discount_factor

January 30, 2018

In [95]:
```
import csv
import datetime
import numpy as np
from scipy.interpolate import interp1d

def extract_1d_list(nested_list, index):
    extracted_list = []
    for i in range(len(nested_list)):
        extracted_list.append(nested_list[i][index])
    return extracted_list

def interpolation_extract_list(original_list, index_xaxis, index_yaxis):
    xaxis = []
    yaxis = []
    for i in range(len(original_list)):
        xaxis.append(float(original_list[i][index_xaxis]))
        yaxis.append(float(original_list[i][index_yaxis]))
    f_interpolation = interp1d(xaxis, yaxis)
    return f_interpolation

def calc_days(tenor_name):
        if (tenor_name[-1] == 'Y'):
            tenor_days = float(tenor_name[0:-1]) * 365
        elif (tenor_name[-1] == 'M'):
            tenor_days = float(tenor_name[0:-1]) * 30
        elif (tenor_name[-1] == 'W'):
            tenor_days = float(tenor_name[0:-1]) * 7
        elif (tenor_name == ' O/N' or 'T/N'):
            tenor_days = 1
        return int(tenor_days)

def calc_month(tenor_name):
        if (tenor_name[-1] == 'Y'):
            tenor_month = float(tenor_name[0:-1]) * 12
        elif (tenor_name[-1] == 'M'):
            tenor_month = float(tenor_name[0:-1]) * 1
        elif (tenor_name[-1] == 'W'):
            tenor_month = float(tenor_name[0:-1])  / 4
```

```
        elif (tenor_name == ' O/N' or 'T/N'):
            tenor_month = 1/30
        return int(tenor_month)

def calc_trade_days(start_day, end_day):
    datetime_obj_start = datetime.datetime.strptime(start_day, '%Y/%m/%d')
    datetime_obj_end = datetime.datetime.strptime(end_day, '%Y/%m/%d')
    return (datetime_obj_end - datetime_obj_start).days

class discount_factor:
    def __init__(self, ir_list_name, ccy):
        ## コンストラクタの順番に注意.　先に_ir_listを定義し，_load_ir_listの中で_base_
        ## を呼び出すと，_base_dateがまだ定義されていないのでエラーがでる.
        # base_date -> trade_date??
        self._base_date = ir_list_name[0:4] + '/' + ir_list_name[4:6] + '/' + ir_list_n
        # spot dateを２営業日後としている.
        self._spot_date = self._calc_end_date(self._base_date, '2.0D')
        self._ir_list = self._set_ir_list(ir_list_name, ccy)
        self._roll_month = float(self._ir_list[-1][4][0])
        self._str_roll_month = str(self._roll_month) + 'M'
        self._convention = int(self._ir_list[0][5][-3:])
        self._str_convention = self._ir_list[0][5]
        self._string_swap = self._ir_list[-1][1]
        self._string_mm = self._ir_list[0][1]
        self._ccy = self._ir_list[0][2]
#         self._ir_list_DF_mm = self._calc_DF_money_market()

    def _csv_read_ir_list(self, ir_list_name):
        with open(ir_list_name, 'r') as csvfile:
            reader_obj = csv.reader(csvfile)
            # rewritten header_obj by using next method(???)
            header_obj = next(reader_obj)
            ir_list = []
            for row in reader_obj:
                ir_list.append(row)
            return ir_list

    def _set_ir_list(self, ir_list_name, ccy):
        ir_list = self._select_ccy_ir_list(ir_list_name, ccy)
        # change int type to float type (ex. 1Y -> 1.0Y)
        temp_num = [[] for i in range(len(ir_list))] # comprehension expression for mak
        for i in range(len(ir_list)):
            if (ir_list[i][0][0].isdigit()):
                num_tenor = ir_list[i][0][0: len(ir_list[i][0])-1]
                unit_tenor = ir_list[i][0][-1]
                temp_num[i] = "{:.1f}".format(int(num_tenor))
                ir_list[i][0] = temp_num[i] + unit_tenor
        ir_list_with_cf = self._add_cash_flow(ir_list)
```

```
        return ir_list_with_cf

    def _select_ccy_ir_list(self, ir_list_name, ccy):
        ir_list = self._csv_read_ir_list(ir_list_name)
        indices_selected_ccy = self._select_ccy(ir_list, ccy)
        first_index_selected_ccy = indices_selected_ccy[0]
        last_index_selected_ccy = indices_selected_ccy[-1] + 1
        selected_ccy_ir_list = []
        for i in range(first_index_selected_ccy, last_index_selected_ccy):
            selected_ccy_ir_list.append(ir_list[i])
        return selected_ccy_ir_list

    def _select_ccy(self, ir_list, ccy):
        extract_ccy_list = extract_1d_list(ir_list, 2)
        indices_ccy = [i for i, ccy_name in enumerate(extract_ccy_list) if (ccy_name ==
        return indices_ccy

    def _add_cash_flow(self, ir_list):
        obj_trade_date = datetime.datetime.strptime(self._base_date, '%Y/%m/%d')
        over_night_date = (obj_trade_date + datetime.timedelta(days=1)).strftime('%Y/%m
        spot_date = (obj_trade_date + datetime.timedelta(days=2)).strftime('%Y/%m/%d')
        for i in range(len(ir_list)):
            if (ir_list[i][0] == 'O/N'):
                ir_list[i].append(self._base_date)
                ir_list[i].append(over_night_date)
            elif (ir_list[i][0] == 'T/N'):
                ir_list[i].append(over_night_date)
                ir_list[i].append(spot_date)
            else:
                ir_list[i].append(spot_date)
                ir_list[i].append(self._calc_end_date(spot_date, ir_list[i][0]))
        return ir_list

    def _calc_end_date(self, start_day, str_maturity):
        datetime_obj_start = datetime.datetime.strptime(start_day, '%Y/%m/%d')
        unit = str_maturity[-1]
