Parallel Machine Learning Algorithms in Bioinformatics and Global Optimization

Scott Clark[1]
Peter Frazier[2]
Zhong Wang[3], Rob Egan[3]
Nick Hengartner[4], Joel Berendzen[4]

[1]Cornell University Center for Applied Mathematics, Ithaca, NY
[2]Cornell University Department of ORIE, Ithaca, NY
[3]Joint Genome Institute, Walnut Creek, CA
[4]Los Alamos National Laboratory, Los Alamos, NM

May 4, 2012

Outline

- Motivation
  - What is next-generation sequencing?

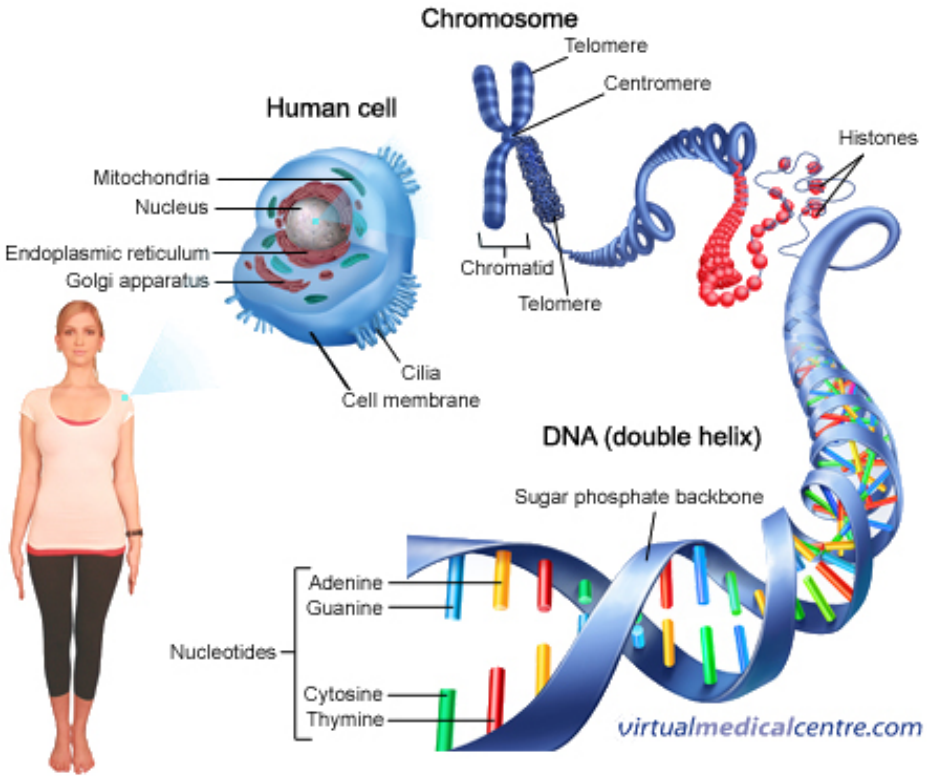
Outline

- Motivation
  - What is next-generation sequencing?
- Validating Assemblies
  - What is an assembly?
  - Current methods
  - Using Bayes to build likelihoods
  - Dependence on assembly quality

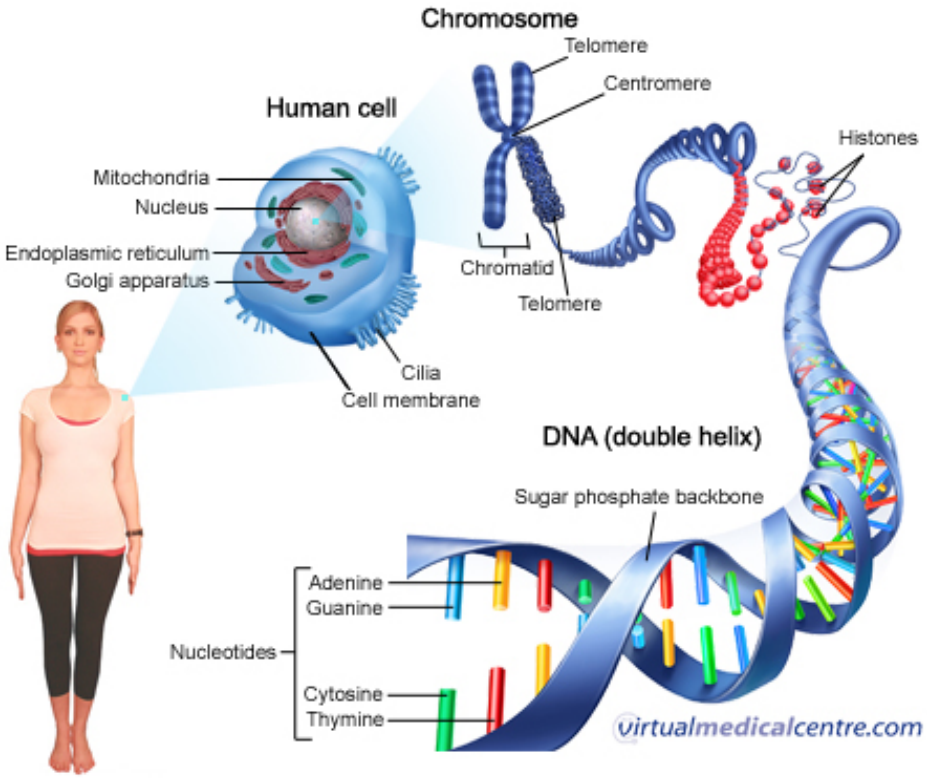
Outline

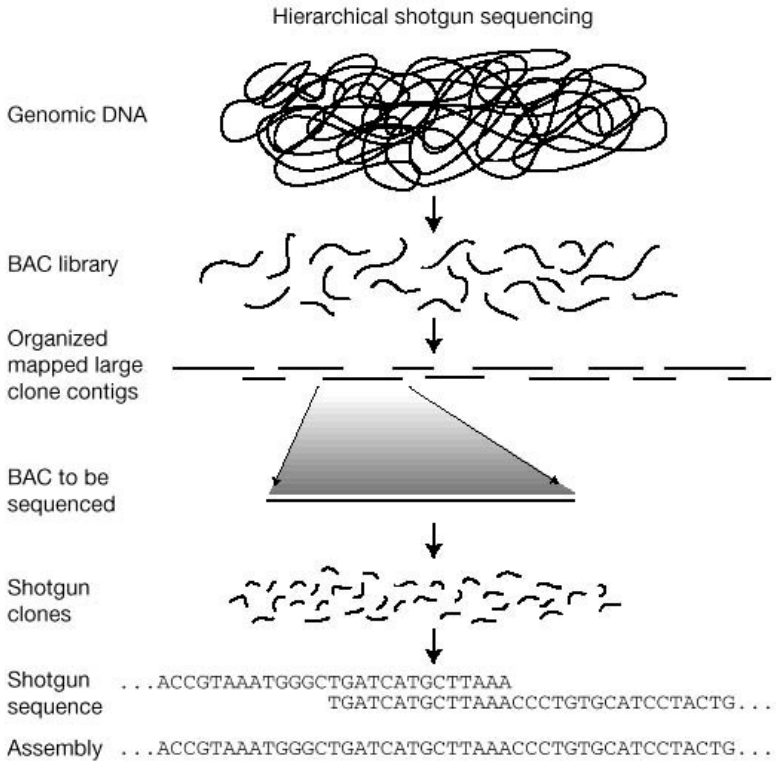
- Motivation
  - What is next-generation sequencing?
- Validating Assemblies
  - What is an assembly?
  - Current methods
  - Using Bayes to build likelihoods
  - Dependence on assembly quality
- Expected Parallel Improvement
  - Parallel global optimization of expensive functions

What is next-generation sequencing?

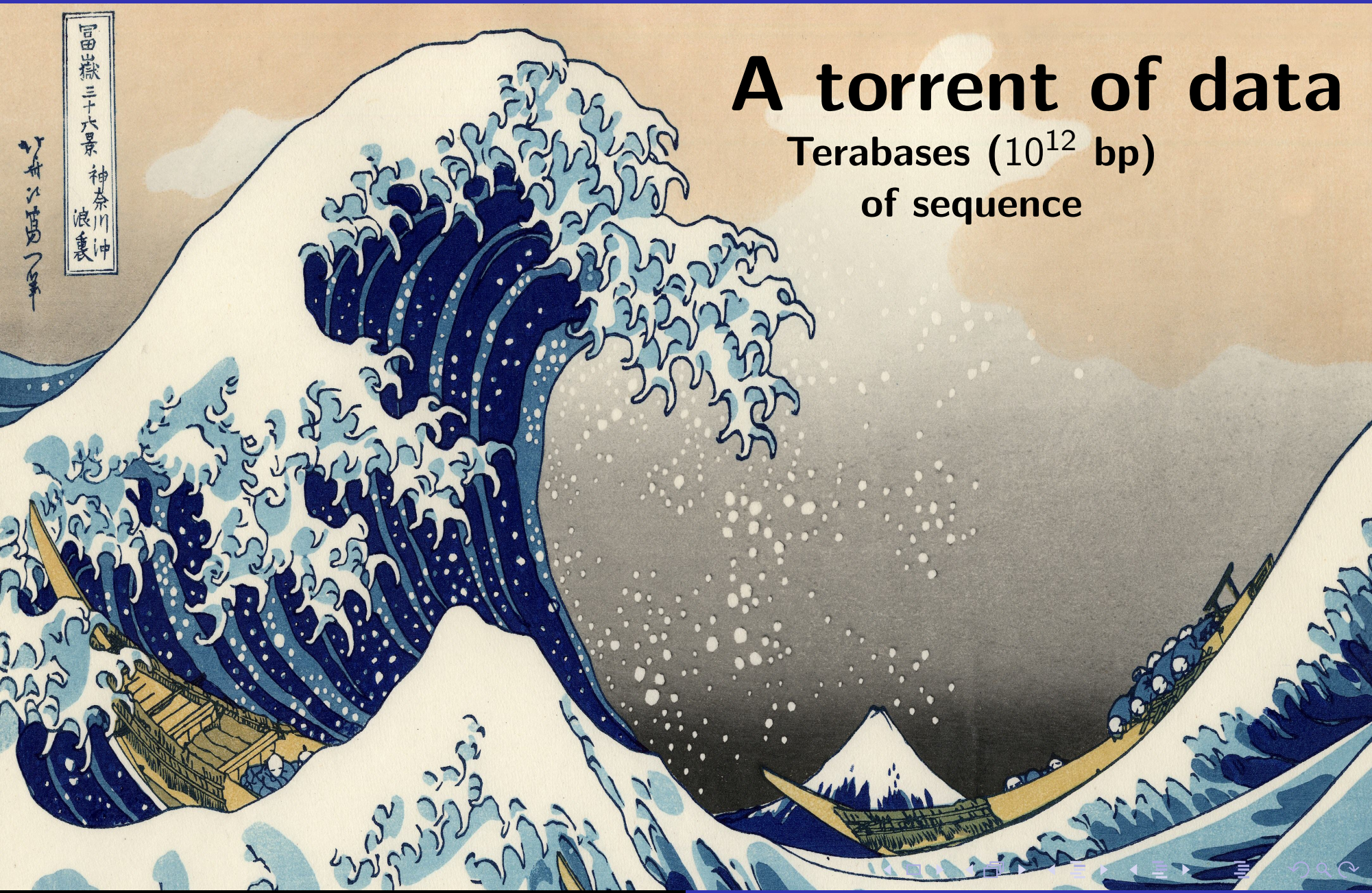


What is next-generation sequencing?

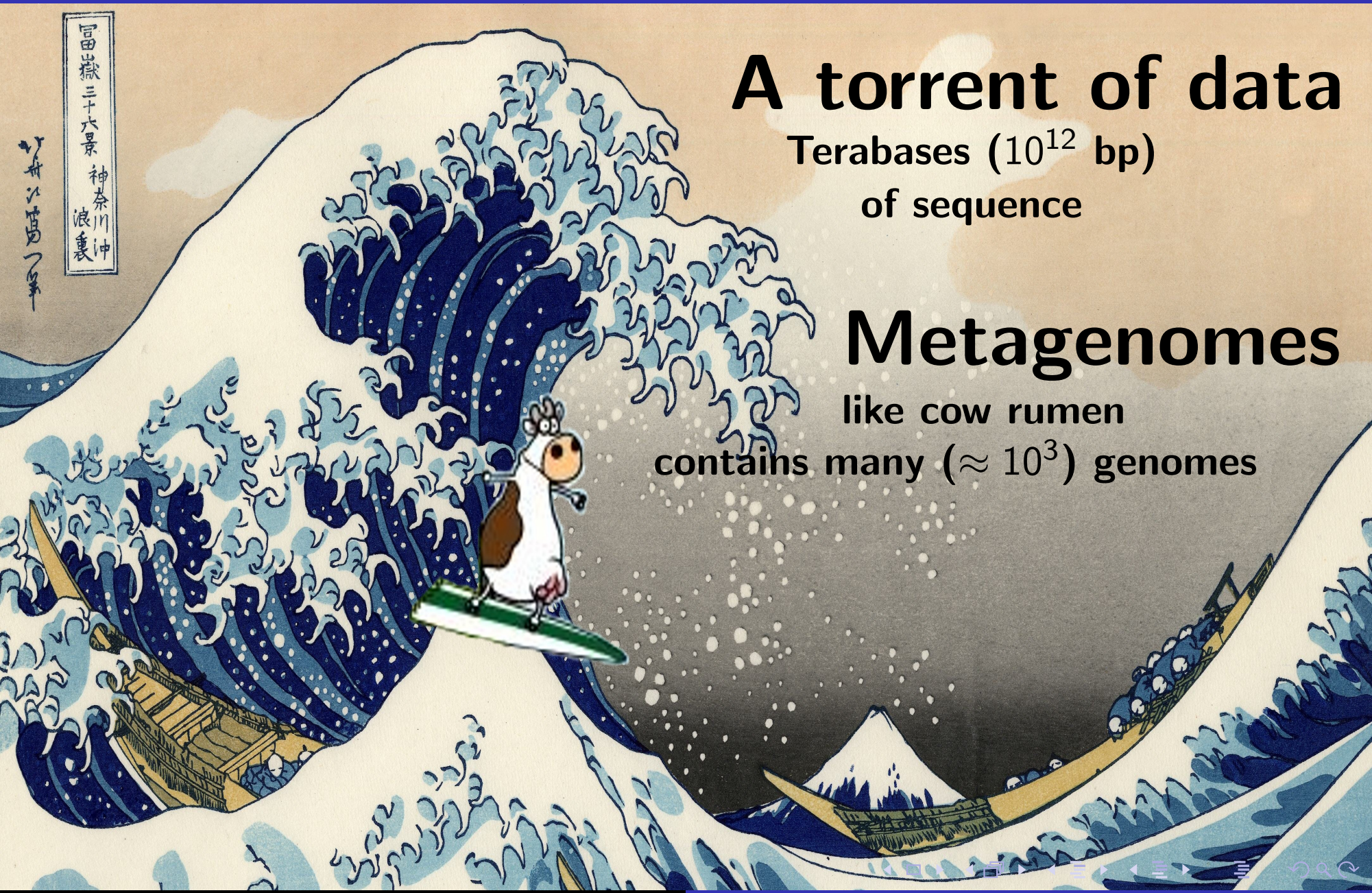




What is the problem?



