Optimization of Financial Models Using Evolutionary Algorithms and GPU Computing

Michal Hojčka, Riccardo Gismondi

R7 CORP k.s.

30.5.2019, Modern Tools for Financial Analysis and Modeling Conference, Bratislava

OVERVIEW

- GPU computing
- Evolutionary Algorithms
- Application for Financial Models

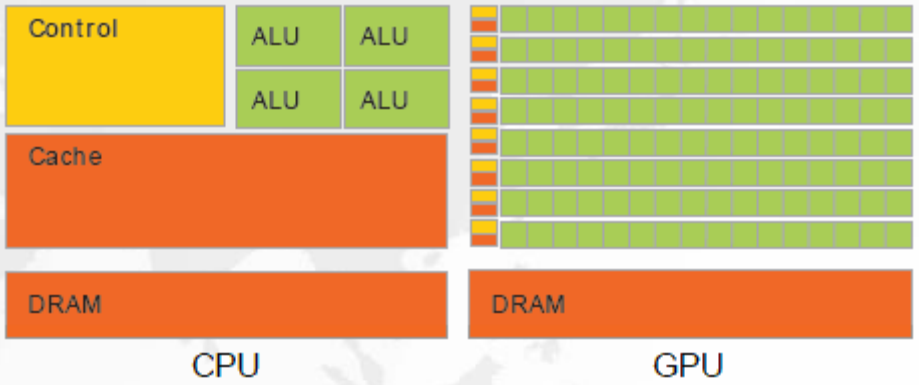
GPU COMPUTING

GPU COMPUTING IN MATLAB

- Use Parallel Computing Toolbox (possibly also GPU Coder)
  - No CUDA programming needed
  - Lot of existing predefined functions
- Compile native code as mex file in order to be used in MATLAB
  - More control over the code
  - We understand what is happening inside
  - Possible use in other programming languages
  - ➢ We choose this approach

GPU VS CPU – ARCHITECTURE

- CPU: specialized for flow control and fast serial computation (optimized for latency)
- GPU: specialized for compute-intensive, highly parallel computation needed for graphic rendering (optimized for throughput)
- ALU: arithmetic logic unit
- Cache: fast temporary memory
- DRAM: main memory
- Control: flow control unit



GPU VS CPU – FLOPS

- FLOP: floating-point operation per second, measure of raw computational power

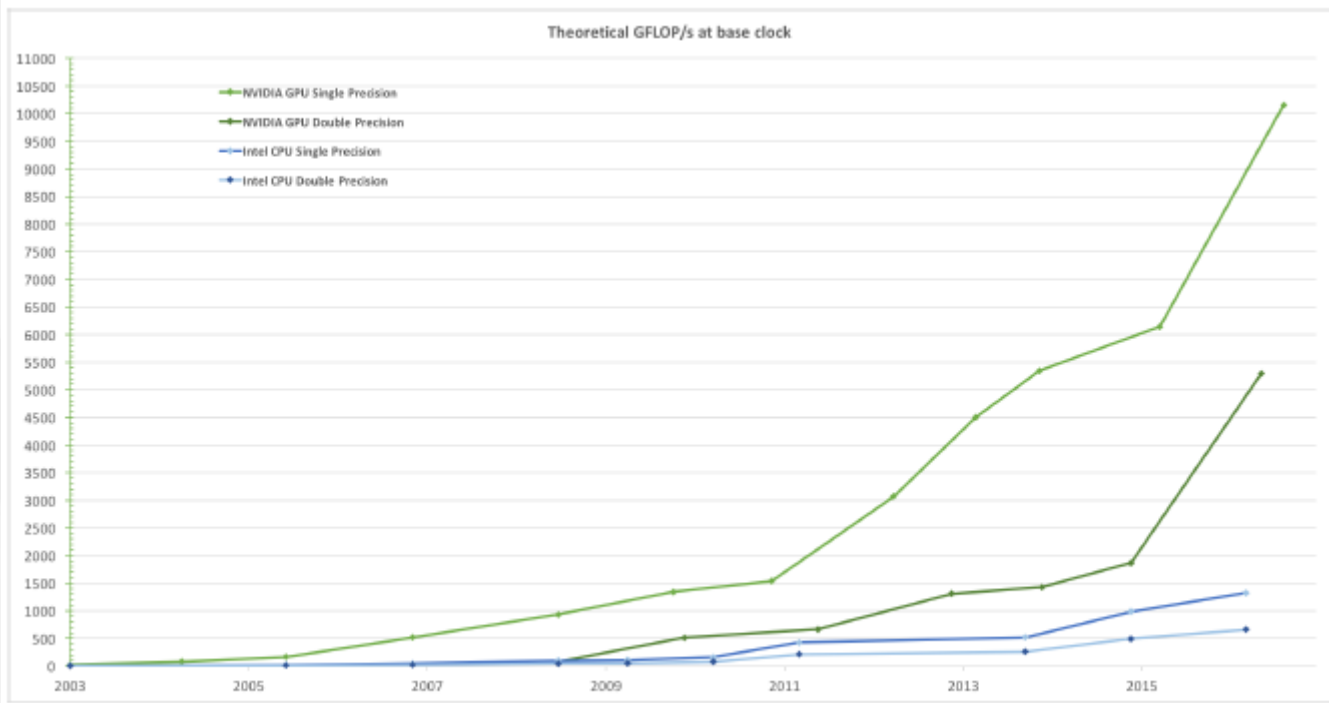


Figure 1 Floating-Point Operations per Second for the CPU and GPU

GPU VS CPU - MEMORY BANDWIDTH

- Memory Bandwidth: amount of data that can be theoretically processed per second