        # extract a part of integer from the float plus unit type.
        # ex. '10.0Y'[0: len(10.0Y)- 3] -> '10'
        int_num = int(str_maturity[0:len(str_maturity)-3])
        if (unit == 'D'):
            trade_days = int_num
        elif (unit == 'W'):
            trade_days = int_num * 7
        elif (unit == 'M'):
            trade_days = int_num * 30
        elif (unit == 'Y'):
            trade_days = int_num * 365
        end_day = datetime_obj_start + datetime.timedelta(days=trade_days)
```

```
        return end_day.strftime('%Y/%m/%d')

    def _calc_day_count_fraction(self, start_date, end_date):
        datetime_obj_start = datetime.datetime.strptime(start_date, '%Y/%m/%d')
        datetime_obj_end = datetime.datetime.strptime(end_date, '%Y/%m/%d')
        daycount = float((datetime_obj_end - datetime_obj_start).days / self._conventio
        return daycount

    def get_convention(self):
        return self._convention

    def get_ir_list(self):
        return self._ir_list

    def get_base_date(self):
        return self._base_date

    def get_ccy(self):
        return self._ccy

    def get_roll_month(self):
        return self._roll_month

    def get_ir_list_with_DF_money_market(self):
        ir_list_DF_mm = self._calc_DF_money_market()
        return ir_list_DF_mm

    def get_ir_list_with_DF_swap_rate(self):
        ir_list_DF_sr = self._calc_DF_swap_rate()
        return ir_list_DF_sr

    def get_ir_list_interpolated_swap_rate(self):
        ir_list_interpolated_swap_rate = self._interpolated_ir_list_for_bootstrap()
        return ir_list_interpolated_swap_rate

    def get_ir_list_interpolated_DF_money_market(self):
        f_interpolation_DF_mm = self.interpolate_DF_money_market()
        return f_interpolation_DF_mm

    def _calc_DF_money_market(self):
        len_MM = 0
        for i in range(len(self._ir_list)):
            if (self._ir_list[i][1] == 'Money Market'):
                len_MM += 1
        ir_list_DF_money_market = [['','','','','','','','',''] for i in range(len(self
        temp_discount_factor = np.zeros(len_MM)
        extract_date_list = extract_1d_list(self._ir_list, 0)
        len_ir_list = len(self._ir_list)
```

```
len_one_list_with_DF = len(ir_list_DF_money_market[0])
index_DF  = len_one_list_with_DF - 1

for i in range(len_MM):
    TN_flag = self._ir_list[i][0] in 'T/N'

if (TN_flag == True):
    # O/N
    index_ON = extract_date_list.index('O/N')
    temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction
                                                * float(self._ir_list[i
    # T/N
    index_TN = extract_date_list.index('T/N')
    temp_discount_factor[index_TN] = temp_discount_factor[index_ON] / \
                                                                    (1.0 + self
                                                                     * float(se
    # libor
    for i in range(2, len_MM):
        temp_discount_factor[i] = temp_discount_factor[index_TN] \
                                                / (1.0 + self._calc_day_cou

    for i in range(len_MM):
        for j in range(len_one_list_with_DF - 1):
            ir_list_DF_money_market[i][j] =self._ir_list[i][j]
        ir_list_DF_money_market[i][index_DF] = temp_discount_factor[i]
    for i in range(len_MM, len_ir_list):
        for j in range(len_one_list_with_DF - 1):
            ir_list_DF_money_market[i][j] = self._ir_list[i][j]
        ir_list_DF_money_market[i][index_DF] = 0.0

elif (TN_flag == False):
    # O/N
    index_ON = extract_date_list.index('O/N')
    temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction
                                                                    * float(se
    # libor
    for i in range(1, len_MM):
        temp_discount_factor[i] = temp_discount_factor[index_ON] * temp_discoun
                                                / (1.0 + self._calc_day_cou

    for i in range(len_MM):
        for j in range(len_one_list_with_DF - 1):
            ir_list_DF_money_market[i][j] =self._ir_list[i][j]
        ir_list_DF_money_market[i][index_DF] = temp_discount_factor[i]
    for i in range(len_MM, len_ir_list):
        for j in range(len_one_list_with_DF - 1):
            ir_list_DF_money_market[i][j] = self._ir_list[i][j]
        ir_list_DF_money_market[i][index_DF] = 0.0
```

```
        return ir_list_DF_money_market

    def interpolate_DF_money_market(self):
        ir_list_DF_money_market = self._calc_DF_money_market()
        extract_date_list = extract_1d_list(ir_list_DF_money_market, 0)
        extract_DF_list = extract_1d_list(ir_list_DF_money_market, 8)
        index_1m = extract_date_list.index('1.0M')
        index_1y = extract_date_list.index('1.0Y')
        extract_date_list_mm_tenor = extract_date_list[index_1m : index_1y + 1]
        extract_date_list_mm_tenor[-1] = '12.0M'
        for i in range(len(extract_date_list_mm_tenor)):
            extract_date_list_mm_tenor[i] = float(extract_date_list_mm_tenor[i][0:-3])
        extract_DF_list_mm_tenor = extract_DF_list[index_1m : index_1y + 1]
        f_interpolated_DF_money_market = interp1d(extract_date_list_mm_tenor, extract_D
        return f_interpolated_DF_money_market

    def interpolated_ir_list_DF_money_market(self):
        ir_list_DF_mm = self._calc_DF_money_market()
        extract_date_list = extract_1d_list(ir_list_DF_mm, 0)
        len_one_list_with_DF = len(ir_list_DF_mm[0])
        len_ir_list = len(ir_list_DF_mm)
