

Section 7

<input type="checkbox"/> Date	
<input type="checkbox"/> Property	
<input checked="" type="checkbox"/> Weeks	4주차

3. From Asset Management to Asset-Liability Management

PENSION FUND CRISIS

1. 1999.12 → 2003.05

▼ Hello

Year	NASDAQ INDEX
March,2000	5132.52
Sep,2000	4234.33
Jan,2001	2291.86
Oct,2002	1108.49

2. 2008 금융위기

3. 2017

INTRODUCING A LIABILITY BENCHMARK

$$F_t = \frac{A_t}{L_t}$$

- F : Funding Ratio

▼ A : Asset

- Cash
- Investments
- Inventory
- Office equipment
- Machinery
- Real estate

- Company-owned vehicles

▼ L : Liability

- Bank debt
- Mortgage debt
- Money owed to suppliers (accounts payable)
- Wages owed
- Taxes owed

Asset이 떨어지더라도 Liability이 더 많이 떨어진다면 Funding Ratio는 증가함

$$S_t = A_t - L_t$$

▼ S : Surplus

▼ A : Asset

▼ L : Liability

Positive : Surplus, Negative : Deficit

WRAP-UP

Funding Ratio → 자산 중 얼마나 커버가 되는가?, 100 or 그 이상의 비율이 좋은 형태임

4. Lab - Present Values, liabilities and funding ratio

The purpose of investing : make sure that I will have the money that I need to do in the things that I want to do in the future.

→ I will have the resources to be able to have the consumption patterns that I want to do in the future.

1. what is this future liability

2. to measure do I have enough money today to be able to meet that, and how do I measure

When funding ratio is low

1. get more assets, And in a pension fund this amounts to the sponsoring company actually putting in more cash into the company
2. you have to get a higher rate of return. finding some way to improve returns.

```
def discount(t, r):
    """
    Compute the price of a pure discount bond that pays $1 at time t where t is in years and r is the annual interest rate
    """
    return (1+r)**(-t)

def pv(l, r):
    """
    Compute the present value of a list of liabilities given by the time (as an index) and amounts
    """
    pass
```

```

"""
dates = l.index
discounts = discount(dates, r)
return (discounts*l).sum()

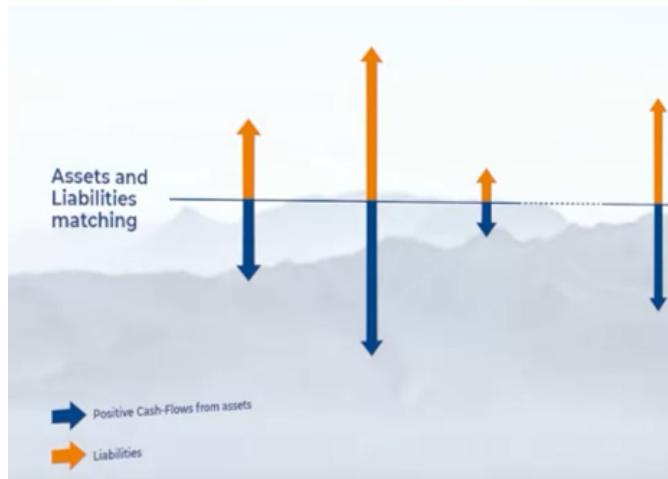
def funding_ratio(assets, liabilities, r):
"""
    Computes the funding ratio of a series of liabilities, based on an interest rate and current value of assets
"""
    return assets/pv(liabilities, r)

```

5. Liability Hedging Portfolios

Investor's main concern : Unexpected increase in the present value of their liabilities.

- 부채의 가치가 급격하게 커지는 것은 위험하다.



Hedging = Cash-flow matching.

