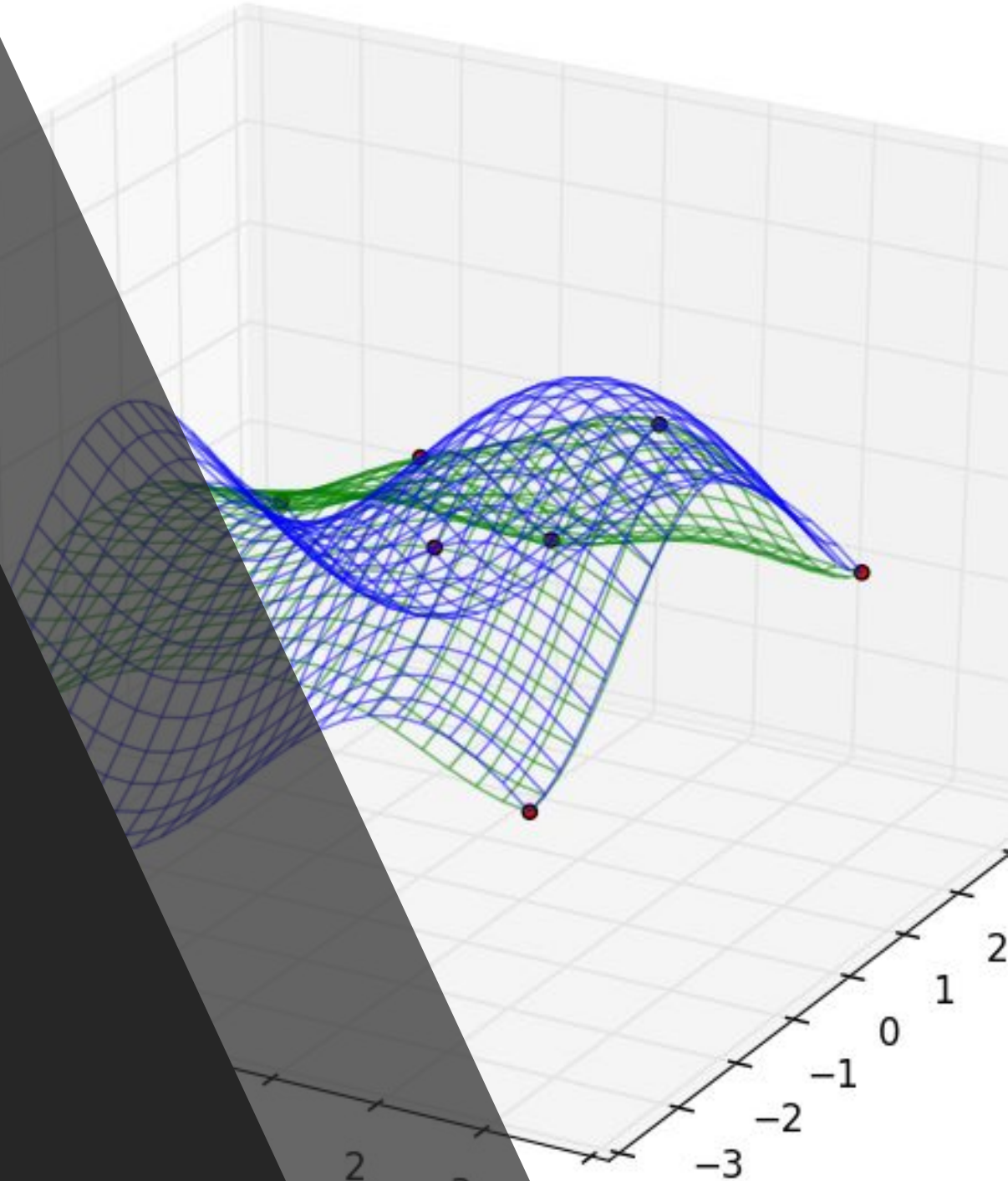
Daan Nilis, Julien Lamour, Linus Handschin

Fast GP-UCB

(**G**aussian **P**rocess **U**pper **C**onfidence **B**ound)

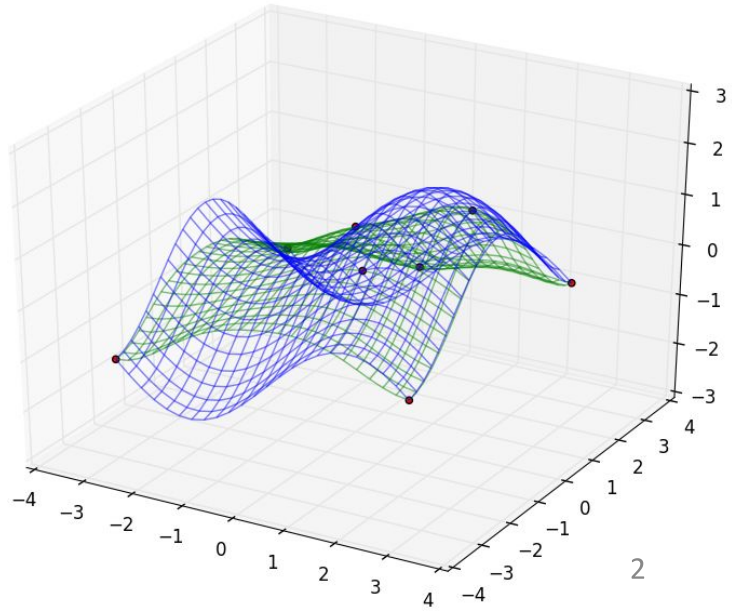
Inspired by:
Srinivas, N., Krause, A., Kakade, S. M., & Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design.



