Project Document

He Junjie, Huang Liyun, Liang Zhihui, Luo Jiarong, Xiao Chao

1. Black formula

1.1 getBlackCall

1.1.1 Implementation

The purpose of this function is to calculate forward price of a call option whose basic formula is:

$$C(K,T) = G_0(T)N(d_1) - KN(d_2)$$

$$d_1 = \frac{lnG_0(T) - \ln(K)}{\sigma\sqrt{T}} + 0.5 * \sigma\sqrt{T}$$

$$d2 = d1 - \sigma\sqrt{T}$$

And this function accepts vector input.

Steps of implementation are as follows:

1. Input variable check.
2. Calculate and save intermediate variable $\sigma\sqrt{T}$, d_1, d_2
3. Calculate option price based on different input T and vector Ks variable:
   a. T time to expiry is zero, the forward price of the call is MAX(forward spot – strike price,0)
   b. K strike price is zero, the price is obtained as

   $$\lim_{K\to 0^+} C(K,T) = G_0(T)$$

   c. T≠0 and K≠0, implement the normal Black formula

1.1.2 Test for input variable

Implement test on input variable check:

1. Forward Spot, Time Expiry, Strike Price should be non-negative.
2. Implied volatilities should be positive.
3. Dimension of strike price and implied volatilities should be the same.

Any violation of the above condition will display corresponding error message and the function will stop.

1.1.3 Test for Call-Put Parity

This function test that European call options forward prices obtained by getBlackCall function satisfy call-put parity.

Steps for test are as follows:

1. Obtained market data given in project and use getStrikeFromDelta function to obtain corresponding strike price from implied volatilities and deltas.
2. Use black formula to calculate forward price of call option by getBlackCall function.
3. Calculate forward price of put option with the same forward spot, strike price and expiration strike.
4. Test whether C + K = P + FWD where FWD stands for forward spot.
5. As there may exist some rounding error, if abs(C+K-P-FWD) <e^{-15}, the call-put parity test

pass, otherwise fail.

2. Interest rate interpolation

2.1 makeDepoCurve

2.1.1 Implementation

With the assumption that interest rates are constant between t_i and t_{i+1}, this function calculate the interest rate to make Depocurve based on the formula bellows.

$$r = -\ln(df) \div ts$$

We also calculate rate integral and forward interest rate as follows, which reduces calculations.

$$integ_i = r_i \times ts_i$$

$$fwdir_i = (integ_{i+1} - integ_i)/(ts_{i+1} - ts_i)$$

2.1.2 Description of Depocurve

The Depocurve represents the term structure of interest rates for a country.

Curve.ir = $[r_0, r_1, r_2, \dots r_n]$, It is the interest rate.

Curve.ts = $[t_0, t_1, t_2, \dots t_n]$ It is the times to settlement in years.

Curve.integ = $[\text{integ}_0, \text{integ}_1, \text{integ}_2, \dots \text{integ}_n]$, It reduces calculation in getRateIntegral.

Curve.fwdir = $[\text{fwdir}_0, \text{fwdir}_1, \text{fwdir}_2, \dots \text{fwdir}_{n-1}]$, It save the forward interest rate between t_i and t_{i+1}.

2.2 getRateIntegral

2.2.1 Implementation

For $t < t_0$, we assume that the interest rate between 0 and t_0 is constant r_0. Then the rate integral should be $r_0 \times t$.

For $t_0 \le t < t_n$, let's assume $t_i \le t < t_{i+1}$. The first part is rate integral between 0 and t_i, which is $integ_i$. The second part is $fwdir_i \times (t - t_i)$.

For $t_n \le t \le t_n + 30/365$, we assume that the interest rate after t_n is constant r_n. Then the rate integral should be $r_n \times t$.

2.2.2 Test

1. testInputCheck

This part checks whether the inputCheck function works successfully. Our test includes the following cases:

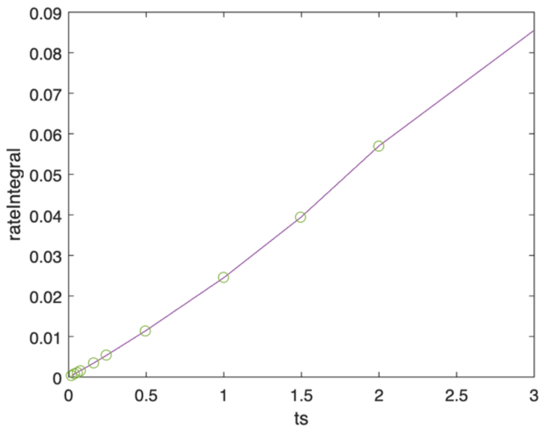
a. Curve not complete, dimension not match or empty
b. Negative or zero value in curve.ts
c. T is not a scalar, or negative T
d. $T > t_n + 30/365$

2. testOriginalPoint

For this test, we test whether the original discount factors are equal to the result of getRateIntegral function.