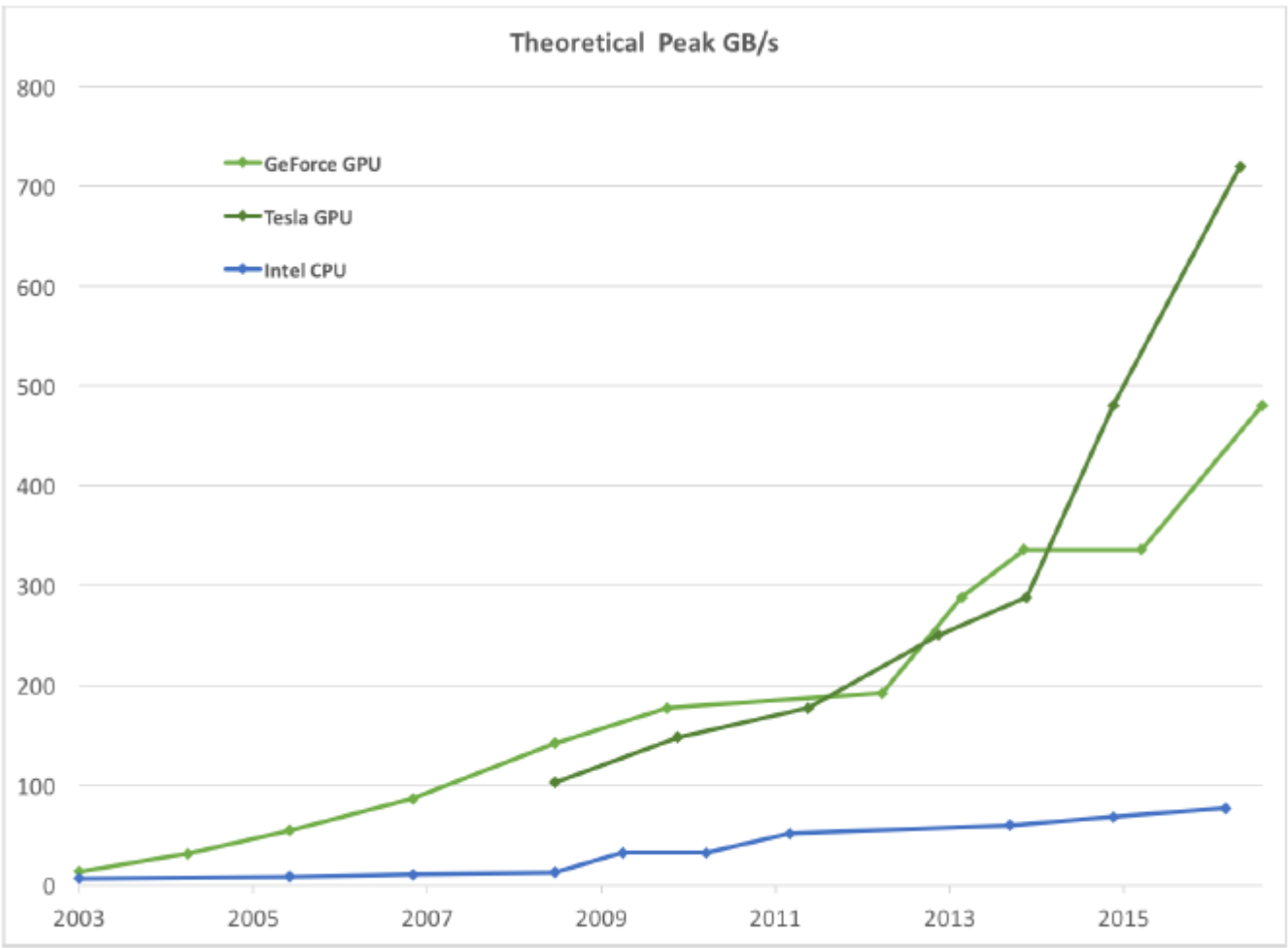


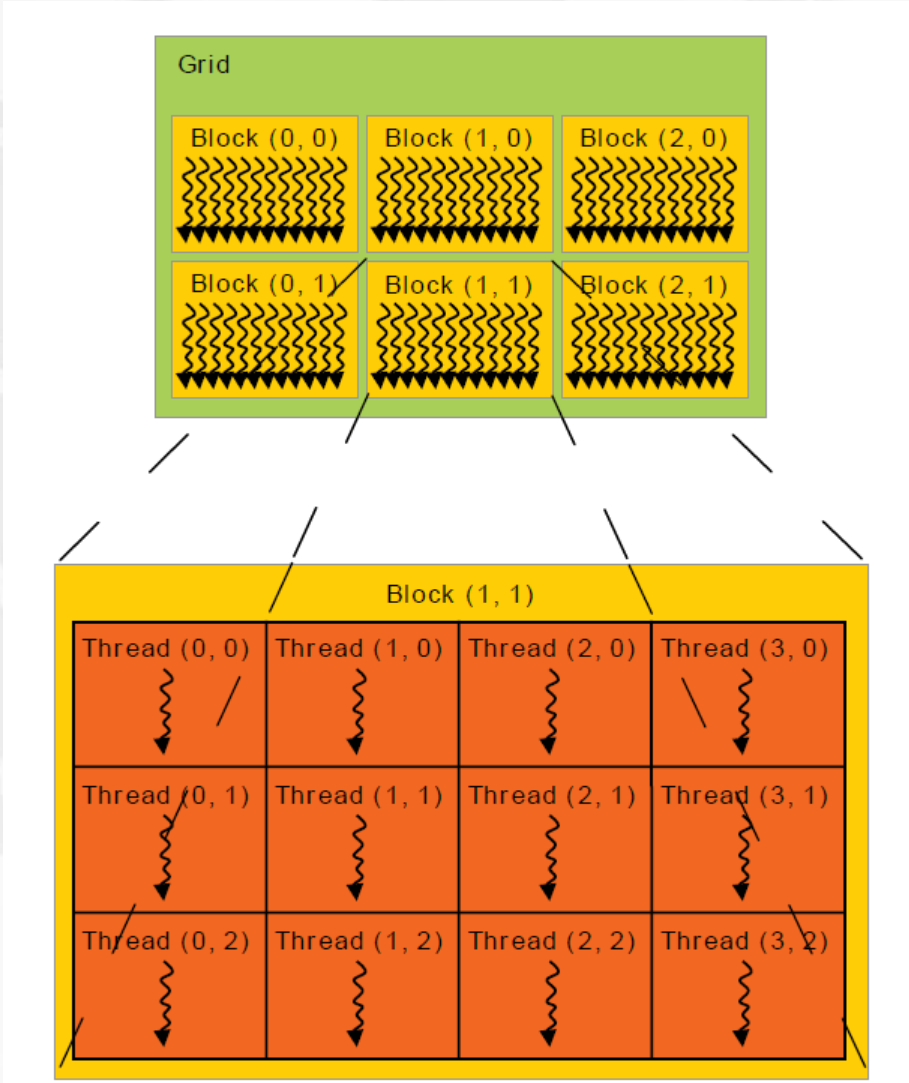
Figure 2 Memory Bandwidth for the CPU and GPU

CUDA

- Stands for Compute Unified Device Architecture
- Introduced by NVIDIA in 2007
- Extension to the C language that allows to program GPU without need to learn complex programming concepts or to use graphic primitive types
- CUDA Toolkit: can be downloaded from NVIDIA webpage

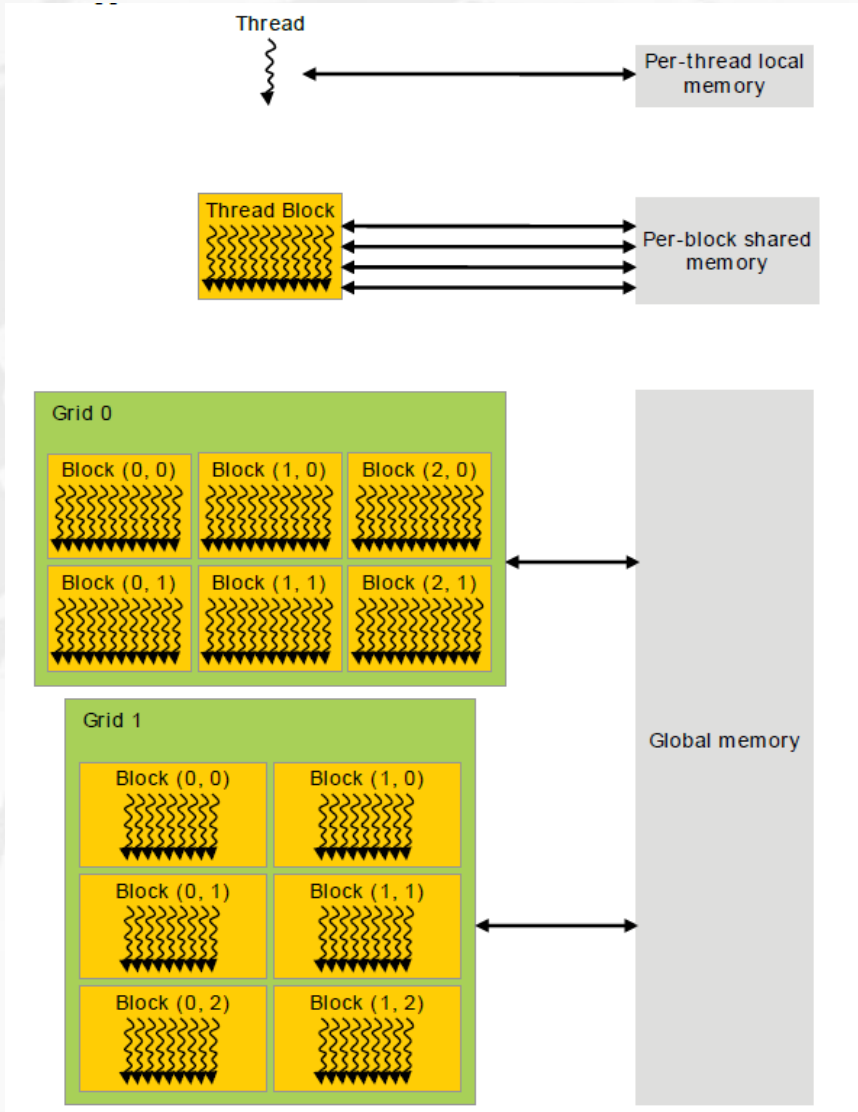
PROGRAMMING MODEL

- SIMT Architecture – single-instruction, multiple-thread
- We write program (kernel) for one thread -> it will be executed on many threads
- Block of Threads – max 1024
- Grid of Blocks



