EFM Fall 2014, Week 6: The Capital Asset Pricing Model

Zachry Wang, Allan Zhang, Jason Phang

November 5, 2014

Table of Contents

Table of Contents

NumPy

NumPy

- Numpy is a library for efficient and optimized mathematical operations.

NumPy

NumPy

- Numpy is a library for efficient and optimized mathematical operations.
- The structures in NumPy are more restrictive, but also a lot more powerful.

NumPy

- Numpy is a library for efficient and optimized mathematical operations.
- The structures in NumPy are more restrictive, but also a lot more powerful.
- It will be your primary toolset for numerical and data manipulation (until you learn **pandas**, another library which is built on NumPy)

NumPy

NumPy

- Numpy is a library for efficient and optimized mathematical operations.
- The structures in NumPy are more restrictive, but also a lot more powerful.
- It will be your primary toolset for numerical and data manipulation (until you learn **pandas**, another library which is built on NumPy)
- To use NumPy (commonly imported as **np**):

Import NumPy

```
import numpy as np
print np.e
```

NumPy

NumPy

- Numpy is a library for efficient and optimized mathematical operations.
- The structures in NumPy are more restrictive, but also a lot more powerful.
- It will be your primary toolset for numerical and data manipulation (until you learn **pandas**, another library which is built on NumPy)
- To use NumPy (commonly imported as **np**):

Import NumPy

```
import numpy as np
print np.e
```

NumPy Arrays

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient
- Similar to Python lists, but different in many crucial ways

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient
- Similar to Python lists, but different in many crucial ways

Python Lists	**NumPy Arrays**

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient
- Similar to Python lists, but different in many crucial ways

Python Lists	**NumPy Arrays**
Variable length	Fixed Length

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient
- Similar to Python lists, but different in many crucial ways

Python Lists	**NumPy Arrays**
Variable length	Fixed Length
Variable type	Single type (usually Ints/Floats)

NumPy Arrays

- One of the fundamental data structures in NumPy is the **NumPy array**.
- An array is a data structure code based on underlying arrays in C, and thus is more compact and more efficient
- Similar to Python lists, but different in many crucial ways

Python Lists	**NumPy Arrays**
Variable length	Fixed Length
Variable type	Single type (usually Ints/Floats)
Slow(-ish)	Fast!

NumPy

Basic NumPy

```
import numpy as np

print range(10)
print np.array(range(10))
print np.arange(10)
```

NumPy

Basic NumPy

```
import numpy as np

print range(10)
print np.array(range(10))
print np.arange(10)
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

NumPy Benefits

NumPy Benefits

- The first most obvious benefit of NumPy arrays is the ability to do vector-wise operations

NumPy Benefits

- The first most obvious benefit of NumPy arrays is the ability to do vector-wise operations
- This involves treating your arrays like regular vectors in math

NumPy Benefits

NumPy Benefits

Vector-wise Operations

```
print np.arange(10)*2
print np.arange(10)**2
print np.arange(10) + 5.5
print np.arange(10)**2 - np.arange(10)*2
```

NumPy Benefits

Vector-wise Operations

```
print np.arange(10)*2
print np.arange(10)**2
print np.arange(10) + 5.5
print np.arange(10)**2 - np.arange(10)*2
```

Output

```
[ 0  2  4  6  8 10 12 14 16 18]
[ 0  1  4  9 16 25 36 49 64 81]
[  5.5   6.5   7.5   8.5   9.5  10.5  11.5  12.5  13.5  14.5]
[ 0 -1  0  3  8 15 24 35 48 63]
```

NumPy Benefits

NumPy Benefits

- The second benefit of NumPy arrays if the ability to do boolean (T/F) indexing

NumPy Benefits

- The second benefit of NumPy arrays if the ability to do boolean (T/F) indexing
- Instead of indexing by a number, you index by providing a boolean array

NumPy Benefits

NumPy Benefits

Boolean Indexing

```
nums = np.arange(1,20)
print nums

indexer = (nums%3==0)
print indexer

print nums[indexer]
```

NumPy Benefits

Boolean Indexing

```
nums = np.arange(1,20)
print nums

indexer = (nums%3==0)
print indexer

print nums[indexer]
```

Output

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
[False False  True False False  True False False  True False False
 False False  True False False  True False]
[ 3  6  9 12 15 18]
```

NumPy Benefits

NumPy Benefits

- Another benefit of NumPy is the NaN type, which is a placeholder for problematic/missing numbers.

