FMO6 — Web: https://tinyurl.com/ycaloqk6 Polls: https://pollev.com/johnarmstron561

Lecture 10 - Optimization

Dr John Armstrong

King's College London

August 22, 2020

Introduction

Outline

- "Modern" Portfolio Theory
  - Markowitz's theory
  - The efficient frontier
  - `quadprog` for quadratic optimization
- Calibrating models
  - The jump diffusion model
  - Incomplete market models
  - `fminunc` for unconstrained, nonlinear optimization
- Dynamic optimization
  - Revision of delta hedging strategy
  - The Galerkin method
  - `fmincon` for constrained, nonlinear optimization

Strange terminology

- "Programming" is another term for optimization.
  - Linear programming = solving linear optimization problems
  - Quadratic programming = solving nonlinear optimization problems
  - "Linear programming" is so-called because it was originally used for optimizing military questions, which was phrased as how best to conduct military programs.
- "Operations Research", is, according to Wikipedia, a discipline that deals with the application of advanced analytical methods to help make better decisions.
  - The name comes from military operations
  - If you apply optimization to real world business problems, you are performing operations research.

Uniqueness

Find x that minimizes $f(x)$ subject to the constraint $x \in X$.

- If x exists it may not be unique. This is not a problem it just means that two equally good solutions can be found.
- When finding numerical solutions, there may be a variety of equally good solutions found
- When testing the correctness of your optimization, you should see if two competing methods both give the same value for $f(x)$ rather than seeing if they give the same value for x.

Convexity

- For convex functions f on convex domains X, a local minimum is a global minimum.
- For this reason it is much easier to minimize a convex function numerically than a general function.
- Convex problems have lots of nice properties, e.g. a theory of "dual problems". We won't cover this in this course.

Algorithms

- There are many optimization algorithms. We will let MATLAB choose the best algorithm for us. For large/difficult problems one may wish to choose the algorithm carefully.
- Different algorithms exist for small, medium, large and ridiculously large problems.
- Specialist algorithms exist for certain problems (e.g. quadratic and linear problems)
- Most algorithms assume f is smooth, but algorithms do exist for non-smooth, convex f.
- We will be interested in local algorithms that find local minima. This is obviously not a problem for convex problems. Global algorithms are, inevitably, slower.

Modern Portfolio Theory

Modern Portfolio Theory

Theory developed my H. Markowitz in 1950's. Not very modern!

- There are n assets whose returns over some fixed time period T are random variables R_i.
- $$R_i = \frac{\text{price of asset } i \text{ at time } T \text{ - current price of asset } i}{\text{current price of asset } i}$$
- The random variables R_i are assumed to have a known mean μ and covariance matrix Σ.
- We wish to invest a fixed amount P over the time period T in a fixed portfolio of the assets.
- The portfolio is determined by choosing the weights w_i to assign to each asset with $\sum w_i = 1$.
- We are allowed to short sell so the w_i may take any real values.

Optimization problem

- Suppose we wish to have an expected return on our portfolio of r, what portfolio should we choose to minimize the standard deviation of the portfolio?
- If $\underline{w}$ is the vector of weights, then the return of the portfolio has a distribution with mean:

$$\underline{\mu}^T \underline{w}$$

and variance:

$$\underline{w}^T \Sigma \underline{w}$$

- So problem can be stated as:
  - Minimize $\underline{w}^T \Sigma \underline{w}$
  - Subject to the constraints $\sum_i w_i = 1$ and $\underline{\mu}^T \underline{w} = r$.
  - Σ is a positive definite matrix, $\underline{\mu}$ is a vector.

Quadratic optimization problem

- Minimize
$$\frac{1}{2}\underline{x}^T H \underline{x} + f^T \underline{x}$$
- Optionally subject to constraints of the form:
$$A\underline{x} \leq \underline{b}$$
$$A'\underline{x} = \underline{b}'$$
$$\underline{l} \leq \underline{x} \leq \underline{u}$$
For some matrices A and A' and vectors $vecb$, $\underline{b}'$, $\underline{l}$ and $\underline{u}$.
- We write $\underline{x} \leq \underline{y}$ if every entry of the vector x is less than or equal to the corresponding entry in y.
- The vectors $\underline{l}$ and $\underline{u}$ may contain the values $-\infty$ and $+\infty$ respectively.
- To solve this problem simply use `quadprog`

MATLAB quadprog

For a problem as defined on the previous slide with $A' =$ Aeq and $b' =$ beq

```
[x,fVal,exitFlag]=quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options);
```

- x is the optimal value of the vector
- *fVal* is the value of $\frac{1}{2}\underline{x}^T H \underline{x} + \underline{f}^T \underline{x}$
- exitFlag indicates whether the optimization worked. You *must* check this!
- You can use the empty array [] where a constraint is not needed.
- The options parameter allows you to fine tune the optimization if you wish. It can be omitted.
- quadprog standard for "quadratic programming"

Markowitz problem as quadratic optimization

- The condition $\sum_i w_i = 1$ and $\underline{\mu}^T \underline{w} = r$ can be rewritten $A'\underline{w} = \underline{b'}$ where

$$A' = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ \mu_1 & \mu_2 & \mu_3 & \dots & \mu_n \end{pmatrix}, \quad b' = \begin{pmatrix} 1 \\ r \end{pmatrix}$$

- Thus the Markowitz problem is just a special case of quadratic programming.

MATLAB implementation

```
function [ ret, variance, w ] = markowitzOptimizeRet(...
      r, mu, sigma )

H = sigma;
n = size(sigma,1);
f = zeros(n,1);
Aeq = [ ones(1,n); mu];
beq = [1; r];

[w,~,exitFlag] = quadprog(H,f,[],[],Aeq,beq,[],[],[]);
assert(exitFlag>0);
ret = mu * w;
variance = w' * sigma * w;

end
```

Remarks

- If you run the code on the previous slide, MATLAB prints out a lot of messages and a warning.
- To switch off the messages you can customize the options
- To switch off the warning you use the `warning` command to indicate that you aren't interested in the specific warning message.

