discount_factor

January 28, 2018

In [210]:

```python
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):
        ## 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．
        print("call constructor")
        # base_date -> trade_date??
        self._base_date = ir_list_name[0:4] + '/' + ir_list_name[4:6] + '/' + ir_list_
        # 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._load_ir_list(ir_list_name)
        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 _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)
            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
        ir_list_with_cf = self._add_cash_flow(ir_list)
        return ir_list_with_cf

    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/%
        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._conventi
        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(self._ir_list)
#        return self._ir_list_DF_mm
        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 _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(sel
        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_fractio
                                                                * float(self._ir_list[
            # T/N
            index_TN = extract_date_list.index('T/N')
            temp_discount_factor[index_TN] = temp_discount_factor[index_ON] / \
                                                                            (1.0 + sel
                                                                             * float(s
            # libor
            for i in range(2, len_MM):
                temp_discount_factor[i] = temp_discount_factor[index_TN] \
                                                            / (1.0 + self._calc_day_co

            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_fractio
                                                                             * float(s
            # libor
            for i in range(1, len_MM):
                temp_discount_factor[i] = temp_discount_factor[index_ON] * temp_discou
                                                            / (1.0 + self._calc_day_co

            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

    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[
        extract_swap_rate_list = extract_rate_list[index_1y:]
        f_interpolated_swap_rate = interp1d(extract_date_list_swap_tenor , extract_swa
        return f_interpolated_swap_rate

    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')
        f_interpolated_swap_rate = self._interpolate_swap_rate()
        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_l
        # 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 uni
        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_boo
            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_boo
        # 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(ir_list[index_roll_tenor][6
        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 - sw
            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:conve
        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.
            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_tr
        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)).str
        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_D
            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_li
        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_D
            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_li
        interpolated_DF_list[0][2] = 1.0
        return interpolated_DF_list
```

```
In [211]: usd_IR_obj = discount_factor('20180118_IR.csv')
          ir_list = usd_IR_obj.get_ir_list()
          #usd_IR_obj .calc_DF_money_market()
          #interpolated_swap_rate_list = usd_IR_obj.interpolated_ir_list_for_bootstrap()
          #interpolated_swap_rate_list
          #usd_IR_obj.calc_DF_swap_rate()
          #usd_IR_obj.calc_annuity(interpolated_swap_rate_list, interpolated_swap_rate_list[7][0
          #DFlist = usd_IR_obj.get_ir_list_with_DF_swap_rate()
          #DFlist
          #f = usd_IR_obj.interpolate_DF()
          #f(10802)
          usd_IR_obj.get_DF('2018/01/30')
          DF_list = usd_IR_obj.get_DF_list()

call constructor
10802


In [207]: import csv

          with open('DF.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) # 2次元配列も書き込める

