R Notebook for Chapter 4: Option Pricing and Sensitivities

Companion Code to Gaussian Process Models for Quantitative Finance

Mike Ludkovski, Jimmy Risk

04/15/2024

This RMarkdown file presents an illustrative use of Gaussian Process surrogates for estimation of option prices and sensitivities, linking to **Chapter 4** of the book. We directly embed `R` code snippets to showcase the straightforward use of the methodology.

Section 4.1: Learning a univariate Heston Pricing Formula

As the first example, we illustrate building a GP model to learn the Heston Pricing formula for Calls. The included CSV file contains a matrix of Calls index by maturity date T and strikes $\mathcal{K}$.

```
df_Calls <- read.csv("data/ch4_HestonPrices.csv")
```

We fix S_0 and learn the input-output map

$$T \mapsto \mathbb{E}^Q[e^{-rT}(S_T - \mathcal{K})_+].$$

For our Gaussian Process Regression model, we utilize the package **DiceKriging** available on CRAN. We select the Matern-52 kernel family k_{M52}, linear prior mean function $\mu(T) = \beta_0 + \beta_1 T$, estimated constant observation noise (`nugget`) and genetic-algorithm optimizer for maximum likelihood estimation of the GP hyperparameters.

The function **km()** is used to fit a GP model based on input-output set $(\mathbf{X}, \mathbf{y})$ and the following parameters:

- *formula* determines the prior mean function $\mu(x)$.
- *covtype* refers to kernel type, taken to be Matern-5/2 below.
- *nugget.estim=TRUE* tells km() to infer the intrinsic (homoskedastic, cell-independent) noise variance σ^2 as part of the model.
- *optim.method="gen"* tells km() to utilize a genetic optimization algorithm from library **rgenoud**, which is recommended for performing hyperparameter MLE by **DiceKriging** authors.
- *control=...* are internal (recommended) parameters of the above optimization algorithm
- See https://cran.r-project.org/web/packages/DiceKriging/DiceKriging.pdf for a more detailed explanation of **km()** options.

The output below shows the MLE of the resulting hyperparameters (range is the **DiceKriging** terminology for ℓ_{len}).

```
train.set <- c(3,5,7,9, 13, 17, 23, 29, 35, 41, 47)  # indices of training dates
train.dates <- c(2/12, 3/12, 4/12, 5/12, 7/12, 9/12, 1, 1.25, 1.5, 1.75, 2)
train.design <- data.frame(T=train.dates, P=(df_Calls[train.set,25]))

gpModel_Heston1D <- km(   formula = y ~ 1 + T ,  # linear prior mean function
                  design =data.frame(T=train.design[,"T"]),  response=train.design[,"P"],
```

```r
                 nugget.estim=TRUE,     # learn the observation noise
                 covtype="matern5_2",   # can also switch to "gauss" or "matern3_2"
                 optim.method="gen",
                 estim.method = "MLE",
                 lower=0.1, upper=2,    # bounds on lengthscale
                 # the "control" parameters below handle speed versus risk of
                 # converging to local minima.  See "rgenoud" package for details
                 control=list(max.generations=100,pop.size=100,
                              wait.generations=8,
                              solution.tolerance=1e-5,
                              maxit = 1000, trace=F
                 ))
print(coef(gpModel_Heston1D))
```

```
## $trend1
## [1] -0.4607411
##
## $trend2
## [1] 9.084496
##
## $range
## [1] 1.459501
##
## $shape
## numeric(0)
##
## $sd2
## [1] 13.74803
##
## $nugget
## [1] 0.0002212341
```

```r
train.set2 <- c(3, 5,7,9, 13, 17, 23, 29, 35, 39, 41, 47)  # indices of training dates  # add 38
train.dates2 <- c(2/12,  3/12, 4/12, 5/12, 7/12,  9/12, 1, 1.25, 1.5, 20/12, 1.75, 2)

train.design2 <- data.frame(T=train.dates2, P=(df_Calls[train.set2,25]))

gpModel_Heston2 <- km(   formula = y ~ 1 + T ,  # linear prior mean function
                 design =data.frame(T=train.design2[,"T"]),  response=train.design2[,"P"],
                 covtype="matern5_2",   # can also switch to "gauss" or "matern3_2"
                 optim.method="gen",
                 coef.trend= c(coef(gpModel_Heston1D)$trend1,coef(gpModel_Heston1D)$trend2),
                 coef.cov= coef(gpModel_Heston1D)$range,
                 coef.var = coef(gpModel_Heston1D)$sd2,
                 nugget = coef(gpModel_Heston1D)$nugget,
                 lower=0.1, upper=2,    # bounds on lengthscale
                 # the "control" parameters below handle speed versus risk of
                 # converging to local minima.  See "rgenoud" package for details
                 control=list(max.generations=100,pop.size=100,
                              wait.generations=8,
                              solution.tolerance=1e-5,
                              maxit = 1000, trace=F
                 ))
```

```r
seqT <- seq(1/12.0, 2.0, by=1/24.0)  # test set of maturities

plot(train.design[,"T"], rep(0, length(train.set)), col="black", pch=16, cex=2,
     ylim=c(-0.09, 0.09), bty='n', xlab="Maturity T (yrs)", ylab="Pricing Error ($)")
points(20/12, 0, col="red",pch=16, cex=3)

# Predict from the fitted GPR model
PriceTest <-  predict(gpModel_Heston1D, newdata=data.frame(T=seqT), type="UK")
lines(seqT, PriceTest$mean  - df_Calls[,25], col="orange", lwd=3)
lines(seqT, PriceTest$lower95  - df_Calls[,25], col="orange", lty=2)
lines(seqT, PriceTest$upper95  - df_Calls[,25], col="orange", lty=2)
PriceTest2 <-  predict(gpModel_Heston2 , newdata=data.frame(T=seqT), type="UK")
lines(seqT, PriceTest2$mean  - df_Calls[,25], col="darkgreen", lwd=3)
lines(seqT, PriceTest2$lower95  - df_Calls[,25], col="darkgreen", lty=2)
lines(seqT, PriceTest2$upper95  - df_Calls[,25], col="darkgreen", lty=2)
legend("topright", c("New GP", "Old GP"),  lwd=c(2,2,2,-1),
       pch=c(-1, -1, -1, 16), col=c( "darkgreen","orange"), bty='n')
```