NumPy Benefits

- Another benefit of NumPy is the NaN type, which is a placeholder for problematic/missing numbers.
- NaN is short for "Not a Number"

NumPy Benefits

- Another benefit of NumPy is the NaN type, which is a placeholder for problematic/missing numbers.
- NaN is short for "Not a Number"
- NaN is invariant to any kind of mathematical operation

NumPy Benefits

- Another benefit of NumPy is the NaN type, which is a placeholder for problematic/missing numbers.
- NaN is short for "Not a Number"
- NaN is invariant to any kind of mathematical operation
- There are also Inf and -Inf types.

NumPy Benefits

NumPy Benefits

NaNs

```
nums = np.arange(0,1,0.1)
nums[2] = np.NaN
print nums
print nums * 100
```

NumPy Benefits

NaNs

```
nums = np.arange(0,1,0.1)
nums[2] = np.NaN
print nums
print nums * 100
```

Output

```
[ 0.   0.1  nan  0.3  0.4  0.5  0.6  0.7  0.8  0.9]
[  0.  10.  nan  30.  40.  50.  60.  70.  80.  90.]
```

NumPy Benefits

NumPy Benefits

- NumPy also comes with a plethora of other mathematical functions, almost all of which work vector-wise.

NumPy Benefits

NumPy Benefits

NumPy Functions

```
nums = np.arange(5)*np.pi
print sin(nums)
print exp(nums)
```

NumPy Benefits

NumPy Functions

```
nums = np.arange(5)*np.pi
print sin(nums)
print exp(nums)
```

Output

```
[  0.00000000e+00   1.22464680e-16  -2.44929360e-16   3.67394040e-16
  -4.89858720e-16]
[  1.00000000e+00   2.31406926e+01   5.35491656e+02   1.23916478e+04
   2.86751313e+05]
```

NumPy Benefits

NumPy Benefits

- NumPy also has a much nicer random library to work with.

NumPy Benefits

NumPy Benefits

NumPy Random

```
np.random.seed(1234)
hist(np.random.randn(10000))
```

NumPy Benefits

NumPy Random

```
np.random.seed(1234)
hist(np.random.randn(10000))
```

Output

```
(array([   10.,    54.,   356.,  1059.,  2287.,  2799.,  2113.,   997.,
          288.,    37.]),
 array([-3.88089841, -3.16402978, -2.44716115, -1.73029252, -1.01342389,
        -0.29655526,  0.42031337,  1.137182  ,  1.85405064,  2.57091927,
         3.2877879 ]),
 <a list of 10 Patch objects>)
```

NumPy ndarrays and matrices

NumPy ndarrays and matrices

- Aside from arrays, NumPy also has **ndarrays** (multi-dimensional arrays), and **matrices** (a 2-d structure which you've seen in math)

NumPy ndarrays and matrices

- Aside from arrays, NumPy also has **ndarrays** (multi-dimensional arrays), and **matrices** (a 2-d structure which you've seen in math)
- Note that a Matrix is a different type from a ndarray with 2-dimensions, although you can easily switch between the two.