        index_1m = extract_date_list.index('1.0M')
        index_1y = extract_date_list.index('1.0Y')
        len_1m = index_1m +1
        len_1y = index_1y + 1
        len_interpolated_ir_list_DF_mm = (len_1m - 1) + 12 + (len_ir_list - len_1y)
        interpolated_ir_list_DF_mm = [['', '', '', '', '', '', '', '', ''] for i in ran
        f_interpolated_DF_money_market = self.interpolate_DF_money_market()
        # 1.0Mの前までコピー
        for i in range(0, len_1m - 1):
            for j in range(len_one_list_with_DF):
                interpolated_ir_list_DF_mm[i][j] = ir_list_DF_mm[i][j]
        # 1.0M 12.0Mまで DFを補完
        for i in range(len_1m - 1, len_1m - 1 + 12 - 1):
            interpolated_ir_list_DF_mm[i][0] = "{}M".format(float((i - (len_1m - 1) + 1
        for i in range(len_1m - 1, len_1m - 1 + 12 - 1):
            interpolated_ir_list_DF_mm[i][1] = ir_list_DF_mm[i][1]
            interpolated_ir_list_DF_mm[i][2] = ir_list_DF_mm[i][2]
            interpolated_ir_list_DF_mm[i][3] = ''
            interpolated_ir_list_DF_mm[i][4] = ir_list_DF_mm[i][4]
            interpolated_ir_list_DF_mm[i][5] = ir_list_DF_mm[i][5]
            interpolated_ir_list_DF_mm[i][6] = self._spot_date
            interpolated_ir_list_DF_mm[i][7] = self._calc_end_date(self._spot_date, int
            interpolated_ir_list_DF_mm[i][8] = float(f_interpolated_DF_money_market(flo
        for i in range( len_1m - 1 + 12 - 1 , len_interpolated_ir_list_DF_mm):
            for j in range(len_one_list_with_DF):
                interpolated_ir_list_DF_mm[i][j] = ir_list_DF_mm[i - index_1y -1][j]
```

```
        return interpolated_ir_list_DF_mm

    def _interpolate_swap_rate(self):
        extract_date_list = extract_1d_list(self._ir_list, 0)
        extract_rate_list = extract_1d_list(self._ir_list, 3)
        index_1y = extract_date_list.index('1.0Y')
        extract_date_list_swap_tenor = extract_date_list[index_1y:]
        for i in range(len(extract_date_list_swap_tenor)):
            extract_date_list_swap_tenor[i] = calc_month(extract_date_list_swap_tenor[i
        extract_swap_rate_list = extract_rate_list[index_1y:]
        f_interpolated_swap_rate = interp1d(extract_date_list_swap_tenor , extract_swap
        return f_interpolated_swap_rate

    def _interpolated_ir_list_for_bootstrap(self):
        ir_list_for_bootstrap = self.interpolated_ir_list_DF_money_market()
        extract_date_list = extract_1d_list(ir_list_for_bootstrap, 0)
        #
        index_1y = extract_date_list.index('1.0Y')
        len_index_1y = index_1y + 1
        f_interpolated_swap_rate = self._interpolate_swap_rate()
        max_maturity_in_unit_month = calc_month(ir_list_for_bootstrap[-1][0])
        seq_len_for_bootstrap = int( ((max_maturity_in_unit_month) - 12.0)  / self._rol
        seq_for_bootstrap = [['', '', '', '', '', '', '', '', ''] for i in range(seq_le
        # base_tenor はスワップレートのテナーでもっとも短いテナーという気持ち
        base_tenor = int(float(ir_list_for_bootstrap[index_1y][0][0:-1]) * 12.0)  # cha
        for i in range(index_1y , seq_len_for_bootstrap + index_1y):
            seq_for_bootstrap[i][0] = "{}M".format(base_tenor + (i - index_1y)  * self.
        for i in range(index_1y):
            seq_for_bootstrap[i] = ir_list_for_bootstrap[i]
        for i in range(index_1y, seq_len_for_bootstrap + index_1y):
            seq_for_bootstrap[i][1] = self._string_swap
            seq_for_bootstrap[i][2] = self._ccy
            seq_for_bootstrap[i][3] = float(f_interpolated_swap_rate(float(seq_for_boot
            seq_for_bootstrap[i][4] = self._str_roll_month
            seq_for_bootstrap[i][5] = self._str_convention
            seq_for_bootstrap[i][6] = self._spot_date
            seq_for_bootstrap[i][7] = self._calc_end_date(self._spot_date, seq_for_boot
        # add discount factor for 12.0M calculated by calc_DF_money_market.
        seq_for_bootstrap[index_1y][8] = ir_list_for_bootstrap[index_1y][8]
        return seq_for_bootstrap
#TODO complete interpolated_ir_list_for_bootstrap

    def _calc_annuity(self, ir_list, target_tenor):
        extract_date_list = extract_1d_list(ir_list, 0)
        index_target_tenor = extract_date_list.index(target_tenor)
        index_roll_tenor = extract_date_list.index(self._str_roll_month)
        annuity = 0
        day_count_fraction = self._calc_day_count_fraction(ir_list[index_roll_tenor][6]
```

```
        for i in range(index_roll_tenor, index_target_tenor):
            annuity += ir_list[i][8] * day_count_fraction
        return annuity

    def _calc_DF_swap_rate(self):
        interpolated_ir_list = self._interpolated_ir_list_for_bootstrap()
        extract_date_list = extract_1d_list(interpolated_ir_list, 0)
        index_1y = extract_date_list.index('12.0M')
        index_roll_tenor = extract_date_list.index(self._str_roll_month)
        index_start_tenor = index_1y + 1
        index_end_tenor = len(interpolated_ir_list)
        day_count_fraction = self._calc_day_count_fraction(self._ir_list[index_roll_ten
        DF_swap_rate = np.zeros(len(interpolated_ir_list))
        for i in range(index_start_tenor, index_end_tenor):
            annuity = self._calc_annuity(interpolated_ir_list, interpolated_ir_list[i][
            swap_rate = interpolated_ir_list[i][3]
            DF_swap_rate[i] = 1.0 / (1.0 + day_count_fraction * swap_rate) * (1.0 - swa
            interpolated_ir_list[i][8] = DF_swap_rate[i]
        return interpolated_ir_list

    def _interpolate_DF(self):
