GPU Applications for Modern Large Scale Asset Management

GPU Club – University of Manchester

Dr. Daniel Egloff

QuantAlea & IncubeAdvisory

October 2, 2013
Outline

Introduction
Outline

Introduction

Mean Variance Portfolio Optimization
Outline

Introduction

Mean Variance Portfolio Optimization

Views via Entropy Approach
  Tail Views Example
  Large Scale Convex Optimization
  Large Scale Convex Optimization with GPUs
Introduction

Mean Variance Portfolio Optimization

Views via Entropy Approach
  Tail Views Example
  Large Scale Convex Optimization
  Large Scale Convex Optimization with GPUs

Bayesian Methods
Introduction

Mean Variance Portfolio Optimization

Views via Entropy Approach
  Tail Views Example
  Large Scale Convex Optimization
  Large Scale Convex Optimization with GPUs

Bayesian Methods

Dynamic CUDA with Alea.cuBase
About Us

- Software and solution provider
- Specialized in GPUs and HPC
- Focus on financial service industry
- Early adopter of GPUs in finance

Consulting in finance and risk management
- Servicing banks, insurance firms, trading firms
- Strong business experience
- Interdisciplinary teams
Portfolio Optimization

- Mean variance portfolio optimization (MVO) introduced by Markowitz
- Tradeoff between risk and performance
- Optimization problem

\[
    w_{\lambda}^* = \arg \max_{w, \mathbf{1}^T w = b} U_\lambda(w)
\]  

- Utility function \( U_\lambda \) is defined as

\[
    U_\lambda(w) = \mu^T w - \lambda w^T \Sigma w
\]

- Main inputs are means \( \mu \) and covariances \( \Sigma \) of asset returns
- Additional constraints on \( w \) require numerical solution
Critics

- MVO is highly unstable.
- Consider the optimal solution with budget constraints

\[ w^*_\lambda = \frac{1}{2\lambda} \Sigma^{-1}(\mu - \delta 1) \]  

with \( \delta \) the Lagrange multiplier

\[ \delta = \frac{1^\top \Sigma^{-1} \mu - 2\lambda b}{1^\top \Sigma^{-1} \mu} \]  

- Optimal weights are function of inverse covariance matrix.
- Small eigenvalues of \( \Sigma \) influence \( w^*_\lambda \) most.
- Amplification of small errors and noise.
Two main avenues

- Reducing the estimation errors of the input parameters
  - Resampling (Michaud, 1998)
  - Shrinkage approaches (Ledoit and Wolf 2003)
  - Market equilibrium combined with views (Black and Litterman 1992, Meucci)

- Shrinkage
  - Weight bounds
  - Penalization of the objective function
  - Regularization of input parameters
  - Constraints on MVO can be interpreted as mean and covariance modification (Jagannathan and Ma 2003)

Other alternative to move to risk budgeting allocation with integrated expected returns
Using Views

How to consistently incorporate expert views, beliefs and constraints into the asset return distribution?

- Views are statements to distort a prior distribution
- Resulting posterior distribution can be used in MVO, risk parity, ...
- Linear constraints on mean return, pioneered by Black and Litterman
- Discrete nonparametric entropy based approach (Meucci 2010)
- Two main approaches to incorporate views
  - Entropy based approach
  - Bayesian methods
Entropy Approach

- Maximum entropy principle (Jaynes 1957)
  - Constructive approach to produce probability distribution based on partial knowledge with minimal assumptions
- Minimum discrimination principle (Kullback and Leibler 1951)
  - Infer distribution s.t. it satisfies constraints on certain moments and is as hard as possible to differentiate from given prior distribution
- Minimum discrimination principle suitable methodology to incorporate views realized as linear, quadratic, or convex constraints for a prior distribution
Example

Consider \( d\mu = f(x)dx \), with density \( f(x) \), \( F(x) = \int_{-\infty}^{x} f(y)dy \).

Tail probability constraint: Given value \( a \), find density \( f^*(x) \) such that

\[
\int 1(x > a) f^*(x) dx = a
\]

Optimal density is given by

\[
f^*(x) = \frac{1(x \leq a) + e^{\lambda^*} 1(x > a)}{F(a) + e^{\lambda^*}(1 - F(a))} f(x)
\]

Here \( \lambda^* \) is Lagrange multiplier, can be calculated from \( f, a \).

Note that optimal density is discontinuous.
Figure: Optimal density for tail probability constraints

Figure: Multiple tail probability constraints
Explicit calculations as above only work for one-dimensional prior distributions.

