Computer Simulations and Risk Assessment – Lecture 9

Fall 2019

Brandeis International Business School

Course Information - Schedule

Class Date	Text Chapters
Aug. 30, 2019 – L1	• Course Introduction/Python Installation • Introduction to Quantitative Finance Career • Python basics
Sep. 6, 2019 – L2	• Advanced Python Topics
Sep. 13, 2019 – L3	• Advanced Python Topics
Sep. 20, 2019 – L4	• Sourcing and handling Data • Stylized financial data analysis using Python
Sep. 27, 2019 – L5	• Value at Risk
Oct. 4, 2019 – L6	• Conditional Value at Risk (Expected Shortfall) + Mid-term Review
Oct. 11, 2019	• Mid-term
Oct. 18, 2019 – L7	• Modeling Volatility I
Oct. 25, 2019 – L8	• Modeling Volatility II
Nov. 1, 2019 – L9	• Practical application case Studies I
Nov. 8, 2019 – L10	• Practical application case Studies II
Nov. 15, 2019 – L11	• Back Testing + Conditional risk prediction
Nov. 22, 2019 – L12	• Research project presentation
Dec. 6, 2019 – L13	• Final Review

Lecture 9 – Outline

Optimization using Python

- Optimization setup
- How to do it in Python

Sample risk optimization strategies

- Designing a ETF/index fund through tracking error minimizatio

Part I

Optimization using Python

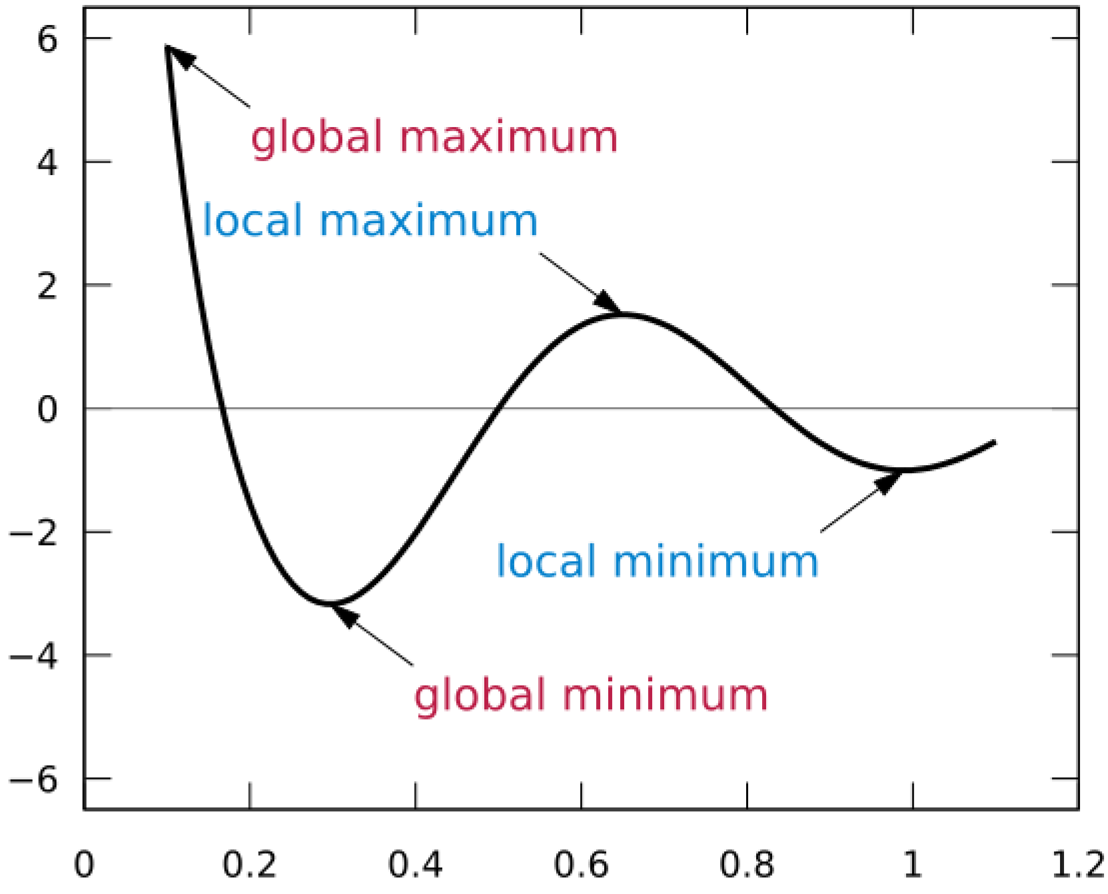
- **Optimization setup**
- **How to do it in Python**