3. testIntegralLinear

In this test, we get 200 points between each t_i and t_{i+1} on equal interval, and test if their rateIntegral are linear and continuous.



The dot in this graph are the points in curve.integ. We can see the rate integral between t_i and t_{i+1} are linear and continuous. But for point t_i, it's not continuous since the interest rate before and after t_i are different.

3. Forward spot $G_0(T)(= S_T)$

3.1 makeFwdCurve

3.1.1 Implementation

Firstly, we calculated the cash rate.

Since

$$S_0 = F_t(0+\tau) = X_0 e^{\int_0^\tau [r(u)-y(u)]du}$$

Therefore

$$X_0 = S_0 e^{-\int_0^\tau [r(u)-y(u)]du}$$

Then we save the domCurve, forCurve, X_0 and τ in the FwdCurve for the calculation of fwd in getFwdSpot function.

3.1.2 Description of Depocurve

1. FwdCurve.domCurve is domestic IR curve.
2. FwdCurve.forCurve is foreign IR curve.
3. FwdCurve.X0 is cash rate.
4. FwdCurve.tau is lag between spot and settlement.

3.2 getFwdSpot

3.2.1 Implementation

We apply the formula below to get the forward spot price at any time with cash rate.

$$fwd_i = G(T) = F_t(T+\tau) = X_0 e^{\int_0^{T+\tau} [r(u)-y(u)]du}$$

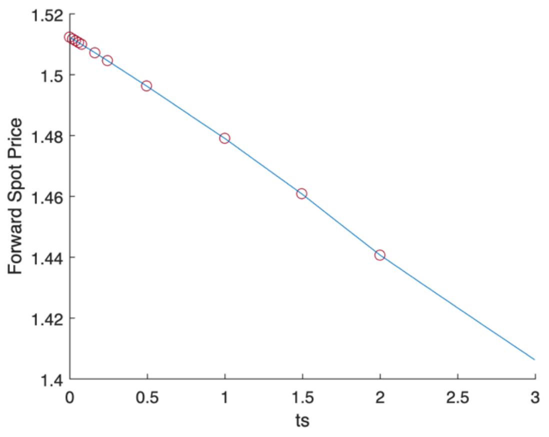
3.2.2 Test

1. testInputCheck

The test cases of this part are similar to the test case for getRateIntegral. In addition, we also need to check the X0 and tau in FwdCurve.

2. testIntegralLinear

In this test, we get 200 points between each t_i and t_{i+1} on equal interval.



The dot in this graph are the points in fwdCurve.fwd. We can see the forward spot price between t_i and t_{i+1} are linear and continuous. But for point t_i, it's not continuous since the forward interest rate before and after t_i are different.

4. Conversion of deltas to strikes

4.1 getStrikeFromDelta

4.1.1 Implement

The purpose of this function is to calculate option strike price given implied volatility and delta using inverse Black formula. And it uses zeroin algorithm, which combines bisection, secant and inverse quadratic interpolation methods, as root search method to improve robustness and speed.

Steps of implementation are as follows:

1. Input variable check.
2. Generate the target equation on variable K:

$$d_1 = \frac{\ln(F_T) - \ln(K)}{\sigma\sqrt{T}} + 0.5 * \sigma\sqrt{T}$$

$$|\triangle_{call}| = N(d_1)$$

$$|\triangle_{put}| = N(d_2)$$

3. Use zeroin algorithm to search root and set [fwd*0.01, fwd*100] as initial search range for K. The outline of the algorithm is as follows:
    a. Set initial value as a and b and ensure f(a) and f(b) have opposite signs.
    b. Use a secant step to calculate c between a and b
    c. Repeat following steps until |b-a| < ε|b| or f(b)=0
    d. Arrange a, b and c so that
        - f(a) and f(b) have opposite signs
        - |f(b)| ≤ |f(a)|
        - c is the precious value of b
    e. If c≠a, consider IQI step

f. If c=a, consider secant step

g. If the IQI or secant step is in the interval [a,b], take it, else use bisection

4.1.2 Test

Implement test on input variable check:

a. Forward spot, Time Expiry, Implied Volatilities, Absolute delta value should be positive.

b. Flag should be 1 or -1 for call and put option separately.