        # make list including days between start_day and end_day in fourth column.
        # DF_list = [0:tenor_name, 1:market_name, 2:ccy, 3:rate, 4:roll_month, 5:convet
        DF_list = self._calc_DF_swap_rate()
        len_DF_list = len(DF_list)
        # interpolated_DF_list [0: tenor_name, 1:days, 2:DF]
        interpolated_DF_list = [["", 0.0, 0.0] for i in range(len_DF_list)]
        for i in range(len_DF_list):
            interpolated_DF_list[i][0] = DF_list[i][0]
            interpolated_DF_list[i][2] = DF_list[i][8]
        # calc days from %Y/%m/%d
        for i in range(len_DF_list):
            if (interpolated_DF_list[i][0] == 'O/N'):
                interpolated_DF_list[i][1] = calc_trade_days(DF_list[i][6], DF_list[i][
                # TODO going to revise 1 and 2 day-count. have to consider Sat., Sun. a
            elif (interpolated_DF_list[i][0] == 'T/N'):
                interpolated_DF_list[i][1] = calc_trade_days(DF_list[i][6], DF_list[i][
            else:
                interpolated_DF_list[i][1] = calc_trade_days(DF_list[i][6], DF_list[i][
        # interpolate DF
        index_trade_days = 1
        index_DF = 2
        f_interpolation_DF = interpolation_extract_list(interpolated_DF_list, index_tra
        return f_interpolation_DF

    def _calc_end_date_input_days(self, days_from_base_date):
        obj_start_date = datetime.datetime.strptime(self._base_date, '%Y/%m/%d')
        end_date = (obj_start_date + datetime.timedelta(days=days_from_base_date)).strf
```

```
            return end_date

        def get_DF(self, date):
            DF_list = self._calc_DF_swap_rate()
            max_maturity_days = calc_trade_days(DF_list[-1][6], DF_list[-1][7]) + 2
            print(max_maturity_days)
            interpolated_DF_list = [[i, '', 0.0] for i in range(max_maturity_days + 1)]
            f_interpolation_DF = self._interpolate_DF()
            for i in range(1, max_maturity_days + 1):
                interpolated_DF_list[i][1] = self._calc_end_date_input_days(interpolated_DF
                interpolated_DF_list[i][2] = float(f_interpolation_DF(interpolated_DF_list[
            # today's DF betauchi
            interpolated_DF_list[0][0] = 0
            interpolated_DF_list[0][1] = self._calc_end_date_input_days(interpolated_DF_lis
            interpolated_DF_list[0][2] = 1.0
            extract_end_date_list = extract_1d_list(interpolated_DF_list, 1)
            index_target_date = extract_end_date_list.index(date)
            return interpolated_DF_list[index_target_date][2]

        def get_DF_list(self):
            DF_list = self._calc_DF_swap_rate()
            max_maturity_days = calc_trade_days(DF_list[-1][6], DF_list[-1][7]) + 2
            interpolated_DF_list = [[i, '', 0.0] for i in range(max_maturity_days + 1)]
            f_interpolation_DF = self._interpolate_DF()
            for i in range(1, max_maturity_days + 1):
                interpolated_DF_list[i][1] = self._calc_end_date_input_days(interpolated_DF
                interpolated_DF_list[i][2] = float(f_interpolation_DF(interpolated_DF_list[
            # today's DF betauchi
            interpolated_DF_list[0][0] = 0
            interpolated_DF_list[0][1] = self._calc_end_date_input_days(interpolated_DF_lis
            interpolated_DF_list[0][2] = 1.0
            return interpolated_DF_list
```

In [103]:
```
jpy_IR_obj = discount_factor('20180118_IR_data.csv', 'JPY')
ir_list_jpy = jpy_IR_obj.get_ir_list()
ir_list_jpy
DF_list_jpy = jpy_IR_obj.get_DF_list()
usd_IR_obj = discount_factor('20180118_IR_data.csv', 'USD')
ir_list_usd = usd_IR_obj.get_ir_list()
ir_list_usd
DF_list_usd = usd_IR_obj.get_DF_list()
#jpy_IR_obj.interpolated_ir_list_DF_money_market()
#jpy_IR_obj.get_ir_list_with_DF_swap_rate()
#jpy_IR_obj.get_ir_list_interpolated_swap_rate()
#f = jpy_IR_obj.interpolate_swap_rate()
#f(12)
```

In [104]:
```
import csv
```

```
        with open('DF_list_jpy.csv', 'w') as f:
            writer = csv.writer(f, lineterminator='\n') # 改行コード（\n）を指定しておく
            writer.writerows(DF_list_jpy) # 2次元配列も書き込める
```

```
In [18]: test_L = [500,0,0,0,0,0,0,500,200]
         index_num = [n for n, v in enumerate(test_L) if v == 500]
         print(index_num)
```

```
[0, 7]
```

```
In [21]: index_num = [i for i, v in enumerate(test_L) if v ==]
         index_num
```

```
Out[21]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [6]: usd_IR_obj.get_ir_list_with_DF_swap_rate()
```

```
Out[6]: [['O/N',
          'Money Market',
          'USD',
          '0.014375',
          'None',
          'act/365',
          '2018/01/18',
          '2018/01/19',
          0.99996061798935998],
         ['1.0W',
          'Money Market',
          'USD',
          '0.0146533',
          'None',
          'act/365',
          '2018/01/20',
          '2018/01/27',
          0.99964031641695072],
         ['1.0M',
          'Money Market',
          'USD',
          '0.0156118',
          'None',
          'act/365',
          '2018/01/20',
          '2018/02/19',
          0.99863982121507266],
         ['2.0M',
          'Money Market',
          'USD',
          '0.0163482',
```

```
 'None',
 'act/365',
 '2018/01/20',
 '2018/03/21',
 0.99724127591428158],
['3.0M',
 'Money Market',
 'USD',
 '0.017447',
 'None',
 'act/365',
 '2018/01/20',
 '2018/04/20',
 0.99563800284143877],
['6.0M',
 'Money Market',
 'USD',
 '0.019255',