What is the problem?



A torrent of data

Terabases (10^{12} bp)

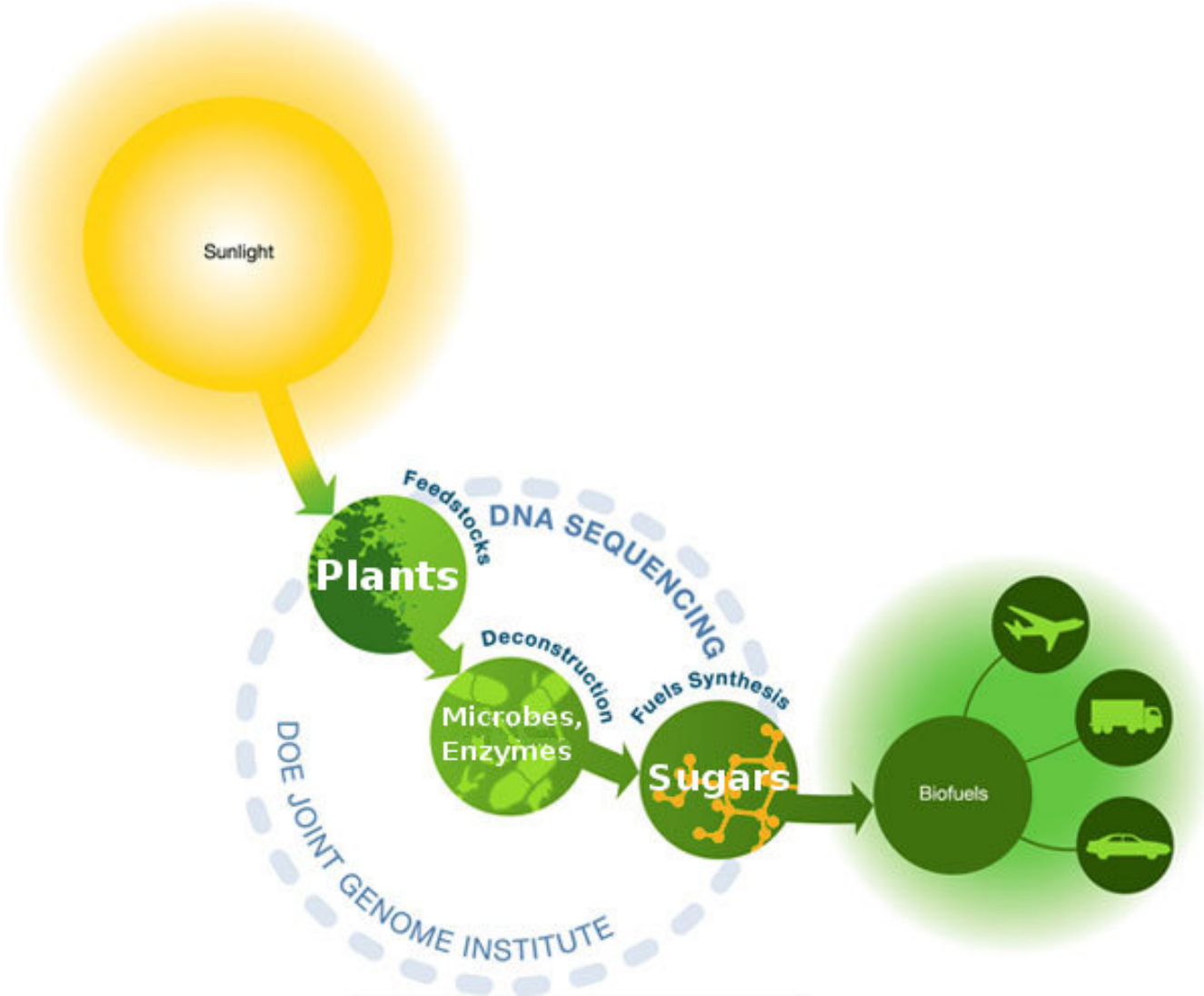
of sequence

Metagenomes

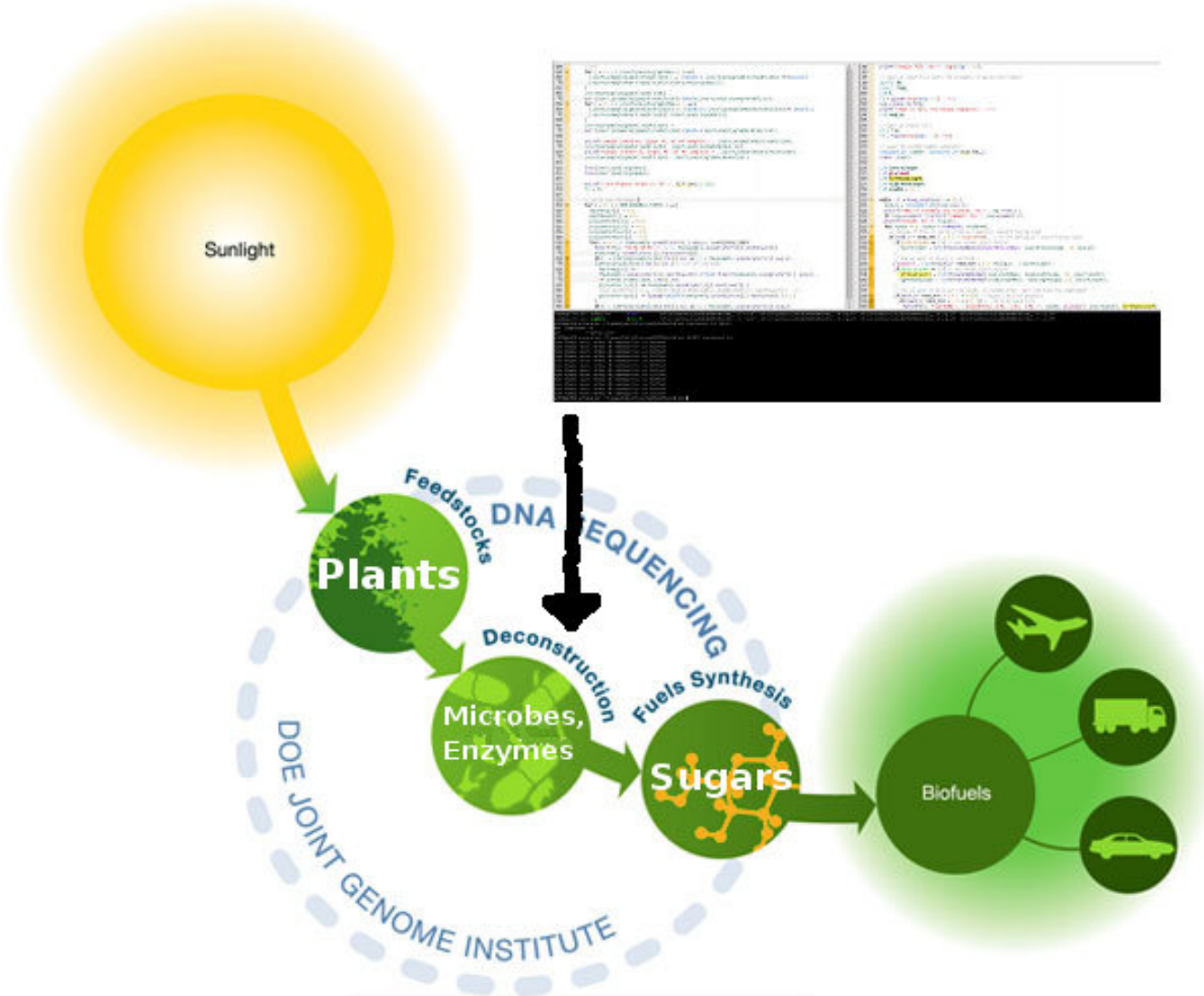
like cow rumen

contains many ($\approx 10^3$) genomes

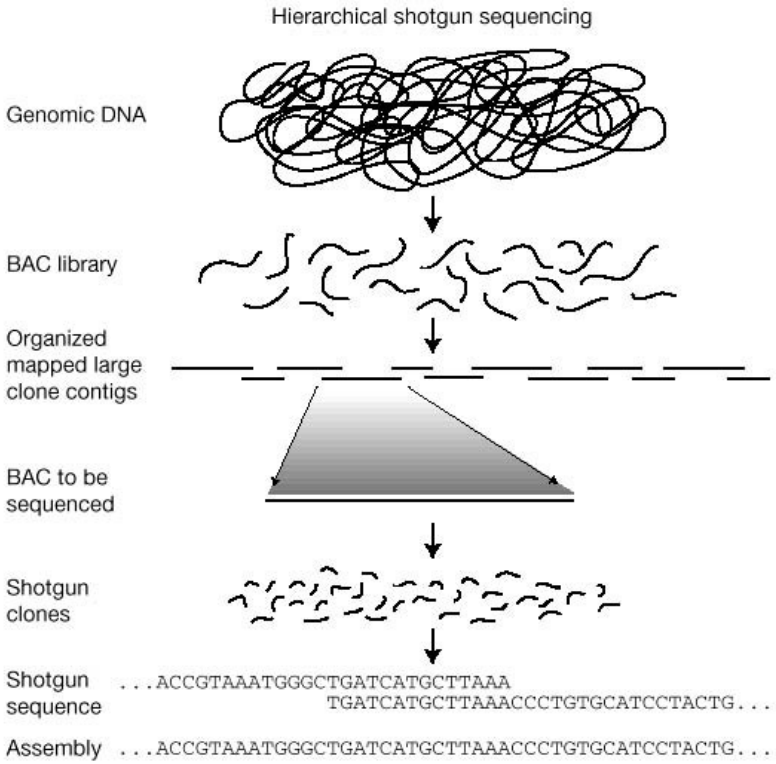
Why do we (or the DOE) care?

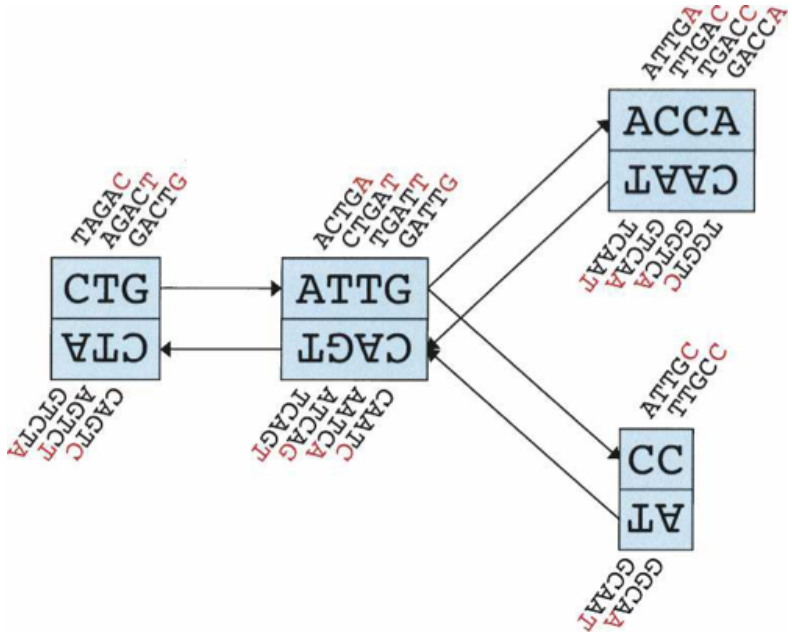


Why do we (or the DOE) care?

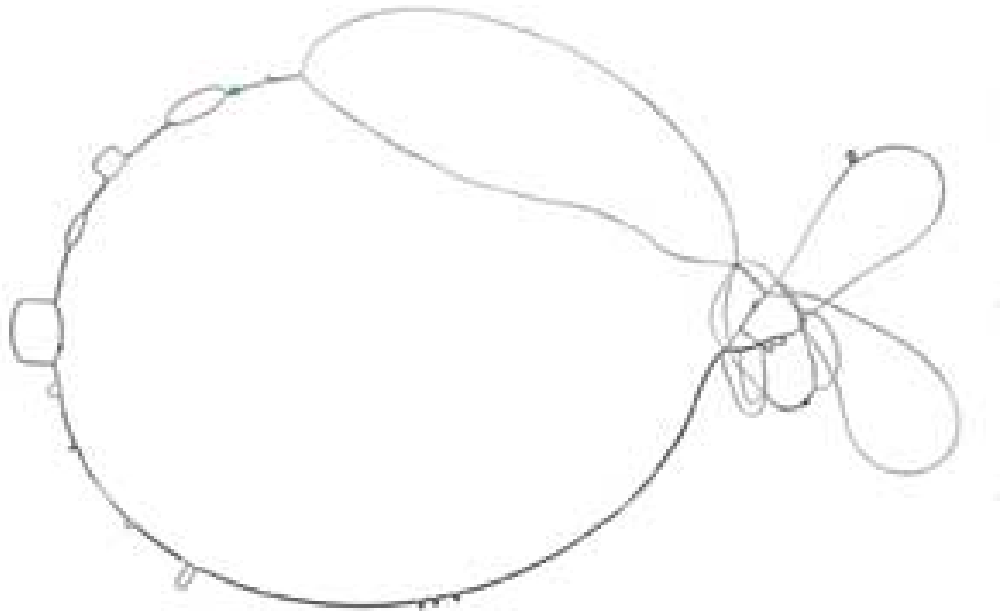


What is an assembly?