In [121]: array(0.4484070693339933)

Out[121]: 10950

In [5]: usd_IR_obj.interpolated_ir_list_for_bootstrap()

Out[5]: [['O/N',
          'Money Market',
          'USD',
          '0.014375',
          'None',
          'act/365',
          '2018/01/18',
          '2018/01/19',
          0.9999606179893599 8],
          'Money Market',
          'USD',
          '0.0156118',
          'None',
          'act/365',
          '2018/01/20',
          '2018/02/17',
          0.99872514678198365],
         ['2.0M',
          'Money Market',
          'USD',
          '0.0163482',
          'None',
          'act/365',
          '2018/01/20',
          '2018/03/19',
          0.99733037650296008],
         ['3.0M',
          'Money Market',
          'USD',
          '0.017447',
          'None',
          'act/365',
          '2018/01/20',
          '2018/04/18',
          0.99573278713386282],
         ['6.0M',
          'Money Market',
          'USD',
          '0.019255',
          'None',
          'act/365',
          '2018/01/20',
          '2018/07/17',
          0.99061921454408386],
         ['12.0M', '', '', '', '', '', '', '', ''],
         ['18.0M', '', '', '', '', '', '', '', ''],
         ['24.0M', '', '', '', '', '', '', '', ''],
         ['30.0M', '', '', '', '', '', '', '', ''],
         ['36.0M', '', '', '', '', '', '', '', ''],
         ['42.0M', '', '', '', '', '', '', '', ''],
         ['48.0M', '', '', '', '', '', '', '', ''],
         ['54.0M', '', '', '', '', '', '', '', ''],
         ['60.0M', '', '', '', '', '', '', '', ''],
         ['66.0M', '', '', '', '', '', '', '', ''],
         ['72.0M', '', '', '', '', '', '', '', ''],
         ['78.0M', '', '', '', '', '', '', '', ''],
         ['84.0M', '', '', '', '', '', '', '', ''],
         ['90.0M', '', '', '', '', '', '', '', ''],
         ['96.0M', '', '', '', '', '', '', '', ''],
         ['102.0M', '', '', '', '', '', '', '', ''],
         ['108.0M', '', '', '', '', '', '', '', ''],
         ['114.0M', '', '', '', '', '', '', '', ''],
         ['120.0M', '', '', '', '', '', '', '', ''],
         ['126.0M', '', '', '', '', '', '', '', ''],
         ['132.0M', '', '', '', '', '', '', '', ''],
         ['138.0M', '', '', '', '', '', '', '', ''],
         ['144.0M', '', '', '', '', '', '', '', ''],
         ['150.0M', '', '', '', '', '', '', '', ''],
         ['156.0M', '', '', '', '', '', '', '', ''],
         ['162.0M', '', '', '', '', '', '', '', ''],
         ['168.0M', '', '', '', '', '', '', '', ''],
         ['174.0M', '', '', '', '', '', '', '', ''],
         ['180.0M', '', '', '', '', '', '', '', ''],
         ['186.0M', '', '', '', '', '', '', '', ''],
         ['192.0M', '', '', '', '', '', '', '', ''],
         ['198.0M', '', '', '', '', '', '', '', ''],
         ['204.0M', '', '', '', '', '', '', '', ''],
         ['210.0M', '', '', '', '', '', '', '', ''],
         ['216.0M', '', '', '', '', '', '', '', ''],
         ['222.0M', '', '', '', '', '', '', '', ''],
         ['228.0M', '', '', '', '', '', '', '', ''],
         ['234.0M', '', '', '', '', '', '', '', ''],
         ['240.0M', '', '', '', '', '', '', '', ''],
         ['246.0M', '', '', '', '', '', '', '', ''],
         ['252.0M', '', '', '', '', '', '', '', ''],
         ['258.0M', '', '', '', '', '', '', '', ''],
         ['264.0M', '', '', '', '', '', '', '', ''],
         ['270.0M', '', '', '', '', '', '', '', ''],
         ['276.0M', '', '', '', '', '', '', '', ''],
         ['282.0M', '', '', '', '', '', '', '', ''],
         ['288.0M', '', '', '', '', '', '', '', ''],
         ['294.0M', '', '', '', '', '', '', '', ''],
         ['300.0M', '', '', '', '', '', '', '', ''],
         ['306.0M', '', '', '', '', '', '', '', ''],
         ['312.0M', '', '', '', '', '', '', '', ''],
         ['318.0M', '', '', '', '', '', '', '', ''],
         ['324.0M', '', '', '', '', '', '', '', ''],
         ['330.0M', '', '', '', '', '', '', '', ''],
         ['336.0M', '', '', '', '', '', '', '', ''],
         ['342.0M', '', '', '', '', '', '', '', ''],
         ['348.0M', '', '', '', '', '', '', '', ''],
         ['354.0M', '', '', '', '', '', '', '', ''],
         ['360.0M', '', '', '', '', '', '', '', '']]
```

```
In [189]: 365 * 1
```

```
Out[189]: 365
```

```
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 [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
```