 'None',
 'act/365',
 '2018/01/20',
 '2018/07/19',
 0.99051568055097328],
['12.0M',
 'Swap',
 'USD',
 0.02045,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2019/01/15',
 0.97988263759092809],
['18.0M',
 'Swap',
 'USD',
 0.02151,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2019/07/14',
 0.96882172015268786],
['24.0M',
 'Swap',
 'USD',
 0.02257,
 '6.0M',
 'act/365',
 '2018/01/20',
```

```
 '2020/01/10',
 0.95663750451122498],
['30.0M',
 'Swap',
 'USD',
 0.023115,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2020/07/08',
 0.9448202517809936],
['36.0M',
 'Swap',
 'USD',
 0.02366,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2021/01/04',
 0.93263727610124303],
['42.0M',
 'Swap',
 'USD',
 0.023965,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2021/07/03',
 0.92088555279582884],
['48.0M',
 'Swap',
 'USD',
 0.02427,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2021/12/30',
 0.90899907343571795],
['54.0M',
 'Swap',
 'USD',
 0.024475,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2022/06/28',
 0.89739893893085165],
['60.0M',
```

```
 'Swap',
 'USD',
 0.02468,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2022/12/25',
 0.88575902663066808],
['66.0M',
 'Swap',
 'USD',
 0.02486,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2023/06/23',
 0.87420827274804536],
['72.0M',
 'Swap',
 'USD',
 0.02504,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2023/12/20',
 0.86264510466030198],
['78.0M',
 'Swap',
 'USD',
 0.0252,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2024/06/17',
 0.85118936916311649],
['84.0M',
 'Swap',
 'USD',
 0.02536,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2024/12/14',
 0.83974246739071678],
['90.0M',
 'Swap',
 'USD',
 0.025505,
```

```
 '6.0M',
 'act/365',
 '2018/01/20',
 '2025/06/12',
 0.82840662870361925],
['96.0M',
 'Swap',
 'USD',
 0.02565,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2025/12/09',
 0.8170953958685252],
['102.0M',
 'Swap',
 'USD',
 0.025779999999999997,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2026/06/07',
 0.80592236076286017],
['108.0M',
 'Swap',
 'USD',
 0.02591,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2026/12/04',
 0.79478825734226177],
['114.0M',
 'Swap',
 'USD',
 0.026029999999999998,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2027/06/02',
 0.78377672001846044],
['120.0M',
 'Swap',
 'USD',
 0.02615,
 '6.0M',
 'act/365',
 '2018/01/20',
```

```
 '2027/11/29',
 0.77281379457346666],
['126.0M',
 'Swap',
 'USD',
 0.026225,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2028/05/27',
 0.76230343258721178],
['132.0M',
 'Swap',
 'USD',
 0.0263,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2028/11/23',
 0.75187197560012642],
['138.0M',
 'Swap',
 'USD',
 0.026375,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2029/05/22',
 0.74151955308576945],
['144.0M',
 'Swap',
 'USD',
 0.02645,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2029/11/18',
 0.73124628130839531],
['150.0M',
 'Swap',
 'USD',
 0.026525,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2030/05/17',
 0.72105226347229001],
['156.0M',
```

```
 'Swap',
 'USD',
 0.0266,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2030/11/13',
 0.71093758987203215],
['162.0M',
 'Swap',
 'USD',
 0.026675,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2031/05/12',
 0.70090233804363056],
['168.0M',
 'Swap',
 'USD',
 0.02675,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2031/11/08',
 0.6909465729164892],
['174.0M',
 'Swap',
 'USD',
 0.026825,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2032/05/06',
 0.68107034696614899],
['180.0M',
 'Swap',
 'USD',
 0.0269,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2032/11/02',
 0.67127370036775769],
['186.0M',
 'Swap',
 'USD',
 0.026932,
```

```
 '6.0M',
 'act/365',
 '2018/01/20',
 '2033/05/01',
 0.66208909155087248],
['192.0M',
 'Swap',
 'USD',
 0.026964000000000002,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2033/10/28',
 0.65300438827564655],
['198.0M',
 'Swap',
 'USD',
 0.026996,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2034/04/26',
 0.64401870240611714],
['204.0M',
 'Swap',
 'USD',
 0.027028,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2034/10/23',
 0.63513115132037501],
['210.0M',
 'Swap',
 'USD',
 0.02706,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2035/04/21',
 0.62634085790667748],
['216.0M',
 'Swap',
 'USD',
 0.027092,
 '6.0M',
 'act/365',
 '2018/01/20',
```

```
 '2035/10/18',
 0.61764695055909458],
['222.0M',
 'Swap',
 'USD',
 0.027124000000000002,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2036/04/15',
 0.6090485631727004],
['228.0M',
 'Swap',
 'USD',
 0.027156,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2036/10/12',
 0.60054483513831336],
['234.0M',
 'Swap',
 'USD',
 0.027188,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2037/04/10',
 0.5921349113367923],
['240.0M',
 'Swap',
 'USD',
 0.02722,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2037/10/07',
 0.58381794213289517],
['246.0M',
 'Swap',
 'USD',
 0.0272165,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2038/04/05',
 0.57613861780263997],
['252.0M',
```

```
 'Swap',
 'USD',
 0.027213,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2038/10/02',
 0.56856294864477175],
['258.0M',
 'Swap',
 'USD',
 0.0272095,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2039/03/31',
 0.56108952348270025],
