An Introduction to Software Testing and Test Strategies, in context with HO and with an example with TA and CAPM.

Chris Sharpe
Department of Economics
Justus-Liebig University, Giessen

Fellows' Workshop on 'Numerical Methods and Optimization in Finance'
, London

June 19, 2009

- Introduction to Software Testing

- Introduction to Software Testing
- A Few Anecdotes

- Introduction to Software Testing
- A Few Anecdotes
- Test Methodology
  - Test Standards and Test Tools; Software Specification; Test Definitions

- Introduction to Software Testing
- A Few Anecdotes
- Test Methodology
  - Test Standards and Test Tools; Software Specification; Test Definitions
- Test Techniques
  - Test Types - White Box Testing, Black Box Testing, Performance Testing; Test Process

- Introduction to Software Testing
- A Few Anecdotes
- Test Methodology
  - Test Standards and Test Tools; Software Specification; Test Definitions
- Test Techniques
  - Test Types - White Box Testing, Black Box Testing, Performance Testing; Test Process
- Testing in Context with Heuristic Optimisation Techniques

- Introduction to Software Testing
- A Few Anecdotes
- Test Methodology
  - Test Standards and Test Tools; Software Specification; Test Definitions
- Test Techniques
  - Test Types - White Box Testing, Black Box Testing, Performance Testing; Test Process
- Testing in Context with Heuristic Optimisation Techniques
- Test Specification Example: CAPM with Threshold Accepting
  - CAPM model and problem specification; Coding Blocks; Data; Alpha and Beta; Objective Function; Threshold Accepting Algorithm

Introduction to Software Testing

- What is Software Testing?
- Process of validating and verifying that a program does what it is expected to do.
- From Thought to Expression - Where the Bugs come in.
- "Beware of bugs in the above code; I have only proved it correct, not tried it.", Donald Knuth
- How do you test Software?
- By selecting the right techniques and applying them systematically.

A Few Anecdotes

- The Classic tale - NASA, 10 metres for 10 feet.
- The Hedge fund's tale - plug it in and it'll make money!
- The Robot's tale - but it works!
- The Applications's tale - Matlab, are we building on sand?

- But what about me, a student/researcher?
- Proper Testing saves time and stress in the long term.
- It installs confidence to know that something works.
- It makes it easier to extend a program later on.
- Experiment results are valid and credible.

Test Standards and Test Tools

- Test techniques haven't developed much in 30 years!
- IEEE Software Standards - extensive, wide ranging set of standards.
- Many Testing standards working parties (i.e. BCS) - they cover general methodology.
- Test Tools (i.e. ClearCase) - for large projects, various code coverage tools, language specific, expensive.

Software Specification and Test Case Description

- Software Specification - first write down your program clearly before you touch the computer.
- Divide Problem into a set of Requirements.
- Reason why you are doing what you are doing.
- Devise an appropriate test case from this. It is a process of selecting the right methods and applying them properly.

Software Specification - Formal Structure

For each functionally distinct part of code...

- Functional Requirement: A clear statement of what something should do.
- Non Functional Requirement: The constraints under which your program shall operate.
- Rationale: explain why you are doing something.
- Exceptions: a descriptions of how you handle illegal' events.
- Test Case: select and list test cases for the requirement.

Test Definitions

- A Test Case - test technique used, description of how code should behave, the output expected for given input, and how it can (may) fail. The means to test these events. The test case implementation - code and data generator, to exercise other code in some way.
- Bug - wrong/unexpected results/behaviour produced for given input.

- Exception Handling - description of how failure is handled by the code.
- Debugging - the process of exposing and fixing bugs.
- Test Outcome - an enumeration of possible outcomes: Pass, Fail, Not Verified, Timeout, Memory Dump etc.

Test Techniques

- White Box Testing - Testing the logic and structure of code, several lines of code, individual statements, loops.
- Black Block Testing - Testing functional behaviour, mapping input to output.
- Performance Testing - Metrics: how fast, how much memory, how much processor percentage used.
- Other Test Types - Regression testing, stress testing, robustness testing, specification test.

Statement Testing

- Definition: Identify if all paths in the code are executable.
- Meaning: There are no 'dead' code paths, all code is exercised at least once.
- Technique: Identify input sequences that should cause all parts of the code to be reached.

Branch Decision Testing

- Definition: Exercise executable decision points in code that may transfer control to other parts of code, based on its logic.
- Meaning: What should happen when your loop/case/simple if statement is executed, and is the processing activity done correctly.
- Technique: Execute the loop/case/simple if statement, and trace input processing - and termination conditions for a loop.

Branch Condition Testing

- Definition: Exercise all combinations of a decision point that use compound Boolean statements.
- Meaning: What should happen when you use if, else, else if, that contain more than one Boolean operator to determine code execution.
- Technique: Use Boolean truth tables to identity input combinations.

Code Inspection

- Definition: Read through the code or function specification to verify that it does what it should do against the specification.
- Meaning: It's the means to get an immediate overview of a code block or function and do a 'thought' experiment.
- Technique: Print out your code and read through it carefully, or read through an applications function specification (API).

Code Discussion

- Definition: Discuss with other people what you think your code is doing.
- Meaning: It is good to talk!
- Technique: Get somebody with programming knowledge and in your field and explain your what you think your code is doing.

Equivalence and Boundary Value Partitioning

- Definition: Partition into sets of representative of input, including boundaries and 'invalid' input, and map it to output.
- Meaning: This is examining the behavior of a function as defined by its specification.
- Technique: List set $I \mapsto \mathit{Valid}(O)$, and $I \mapsto \mathit{Invalid}(O)$.