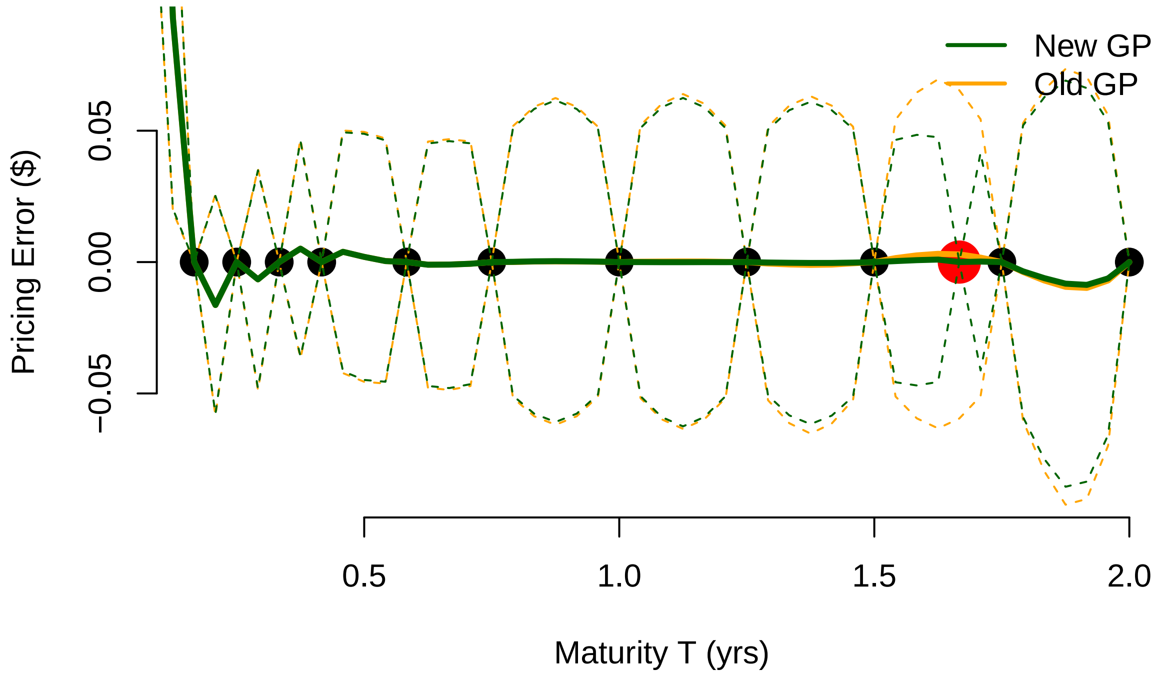


Figure 4.1: Assessing Goodness-of-fit for Heston Call prices

To compare the quality of the GP surrogate, we fit a **quadratic polynomial** and a **cubic spline** with manually pre-selected 5 knots.

```r
quad.model <- lm(train.design$P ~ train.design$T + I(train.design$T^2) )
cubic.spl <-  lm(P ~ bs(T,knots = c(0.5,1)),data = train.design )
```