NumPy ndarrays and matrices

NumPy ndarrays and matrices

NumPy Matrices (1)

```
nums = np.arange(25)
print nums
square = np.reshape(nums,(5,5))
print square
mat = np.matrix(square)
print mat
```

NumPy ndarrays and matrices

NumPy Matrices (1)

```
nums = np.arange(25)
print nums
square = np.reshape(nums,(5,5))
print square
mat = np.matrix(square)
print mat
```

Output

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

NumPy ndarrays and matrices

NumPy ndarrays and matrices

NumPy Matrices (2)

```
print mat*mat
print mat.T
print np.matrix([[1,2],[3,4]]).I
```

NumPy Matrices (2)

```
print mat*mat
print mat.T
print np.matrix([[1,2],[3,4]]).I
```

Output

```
[[ 150  160  170  180  190]
 [ 400  435  470  505  540]
 [ 650  710  770  830  890]
 [ 900  985 1070 1155 1240]
 [1150 1260 1370 1480 1590]]
[[ 0  5 10 15 20]
 [ 1  6 11 16 21]
 [ 2  7 12 17 22]
 [ 3  8 13 18 23]
 [ 4  9 14 19 24]]
[[-2.   1. ]
 [ 1.5 -0.5]]
```

NumPy ndarrays and matrices

NumPy ndarrays and matrices

- Note that you can run NumPy functions across specific axes of an ndarray or matrix.

NumPy ndarrays and matrices

NumPy Matrices (3)

```
print mat
print np.sum(mat,0)
print np.sum(mat,1)
```

NumPy ndarrays and matrices

NumPy Matrices (3)

```
print mat
print np.sum(mat,0)
print np.sum(mat,1)
```

Output

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[[50 55 60 65 70]]
[[ 10]
 [ 35]
 [ 60]
 [ 85]
 [110]]
```

NumPy ndarrays and matrices

NumPy Matrices (4)

```
print square
print np.sum(square,0)
print np.sum(square,1)
```

NumPy ndarrays and matrices

NumPy Matrices (4)

```
print square
print np.sum(square,0)
print np.sum(square,1)
```

Output

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[50 55 60 65 70]
[ 10  35  60  85 110]
```

NumPy: Summary

NumPy: Summary

- NumPy is incredibly optimized, and much faster than working with Python lists for any kind of mathematical computation

NumPy: Summary

- NumPy is incredibly optimized, and much faster than working with Python lists for any kind of mathematical computation
- You should learn how to manipulate them very well

NumPy: Summary

- NumPy is incredibly optimized, and much faster than working with Python lists for any kind of mathematical computation
- You should learn how to manipulate them very well
- They are also the foundation for other commonly used libraries, like **SciPy** and **pandas**

NumPy: Summary

- NumPy is incredibly optimized, and much faster than working with Python lists for any kind of mathematical computation
- You should learn how to manipulate them very well
- They are also the foundation for other commonly used libraries, like **SciPy** and **pandas**
- Whenever you're working with lists or data from here on out, think to yourself: "Should I be using NumPy?"

NumPy: Summary

- NumPy is incredibly optimized, and much faster than working with Python lists for any kind of mathematical computation
- You should learn how to manipulate them very well
- They are also the foundation for other commonly used libraries, like **SciPy** and **pandas**
- Whenever you're working with lists or data from here on out, think to yourself: "Should I be using NumPy?"
  - The answer isn't always yes!

