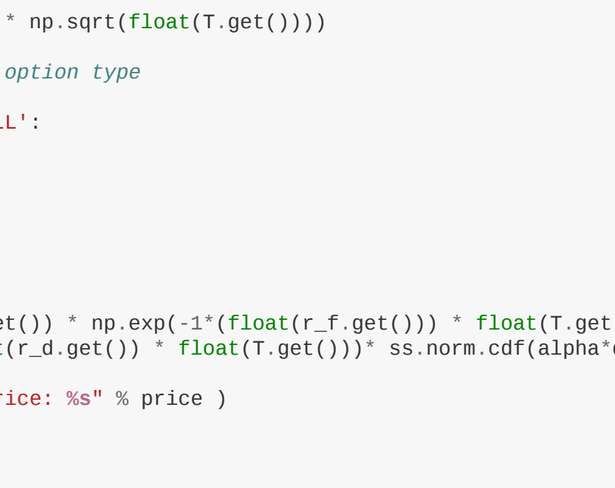
Garman-Kohlhagen Option Pricing Model

The standard Black-Scholes Model assumes that the interest rate is constant and homogeneous across all countries. Garman-Kohlhagen Model introduces a diffential component between domestic and foreign risk-free interest rate, since each country has exposure to different risk factors. As a result, this model has been used to compute the fair value of European FX Options.

In [6]:
```
__author__ = 'f-araujo8'

### Import module ###
import numpy as np
import scipy.stats as ss
from tkinter import *

### GUI configuration ###
root = Tk()
root.title("Garman-Kohlhagen Option Pricing Model")

## Setting geometry ##
root.geometry('400x300+600+200')
root.resizable(False,False)

### Label ###

text_array = ["SPOT:","STRIKE:","TIME TO MATURITY:",
"DOMESTIC RISKLESS INTEREST RATE","FOREIGN RISKLESS INTEREST RATE:",
"VOLATILITY OF THE UNDERLYING ASSET:","OPTION TYPE:"]

# Dictionary (Parameters)
parameter_code = ['S', 'K', 'T', 'r_d', 'r_f','sigma']

zip_iterator = zip(text_array, parameter_code)

param_dictionary = dict(zip_iterator)

for label_text in text_array:
    temp_variable_label = '{}'.format(label_text)
    Label(root, text=temp_variable_label).grid(row = text_array.index(temp_variable_label), sticky = W)

    ## User Input ##
    dictionary_value = dict.get(param_dictionary, temp_variable_label)
    locals()[dictionary_value] = Entry(root)

    if label_text != text_array[-1]:
        locals()[dictionary_value].grid(row=text_array.index(temp_variable_label), column=1)


# Temp variable
temp_variable = StringVar()

## Dropdown Menu (Option_type) ##
dropDownList = ["CALL", "PUT"]

drop_down = OptionMenu(root, temp_variable, *dropDownList)

temp_variable.set(dropDownList[0])

drop_down.grid(row=len(text_array) - 1, column=1)

### output label ###
rlabel1 = Label(root)
rlabel1.grid(row = len(text_array), sticky = W)

### main function - Garman-Kohlhagen Model ###
def FX_vanilla() :

    d1 = (np.log((float(S.get())) / (float(K.get()))) + ((float(r_d.get()) - float(r_f.get())) + 0.5 * (float(sigma.
get())) ** 2) * (float(T.get()))) / ((float(sigma.get())) * np.sqrt(float(T.get())))

    d2 = d1 - (float(sigma.get()) * np.sqrt(float(T.get())))

    ### If condition based on the option type

    if temp_variable.get() == 'CALL':

        alpha = 1

    else:

        alpha = -1

    price = int(alpha)*(float(S.get()) * np.exp(-1*(float(r_f.get())) * float(T.get()))*ss.norm.cdf(alpha*d1, 0, 1)
- float(K.get()) * np.exp(-1*float(r_d.get()) * float(T.get()))* ss.norm.cdf(alpha*d2, 0, 1))

    rlabel1.config(text="Option price: %s" % price )

    return

### button ###
btn = Button(root, text="Compute the price", command=FX_vanilla).grid(row = 7, column = 1, sticky = E)

### mainloop ###
root.mainloop()
```



The user enters valid parameters of the model and it returns the fair value based on the option type.