Optimization formulation for finance

- Three parts in optimization formulation
  1. Define the decision variables (**w**) – in most cases, the weights of the assets in your portfolio

     Var?Return?
  2. Define and calculate an objective function, which is a function of the decision variables. The objective function can be maximized or minimized depends on its content. Example: portfolio returns to be maximized or portfolio risk that is to be minimized
  3. A set of constraints. For example, no shorting means no weights below zero or no leverage means sum of weights not above 100%

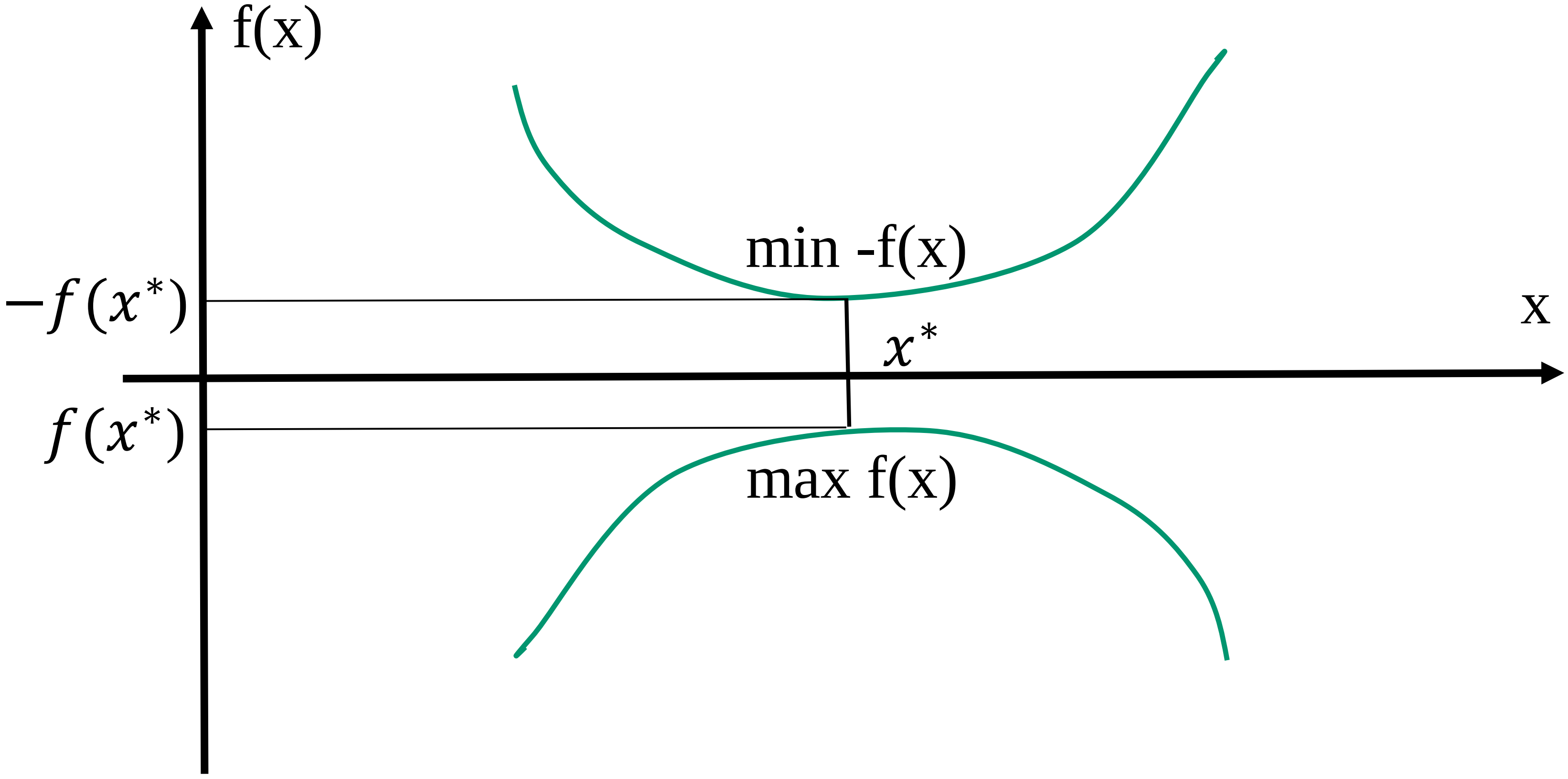
Local vs. global maximum (minimum)