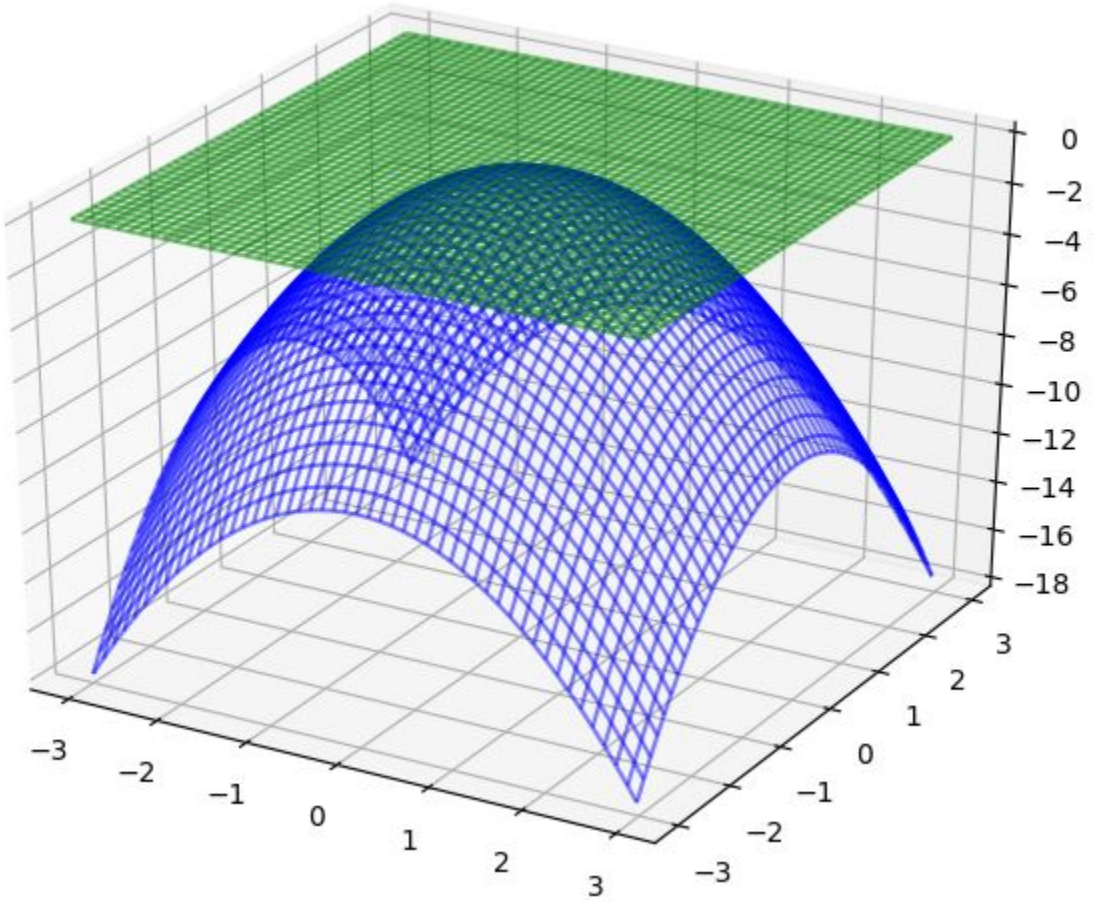
GPUCB - Algorithm

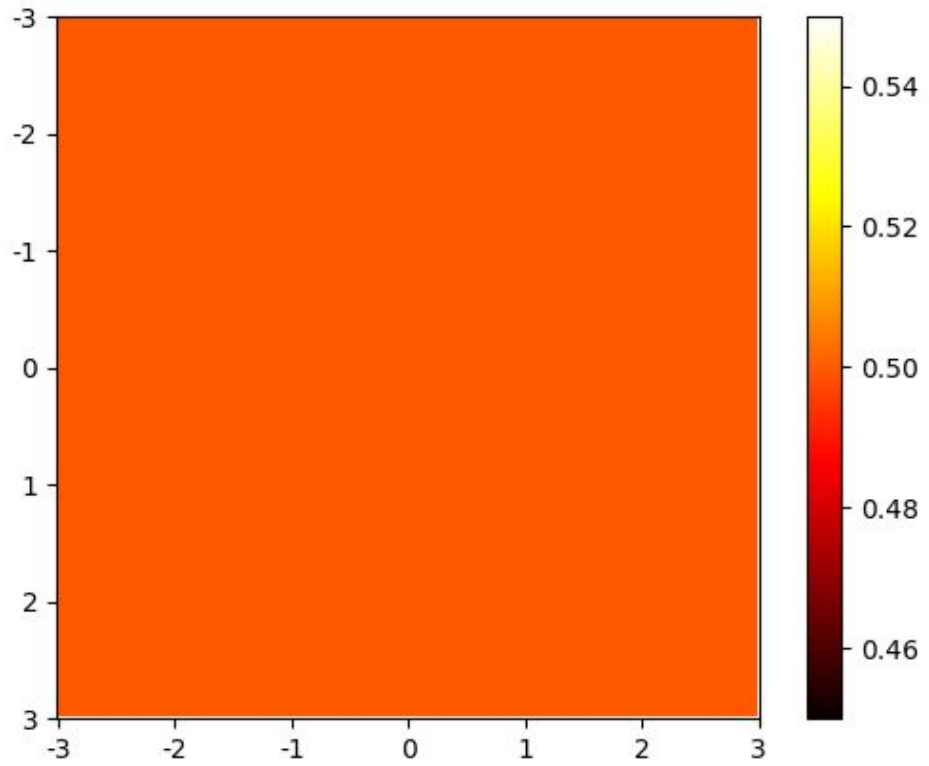
- **GOAL:** Find maximum of an expensive function (f)

  example: highest temperature in building by turning on sensors
- **INPUT**: search grid, sampling function, #iterations (**I**)
- **OUTPUT**: mean and variance for every grid point (N^2)
- **RUNTIME**: $O(N^2 * I^3)$
  - **I**: iterations (how often to sample f)
  - N: search grid size (2D)

