Implementing Bermudan Swaption using QuantLib and Stochastic Models (Hull-White, Black-Karasinski and G2++) for Calibration in Python

AARON DE LA ROSA

A **Bermudan Swaption** is a type of financial derivative that combines aspects of both a swap and an option with multiple exercise dates, falling between an American and European option in terms of exercise flexibility. Here's a breakdown:

1. Definition and Features:

- A **swaption** (swap + option) gives the holder the right, but not the obligation, to enter into an interest rate swap at specified terms.
- In a **Bermudan swaption**, the holder has the right to start the swap on any of several predetermined dates within a specific period, known as "exercise dates." These dates are usually aligned with the reset dates of the swap's floating leg.
- This flexibility to exercise on multiple dates differentiates Bermudan swaptions from **European swaptions** (exercisable only on a single date) and **American swaptions** (exercisable on any date up to and including the expiration date).

2. Use Cases:

- **Interest Rate Risk Management**: Bermudan swaptions are commonly used by banks, financial institutions, and corporations to hedge against interest rate movements. For example, a company with floating-rate debt might buy a Bermudan swaption to lock in a fixed rate if rates increase, but with the flexibility to delay entering the swap if rates move favorably.
- **Speculation on Rate Movements**: Investors may use Bermudan swaptions to speculate on interest rate movements with the potential to enter a swap only if the rates align with their desired outcomes over time.

3. Pricing a Bermudan Swaption:

Pricing a Bermudan swaption is more complex than pricing European or American options due to the multiple exercise dates.

A. Models Used in Pricing:

- **Lattice Models** (Binomial or Trinomial trees): These models discretize interest rate movements over time. For each exercise date, they allow backward induction to compute the swaption's value.
- **Stochastic Models** (Hull-White, Black-Karasinski, or G2++): These models incorporate stochastic interest rate processes. The Hull-White model, for instance, assumes mean-reverting interest rates, which can be calibrated to market data.

- **Monte Carlo Simulation**: For more complex payoffs or if the swap has many possible exercise dates, Monte Carlo simulation is sometimes used, though it can be computationally expensive.

B. Backward Induction:

- For lattice-based methods, backward induction is applied starting from the last possible exercise date and moving backwards to earlier dates.
- At each exercise date, the model calculates the value of the swap (entering the swap) versus the continuation value (deferring the decision).
- The holder of the Bermudan swaption would exercise if the swap value exceeds the continuation value at any given date.

C. Key Inputs for Pricing:

- **Volatility Structure**: Interest rate volatilities at various tenors, often derived from swaption volatility surfaces.
- **Yield Curve**: Used for discounting cash flows and as a basis for constructing forward rates.
- **Model Calibration**: Models like Hull-White or G2++ require calibration of current market data, ensuring they align with observed prices of similar instruments.

4. Example of Pricing a Bermudan Swaption:

In QuantLib, the Bermudan swaption is priced by:

- Setting up a **time grid** for discrete exercise dates.
- Creating a **Bermudan exercise** object with these dates.
- Selecting a **pricing engine** that uses an interest rate model (TreeSwaptionEngine with a Hull-White model).
- Calculating the **net present value (NPV)** of the swaption based on exercise opportunities and projected interest rate paths.

By comparing NPVs across different models or calibration techniques, analysts can choose the pricing method that best suits the swaption's characteristics and market data.

Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

Compartir

+ Código + Texto Conectar Gemini

```python
import QuantLib as ql
import pandas as pd

# ### Setup

todaysDate = ql.Date(15, ql.October, 2024)
ql.Settings.instance().evaluationDate = todaysDate
calendar = ql.TARGET()
settlementDate = ql.Date(19, ql.October, 2024)


def calibrate(model, helpers, l, name):
    print("Model: %s" % name)

    method = ql.Simplex(l)
    model.calibrate(helpers, method, ql.EndCriteria(1000, 250, 1e-7, 1e-7, 1e-7))

    print("Parameters: %s" % model.params())

    totalError = 0.0
    data = []
    for swaption, helper in zip(swaptionVols, helpers):
        maturity, length, vol = swaption
        NPV = helper.modelValue()
```