Quieter MATLAB implementation

```
function [ ret, variance, w ] = markowitzOptimizeRet( ...
    r, mu, sigma )
warning('off','optim:quadprog:WillRunDiffAlg');


H = sigma;
n = size(sigma,1);
f = zeros(n,1);
Aeq = [ ones(1,n); mu];
beq = [1; r];

options = optimset('quadprog');
options = optimset(options,'Display','off');
[w,~,exitFlag]=...
    quadprog(H,f,[],[],Aeq,beq,[],[],[],options);
assert(exitFlag>0);
ret = mu * w;
variance = w' * sigma * w;

end
```

Creating options

- To create a set of options first call `optimset` passing in a single parameter, the name of the optimization routine you will call. This returns an `options` object.

```
options = optimset('quadprog');
```

- Call `optimset` a second time to modify the options. This time you should pass in your `options` object and then a set of key/value pairs that indicate what options you would like to set. We're setting the option `Display` to `off`.

```
options = optimset(options,'Display','off');
```

The function `optimset` returns the new modified options object.

Using the options

- You can now pass this customized set of `options` object to the optimization function.

```
[w,~,exitFlag]=...
    quadprog(H,f,[],[],Aeq,beq,[],[],[],options);
```

- The reason for this procedure is to make sure that you choose the default values for all options except the ones you wish to customize.

Available options

When optimizing in MATLAB you can use options to set things like:

- How much information to print during the calculation (`Display`)
- The actual algorithm to use (`Algorithm`)
- How accurate the answer needs to be
- The maximum number of steps to perform of the algorithm before giving in
- etc.
- Consult the MATLAB documentation for details of the available options and their settings.

Portfolio optimization example

- We wish to select an optimal portfolio of FTSE 100 stocks.
- The file `ukx.xslx` contains data downloaded from Bloomberg for the FTSE 100 index.
- It contains weekly prices for each of entry of the FTSE 100 since the year 2000
  - For the vast majority of FTSE 100 companies, we have a complete series of data
  - For a couple of companies there are missing entries. We will exclude those from our Portfolio.
- For the remaining companies, we'd like to compute the weekly returns and hence estimate the sample covariance matrix Σ and the mean weekly vector $\underline{\mu}$ for the FTSE 100.
- We then assume that over the next week, the stock returns follow a multivariate normal distribution with parameters $\underline{\mu}$ and Σ.

Reading the Excel data

```
function data = ukxData( nSecurities )

% Read the raw data from the excel file
bloombergData = xlsread( 'ukx.xlsx', 'A3:KP736' );

% n is the number of securities that we've read
n = min( nSecurities, floor(size(bloombergData,2)/3));

% Now eliminate the empty columns and any data
% for a security where
% we don't have a full history of returns
index = 1;
for i=1:n
    col = bloombergData(:,(i-1)*3+1);
    if (~isnan(col(end)))
        data(:,index)=col;
        index = index+1;
    end
end

end
```

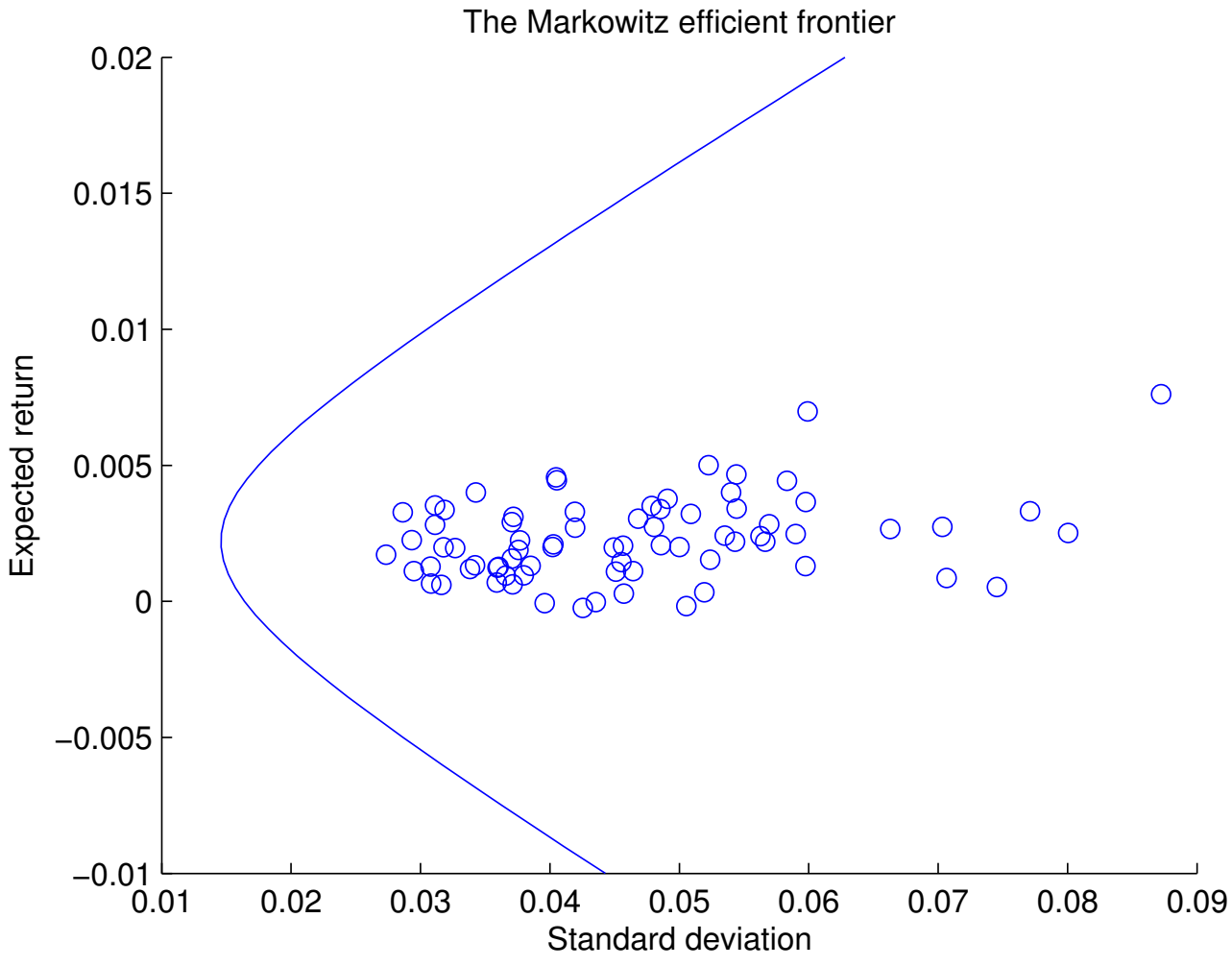
Scatter plot of UKX returns and sds

```
data = ukxData( nSecurities );
returns = (data(2:end,:) - data(1:end-1,:))./data(1:end-1,:);

% We assume that historical returns allow us to estimate
% expected return (mu) and covariance (sigma)
sigma = cov(returns);
mu = mean( returns );

% Plot a scatter plot of the return for each consitutent
sds = sqrt( diag( sigma ));
scatter( sds, mu );
xlabel('Standard deviation');
ylabel('Expected return');
```