GPU HARDWARE – MEMORY

- Per-thread local memory: very fast, very small
- Per-block shared memory: can be used within each thread-block, little bit slower
- Global memory: can be used from anywhere on GPU, much slower
- Copying from CPU to GPU Global memory is very time-consuming and ineffective



STREAMING MULTIPROCESSORS

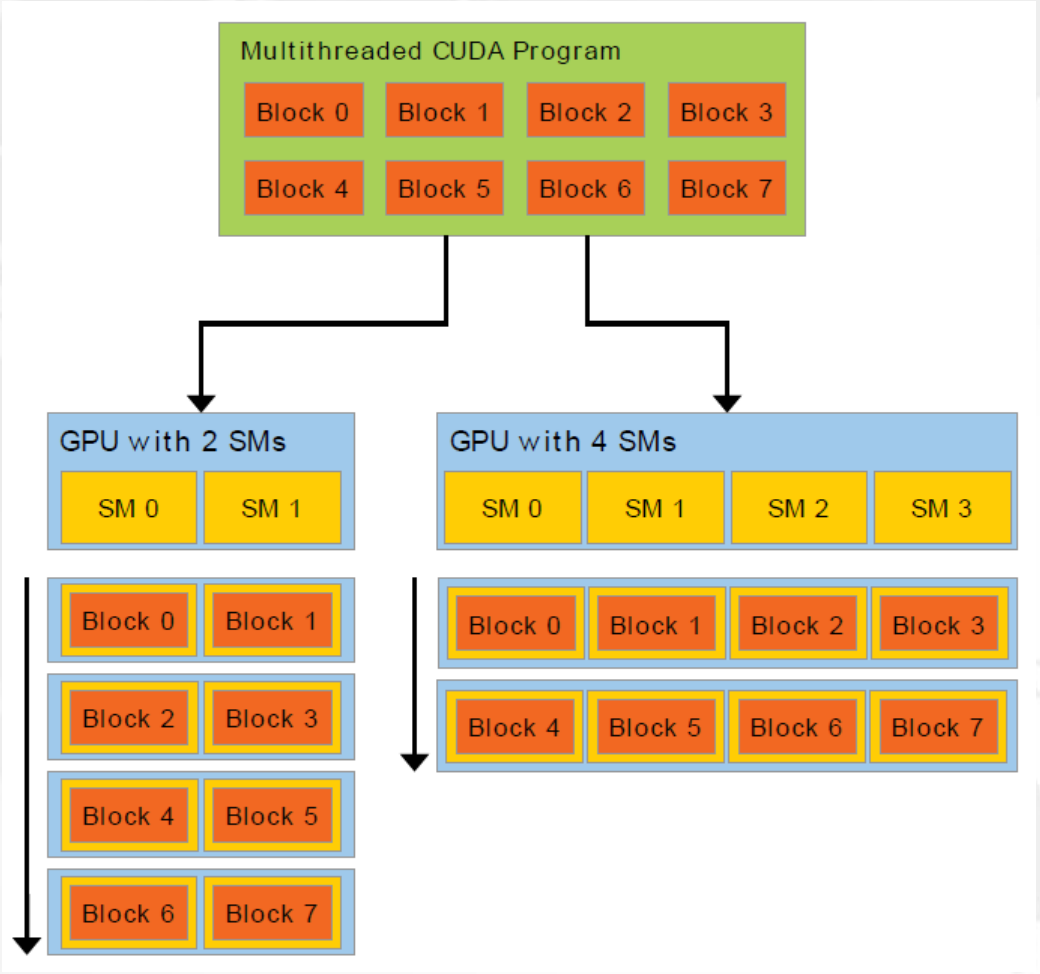
- Manage the execution of Threads, memory access and the distribution of arithmetic operations on the CUDA Cores
- Each Thread Block runs on a single SM, each SM can manage multiple Thread Blocks, depending on the available memory
- There can be up to ~20 SMs on the graphic card, depending on the model

CUDA CORES

- Unit for performing arithmetic operations
- Their architecture depends on the Compute Capability of the GPU
- Typical NVIDIA GPU has 100s – 1000s CUDA Cores

SCALABILITY

- Compatibility across multiple devices
- Same code can run from smartphones with 2 SMs to newest GPUs with 10s of SMs



GPU CARDS

- Scientific GPUs:
  - Pros: better computing capability, optimized to calculate with double precision
  - Cons: much more expensive, cannot be used for monitor
- Gaming GPUs:
  - Pros: cheaper, can be used also for monitor
  - Cons: expensive double precision computing
- Example: GeForce GTX 1050 card (quite basic gaming card): CUDA Capability version 6.1, 5 SMs, 128 CUDA Cores each -> 640 CUDA cores

SINGLE VS. DOUBLE PRECISION

- Gaming GPUs are optimized for single-precision calculations
- From the performance point of view is important to use double precision only where necessary
- Example: GeForce GTX 1050 card have 32-times smaller double precision computing capability, which means ~10 times slower performance

	Compute Capability							
	3.0, 3.2	3.5, 3.7	5.0, 5.2	5.3	6.0	6.1	6.2	7.x
32-bit floating-point add, multiply, multiply-add	192	192	128	128	64	128	128	64
64-bit floating-point add, multiply, multiply-add	8	64	4	4	32	4	4	32

CUDA EXAMPLE - MATRIX MULTIPLICATION

- AB = C (each matrix has shape N x N)
- Serial code: N^3 arithmetic operations (N multiplications for each of N^2 elements)
- Parallel code: N arithmetic operations done by each of N^2 threads
- Actual speedup will depend on the amount of possible parallel threads and the effectivity of memory access

CUDA CODE COMPILATION FOR MATLAB

- C/C++/CUDA code can be compiled to be used in MATLAB as mex files (.mexw64)
- Mex, Nvmex, CUDA mex etc. did not work for us
- Exporting DLL from Visual Studio with the help of mex libraries did work for us

EVOLUTIONARY ALGORITHMS

OPTIMIZATION PROBLEM

- General optimization problem have the following form:

$$\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = \left[f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x}) \right]^T$$

subject to

$$g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m_{\text{ieq}},$$

$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, m_{\text{eq}}.$$

OPTIMIZATION TYPES

- Deterministic search methods:
  - Gradient methods: e.g. Quasi-Newton methods (DFP, BFGS)
  - Gradient-free methods: e.g. Direct methods, Surrogate methods
  - Require some assumptions about the smoothness or continuity of the objective function
  - Extremely dependant on the starting point in case of multimodal functions
- Stochastic search methods:
  - Random search
  - Evolutionary algorithms

EVOLUTIONARY ALGORITHMS

- Motivated by the natural processes, lot of different types, such as:
  - Genetic Algorithm (GA)
  - Evolutionary Programming
  - Evolution Strategies (ES)
  - Genetic Programming
  - Simulated Annealing
  - Particle Swarm Optimization (PSO)
  - Ant Colony Optimization

EVOLUTIONARY ALGORITHMS

- No assumptions regarding the objective function, ‘black box’ optimization
- General structure of the algorithm:
  - Start with random initial population
  - Evaluate the fitness of the individuals based on the objective function
  - Create new generation based on the population fitness
  - Repeat last two steps for predefined number of iterations or until our requirements are met