What is an assembly?



⇓

Current methods

Once we have an assembly from some method we want to know how good it is.

- Current Methods
  1. Size of pieces (contigs), ie N50 size
  2. Map it onto a known reference
  3. Pathway analysis

Current methods

Once we have an assembly from some method we want to know how good it is.

- Current Methods
  1. Size of pieces (contigs), ie N50 size
  2. Map it onto a known reference
  3. Pathway analysis
- Problems
  1. Accuracy irrelevant
  2. Sometimes there is no known reference
  3. Only works on specific pathway (with known reference)

Using Bayes to build likelihoods

We want to be able to take an assembly and the reads that generated it and find a score, or how likely that assembly is,

$$P(\text{Assembly} \,|\, \text{All the Reads})$$

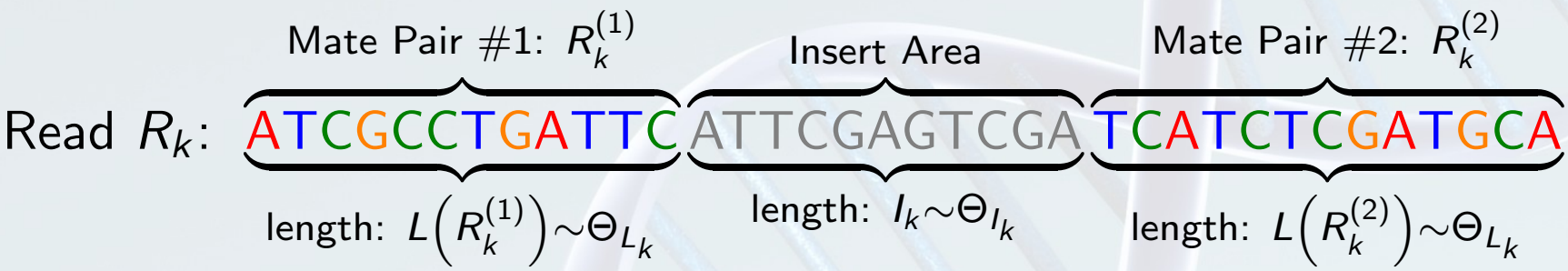
Which by Bayes rule is proportional to

$$P(\text{All the Reads} \,|\, \text{Assembly})$$

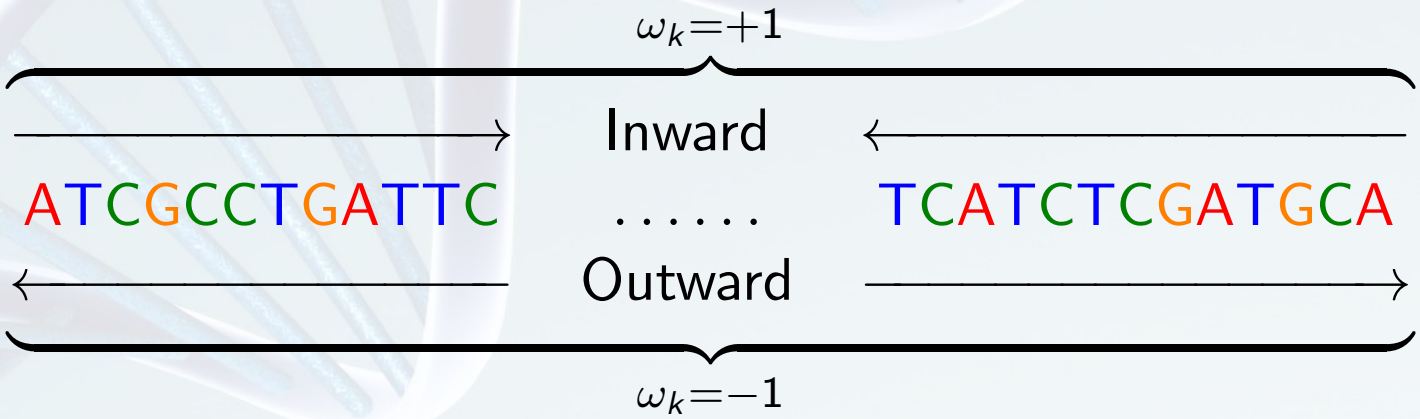
This will be our likelihood function.

Read properties

Mated Read R_k Properties



Orientation ω_k of mated read R_k



Components of the likelihood function

The Likelihood needs to take at least the following into account:

- Read **sequences** need to agree with assembly.
- **Insert lengths** must be consistent.
- **Orientation** of mated reads needs to be consistent.
- **Number of reads mapping** onto the assembly.
- **Coverage** Poisson distributed (Lander 1988).
- *k*-**mer frequency** must be consistent. (within a contig)

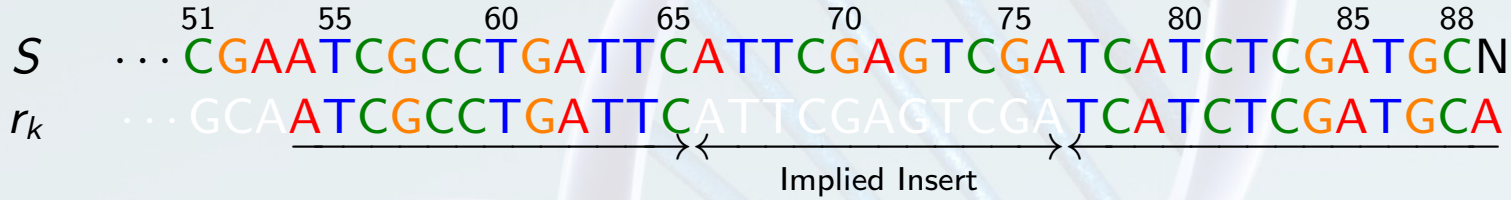
ALE placement score

$$P_{\text{matches}}\left(r_i|S\right)(j) = \begin{cases} Q_j & \text{if } r_k(j) = S(j+\Omega) \\ \frac{1-Q_j}{4} & \text{if } r_k(j) \neq S(j+\Omega) \text{ and } r_k(j) \notin A \\ A_j & \text{if } r_k(j) \in A \end{cases}$$

$$P_{\text{placement}}\left(r_i|S\right) = P_{\text{matches}}\left(r_i|S\right) P_{\text{orientation}}\left(r_i|S\right)$$

ALE insert score

Total ALE score needs to take insert size into account



$$P_{\text{insert}}\left(r_i|S\right) = \text{TruncatedNormal}\left(L_i; \mu, \sigma^2\right)$$

ALE depth score

We expect the coverage of the reads when mapped back onto the seeds to be Poisson distributed (Lander 1988)

$$P_{\text{depth}}(d_j|S, X_i) = \text{Poisson}(d_j; Y_i)$$

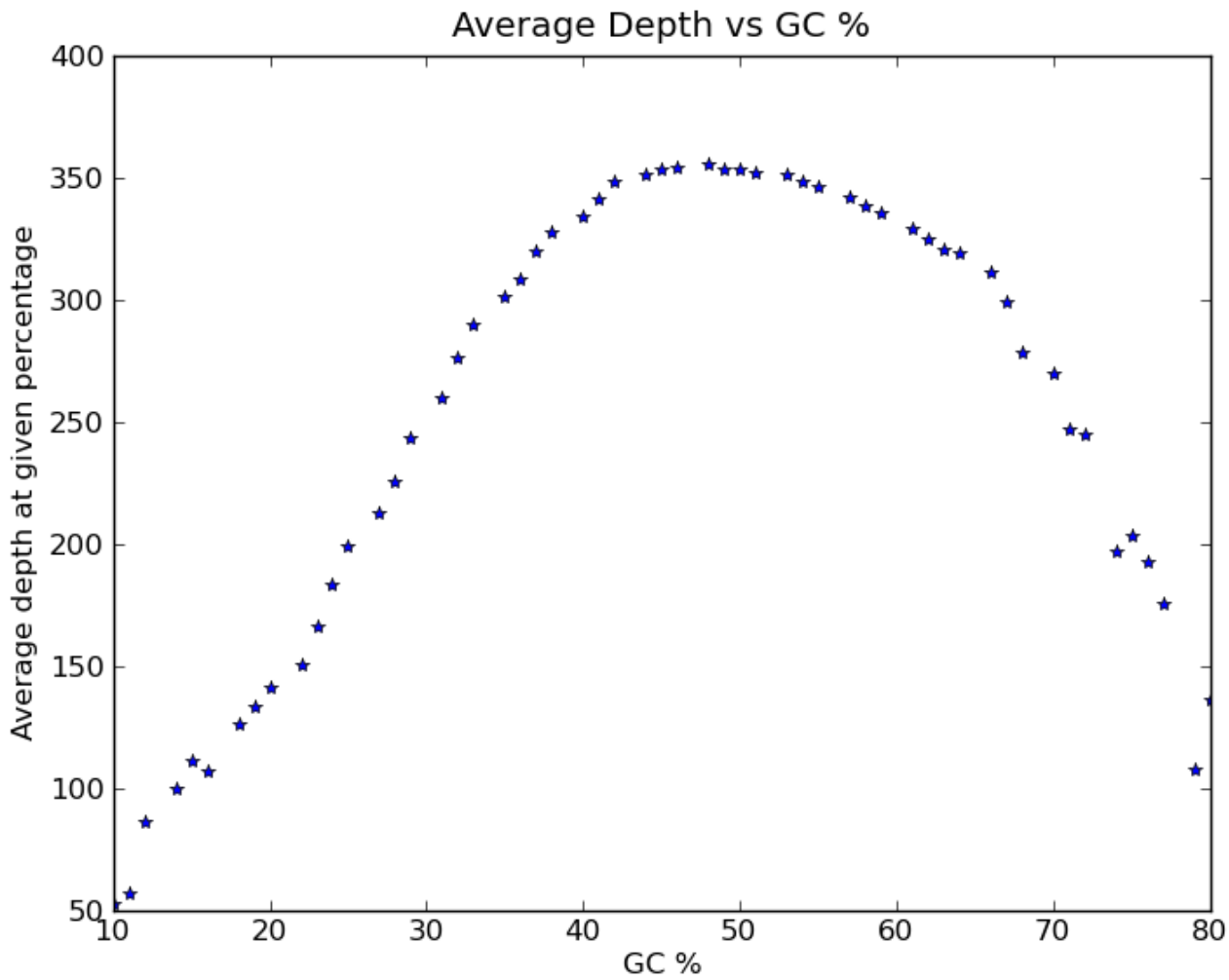
$$Y_i \sim \text{Exponential}(1/\max(10, \mu_{\text{depth}(X_i)}))$$

$$P_{\text{depth}}(d_j|S, X_i) = \text{NegBinom}(d_j; \max(10, \mu_{\text{depth}(X_i)}), 1/2)$$

GC correction

- The sequencing technology introduces a "GC bias"
- Those bases are overrepresented in the data
- We compute 100 different Poisson distributions based on the % of GC in each read

ALE depth score: GC bias



ALE kmer score

What are k-mers?

- Contiguous subsequences of length k
- The frequency a k-mer appears (for $k \geq 4$) is relatively distinct between species
- We can use this information to distinguish individual species from a metagenome

$$f_i = \frac{n_i}{\sum_{j \in K} n_j}$$

$$P_{\text{kmer}}(S) = \prod_{i \in K} f_i^{n_i}$$

Normalization and total scores

The total ALE score

$$P(S|R) = \frac{P(R|S)P(S)}{Z}$$

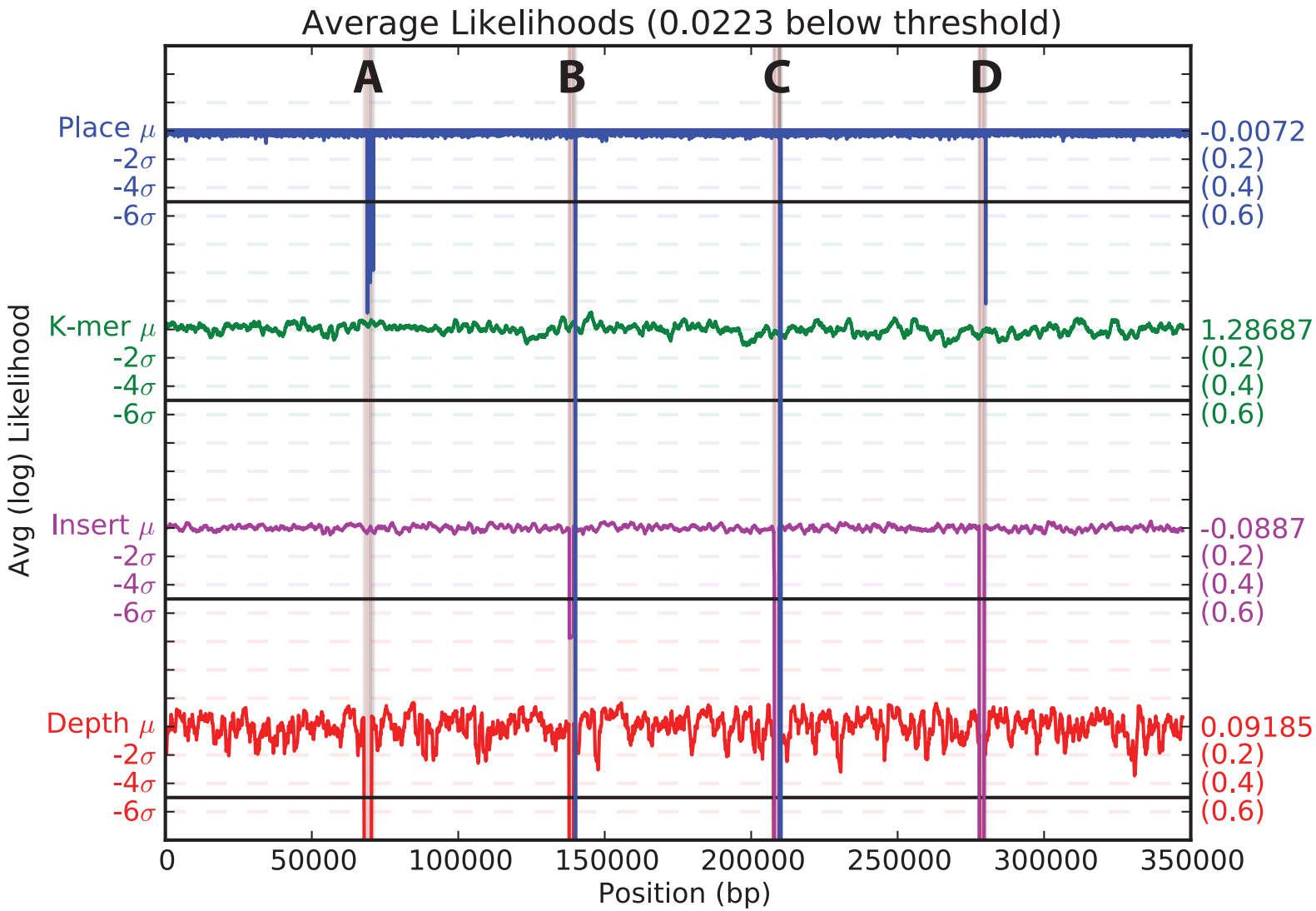
$$P(R|S) = P_{\text{placement}}(R|S)P_{\text{insert}}(R|S)P_{\text{depth}}(R|S)$$