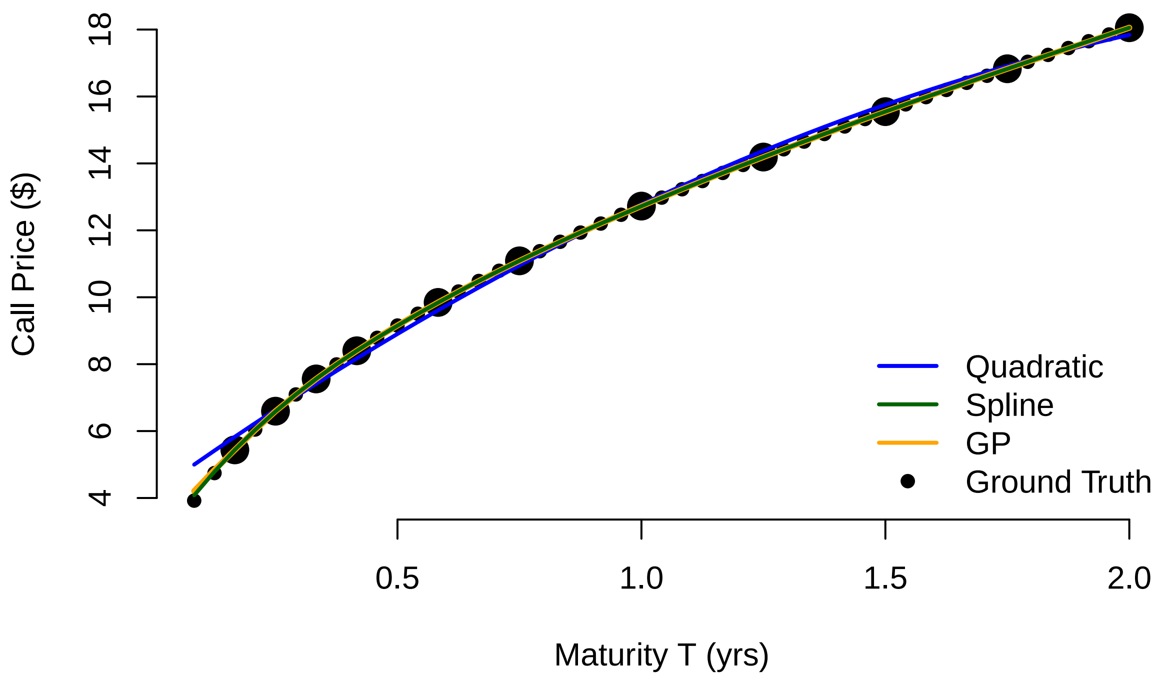
The next plot matches the left panel of Figure 4.1 in the article. It compares the three statistical surrogates on a test set of maturities ranging from 1 to 24 months.

```r
seqT <- seq(1/12.0, 2.0, by=1/24.0)  # test set of maturities

plot(seqT, df_Calls[,25],
      bty='n', xlab="Maturity T (yrs)", ylab="Call Price ($)", pch=16)
```

```r
points(train.design[,"T"], train.design[,"P"], col="black", pch=16, cex=2)
betas <- coef(quad.model)
lines(seqT, betas[1] + betas[2]*seqT + betas[3]*seqT^2, col="blue", lwd=2 )

# Predict from the fitted GPR model
PriceTest <-  predict(gpModel_Heston1D, newdata=data.frame(T=seqT), type="UK")
lines(seqT, PriceTest$mean, col="orange", lwd=3)
lines(seqT, predict(cubic.spl,newdata = list(T=seqT)), col="darkgreen", lwd=2)
```

```
## Warning in bs(T, degree = 3L, knots = c(0.5, 1), Boundary.knots =
## c(0.166666666666667, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```r
legend("bottomright", c("Quadratic", "Spline", "GP", "Ground Truth"),  lwd=c(2,2,2,-1),
       pch=c(-1, -1, -1, 16), col=c( "blue", "darkgreen","orange", "black"), bty='n')
```

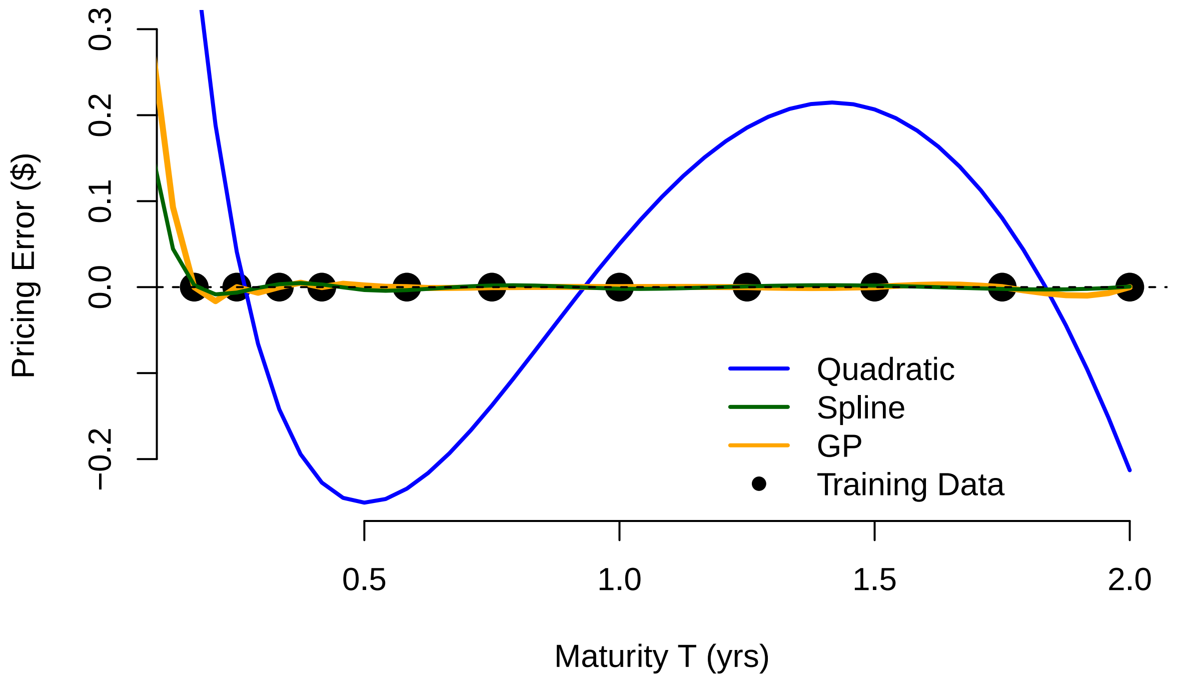


In the second plot (right panel of Figure 4.1) we zoom-in to show the differences between the surrogate-predicted option price and the ground truth on the test set:

```r
plot(train.design[,"T"], rep(0, length(train.set)), col="black", pch=16, cex=2,
     ylim=c(-0.25, 0.3), bty='n', xlab="Maturity T (yrs)", ylab="Pricing Error ($)")
betas <- coef(quad.model)
lines(seqT, betas[1] + betas[2]*seqT + betas[3]*seqT^2 - df_Calls[,25], col="blue", lwd=2 )
lines(seqT, PriceTest$mean - df_Calls[,25], col="orange",lwd=3)
lines(seqT, predict(cubic.spl,newdata = list(T=seqT))-df_Calls[,25],col="darkgreen",lwd=2)
```

```
## Warning in bs(T, degree = 3L, knots = c(0.5, 1), Boundary.knots =
## c(0.166666666666667, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```r
legend(x=1.2, y=-0.05, c("Quadratic", "Spline", "GP", "Training Data"), lwd=c(2,2,2,-1),
       pch=c(-1, -1, -1, 16), col=c( "blue", "darkgreen","orange", "black"), bty='n')
abline(h=0,lty=2)
```