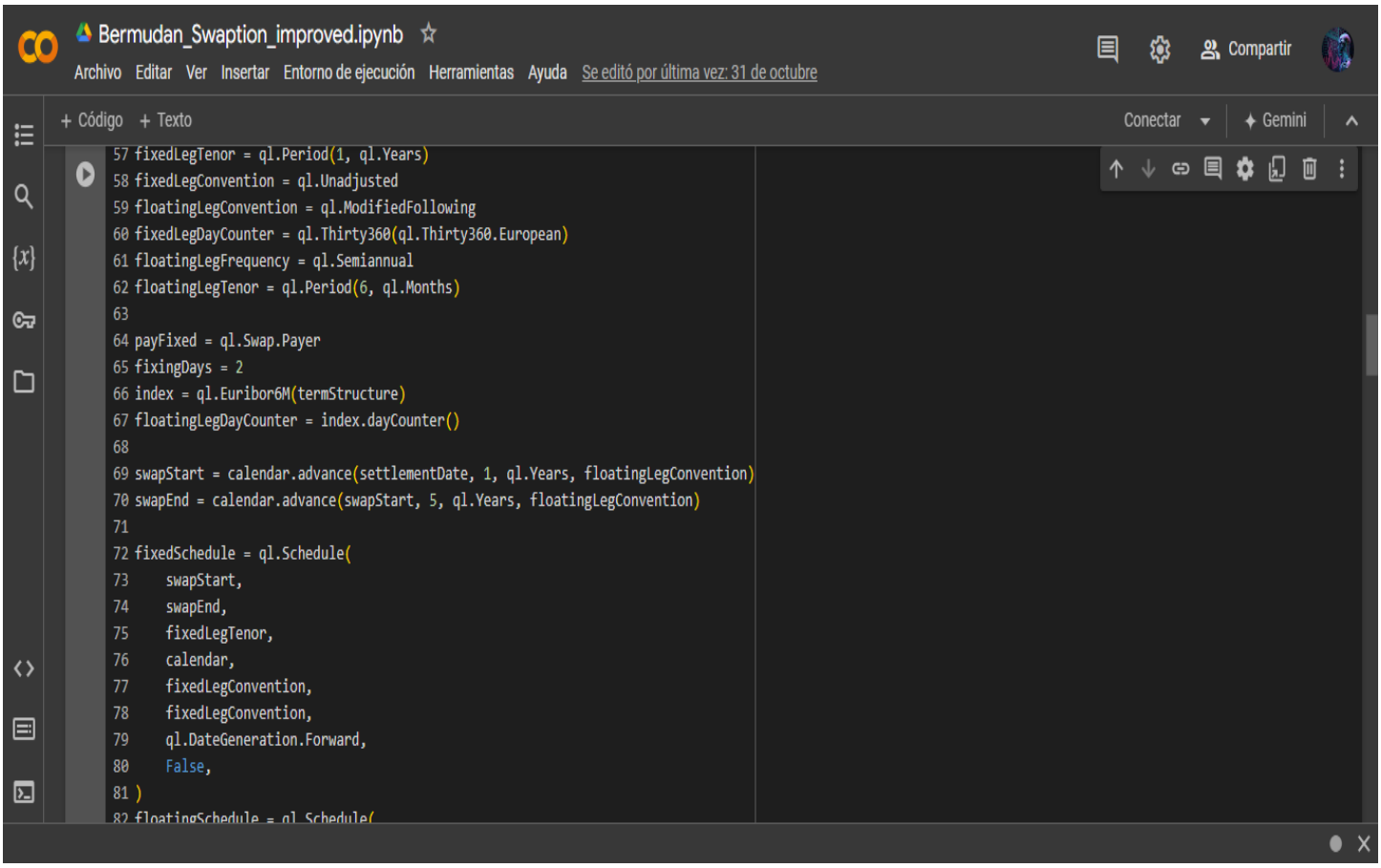
Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