To obtain a general framework, sample the probability distributions:

- Discrete sample space $\mathcal{X} = \{x_1, \ldots, x_m\}$
- Probabilities elements of the standard simplex

$$p = (p(x_1), \ldots, p(x_m))^\top = (p_1, \ldots, p_m)^\top \in \Delta^m \subset \mathbb{R}^m$$

Consider views as distortions of the prior distribution by changing probabilities $p_i$ of samples $x_i \in \mathcal{X}$.

Express views on feature functions $f_j : \mathcal{X} \to \mathbb{R}$, $j = 1, \ldots, n$.

Feature vector $\mathbf{f} = (f_1, \ldots, f_n)^\top : \mathcal{X} \to \mathbb{R}^n$.

Identify $\mathbf{f}$ with the matrix $F = (f_{ji}) \in \mathbb{R}^{n \times m}$, with $f_{ji} = f_j(x_i)$. 

---

Framework

QuantAlea
Linear Views

- Expectation of feature $f : \mathcal{X} \rightarrow \mathbb{R}$ w.r.t measure $q \in \Delta^m$ is

$$E_q[f] = \sum q_i f(x_i) = q^\top f$$

- Minimum discrimination principle for probabilities on $\mathcal{X}$ leads to

$$p^* = \arg \min_{q \in \Delta^m} q^\top (\log(q) - \log(p))$$

subject to constraints on features $f_j, \tilde{f}_j$

$$E_q[f_j] \leq c_j, \quad j = 1, \ldots, n, \quad E_q[\tilde{f}_j] = \tilde{c}_j, \quad j = 1, \ldots, \tilde{n}.$$
Optimization Problem

- Expectation views (10) lead to linear constraints

\[ p^* = \arg \min_{q \in \mathbb{R}^m_+} q^\top (\log(q) - \log(p)) \quad (11) \]
\[ q^\top 1 = 1 \]
\[ Fq \leq c, \tilde{F}q = \tilde{c} \]

- Need large number of samples for accurate representation of distributions with \( m \) in order \( 10^5 \) to \( 10^6 \)

- Apply Lagrange duality to solve the optimal solution
  - Strong duality, i.e. no duality gap
  - Converting problem of dimension \( m \) to much smaller problems of dimension \( n + \tilde{n} \) to find optimal Lagrange multipliers
Optimal Solution

Optimal solution to (11) is given by

\[ q^* = q^*(\lambda, \tilde{\lambda}) = p \cdot \exp(-F^T \lambda - \tilde{F}^T \tilde{\lambda}) \exp(-N(\lambda, \tilde{\lambda})) \]

where

\[ N(\lambda, \tilde{\lambda}) = \log(1^T p \cdot \exp(-F^T \lambda - \tilde{F}^T \tilde{\lambda})) \]

and \( \lambda, \tilde{\lambda} \) optimal Lagrange multiplier for \( Fq \leq c, \tilde{F}q = \tilde{c} \)

Note that constraints \( q \in \mathbb{R}_+^m \) and \( 1^T q = 1 \) are absorbed in solution expression

Here \( x \cdot y \) is element wise product of \( x \) and \( y \)
Extension to Variance Constraints

- How to extend to variance and covariance constraints?
  - Consider a variance constraint for a feature $f$
    \[ q^\top f^2 - (q^\top f)^2 \leq c \]
  - Poor mans approach is to linearize the variance constraints
    \[ q^\top f^2 \leq c + (p^\top f)^2 \]
    to get a second moment constraint
  - Often the linearized version is not good enough, also because MVO is very sensitive to covariances
Extension to Variance Constraints

- Can we generalize to quadratic covariance constraints?
- For some quadratic constraints like $q^\top f^2 - (q^\top f)^2 \geq c$ we can generalize the above Lagrange duality approach.
- The resulting dual objective function contains quadratic over linear terms.
- It is numerically very sensitive so that no precise solutions result.
- Moreover upper variance bounds $q^\top f^2 - (q^\top f)^2 \leq c$ cannot be handled with duality because dual Lagrangian is unbounded.
- We need another approach.
We can use large scale convex optimization methods to solve directly the primal problem.

Newton methods or interior point methods are not applicable for large scale problems.

Several accelerated first order methods (FOM) for convex optimization algorithms have been developed.

Modern algorithms adjust to the problem’s geometry.