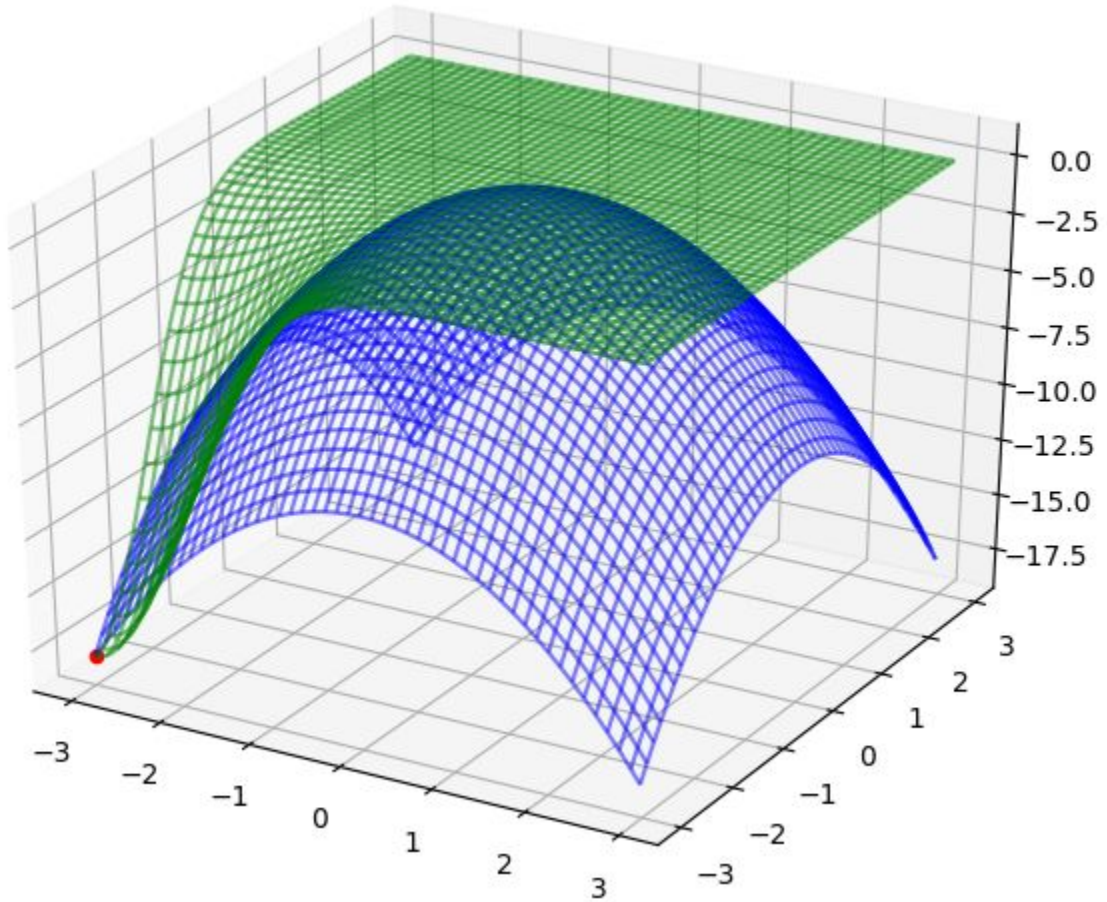


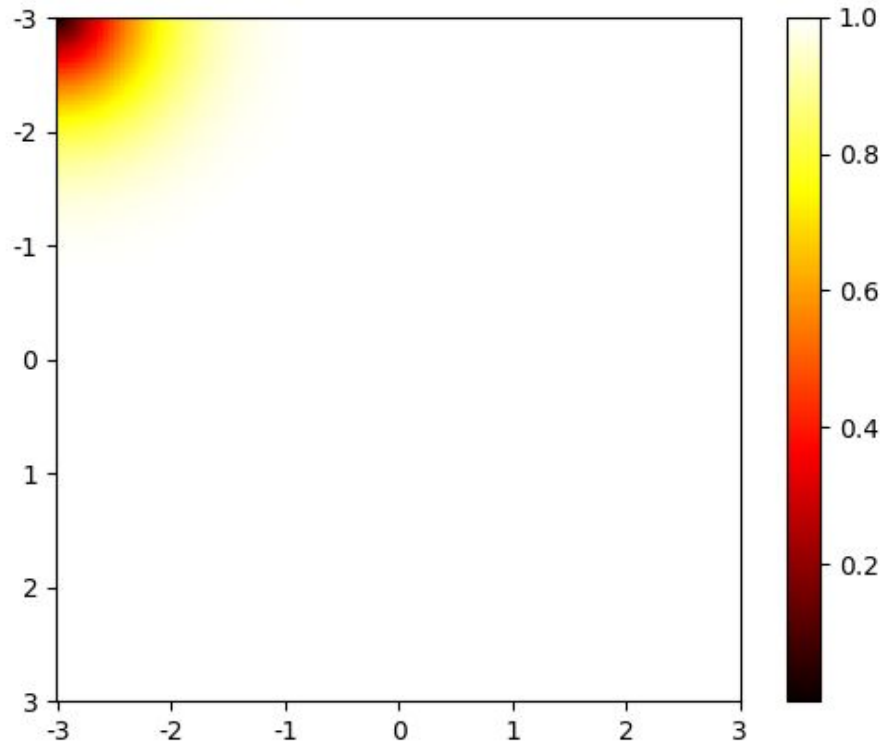
Example: I=0



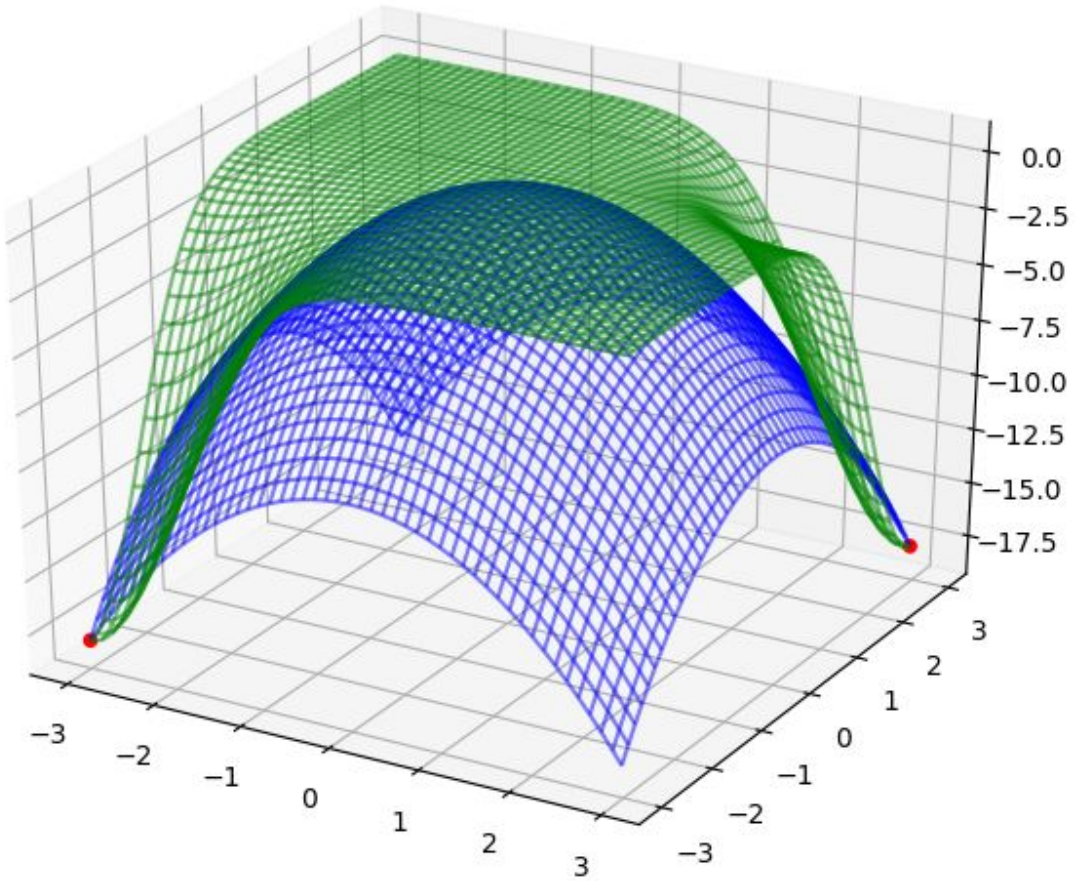


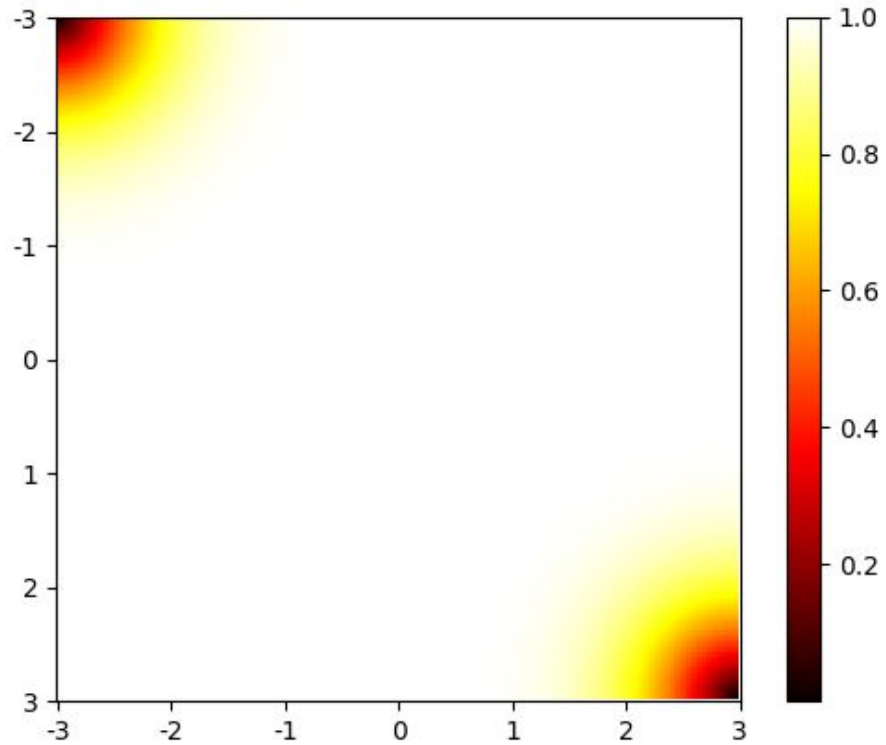
Example: I=1



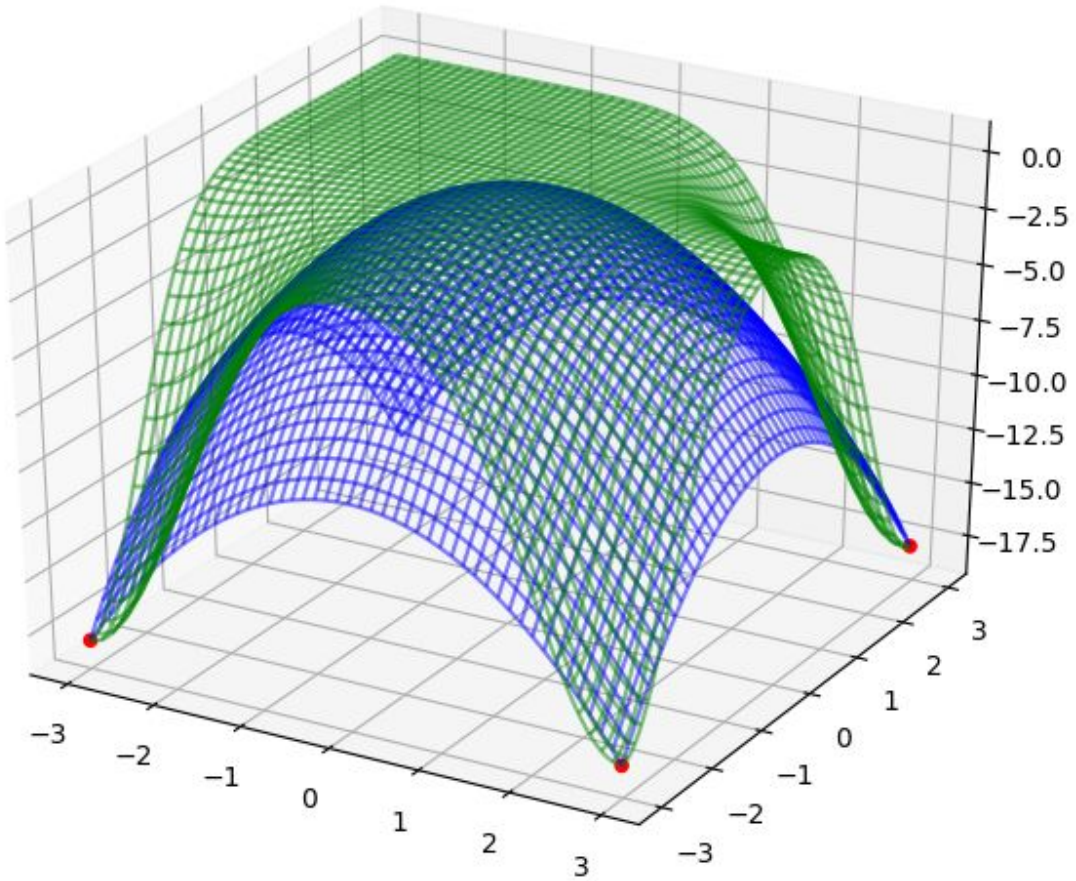


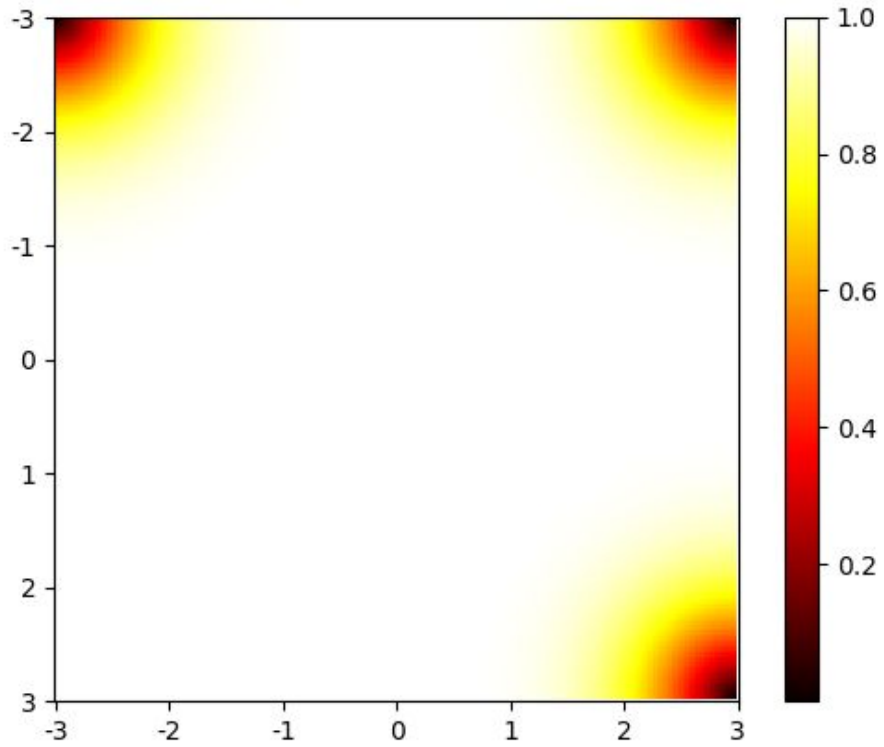
Example: I=2