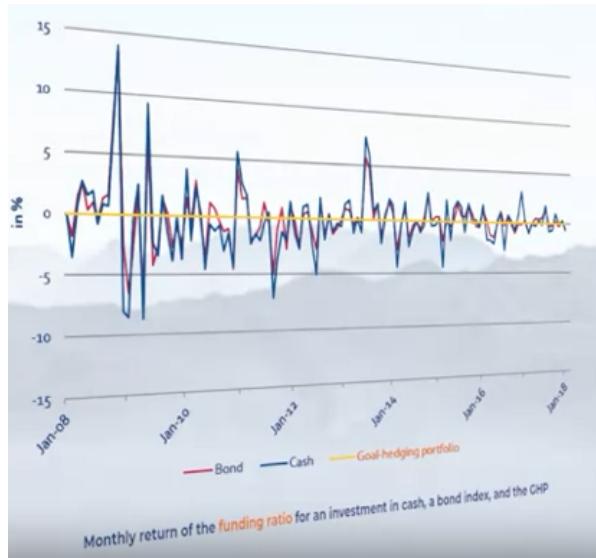
These cashflows will match the date and nominal amount of those liability or goal payments. So in other words, the **liability or goal hedging portfolio** is an asset portfolio that behaves exactly like the liabilities that we are trying to immunize against or that we are trying to protect

Payoffs matching the date and nominal amount of liability/goal payments

standard bond : not safe / start paying coupon payments on annual or semi annual basis until maturity date.

retirement bond : it's a bond that's going to pay a series of constants or perhaps inflation link cashflows starting at retirement date and not immediately but only at retirement date.

→ long term duration / volatile



자산 측면에서 현금은 안전한 것처럼 보인다.

long duration 에서는 오히려 현금이 위험. return on the funding ratio 에 집중. 즉, 자산은 변함없이 있어도 부채가 계속 변하면 Funding ration가 변하기 때문에 funding ratio가 유지되는 전략을 사용해야 한다.

What is the safe liability-hedging asset for a 45Y old investor preparing for retirement at age 65.

- 3 month T-Bills?
- 20-year inflation-linked pure discount bond?
- Deferred inflation-linked annuity with a 20 years deferral period

맞습니다

A deferred inflation-linked annuity with a 20 year deferral period will start paying cash flows when the investor reaches age 65, and will pay replacement income cash-flow as long as the investor is alive. This make it the perfect safe retirement asset.

- 목표시기면역전략

미래의 특정한 시점에서 자금의 유출이 필요한 기업, 개인은 현재 보유한 자금을 채권에 투자하여 유출이 필요한 시점에 이를 이용하고자 한다. 이 때 향후 이자율의 변동에 따라 채권투자로 인한 현금 유입액이 현금유출액과 다르게 되어 위험에 노출 될 수 있다.

이 때 향후 이자율의 변동에 따라 채권투자로 인한 현금유입액이 현금유출액과 다르게 되어 위험에 노출될 수 있다. 이러한 경우 채권투자자는 위험을 제거하고자 하는 시점(목표시점 ex. 은퇴시점)과 드레이션을 일치시켜 목표시점에서 이자율의 변화에 상관

없는 부를 확보할 수 있다. 이를 위해서는 변화하는 채권 포트폴리오의 드레이션을 잔여목표기간과 계속적으로 일치시켜야 특정 시점에서 발생할 이자율의 위험을 없앨 수 있다. 즉, 목표시기면역전략은 동적해지 전략이다.

단, 이는 수익률곡선이 평행할 때 효과적인 전략이며, 수익률곡선이 비평행이동 하는 경우 면역전략은 정확히 이행되지 않아 목표수익률을 달성하지 못할 가능성이 발생하는데, 이를 면역전략위험(immunization risk)이라고 한다. 이는 채권의 볼록성 성질 때문이다.

6. Lab Session-CIR Model and cash vs ZC bonds

CIR Model (Cox–Ingersoll–Ross)

$$dr_t = a(b - r_t)dt + \sigma \sqrt{r_t} dW_t$$

- dr_t : t 순간의 이자율 변화
- a : 얼마나 빨리 장기 금리로 되돌릴 것인가? (mean reversion speed, parameter)
 - i) a=0 → 장기 금리가 어떠한 역할도 하지 못함
 - ii) a≠0 → a가 커질수록 더 빨리 장기 금리로 수렴함
- b : 장기 금리의 평균 (long-term interest rate, parameter)
- r_t : 현재의 이자율
→ b-r_t : 현재의 이자율이 평균으로 부터 얼마나 차이나는가?
- sigma : volatility
 - i) sigma = 0 → 계속 평균치와 같았음
 - ii) sigma ≠ 0 → 클수록 해당 기간동안 큰 변화가 발생하였음
- r_t : Vasicek model에서 얻을 수 있으며, 수익률이 음수가 되지 않도록 조절함
 - i) r_t = 0 → shock amount가 없음, no impact
 - ii) r_t ≠ 0 → 작을 수록 shock가 점점 더 작아짐

