Monte-Carlo-Simulation und Single- v.s. Multi-Processing

April 14, 2020

1 Monte-Carlo-Simulation und Single- v.s. Multi-Processing

© Thomas Robert Holy 2020 Version 1.0 Visit me on GitHub: https://github.com/trhOly ## Grundlegende Einstellungen: Zunächst müssen die notwendigen Pakete (auch Module) importiert werden, damit auf diese zugegriffen werden kann.

[1]:
```
import pandas as pd
import numpy as np
from riskmeasure_module import risk_measure as rm
import math
from multiprocessing import Process
from multiprocessing import Manager
from multiprocessing.pool import Pool
import datetime as dt
import operator
from IPython.core.display import display, HTML
from multi_lb import repeat_parallel, RM_frame_func, plotty_func
from Monte_Carlo_Simulation_lite import var_covar_matrix_func, var_func, cholesky_func, verteilung_func, copula_sim, hist_func
```

Anschließend werden Einstellungen definiert, die die Formatierung der Ausgaben betreffen. Hierfür wird das Modul `operator` genutzt. Außerdem wird die Größe der Grafiken modifiziert, welche später angezeigt werden sollen.

[2]:
```
%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999;
```

```
<IPython.core.display.Javascript object>
```

[3]:
```
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

1.1 Definition einiger Funktionen zum Vergleich von Single- v.s. Multi-Processing

1.1.1 Definition der Multi-Processing-Funktion

[4]:
```
# Definiere die Funktion "run" in welcher die Parallelisierung der Monte-Carlo-Simulation auf "Multi_lb" gemanaget wird
def run(runs_func, runs_sim, rand_x, rand_y, mu, std_list, corr_list, alpha, gamma, draw=False, SCREEN_WIDTH=115):
    RM_VaR_list = []
    RM_CVaR_list = []
    RM_PSRM_list = []
    mega_summe_list = []

    centered=operator.methodcaller('center', SCREEN_WIDTH)

    start = dt.datetime.now() # Startpunkt Zeitmessung
    with Manager() as manager: # Verwendung Funktion "Manager()" aus "multiprocessing" als Manager der geteilten Listen zwischen den parallel laufenden Simulationen
        shared_list = manager.list() # Legt die leere Liste "shared_list" an, welche zwischen den parallel laufenden Simulationen geteilt wird und alle Realisationen den einzelnen Simulationsläufe enthält
        VaR_list = manager.list() # Legt die leere Liste "VaR_list" an, welche zwischen den parallel laufenden Simulationen geteilt wird und alle VaR enthält
        CVaR_list = manager.list() # Legt die leere Liste "CVaR_list" an, welche zwischen den parallel laufenden Simulationen geteilt wird und alle CVaR enthält
        PSRM_list = manager.list() # Legt die leere Liste "PSRM_list" an, welche zwischen den parallel laufenden Simulationen geteilt wird und alle P-SRM enthält
        processes = [] # Legt die leere Liste "processes" an, in welcher die auszuführenden Prozesse abgelegt werden
        # Für jedes i in der Range 0 bis runs_func (Simulationsläufe)...
        for i in range(runs_func):
            p = Process(target=repeat_parallel, args=(runs_sim, rand_x, rand_y, mu, std_list, corr_list, alpha, gamma, shared_list, VaR_list, CVaR_list, PSRM_list, i)) # Erstelle den Prozess "p", welcher die Funktion "repeat_parallel" ausführt mit den gebenen Parametern
            p.start() # Starte den Prozess
            processes.append(p) # Füge den Prozess "p" der Liste "processes" an
        # Für jeden Prozess "p" in "processes" führe due Funktion join() aus (Clean Exit Process)
        for p in processes:
            p.join()
        RM_VaR_list += VaR_list # Füge der "RM_VaR_list" den jeweiligen VaR an
        RM_CVaR_list += CVaR_list # s.o.
        RM_PSRM_list += PSRM_list # s.o.
        # Sofern draw == 'True' füge die "shared_list" in der "mega_summe_list" an
        if draw == True:
            mega_summe_list += shared_list
    end = dt.datetime.now() # Endpunkt für die Zeitmessung
    RM_frame_func(runs_sim, runs_func, RM_VaR_list, RM_CVaR_list, RM_PSRM_list, SCREEN_WIDTH, centered) # Ausführen der Funktion, welche den DataFrame mit den Risikomaßen ausgibt
    # Sofern draw == 'True' führe die Funktion aus, welche die Verteilungfunktionen der Simulationsläufe plottet
    if draw == True:
        plotty_func(runs_sim, runs_func, mega_summe_list)
    print(end-start) # Gibt das Zeit-Delta zurück