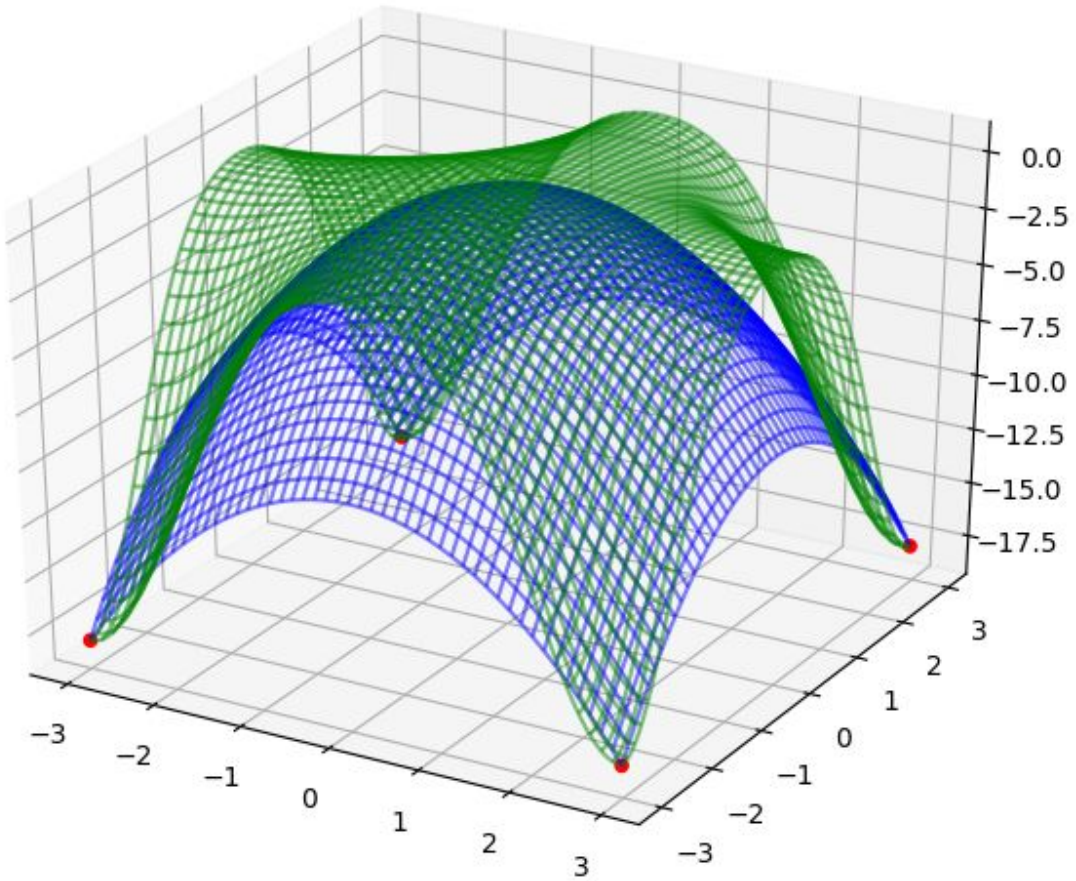


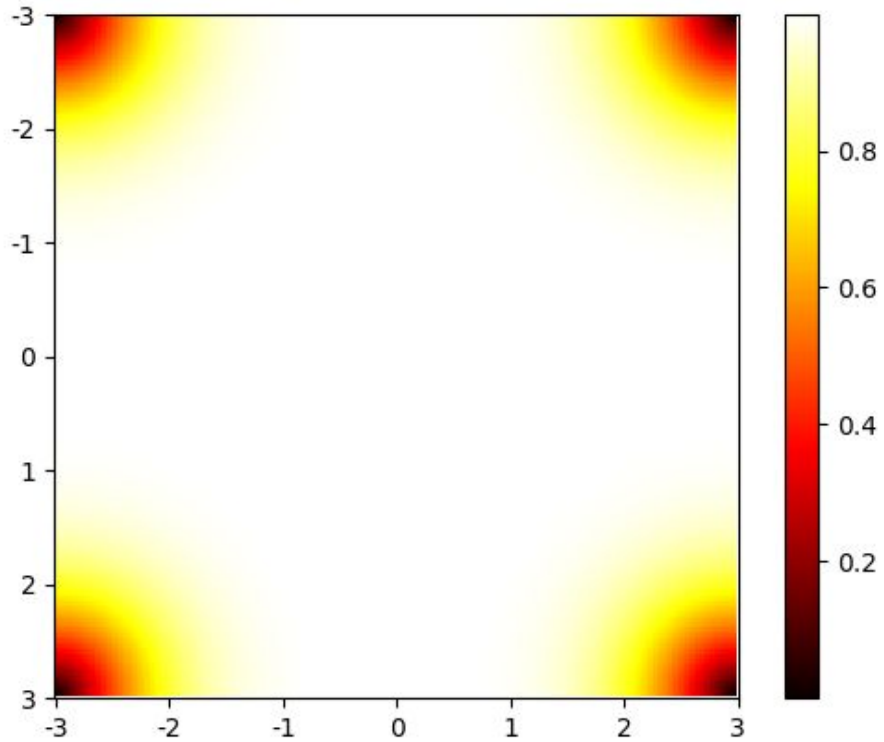
Example: I=3



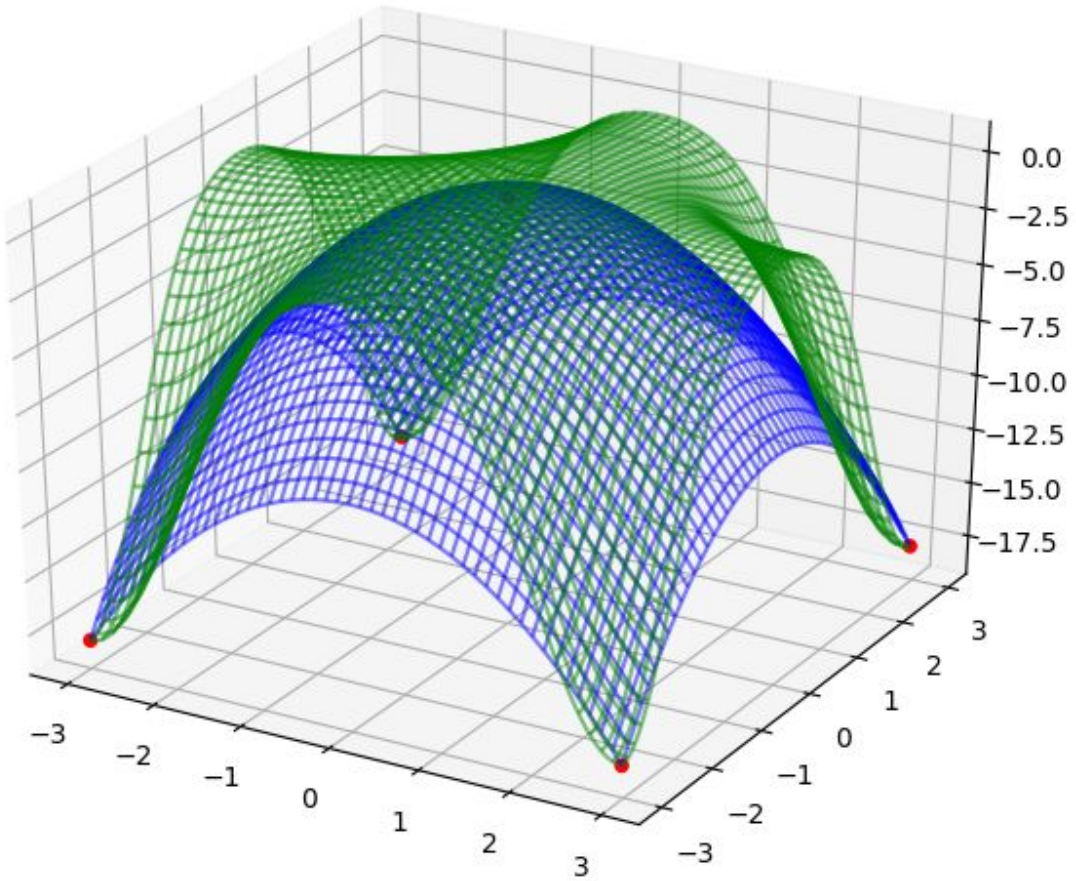


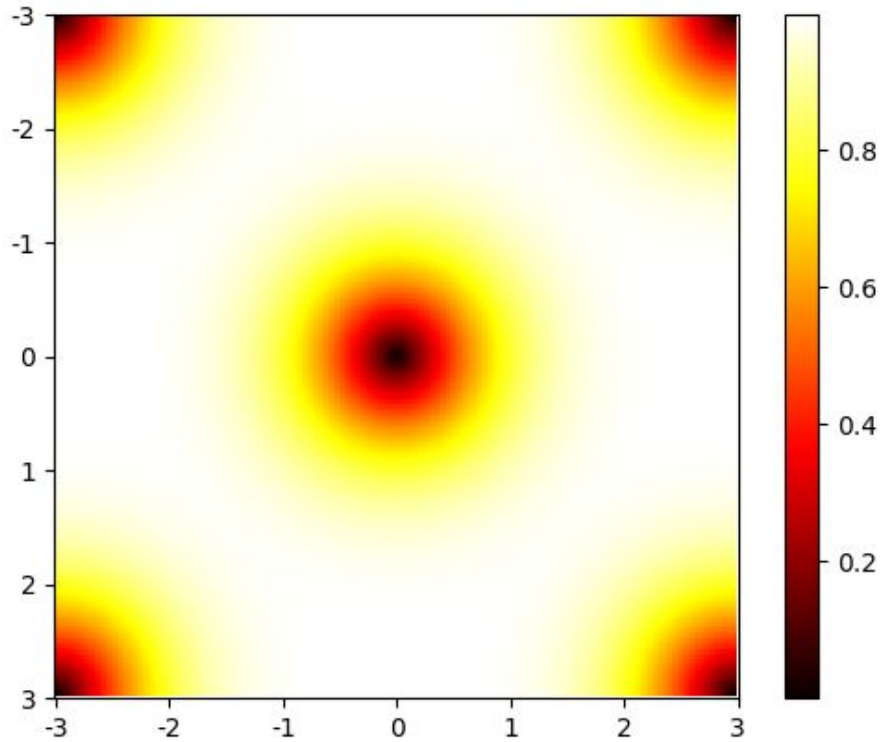
Example: I=4





Example: I=5





Baseline Implementation

- Cost measure: $C(N, \mathbf{I}) = C_{add}(N, \mathbf{I}) + C_{mul}(