$$P(S) = P_{\text{kmer}}(S)$$

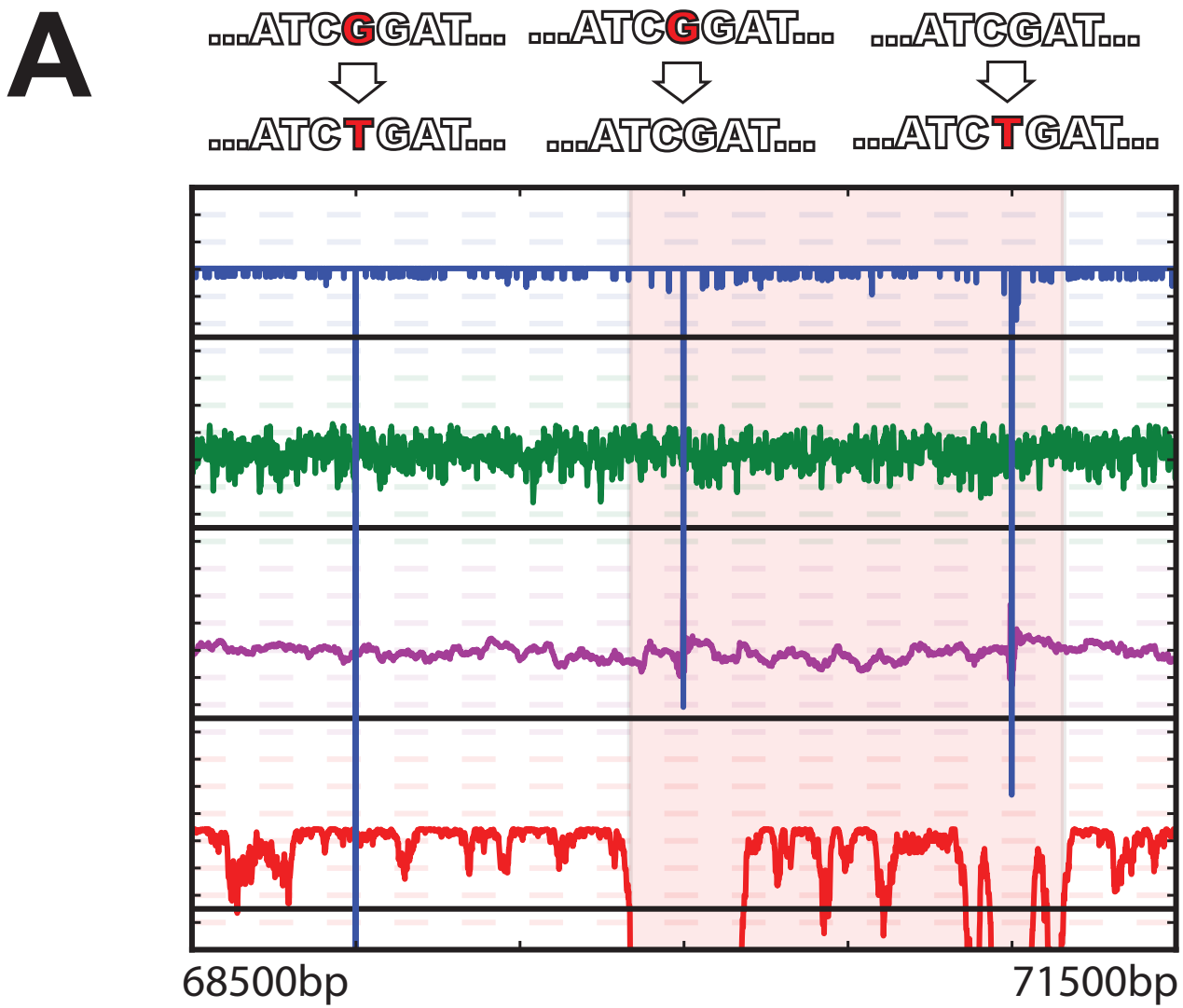
Comparing two assemblies

$$A_1 - A_2 = \log\left(\frac{P\left(R|S_1\right)P\left(S_1\right)}{P\left(R|S_2\right)P\left(S_2\right)}\right)$$

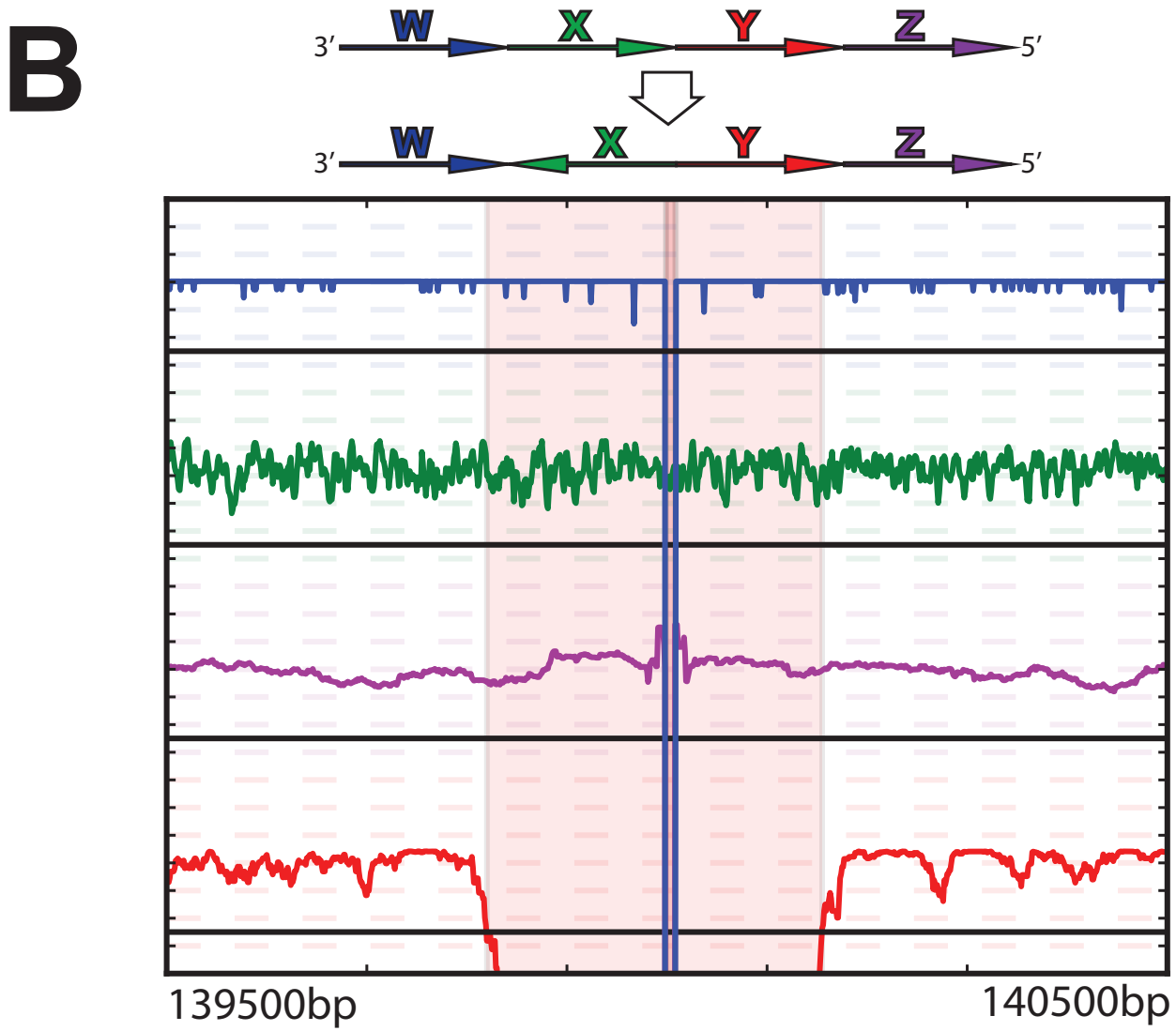
Likelihood with permutation from a known assembly



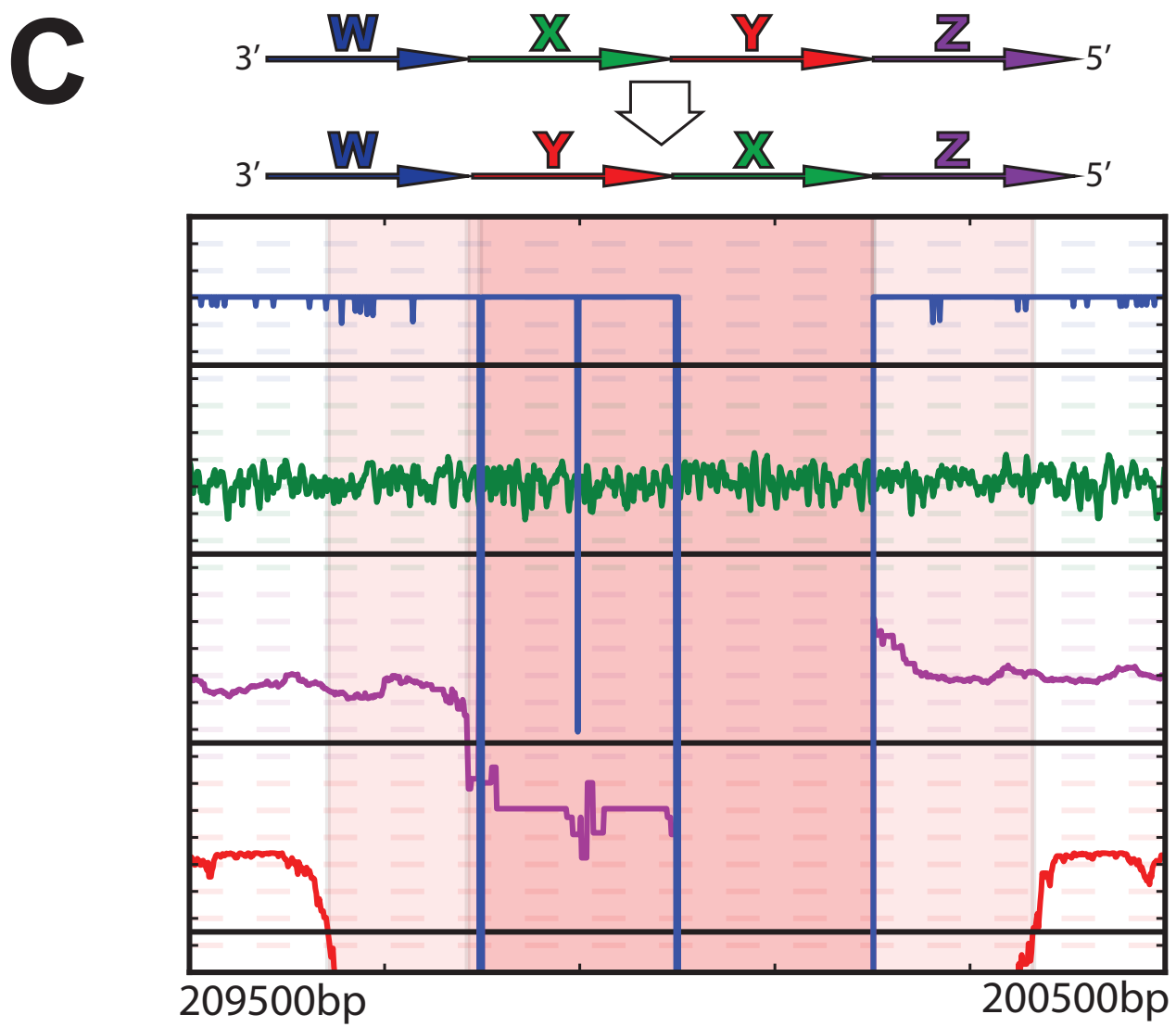
Likelihood with permutation from a known assembly