Examples:
- Mirror descent algorithm
- Level bundle methods
- Bundle mirror descent
- Many different variations
Exploiting Geometry

- Gradient methods regularized with quadratic proximity term
  \[
  y = \arg \min_{x \in \mathcal{X}} \{ \gamma \langle \xi, \nabla f(x) \rangle + c \| \xi - x \|^2 \}
  \]

- Replace \( \| \cdot \|^2 \) with some distance like function \( D(\cdot, \cdot) \) that better exploits the geometry of \( \mathcal{X} \) through projection like map
  \[
  p(\xi, x) = \arg \min_{u \in \mathcal{X}} \{ \langle \xi, u \rangle + D(u, x) \}
  \]
  which should be simple to calculate

- Bregman type distance based on kernel \( \omega : \mathcal{X} \to \mathbb{R} \)
  \[
  D_\omega(x, y) \equiv \omega(x) - \omega(y) - \langle x - y, \omega'(y) \rangle
  \]
Bregman Distances

- Using \( \omega(u) = \frac{1}{2} \|u\|^2 \) gives Euclidian norm distance

\[
D_\omega(x, y) = \frac{1}{2} \|x - y\|^2
\]

- The entropy \( \omega(x) = x^\top \log(x) \) on \( \mathcal{X} = \Delta^n \) leads to

\[
D_\omega(x, y) = x^\top \log \left( \frac{x}{y} \right)
\]

which is 1-strongly convex with respect to the \( L_1 \) norm

\[
D_\omega(x, y) \geq \frac{1}{2} \|x - y\|_1^2 \quad \forall x, y \in \Delta^n
\]
Mirror Descent Setup

- Convex problem $\min_{x \in \mathcal{X}} f_0(x)$ subject to $f_1(x) \leq 0$
- $\mathcal{X} \subset E$, closed convex subset of f.d. vector space $E$
- $\| \cdot \|$ norm on $E$, $\| \cdot \|_*$ dual norm on $E^*$
- Duality pairing $\langle \cdot, \cdot \rangle : E^* \times E \rightarrow \mathbb{R}$
- Bregman distance $D_\omega(x, y)$ from kernel $\omega : \mathcal{X} \rightarrow \mathbb{R}$
- Prox mapping $\text{Prox}_x : E^* \rightarrow \mathcal{X}^*$

$$\text{Prox}_x(\xi) = \arg \min_{u \in \mathcal{X}} \{\langle \xi, u \rangle + D_\omega(u, x)\}$$
Mirror descent with constraints is a simple FOM

Initial point \( x_1 = x_\omega = \arg\min_{x \in \mathcal{X}} \omega(x) \)

For \( t = 1, \ldots, N \) do

- If \( f_1(x_t) \leq \text{tol} \) then \( i(t) = 0 \) else \( i(t) = 1 \)
- Select step size \( \gamma_t \)
- Update \( x_{t+1} = \text{Prox}_{x_t}(\gamma_t f'_{i(t)}(x_t)) \)

Approximate solution after \( N \) steps

\[
\hat{x}_N = \arg\min_{x \in \{x_t \mid i(t) = 0\}} f_0(x)
\]

Possible choice \( \text{tol} = \gamma \| f'_1(x_t) \|_* \) and \( \gamma_t = \gamma \| f'_{i(t)}(x_t) \|_*^{-1} \)
Simplex Setup

- Optimization problem (11) fits the simplex setup
- Prox mapping allows analytical expression
  \[ \text{Prox}_x(\xi) = \left( x^\top e^{-\xi} \right)^{-1} x \cdot e^{-\xi} \]
- Initial point \( x_\omega = n^{-1} 1_n \) the uniform distribution
- Dual norm of \( \| \cdot \|_1 \) is \( \| \cdot \|_\infty \)
- More generally, the regularized entropy
  \[ \omega(x) = (x + \sigma)^\top \log(x + \sigma), \quad \sigma = \delta n^{-1} 1_n \]
  for some small value \( \delta \) is used
- Regularized entropy also allows efficient prox map calculation
Mirror Descent on GPU

- How to implement mirror descent algorithms with standard setup for very large $n$ on GPUs?
- Various parallel elements
  - Parallel coordinate-wise calculation of gradients $f'_0(x)$ and $f'_1(x)$ for objective function and constraints
  - Vector reductions to calculate objective value, norms and scalar products
  - Parallel coordinate-wise calculation of prox vector, search points and execution of gradient step
- Gradient steps are serial and cannot be parallelized
- Is this enough parallelism?
- What are the problems of a CUDA implementation?
Schematic Gradient Step

Gradient step:
- Calculate objective value
- Calculate constraints and objective values and gradients
- Calculate dual norm of gradients and update approximate solution
- Calculate unnormalized prox vector
- Calculate prox normalization factor
- Normalize prox vector and update search point

Test convergence

No

Parallel reduction algorithm (2 kernels)
Parallel vector transform kernel
Efficient Parallel Reduction

- Several parallel reductions have to be executed
  - Sum reduction for normalization
  - Max reduction for dual norm
  - Scalar product in prox map and objective function
- Use shared memory
- Process multiple elements per thread
- Use warp level multi reduction
- Matrix reductions along columns for multiple vectors in parallel
Kernel Fusion and Tuning