Brandeis

Maximization vs. minimization

- Maximization of the objective function is equivalent to minimizing -1 multiplies the same objective function



Brandeis

Types of optimization problems

- Linear
  - The objective and the constraints are all linear expressions
- Nonlinear
  - Either the objective or the constraints (or both) are nonlinear expressions
  - Local optima
  - Some nonlinear classes of problems are "easier"
    - Quadratic objective, linear constraints
- Integer / Binary
  - The decision variables are integer/binary numbers
- Mixed integer
  - Some decision variables are integers, others are continuous numbers

Two formulations of the risk optimization problem

- Minimizing tracking error
- Benchmark exposure and tracking error constraints

$$\min_{\boldsymbol{w}} (\mathbf{w} - \mathbf{w}_b)'\boldsymbol{\Sigma}\,(\mathbf{w} - \mathbf{w}_b) \quad \text{or}$$

$$(\mathbf{w} - \mathbf{w}_b)'\boldsymbol{\Sigma}\,(\mathbf{w} - \mathbf{w}_b) \leq \sigma^2_{\text{TE,target}}$$

- Risk-parity

$$\min_{\boldsymbol{w}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \left(w_i \frac{(\boldsymbol{\Sigma w})_i}{\sigma_{\boldsymbol{p}}^{\mathbf{2}}} - b_i \right)^{\mathbf{2}}$$

Optimization Problem Setup

1. Choose a optimization solver/algorithm
   – Choose the most appropriate solver/algorithm
2. Define/Write objective function
   – Define the function to minimize or maximize, representing your problem
3. Define/Write constraints
   – Provide bounds, linear constraints, and nonlinear constraints
4. Set Options
   – Set optimization options
5. Putting it all together

Step 1: How to choose a solver/algorithm

- By the objective type and the constraint type
  - Objective type: linear, quadratic, smooth nonlinear etc.
- By constraint type
  - bounds, linear, discrete etc.

Introduction to Python's scipy.optimize

- The scipy.optimize package provides several commonly used optimization algorithms
  - https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
  - https://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize
- The available algorithm (solvers) include:
  - Unconstrained and constrained minimization of multivariate scalar functions (minimize) using a variety of algorithms (e.g. BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP)
  - Global (brute-force) optimization routines (e.g. basinhopping, differential_evolution)
  - Least-squares minimization (least_squares) and curve fitting (curve_fit) algorithms
  - Scalar univariate functions minimizers (minimize_scalar) and root finders (newton)
  - Multivariate equation system solvers (root) using a variety of algorithms (e.g. hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov).

Example: Python's *scipy.optimize.minimize* function

```
optimized = optimize.minimize(obj, W, (R, C),
            method='SLSQP', constraints=c_, bounds=b_,
            options={'ftol':1e-8, 'maxiter': 1000, 'disp': False})
```

scipy.optimize.minimize(*fun*, *x0*, *args=()*, *method=None*, *jac=None*, *hess=None*, *hessp=None*, *bounds=None*, *constraints=()*, *tol=None*, *callback=None*, *options=None***)**

Description of parameter:

- obj – objective function
- W – weights vector
- (R, C) - Extra arguments (tuple data type) passed to the objective function and its derivatives. In our case, the return vector and covariance matrix
- Method = 'SLSQP' - Minimize a scalar function of one or more variables using Sequential Least Squares Programming (SLSQP)
- Constraints = **c_** - define constraints on weights, we will see example next
- Bounds = **b_** - lower/higher bounds for weights , we will see example next
- Options – setting for the optimization problem such as convergence criterion

Type of objective functions and corresponding solvers

Objective Type	Solvers/Methods
Scalar	minimize minimize_scalar
least squares & curve fitting	least_squares curve_fit
Multivariate equation solving	root
Linear programming	linprog
Global minimum of functions	Basinhopping Brute differential_evolution

Step 2: How to choose/write the objective function

- Define what are the decision variables
  - E.g., for our case, asset weights used to form a portfolio
- Define the objective: e.g., what are you maximizing or minimizing as a function of the decision variables
  - E.g., minimize portfolio variance in our case