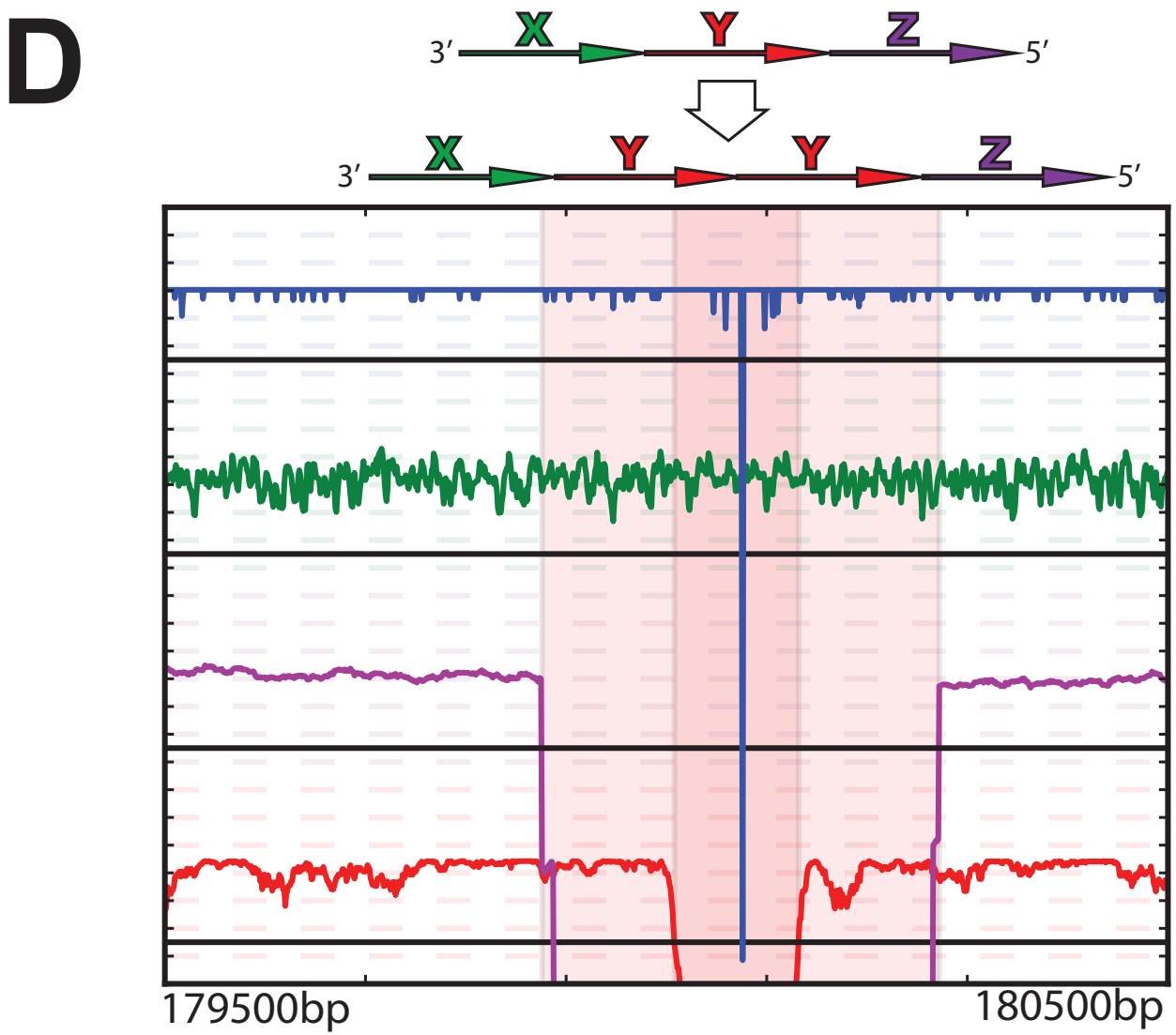
Likelihood with permutation from a known assembly



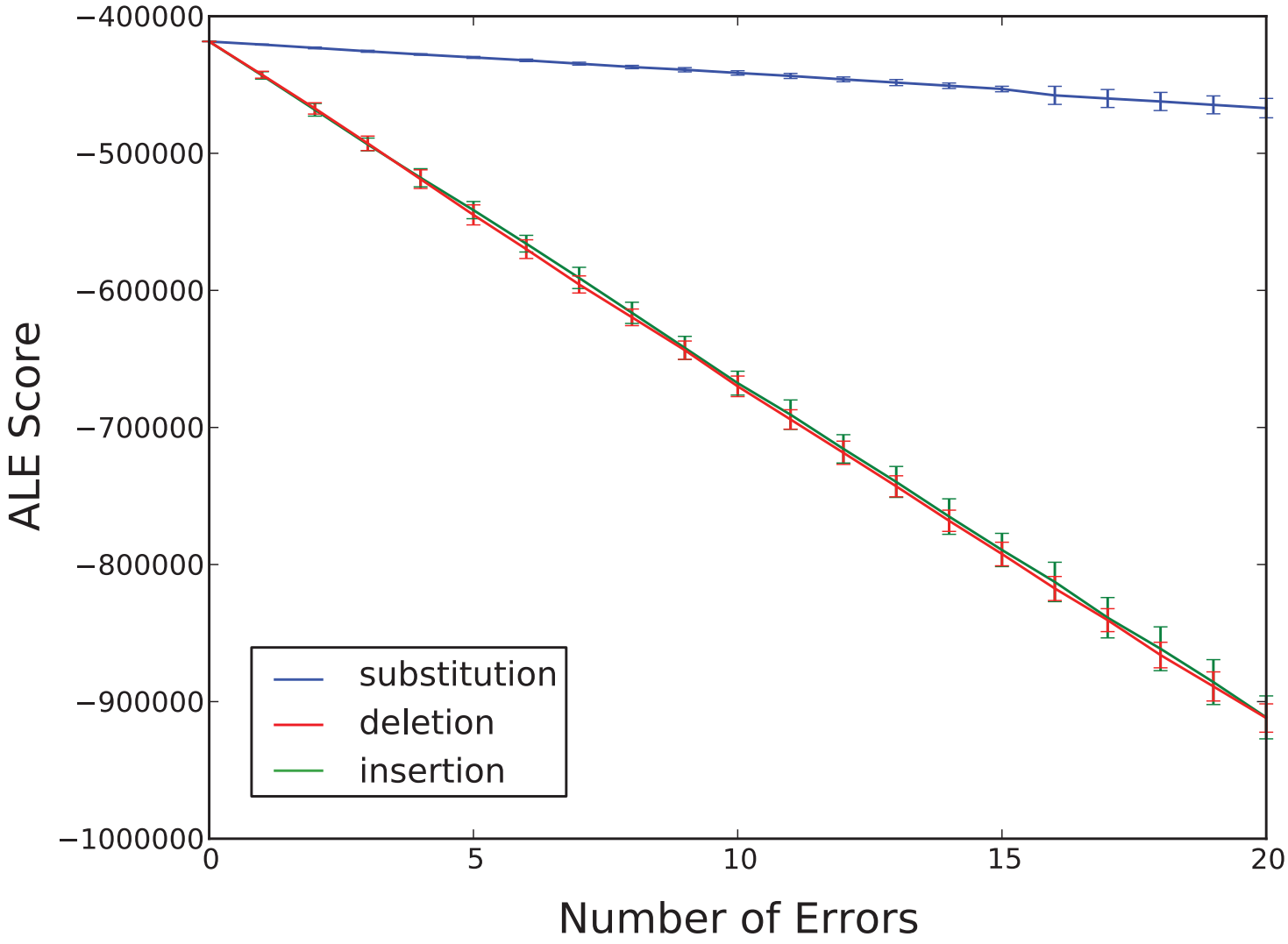
Likelihood with permutation from a known assembly



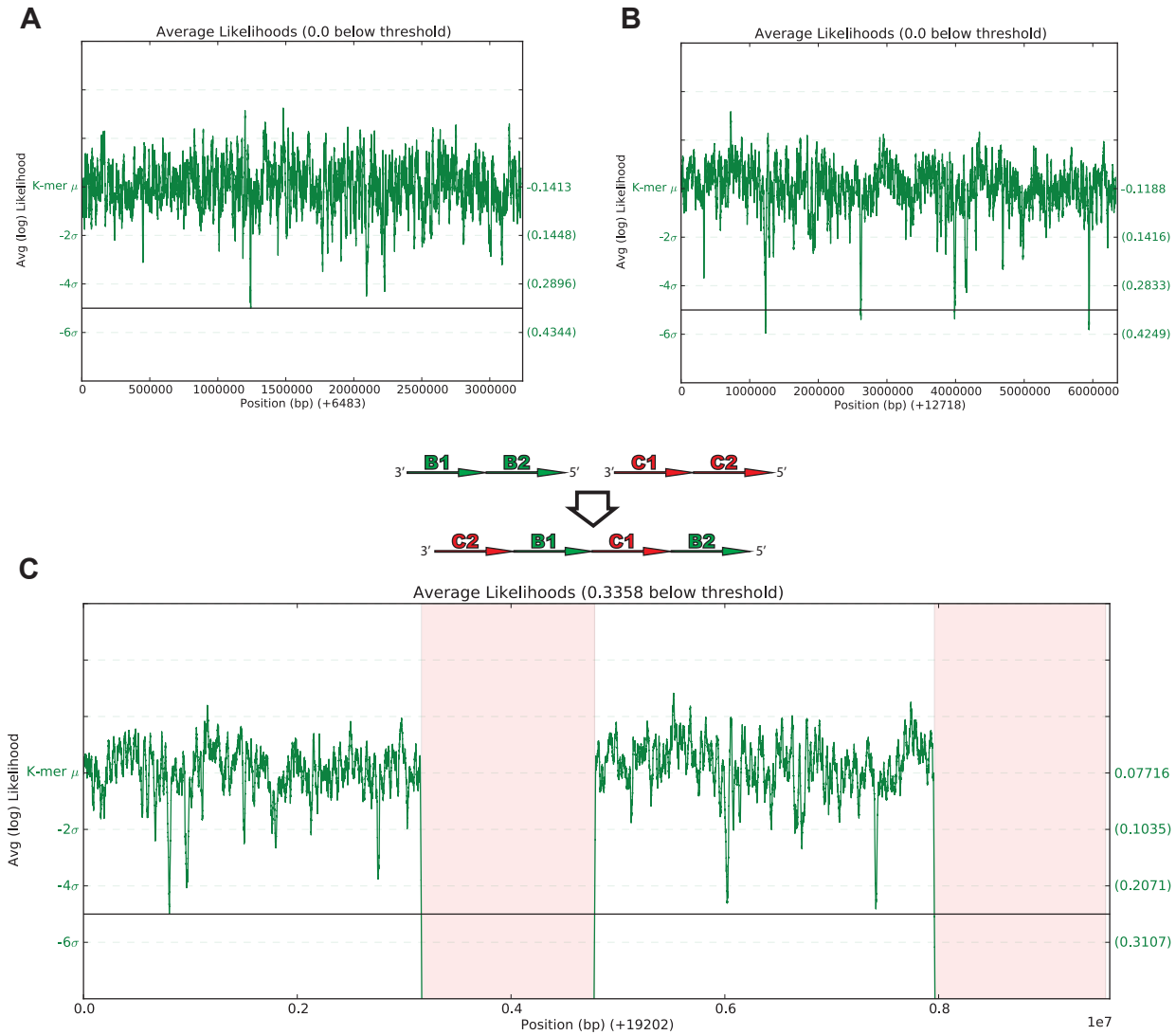
Likelihood with permutation from a known assembly



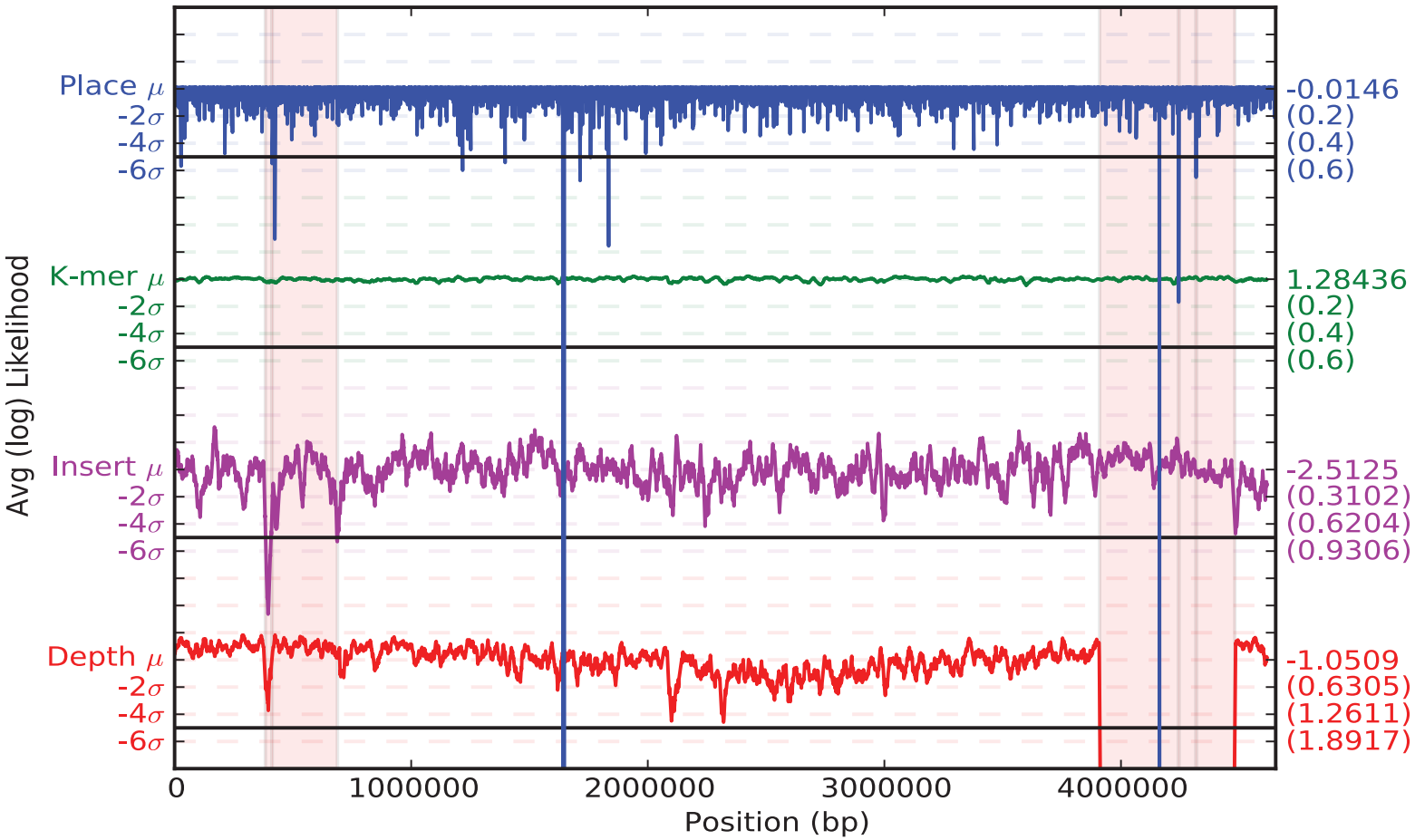
Likelihood vs # permutations from a known assembly