Efficient frontier using all stocks

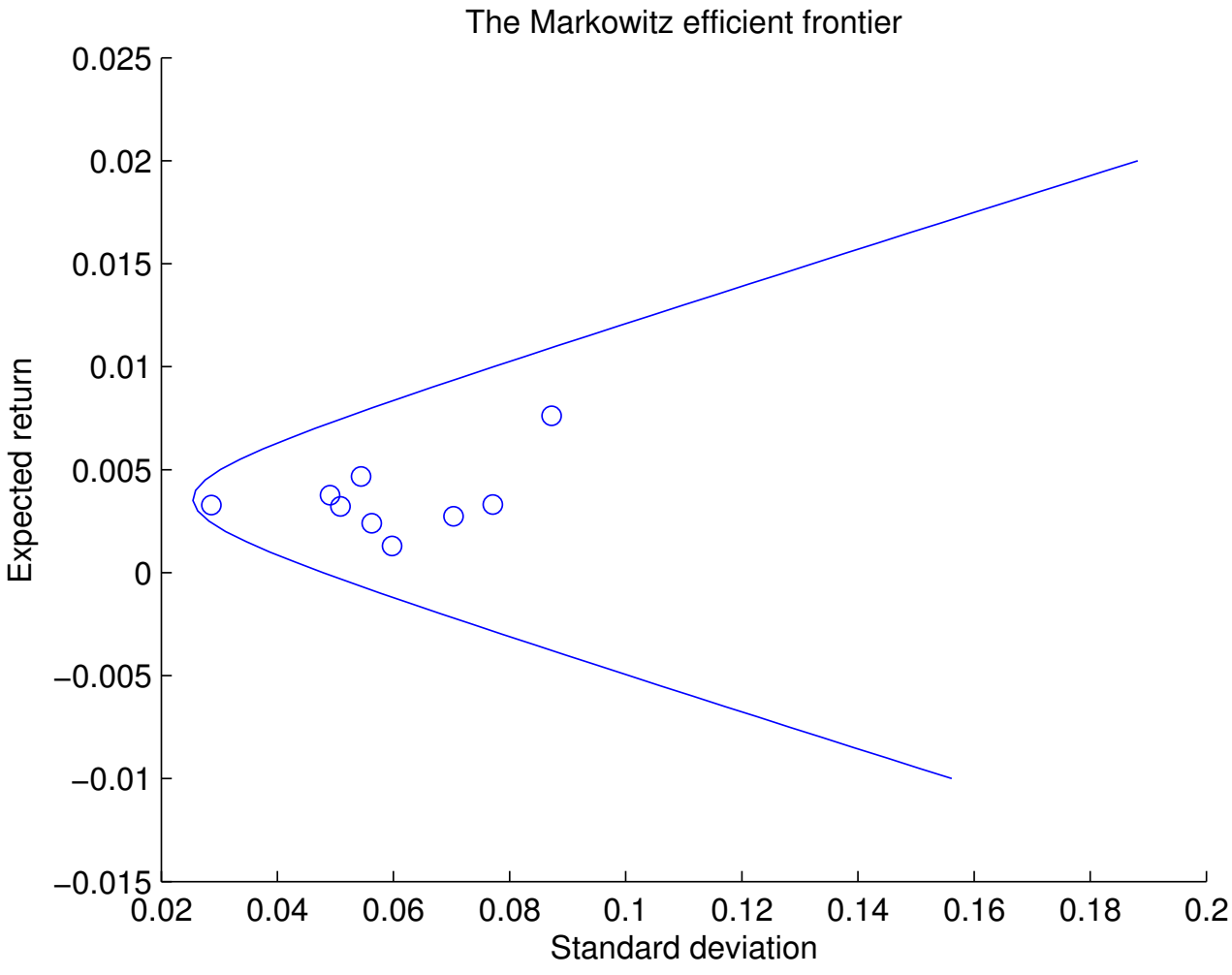


- Given an expectation r, there is a corresponding minimum standard deviation.
- As r varies, this traces out the *efficient frontier*.
- The points represent the performance of individual stocks in the FTSE 100
- They all lie inside the efficient frontier.