Short Rate vs Annualized

$$(1 + \frac{r}{N})^N$$

- i) $1 + 1 \times 1 = 2$
- ii) $1 + (1 \times 0.5) + (1 + 1 \times 0.5) \times 0.5 = 2.25$
- iii) $1 + (1 \times 0.25) + (1 + 1 \times 0.25) \times 0.25 + (1 + (1 \times 0.25) + (1 + 1 \times 0.25) \times 0.25) \times 0.25 \dots = 2.44140625$

이때 N을 더 크게 만든다면 일반화 시킬 수 있음

$$\begin{aligned} 1 + r_{\text{annual}} &= e^{r_{\text{inst}}} \\ r_{\text{inst}} &= \ln(1 + r_{\text{annual}}) \end{aligned}$$

```
import numpy as np
import pandas as pd
import edhec_Risk_kit as erk
```

```

def inst_to_ann(r):
    # convert short rate to an annualized rate
    return np.expm1(r)
def ann_to_inst(r):
    # convert annualized to a short rate
    return np.log1p(r)

```

$$dr_t = a(b - r_t)dt + \sigma\sqrt{r_t}dW_t$$

```

import numpy as np
import pandas as pd
import edhec_risk_kit_125 as erk
%load_ext autoreload
%autoreload 2

def inst_to_ann(r):
    """
    Convert an instantaneous interest rate to an annual interest rate
    """
    return np.expm1(r)

def ann_to_inst(r):
    """
    Convert an instantaneous interest rate to an annual interest rate
    """
    return np.log1p(r)

def cir(n_years = 10, n_scenarios=1, a=0.05, b=0.03, sigma=0.05, steps_per_year=12, r_0=None):
    """
    Generate random interest rate evolution over time using the CIR model
    b and r_0 are assumed to be the annualized rates, not the short rate
    and the returned values are the annualized rates as well
    """
    if r_0 is None: r_0 = b
    r_0 = ann_to_inst(r_0)
    dt = 1/steps_per_year
    num_steps = int(n_years*steps_per_year) + 1 # because n_years might be a float

    # scale : Standard deviation, size : Output Shape
    shock = np.random.normal(0, scale=np.sqrt(dt), size=(num_steps, n_scenarios))
    rates = np.empty_like(shock)
    rates[0] = r_0
    for step in range(1, num_steps):
        r_t = rates[step-1]
        d_r_t = a*(b-r_t)*dt + sigma*np.sqrt(r_t)*shock[step]
        rates[step] = abs(r_t + d_r_t) # just in case of roundoff errors going negative

    return pd.DataFrame(data=inst_to_ann(rates), index=range(num_steps))

%matplotlib inline

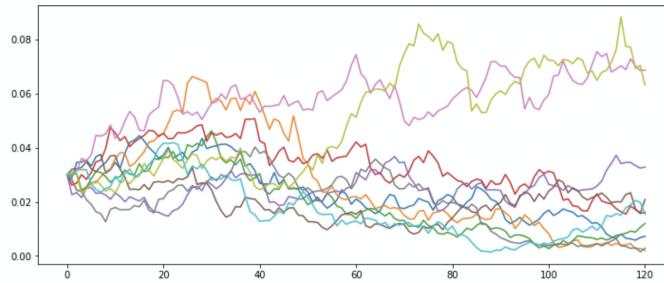
```

10개의 시나리오 예시

```

: cir(n_scenarios=10).plot(legend=False, figsize=(12,5))
: <matplotlib.axes._subplots.AxesSubplot at 0x12f228a10>