Learning the Black-Scholes Prices and Greeks

In the second case study, we work with the Black-Scholes model that postulates Geometric Brownian Motion dynamics for the univariate underlying asset price (S_t):

$$dS_t = (r - q)S_t dt + \sigma S_t dW_t,$$

where (W_t) is a Wiener process with respect the pricing measure Q, and the interest rate r, dividend yield q, and volatility σ are given constants. We again consider the Call payoff $G(S_T) = \max(S_T - \mathcal{K}, 0)$ whose value admits an explicit representation given by the Black-Scholes formula. Taking partial derivatives of the Black-Scholes formula, one can obtain the related sensitivities or Greeks, in particular Delta $(\partial_P/\partial S)$, Theta $(-\partial_P/\partial t)$ and Gamma $(\partial^2 P/\partial S^2)$.

The helper function implements the Black-Scholes formula and related Greeks:

```
BScall <- function(t=0,T,S,K,r,q=0,sigma,isPut=0) {
# t and T are measured in years; all parameters are annualized
# r is the cont interest rate; q is the continuous dividend yield
# isPut=0: Call payoff; isPut=1: Put payoff
d1 <- (log(S/K)+(r-q+sigma^2/2)*(T-t))/(sigma*sqrt(T-t))
d2 <- d1-sigma*sqrt(T-t)

# Call Greeks at t
Delta <- exp(-q*(T-t))*pnorm(d1)
Gamma <- exp(-q*(T-t))*exp(-d1^2/2)/sqrt(2*pi)/S/sigma/sqrt(T-t)
Vega <- S*exp(-q*(T-t))/sqrt(2*pi)*exp(-d1^2/2)*sqrt(T-t)
Theta <- -S*exp(-q*(T-t))*sigma/sqrt(T-t)/2*dnorm(d1) - r*K*exp(-r*(T-t))*pnorm(d2) +
    q*S*exp(-q*(T-t))*pnorm(d1)
Rho <- (T-t)*K*exp(-r*(T-t))*pnorm(d2)

# Black-Scholes formula for Calls G(S_T) = (S_T-{\cal K})_+
BSprice <- S*Delta-K*exp(-r*(T-t))*pnorm(d2)
# Respective equations if the payoff is a Put, G(S_T) = ({\cal K}-S_T)_+
if (isPut == 1) {
    Delta <- -exp(-q*(T-t))*pnorm(-d1)
    BSprice <- S*Delta+K*exp(-r*(T-t))*pnorm(-d2)
    Theta <- -S*exp(-q*(T-t))*sigma/sqrt(T-t)/2*dnorm(d1) +
      r*K*exp(-r*(T-t))*pnorm(-d2) - q*S*exp(-q*(T-t))*pnorm(-d1)
    Rho <- -(T-t)*K*exp(-r*(T-t))*pnorm(-d2)
}

return (list(Delta=Delta,Gamma=Gamma,Theta=Theta,Vega=Vega,Rho=Rho,Price=BSprice))
}
```

We will train a 2D GP surrogate that learns to price Calls as a function of initial stock price S_0 and Call maturity T. For visualization purposes, we will test on a univariate test set with a fixed $T = 0.5$.

```
set.seed(2024)
xTest <- seq(60,130,by=1);  # test set to be plotted: moneyness in [0.6, 1.3]
tTest = 0.5;                # fixed test maturity
r <- 0.05; q <- 0.01        # interest rate and div yield
```

To generate training data, we rely on a vanilla Monte Carlo simulator that approximates option prices as empirical average of discounted payoffs. The simulator returns both the training output $y(x)$, as well as the corresponding noise variance $\sigma^2(x)$ (via the empirical MC variance).

```
# generate data from a noisy Black-Scholes option pricer that
# utilizes vanilla Monte Carlo estimation based on M paths
BS_mc <- function(M, r=0.05, q=0, sigma=0.2, T=1, K = 100, S0=100)
{
  Z <- rnorm(M)

  S_T <- S0*exp( (r-q-sigma^2/2)*T + sigma*sqrt(T)*Z)  # terminal values
  Payoff <- exp(-r*T)*pmax(S_T -K, 0)   # discounted payoffs
  return( list(mean=  mean(Payoff), sd= sd(Payoff)/sqrt(M) ) )

}
```

We proceed to learn the Prices and the Greeks of a **Call option** with fixed strike $\mathcal{K} = 100$. To do so, we employ a two-dimensional training set of 450 total training locations, with 400 actual inputs plus another 50 "virtual" inputs to capture the boundary conditions. Our task is to learn the Delta/Theta/Gamma of a Call as a function of current stock price S_t (henceforth the spot) and time t. The inputs themselves are in the rectangle $[60, 130] \times [0.05, 0.6]$.