Any violation of the above condition will display corresponding error message and the function will stop.

5. Interpolation of implied volatility in strike direction

5.1 makeSmile

5.1.1 Implement

Target:

Construct a function that interpolates the smile in strike for a given tenor T, and then design an extrapolation scheme as following formula:

$$\sigma(K) = \begin{cases} spline(K_1) + a_L \tanh(b_L(K_1 - K)) & if\ K < K_1 \\ spline(K) & if\ K_1 < K < K_N \\ spline(K_N) + a_R \tanh(b_R(K - K_N)) & if\ K > K_N \end{cases}$$

Steps of implementation:

1. Check for invalid input.
2. Calculate K and C for each option (Input fwd, T, cps, vols and deltas).
3. Check for no arbitrage constraints.
4. Calculate natural cubic spline coefficients with function csape.
5. Calculate extrapolation coefficients with mathematic definition.

5.1.2 Test

1. check invalid inputs:
   a. Input vectors which dimension do not match.
   b. Input negative vols.
   c. Input vectors whose length smaller than 4.
2. check arbitrage constraints:
   a. Arbitrage constraint error.

5.2 getSmileVol

5.2.1 Implement

Target:

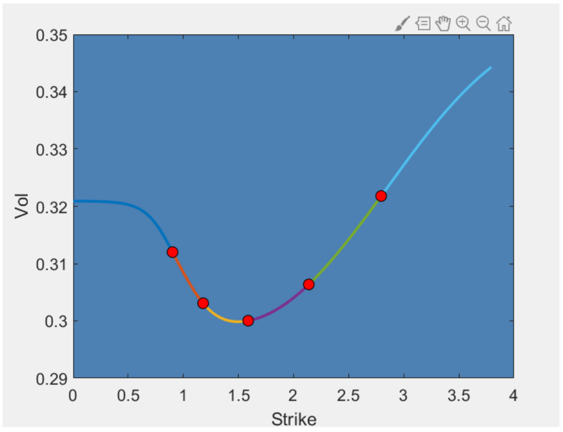
Utilize the smile curve we get from function makeSmile, and then performs the actual interpolation and extrapolation.

Steps of implementation:

1. Check for invalid input.
2. Calculate interpolation value (vols) with function ppval.
3. Calculate extrapolation value (vols) with mathematic definition.

5.2.2 Output

We utilize options data (T=2) to construct Smile curve, and then split each interval into 50 points and calculate their predicted implied volatilities. From following chart, we find that their output is smooth and behave as expected. At K1 and KN, 1st derivatives of the spline functions are continuous and 2nd derivatives of the spline functions are zero (natural boundaries).



5.2.3 Test

1. check invalid inputs
   a. Input K < 0
2. check whether the plot of a smile curve look smooth:
   a. Utilize function testPlotVolcurveK2D.
3. check 1st deriv(continuous) and 2nd deriv(natural) at K1 and KN:
   a. 1st derivatives are continuous at K1 and KN (smooth).
   b. 2nd derivatives are zero at K1 and KN (natural boundaries).

6. Construction of implied volatility surface

6.1 makeVolSurface

6.1.1 Implement

The purpose of this function is to create a volSurface structure.

The fields of this structure include:

spots: a vector of $G_0(T_i)$ calculated with fwdCurve and Ts input;

Ts: input Ts in ascending order;

fwdCurve: a replication of input

slopes: storing of precomputed $\frac{1}{T_{i+1}-T_i}$

smiles: storing of precomputed smile structure $\sigma(T_i)$ in terms of input Ts

Steps of implementation:

1. Invalid input testing
2. Sort the Ts and vols in terms of the length of time

3. Calculate spot price in terms of the Ts vector calculated with fwdCurve and Ts input $G_0(T_i)$
4. Precompute a series of smile structs in terms of input Ts and store them in the struct
5. Perform no-arbitrage checking (see below)