Plotting the efficient frontier

```
% Now compute the efficient frontier and plot it
r = -0.01:0.0005:0.02;
frontierX = zeros(length(r),1);
frontierY = zeros(length(r),1);
for i=1:length(r);
    [ret,var]=markowitzOptimizeRet( r(i), mu, sigma );
    frontierX(i)=sqrt(var);
    frontierY(i)=ret;
end
hold on;
plot( frontierX, frontierY );
title('The Markowitz efficient frontier');
hold off;
```

Efficient frontier with 10 stocks



Remarks on Markowitz theory

- The Markowitz model only makes sense if you think standard deviation is a good measure of risk. For normal distributions, this is uncontroversial. For more complex distributions more general utility optimization makes more sense. This is why normally distributed returns are often assumed.
- For large markets, the assumptions are very unrealistic
  - Precisely known drift and covariance matrix for returns
  - Unlimited buying and selling
- It tends to produce unreasonable portfolios that exploit quirks in the data.
  - Example: suppose X and Y are two highly correlated stocks with the same price. Markowitz optimization might construct a portfolio consisting of 1000 units of X and sell 999 units of Y.
- Despite its shortcomings, it is enormously influential and is where the idea of portfolio optimization originated.

Feedback Feedback Feedback

- It's module feedback season, please take a moment to give feedback on my course. You should be able to see the link on Keats.
- Thanks to everyone who has already given feedback.

Calibrating pricing models

Calibration

- The Black–Scholes model does not fit the data:
  - Stock returns have fat tails
  - There is a volatility smile.
  - (Question: what is implied volatility?)
- Many other models have been proposed, many of which are *incomplete* market models.
  - Stochastic volatility models: the volatility itself follows a stochastic process. E.g. the Heston model.
  - Jump diffusion models: in addition to the usual white noise that changes the stock price continuously, there is also the possibility of instantaneous jumps in the stock price.

Incomplete market models

- There is more than one source of randomness effecting the stock price, you can only hedge one source of risk by trading in the underlying
- As a result there is more than one equivalent martingale measure
- When using an incomplete market model:
  - Choose an equivalent martingale $\mathbb{Q}$-measure
  - Use this to model both the underlying *and* vanilla option prices as discounted expectations for this measure.
- Prices obtained in this way will be arbitrage free. If option prices and the stock follow the model, it will be possible to find dynamic replication strategies for exotics by trading in both the underlying and in vanilla options.

A jump diffusion model

- We suppose that the risk neutral price process follows:

$$\frac{\mathrm{d}S_t}{S_t} = \mu \mathrm{d}t + \sigma \mathrm{d}W_t$$

  between jumps.
- We assume that the occurence of jumps is a Poisson process with intensity λ.
- J is a constant representing the size of a jump. When a jump event event occurs, the price jumps from S_t to JS_t. Thus $J < 1$ represents a fixed downward jump and $J > 1$ represents a fixed size upward jump.
- For the discounted process to be a martingale we need $\mu = -(J-1)\lambda + r$
- Fixed size jumps with constant intensity don't give a very plausible financial model, we've chosen this model because it gives a simple example of an incomplete model.

Option prices

- We now assume that options can be priced as expectations of this model.
- It is fairly easy to compute that the price of a call option is:

$$\sum_{j=0}^{\infty} e^{-\lambda T} \frac{(\lambda T)^j}{j!} \mathrm{BS}(K, T, S_0 e^{(\mu - r)T} J^j, r, \sigma)$$

- Here $\mathrm{BS}(K, T, S_0, r, \sigma)$ is the usual Black Scholes pricing formula.
- See Joshi "The concepts and practice of mathematical finance" chapter 15 for the derivation and a more thorough discussion of incomplete market models.

The relevance of optimization

- How do we choose the parameters σ, λ and J?
- By calibrating the model to market option prices.
- i.e. we find the values that give the "best fit" to market prices.
- Choose some metric to measure the error of the fit, use optimization algorithm to minimize the error.
- We will define the error to be the sum of the squared errors when comparing the predicted price of an option with the market price. Using squared errors rather than absolute values ensures our error function is smooth.

The optimization problem

- We have n exchange traded call options all with maturity T. Option i has strike K_i.
- Let C_i denote the market price of the option i.
- Let $C_i'(\sigma, \lambda, J)$ denote the jump-diffusion predicted price with the given parameters.
- Define:

$$\text{error}(\sigma, \lambda, J) = \sum_i (C_i - C_i'(\sigma, \lambda, J))^2$$

- The optimization problem is minimize the error subject to the constraints $\sigma > 0$, $\lambda > 0$, $J > 0$.
- By writing $\sigma = e^{x_1}$, $\lambda = e^{x_2}$, $J = e^{x_3}$ we can convert this to an unconstrained optimization problem of choosing (x_1, x_2, x_3) to minimize the error.

Remarks

- The quadratic program we solved was *convex*. That is we wished to minimize a convex function defined over a convex domain.
- Convexity ensures that the optimization problem is "nice". For example a local minimum of the function is an absolute minimum.
- The calibration problem is smooth, but non-convex. The solver we will use will only find a local minimum of the problem. We'll just have to hope that the error achieve for our fit is small, we can't guarantee that it is the best possible fit.

Using fminunc

- MATLAB provides a function fminunc for unconstrained optimization.
- The name is short for "function minimization unconstrainted".
- To use fminunc you must provide an *objective function* — i.e. the function to minimize.
- The objective function should take a vector of parameters and returns a single real number.

```
[x,fVal,exitFlag]=fminunc( @f, x0, options );
```

- x0 is your initial guess at the solution.
- x, fVal and exitFlag and options have the same meaning as for quadprog.