State Transition

- Definition: Identify all possible states of a data item, the transitions between states, and the events that cause the transition.
- Meaning: Trace the changes of a data item or data structure before and after some actions are (deterministically or probabilistically) applied.
- Technique: List states, operators, and transition, depicted as a state diagram, or a state transition table.

Performance Testing

- Definition: Measure and quantify the computer resources that any part of a program uses during execution.
- Meaning: This records how much in real time code takes to complete a significant action, how much memory is used, and the processor percentage utilised.
- Technique: Record performance benchmarks on a given platform, track memory leaks, identify computational bottle necks.

Test Process and Strategies

- Code and Test Specification ⇒ Coding and White Box Testing ⇒ Black Box Testing ⇒ Performance Testing.
- Common Problems: poor specification, the deadly false positives, introduce new errors, random testing.
- Debugging Effort: will take around 50 % of your time - add this on to any promise you make for results.

- Good Strategies - incremental testing, repeat the set up that causes a failure and identify bug by exclusion.
- Validation: view code as one large conjunction of Boolean statements.
- What you need - resolve, patience, a clear idea of what you are doing, knowing when to stop!

Some Experiment Assumptions

- A formally specified model, a data set/data generator - desire is to find parameter combination that gives approximate 'best' solution.
- HO is the core part of the program - around 500 - 1500 lines of code, add another 1000+ if you write the objective function from scratch.
- HO implemented in a procedural language.

- The code will execute on a standard 32 or 64 bit machine with common OS, and is allocated one main processing thread for the task.
- The objective function calculation uses most resources.
- Running time for a full experiment can last for several hours to a few days.

Algorithm 1 General Heuristic Optimisation Algorithm.

1: Initialise starting state, $s^c \in S$,Evaluate quality of s^c, I
2: **for** $i = 1$ to I **do**
3: A finite set of operators Σ, Generate New state $T : s^c \times \Sigma \longrightarrow P(s^n)$
4: Evaluate quality of s^n
5: Under some criteria $s^n = s^c$
6: **end for**

Initialisation

- Initial state, $s^c \in S$.
- Set of parameters $x_1 ... x_n$, drawn at random from some bounded search space.
- Objective function $f(x_1 ... x_n) \mapsto \mathbb{R}$. item I is the number of iterations the optimistaion process executes.

Generating New States

- $T : s^c \times \Sigma \rightarrow P(s^n)$
- A move in a random direction within a given distance.
- New state is non-deterministic.

Objective Function and New Solution Acceptance Criteria

- $s^n \mapsto \mathbb{R}$
- $s^n = s^c$
- New state maybe deterministically or non-deterministically adopted.

Algorithm Termination

- $i = 1$ to I
- HO $\rightarrow D_I$
- Convergence theory, I increase, converges f_{min}.
- Mean and Variance $\mu \rightarrow f_{min}$, $\sigma \rightarrow 0$ within a finite search space.

Experiment Overview

- LMS Estimation by HO with an Application to the CAPM and a Multi Factor Model.
- TA and DE applied to LMS calculation, a local and population based techniques.
- Here we show basic CAPM model with TA taken as an example.
- Historic Dow Jones stock returns used as data.

Capital Asset Pricing Model (CAPM):

$$r_{i,t} - r_t^s = \alpha + \beta(r_{m,t} - r_t^s) + \varepsilon_{i,t} \quad (1)$$