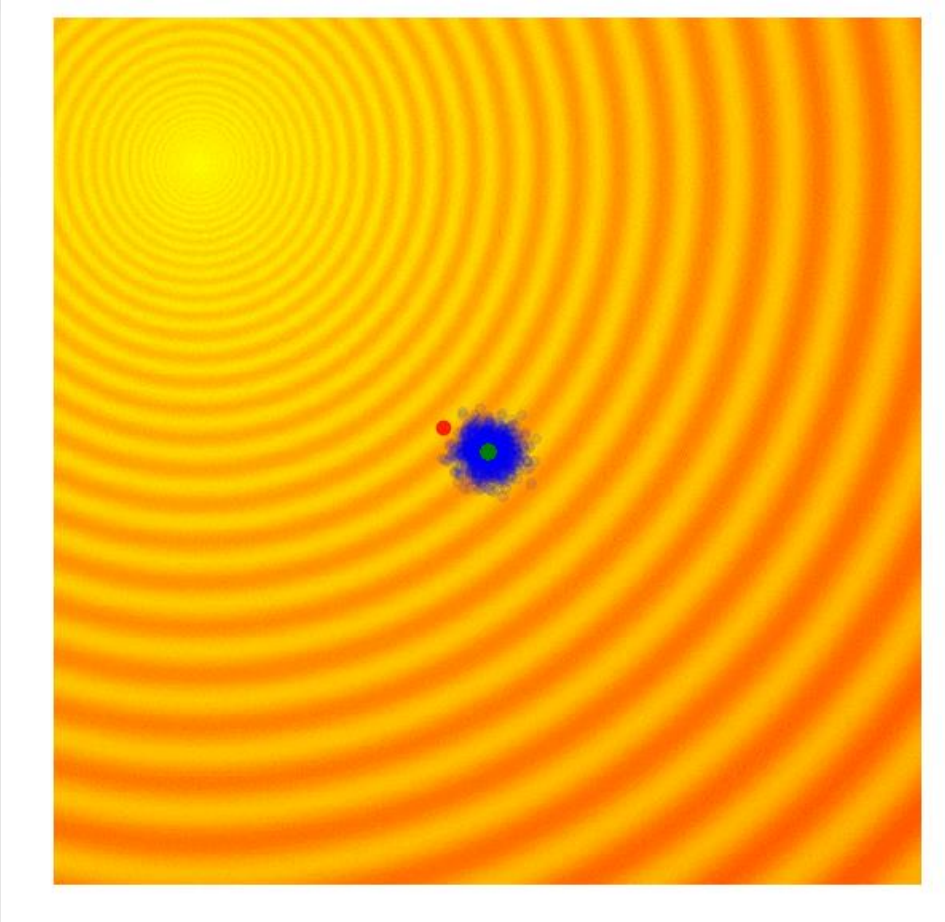
GENETIC ALGORITHM

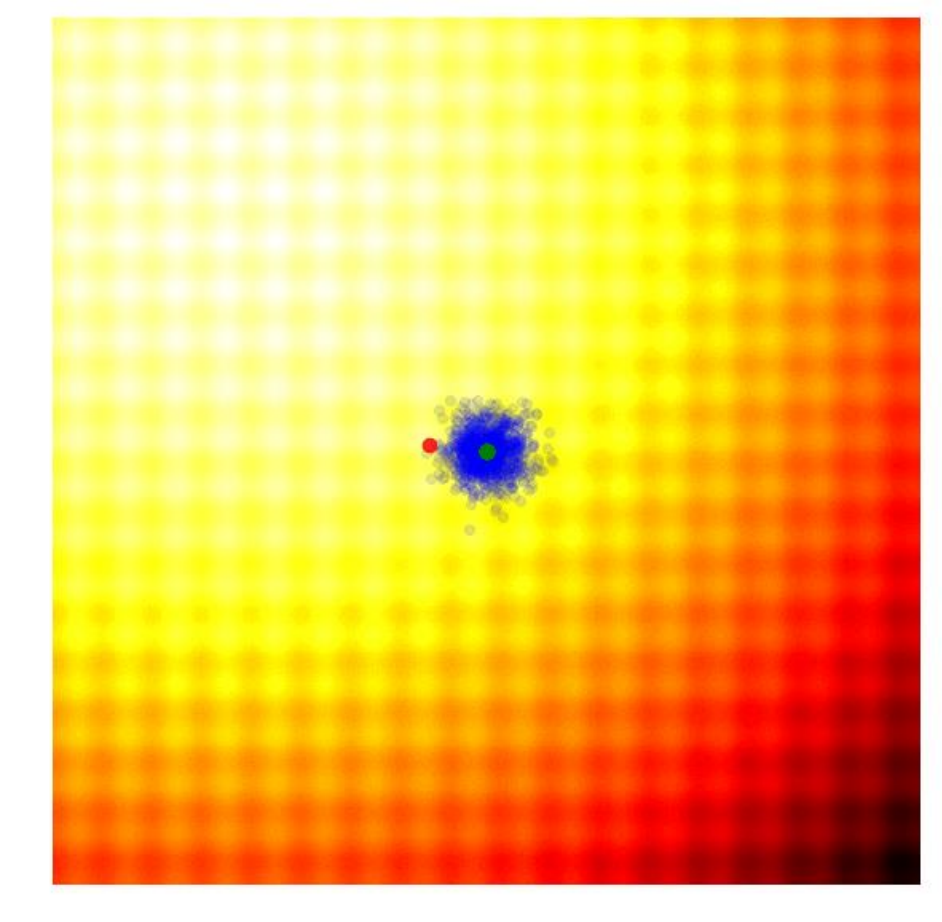
- Method that mimics the process of natural evolution
- Oldest class of Evolutionary Algorithms (Holland - 1975)
- Originally, chromosomes were binary vectors
- Later, real-coded version appears, which works also with continuous real vectors

GENETIC ALGORITHM

- Each iteration we:
  - Evaluate fitness of each individual based on the objective function
  - Select parents for the new generation based on this fitness
  - Create children using the crossover operation on the parents
  - With small probability mutate each element of the children
  - Replace the old population with the new one (we can keep some number of elite individuals)

