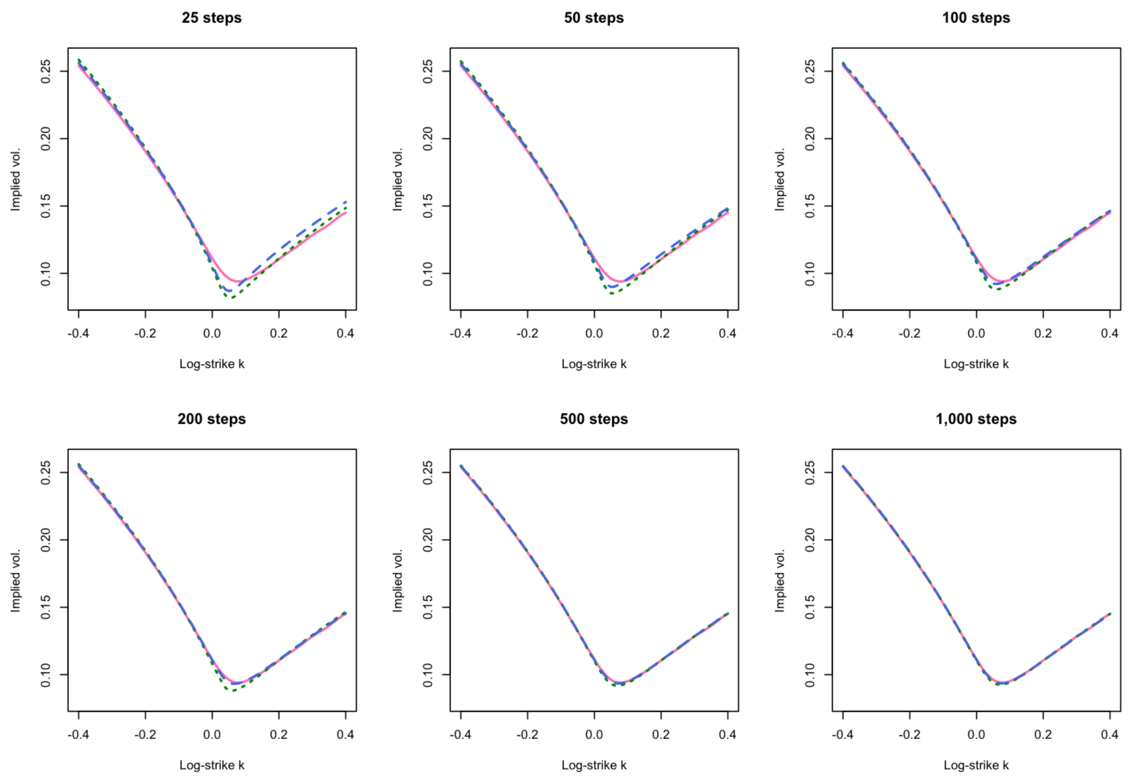

Figure 15: A 1-year rough Heston smile. The pink reference curve is the Adams reference smile. The green-dotted and blue-dashed curves are from RSQE and HQE simulations respectively with 10^6 paths.

Convergence of the HQE scheme

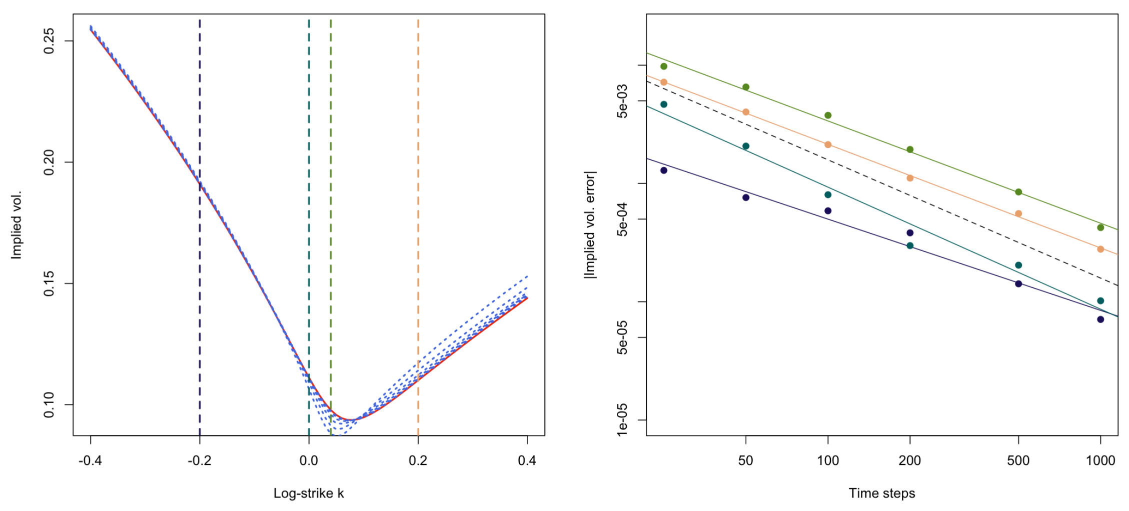

Figure 16: In the LH plot, the pink curve is the Richardson extrapolated HQE smile with 500 steps. The blue dotted lines are HQE smiles S_n computed with $n \in \{25, 50, 100, 200, 500, 1000\}$. In the RH plot, we plot absolute implied volatility

errors. The dashed black line with slope -1 is plotted for reference, clearly demonstrating order one weak convergence. All simulations are with 10^6 paths.