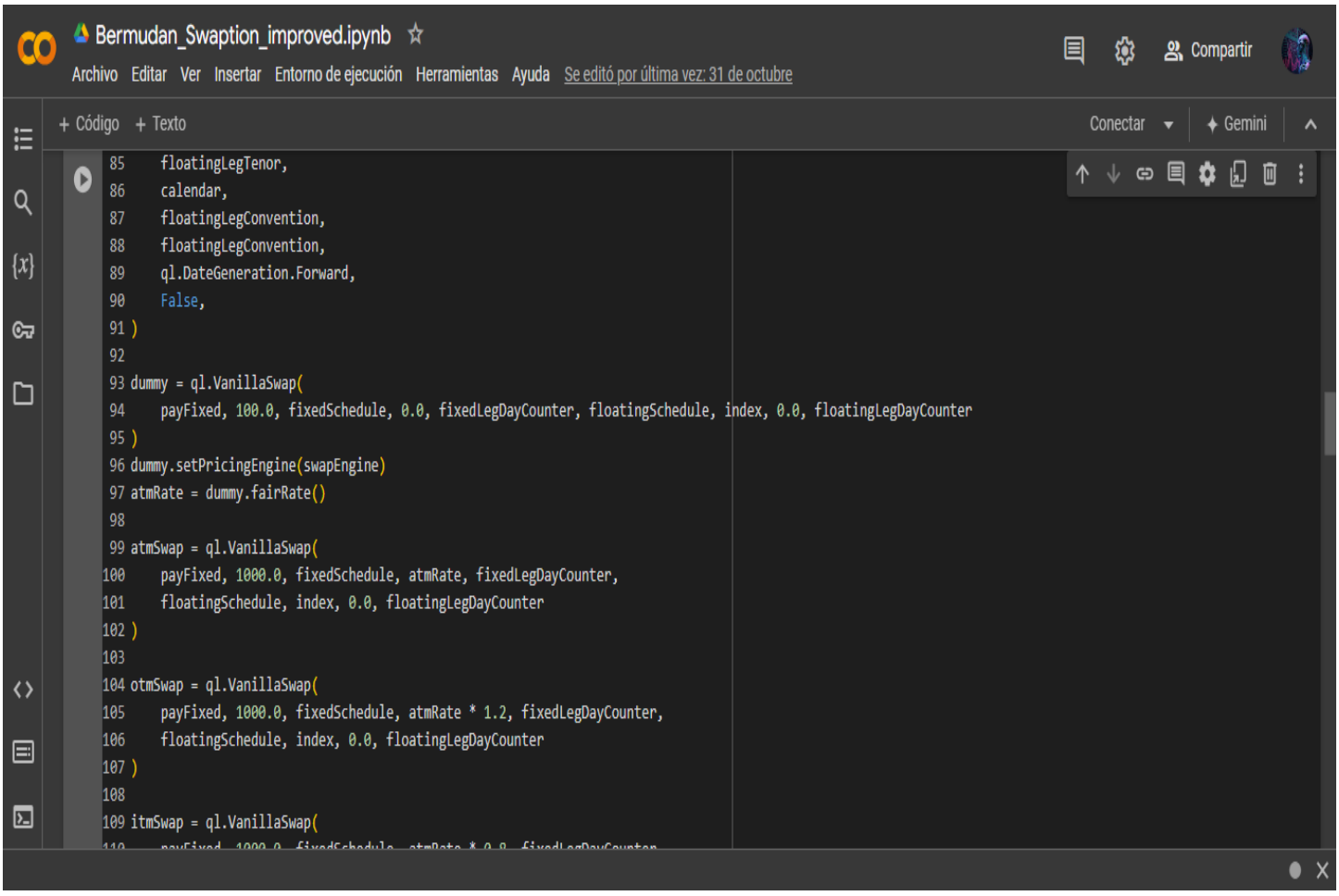
Compartir

+ Código + Texto Conectar Gemini

```python
    print(pd.DataFrame(data, columns=["maturity", "length", "volatility", "implied", "error"]))

    print(f"Average error: {averageError:.4f}")


# ### Market data

swaptionVols = [
    # maturity,          length,            volatility
    (ql.Period(1, ql.Years), ql.Period(5, ql.Years), 0.1148),
    (ql.Period(2, ql.Years), ql.Period(4, ql.Years), 0.1108),
    (ql.Period(3, ql.Years), ql.Period(3, ql.Years), 0.1070),
    (ql.Period(4, ql.Years), ql.Period(2, ql.Years), 0.1021),
    (ql.Period(5, ql.Years), ql.Period(1, ql.Years), 0.1000),
]

# This is a flat yield term structure implying a 1x5 swap at 5%.

rate = ql.QuoteHandle(ql.SimpleQuote(0.04875825))
termStructure = ql.YieldTermStructureHandle(ql.FlatForward(settlementDate, rate, ql.Actual365Fixed()))

# Define the ATM/OTM/ITM swaps:

swapEngine = ql.DiscountingSwapEngine(termStructure)
```




```
fixedLegTenor = ql.Period(1, ql.Years)
fixedLegConvention = ql.Unadjusted
floatingLegConvention = ql.ModifiedFollowing
fixedLegDayCounter = ql.Thirty360(ql.Thirty360.European)
floatingLegFrequency = ql.Semiannual
floatingLegTenor = ql.Period(6, ql.Months)

payFixed = ql.Swap.Payer
fixingDays = 2
index = ql.Euribor6M(termStructure)
floatingLegDayCounter = index.dayCounter()

swapStart = calendar.advance(settlementDate, 1, ql.Years, floatingLegConvention)
swapEnd = calendar.advance(swapStart, 5, ql.Years, floatingLegConvention)

fixedSchedule = ql.Schedule(
    swapStart,
    swapEnd,
    fixedLegTenor,
    calendar,
    fixedLegConvention,
    fixedLegConvention,
    ql.DateGeneration.Forward,
    False,
)
floatingSchedule = ql.Schedule(
```




```
    floatingLegTenor,
    calendar,
    floatingLegConvention,
    floatingLegConvention,
    ql.DateGeneration.Forward,
    False,
)

dummy = ql.VanillaSwap(
    payFixed, 100.0, fixedSchedule, 0.0, fixedLegDayCounter, floatingSchedule, index, 0.0, floatingLegDayCounter
)
dummy.setPricingEngine(swapEngine)
atmRate = dummy.fairRate()

atmSwap = ql.VanillaSwap(
    payFixed, 1000.0, fixedSchedule, atmRate, fixedLegDayCounter,
    floatingSchedule, index, 0.0, floatingLegDayCounter
)

otmSwap = ql.VanillaSwap(
    payFixed, 1000.0, fixedSchedule, atmRate * 1.2, fixedLegDayCounter,
    floatingSchedule, index, 0.0, floatingLegDayCounter
)

itmSwap = ql.VanillaSwap(
```

Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

Compartir

+ Código + Texto

Conectar ✦ Gemini

```
113
114 atmSwap.setPricingEngine(swapEngine)
115 otmSwap.setPricingEngine(swapEngine)
116 itmSwap.setPricingEngine(swapEngine)
117
118 helpers = [
119     ql.SwaptionHelper(
120         maturity,
121         length,
122         ql.QuoteHandle(ql.SimpleQuote(vol)),
123         index,
124         index.tenor(),
125         index.dayCounter(),
126         index.dayCounter(),
127         termStructure,
128     )
129     for maturity, length, vol in swaptionVols
130 ]
131
132 times = {}
133 for h in helpers:
134     for t in h.times():
135         times[t] = 1
136 times = sorted(times.keys())
137
```

Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

Compartir

+ Código + Texto

Conectar ✦ Gemini

```
145 # ### Calibrations
146
147 for h in helpers:
148     h.setPricingEngine(ql.G2SwaptionEngine(G2model, 6.0, 16))
149 calibrate(G2model, helpers, 0.05, "G2 (analytic formulae)")
150
151 for h in helpers:
152     h.setPricingEngine(ql.JamshidianSwaptionEngine(HWmodel))
153 calibrate(HWmodel, helpers, 0.05, "Hull-White (analytic formulae)")
154
155 for h in helpers:
156     h.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, grid))
157 calibrate(HWmodel2, helpers, 0.05, "Hull-White (numerical calibration)")
158
159 for h in helpers:
160     h.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, grid))
161 calibrate(BKmodel, helpers, 0.05, "Black-Karasinski (numerical calibration)")
162
163
164 # ### Price Bermudan swaptions on defined swaps
165
166 bermudanDates = [d for d in fixedSchedule][:-1]
167 exercise = ql.BermudanExercise(bermudanDates)
168
169 atmSwaption = ql.Swaption(atmSwap, exercise)
```

Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

Compartir

+ Código + Texto

Conectar Gemini

```
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(G2model, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(G2model, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(G2model, 50))

data.append(("G2 analytic", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))

data.append(("HW analytic", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))

data.append(("HW numerical", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))

```

Bermudan_Swaption_improved.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se editó por última vez: 31 de octubre