Linear objective function example

- Objective: Maximize expected return $E[r_P]$

where

$E[r_P] = w(1)*E[r(1)] + w(2)*E[r(2)] +\ldots$ is a <u>linear function of the decision variable</u>, the weights of the assets considered w(i), i=1…N

- E.g., in the sample python code risk_opt.py we define a function called obj_ret to calculate return of a portfolio, given the weights and returns of each assets:
- Note, because we are using a minimizing solver, we change the objective calculation to negative portfolio returns

```
def obj_ret(W, R, C):
     return(-port_ret(W,R))

def port_ret(W,R):
    return(np.dot(R,W))
```

Higher order objective function example

- Objective: Minimize portfolio variance:

$$\boldsymbol{w}'\Sigma\boldsymbol{w}$$

where **w** is the weights (decision variable) and **Σ** is the covariance matrix

- The objective function is of quadratic order of the decision variable (weights): e.g., one of the terms is $-w_1^2\sigma_1^2$
- E.g., in sample python code risk_opt.py we define a function called obj_var to calculate the variance of a portfolio, given the weights of each assets and the covariance matrix:

```
def obj_var(W, R, C):
    return(np.dot(np.dot(W, C), W))
```

Step 3: Defining/writing constraints

1. Bounds
2. Linear equalities
3. Linear inequalities
4. Nonlinear equalities
5. Nonlinear inequalities

Bounds constraints

- Lower and upper bounds limit the components of the decision variable **w**
- Bounds on the decision variable help obtain faster and more reliable solutions
- Give bounds as a list for each of the decision variables. E.g., in our case, weights for each of the assets are bounded between 0 and 1
- Example from our sample code risk_opt.py:

```
if allow_short == False :
    b_ = [(0.,1.) for i in range(len(R))]   # No leverage, no shorting
else:
    b_ = [(-1.,1.) for i in range(len(R))] # allows leverage and shorting
```

Defining equality/inequality constraints in Python

- Defining constraints in Python, through the optimize.minimize method:
  - Define equality/inequality constraints using data type dictionary or sequence of dictionary

constraints : *dict or sequence of dict, optional*

Constraints definition (only for COBYLA and SLSQP). Each constraint is defined in a dictionary with fields:

type : *str*

Constraint type: 'eq' for equality, 'ineq' for inequality.

fun : *callable*

The function defining the constraint.

jac : *callable, optional*

The Jacobian of *fun* (only for SLSQP).

args : *sequence, optional*

Extra arguments to be passed to the function and Jacobian.

Equality constraint means that the constraint function result is to be zero whereas inequality means that it is to be non-negative. Note that COBYLA only supports inequality constraints.

Linear equality constraints

- Linear equalities have the form ***f(w)***, which represents *the linear equality constraint of f(**w**)=0*
- Example – two linear equality constraints:
  1. Sum of weights need to add up to 1
  2. Portfolio returns equals predetermined value

Key word 'type' followed by 'eq' – telling python this defines an equality constraint

Key word 'fun' followed definition of the function f(**w**) = sum(**w**)-1

```
c_ = ({'type':'eq', 'fun': lambda W: sum(W)-1. }, # Sum of weights = 100%
      {'type':'eq', 'fun': lambda W: port_ret(W, R)-r })    # return = required return
```

function f(**w**) = **w'μ**-r

Linear inequality constraints

- Linear inequality constraints have the form *f*(**w**) ≥ 0. *where f is the function with the decision variable* **w** *as its inputs*
- For example, suppose that you have the following linear inequalities as constraints:

$$w_1 + w_2 \geq 0.5,$$

The constraint can then be define in Python by:

```
# inequality constraints
c_ineq_ = {'type': 'ineq', 'fun': constrain_ineq}

def constrain_ineq(W):
    return W[0]+W[1]-0.5
```

- Note by default, Python assumes the inequality relationship is a **≥ relationship**

Step 4: Define Optimization Option

- Options are a way of combining a set of name-value pairs. They are useful because they allow you to:
  - Tune or modify the optimization process.
  - Select extra features, such as output functions and plot functions.
  - Set convergence criterion etc.
- E.g., in sample python code risk_opt.py we set the options for the optimization solver below to
  - Set Precision goal for the value of the function in the stopping criterion to 1e-8
  - Set maximum number of iterations to 10000
  - Choose not to display convergence messages

```
options={'ftol':1e-8, 'maxiter': 1000, 'disp': False}
```