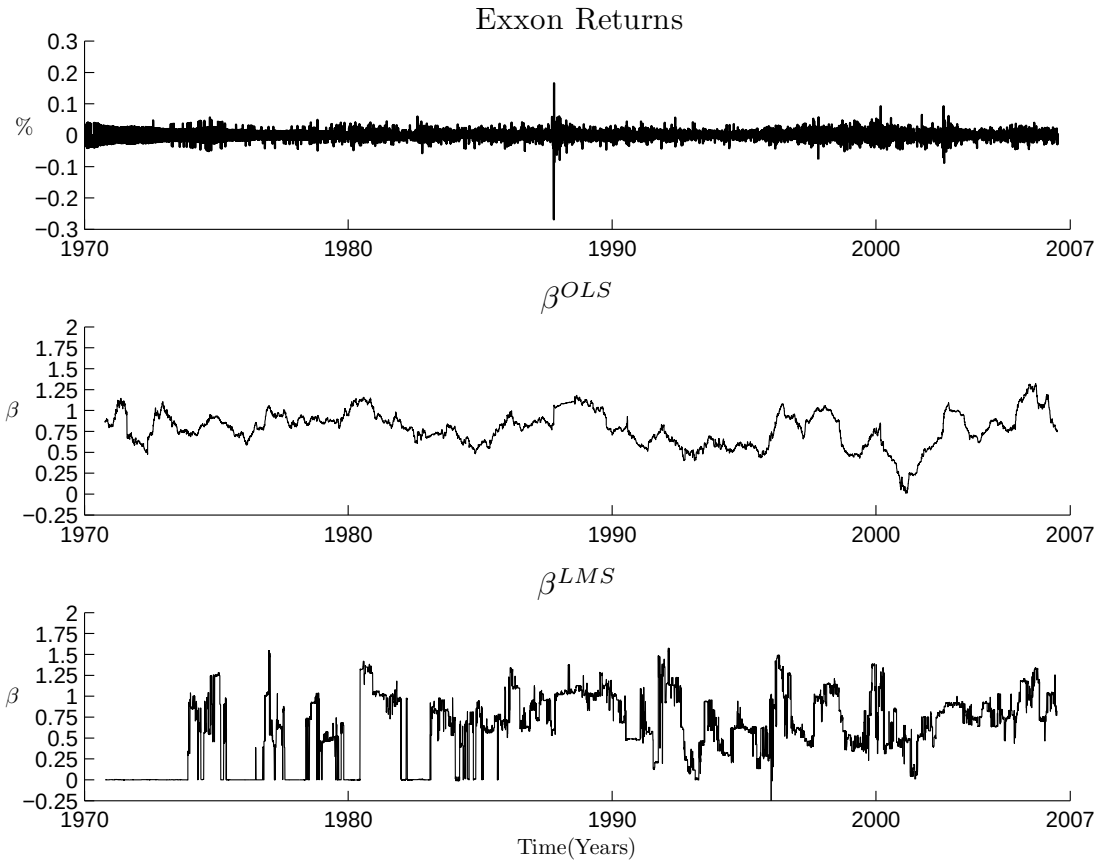
Coding Blocks

- Sequence of events, Data In $\Rightarrow$ Process $\Rightarrow$ Data Out
- Read In Dow Jones Data $\Rightarrow$ Init. Parameters $\Rightarrow$ Execute TA $\Rightarrow$ Store Results
- Valid(Experiment) if $\{f_1 \wedge f_2 \wedge \cdots \wedge fn\}$
- $f = S$, where S is $\{s_1, s_2 \ldots sn\}$
- Valid(f) if $\{s_1^* \wedge s_2^* \wedge \cdots \wedge s_n^*\}$, $S^* \subset S$

Data Specification

- Non Functional Requirement: The experiment uses Dow Jones stock returns from the period 1970 - 2006. We take the general market returns, and returns for a particular stock. For all returns we take the log differences.
- Rationale: The data required is as specified in the CAPM model. All serious empirical work is done in terms of log differences. It is more aligned to theory and produces better results.
- Test: Validate the provenance of the data. Check for anomalies in the particular stock selected.

Figure: Estimates of β for EXXON and the period between 1970 and 2006.



Alpha and Beta Initialisation

- Functional Requirement: The CAPM parameters α and β shall be drawn at random within given limits. Such that, $a1 \leq \alpha \leq a2 : a1 = -0.01, a2 = 0.01$, and $b1 \leq \beta \leq b2 : b1 = 1.3, a2 = 1.5$ The initial neighbourhood is the entire search space.
- Rationale: These are reasonable boundaries for the search space.
- Test: White Box - code inspection: check the *rand*() generator specification to ensure it produces non-repeating uniform random variables, and that we scale the output correctly for the boundary values.

Objective Function

- Functional Requirement: The objective function is the the function to be minimised for random values α and β, for a set of observation x_t and y_t.
  It is specified as $min_{\alpha,\beta,x_t,y_t}, (med(\varepsilon_t^2)) \mapsto \mathbb{R}$
  By rearragement of CAPM,
  $(med(\varepsilon_t^2)) = med((y_t - (alpha + beta * x_t)).^2)$
- Rationale: This is the objective function for the heuristic.
- Test: White Box - Code inspection: CAPM rearranged correctly, and check specification for native platform median function.

- Test: Black Box - Equivalence and Boundary Value Partitioning: understand statistical profile for *I* iterations.
- Test: Visualisation - 3D plot of objective function landscape,select α and β pairs to cover search space.
- Test: Performance Testing - average execution time for *I* iterations.