GENETIC ALGORITHM

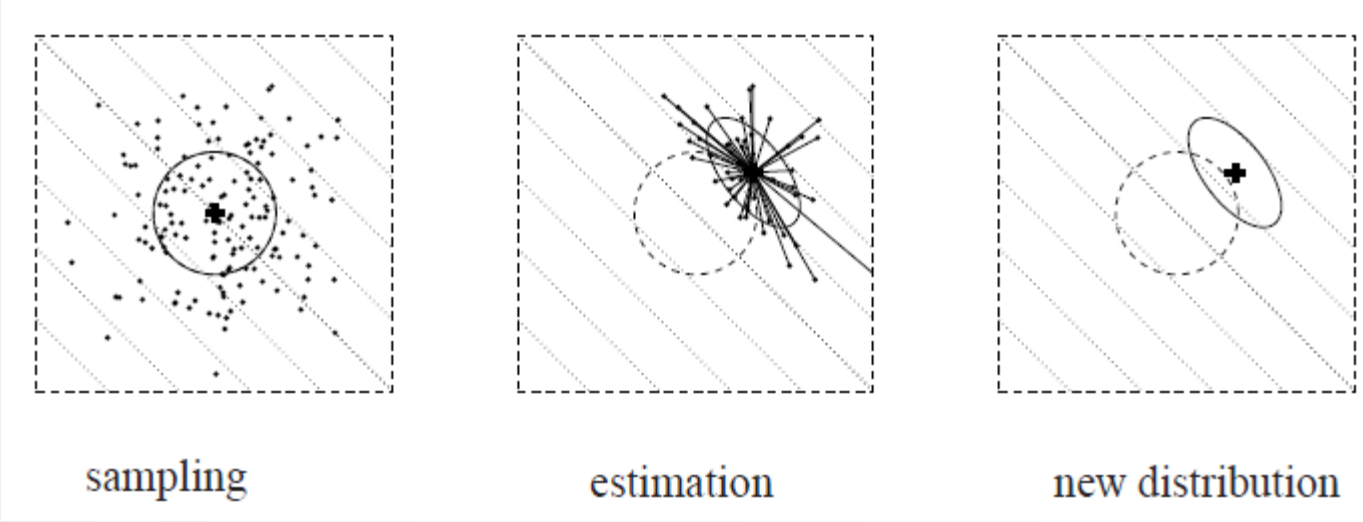




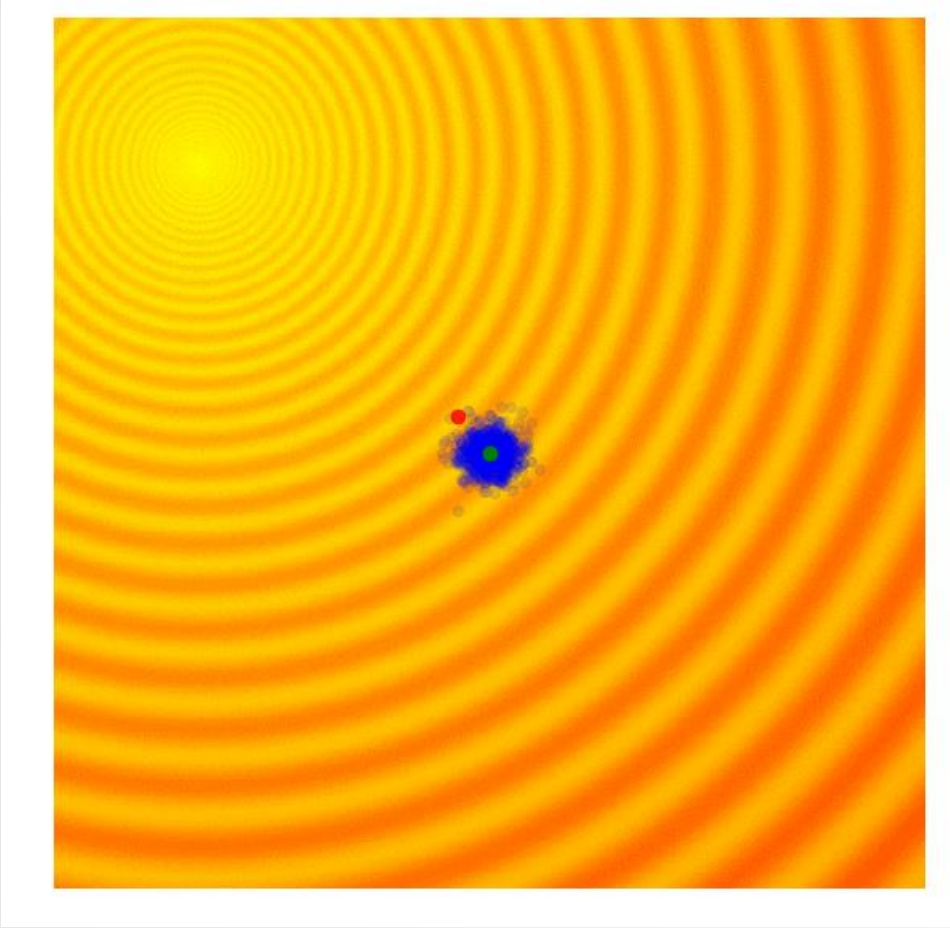
From http://blog.otoro.net/2017/10/29/visual-evolution-strategies/ by David Ha

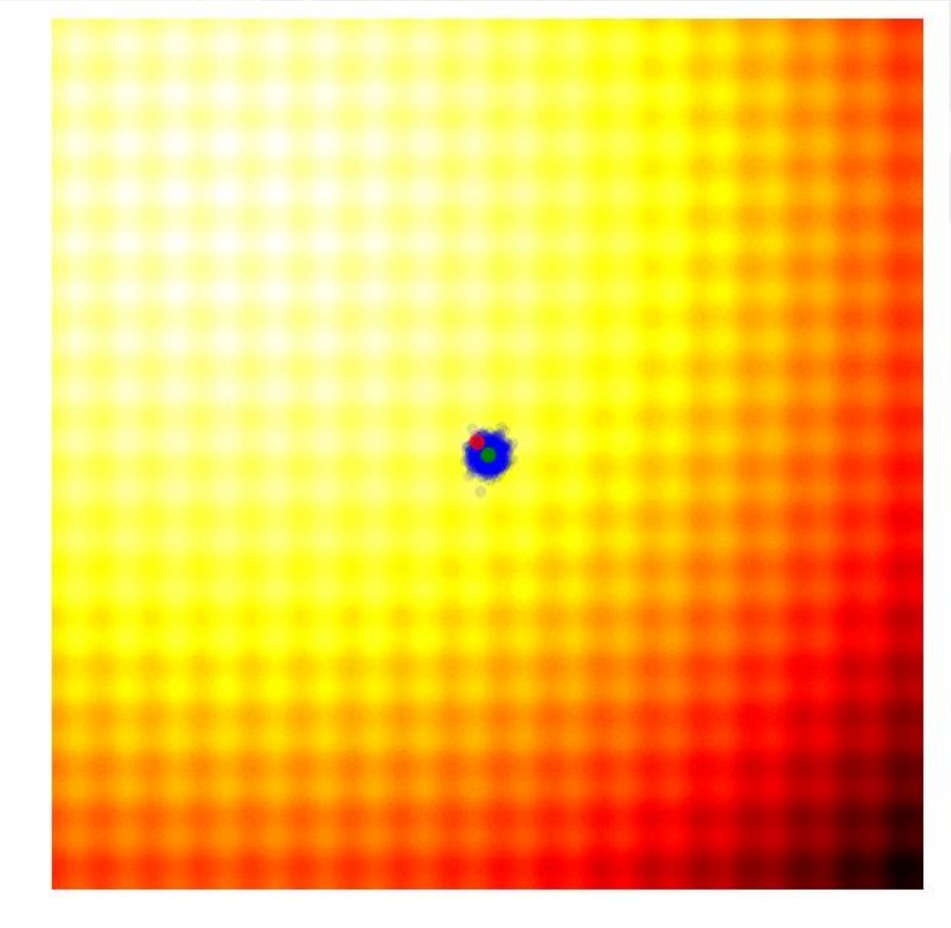
CMA – ES

- Stands for Covariance Matrix Adaptation Evolution Strategies
- Each iteration we:
  - Generate random points based on the covariance matrix and the starting point
  - From these points choose the ones with ‘good’ fitness
  - Choose next starting point as weighted average of ‘good’ points
  - Update covariance matrix based on ‘good’ points