Table of Contents

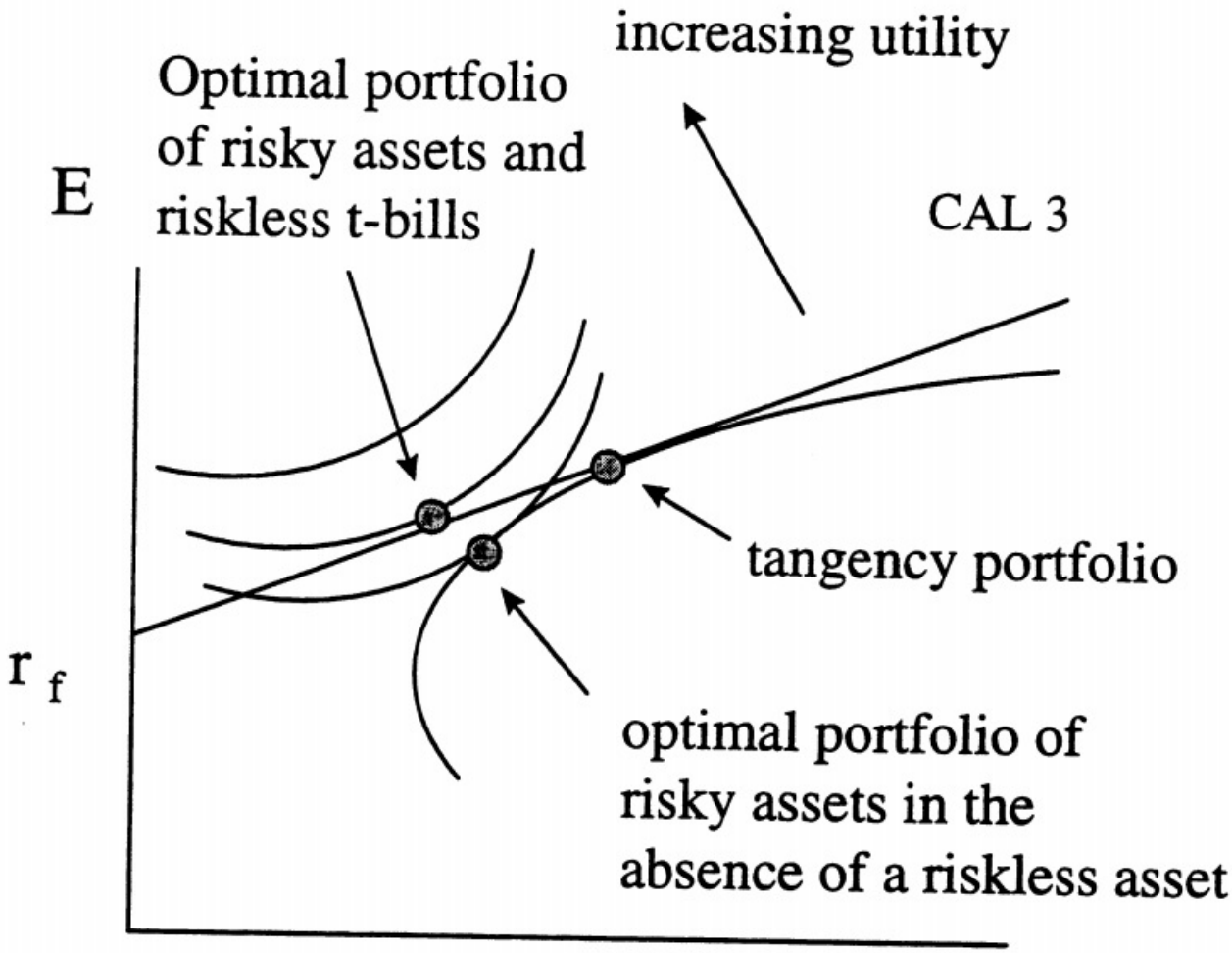
Brief Recap

In Mean-Variance Analysis, we

- Took risk and return of all securities as given.
- Plotted all feasible portfolios.
- Chose the best one based on a utility function (though, this step is not really necessary)

Results from Mean-Variance Analysis

- The optimal portfolio of risky assets is the tangency portfolio.
- Every investor would hold some combination of the risk-free asset and the tangency portfolio.



Motivating the CAPM

- In the Mean-Variance Analysis, we were **given a set of assets and their characteristics (mean, variances, and covariances)**, then we found an optimal portfolio.
- The CAPM seeks to answer a different question: why do some securities earn higher returns on average than others?

Motivating the CAPM

- In the Mean-Variance Analysis, we were **given a set of assets and their characteristics (mean, variances, and covariances)**, then we found an optimal portfolio.
- The CAPM seeks to answer a different question: why do some securities earn higher returns on average than others?
  - Answer: Risk, but how do we measure risk?
  - How much extra expected return do we require to bear the risk?
  - CAPM uses an economic argument to answer this question.