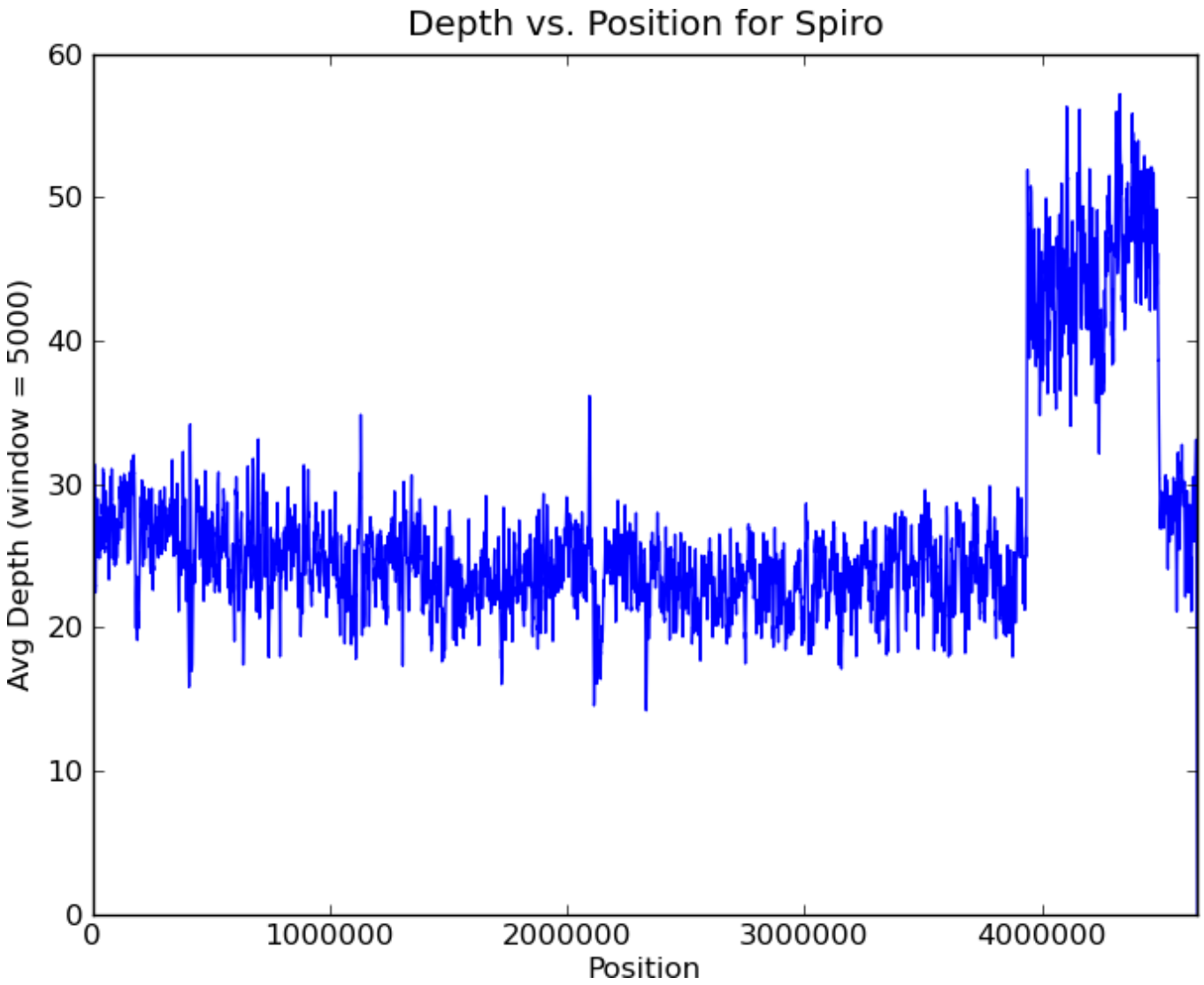
Detecting chimeric assemblies in a synthetic metagenome



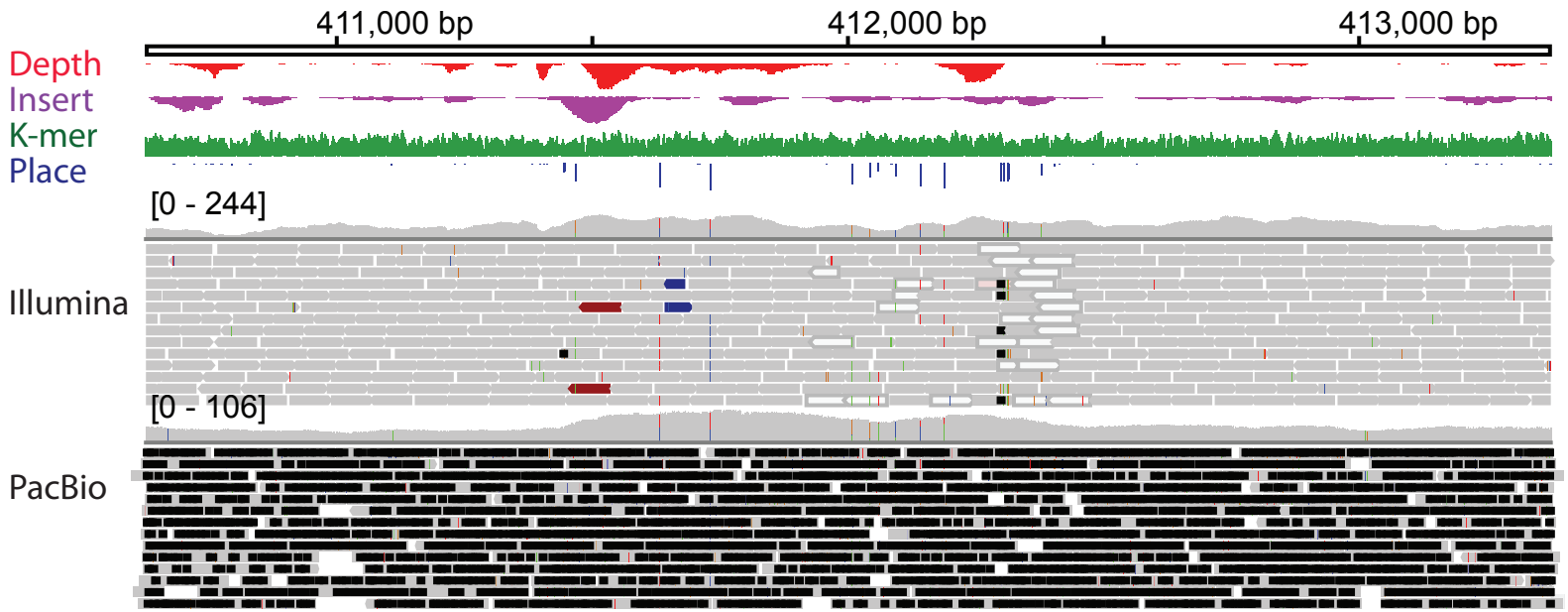
Discovery of errors in real genome assemblies



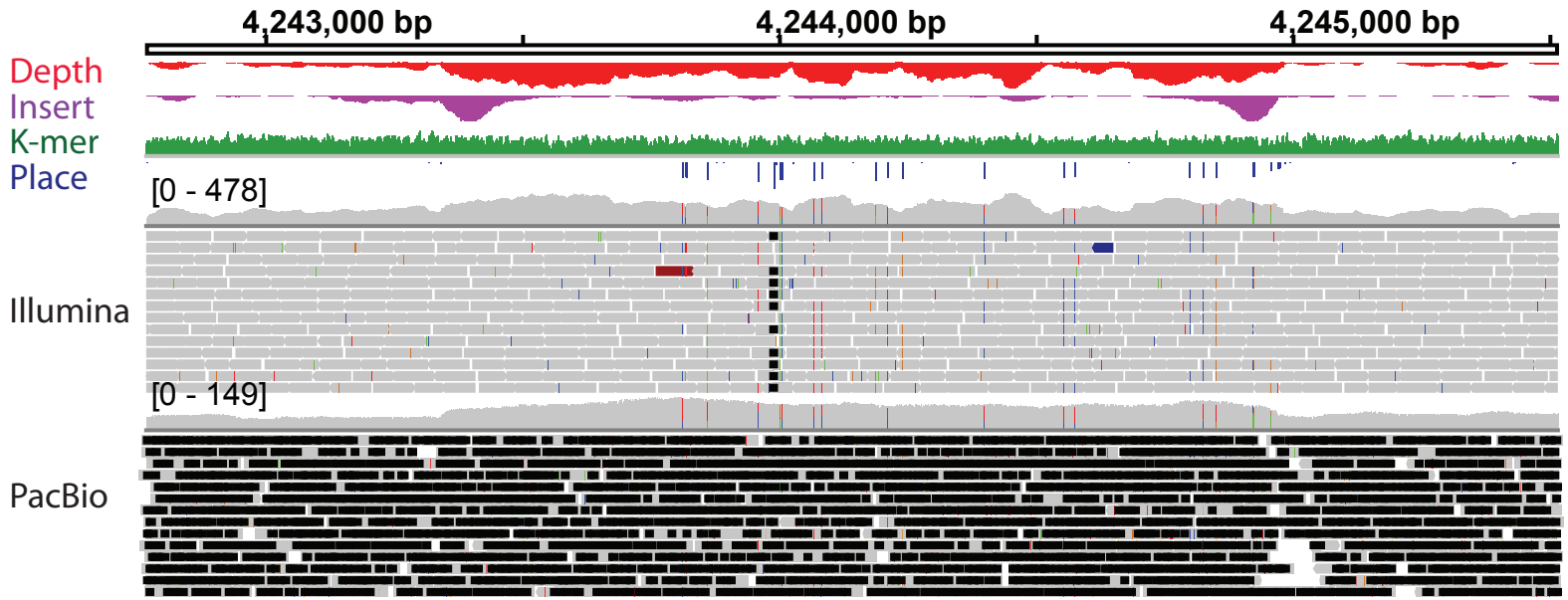
Discovery of errors in real genome assemblies



Discovery of errors in real genome assemblies



Discovery of errors in real genome assemblies



Performance with Pacific Biosciences RS data

Operation Type	Mutation Details	Position	548x Evicon (PacBio)	50x Rank (ALE)	548x Rank (ALE)
Sub	C → A	881	1	5	5
Ins	CC → CCC	2161	-	14	12
Del	G → -	3681	1	9	8
N/A	-	15712	-	7	-
Del	AACGGGCAGA	16561	1	4	4
Ins	AACGGGCAGA	17030	1	3	2
Sub	A → G	22881	1	10	7
Sub	T → A	28561	1	11	10
Del	T	34560	1	12	11
Sub	G → C	36560	1	8	9
Ins	ACGTACGT	40721	1	1	1
N/A	-	41318	-	13	-
Del	TCATCGCG	43200	-	6	6
Ins	C	47600	1	2	3

ALE Conclusion

In ALE we have developed an algorithm and software package that

- Can validate assemblies in a rigorous and probabilistic way*
- Allows for the quick discovery of errors in assemblies
- Can validate metagenomic assemblies and datasets*
- Can be used to help "finish" genomes
- Open source, easy to install, easy to fit into a pipeline