```



기존의 식을 조금 더 확장한다면

Generating the random price evolution of a Zero-Coupon Bond

The model can also be used to generate the movement of bond prices for a zero coupon bond that are implied by the generated interest rate, using the following equations:

$$P(t, T) = A(t, T)e^{-B(t, T)r_t}$$

where

$$A(t, T) = \left(\frac{2he^{(a+h)\tau/2}}{2h + (a+h)(e^{th} - 1)} \right)^{2ab\sigma^2}$$

and

$$B(t, T) = \frac{2(e^{th} - 1)}{2h + (a+h)(e^{th} - 1)}$$

and

$$h = \sqrt{a^2 + 2\sigma^2}$$

and

$$\tau = T - t$$

다음과 같이 나타낼 수 있는데

```
import math
def cir(n_years = 10, n_scenarios=1, a=0.05, b=0.03, sigma=0.05, steps_per_year=12, r_0=None):
    """
    Generate random interest rate evolution over time using the CIR model
    b and r_0 are assumed to be the annualized rates, not the short rate
    and the returned values are the annualized rates as well
    """
    if r_0 is None: r_0 = b
    r_0 = ann_to_inst(r_0)
    dt = 1/steps_per_year
    num_steps = int(n_years*steps_per_year) + 1 # because n_years might be a float

    shock = np.random.normal(0, scale=np.sqrt(dt), size=(num_steps, n_scenarios))
    rates = np.empty_like(shock)
    rates[0] = r_0

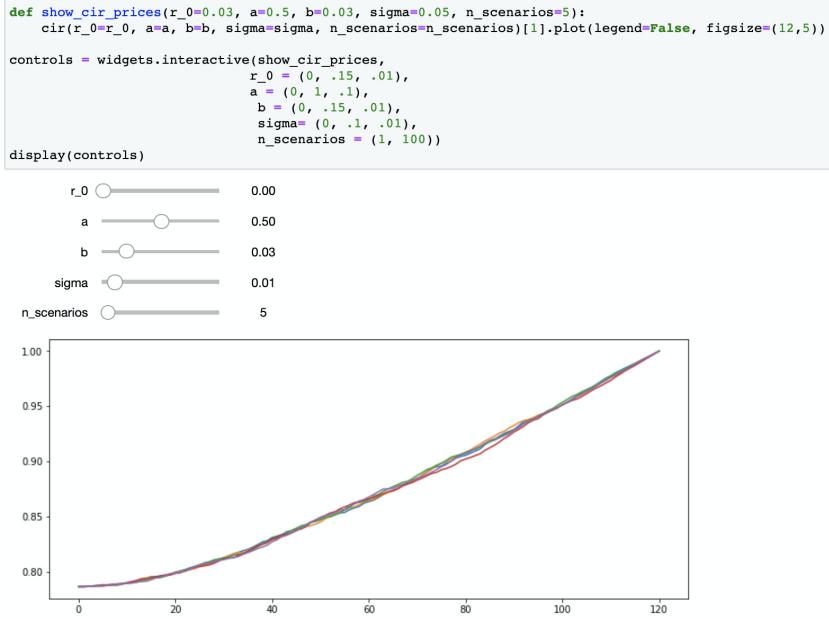
    # 기존의 코드에서 이 부분을 추가하여 좀 더 복잡해졌음
    ## For Price Generation
    h = math.sqrt(a**2 + 2*sigma**2)
    prices = np.empty_like(shock)
    ####

    def price(ttm, r):
        _A = ((2*h*math.exp((h+a)*ttm/2))/(2*h+(h+a)*(math.exp(h*ttm)-1)))**((2*a*b/sigma**2))
        _B = (2*(math.exp(h*ttm)-1))/(2*h + (h+a)*(math.exp(h*ttm)-1))
        _P = _A*np.exp(-_B*r)
        return _P
    prices[0] = price(n_years, r_0)
    ####

    for step in range(1, num_steps):
        r_t = rates[step-1]
        d_r_t = a*(b-r_t)*dt + sigma*np.sqrt(r_t)*shock[step]
        rates[step] = abs(r_t + d_r_t)
        # generate prices at time t as well ...
        prices[step] = price(n_years-step*dt, rates[step])

    rates = pd.DataFrame(data=inst_to_ann(rates), index=range(num_steps))
    ### for prices
    prices = pd.DataFrame(data=prices, index=range(num_steps))
    ###
    return rates, prices
```

어떻게 수치를 변경하든 결과적으로는 모든 값이 1로 수렴한다.



Now that we can generate randomly evolving interest rates and bond prices we can add the new code for `cir` to the toolkit.