Data used for fit

- The file `goog-options.xslx` contains option data for Google taken from Bloomberg on 20 March 2014.
- It contains strikes and mid prices for numerous options all with the same maturity date of 19 April.
- The stock price was 1205.415
- We will assume a risk free rate of 0.16%
- The time to maturity is 31/365.

A function to load the Bloomberg data

```
function [strike,T,S0,r,mid] = googOptionData()

% Read the raw data from the excel file
T = 31/365;
S0 = 1205.4;
r = 0.16/100;

bloombergData = xlsread( 'goog-options.xlsx', 'C5:D54' );
mid = bloombergData(:,1);
strike = bloombergData(:,2);

end
```

A jump diffusion pricer

```
function total = jumpDiffusionPrice( ...
    K, T, S0, r, sigma, lambda, J )
total = 0.0;
coefficient = exp( - lambda*T );
j = 0;
mu = -(J-1)*lambda + r;
while true
    term = coefficient * blackScholesCallPrice(...
        K, T, S0*exp((mu-r)*T)*J^j, r, sigma );
    total = total + term;
    if (abs(term)/abs(total)<1e-6)
        return;
    end
    j = j+1;
    coefficient = coefficient * lambda*T / j;
end
end
```

Calibration function page 1

```
function [ sigma, lambda, J ] = calibrateJumpDiffusion()
[strike,T,S0,r,mid] = googOptionData();

function e = errorFunction( x )
    sigma = exp(x(1));
    lambda = exp(x(2));
    J = exp(x(3));
    fprintf( 'Sigma=%d, lambda=%d, J=%d\n',sigma,lambda,J );
    e = 0;
    for i=1:length(strike)
        K = strike(i);
        p1 = jumpDiffusionPrice(K,T,S0,r,sigma,lambda,J);
        p2 = mid(i);
        e = e + (p1-p2)^2;
    end
end
```

Calibration function page 2

```
% Note that we may only find a local minimum, so
% a good choice of x0 is important
x0 = [ log(0.25), 0.3,0.9 ];

[xOpt,~,exitFlag] = fminunc( @errorFunction, x0 );
assert(exitFlag>0);

sigma = exp( xOpt(1) );
lambda = exp( xOpt(2) );
J = exp( xOpt(3) );

end
```

Care with optimization

- `fminunc` only finds local minima. Only if your problem is convex will it find global minima.
- The numerical algorithms assume that the problem is centered and scaled.
  - Centered: the "interesting" part of the problem is reasonably near the origin. We should expect to find the local minimum somewhere near the origin.
  - Scaled: the natural scale of the problem is roughly 1. The vector space to search for the optimal solution should be reasonably similar to the unit cube.
- The reasons for these requirements are:
  - We are using numerical methods with nonlinear functions and with rounding errors. We don't want all the information in our problem to be hidden by rounding errors.
  - The numerical methods use discrete approximations to derivatives and differential equations. They choose step sizes δx on the assumption of good scaling.

How `fminunc` works

For moderate sized problems, the algorithm MATLAB uses is roughly this:

- MATLAB assumes your objective function is smooth
- Given a guess $\underline{x_i}$, MATLAB estimates the first and second derivatives of the objective function numerically
- It then approximates your function with a quadratic and finds an estimate for where the minimum will be using `quadprog`. This is $\underline{x_{i+1}}'$
- MATLAB searches along the line joining $\underline{x_i}$ and $\underline{x_{i+1}}'$ to find the minimum on this line using a "line search algorithm". This gives the point $\underline{x_{i+1}}$.
- Repeat until the derivative of the objective function is satisfactorily close to zero.

This is the Newton method.