The 400 training input-output tuples are constructed by sampling 400 locations via the space-filling Halton sequence (available in `randtoolbox` package) and then running Monte Carlo approximation of the respective option price through a plain Monte Carlo draw of $M' = 2500$ i.i.d. samples based on the log-normal law of S_T (this simulation engine is viewed as a black-box for the modeler). We then record the mean $\bar{y}^i$ and the standard deviation of these M' samples.

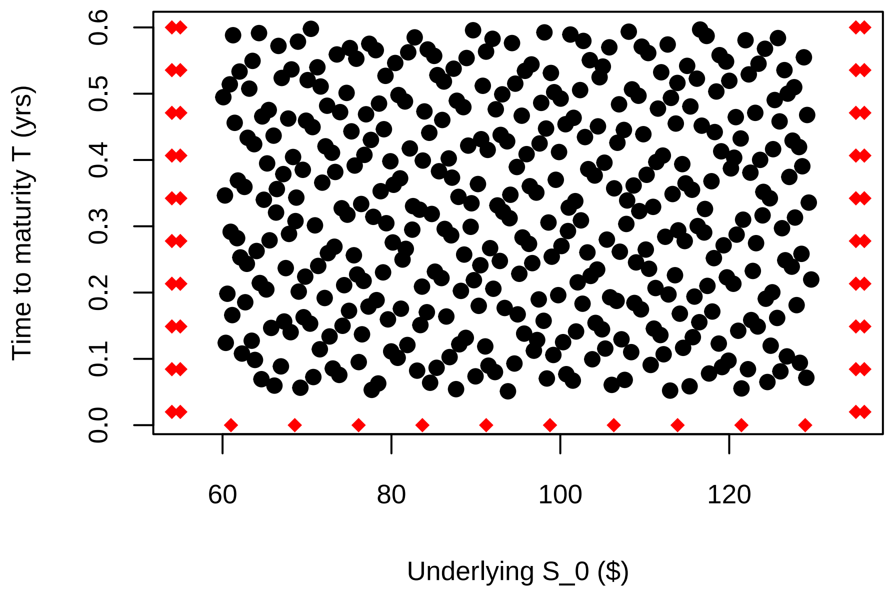
```
N_tr <- 450
hltn <- halton(400, dim=2)  # space-filling QMC sequence on [0,1] x [0,1]

# create simulation design rescaled to (t,x) \in [0.05, 0.6] x [60, 130]
simDsgn <- cbind(t=c(0.05+0.55*hltn[,1],rep(0,50)),
                spot=c(60+70*hltn[,2],rep(50,50)),
                price=rep(0,N_tr), noise=rep(0,N_tr) )
```

```r
for (i in 1:(N_tr-50)) {
  mcPrice <- BS_mc(2500,K=100,r=0.05, q=0.01, sigma=0.2,
                   T=simDsgn[i,"t"], S0=simDsgn[i,"spot"])
  simDsgn[i,"price"] <- mcPrice$mean
  simDsgn[i,"noise"] <- mcPrice$sd
}
```

The next snippet creates 50 additional "virtual" training points at the edges of the above training set, namely 20 deep in-the-money ($S \in \{135, 136\}$), 20 deep out-of-the-money $S \in \{54, 55\}$) and 10 training points at maturity to capture the final payoff shape.

```r
tSeq <- seq(0.02,0.6,len=10)
simDsgn[(N_tr-39):(N_tr),"t"] <- rep(tSeq,4)
simDsgn[(N_tr-49):(N_tr),"noise"] <- 0

# Calls have intrinsic value e^{-q T}S_0 - e^{-r T}K deep in-the-money
simDsgn[(N_tr-9):(N_tr),"price"] <- exp(-q*tSeq)*135-exp(-r*tSeq)*100
simDsgn[(N_tr-9):(N_tr),"spot"] <- 135

simDsgn[(N_tr-19):(N_tr-10),"price"] <- exp(-q*tSeq)*136-exp(-r*tSeq)*100
simDsgn[(N_tr-19):(N_tr-10),"spot"] <- 136

# Calls are worth zero deep out-of-the-money
simDsgn[(N_tr-29):(N_tr-20),"price"] <- 0
simDsgn[(N_tr-29):(N_tr-20),"spot"] <- 55

simDsgn[(N_tr-39):(N_tr-30),"price"] <- 0
simDsgn[(N_tr-39):(N_tr-30),"spot"] <- 54

# When T=0, the Call value is the payoff
simDsgn[(N_tr-49):(N_tr-40),"t"] <- 0
simDsgn[(N_tr-49):(N_tr-40),"spot"] <- seq(61,129,len=10)
simDsgn[(N_tr-49):(N_tr-40),"price"] <- pmax(0, simDsgn[(N_tr-49):(N_tr-40),"spot"]-100)
```

The Figure below visualizes the overall training set, showing the 400 MC-based inputs (in black) and the 50 virtual training points around the edges (in red).