- Usually kernel launch time ($\approx 50\mu s$) not an issue
- For algorithms with many iterations, this becomes crucial
- Use kernel fusion technique to reduce number of kernel calls per gradient step
  - Obvious kernel fusion for reductions of multiple stacked vectors of same length
  - Still 9 kernels to call per iteration
  - Reorganize threads for reduction suitable to reduce kernel calls further, at expense of much more complicated code
- Avoid data copy between host and device, keep all data on GPU
- Minimize number of arguments to pass to each kernel
- With compute capability 3.5 use dynamic parallelism (do not expect great improvements!)
Constraints calculation kernel for multiple constraints must be efficient

Cannot afford complicated data structures to handle a large constraints variation

Express constraints in terms of a DSL

Transform the DSL description of constraints into a constraints evaluation kernel

Identify various expressions which can be precalculated once

Ideal application of Alea.cuBase and F#
Bayesian Approach

- Bayesian methods treat the model parameters as random variables
- Here the views go into the prior distributions of the model parameters
- Historical data is then used to update the prior distribution
- Not as flexible as entropy approach
- Does not rely on optimization hence less fragile, but sampling error adds additional noise
- Work horse is a parallel Metropolis Hastings Markov chain Monte Carlo sampler
Parallelization of Metropolis Hastings through multiple parallel chains

Further parallelization through pre-fetching

Require efficient parallel random number generation and branch free methods to sample from proposal distribution

For efficiency reason the transform kernels should be compiled

Ideal application for our new F# based CUDA software stack
Parallel Metropolis Hastings

GPU Metropolis Hastings sampler generation from generic target and proposal distribution expressions

```ml
let inline mhsamples (targetpdf : Expr<'T -> 'T>)
       (proppdf : Expr<'T -> 'T -> 'T -> 'T>)
       (proprnd : Expr<'T -> 'T -> 'T -> 'T>) = cuda {

    let! proppdf = proppdf |> Compiler.DefineInlineFunction
    let! proprnd = proprnd |> Compiler.DefineInlineFunction
    let! targetpdf = targetpdf |> Compiler.DefineInlineFunction

    let! kernel =
        <| fun (steps:int) (scale:'T) (initial:deviceptr<'T>)
            (numbers:deviceptr<'T>) (result:deviceptr<'T>) ->
            let idx = blockIdx.x * blockDim.x + threadIdx.x
            let nchains = gridDim.x * blockDim.x
            ...
```

QuantAlea
The cuda work flow is used to specify all GPU resources

The kernel is coded inside a code quotation `<@ . . . @>`

Most of the known CUDA keywords and constructs can be used to code kernels

```ocaml
/// Simple vector transform
let transform op = cuda {
    /// Definition of GPU resources
    let! kernel =
        `<@ fun (n:int) (input:deviceptr<float>)
            (output:deviceptr<float>) ->
            let start = blockIdx.x * blockDim.x + threadIdx.x
            let stride = gridDim.x * blockDim.x
            let mutable i = start
            while i < n do
                output.[i] <- (%op) input.[i]
                i <- i + stride
        @> |> Compiler.DefineKernel

    ... }
```
Launch Orchestration

- The work flow returns an Entry object which orchestrates the data transfer and GPU computation

```ml
/// Simple vector transform
let transform op = cuda {
    /// Definition of GPU resources
    . . .

    return Entry(fun (program:Program) (input:float[]) ->
        let kernel = program.Apply(kernel)
        use input = program.Worker.Malloc(input)
        use output = program.Worker.Malloc(input.Length)
        
        let blockSize = 256
        let numSm = program.Worker.Device.Attributes.MULTIPROCESSOR_COUNT
        let gridSize = min numSm (Util.divup input.Length blockSize)
        let lp = LaunchParam(gridSize, blockSize)
        
        kernel.Launch lp n input.Ptr output.Ptr
        output.Gather() }
```
The GPU resource definitions can be compiled at runtime to executable CUDA code.

```csharp
let worker = Engine.Worker.Default

/// Compile and load GPU resources onto a GPU card, use
/// a lambda function to be spliced into the transform code
let transform =
    (transform <@ fun x -> x*x @>) |> Util.load Worker.Default

/// Create some data and execute the GPU computations
let input = Array.init 10 (fun i -> float(i))
let output = transform.Run input

printfn "%A" output
```
Alea.cuBase at a Glance

- Alea.cuBase is quick, as fast as compiled C++
- Dynamically create GPU code on the fly
- Interactive CUDA scripting
- Increase your coding productivity with type inference, IntelliSense support
- Easy deployment, just one assembly, no need for any further NVIDIA CUDA tools
- Advanced CUDA 5.5 features such as device linking
- Full debugging capabilities in development