Optional: providing a gradient and Hessian

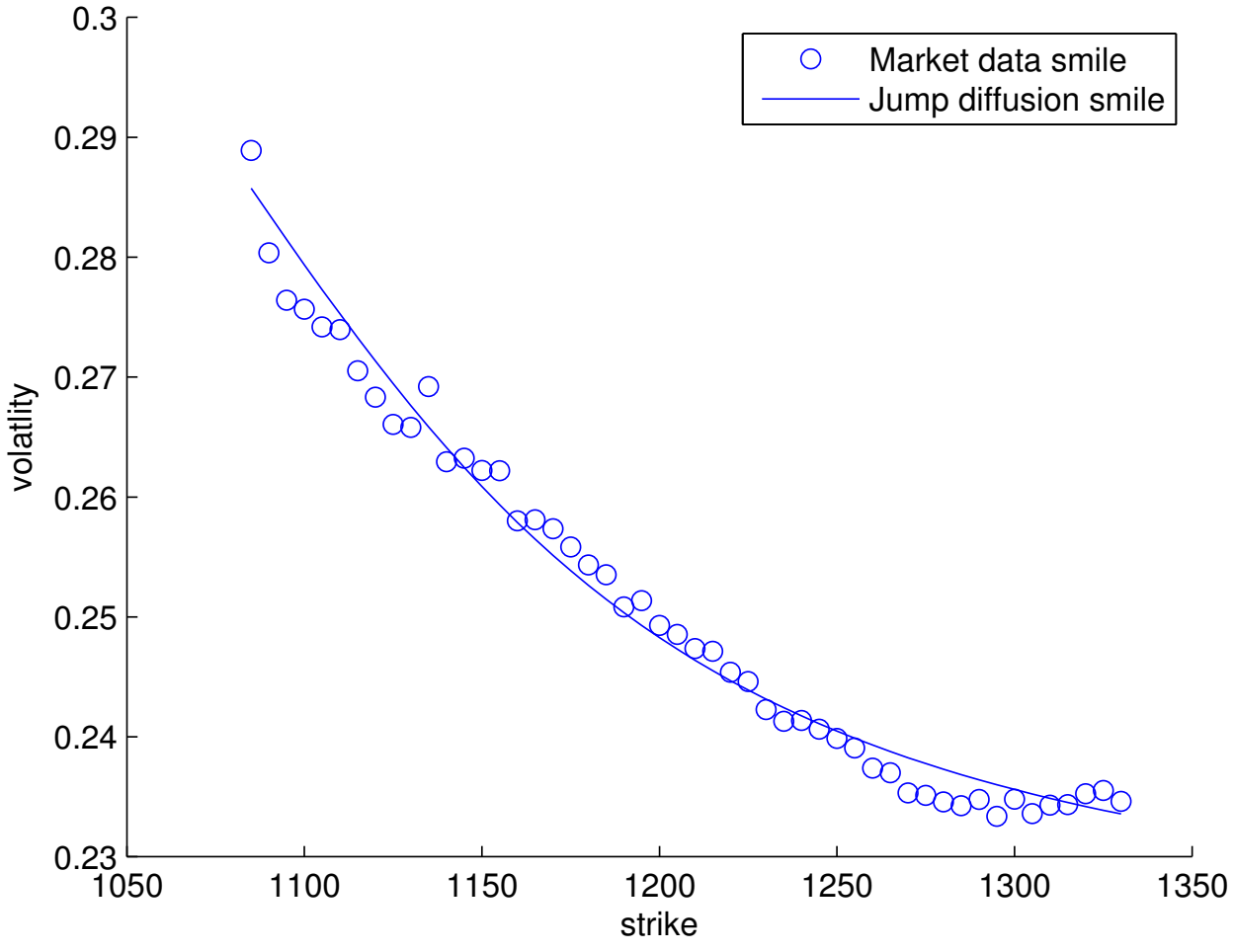
- In order to calculate the minimum, MATLAB will call your objective function repeatedly both to estimate the value and to estimate derivatives.
- If you wish you can write your function so that it returns:
  - The value of the objective function, f at $\underline{x}$
  - A vector representing the gradient of f at $\underline{x}$. In other words the vector of values $\frac{\partial f}{\partial x_i}$.
  - A matrix containing the second derivatives of f at $\underline{x}$, $\frac{\partial^2 f}{\partial x_i \partial x_j}$. This matrix is called the *Hessian* matrix.

Choosing options

`fminunc` gives us lots of choices.

- When estimating derivatives what value should we use for δx?
- When do we consider the gradient to be sufficiently close to zero?
- How many steps of the algorithm should we perform before giving in?
- Should we use a different algorithm entirely (e.g. for large scale problems where x is high dimensional)?
- Typically the best approach is to ensure that your problem is centered and scaled and then use the defaults.
- Always check the `exitFlag` however!

Fit of jump diffusion model to data



Remarks on jump diffusion model

- Our choice of jump diffusion model is very crude. It wouldn't fit the full pricing surface at different maturities very well.
- Notice that the parameters found by fitting are for a risk neutral model and not the real world model.
- This means that the model tells us as about market beliefs and risk preferences rather than about actual probabilities.

Calibration of other models

- We chose a jump diffusion model because it is easy to implement.
- You can repeat a similar process for other incomplete models.
- Note that `fminunc` makes many calls to our pricing function. So to calibrate a model, we need to be able to price vanilla options fast.
- This is why traders like models that allow us to quickly price vanilla options.
- The Heston model can be priced rapidly using fourier transform methods.
- If you write down a random stochastic model for the stock it can probably only be priced by Monte Carlo, so it will be hard to calibrate.

Optimizing utility

General setup

- Suppose that we have n assets X_1, X_2, $\ldots X_n$ and we have a stochastic model for the price of these assets.
- There are various constraints on our trading — for example we might only be allowed to buy or sell a certain quantity of each asset.
- Suppose that we have a utility function $u : \mathbb{R} \longrightarrow \mathbb{R}$
- We wish to find the trading strategy that maximizes our expected utility.

Utility functions

A utility function is a concave function. Popular examples are:

- *power utility* with risk aversion parameter η

$$u(x) = \begin{cases} \frac{x^{1-\eta}-1}{1-\eta} & \eta \neq 1, \quad x > 0 \\ \ln(x) & \eta = 1, \quad x > 0 \\ -\infty & x \leq 0 \end{cases}$$

- *exponential utility* with risk aversion parameter λ

$$u(x) = 1 - e^{-\lambda x}$$

Note that power utility assigns infinite negative utility to losing money. This means that only trading strategies that have 0 chance of bankruptcy will ever be considered. For example, using power utility in a Markowitz type problem would effectively mean prohibiting short selling.

Computing expected utility

Computing expected utility is just a matter of computing an expectation so we can use all the techniques we have developed for computing expectations

- Rectangle rule, converges with rate ...
- Trapezium rule, converges with rate ...
- Simpson's rule, converges with rate ...
- Monte Carlo, converges with rate ...
- Gaussian quadrature, converges with rate ...
- Compute the confidence interval for Monte Carlo using ...
- Try to reduce the variance of Monte Carlo using ...

One period problem

- A trader wishes to invest in a number of stocks that all follow multivariate geometric Brownian motion model.

$$\mathrm{d}S_t^i = S_t^i(\mu^i \mathrm{d}t + \sigma^i \mathrm{d}W_t^i)$$

  where each S_t^i is a vector of stock prices, μ^i a drift vector, σ^i is a volatility. The W^i are Brownian motions with given correlation matrix.
- Equivalently

$$\mathrm{d}(\log S^i)_t = \left(\mu^i - \frac{1}{2}(\sigma^i)^2\right)\mathrm{d}t + \sigma^i \mathrm{d}W_t^i$$

- We wish to choose a static portfolio of given quantities in each stock to maximize the expected utility at time T. Trading at intermediate times is not allowed.