N, \mathbf{I}) + C_{div}(N, \mathbf{I}) + C_{exp}(N, \mathbf{I})$
- Baseline implementation: linear algebra (argmax + Bayesian update)
- Operational Intensity:

$$W(N, \mathbf{I}) = N^2 * \left(\sum_{i=1}^{\mathbf{I}} (i^2 + i) + \mathbf{I} * k \right) + O(\mathbf{I}^3)$$

$$Q(N) = 4 * N^2$$

$$I(N, \mathbf{I}) = \frac{N^2 * \sum_{i=1}^{\mathbf{I}} (i^2 + i) + O(N^2\mathbf{I} + \mathbf{I}^3)}{4 * N^2}$$

$$= \frac{4I^3}{3} + O\left(\frac{\mathbf{I}^3}{N^2} + \mathbf{I} \right) >> 1$$

Optimization Overview

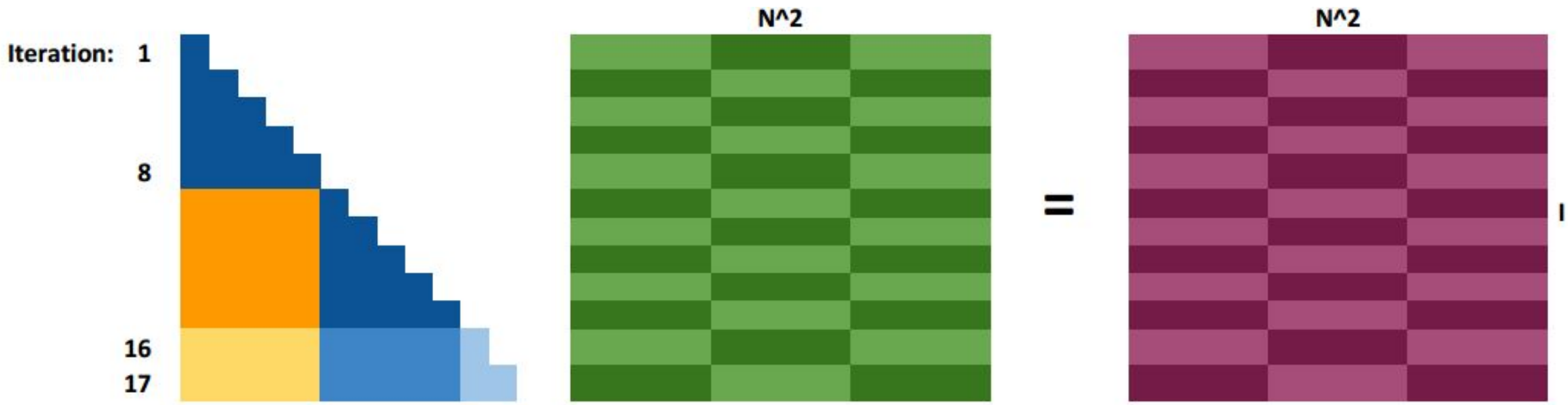
2: $L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$