CMA – ES



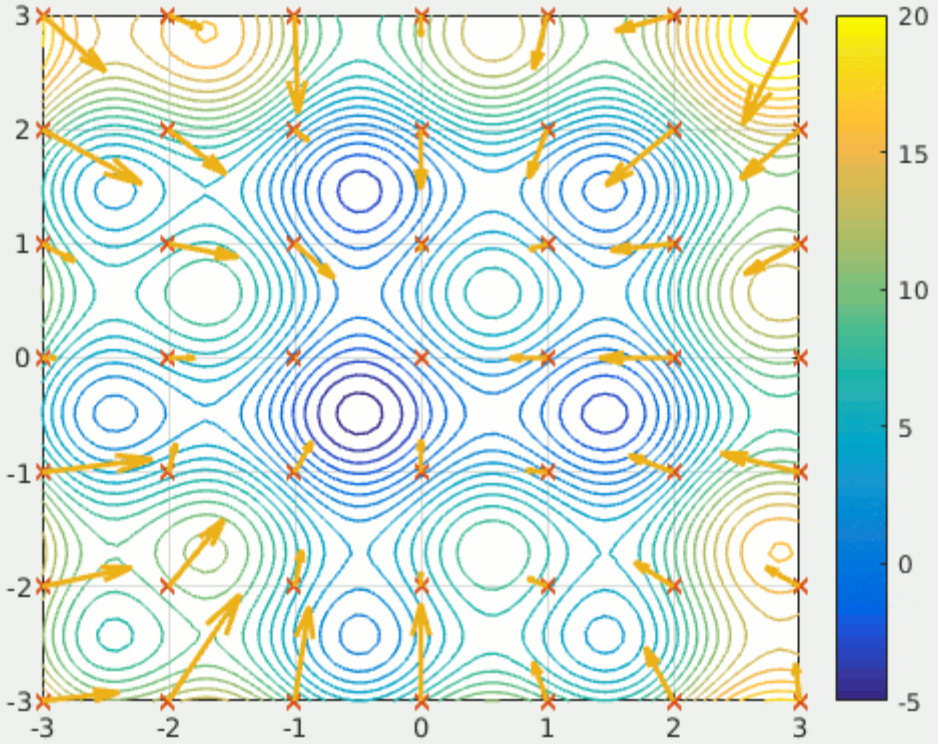


From http://blog.otoro.net/2017/10/29/visual-evolution-strategies/ by David Ha

PARTICLE SWARM OPTIMIZATION

- Imitation of human/animal social behaviour
- Swarm of particles, each particle is defined by its position and velocity
- Each iteration, velocity is adjusted to take into account:
  - Actual velocity
  - Each particle's best position in the past
  - Best actual position of any 'neighbour' particle
  - Best overall achieved position of the whole swarm
- New position is obtained using this velocity

PARTICLE SWARM OPTIMIZATION



From https://commons.wikimedia.org/wiki/File:ParticleSwarmArrowsAnimation.gif

PARALLELIZATION OF EA

- Evaluation of objective function for each individual in the population is naturally parallel operation
- Same holds for selection, crossover and mutation in Genetic Algorithm
- It is also possible to take advantage in sorting, matrix multiplication and other procedures
- It is possible to use much larger populations/more iterations during the same time

➢ GPU computing can be utilized very efficiently

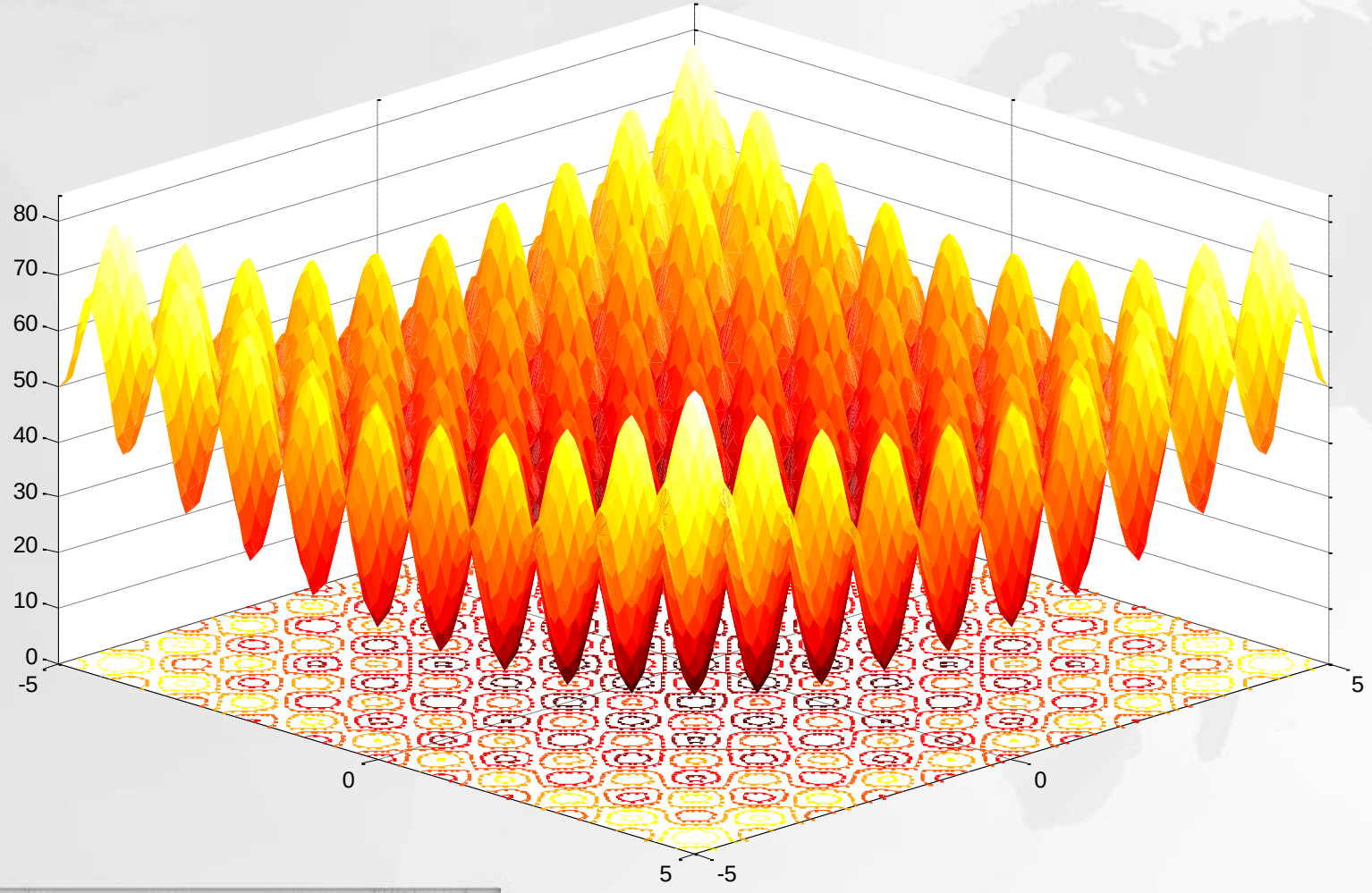
HYBRID OPTIMIZATION

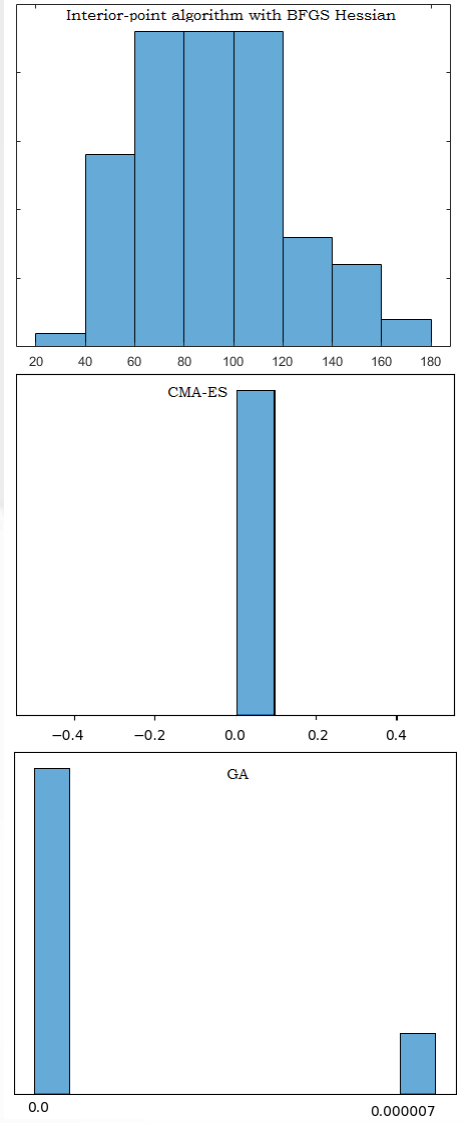
- Evolutionary Algorithms: very robust, can get out of a local minima
- Gradient-based algorithms: very good at finding local minima