['264.0M',
 'Swap',
 'USD',
 0.027206,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2039/09/27',
 0.5537169505190086],
['270.0M',
 'Swap',
 'USD',
 0.0272025,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2040/03/25',
 0.54644385706707121],
['276.0M',
 'Swap',
 'USD',
 0.027199,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2040/09/21',
 0.53926888928642103],
['282.0M',
 'Swap',
 'USD',
 0.0271955,
```

```
 '6.0M',
 'act/365',
 '2018/01/20',
 '2041/03/20',
 0.53219071192180922],
['288.0M',
 'Swap',
 'USD',
 0.027192,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2041/09/16',
 0.52520800804590917],
['294.0M',
 'Swap',
 'USD',
 0.0271885,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2042/03/15',
 0.51831947880561224],
['300.0M',
 'Swap',
 'USD',
 0.027185,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2042/09/11',
 0.51152384317186295],
['306.0M',
 'Swap',
 'USD',
 0.0271815,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2043/03/10',
 0.50481983769298699],
['312.0M',
 'Swap',
 'USD',
 0.027178,
 '6.0M',
 'act/365',
 '2018/01/20',
```

```
 '2043/09/06',
 0.49820621625145972],
['318.0M',
 'Swap',
 'USD',
 0.0271745,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2044/03/04',
 0.49168174982406981],
['324.0M',
 'Swap',
 'USD',
 0.027171,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2044/08/31',
 0.48524522624542699],
['330.0M',
 'Swap',
 'USD',
 0.0271675,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2045/02/27',
 0.47889544997477052],
['336.0M',
 'Swap',
 'USD',
 0.027164,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2045/08/26',
 0.47263124186602856],
['342.0M',
 'Swap',
 'USD',
 0.0271605,
 '6.0M',
 'act/365',
 '2018/01/20',
 '2046/02/22',
 0.46645143894108515],
['348.0M',
```

```
           'Swap',
           'USD',
           0.027157,
           '6.0M',
           'act/365',
           '2018/01/20',
           '2046/08/21',
           0.46035489416620845],
          ['354.0M',
           'Swap',
           'USD',
           0.0271535,
           '6.0M',
           'act/365',
           '2018/01/20',
           '2047/02/17',
           0.4543404762315959],
          ['360.0M',
           'Swap',
           'USD',
           0.02715,
           '6.0M',
           'act/365',
           '2018/01/20',
           '2047/08/16',
           0.44840706933399332]]

In [63]: a = [[1,2], [2,3]]
         b = [[3,4], [25,36]]
         a + b

Out[63]: [[1, 2], [2, 3], [3, 4], [25, 36]]

In [81]: 29 * 12 / 3

Out[81]: 116.0

In [327]: usd_IR_obj.get_ir_list()

Out[327]: [['O/N',
            'Money Market',
            'USD',
            '0.014375',
            'None',
            'act/365',
            '2018/01/18',
            '2018/01/19'],
           ['1.0W',
            'Money Market',
```

```
 'USD',
 '0.0146533',
 'None',
 'act/365',
 '2018/01/20',
 '2018/01/25'],
['1.0M',
 'Money Market',
 'USD',
 '0.0156118',
 'None',
 'act/365',
 '2018/01/20',
 '2018/02/17'],
['2.0M',
 'Money Market',
 'USD',
 '0.0163482',
 'None',
 'act/365',
 '2018/01/20',
 '2018/03/19'],
['3.0M',
 'Money Market',
 'USD',
 '0.017447',
 'None',
 'act/365',
 '2018/01/20',
 '2018/04/18'],
['6.0M',
 'Money Market',
 'USD',
 '0.019255',
 'None',
 'act/365',
 '2018/01/20',
 '2018/07/17'],
['1.0Y',
 'Money Market',
 'USD',
 '0.02045',
 '6M',
 'act/365',
 '2018/01/20',
 '2019/01/18'],
['2.0Y',
 'Swap',
```

```
 'USD',
 '0.02257',
 '6M',
 'act/365',
 '2018/01/20',
 '2020/01/18'],
['3.0Y',
 'Swap',
 'USD',
 '0.02366',
 '6M',
 'act/365',
 '2018/01/20',
 '2021/01/17'],
['4.0Y',
 'Swap',
 'USD',
 '0.02427',
 '6M',
 'act/365',
 '2018/01/20',
 '2022/01/17'],
['5.0Y',
 'Swap',
 'USD',
 '0.02468',
 '6M',
 'act/365',
 '2018/01/20',
 '2023/01/17'],
['6.0Y',
 'Swap',
 'USD',
 '0.02504',
 '6M',
 'act/365',
 '2018/01/20',
 '2024/01/17'],
['7.0Y',
 'Swap',
 'USD',
 '0.02536',
 '6M',
 'act/365',
 '2018/01/20',
 '2025/01/16'],
['8.0Y',
 'Swap',
```

```
 'USD',
 '0.02565',
 '6M',
 'act/365',
 '2018/01/20',
 '2026/01/16'],
['9.0Y',
 'Swap',
 'USD',
 '0.02591',
 '6M',
 'act/365',
 '2018/01/20',
 '2027/01/16'],
['10.0Y',
 'Swap',
 'USD',
 '0.02615',
 '6M',
 'act/365',
 '2018/01/20',
 '2028/01/16'],
['15.0Y',
 'Swap',
 'USD',
 '0.0269',
 '6M',
 'act/365',
 '2018/01/20',
 '2033/01/14'],
['20.0Y',
 'Swap',
 'USD',
 '0.02722',
 '6M',
 'act/365',
 '2018/01/20',
 '2038/01/13'],
['30.0Y',
 'Swap',
 'USD',
 '0.02715',
 '6M',
 'act/365',
 '2018/01/20',
 '2048/01/11']]
```

In [37]:
```
def calc_end_date(start_day, str_maturity):
```

```
                datetime_obj_start = datetime.datetime.strptime(start_day, '%Y/%m/%d')
                unit = str_maturity[-1]
                int_num = int(str_maturity[0:len(str_maturity)-3])
                if (unit == 'D'):
                    trade_days = int_num
                elif (unit == 'W'):
                    trade_days = int_num * 7
                elif (unit == 'M'):
                    trade_days = int_num * 30
                elif (unit == 'Y'):
                    trade_days = int_num * 365
                print(int_num)
                print(trade_days)
                end_day = datetime_obj_start + datetime.timedelta(days=trade_days)
                return end_day.strftime('%Y/%m/%d')
```

```
In [328]:     def get_ir_list(ir_list_name):
                  with open(ir_list_name, 'r') as csvfile:
                      reader_obj = csv.reader(csvfile)