#----------------------------------------
# Vereinfachung
def multithreading(n, rand_x, rand_y, var_x, var_y, corr_list, std_list, mu, alpha, gamma, runs_func, runs_sim):
    run(runs_func, runs_sim, rand_x, rand_y, mu, std_list, corr_list, alpha, gamma, draw=False)
```

1.1.2 Definition der Sinlge-Processing-Funktion

[5]:
```
def single_thread(n, rand_x, rand_y, var_x, var_y, corr_list, std_list, mu, alpha, gamma, runs_func, runs_sim):

    SCREEN_WIDTH = 115
    centered = operator.methodcaller('center', SCREEN_WIDTH)

    RM_VaR_list, RM_CVaR_list = [], []
    RM_PSRM_list, mega_summe_list = [], []

    # Führe die Simulation "runs_func mal" durch und speichere die Ergebnisse in der jeweiligen Liste
    start = dt.datetime.now()
    for i in range(0, runs_func):
        _, _, _, total_summe_liste = copula_sim(runs_sim, rand_x, rand_y, mu, std_list, corr_list, full_log=False)
        mega_summe_list += total_summe_liste
        x = rm(total_summe_liste, alpha, gamma)
        RM_VaR_list.append(x.VaR())
        RM_CVaR_list.append(x.CVaR())
        RM_PSRM_list.append(x.Power())
    end = dt.datetime.now()
    print(end-start)

    #---------------------------
    # Erzeuge ein DataFrame mit den Simulationsergebnissen
    # und deren prozentualen Änderung vom jeweils vorherigen Ergebnis
    RM_frame = pd.DataFrame()
    RM_frame['VaR'] = RM_VaR_list
    RM_frame['VaR-Change'] = RM_frame['VaR'].pct_change()
    RM_frame['CVaR'] = RM_CVaR_list
    RM_frame['CVaR-Change'] = RM_frame['CVaR'].pct_change()
    RM_frame['Power'] = RM_PSRM_list
    RM_frame['Power-Change'] = RM_frame['Power'].pct_change()

    #---------------------------
    # Ermittle die kleinste und größte Relaisation des jweiligen Risikomaßes
    Min_Max_VaR = (min(RM_VaR_list), max(RM_VaR_list))
    Min_Max_CVaR = (min(RM_CVaR_list), max(RM_CVaR_list))
    Min_Max_PSRM = (min(RM_PSRM_list), max(RM_PSRM_list))

    #---------------------------
    # Gib den DataFrame und die Infos zurück
    print('#' + SCREEN_WIDTH * '-' + '#')
    print('|' + centered('[INFO] Der DataFrame mit den auf den auf ' +str(runs_func) + ' mal ' + str(runs_sim) + ' Durchläufen beruhenden Risikomaßen ergibt sich wie folgt:') + '|')
    print('#' + SCREEN_WIDTH * '-' + '#')
    print(RM_frame)
    print('#' + SCREEN_WIDTH * '-' + '#')
    print('|' + centered('Nach ' + str(runs_func) + ' Simulationsläufen mit je ' + str(runs_sim) + ' Durchläufen beträgt der kleinste VaR ' + str(round(Min_Max_VaR[0],2)), 'größte ' + str(round(Min_Max_VaR[1],2)) + ' (\u0394 = ' + str(round((float(Min_Max_VaR[0]/Min_Max_VaR[1])-1)*100,2))) + '%).') + '|')
    print('|' + centered('Nach ' + str(runs_func) + ' Simulationsläufen mit je ' + str(runs_sim) + ' Durchläufen beträgt der kleinste CVaR ' + str(round(Min_Max_CVaR[0],2)), 'größte ' + str(round(Min_Max_CVaR[1],2)) + ' (\u0394 = ' + str(round((float(Min_Max_CVaR[0]/Min_Max_CVaR[1])-1)*100,2))) + '%).') + '|')
    print('|' + centered('Nach ' + str(runs_func) + ' Simulationsläufen mit je ' + str(runs_sim) + ' Durchläufen beträgt das kleinste P-SRM ' + str(round(Min_Max_PSRM[0],2)), 'das größte ' + str(round(Min_Max_PSRM[1],2)) + ' (\u0394 = ' + str((round((float(Min_Max_PSRM[0]/Min_Max_PSRM[1])-1)*100,2))) + '%).') + '|')
    print('#' + SCREEN_WIDTH * '-' + '#')
```

1.2 Variablen spezifizieren

Die für die Simulation notwenigen Variablen werden definiert.

[6]:
```
#----------------------------------------
# Parameter für die Simulation
#----------------------------------------
# Anzahl Simulationsdurchläufe
n = 10000
# Neue Randverteilungen (Gleichverteilung)
rand_x = [10,20]
rand_y = [8,22]
# Varianzen und Korrelation(en)
var_x = 4
var_y = 9
corr_list = [0]
std_list = [math.sqrt(var_x), math.sqrt(var_y)]
# Erwartungswerte
mu = [2, 3]
#----------------------------------------
#----------------------------------------
# Parameter Risikomaße
alpha = 0.05
gamma = 0.5
#----------------------------------------
# Anzahl Simulationsläufe und Durchläufe pro Simulation
runs_func = 10
runs_sim = 100000
```

1.3 Funktionsaufruf

[7]:
```
if __name__=='__main__':
    print('Ergebnis Multi-threaded:\n')
    multithreading(n, rand_x, rand_y, var_x, var_y, corr_list, std_list, mu, alpha, gamma, runs_func, runs_sim)
    print('\n\nErgebnis Single-threaded:\n')
    single_thread(n, rand_x, rand_y, var_x, var_y, corr_list, std_list, mu, alpha, gamma, runs_func, runs_sim)
```

Ergebnis Multi-threaded:

```
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
| [INFO] Der DataFrame mit den auf den auf 10 mal 100000 Durchläufen beruhenden
Risikomaßen ergibt sich wie folgt:  |
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
        VaR  VaR-Change       CVaR  CVaR-Change      Power  Power-Change
0 -21.758507         NaN -20.504731         NaN  26.828263           NaN
1 -21.707793   -0.002331 -20.462340   -0.002067  26.817743    -0.000392
2 -21.717064    0.000427 -20.485782    0.001146  26.799571    -0.000678
3 -21.732111    0.000693 -20.483403   -0.000116  26.834661     0.001377
4 -21.740721    0.000396 -20.492935    0.000465  26.834742    -0.000064
5 -21.778684    0.001746 -20.540957    0.002343  26.870833     0.001345
6 -21.763236   -0.000709 -20.519158   -0.001061  26.850804    -0.000745
7 -21.730395   -0.001509 -20.482281   -0.001797  26.830540    -0.000755
8 -21.721129   -0.000426 -20.482142   -0.000007  26.826441    -0.000153
9 -21.789455    0.003146 -20.531516    0.002411  26.862185     0.001332
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt der kleinste VaR
-21.79, der größte -21.71 (0.38%).|
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt der kleinste CVaR
-20.54, der größte -20.46 (0.38%).|
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt das kleinste P-SRM
26.8, das größte 26.87 (-0.27%).|
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
0:00:42.612356


Ergebnis Single-threaded:

0:03:51.417429
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
| [INFO] Der DataFrame mit den auf den auf 10 mal 100000 Durchläufen beruhenden
Risikomaßen ergibt sich wie folgt:  |
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
        VaR  VaR-Change       CVaR  CVaR-Change      Power  Power-Change
0 -21.721779         NaN -20.473054         NaN  26.811895           NaN
1 -21.731741    0.000459 -20.487463    0.000704  26.821491     0.000358
2 -21.749623    0.000823 -20.497513    0.000491  26.829076     0.000283
3 -21.759824    0.000469 -20.514984    0.000852  26.834125     0.000188
4 -21.759102   -0.000033 -20.514246   -0.000036  26.832232    -0.000071
5 -21.734240   -0.001143 -20.481392   -0.001602  26.826350    -0.000219
6 -21.769326    0.001614 -20.525132    0.002136  26.849975     0.000881
7 -21.744877   -0.001123 -20.499863   -0.001231  26.810697    -0.001463
8 -21.736873   -0.000368 -20.483276   -0.000809  26.816251     0.000207
9 -21.681204   -0.002561 -20.460268   -0.001123  26.813104    -0.000117
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt der kleinste VaR
-21.77, der größte -21.68 (0.41%).|
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt der kleinste CVaR
-20.53, der größte -20.46 (0.32%).|
|Nach 10 Simulationsläufen mit je 100000 Durchläufen beträgt das kleinste P-SRM
26.81, das größte 26.85 (-0.15%).|
#-------------------------------------------------------------------------------------------------------
-------------------------------------#
```