6.1.2 Test

1. Invalid input testing:
   a. Negative Ts
   b. Invalid cps (values given other than -1/1)
   c. Invalid deltas (values outside [-1,1])
   d. Invalid vols (negative volatility)
   e. Inconsistent dimensions of inputs (row nums of vols must equal to the length of Ts, col nums of vols must equal to cps and deltas)
2. Violation of calendar no-arbitrage constraint testing: modify a figure in vols to sufficiently low level (but still positive), so that this volatility surface violates the calendar no-arbitrage constraint

6.2 getVol

6.2.1 Implement

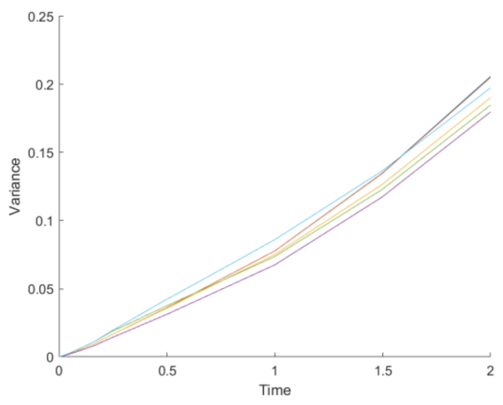
The purpose of this function is to calculate the interpolated volatility given volSurface struct, T and a vector of Ks.

Steps of implementation:

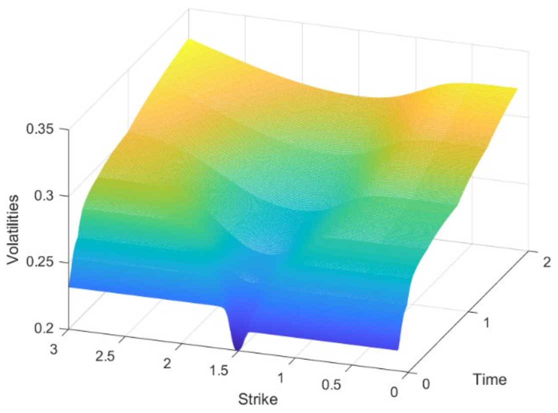
1. Invalid input testing
2. Calculate $\mathrm{K}/G_0(T)$
3. Check the interval the input T lies in with respect to volSurface.Ts, and calculate with corresponding formula:
   a. If T lies in $[0, \mathrm{T}_1]$, this is the first case to calculate
   b. If T lies in $(\mathrm{T}_\mathrm{N}, +\infty)$, the script will throw an error
   c. If T lies in $(\mathrm{T}_1, \mathrm{T}_\mathrm{N}]$, perform the interpolation sheme listed in 6.1.1

6.2.2 Test

1. Invalid input testing
   a. Incomplete fields in volSurface struct
   b. Feed T as a vector
   c. Invalid T (larger than Tn or negative)
   d. Invalid Ks (negative)
2. Positive test: should not throw error if fed scalar as well as vector Ks
3. Variance linearity test: plot the variance line with respect to time to expiry in different strikes and check whether the lines are continuous and segment-wise linear (testPlotVolcurveT2D.m);



4. Continuity test: plot the volatility surface and see if there are jumps in the surface (testPlotVolcurveKT3D.m):



7. Compute the probability density function (PDF) of S_T

7.1 getPdf

7.1.1 Implementation

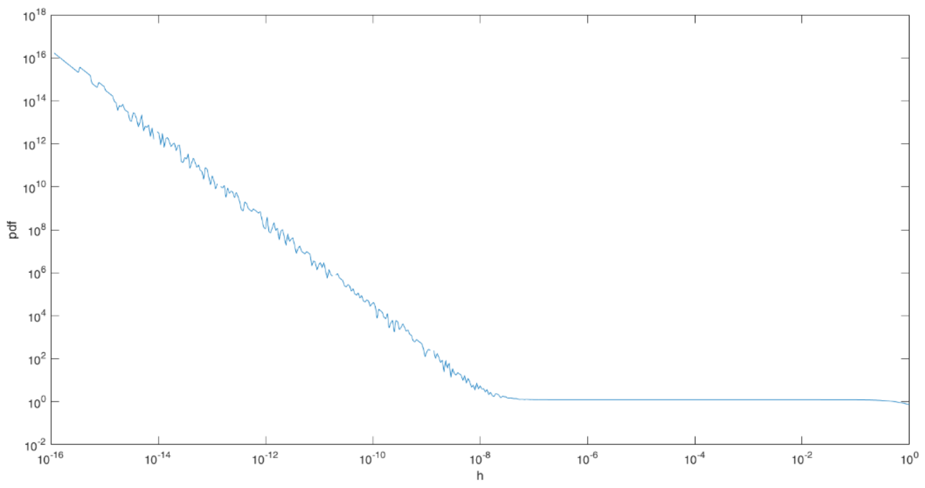
The approximation of the second derivative used in *getPdf* is