Training the GP surrogate

With the training set of approximate option prices constructed, we are ready to train a GP surrogate. We generate two different models, one with a anisotropic SE kernel k_{SE} and a second with a Matern kernel k_{M52}. For both cases we use a prior mean function of the form $\mu(S) = \beta_0 + \beta_1 S$.

```
gpModel_M52 <- km(   formula = y ~ 1 + spot,  # linear trend function
                 design =simDsgn[,1:2],  response=simDsgn[,"price"],
                 nugget.estim=TRUE,  # learn
                 # alternatively: use the estimated simulation noise
                 #noise.var=pmax(1e-7, simDsgn[,"noise"]),
                 covtype="matern5_2",   # can also try "gauss" or "matern3_2"
                 optim.method="gen",
                 estim.method = "MLE",
                 lower=c(0.1,10), upper=c(2,100),  # bounds on lengthscales
                 # the "control" parameters below handle speed versus risk of
                 # converging to local minima.  See "rgenoud" package for details
                 control=list(max.generations=100,pop.size=100,
                              wait.generations=8,
                              solution.tolerance=1e-5,
                              maxit = 1000, trace=F
                 ))
print(coef(gpModel_M52))
```

```
## $trend1
## [1] -28.09959
##
## $trend2
## [1] 0.4584299
##
## $range
## [1]  1.074271 26.519040
##
## $shape
## numeric(0)
```

```
##
## $sd2
## [1] 29.72628
##
## $nugget
## [1] 0.02412824
```

```
gpModel_SE <- km(   formula = y ~ 1 + spot,  # linear trend function
                 design =simDsgn[,1:2],  response=simDsgn[,"price"],
                 nugget.estim=TRUE,  # learn
                 # alternatively: use the estimated simulation noise
                 #noise.var=pmax(1e-7, simDsgn[,"noise"]),
                 covtype="gauss",
                 optim.method="gen",
                 estim.method = "MLE",
                 lower=c(0.1,10), upper=c(2,100),  # bounds on lengthscales
                 # the "control" parameters below handle speed versus risk of
                 # converging to local minima.  See "rgenoud" package for details
                 control=list(max.generations=100,pop.size=100,
                              wait.generations=8,
                              solution.tolerance=1e-5,
                              maxit = 1000, trace=F
                 ))
print(coef(gpModel_SE))
```

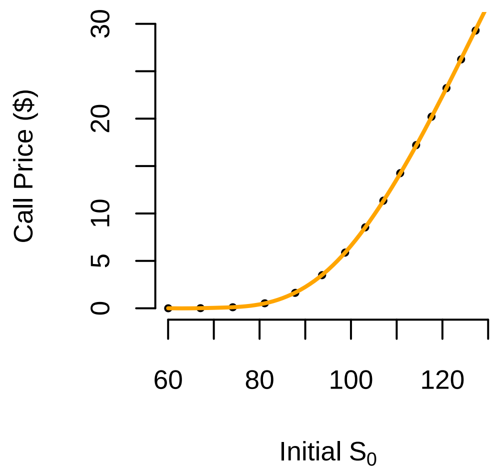
```
## $trend1
## [1] -29.99754
##
## $trend2
## [1] 0.4566743
##
## $range
## [1]  0.9173029 14.0468568
##
## $shape
## numeric(0)
##
## $sd2
## [1] 36.12284
##
## $nugget
## [1] 0.02494027
```

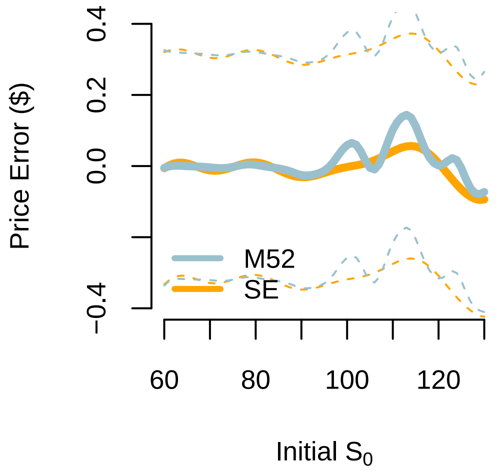
To compare the results we build a one-dimensional test set that uses a fixed time-to-maturity $\tau = 0.5$ and a range of initial stock prices S_0.

```
testSet <- data.frame(t=rep(tTest,length(xTest)),spot=xTest)

# Predict option prices using the GP surrogates
PriceTest_SE <-   predict(gpModel_SE, newdata=testSet,type="UK")
PriceTest_M52 <-  predict(gpModel_M52,newdata=testSet,type="UK")
```

As a check we compare the true price and the GP estimates. We see that the SE kernel gives a bit smoother and more stable (in terms of L_1-norm) error relative to the ground truth. Both kernels yield very similar credible bands on the option price of about ± 0.2.

```
par(mar = c(4,4,1,1),oma = c(1, 1, 1, 1),mfrow=c(1,2))
truePrice <- BScall(t=0, T=0.5, S=xTest, K=100, r=0.05, q=0.01, sigma=0.2, isPut=0)$Price
plot(xTest,truePrice, col="black", type="l", lwd=4, lty=3, cex.lab=0.8, cex.axis=0.8,
     xlab=expression(paste('Initial ', S[0])), ylab='Call Price ($)', bty='n', ylim=c(0,30))
lines(xTest,PriceTest_SE$m, lwd=2, col="orange")

#To better see the difference: show difference to ground truth

plot(xTest,truePrice-PriceTest_SE$m, col="orange", type="l", lwd=4, cex.lab=0.8, cex.axis=0.8,
     xlab=expression(paste('Initial ', S[0])), ylab='Price Error ($)', bty='n', ylim=c(-0.4, 0.4))
lines(xTest,truePrice-PriceTest_M52$m,col="lightblue3",lwd=4)
# also draw the 95% credible bands
lines(xTest,truePrice-PriceTest_SE$m+1.96*PriceTest_SE$sd, col="orange",lwd=1,lty=2)
lines(xTest,truePrice-PriceTest_SE$m-1.96*PriceTest_SE$sd, col="orange",lwd=1,lty=2)
lines(xTest,truePrice-PriceTest_M52$m+1.96*PriceTest_M52$sd, col="lightblue3",lwd=1,lty=2)
lines(xTest,truePrice-PriceTest_M52$m-1.96*PriceTest_M52$sd, col="lightblue3",lwd=1,lty=2)

legend("bottomleft",c("M52", "SE"),col=c("lightblue3","orange"),lwd=3, bty='n', cex=0.8)
```