Compartir

+ Código + Texto

Conectar Gemini

```
data.append(("G2 analytic", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel, 50))

data.append(("HW analytic", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(HWmodel2, 50))

data.append(("HW numerical", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))

# +
atmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))
otmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))
itmSwaption.setPricingEngine(ql.TreeSwaptionEngine(BKmodel, 50))

data.append(("BK numerical", itmSwaption.NPV(), atmSwaption.NPV(), otmSwaption.NPV()))
# -

print(pd.DataFrame(data, columns=["model", "in-the-money", "at-the-money", "out-of-the-money"]))
```

```
Model: G2 (analytic formulae)
Parameters: [ 0.0514855; 0.00593751; 0.0514855; 0.0101892; -0.797363 ]
  maturity length  volatility   implied     error
0       1Y     5Y      0.1148  0.100463 -0.014337
1       2Y     4Y      0.1108  0.104979 -0.005821
2       3Y     3Y      0.1070  0.107050  0.000050
3       4Y     2Y      0.1021  0.108437  0.006337
4       5Y     1Y      0.1000  0.109483  0.009483
Average error: 0.0072
Model: Hull-White (analytic formulae)
Parameters: [ 0.050956; 0.0059453 ]
  maturity length  volatility   implied     error
0       1Y     5Y      0.1148  0.106215 -0.008585
1       2Y     4Y      0.1108  0.106328 -0.004472
2       3Y     3Y      0.1070  0.106352 -0.000648
3       4Y     2Y      0.1021  0.106413  0.004313
4       5Y     1Y      0.1000  0.106601  0.006601
Average error: 0.0049
Model: Hull-White (numerical calibration)
Parameters: [ 0.0533974; 0.00604552 ]
  maturity length  volatility   implied     error
0       1Y     5Y      0.1148  0.102693 -0.012107
1       2Y     4Y      0.1108  0.105110 -0.005690
2       3Y     3Y      0.1070  0.106420 -0.000580
3       4Y     2Y      0.1021  0.107499  0.005399
4       5Y     1Y      0.1000  0.108596  0.008596
Average error: 0.0065
Model: Black-Karasinski (numerical calibration)
Parameters: [ 0.0420413; 0.119726 ]
  maturity length  volatility   implied     error
0       1Y     5Y      0.1148  0.102851 -0.011949
1       2Y     4Y      0.1108  0.105308 -0.005492
2       3Y     3Y      0.1070  0.106518 -0.000482
3       4Y     2Y      0.1021  0.107459  0.005359
4       5Y     1Y      0.1000  0.108400  0.008400
Average error: 0.0063
          model  in-the-money  at-the-money  out-of-the-money
0   G2 analytic     42.752925     14.214943          3.280791
1   HW analytic     42.294361     12.983510          2.534785
2  HW numerical     42.353560     13.129363          2.610155
3  BK numerical     41.820925     13.016318          3.266603
```

Calibration Results

For each model, we see the **parameters** and the implied volatilities compared to market volatilities, along with **average errors**:

1. **G2 (analytic formulae)**:
    - **Parameters**: [0.0514855, 0.00593751, 0.0514855, 0.0101892, -0.797363]
    - **Average Error**: 0.0072
    - The G2 model shows a relatively higher average error than other models, particularly on shorter maturity swaptions (1Y x 5Y and 5Y x 1Y). However, its analytic formula provides a close fit overall.
2. **Hull-White (analytic formulae)**:
    - **Parameters**: [0.050956, 0.0059453]
    - **Average Error**: 0.0049

- This model performs well, achieving the lowest average error. Its implied volatilities are closer to market data, especially for the shorter and medium tenors, making it quite reliable for this dataset.

3. **Hull-White (numerical calibration)**:
    - **Parameters**: [0.0533974, 0.00604552]
    - **Average Error**: 0.0065
    - With a numerical calibration, the Hull-White model's error is slightly higher than the analytic version, but it still fits reasonably well across all maturities.
4. **Black-Karasinski (numerical calibration)**:
    - **Parameters**: [0.0420413, 0.119726]
    - **Average Error**: 0.0063
    - The Black-Karasinski model is comparable to the numerically calibrated Hull-White model in terms of error, with relatively good performance across all maturities and a slightly lower overall error than G2.

Bermudan Swaption Pricing Results

The NPVs for each swaption (in-the-money, at-the-money, and out-of-the-money) under different models are as follows:

Model	In-the-money	At-the-money	Out-of-the-money
G2 analytic	42.75	14.21	3.28
HW analytic	42.29	12.98	2.53
HW numerical	42.35	13.13	2.61
BK numerical	41.82	13.02	3.27

Observations:

- **Consistency Across Models**: The in-the-money values are relatively close across models, suggesting a consistent estimation of swaption value when the swap is highly likely to be exercised.
- **Out-of-the-Money Valuations**: Out-of-the-money valuations are more sensitive to model choice, particularly between Hull-White and Black-Karasinski. The G2 model's out-of-the-money value (3.28) is close to Black-Karasinski, while Hull-White (analytic and numerical) provides slightly lower values.
- **Hull-White Stability**: Hull-White analytic and numerical results are very similar, indicating model robustness in both calibration methods.

Conclusions

1. **Model Preference**: For lower average error and pricing stability, the Hull-White (analytic) model seems preferable given the dataset, though the G2 model also performs adequately for in-the-money swaptions.
2. **In-the-Money Emphasis**: All models price in-the-money swaptions consistently, so any of the models could be reliable for deep in-the-money Bermudan swaptions.
3. **Out-of-the-Money Sensitivity**: The out-of-the-money valuations vary slightly, suggesting that the choice of model might have a more noticeable impact here. Hull-White might provide a slightly conservative approach compared to G2 and Black-Karasinski.

Overall, these results suggest that each model has trade-offs, and the best choice may depend on the specific needs of your analysis (focusing on low errors or prioritizing certain moneyness levels).