Assume our liabilities are to pay one dollar in 10 years time. Clearly, the liability is perfectly matched by the price of a zero coupon bond that matures in 10 years. Therefore, this is perfectly matched by buying a zero coupon bond.

7. Liability-driven investing(LDI)

LDI

현대 투자의 대부분을 차지함

투자에 있어서 '탐욕'과 '공포'에 집중해야 함

1. Greed (탐욕)

성과(performance)가 필요함

2. Fear (공포)

자본이 증가하고 있더라도 언제 떨어질지 모른다는 공포는 존재함

이 두 가지 요소를 따로 관리하여 각각의 포트폴리오를 만들었을 때 최선의 결과가 도출됨

Example : SKIER'S ANALOGY



다음과 같은 두 가지 상태의 눈이 존재할 때 스키 장비는 어떤 상태에 맞춰서 살 것인가?

- 모든 상황에 적합한 밸런스 형 장비를 산다 (50:50)

→ 효율적임

- 2가지 종류의 스키 장비를 산다 (구불구불 & 경사진)

→ 적당히 해야지—

Greeks of LDI

$$\max_w E \left[u \left(\frac{A_T}{L_T} \right) \right] \Rightarrow w^* = \frac{\lambda_{PSP}}{\gamma \sigma_{PSP}} w^{PSP} + \beta_{L,LHP} \left(1 - \frac{1}{\gamma} \right) w^{LHP}$$

Funding Ratio의 기대 값을 최대화

A_T : Asset

L_T : Liability

- 가중치 요소

$$\max_w E \left[u \left(\frac{A_T}{L_T} \right) \right] \Rightarrow w^* = \frac{\lambda_{PSP}}{\gamma \sigma_{PSP}} w^{PSP} + \beta_{L,LHP} \left(1 - \frac{1}{\gamma} \right) w^{LHP}$$

포트폴리오의 performance를 극대화 하기 위한 요소 (Sharpe Ratio를 최대화)

$$\max_w E \left[u \left(\frac{A_T}{L_T} \right) \right] \Rightarrow w^* = \frac{\lambda_{PSP}}{\gamma \sigma_{PSP}} w^{PSP} + \beta_{L,LHP} \left(1 - \frac{1}{\gamma} \right) w^{LHP}$$

포트폴리오의 fear를 최소화 하기 위한 요소 (liability-hedging portfolio)

- 세부 요소

λ_{PSP} is the PSP Sharpe ratio: $\lambda = 0 \Rightarrow$ no investment in PSP

β_{LHP} is the beta of liabilities w.r.t. LHP: $\beta = 0 \Rightarrow$ no investment in LHP

σ_{PSP} is the PSP volatility: $\sigma = \infty \Rightarrow$ no investment in PSP

γ is the risk-aversion: $\gamma = \infty \Rightarrow$ no investment in PSP

PSP(performance seeking portfolio) : Sharpe Ratio, 수익과 관련한 요소들

LHP(liability-hedging portfolio) : Sharpe Ratio, 위험을 발생시키는 것에 관심이 없음

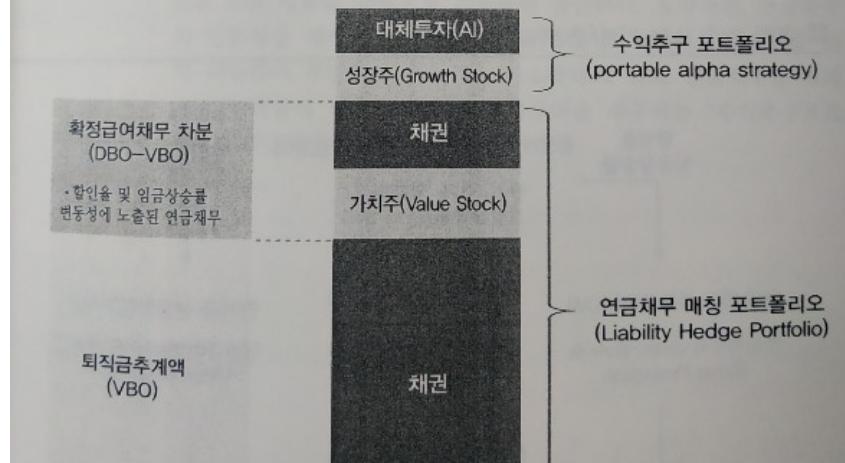
gamma : 위험 회피

는 임여금에 대해서는 초과 수익률을 추구하는 “수익추구 포트폴리오(returns-seeking portfolio: RSP)”를 구성한다. 이러한 전략으로 구성되는 LDI 포트폴리오(LDIP)는 임여금의 변동성을 최소화하는 전통적 PALM에 비하여 적극적으로 초과수익을 추구한다는 점에서 차별화된다. 요약하면, 전략 수립 시점(t)에 대하여