GP inference for the Greeks: Section 4.3

We next use the GP model to generate the posterior mean/variance of the gradients of `gpModel`. This involves differentiating the respective kernels to obtain the GP for the respective gradients. For Delta and Theta, we utilize the helper `gpDerivative` function that implements formulas (5.9)-(5.12) for the gradients of the Squared-Exponential and Matern-5/2 kernels.

```
gpDerivative <- function(fit,xstar,i=2,K_Inv=0){
  # fit is a km object
  #   - requires Gaussian kernel or Matern 5-2
  # xstar is the test set
  # i indicates which coordinate the derivative should be taken in
  #   by default i=2
  # K_Inv is the inverse of the covariance matrix.  It can be given
  #   to reduce computational cost, if it has already been
  #   inverted.  Otherwise, it pulls Delta from the km object
  #   and inverts it.
  # Returns a posterior mean and posterior covariance for the
  #  distribution evaluated at xstar.
```

```r
  xstar <- as.matrix(xstar)
  y <- fit@y
  x <- fit@X
  eta2 <- fit@covariance@sd2
  theta <- fit@covariance@range.val[i]
  c.1 <- covMatrix(fit@covariance,xstar)$C - covMatrix(fit@covariance,xstar)$vn
  c.2 <- covMat1Mat2(fit@covariance,xstar,x, nugget.flag=FALSE)
  if(K_Inv == 0){
    # T is the Choleski decomposition of the covariance matrix
    T <- fit@T
    Delta <- t(T) %*% T
    K_Inv <- solve(Delta)
  }

  # derivative of the squared-exponential kernel
  if(fit@covariance@name == "gauss"){
    dcxx <- -1 / theta ^ 2 * outer(xstar[, i], xstar[, i], '-') * c.1
    d2c <- 1 / theta ^ 2 * (-c.1 + outer(xstar[, i], xstar[, i], '-') * dcxx)
    dcxX <- -1 / theta ^ 2 * outer(xstar[, i], x[, i], '-') * c.2
    dcXx <- -1 / theta ^ 2 * outer(x[, i], xstar[, i], '-') * t(c.2)

    covDiag <- sqrt(pmax(0, -diag(d2c - dcxX %*% K_Inv %*% dcXx)))
  }

  # derivative of the Matern-4/2 kernel
  if(fit@covariance@name == "matern5_2"){

    absDist <- abs(outer(xstar[,i],x[,i],'-') )
    leadTerm1 <- 1 + sqrt(5)/theta*absDist + 5/3/theta^2*absDist*absDist
    numTerm1 <- -5/3/theta^2*outer(xstar[,i],x[,i],'-')-sqrt(5)*5/3/theta^3*absDist*outer(xstar[,i],x[,
    dcxX <- numTerm1/leadTerm1* c.2

    dcXx <- t(dcxX)

    covDiag = sqrt(diag( 5/3/theta^2*c.1 - dcxX %*% K_Inv %*% dcXx))
  }

  # de-trend the mean. Hard-coded about regressing on 1+spot
  detr_y <- fit@y - fit@F %*% fit@trend.coef
  m <- dcxX %*% K_Inv %*% detr_y

  if (i == 2 & length(fit@trend.coef) == 2)
    m <- m + fit@trend.coef[2]

  return(list(m=m,covmat=covDiag))  # only compute the standard errors at x_star, no covariances

}
```

We use the above function to compute the Call Delta and Theta for the test points based on the two above GP models.

```r
# Delta: gradient with respect to x_2
DeltaTest_SE <- gpDerivative(fit=gpModel_SE, testSet,i=2)
DeltaTest_M52 <- gpDerivative(fit=gpModel_M52, testSet,i=2)
```

```r
# Theta -- gradient with respect to x_1
ThetaTest_SE <- gpDerivative(fit=gpModel_SE,testSet,i=1)
ThetaTest_M52 <- gpDerivative(fit=gpModel_M52,testSet,i=1)
```

To compute second order sensitivity, specifically option Gamma, we employ finite differences (to demonstrate that this is also a quick-to-implement alternative):

$$\frac{\partial^2 P}{\partial S^2}(t,S) \simeq \frac{P(t,S+h)-2P(t,S)+P(t,S-h)}{h^2}.$$