4: $f_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$ (mean)

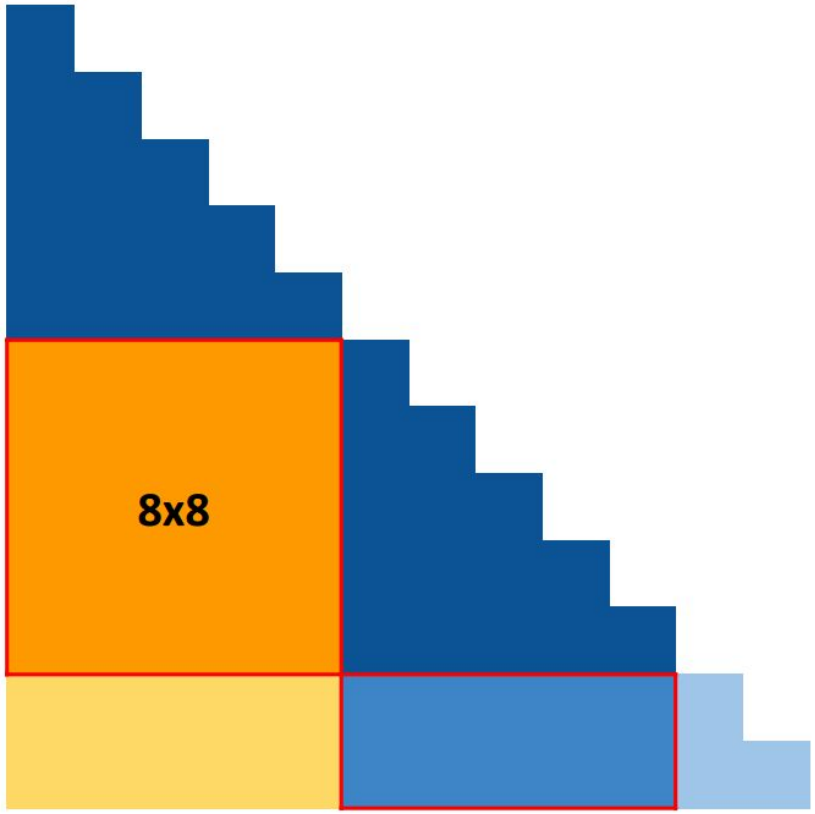
$\mathbf{v} := L \backslash \mathbf{k}_*$

6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$ (variance)



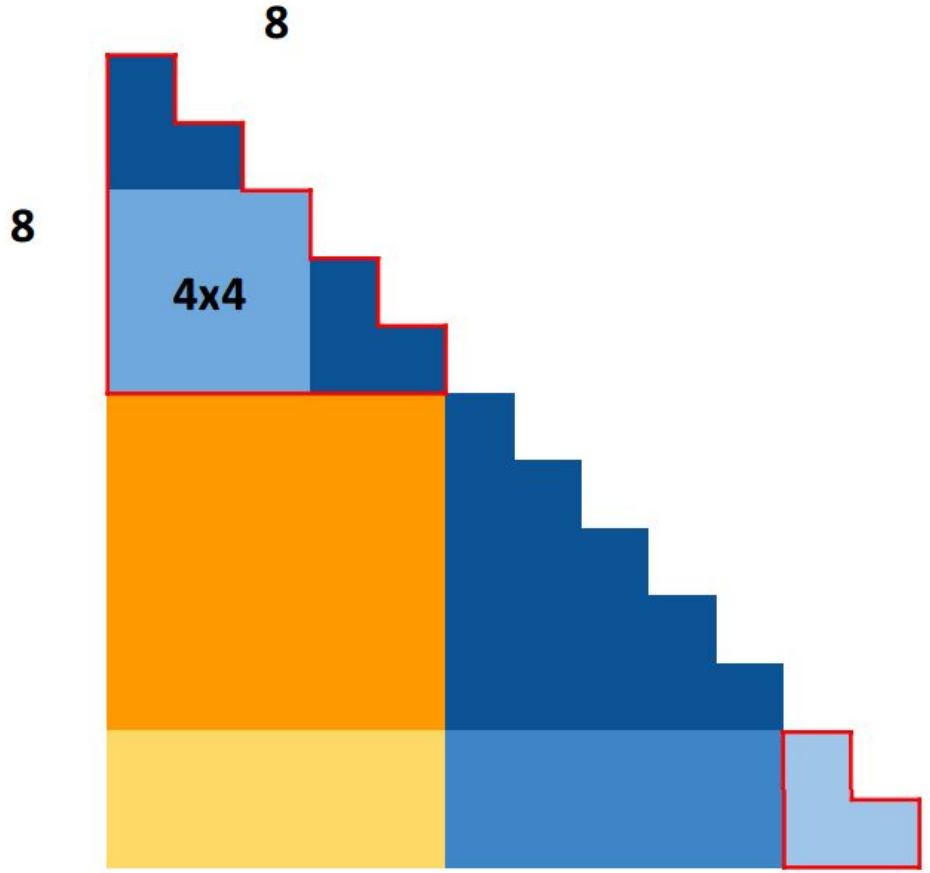
MMM

- Matrix Matrix Multiplication 8*8
- Matrix Matrix Multiplication k*8, k<8
- Achieved: 16 f/c