$$\bullet LDIP(t) = \theta_t \times LMP(t) + (1 - \theta_t) \times RSP(t), 0 < \theta_t < 1$$

여기서 $LMP(t)$ 는 연금채무의 드레이션을 매칭하는 채권위주 투자전략이며, $RSP(t)$ 는 초과수익(portable alpha)을 추구하는 대체투자, 성장주 등으로 구성된다([그림 2-6] 참조). 또한 θ_t 는 사용자의 재무건전성 및 투자위험회피성향 그리고 연금제도의 성숙도 및 적립비율 등을 고려하여 결정된다.

그림 2-6 LDIP 구성 일반 예



8. Lab Session - Liability driven Investing

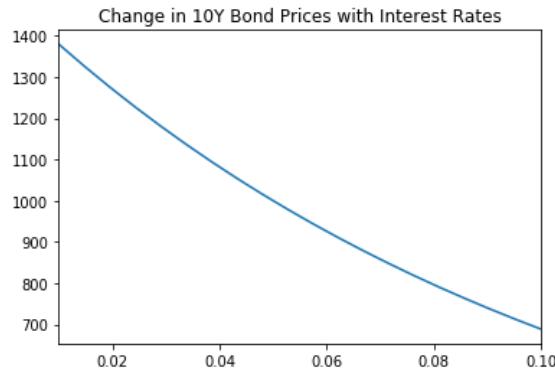
zero-coupon bond는 없다.

```
import math
def bond_cash_flows(maturity, principal=100, coupon_rate=0.03, coupons_per_year=12):
    """
    Returns the series of cash flows generated by a bond,
    indexed by the payment/coupon number
    """
    n_coupons = round(maturity*coupons_per_year)
    coupon_amt = principal*coupon_rate/coupons_per_year
    coupon_times = np.arange(1, n_coupons+1)
    cash_flows = pd.Series(data=coupon_amt, index=coupon_times)
    cash_flows.iloc[-1] += principal # add the principal to the last payment
    return cash_flows

def bond_price(maturity, principal=100, coupon_rate=0.03, coupons_per_year=12, discount_rate=0.03):
    """
    Computes the price of a bond that pays regular coupons until maturity
    at which time the principal and the final coupon is returned
    This is not designed to be efficient, rather,
    it is to illustrate the underlying principle behind bond pricing!
    """
    cash_flows = bond_cash_flows(maturity, principal, coupon_rate, coupons_per_year)
    return pv(cash_flows, discount_rate/coupons_per_year)
```

이자율 변화에 따른 채권 가격의 변화를 확인해보자.

```
rates = np.linspace(.01, .10, num=20)
prices = [erk.bond_price(10, 1000, 0.05, 2, rate) for rate in rates]
pd.DataFrame(data=prices, index=rates).plot(title="Change in 10Y Bond Prices with Interest Rates", legend=False)
```



이자율 변화에 따른 민감도에 동일하게 반응하는 포트폴리오를 구축하고 싶은 것이 목표. 이를 위해 Duration Matching이 필요하다. 이때 Macaulay Duration을 계산한다. 멕컬리 듀레이션은 채권의 가중평균만기일을 의미한다. 즉, 각 시점에서의 CF

의 현가를 구해서 가중평균을 구하는 것을 의미한다. 이 식은 코드로 아래와 같이 구현된다.

```
def macaulay_duration(flows, discount_rate):
    """
    Computes the Macaulay Duration of a sequence of cash flows, given a per-period discount rate
    """
    discounted_flows = discount(flows.index, discount_rate)*flows
    weights = discounted_flows/discounted_flows.sum()
    return np.average(flows.index, weights=weights)

erk.macaulay_duration(erk.bond_cash_flows(3, 1000, .06, 2), 0.06/2)
```