Figure: Objective Function Distribution - $I = 10^5$

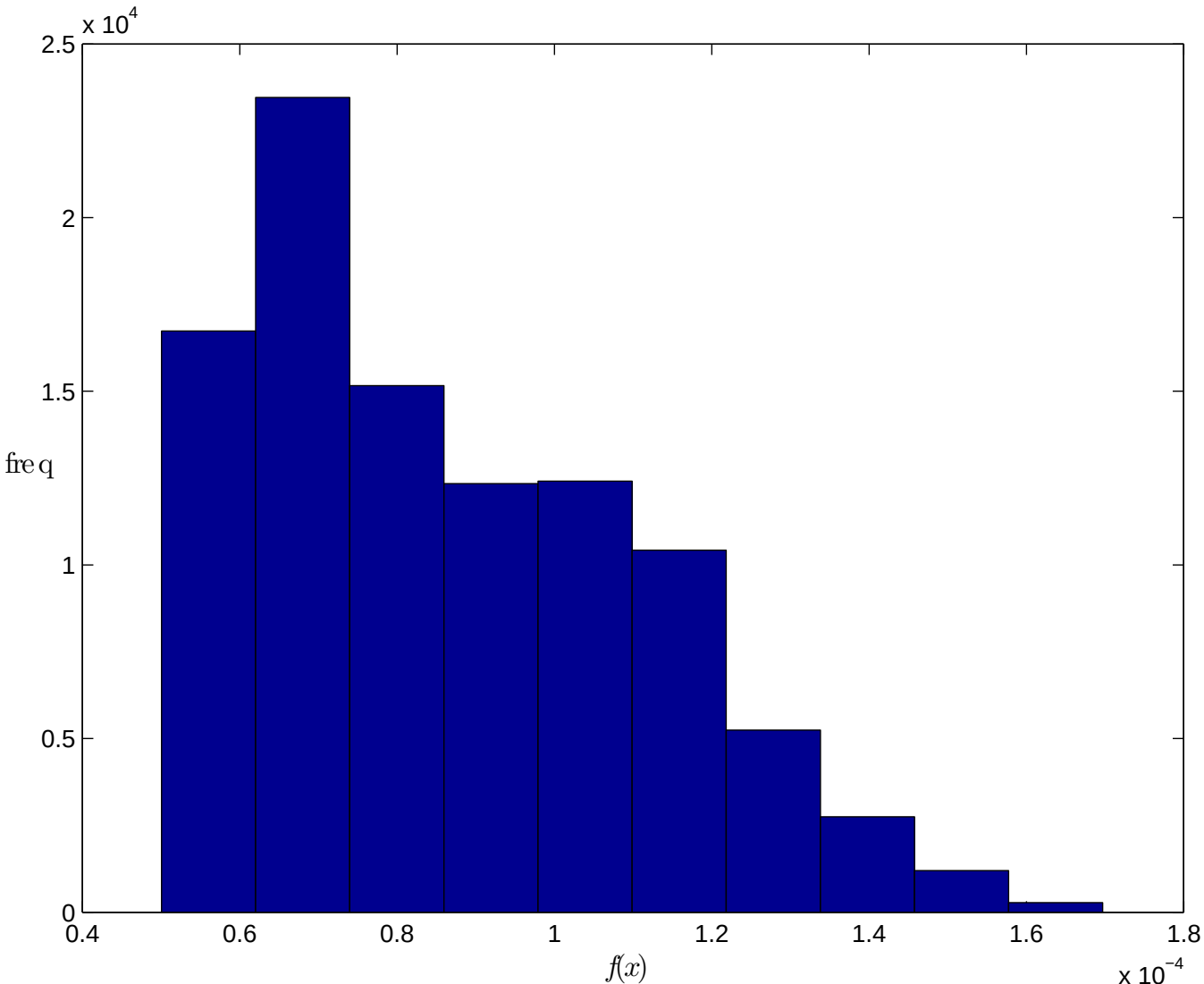
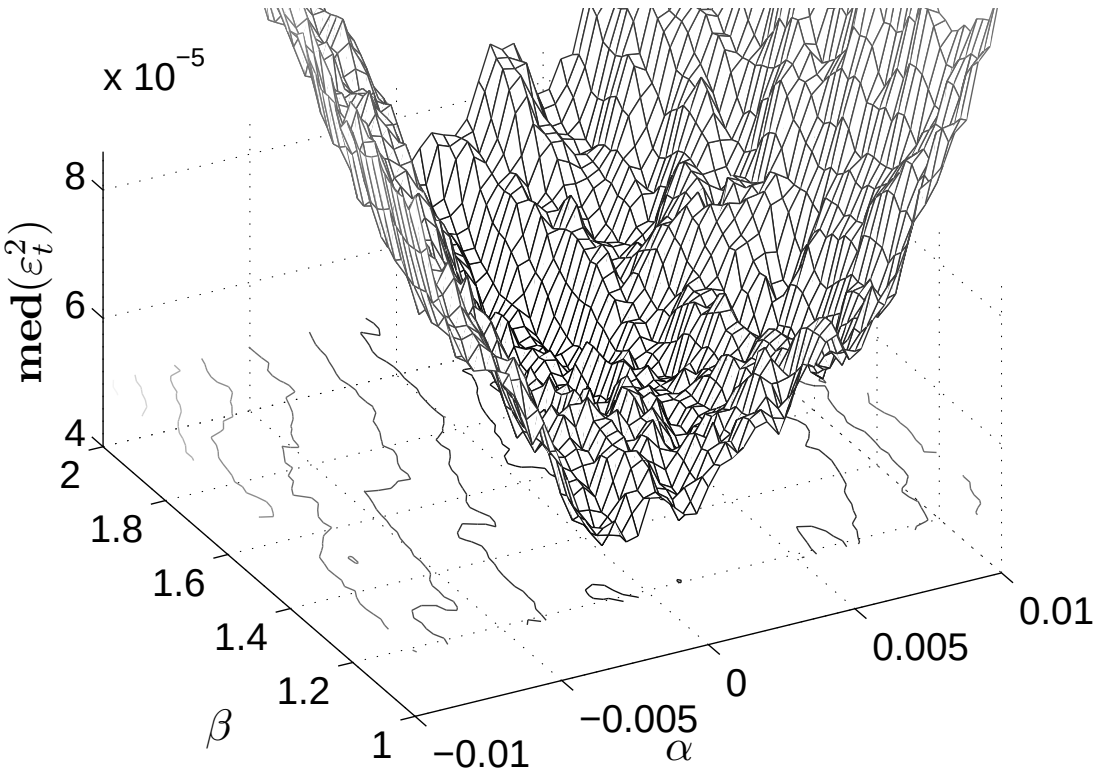


Figure: Least median of squares residuals as a function of α and β



Algorithm 2 Threshold Accepting Algorithm.

1: Initialise n_R, n_{S_τ} , and $\tau_r, r = 1, 2, \ldots, n_R$
2: Generate at random a solution $x^0 \in [\alpha_l \alpha_u] \times [\beta_l \beta_u]$
3: **for** $r = 1$ to n_R **do**
4: **for** $i = 1$ to n_{S_τ} **do**
5: Generate solution at random, $x^1 \in \mathcal{N}(x^0)$
6: **if** $f(x^0) - f(x^1) < \tau_r$ **then**
7: $x^0 = x^1$
8: **end if**
9: **end for**
10: **end for**