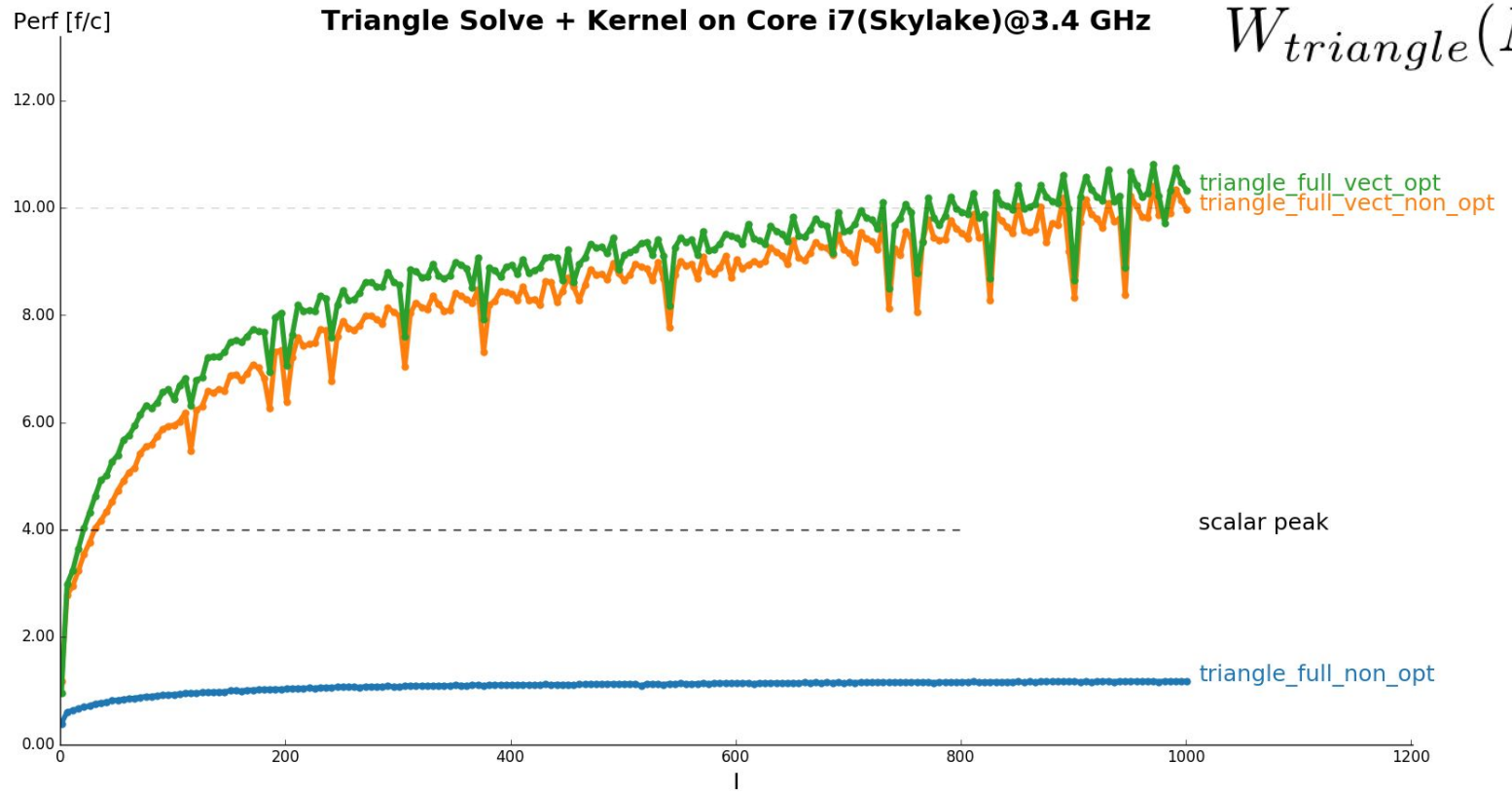


Triangle Solve with kernel computation

- Triangle Solve 8x8, split in:
  - Triangle Solve 4x4
  - MMM 4x4
- Triangle Solve Rest
- Covariance matrix built on the fly



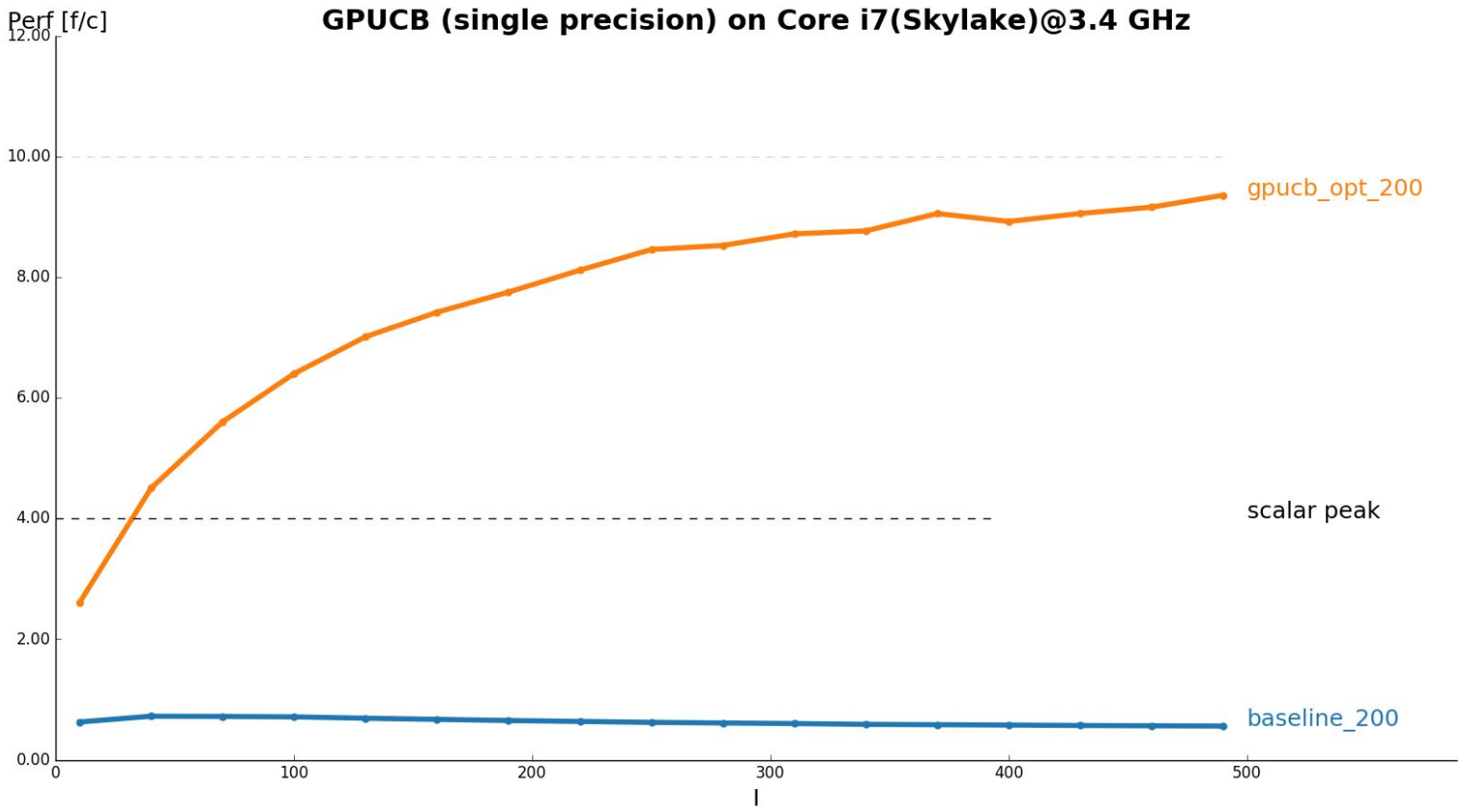
Triangle Solve (with kernel computation on the fly)



$$W_{triangle}(N, \mathbf{I}) = N^2 * \mathbf{I}^2 + N^2 * \mathbf{I} * k$$

gcc 4.8.5 | flags: -O3 -march-avx2 -fno-tree-vectorize | Red Hat 7.3 | search grid 200x200

Fast GPUCB vs Baseline



Fast GPUCB: 30% AVX2 peak

baseline: 12% scalar peak

gcc 4.8.5 | flags: -O3 -march-avx2 -fno-tree-vectorize | Red Hat 7.3 | search grid 200x200