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

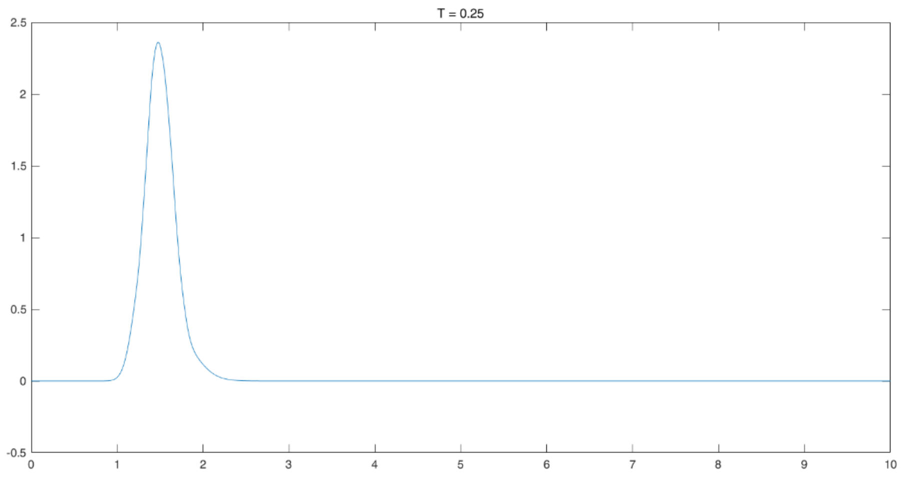
Steps of implementation:

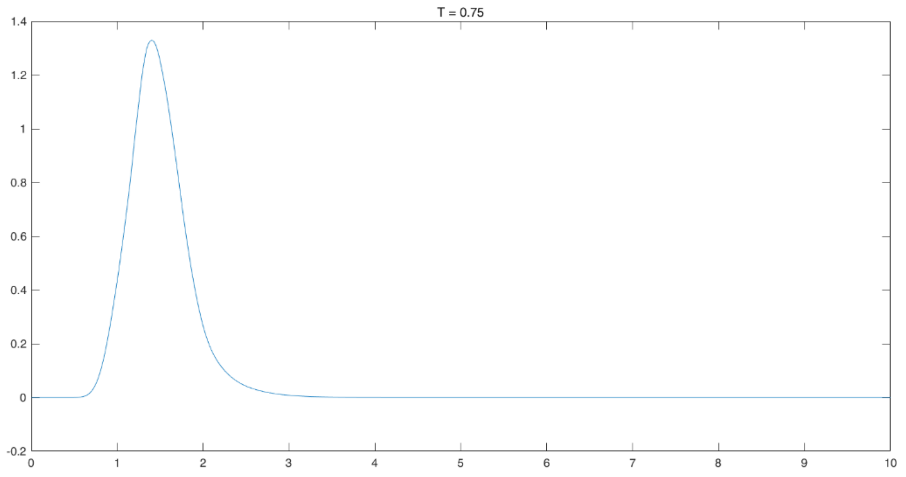
1. Assert validity of the input arguments.
2. Create a $3 \times k$ matrix of *Ks-h, Ks* and *Ks+h,* where *k* is the length of *Ks*, h is 0.0001.
3. Obtain volatilities and the forward prices of the call options by using *getVol* and *getBlackCall*, respectively.
4. If $K - h < 0$, then the PDF value is set to 0. If not, compute the numerical second derivatives by using the MATLAB function *diff.*