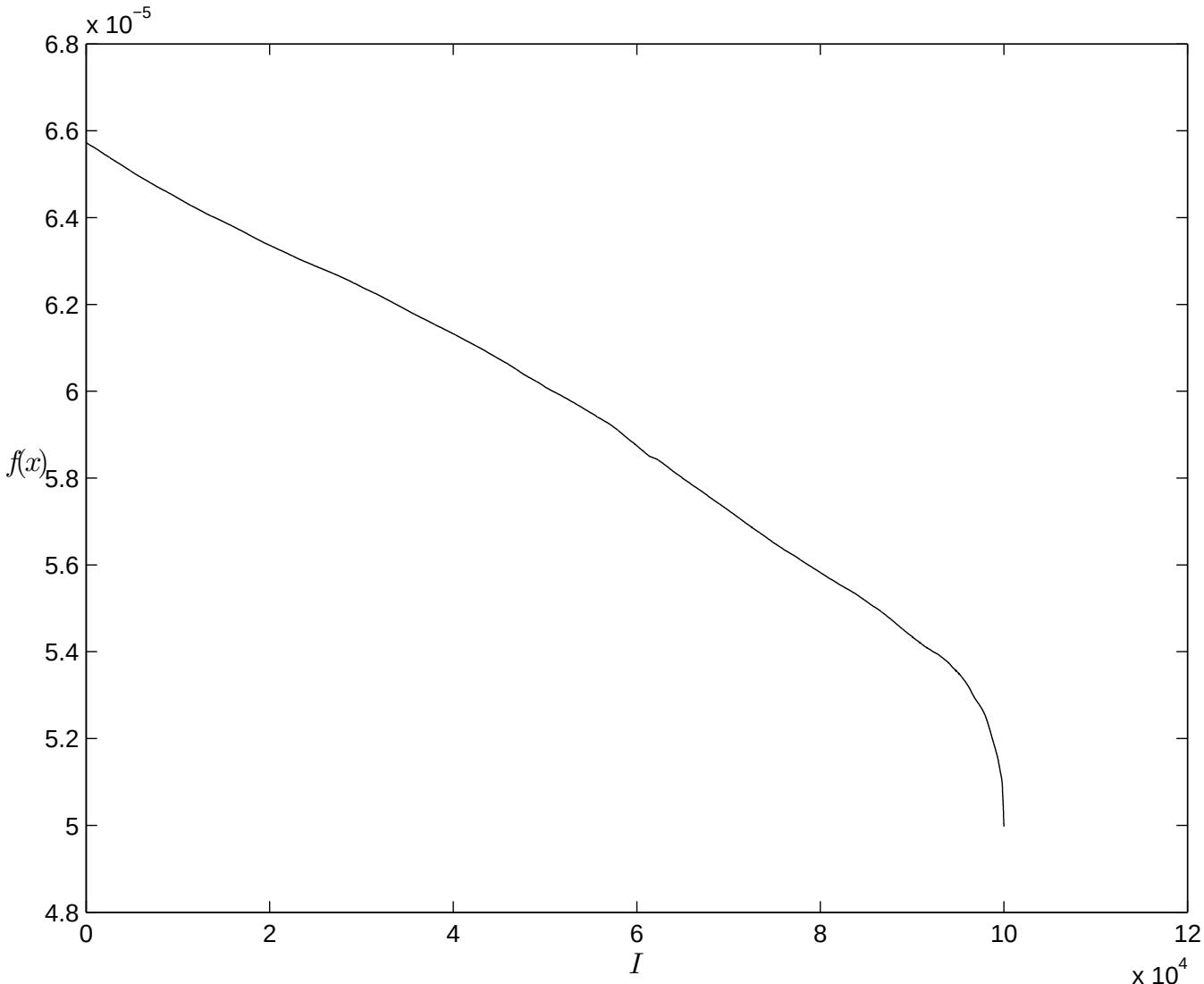
Initial State

- Functional Requirement: Generate a solution at random $x^0 \in [\alpha_l \alpha_u] \times [\beta_l \beta_u]$
- Rationale: as given in the algorithm
- Test: Black Box Testing - State Transition Diagram - State represented as $S1 = \{\alpha, \beta, x^0\}$

Threshold Sequence

- Functional Requirement: Generate Threshold Sequence for τ of length r, the number of threshold sequence reductions. First generate a sequence r * 4 times objective function for t stock return observations. Sort into ascending order as the objective function is to be minimised. Take the last r values.
- Rationale: This is the data driven algorithm.
- Black Box - Equivalence and Boundary Value Partitioning: Take that first and last values and check from objective function distribution.
- Test: Visualisation - Check that Threshold descent behaviour is as expected.

Figure: Data Driven Threshold Sequence



New Solution Generation

- Functional Requirement: Generate solution at random, $x^1 \in \mathcal{N}(x^0)$.
  The new state is in any random direction from the current state but must within the initial neighbourhood boundaries.
- Exceptions: If either α or β exceeds the boundaries, reflect the them back into the search space.
- Rationale: From the algorithm.
- Test: Black Box Testing - State Transition Diagram - $\sum = \{rand()\}$ new state is $S1 \times \sum \rightarrow P(S2)$ state represented as $S2 = \{\alpha, \beta, x^1\}$

Acceptance Criteria

- Functional Requirement: We accept the new state deterministically if $f(x^0) - f(x^1) < \tau_r$, otherwise we remain in the current state.
- Rationale: From the algorithm.
- White Box Testing: Branch Decision Testing - check that the new state is adopted if strictly lower than the Threshold value.

- Test: Black Box Testing - State Transition Diagram -
  $T : S1 \times \sum \rightarrow P(S2)$,
  T = {1,0} (we have a binary state for transition to the next state).
  In other words S1 = S2, otherwise remain at S1.

Neigbourhood Reduction

- The neighbourhood boundaries are defined as $x^0 \in [\alpha_l \alpha_u] \times [\beta_l \beta_u]$. The solution is equidistant from the neighbourhood boundaries. At each iteration n_R we reduce the neighbourhood. There shall be 10% of the original search space available at iteration n_R. The neigbourhood always maintains the same proportions.
- Exceptions: The neighbourhood boundaries are readjusted by the amount they exceed the search space boundary in the event of an upper or lower boundary exceeding the search space.
- Rationale: Neighbourhood space must to focus on a more concentrated (local) space where we believe good solutions lie.