One period solution, sketch

- Use . . . to generate normally distributed random variables with the desired correlation matrix.
- Use the . . . method with one step to simulate log stock prices at time T. We simulate log stock prices so that . . .
- Hence compute the expected utility at time T as a function of the proportions invested in each stock using the Monte Carlo method. This will converge at a rate . . .
- Using `fmincon` or `fminunc` find the optimal proportions that minimize the expected disutility (=minus the utility).

Dynamic Programming Example — hedging

- X_1 is the stock and it follows the Black Scholes model.
- X_2 is a call option on the stock. We know the payoff at maturity and we know that we have a customer who is willing to buy one call option today at the Black Scholes price plus a small commission.
- We have the following constraints on our trading:
  - We can sell up to one unit of the call option at time 0. We can't buy the call option. We can't trade in the call option at other times.
  - We can only trade in the stock at 20 evenly spaced time points. i.e. continuous time trading is not possible.
- How should we trade?

Numerical methods for dynamic programming

- Dynamic programming problems are very hard
- General purpose algorithms exist, but they are not very efficient.
- The basic problem is that the space of possible strategies is infinite dimensional and it is hard to find good low dimensional approximations.
- To resolve this problem we use the *Galerkin* approach.

Galerkin method - idea

- Choose a finite set of candidate strategies based on intuition and experience:
  - Strategy S_1: delta hedging. Sell the customer the option, delta hedge at each time time point (and then pay out as required option).
  - Strategy S_2: no hedging. Sell the customer the option and don't bother hedging.
  - Strategy S_3: stop-loss hedging. Sell the customer the option, perform stop-loss at each time time point.
  - Strategy S_4: trade in the stock alone. Borrow money to buy the stock, wait till maturity and then sell the stock.
- We can form linear combinations of strategies. Suppose that we're a bank, we can instruct trader i to follow strategy i (scaled up or down by a factor of α_i) and then see what the net effect is. This is strategy $\alpha_i S_i$.

Galerkin method

- We have n strategies S_n.
- We can calculate by Monte Carlo N possible scenarios for asset prices and compute the profit and loss of each strategy.
- This gives a vector $x^{(i)}$ of profit and loss for each strategy with rows corresponding to each scenarios. It is crucial to use the same scenarios for each strategy.
- Linear combinations $\sum_i \alpha_i S_i$ are also valid strategies (possibly subject to some constraints on the α_i). The profit and loss of the combined strategy in each scenario is $\sum_i \alpha_i x^{(i)}$.
- Find the values of α_i, subject to constraints, that minimize

$$-\frac{1}{N}\sum_j u(\sum_i \alpha_i x_j^{(i)}).$$

- The index j is running over scenarios. i is running over stratgies.

Output of the Galerkin method

- The Galerkin method will give us a portfolio of strategies.
- This combined strategy is unlikely to be a perfectly optimal solution to the problem, but it is guaranteed to be at least as good as any individual strategy.
- Thus the Galerkin method allows you to improve performance by diversifying over strategies.
- This generalizes the notion of diversification for static trading strategies to dynamic trading strategies.
- Note that the method can be applied equally well to static trading strategies. This allows you to optimize static trading strategies even when Markowitz assumptions do not hold.
- Note: so long as the utility function is smooth and concave, this will be a smooth convex optimization problem.

Our example

- We have already (in previous weeks) shown how to compute the profit and loss of strategies S_1, S_2 and S_3 given scenarios for the stock price. S_4, buying and holding stock, is trivial to evaluate.
- Thus we can compute the vectors $x^{(i)}$.
- The customer is only willing to buy up to one call option. We cannot sell call options. This gives constraints:
  - $\alpha_1 \geq 0$
  - $\alpha_2 \geq 0$
  - $\alpha_3 \geq 0$
  - $\alpha_1 + \alpha_2 + \alpha_3 \leq 1$
- We can write the last equation in matrix form as

$$\begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \alpha \leq 1$$

```
function [ alpha ] = optimizeUtility( ...
    lambda, ...
    pnlArray, ...
    A, ...
    b, ...
    lb, ...
    ub)

function d = expectedDisutility( alpha )
    pnl = pnlArray*alpha;
    u = mean( 1 - exp(-lambda*pnl));
    d = -u;
end

nStrategies = size(pnlArray,2);
alpha0 = zeros(nStrategies,1);
alpha0(1)=1;
options = optimset('fmincon');
options = optimset(options,...
    'Display','off', 'Algorithm', 'active-set');
[alpha,~,exitFlag] = fmincon( ...
    @expectedDisutility, alpha0, A,b,[],[],lb,ub,[],options );
assert( exitFlag>0 );

end
```