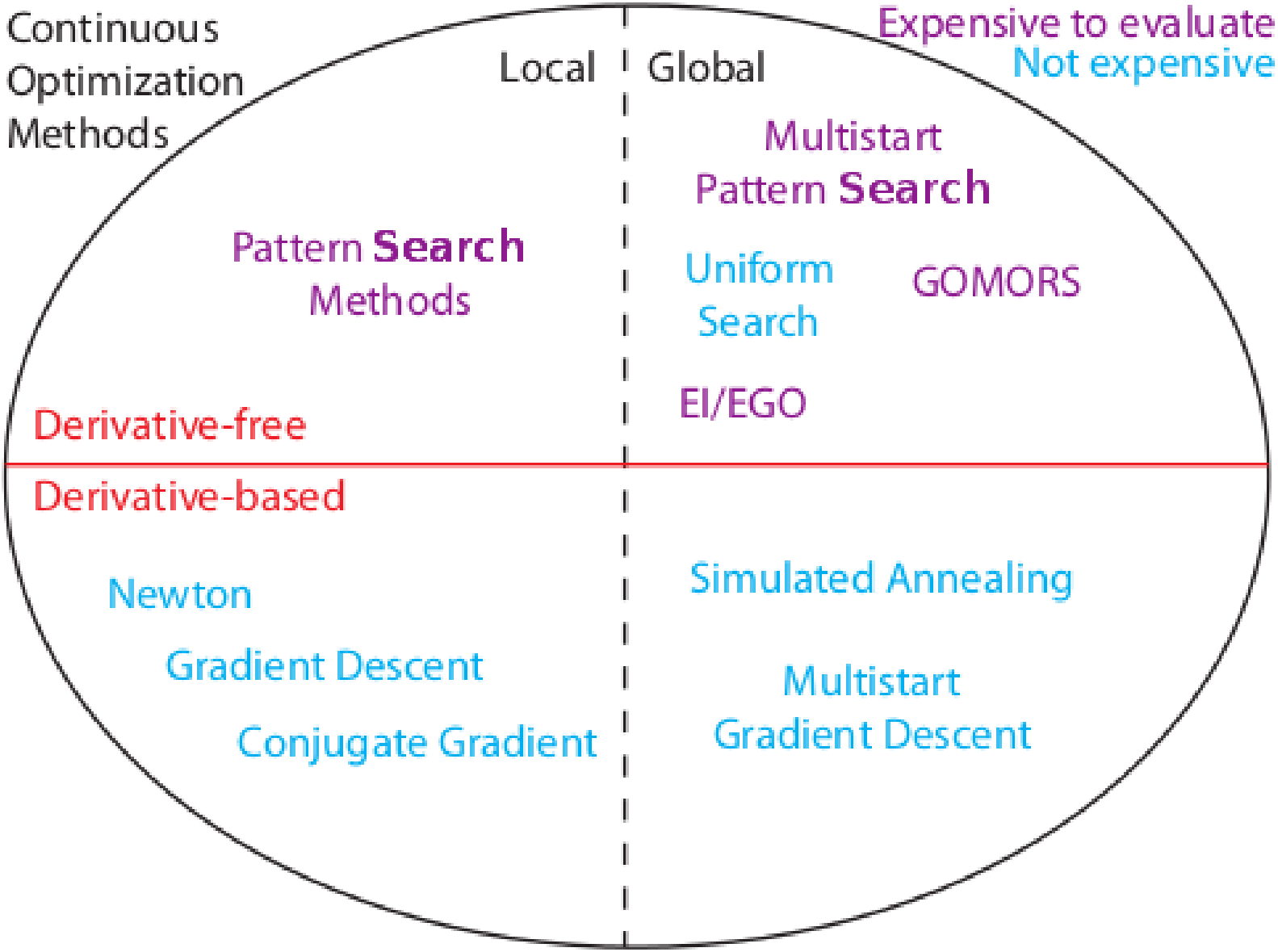
Expected Parallel Improvement

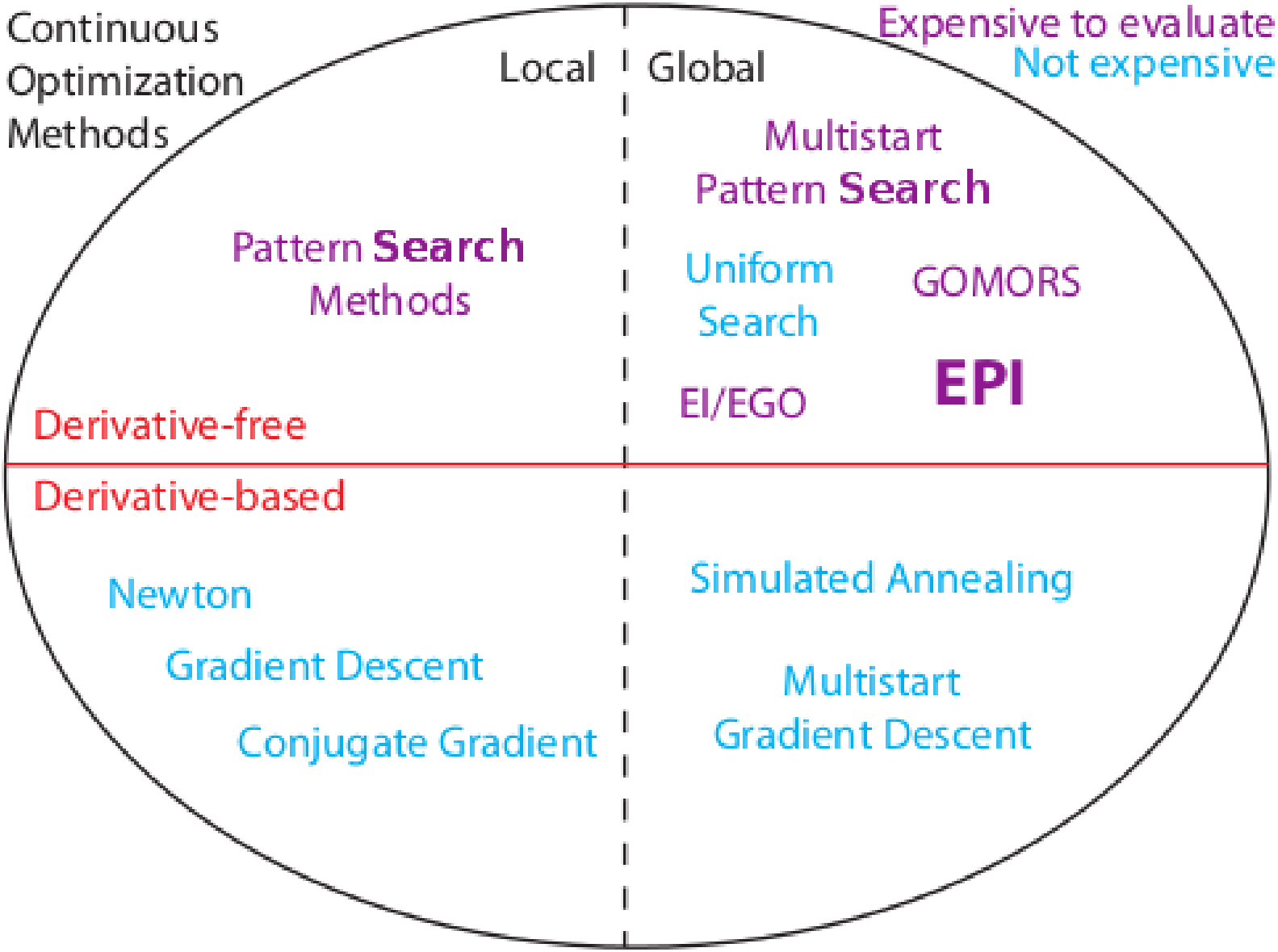
We want a fast/efficient, derivative-free global optimization algorithm for an expensive to evaluate, "black box" and possibly non-convex function f.

$$\min_{x \in A} f(x).$$

The function f has domain $A \subseteq \mathbb{R}^d$, possibly constrained.

This function can be anything, like an assembly likelihood given either the reference or some parameters for an assembler.


Continuous Optimization Methods
Local
Global
Expensive to evaluate
Not expensive
Multistart Pattern Search
Pattern Search Methods
Uniform Search
GOMORS
EI/EGO
Derivative-free
Derivative-based
Newton
Gradient Descent
Conjugate Gradient
Simulated Annealing
Multistart Gradient Descent


Continuous Optimization Methods
Local
Global
Expensive to evaluate
Not expensive
Multistart Pattern Search
Pattern Search Methods
Uniform Search
GOMORS
EPI
EI/EGO
Derivative-free
Derivative-based
Newton
Simulated Annealing
Gradient Descent
Multistart Gradient Descent
Conjugate Gradient

The math (figures next!)

We begin with a Gaussian process prior on a continuous function f.

$$f \sim \mathrm{GP}(\mu(\cdot), \Sigma(\cdot, \cdot))$$

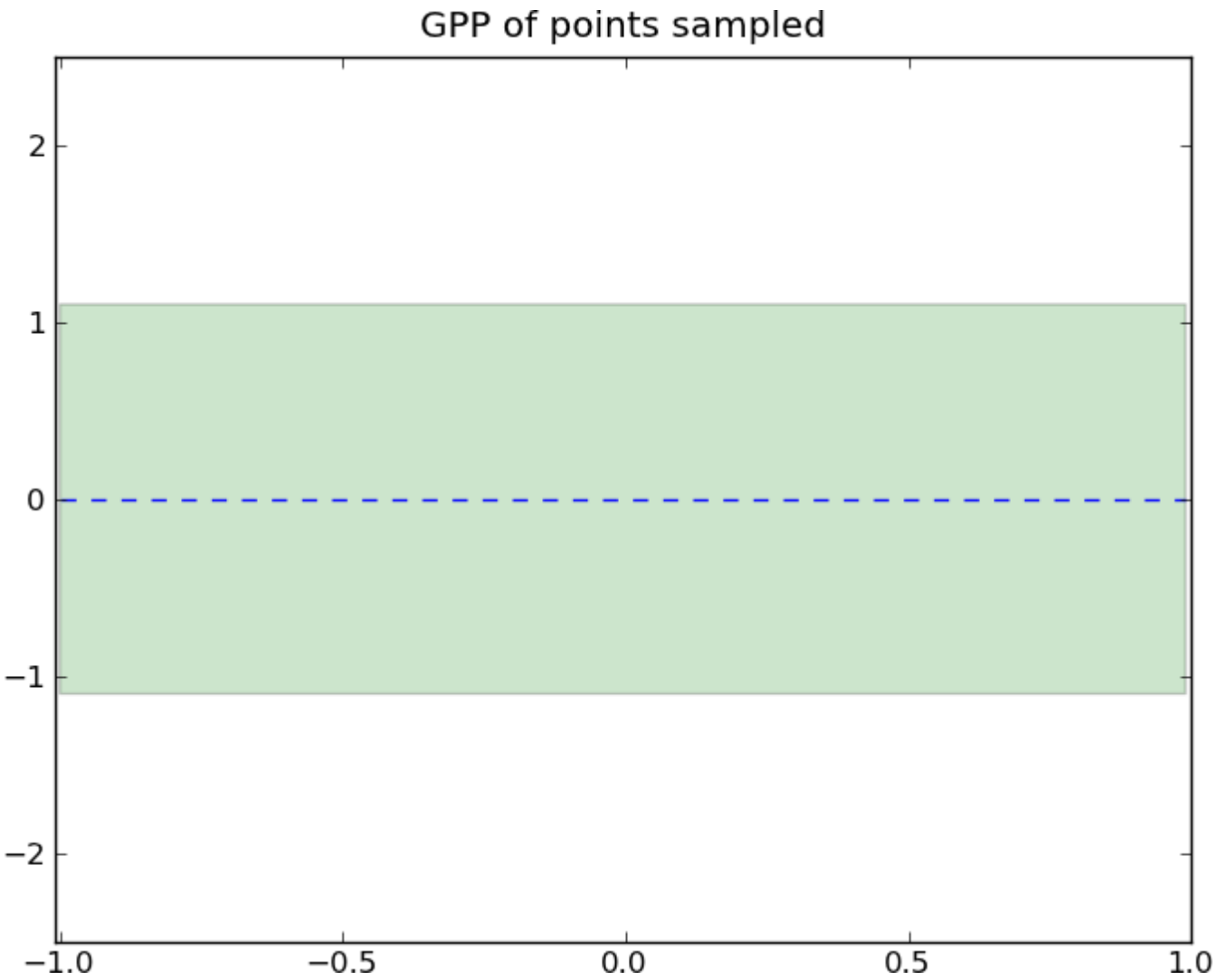
$$(f(\vec{x}_1), \ldots, f(\vec{x}_n)) \sim N\left(\begin{bmatrix} \mu(\vec{x}_1) \\ \vdots \\ \mu(\vec{x}_n) \end{bmatrix}, \begin{bmatrix} \Sigma(\vec{x}_1, \vec{x}_1) & \cdots & \Sigma(\vec{x}_n, \vec{x}_1) \\ \vdots & \ddots & \vdots \\ \Sigma(\vec{x}_1, \vec{x}_n) & \cdots & \Sigma(\vec{x}_n, \vec{x}_n) \end{bmatrix} \right)$$

With the posterior mean and covariance,

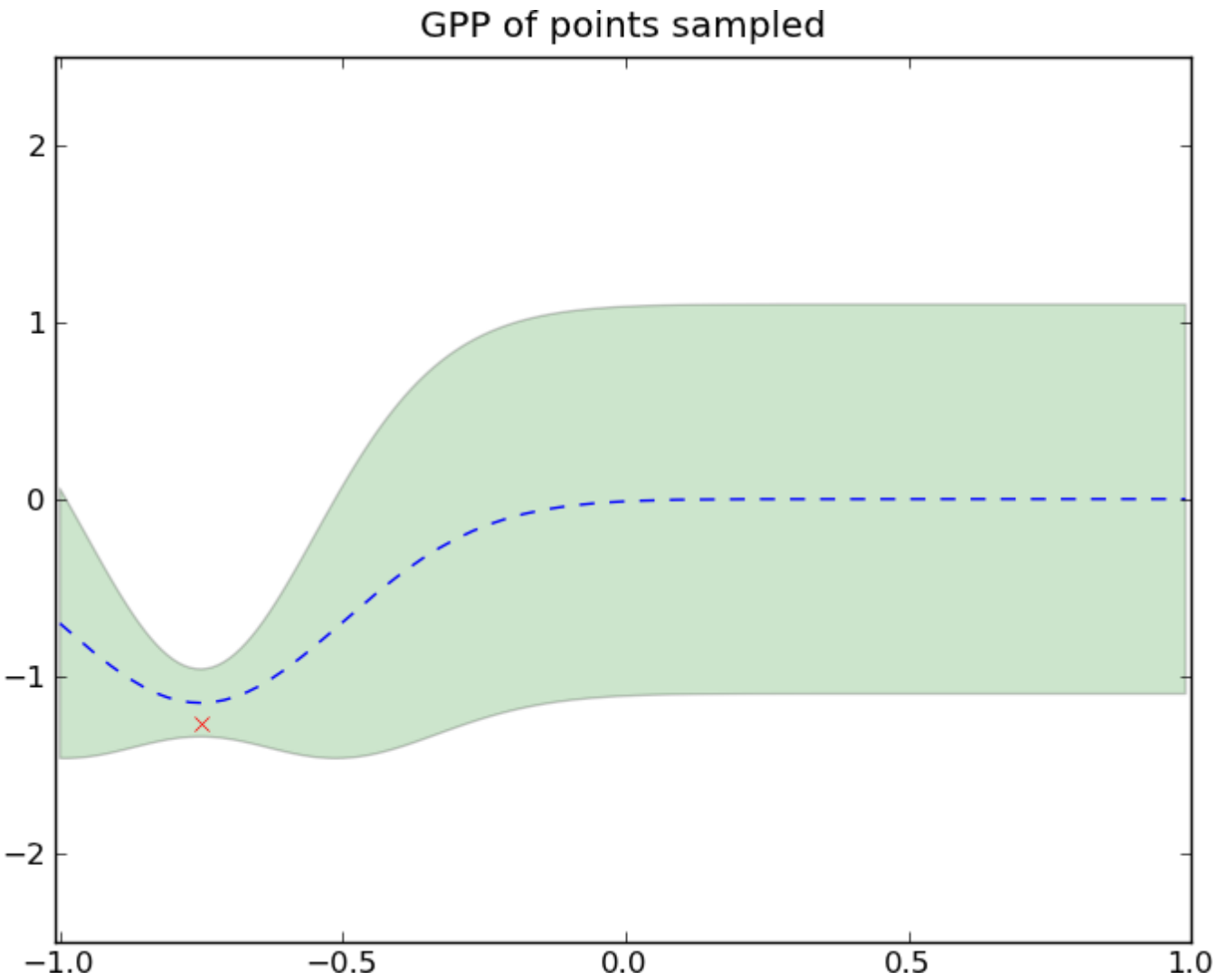
$$\mu_n(\mathbf{x}_\star) = K(\mathbf{x}_\star, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}y, \quad K(\mathbf{W}, \mathbf{Z})_{ij} = \Sigma(\mathbf{W}_i, \mathbf{Z}_j)$$

$$\Sigma_n(\mathbf{x}_\star, \mathbf{x}_\star) = K(\mathbf{x}_\star, \mathbf{x}_\star) - K(\mathbf{x}_\star, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x}_\star)$$