Neigbourhood Reduction

- White Box Testing: Code Inspection - α span $= 0.02$ and β span $= 0.2$, so at 10% α span $= 0.002$ and β span $= 0.02$. The span should be decreased at each iteration by (first span - last span) / n_R. The boundaries are: lower boundary $=$ centre point - span$/2$, upper boundary $=$ centre point $+$ span$/2$.
- White Box Testing: Branch Decision Testing - If any boundary exceeds the search space, add or subtract the amount to the upper or lower boundary value, as required. We have eight tested cases. $\alpha_h >$ search space, $\alpha_l <$ search space etc.

- Black Box Testing: Equivalence and Boundary Value Partitioning - Ensure α and β are 10 of their original values.
- Visualisation: Investigate how the neighbourhood is reduced.

Figure: Neighbourhood Study - Snip with Hyper Rectangle

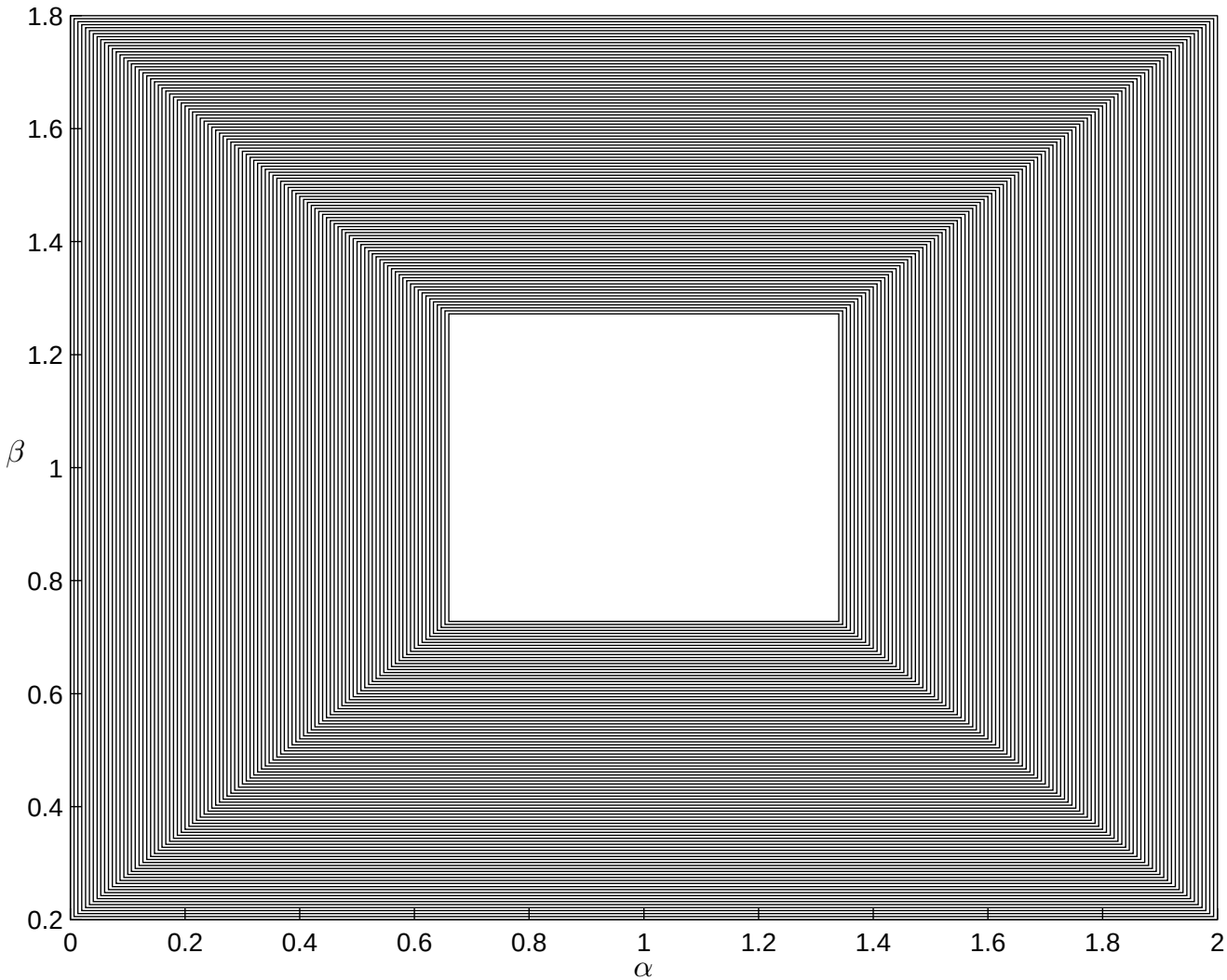
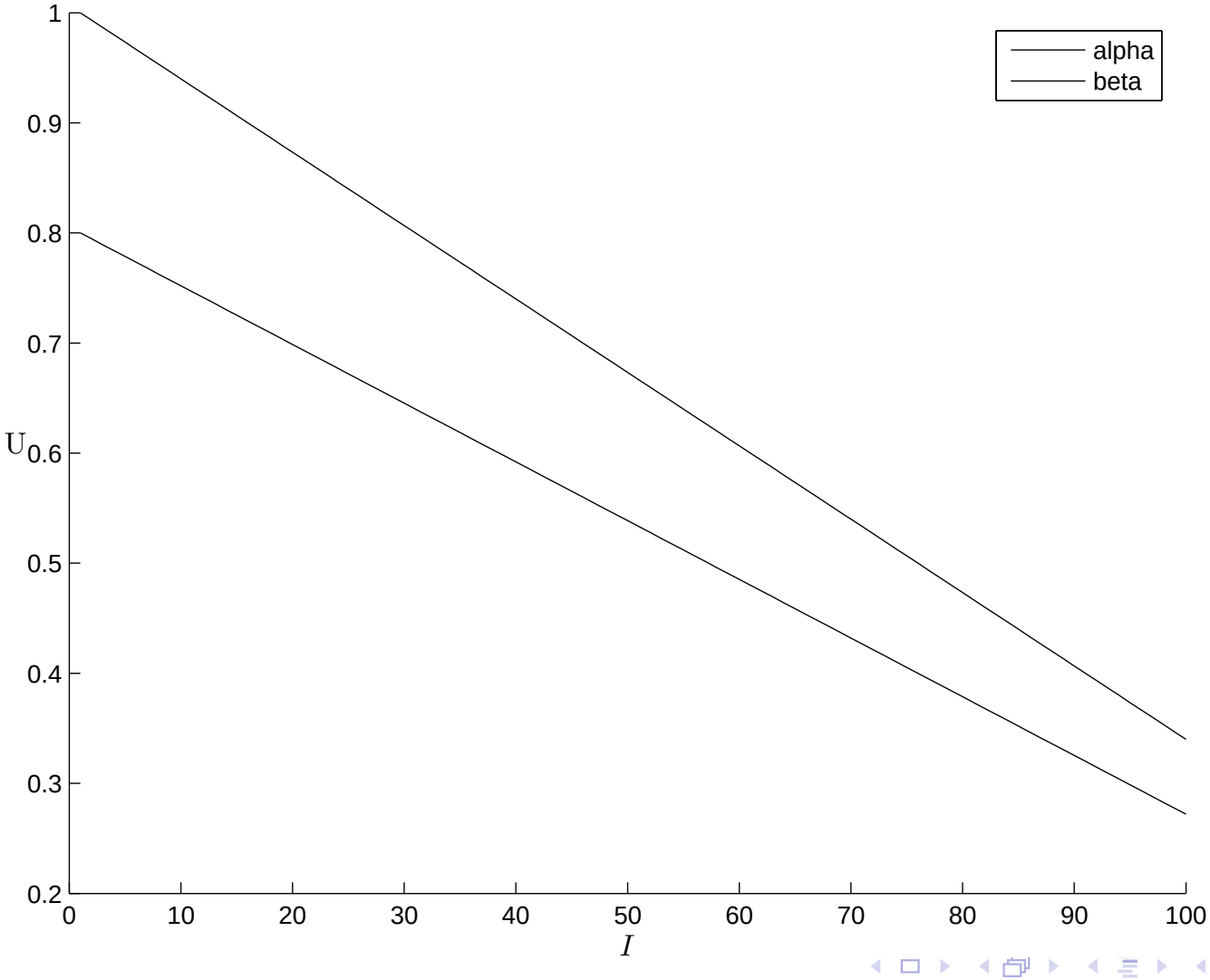