➢ Best practice: find very good & robust starting point using Evolutionary Algorithms and then improve it using deterministic gradient-based algorithm

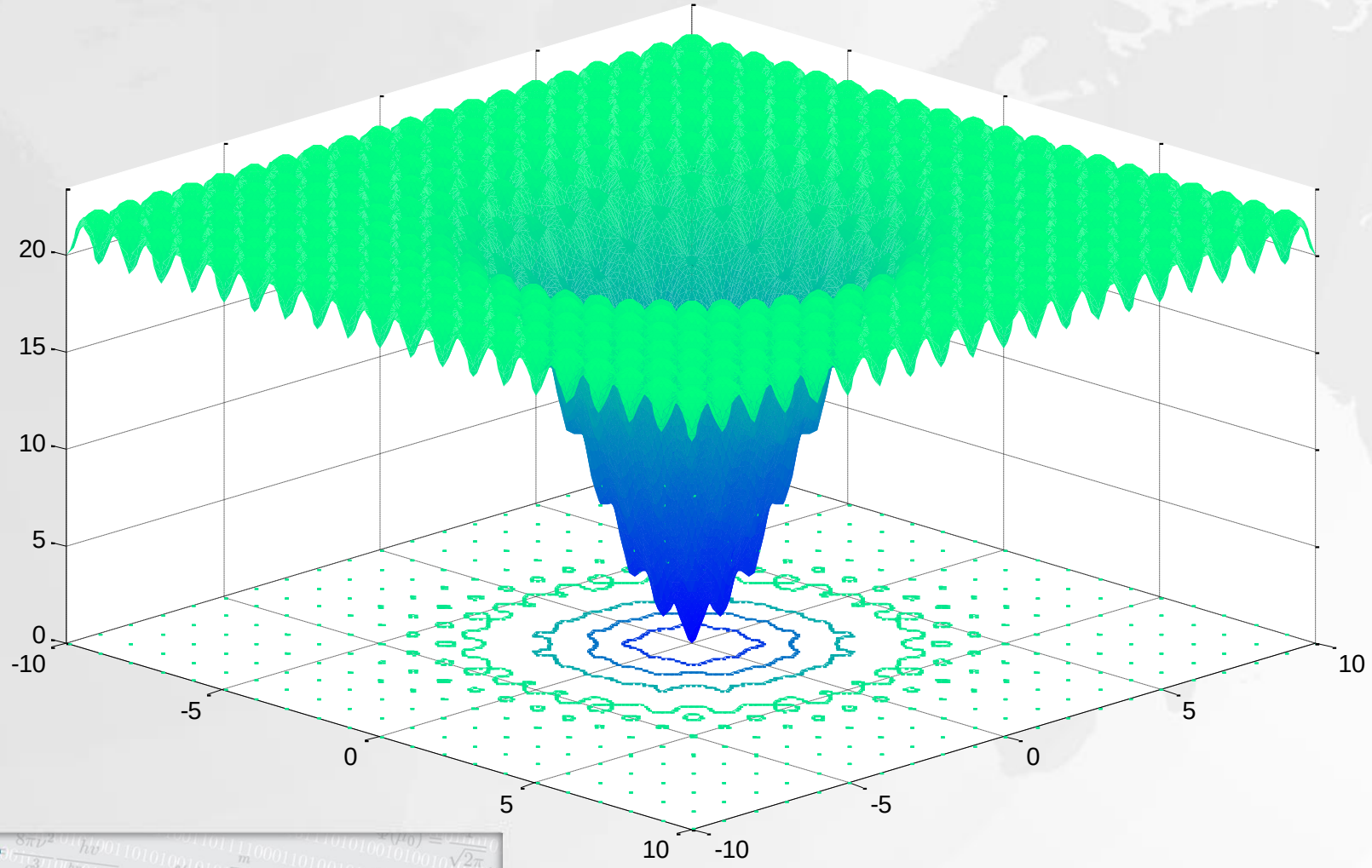
APPLICATION TO FINANCIAL MODELS

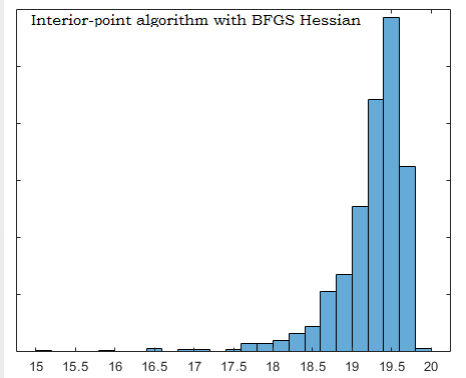
RASTRIGIN FUNCTION MINIMIZATION

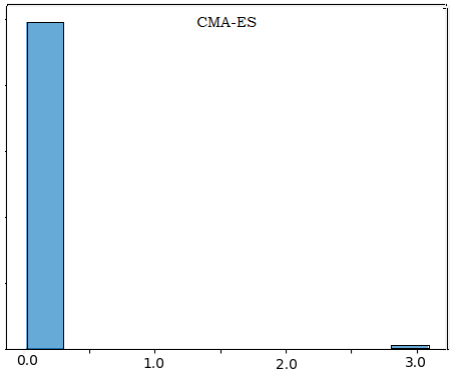


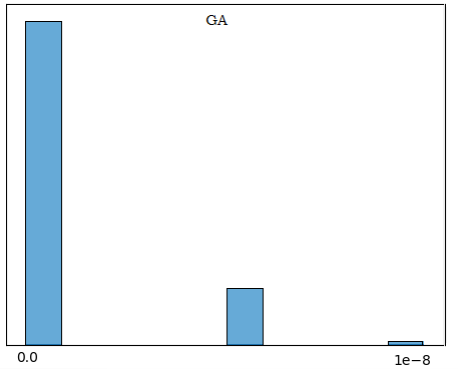


ACKLEY FUNCTION MINIMIZATION





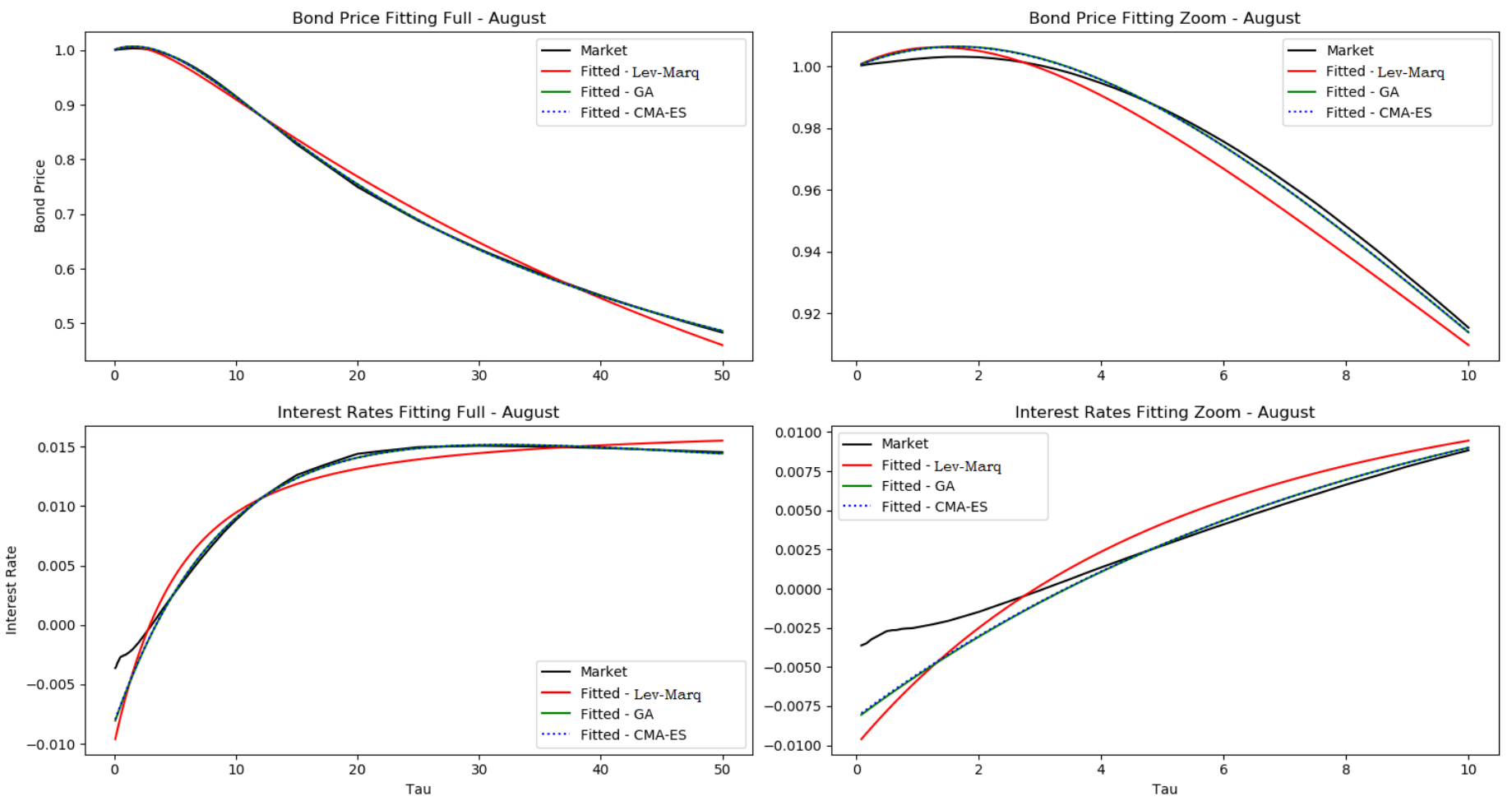




VASICEK MODEL

- Short rate: $r_t = \kappa(\theta - r_t)dt + \sigma dW_t$
- In order to calibrate we try to fit calculated bond prices to market bond prices (or calculated rates to market rates)
- Vasicek bond price:
  - $P(t,T) = A(t,T) * \exp(-r(t) * B(t,T))$
  - $B(t,T) = \frac{1-\exp(-k(T-t))}{k}$
  - $A(t,T) = \exp\left\{\left(\theta - \frac{\sigma^2}{2k^2}\right)(B(t,T) - T + t) - \frac{\sigma^2}{4k}B^2(t,T)\right\}$

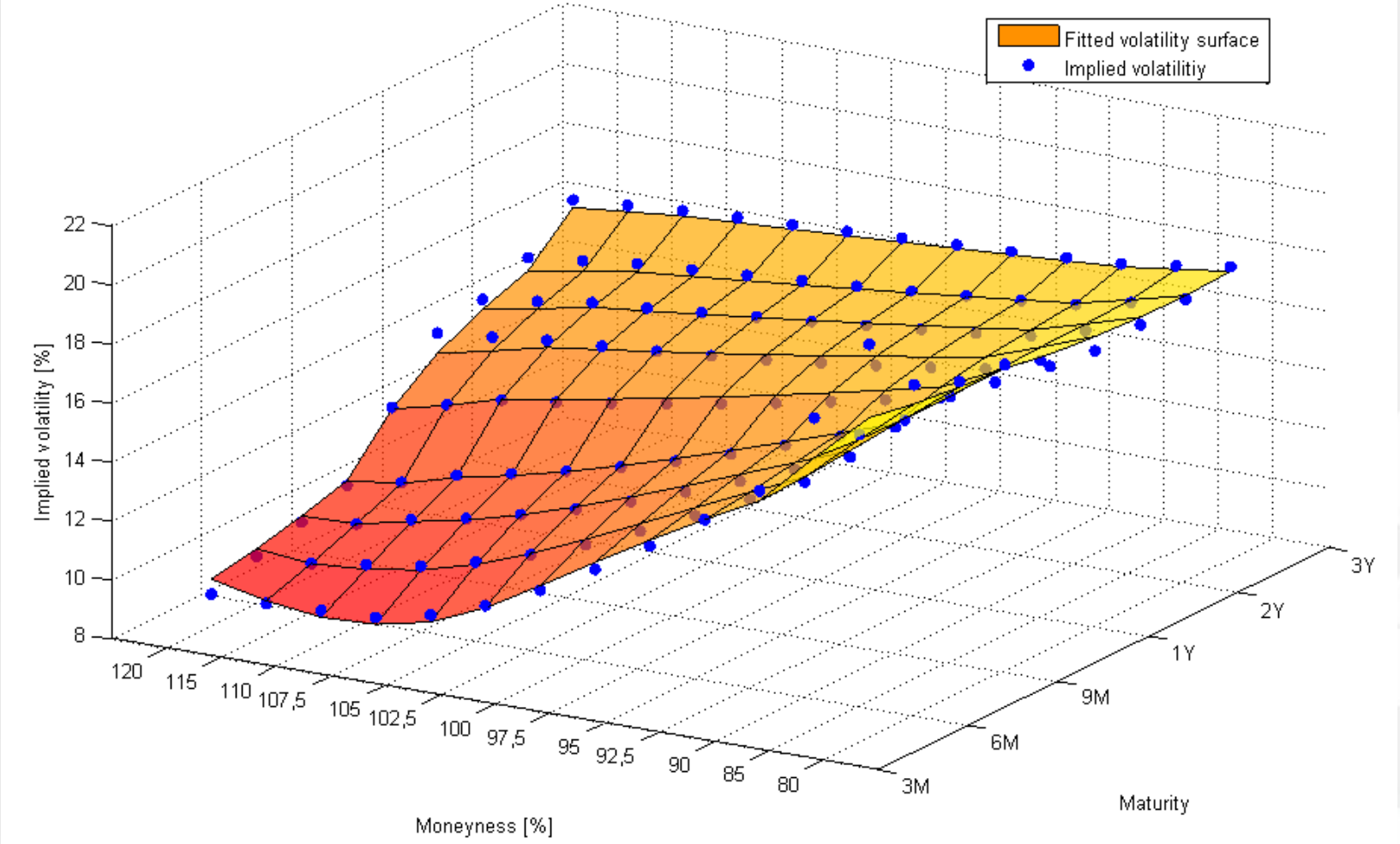
VASICEK CALIBRATION RESULTS - 31/08/18



HESTON MODEL

- Stock price: $dS_t = (\mu - q)S_t dt + \sqrt{v_t} S_t dW_t^1$