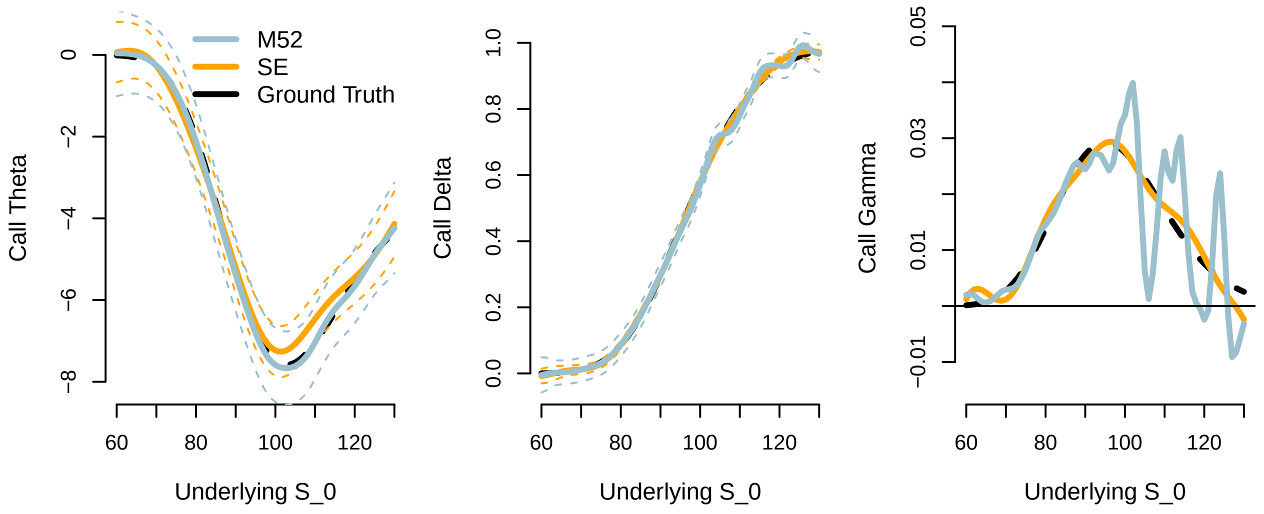
Below we set discretization parameter $h = 0.01$.

```r
h <- 0.01  # h for finite-differencing approximation
GammaTest_M52 <- GammaTest_SE <- xTest  # evaluate one predictive site at a time
for (jj in 1:length(xTest)) {
  GammaTriple <- data.frame(spot=c(xTest[jj]-h,xTest[jj],xTest[jj]+h),
                                                t=c(tTest,tTest,tTest))
  triple <-  predict(gpModel_M52,newdata=GammaTriple,type="UK")$mean
  GammaTest_M52[jj] <- (triple[3]-2*triple[2]+triple[1])/h^2  # Gamma approximation
  triple <-  predict(gpModel_SE,newdata=GammaTriple,type="UK")$mean
  GammaTest_SE[jj] <- (triple[3]-2*triple[2]+triple[1])/h^2
}
```

We now plot the Greeks and their posterior uncertainty (credible bands, shown at 95% level). This is equivalent to the plots in **Figure 4.2** of the book. Observe that the estimate of Gamma is not stable using the M52 kernel (which is only twice-differentiable, while SE is infinitely-differentiable).

```r
par(mar = c(4,4,1,1),oma = c(1, 1, 1, 1), mfrow=c(1,3))
# Theta panel
plot(xTest,BScall(T=0.5, S=xTest, K=100, r=0.05, q=0.01, sigma=0.2, isPut=0)$Theta,
     col="black",type="l", lwd=3, lty=2,  cex.lab=1.1, # ground truth
     xlab='Underlying S_0', ylab='Call Theta', bty='n', ylim=c(-8.2,0.7))
lines(xTest,-ThetaTest_SE$m, lwd=3, col="orange")
lines(xTest,-ThetaTest_SE$m+1.96*ThetaTest_SE$covmat, col="orange",lwd=1,lty=2)
lines(xTest,-ThetaTest_SE$m-1.96*ThetaTest_SE$covmat, col="orange",lwd=1,lty=2)

lines(xTest,-ThetaTest_M52$m, lwd=3, col="lightblue3")
lines(xTest,-ThetaTest_M52$m+1.96*ThetaTest_M52$covmat, col="lightblue3",lwd=1,lty=2)
lines(xTest,-ThetaTest_M52$m-1.96*ThetaTest_M52$covmat, col="lightblue3",lwd=1,lty=2)
legend("topright",c("M52", "SE", "Ground Truth"),
       col=c("lightblue3","orange", "black"), lwd=3, bty='n',cex=1.1)

# Delta panel
plot(xTest,BScall(T=0.5, S=xTest, K=100, r=0.05, q=0.01, sigma=0.2, isPut=0)$Delta,
     col="black",type="l", lwd=3, lty=2, cex.lab=1.1,
     xlab='Underlying S_0', ylab='Call Delta', bty='n', ylim=c(-0.05,1.05))

lines(xTest,DeltaTest_SE$m, lwd=3, col="orange")
lines(xTest,DeltaTest_SE$m+1.96*DeltaTest_SE$covmat, col="orange",lwd=1,lty=2)
lines(xTest,DeltaTest_SE$m-1.96*DeltaTest_SE$covmat, col="orange",lwd=1,lty=2)

lines(xTest,DeltaTest_M52$m, lwd=3, col="lightblue3")
lines(xTest,DeltaTest_M52$m+1.96*DeltaTest_M52$covmat, col="lightblue3",lwd=1,lty=2)
lines(xTest,DeltaTest_M52$m-1.96*DeltaTest_M52$covmat, col="lightblue3",lwd=1,lty=2)

# Gamma panel
```

```r
plot(xTest,BScall(T=0.5, S=xTest, K=100, r=0.05, q=0.01, sigma=0.2, isPut=0)$Gamma,
     col="black",lwd=3, lty=2, type="l", bty='n',ylim=c(-0.015, 0.05),
     xlab='Underlying S_0', ylab='Call Gamma', cex.lab=1.1
)
lines(xTest,GammaTest_SE,type="l", lwd=3, col="orange")
lines(xTest,GammaTest_M52,lwd=3,col="lightblue3")
abline(h=0)
```



License