Figure: SNIP reduction rate



Algorithm Termination Pt 1

- Functional Requirement: The Algorithm should terminate after I Iterations. This should be divided up between n_R and n_{S_τ}. The algorithm should produce an optimised value, such that $min_{\alpha,\beta,x_t,y_t}, (med(\varepsilon_t^2))$.
- Rationale: From Convergence Theory.
- Black Box Testing: Equivalence and Boundary Value Partitioning - $min_{\alpha,\beta,x_t,y_t}, (med(\varepsilon_t^2)) \ll$ or $<$ initial solution.

Algorithm Termination Pt 2

- Black Box Testing: Equivalence and Boundary Value Partitioning - The stochastic distribution should see $min_{\alpha,\beta,x_t,y_t}$ and $\hat{\mu}$ lower bound, and $\hat{\sigma} \mapsto 0$, as I increases.
- Black Box Testing: State Transition Diagram - check the proportion of state transitions - transitions$/n_{S_\tau} = I$ - for first and last Threshold. It should be significantly higher at the start than the finish.
- Performance Testing: Measure how long the algorithm takes to terminate for an increase in I.
- Visualisation: Graph the behaviour of the algorithm through rounds in the Algorithm.

Figure: TA Search Behaviour - First Round.

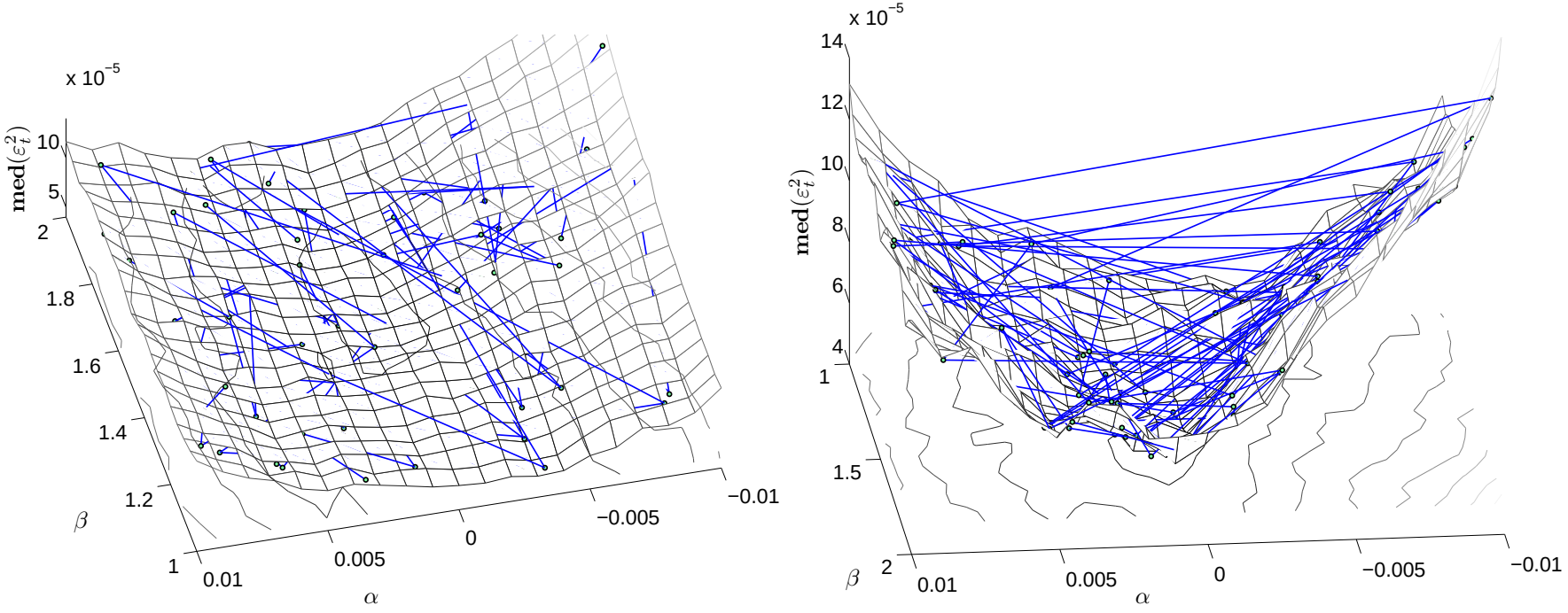
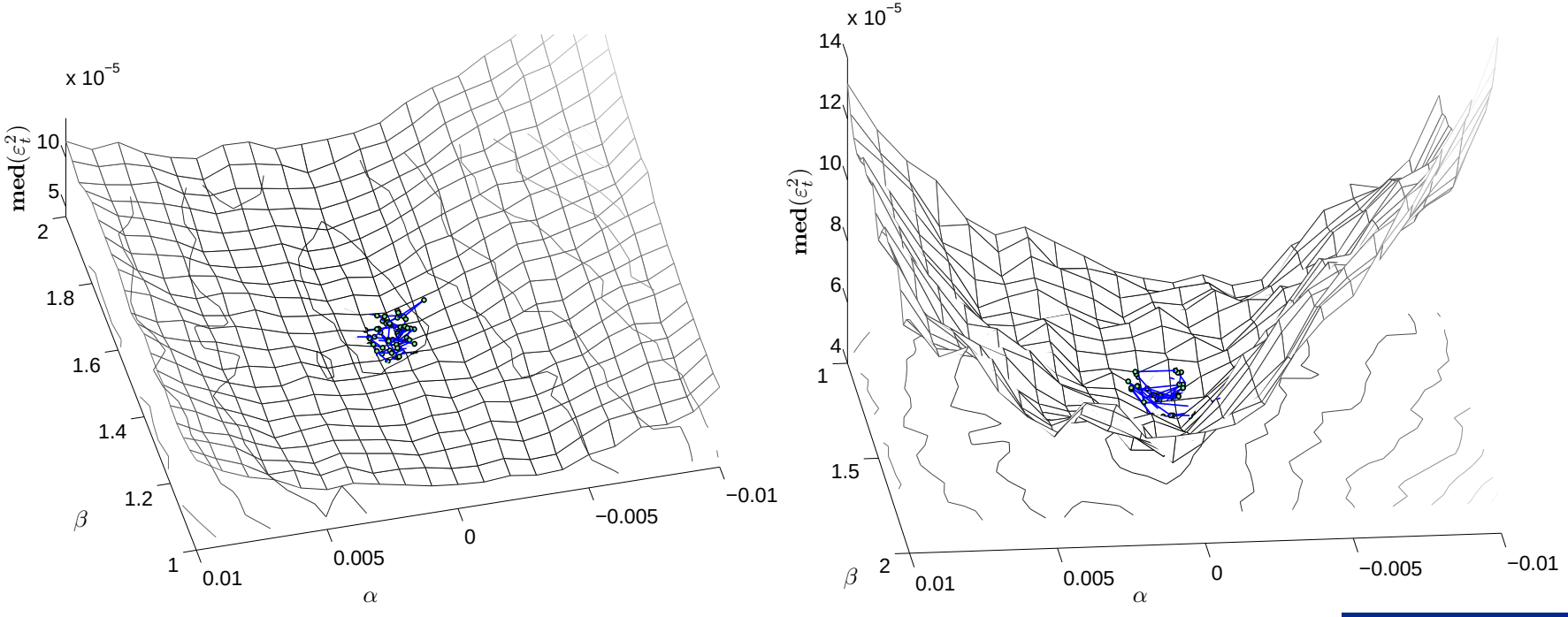


Figure: TA Search Behaviour - Tenth Round.



And Finally.....but see the paper

- Estimate CAPM for a rolling window
- Forecast CAPM based on actual market returns
- Compare results

Testing - General Advice

- Do plan what you are going to do in detail before coding.
- Talk through your code with someone - communicate your ideas.
- Where need be, read in detail the API for application software.
- Ensure each code block is working correctly before moving on.
- Back up code regularly - and create versions.
- Add 50% on to your coding effort for bugs.
- Be very patient and get plenty of coffee on the boil.

Thank You For Listening - Cheers.