Table of Contents

Deriving the CAPM

- In order to identify the tangency portfolio, we showed that it had to have the following property (we use T to denote the tangency portfolio):

$$\frac{\mu_i - \mu_f}{\sigma_{i,T}} = \frac{\mu_j - \mu_f}{\sigma_{j,T}} \qquad \forall i, j$$

- Note this is a defining feature of the tangency portfolio. It comes from the constrained optimization.
- Since this holds for all assets, it must hold for the tangency portfolio itself.

$$\frac{\mu_i - \mu_f}{\sigma_{i,T}} = \frac{\mu_T - \mu_f}{\sigma_T^2} = \frac{\mu_j - \mu_f}{\sigma_{j,T}}$$

Deriving the CAPM

The previous expression can be rearranged to derive a relation between an asset's expected return and risk,

$$\mu_i = \mu_f + \frac{\sigma_{i,T}}{\sigma_T^2}(\mu_T - \mu_f).$$

Remember: the relevant measure of risk is the **covariance** between the asset's returns and the returns on the tangency portfolio.
We typically define this risk as $\beta^{i,T}$:

$$\beta^{i,T} = \frac{\sigma_{i,T}}{\sigma_T^2}.$$

So the previous relationship becomes

$$\mu_i = \mu_f + \beta^{i,T}(\mu_T - \mu_f).$$

What does this expression say?

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.
- In the entire Mean-Variance Analysis, I did not assume anything about other people.

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.
- In the entire Mean-Variance Analysis, I did not assume anything about other people.
- But now, supposed everyone else also uses Mean-Variance Analysis

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.
- In the entire Mean-Variance Analysis, I did not assume anything about other people.
- But now, supposed everyone else also uses Mean-Variance Analysis
  - Everyone sees the same Mean-Variance diagram

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.
- In the entire Mean-Variance Analysis, I did not assume anything about other people.
- But now, supposed everyone else also uses Mean-Variance Analysis
  - Everyone sees the same Mean-Variance diagram
  - Everyone holds some weighted-average of the tangency portfolio and the risk-free asset.

Deriving the CAPM

- Recall, we showed that the optimal portfolio choice(s) would be a mix of the risk-free rate and the tangency portfolio.
- In the entire Mean-Variance Analysis, I did not assume anything about other people.
- But now, supposed everyone else also uses Mean-Variance Analysis
  - Everyone sees the same Mean-Variance diagram
  - Everyone holds some weighted-average of the tangency portfolio and the risk-free asset.
  - **Thus, the proportion between the risky assets is the same for everyone**

Example

Suppose there were only two stocks (IBM and GM) and three guys in the world. Also, suppose the tangency portfolio calls for 1/4 IBM and 3/4 GM.

- Guy 1: Holds \$3 of IBM and \$9 of GM and \$2 of risk-free asset.
- Guy 2: Holds \$5 of IBM and \$15 of GM and \$6 of risk-free asset.
- Guy 3: Holds \$20 IBM and \$60 of GM, \$0 of risk-free asset.

Example

Suppose there were only two stocks (IBM and GM) and three guys in the world. Also, suppose the tangency portfolio calls for 1/4 IBM and 3/4 GM.

- Guy 1: Holds \$3 of IBM and \$9 of GM and \$2 of risk-free asset.
- Guy 2: Holds \$5 of IBM and \$15 of GM and \$6 of risk-free asset.
- Guy 3: Holds \$20 IBM and \$60 of GM, \$0 of risk-free asset.

In total, there is \$28 of IBM and \$84 of GM. This is the entire market. We see that the total market demand of IBM to GM is also 1/4 to 3/4.

Example

Suppose there were only two stocks (IBM and GM) and three guys in the world. Also, suppose the tangency portfolio calls for 1/4 IBM and 3/4 GM.

- Guy 1: Holds \$3 of IBM and \$9 of GM and \$2 of risk-free asset.
- Guy 2: Holds \$5 of IBM and \$15 of GM and \$6 of risk-free asset.
- Guy 3: Holds \$20 IBM and \$60 of GM, \$0 of risk-free asset.