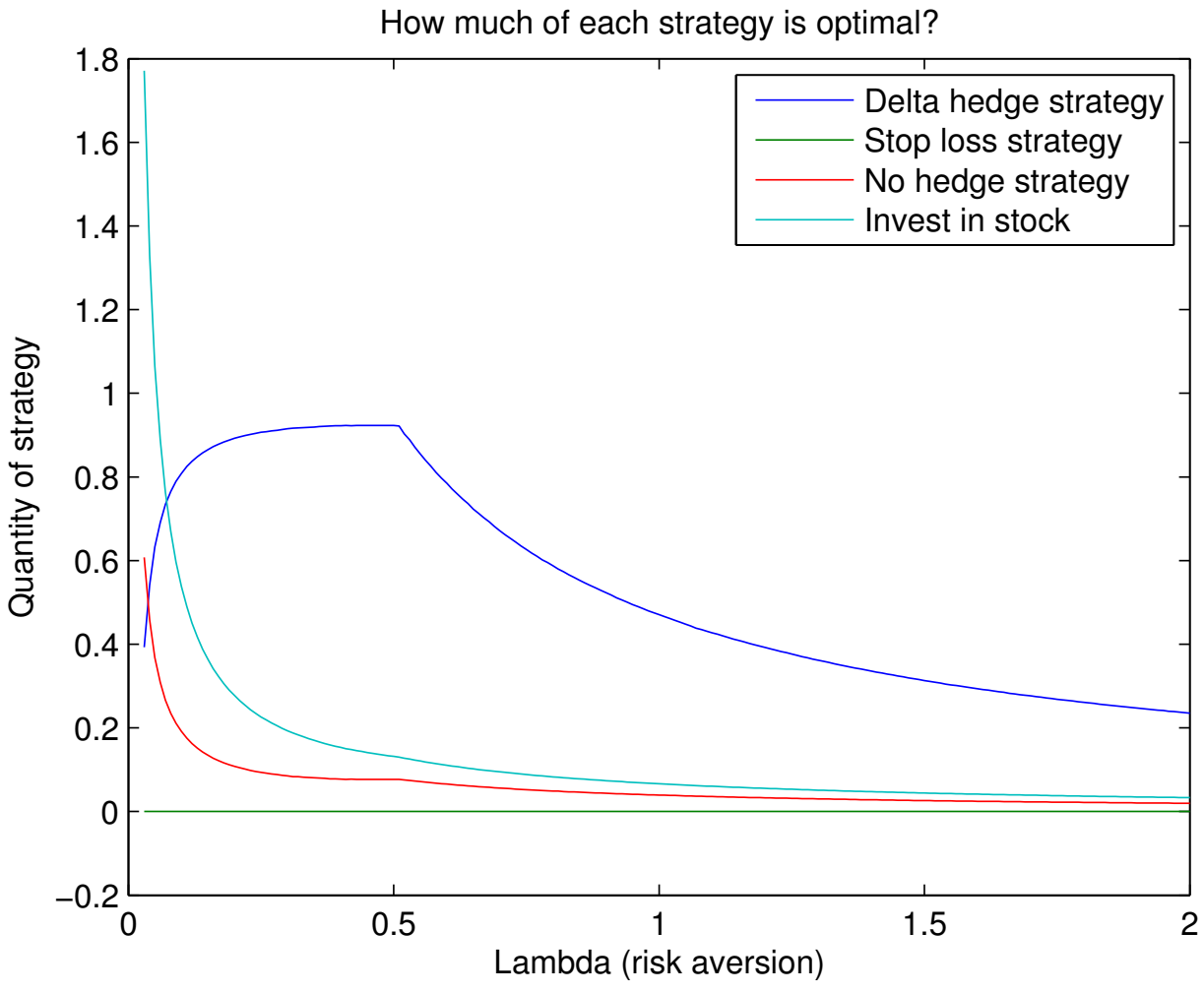
fmincon

- `fmincon` is MATLAB's function for constrained nonlinear optimization
- It takes similar parameters to both `quadprog` and `fminunc`.
- Just like `fminunc` you specify the objective function by creating an appropriate MATLAB function. In this case we use the exponential utility function to compute the objective.
- Just like `quadprog` you specify constraints using various matrices and vectors such as `A` and `b`.
- `fminunc` also allows you to specify a general non linear constraint function if necessary.
- The same considerations of centering, scaling and non-uniqueness apply to `fmincon` as apply to `fminunc`.

Applying this to the problem

- We can now find the optimal combination of strategies to use for our problem.
- We choose a concrete process for the price. S follows the Black Scholes model with $K = 100$, $S = 100$, $T = 0.5$, $r = 0.03$, $\mu = 0.2$, $\sigma = 0.2$
- The customer is willing to pay 1.1 times the Black–Scholes predicted price for the call option.
- We can now generate 10000 stock price scenarios and compute the profit and loss vector $x^{(i)}$ for each strategy.
- We then call `optimizeUtility` to find the optimal combination of strategies.

The optimal portfolo of strategies



Interpretation of results

- The optimal strategy depends upon risk preferences λ.
- Note that our model has a very high drift $\mu = 0.2$. This explains why investing in the stock seems like a good idea.
- Our model is a discrete time model. So perfect delta hedging is impossible. This is reflected in the charge the customer pays on top of the Black–Scholes predicted price.
- Delta hedging is not the optimal strategy. This is because it is risky in discrete time.
- Depending upon our risk aversion we may prefer no hedging to delta hedging
- If our risk aversion is high, we might prefer not to trade since no risk free strategy is available.

Remarks on using Galerkin method

- It is likely that you would want to report the expected utility of the strategy. If you want to do this, note that the returned `fval` will *not* be an unbiased estimate of the disutility of performing our strategy. To estimate the utility correctly, you must try the portfolio on a new random sample of scenarios.
- In general you should always test a strategy on out-of-sample data
- At every time step you can re-run the Galerkin method to find a new strategy which incorporates the information received since the previous time step.

Generalization

This idea can be generalized easily.

- Include transaction costs in the model. We simply need to change the computations of the profit and loss vectors to take this into account.
- Use a different model to generate stock paths. You can use any model at all, or historic data, to simulate price paths and hence compute profit and loss vectors.
- Add in other strategies, assets etc.

Although the Galerkin method does not find the true optimal solution to the problem, it does allow us to find improved solutions easily. Moreover it can be applied very generally.

Summary

We have seen how `quadprog`, `fmincon` and `fminunc` can be used to perform:

- Static portfolio optimization
- Model calibration
- Dynamic portfolio optimization

References:

- Markowitz : "Portfolio Selection". The Journal of Finance (1952)
- Koivu & Pennanen: "Galerkin methods in dynamic stochastic programming". Optimization 59 (2010)