Richardson extrapolation

- It seems that the order of weak convergence of the HQE scheme is one.
- It therefore makes sense to use Richardson extrapolation to increase the order of convergence.

Convergence of Richardson extrapolated HQE smiles

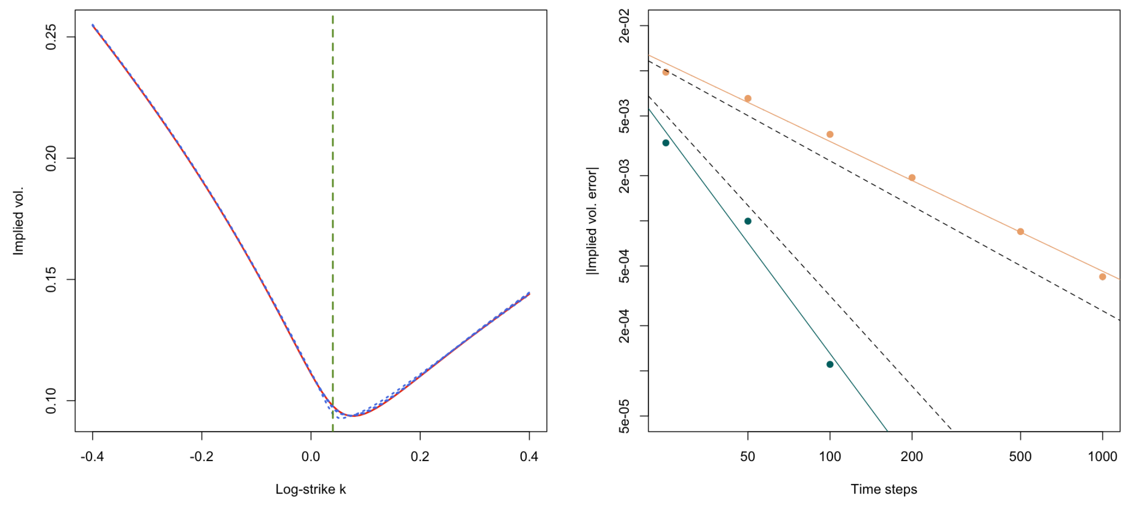

Figure 17: In the LH plot, the pink curve is the 500-step Richardson extrapolated HQE smile. The blue dotted lines are the Richardson-extrapolated smiles S_n^R computed with $n \in \{25, 50, 100\}$. In the RH plot, we plot absolute implied volatility errors vs time steps for log-strike $k = 0.04$, the dashed vertical line in the LH plot, where errors are maximized. Errors without extrapolation are superimposed for reference, as are black-dashed lines with slopes -1 and -2 respectively. We see evidence of order 2 weak convergence of Richardson-extrapolated smiles.

On Markovian approximations

- Eduardo Abi Jaber and Omar El Euch originally suggested expressing rough kernels as sums of exponentials.
  - In the rough Heston case, this is equivalent to solving N classical Heston models.
  - To get reasonable agreement, at least 500 terms are required - very slow!

- More recently, [Bayer and Breneis][4] exhibited an efficient Markovian approximation scheme for the rough Heston model which is apparently competitive with HQE.

Summary of Lecture 4

- We showed how to construct rational approximations of the solution of the rough Heston fractional ODE.
  - These are very fast to compute and thus good for model calibration.
- We presented the hybrid quadratic exponential (HQE) scheme for simulating the rough Heston model.
  - The smiles match!
- However, though rough Heaton is highly tractable, its dynamics are unreasonable.
  - And the parameters we found for February 15, 2023 look weird.

References

1. ^ Leif B G Andersen, Simple and efficient simulation of the Heston stochastic volatility model, *Journal of Computational Finance* **11**(3), 1–42 (2008).
2. ^ Mikkel Bennedsen, Asger Lunde, and Mikko S. Pakkanen, Hybrid Scheme for Brownian Semistationary Processes, *Finance and Stochastics* **21**(4), 931–965(2017).
3. ^ Fabio Baschetti, Giacomo Bormetti, Silvia Romagnoli and Pietro Rossi, The SINC way: A fast and accurate approach to Fourier pricing, *Quantitative Finance* **22**(3), 427-446 (2022).
4. ^ Christian Bayer and Simon Breneis, Efficient option pricing in the rough Heston model using weak simulation schemes, *Quantitative Finance* **24**(9), 1247-1261 (2024).
5. ^ Jim Gatheral, Efficient Simulation of Affine Forward Variance Models, *Risk.net*, *SSRN* 3876680, February (2022).
6. ^ Jim Gatheral and Radoš Radoičić, Rational approximation of the rough Heston solution, *International Journal of Theoretical and Applied Finance* **22**(3) 1950010 (2019).
7. ^ Jim Gatheral and Radoš Radoičić, A generalization of the rational rough Heston approximation, *Quantitative Finance* **24**(2) 329-335 (2024).
8. ^ Blanka Horvath, Antoine Jack Jacquier, and Aitor Muguruza, Functional Central Limit Theorems for Rough Volatility, *Finance and Stochastics* **28**(3), 615–661 (2024).
9. ^ Alan L. Lewis, *Option Valuation under Stochastic Volatility with Mathematica Code* , Finance Press: Newport Beach, CA (2000).

```
In [ ]:
```