                      # rewritten header_obj by using next method(???)
                      header_obj = next(reader_obj)
                      ir_list = []
                      for row in reader_obj:
                          ir_list.append(row)
                      temp_num = [[] for i in range(len(ir_list))] # comprehension expression fo
                      for i in range(len(ir_list)):
                          if (ir_list[i][0][0].isdigit()):
                              num_tenor = ir_list[i][0][0: len(ir_list[i][0])-1]
                              unit_tenor = ir_list[i][0][-1]
                              temp_num[i] = "{:.1f}".format(int(num_tenor))
                              ir_list[i][0] = temp_num[i] + unit_tenor
                  return ir_list
```

```
In [36]: get_ir_list('20180118_IR.csv')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-36-4737ea16428a> in <module>()
----> 1 get_ir_list('20180118_IR.csv')


NameError: name 'get_ir_list' is not defined
```

```
In [55]: import csv
```

```
        with open('20180118_IR.csv', 'r') as csvfile:
            reader_obj = csv.reader(csvfile)
            # rewritten header_obj by using next method(???)
            header_obj = next(reader_obj)
            ir_list = []
            for row in reader_obj:
                ir_list.append(row)
            temp_num = [[] for i in range(len(ir_list))] # comprehension expression for making
            for i in range(len(ir_list)):
                if (ir_list[i][0][0].isdigit()):
                    num_tenor = ir_list[i][0][0: len(ir_list[i][0])-1]
                    unit_tenor = ir_list[i][0][-1]
                    temp_num[i] = "{:.1f}".format(int(num_tenor))
                    ir_list[i][0] = temp_num[i] + unit_tenor

        ir_list
```

```
Out[55]: [['O/N', 'Money Market', 'USD', '0.014375', ''],
          ['1.0W', 'Money Market', 'USD', '0.0146533', ''],
          ['1.0M', 'Money Market', 'USD', '0.0156118', ''],
          ['2.0M', 'Money Market', 'USD', '0.0163482', ''],
          ['3.0M', 'Money Market', 'USD', '0.017447', ''],
          ['6.0M', 'Money Market', 'USD', '0.019255', ''],
          ['1.0Y', 'Swap', 'USD', '0.02045', '6M'],
          ['2.0Y', 'Swap', 'USD', '0.02257', '6M'],
          ['3.0Y', 'Swap', 'USD', '0.02366', '6M'],
          ['4.0Y', 'Swap', 'USD', '0.02427', '6M'],
          ['5.0Y', 'Swap', 'USD', '0.02468', '6M'],
          ['6.0Y', 'Swap', 'USD', '0.02504', '6M'],
          ['7.0Y', 'Swap', 'USD', '0.02536', '6M'],
          ['8.0Y', 'Swap', 'USD', '0.02565', '6M'],
          ['9.0Y', 'Swap', 'USD', '0.02591', '6M'],
          ['10.0Y', 'Swap', 'USD', '0.02615', '6M'],
          ['15.0Y', 'Swap', 'USD', '0.0269', '6M'],
          ['20.0Y', 'Swap', 'USD', '0.02722', '6M'],
          ['30.0Y', 'Swap', 'USD', '0.02715', '6M']]
```

```
In [85]: def make_empty_list(len_list):


          File "<ipython-input-85-153a4cb7ab59>", line 2

        ^
    SyntaxError: unexpected EOF while parsing
```

```
In [118]: a = [0, 1, 2, 3, 4, 5, 6]
          ind = a.index(6)
```

```
        for i in range(ind):
            print(i)
```

```
0
1
2
3
4
5
```

```
In [66]: 119 * 3
```

```
Out[66]: 357
```

```
In [28]: # プライベート変数を_calc_DF_money_marketの中でつかうと，  self._ir_listが更新されて，  lis
             def _calc_DF_money_market(self, ir_list):
                 len_MM = 0
                 for i in range(len(ir_list)):
                     if (ir_list[i][1] == 'Money Market'):
                         len_MM += 1
                 temp_discount_factor = np.zeros(len_MM)
                 extract_date_list = extract_1d_list(ir_list, 0)
                 for i in range(len_MM):
                     TN_flag = ir_list[i][0] in 'T/N'
                 if (TN_flag == True):
                     # O/N
                     index_ON = extract_date_list.index('O/N')
                     temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction
                                                             * float(ir_list[index_O
                     # T/N
                     index_TN = extract_date_list.index('T/N')
                     temp_discount_factor[index_TN] = temp_discount_factor[index_ON] / \
                                                                          (1.0 + self
                                                                           * float(ir
                     # libor
                     for i in range(2, len_MM):
                         temp_discount_factor[i] = temp_discount_factor[index_TN] \
                                                               / (1.0 + self._calc_day_cou

                     for i in range(len_MM):
                         ir_list[i].append(temp_discount_factor[i])

                 if (TN_flag == False):
                     # O/N
                     index_ON = extract_date_list.index('O/N')
                     temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction
                                                                          * float(ir
                     # libor
```

```
                    for i in range(1, len_MM):
                        temp_discount_factor[i] = temp_discount_factor[index_ON] * temp_discoun
                                                                / (1.0 + self._calc_day_cou
                    for i in range(len_MM):
                        ir_list[i].append(temp_discount_factor[i])

                return ir_list


            def interpolated_ir_list_for_bootstrap(self):
                extract_date_list = extract_1d_list(self._ir_list, 0)
                index_1y = extract_date_list.index('1.0Y')
        #         ir_list_for_bootstrap = self._calc_DF_money_market()
                ir_list_for_bootstrap = self.calc_DF_money_market()
                max_maturity_in_unit_month = calc_month(ir_list_for_bootstrap[-1][0])
                seq_len_for_bootstrap = int(max_maturity_in_unit_month / self._roll_month - 1)
                seq_for_bootstrap = [['', '', '', '', '', '', '', '',''] for i in range(seq_len
                # base_tenor はスワップレートのテナーでもっとも短いテナーという気持ち
                base_tenor = int(float(self._ir_list[index_1y][0][0:-1]) * 12.0)  # change unit
                for i in range(index_1y , seq_len_for_bootstrap + index_1y):
                    seq_for_bootstrap[i][0] = "{}M".format(base_tenor + (i - index_1y)  * self.