예를 들어 만기가 3년, 액면가격 1000원, 쿠폰이자율 0.06, 매반기말 이자지급하는 채권을 가정해보자. 즉 반기마다 30원 씩의 쿠폰이 지급되는데 이때의 듀레이션을 계산하면 약 5.58이 나오고 1기간이 6개월이므로 연단위 환산하면 2.79년이라는 시간이 지나야 빌려준 1000원을 회수할 수 있는 기간이라 생각할 수 있다.

```
liabilities = pd.Series(data = [100000, 100000], index=[10, 12])
```

```
erk.macaulay_duration(liabilities, .04)
```

```
10.960799385088393
```

Now assume we have two types of bonds available. We have a 10 year bond and a 15 year bond. Each of them pays a 5% coupon once a year and has a face value of \$1000. What are the durations of these bonds?

```
md_10 = erk.macaulay_duration(erk.bond_cash_flows(10, 1000, .05, 1), .04)
md_10
```

```
8.190898824083233
```

```
md_20 = erk.macaulay_duration(erk.bond_cash_flows(20, 1000, .05, 1), .04)
md_20
```

```
13.544718122145921
```

Therefore, we need to hold a portfolio of these two bonds that has a combined target duration that matches the duration of the liability, which is given by the following expression, where w_s is the weight in the short duration bond which has duration d_s and the duration of the longer bond is d_l . We designate the targeted duration as d_t .

In our case, the fraction in the short duration asset w_s should be such that:

$$w_s \times 8.19 + (1 - w_s) \times 13.54 = 10.96$$

more generally:

$$w_s \times d_s + (1 - w_s) \times d_l = d_t$$

rearranging gives:

$$w_s = \frac{d_t - d_l}{d_l - d_s}$$

각각 10년, 12년 만기인 10만 달러 부채의 듀레이션은 10.96이다.

이때 5% 쿠폰 이자율 10년 만기 채권과 20년 만기 채권으로 헛징을 하려고 한다. 이를 위해 각각의 듀레이션을 구하고 비중을 정한다. 10년 만기 채권의 듀레이션은 8.19, 20년 만기 채권의 듀레이션은 13.54이고 이를 가중 평균하여 각각의 비중을 산정하게 된다.

```
def match_durations(cf_t, cf_s, cf_l, discount_rate):
    """
    Returns the weight W in cf_s that, along with (1-W) in cf_l will have an effective
    duration that matches cf_t
    """
    d_t = macaulay_duration(cf_t, discount_rate)
    d_s = macaulay_duration(cf_s, discount_rate)
    d_l = macaulay_duration(cf_l, discount_rate)
    return (d_l - d_t)/(d_l - d_s)

def funding_ratio(assets, liabilities, r):
    """
```

```

    Computes the funding ratio of a series of liabilities, based on an interest rate and current value of assets
"""
return pv(assets, r)/pv(liabilities, r)

```

위 식을 코드로 적용해 비중을 구해보면 short bond 의 비중은 약 0.48로 정해진다.

```

: p_short = erk.bond_price(10, 1000, .05, 1, 0.04)
: p_long = erk.bond_price(20, 1000, .05, 1, 0.04)
: a_0 = 130000
: dm_assets=pd.concat([a_0*w_s*short_bond/p_short,a_0*(1-w_s)*long_bond/p_long])
: erk.macaulay_duration(dm_assets, 0.04)

: 10.960799385088393

: erk.macaulay_duration(liabilities, 0.04)

: 10.960799385088393

```

이를 이용해 듀레이션을 구해보면 Matching 됨을 알 수 있다.

```

: rates = np.linspace(0, .1, 20)
fr_change = pd.DataFrame({
    "Long Bond": [erk.funding_ratio(lb_assets, liabilities, r) for r in rates],
    "Short Bond": [erk.funding_ratio(sb_assets, liabilities, r) for r in rates],
    "Duration Matched Bonds": [erk.funding_ratio(dm_assets, liabilities, r) for r in rates]
}, index=rates)
fr_change.plot(title='Change in Funding Ratio', figsize=(10,6))

```