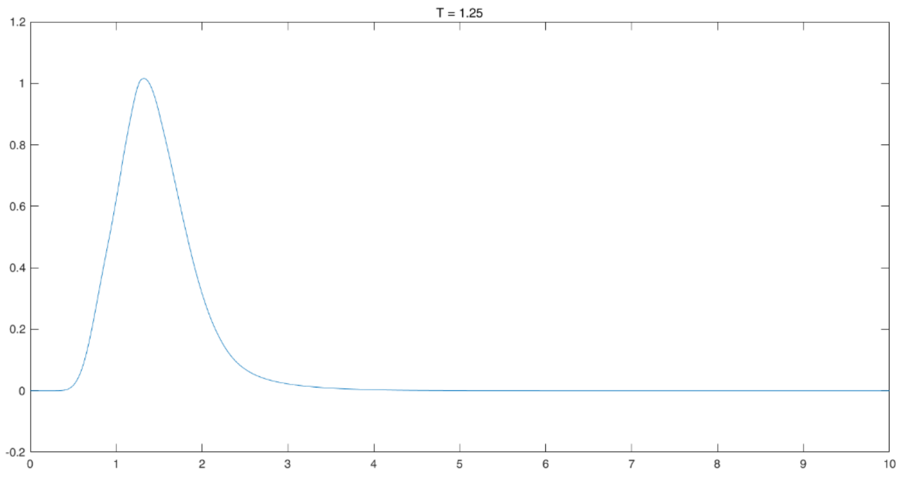
We implement *getPdf* with bump size raging from 1e-16 to 1e+0, then plot the PDF value. We can see that the PDF value decreases wiggly at first and becomes stable thereafter. Thus, we deem $h = 0.0001$ to be reasonable.

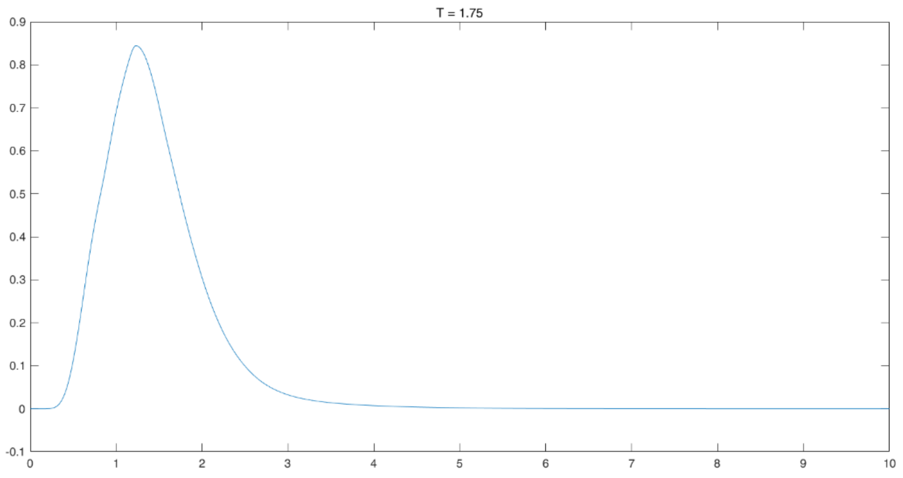


We plot the PDF for each expiry time.









We can see that the PDF behaves like a log-normal distribution as expected.

7.1.2 Test

We implement the function *testGetPdf* to test *getPdf*.
First, we implement *getPdf* with the invalid inputs. The pass criterion is that a meaningful error message is displayed.
Second, we test if the integral of the PDF is 1 by using *getEuropean* with payoff = @(x) 1.
Third, we compute the mean of the PDF by using *getEuropean* with payoff = @(x) x, and compare against the forward spot price.
The pass criterion for the last two tests stated above is that the absolute difference is not greater than 0.00001.

8. Compute forward prices of European options

8.1 getEuropean

8.1.1 Implement

Steps of implementation:

1. Assert validity of function input arguments.

2. Create a handle for the integrand $f(x)\phi(x)$.
3. Use the MATLAB function *integral* to compute numerical integration. If the number of the input arguments is smaller than 4, or *ints* is [0, +Inf], we compute the integration from 0 to Inf. If a repartition of the integration interval is specified, then the integration in sub-intervals is computed separately, and then aggregated.

8.1.2 Test

We implement *testGetEuropean* to test *getEuropean*.

First, we implement *getPdf* with the invalid inputs. The pass criterion is that a meaningful error message is displayed.

Second, we compare the forward price of a call option obtained with Black formula against the price obtained by numerical integration. The pass criterion is that the absolute difference is not greater than 0.00001.