                for i in range(index_1y):
                    seq_for_bootstrap[i] = self._ir_list[i]
                return seq_for_bootstrap
```

```
Out[28]: "    def _calc_DF_money_market(self, ir_list):\n        len_MM = 0\n        for i in ra
```

```
In [ ]: def calc_DF_money_market(self):
            len_MM = 0
            for i in range(len(ir_list)):
                if (ir_list[i][1] == 'Money Market'):
                    len_MM += 1
            ir_list_DF_money_market = [['','','','','','','','',''] for i in range(len(self.
            temp_discount_factor = np.zeros(len_MM)
            extract_date_list = extract_1d_list(self._ir_list, 0)
            len_original = len(self._ir_list[0])

            for i in range(len_MM):
                TN_flag = self._ir_list[i][0] in 'T/N'

            if (TN_flag == True):
                # O/N
                index_ON = extract_date_list.index('O/N')
                temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction(
                                                                * float(self._ir_list[in
                # T/N
                index_TN = extract_date_list.index('T/N')
                temp_discount_factor[index_TN] = temp_discount_factor[index_ON] / \
```

```
                                                                                (1.0 + self.
                                                                                 * float(sel
                # libor
                for i in range(2, len_MM):
                    temp_discount_factor[i] = temp_discount_factor[index_TN] \
                                                                / (1.0 + self._calc_day_coun

                for i in range(len_MM):
                    for j in range(len_original ):
                        ir_list_DF_money_market[i][j] = self._ir_list[i][j]
                        ir_list_DF_money_market[i][len_original] = temp_discount_factor[i]
                for i in range(len_MM, len(ir_list)):
                    ir_list_DF_money_market[i] = self._ir_list[i]

            elif (TN_flag == False):
                # O/N
                index_ON = extract_date_list.index('O/N')
                temp_discount_factor[index_ON] = 1.0 / (1.0 + self._calc_day_count_fraction(
                                                                                * float(sel
                # libor
                for i in range(1, len_MM):
                    temp_discount_factor[i] = temp_discount_factor[index_ON] * temp_discount
                                                                / (1.0 + self._calc_day_coun

                for i in range(len_MM):
                    for j in range(len_original):
                        ir_list_DF_money_market[i][j] =self._ir_list[i][j]
                        ir_list_DF_money_market[i][len_original] = temp_discount_factor[i]
                for i in range(len_MM, len(ir_list)):
                    ir_list_DF_money_market[i] = self._ir_list[i]

            return ir_list_DF_money_market
```

0.1 1/28 バックアップ置き場

- ver1.1 ccy を選べるようにする

In [ ]:
```
    def _load_ir_list(self, ir_list_name):
        with open(ir_list_name, 'r') as csvfile:
            reader_obj = csv.reader(csvfile)
            # rewritten header_obj by using next method(???)
            header_obj = next(reader_obj)
            ir_list = []
            for row in reader_obj:
                ir_list.append(row)
            # change int type to float type (ex. 1Y -> 1.0Y)
            temp_num = [[] for i in range(len(ir_list))] # comprehension expression for
            for i in range(len(ir_list)):
```

```
                if (ir_list[i][0][0].isdigit()):
                    num_tenor = ir_list[i][0][0: len(ir_list[i][0])-1]
                    unit_tenor = ir_list[i][0][-1]
                    temp_num[i] = "{:.1f}".format(int(num_tenor))
                    ir_list[i][0] = temp_num[i] + unit_tenor
        ir_list_with_cf = self._add_cash_flow(ir_list)
        return ir_list_with_cf

    def interpolated_ir_list_DF_money_market(self):
        ir_list_DF_mm = self._calc_DF_money_market()
        extract_date_list = extract_1d_list(ir_list_DF_mm, 0)
        len_one_list_with_DF = len(ir_list_DF_mm[0])
        len_ir_list = len(ir_list_DF_mm)
        index_1m = extract_date_list.index('1.0M')
        index_1y = extract_date_list.index('1.0Y')
        len_1m = index_1m +1
        len_1y = index_1y + 1
        len_interpolated_ir_list_DF_mm = (len_1m - 1) + 12 + (len_ir_list - len_1y)
        interpolated_ir_list_DF_mm = [['', '', '', '', '', '', '', '', ''] for i in rang
        f_interpolated_DF_money_market = self.interpolate_DF_money_market()
        # 1.0Mの前までコピー
        for i in range(0, len_1m - 1):
            for j in range(len_one_list_with_DF):
                interpolated_ir_list_DF_mm[i][j] = ir_list_DF_mm[i][j]
        # 1.0M 12.0Mまで DFを補完
        for i in range(len_1m - 1, len_1m - 1 + 12):
            interpolated_ir_list_DF_mm[i][0] = "{}M".format(float((i - (len_1m - 1) + 1)
        for i in range(len_1m - 1, len_1m - 1 + 12):
            interpolated_ir_list_DF_mm[i][1] = ir_list_DF_mm[i][1]
            interpolated_ir_list_DF_mm[i][2] = ir_list_DF_mm[i][2]
            interpolated_ir_list_DF_mm[i][3] = ''
            interpolated_ir_list_DF_mm[i][4] = ir_list_DF_mm[i][4]
            interpolated_ir_list_DF_mm[i][5] = ir_list_DF_mm[i][5]
            interpolated_ir_list_DF_mm[i][6] = self._spot_date
            interpolated_ir_list_DF_mm[i][7] = self._calc_end_date(self._spot_date, inte
            interpolated_ir_list_DF_mm[i][8] = float(f_interpolated_DF_money_market(floa
        for i in range( len_1m - 1 + 12 , len_interpolated_ir_list_DF_mm):
            for j in range(len_one_list_with_DF):
                interpolated_ir_list_DF_mm[i][j] = ir_list_DF_mm[i - index_1y -1][j]
        return interpolated_ir_list_DF_mm
```