Step 5: Putting it all together

Define the objective function, by passing the handle of the objective function

Calling the optimize.minimize scalar function solver

Define the lower and upper bounds

```
optimized = optimize.minimize(obj, W, (R, C),
            method='SLSQP', constraints=c_, bounds=b_,
            options={'ftol':1e-8, 'maxiter': 1000, 'disp': False})
```

Define optimization options

Define constraints

Sample risk optimization strategies

- Designing a ETF/index fund through tracking error minimization
- Design of Risk-Parity strategies

Risk Allocation Strategies

- Risk allocation / risk budgeting investment strategies aim to allocate assets based on the assets' proportion of risk contributions to the overall total/relative risk of the portfolio
- Examples
  - Minimizing tracking error (relative risk to benchmark)
  - Risk-parity (Equal risk contribution by components of a portfolio)
  - Targeted risk contribution by components of a portfolio
  - Risk-factor beta targeting
  - Etc.

Minimize Tracking Error Strategy

- Minimize tracking error

$$\min_{\boldsymbol{w}} (\mathbf{W}-\mathbf{W}_b)'\boldsymbol{\Sigma}(\mathbf{W}-\mathbf{W}_b) \quad \text{or}$$

$$(\mathbf{W}-\mathbf{W}_b)'\boldsymbol{\Sigma}(\mathbf{W}-\mathbf{W}_b) \le \sigma^2_{\text{TE,target}}$$

- Where
  - $\boldsymbol{\Sigma}$ is forecasted next period covariance matrix
  - $\mathbf{W}_b$ is the benchmark weights for the assets being considered
- Constraints:
  - Only use the some of the assets in the benchmark (e.g., only top 20 weighted stocks)
  - Transaction costs limits

Calculation of the Risk Measure

- We are trying to minimize TE, which is a function of Covariance. We need to forecast it to develop our strategy
- Can try at least three different ways to calculate the covariance matrix
  - MA
  - EWMA
  - Conditional Variance
  - GARCH (but unfortunately we can't do it for multi-variate in Python)

Minimize TE Strategy – ETF Example

- Dow Jones Industry index, with 30 components
- Create an ETF (exchange traded fund) to track the index

Number	Company	Symbol	Weight	Price
1	Boeing Company	BA	9.417522	371.95
2	UnitedHealth Group Incorporated	UNH	6.783928	266.04
3	Goldman Sachs Group Inc.	GS	5.837792	224.36
4	Apple Inc.	AAPL	5.766274	225.95
5	3M Company	MMM	5.385873	211.19
6	Home Depot Inc.	HD	5.30205	207.15
7	McDonald's Corporation	MCD	4.268761	167.29
8	Caterpillar Inc.	CAT	3.908353	152.35
9	International Business Machines Corporation	IBM	3.883488	151.35
10	Visa Inc. Class A	V	3.845551	150.04
11	United Technologies Corporation	UTX	3.580499	139.81
12	Johnson & Johnson	JNJ	3.543074	138.27
13	Travelers Companies Inc.	TRV	3.309808	129.71
14	Chevron Corporation	CVX	3.139345	122.28
15	Walt Disney Company	DIS	2.974521	117.05
16	JPMorgan Chase & Co.	JPM	2.935558	112.90
17	Microsoft Corporation	MSFT	2.932	114.44
18	American Express Company	AXP	2.764582	107.49
19	Walmart Inc.	WMT	2.412889	94.05
20	Exxon Mobil Corporation	XOM	2.198592	85.05
21	NIKE Inc. Class B	NKE	2.167063	84.72
22	Procter & Gamble Company	PG	2.123999	83.23
23	Walgreens Boots Alliance Inc	WBA	1.892015	72.90
24	Merck & Co. Inc.	MRK	1.812807	70.94
25	DowDuPont Inc.	DWDP	1.674642	64.37
26	Verizon Communications Inc.	VZ	1.373704	53.48
27	Cisco Systems Inc.	CSCO	1.238871	48.73
28	Coca-Cola Company	KO	1.179401	46.37
29	Intel Corporation	INTC	1.176069	47.35
30	Pfizer Inc.	PFE	1.125314	43.98