- Volatility: $dv_t = \kappa(\theta - v_t)dt + \xi\sqrt{v_t} dW_t^2$
- In order to calibrate we try to fit implied volatilities from calculated option prices to the market volatility surface
- Call option price under Heston model:
  - $C = S_0 e^{-q\tau}\Pi_1 - K e^{-r\tau}\Pi_2$
  - $\Pi_1 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \Re\left(\frac{e^{-iu\log K}\phi(u-i;\tau)}{iu\phi(-i;\tau)}\right)du$
  - $\Pi_2 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \Re\left(\frac{e^{-iu\log K}\phi(u;\tau)}{iu}\right)du$

HESTON MODEL CALIBRATION RESULTS



REFERENCES

- *CUDA C Best Practices Guide v10.0*, 2018: (docs.nvidia.com/pdf/CUDA_C_Best_Practices_Guide.pdf)
- *CUDA C Programming Guide v10.0*, 2018: (docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf)
- Cook, S: *CUDA Programming*, 2013
- Marthaler, D.E: *An overview of Mathematical Methods for Numerical Optimization*, 2013
- Simon, D.: *Evolutionary Optimization Algorithms*, 2013
- Vasicek, O.: *An Equilibrium Characterisation of the Term Structure*, 1977
- Lewis, A.: *Option Valuation Under Stochastic Volatility*, 2000