In total, there is \$28 of IBM and \$84 of GM. This is the entire market. We see that the total market demand of IBM to GM is also 1/4 to 3/4. How can market capitalizations be exactly this?

Example

Suppose there were only two stocks (IBM and GM) and three guys in the world. Also, suppose the tangency portfolio calls for 1/4 IBM and 3/4 GM.

- Guy 1: Holds \$3 of IBM and \$9 of GM and \$2 of risk-free asset.
- Guy 2: Holds \$5 of IBM and \$15 of GM and \$6 of risk-free asset.
- Guy 3: Holds \$20 IBM and \$60 of GM, \$0 of risk-free asset.

In total, there is \$28 of IBM and \$84 of GM. This is the entire market. We see that the total market demand of IBM to GM is also 1/4 to 3/4. How can market capitalizations be exactly this?

- Prices adjust so that supply equals demand!
- Suppose market cap of IBM and GM were both \$56.
  - People would only demand in 1/4 to 3/4 ratio, so markets will not clear.

The Key Observation

- So what does the market portfolio look like?

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?
- What other portfolio has similar portfolio weights in the market?

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?
- What other portfolio has similar portfolio weights in the market?
- **Conclusion: the returns on the market portfolio are EQUAL to the market returns on the tangency portfolio!!**

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?
- What other portfolio has similar portfolio weights in the market?
- **Conclusion: the returns on the market portfolio are EQUAL to the market returns on the tangency portfolio!!**
- Thus, our previous expression becomes

$$\mu_i = \mu_f + \beta^{i,M}(\mu_M - \mu_f),$$

where M denotes the market.

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?
- What other portfolio has similar portfolio weights in the market?
- **Conclusion: the returns on the market portfolio are EQUAL to the market returns on the tangency portfolio!!**
- Thus, our previous expression becomes

$$\mu_i = \mu_f + \beta^{i,M}(\mu_M - \mu_f),$$

  where M denotes the market.
- Why do we care?

The Key Observation

- So what does the market portfolio look like?
- What are the weights of the market portfolio?
- What other portfolio has similar portfolio weights in the market?
- **Conclusion: the returns on the market portfolio are EQUAL to the market returns on the tangency portfolio!!**
- Thus, our previous expression becomes

$$\mu_i = \mu_f + \beta^{i,M}(\mu_M - \mu_f),$$

  where M denotes the market.
- Why do we care?
  - The tangency portfolio is hard to find!!
  - These numbers are easier to find, so we can now calculate how expected returns for assets!

Brief Recap

- The CAPM says: we live in a world in which we all make decisions based on the same mean-variance diagram (this is a HUGE assumption, completely transparent markets)
- We all choose the same weights in our tangency portfolio, then we pick a balance of the risk-free rate and the tangency portfolio.
- Since we all purchase risky assets at the same ratio, the market participants will collectively move prices so that the market portfolio will be just like the tangency portfolio.

Consequences of CAPM in CAPM world

- We can calculate expected returns of any asset.
- CAPM argues that beta is the correct measure of risk
- What is the beta of the market with itself?
- Two assets which have the same covariance with the market must have the same expected return.

CAPM Intuition: Why are high beta stocks risky?

- Risky assets are assets that pay off in good times, and lose value in bad times.
- Hedges are assets that pay off in bad times, and lose in good times.
- CAPM identifies "good" and "bad" times using the returns of the market (market risk premium).
- As a portfolio holder, you want assets that pay off when your portfolio is doing badly.
- This is why risk is covariance with the tangency portfolio.
  - It is not about variance of the individual variance of an asset, but how covariance affects the returns of a diversified portfolio.
  - You get expected return to compensate you ONLY for covariance risk.

Next Week

- To-do: Test the CAPM. Does the model hold up against real world data?

Next Week

- To-do: Test the CAPM. Does the model hold up against real world data? (Hint: no)
- Next week: We will try to look at models that actually have some empirical validity, namely arbitrage pricing theory and multifactor models, though at the cost of intuition.