Brandeis

Minimize TE Strategy – ETF Example

```
optimized = optimize.minimize(obj_te, W, (W_Bench, C),
            method='SLSQP', constraints=c_, bounds=b_,
            options={'ftol':1e-10, 'maxiter': 1000000, 'disp': False})
```

```
def obj_te(W, W_Bench, C):
    wts_active = W - W_Bench
    return(np.sqrt(np.transpose(wts_active)@C@wts_active))
```

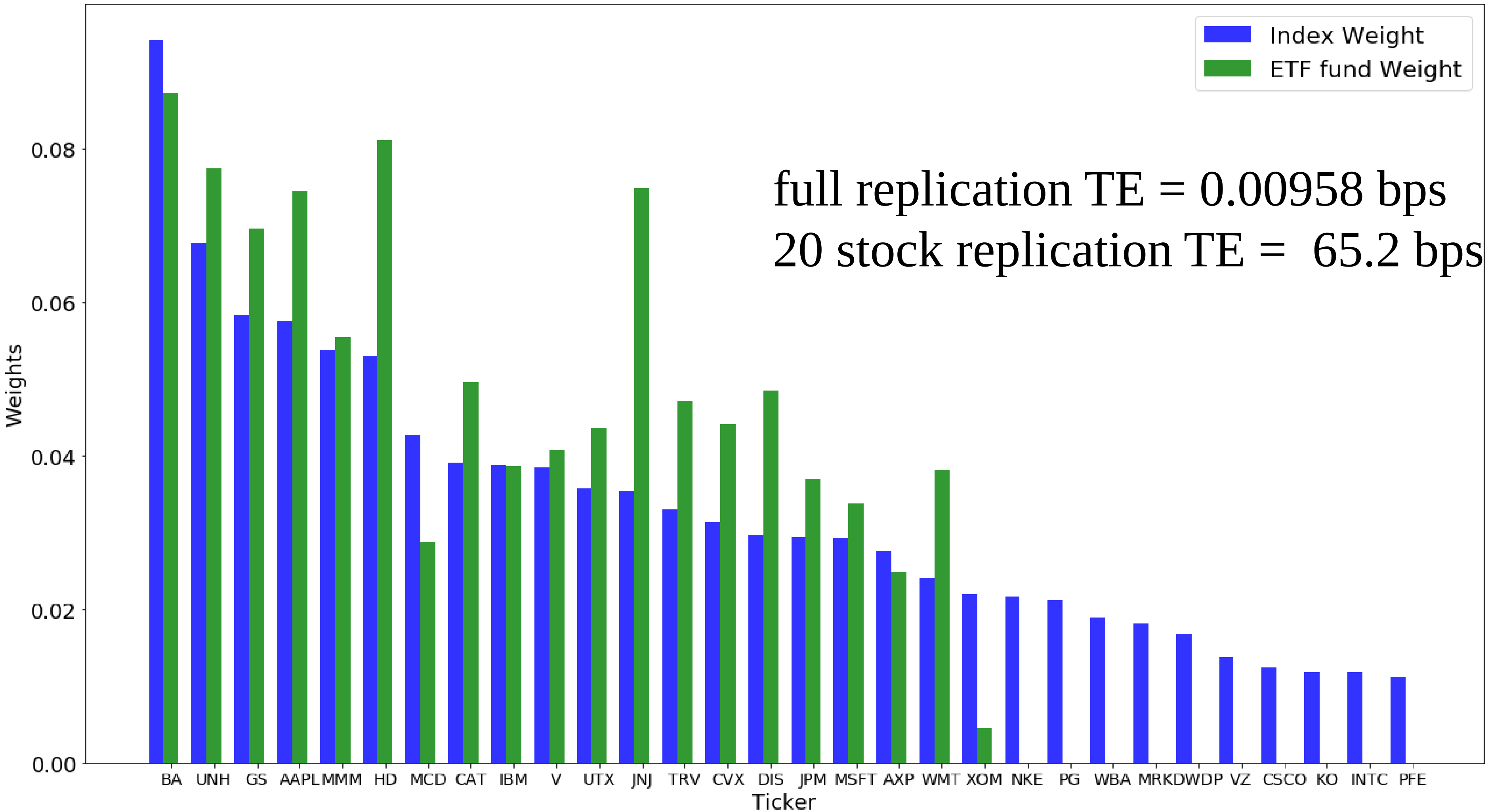
Minimize the tracking error

Constraints:

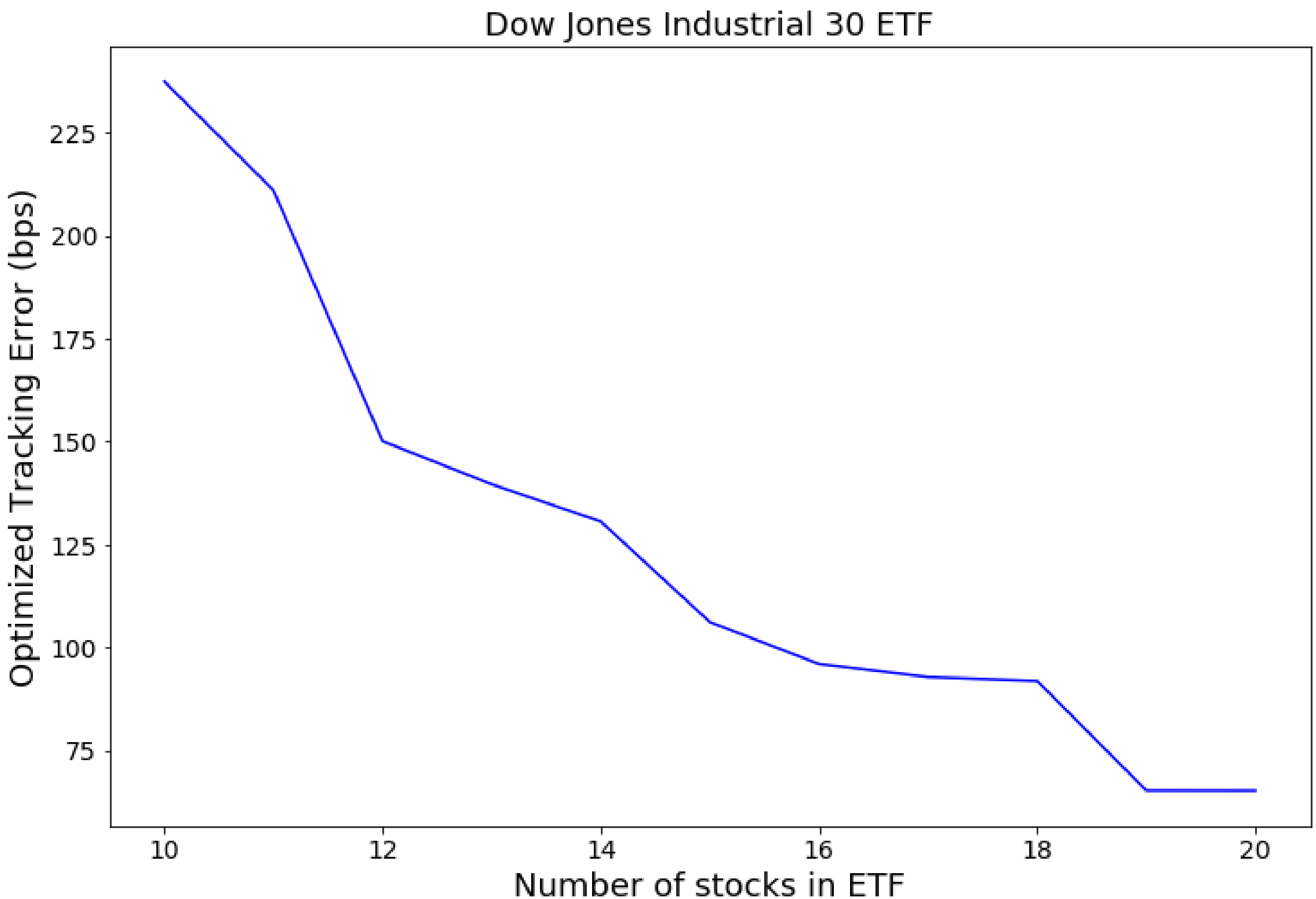
Only use the top 15 weighted stocks

```
#
# Test case - use only the top five stocks with highest index weights
#
num_topwtstock_2include = 15 #good with 10, 15, 20
b1a_ = [(0.0,1.0) for i in range(num_topwtstock_2include)]  # no shorting
b1b_ = [(0.0,0.0) for i in range(num_topwtstock_2include,num_stock)]
b1_ = b1a_ + b1b_
#b1_[num_topwtstock_2include:-1] = (0.0,0.0)
c1_ = ({'type':'eq', 'fun': lambda W: sum(W)-1. })   # Sum of active weights = 100%
wts_min_trackingerror2 = riskopt.opt_min_te(wts_AllStock_DJ, cov_end_annual, b1_, c1_)
```

Minimize TE Strategy – Optimized Weights



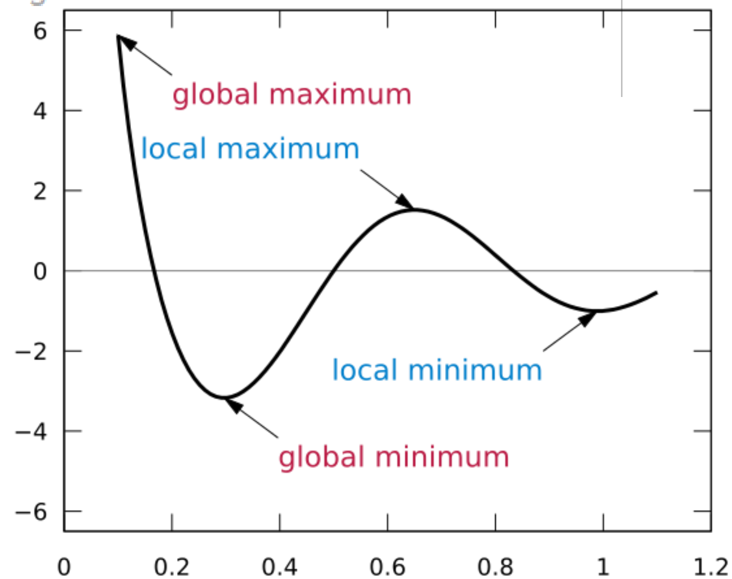
Minimize TE Strategy – Optimized Weights vs. # of stocks



Address the issue of Local vs. global minimum

- How to get the global minimum instead of a local one?
  - Try different initial guesses
  - Try different solver, including global optimization solver

```python
# change the initial guess to help test whether we find the global optimal
guess = 2
#W = rand_weights(n) # start with random weights
if guess==1:
    W = rand_weights(n) # start with random weights
elif guess==2:
    W = W_Bench # Use Bench weight as initial guess
else:
    W = 1/n*np.ones([n,1])
```



Brandeis

Appendix

Incorporate constraints on the #of stocks to include

- Suppose, for example, that we would like to limit the number of stocks in the portfolio to 2.
- How can we tell the solver to optimize the portfolio variance by considering the "best" two out of the three stocks?
  - Introduce additional *binary* variables, one for each asset i, i=1,2,3:

$$\delta_i = 1 \quad \text{if} \quad w_i \neq 0$$

$$\delta_i = 0 \quad \text{if} \quad w_i = 0$$

  - …and request that

$$\sum_{i=1}^{3} \delta_i = 2$$

Incorporate constraints on the # of stocks, Ctd

- In the more general case of N assets of which we want to keep only K in the portfolio, the portfolio optimization formulation becomes

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}'\Sigma\mathbf{w} \\ s.t. \quad & \mathbf{w}'E[\mathbf{R}] \geq \mu_{\text{target}} \\ & \mathbf{w}'\mathbf{e} = 1 \\ & \sum_{i=1}^{N} \delta_i = K \\ & \left.\begin{array}{l} 0 \leq w_i \leq \delta_i, \quad i = 1, \ldots, N \\ \delta_i \text{ binary} \end{array}\right\} \end{aligned}$$

These constraints are necessary to ensure that δ_i cannot be 0 if the weight w_i of asset i is greater than 0.

Incorporate constraints on the # of stocks, Ctd

- This requires **integer modeling**
  - Binary (or, more generally, integer) variables are specified in Solver by declaring them as "changing cells" and then adding the constraints that these changing cells should be "bin" (for "binary") or "int" (for "integer").
- Optimization problems with integer variables are (a lot!) harder than optimization problems with continuous variables.

Python: Lambda Functions

- Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda":
- The benefit is that you have the flexibility to change the content of the function each time you use it.
- a) A code shows the difference between a normal function definition (f) and a Lambda function definition, and b) how it is used in our optimization code is shown below

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

```
c_ = ({'type':'eq', 'fun': lambda W: sum(W)-1. }, # Sum of weights = 100%
      {'type':'eq', 'fun': lambda W: port_ret(W, R)-r })   # return = required return
```

Brandeis