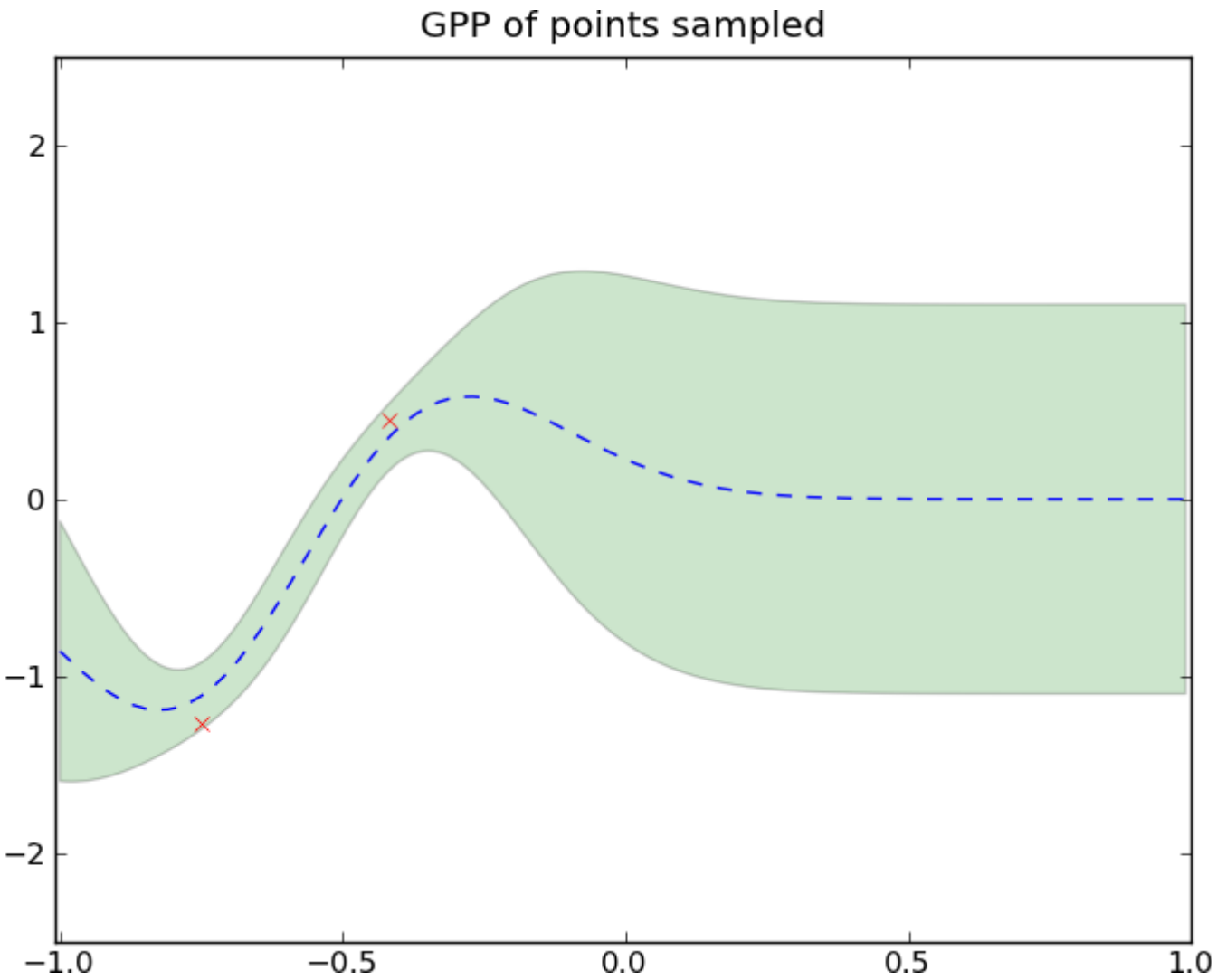
GPP evolving with information



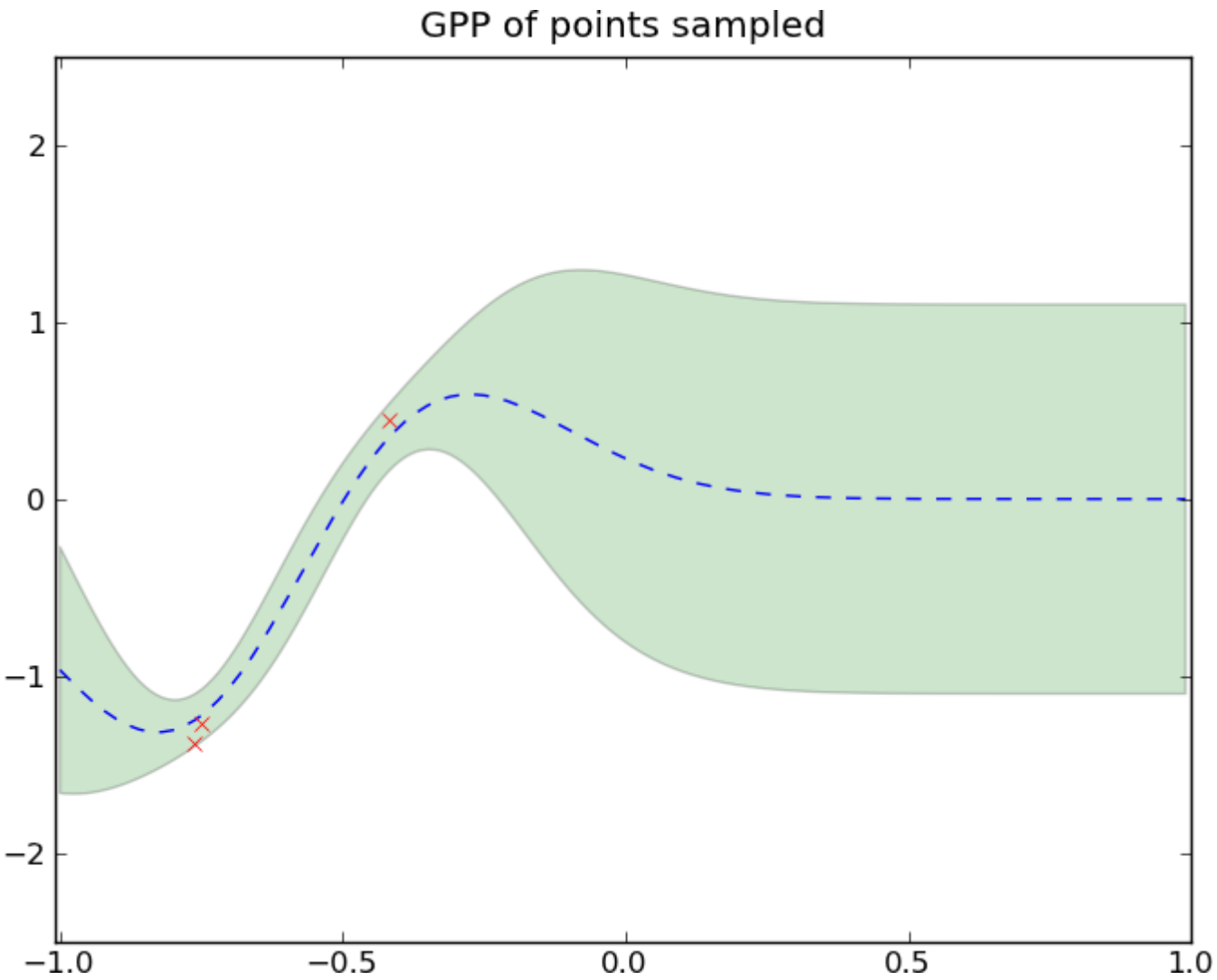
GPP evolving with information



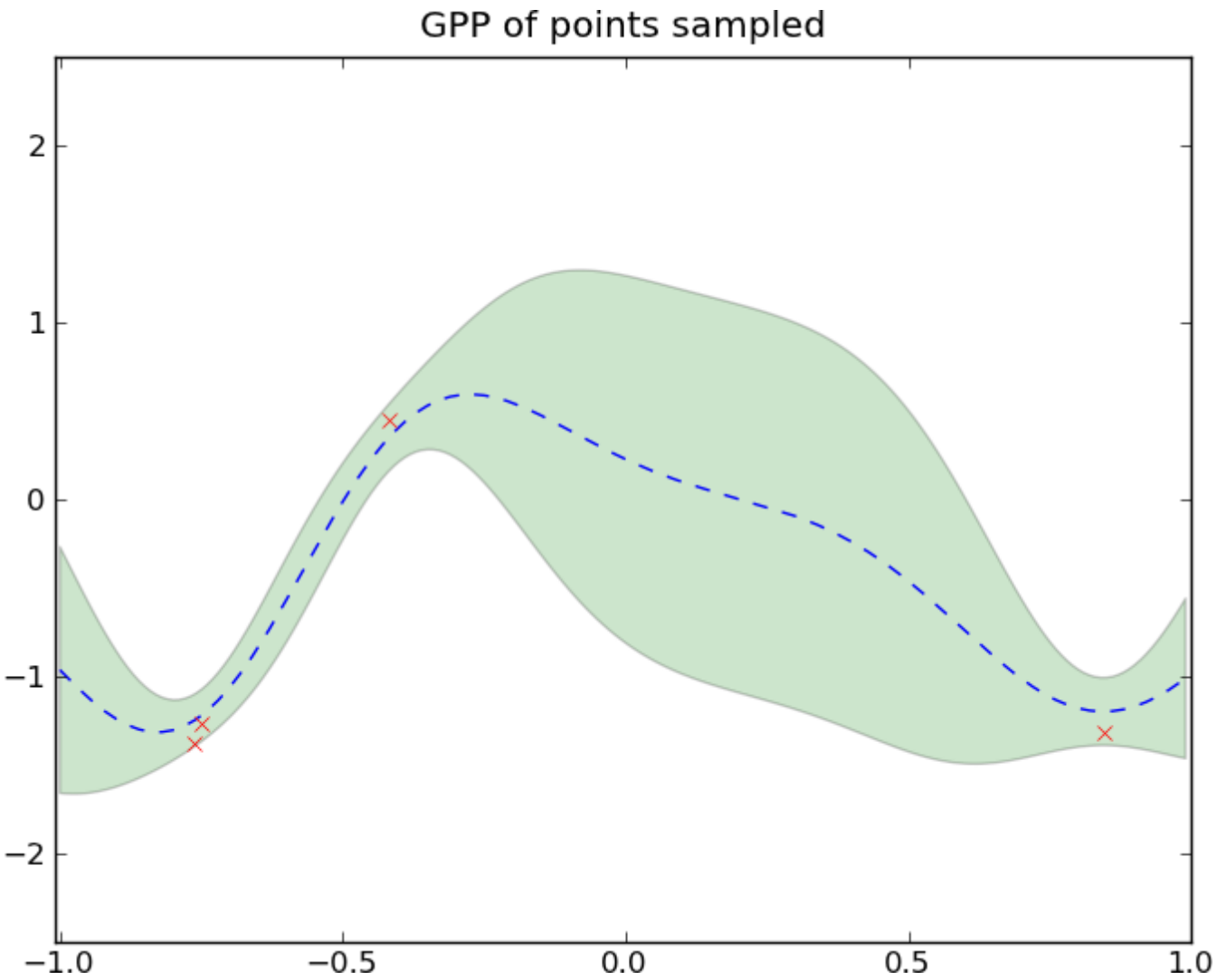
GPP evolving with information



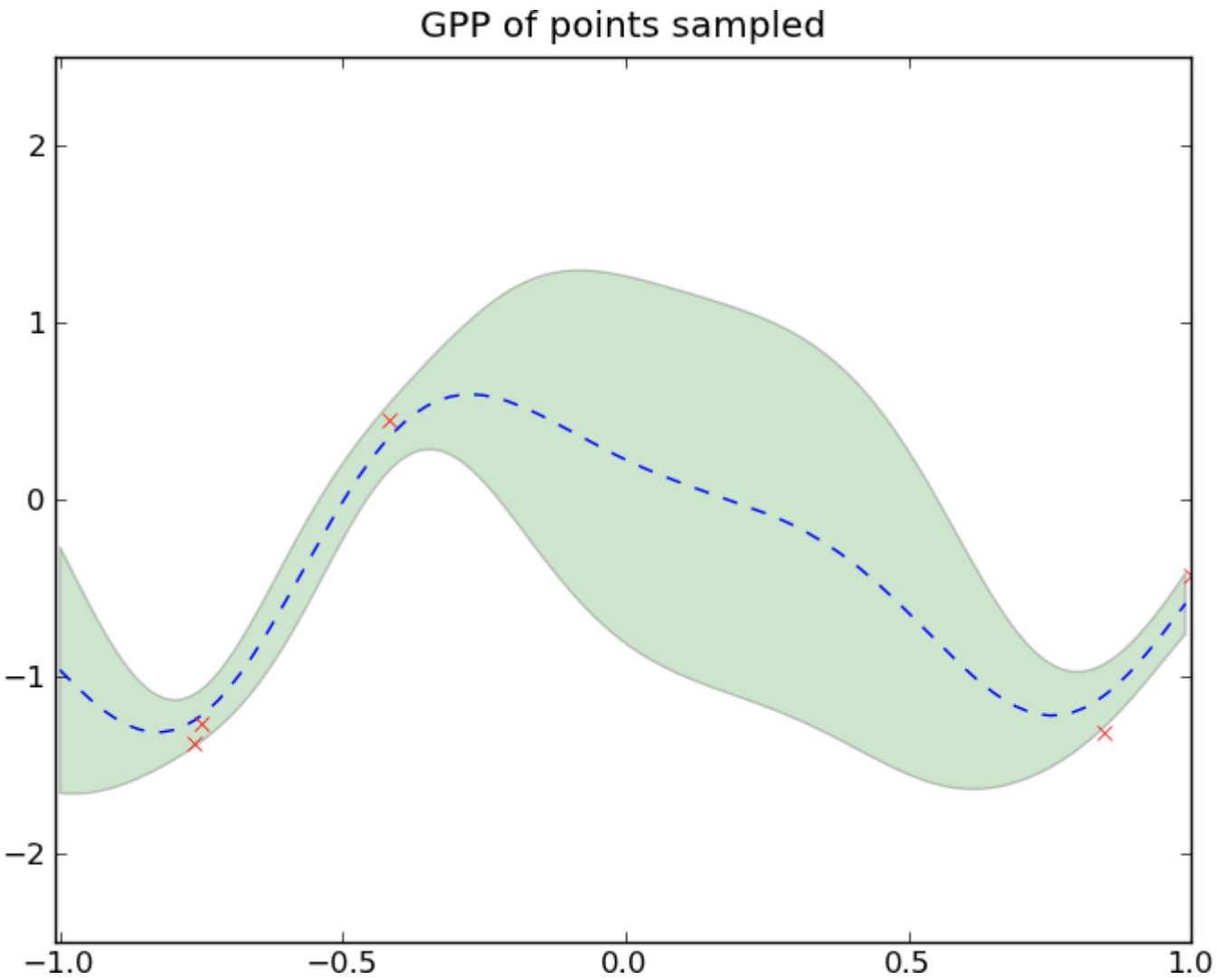
GPP evolving with information



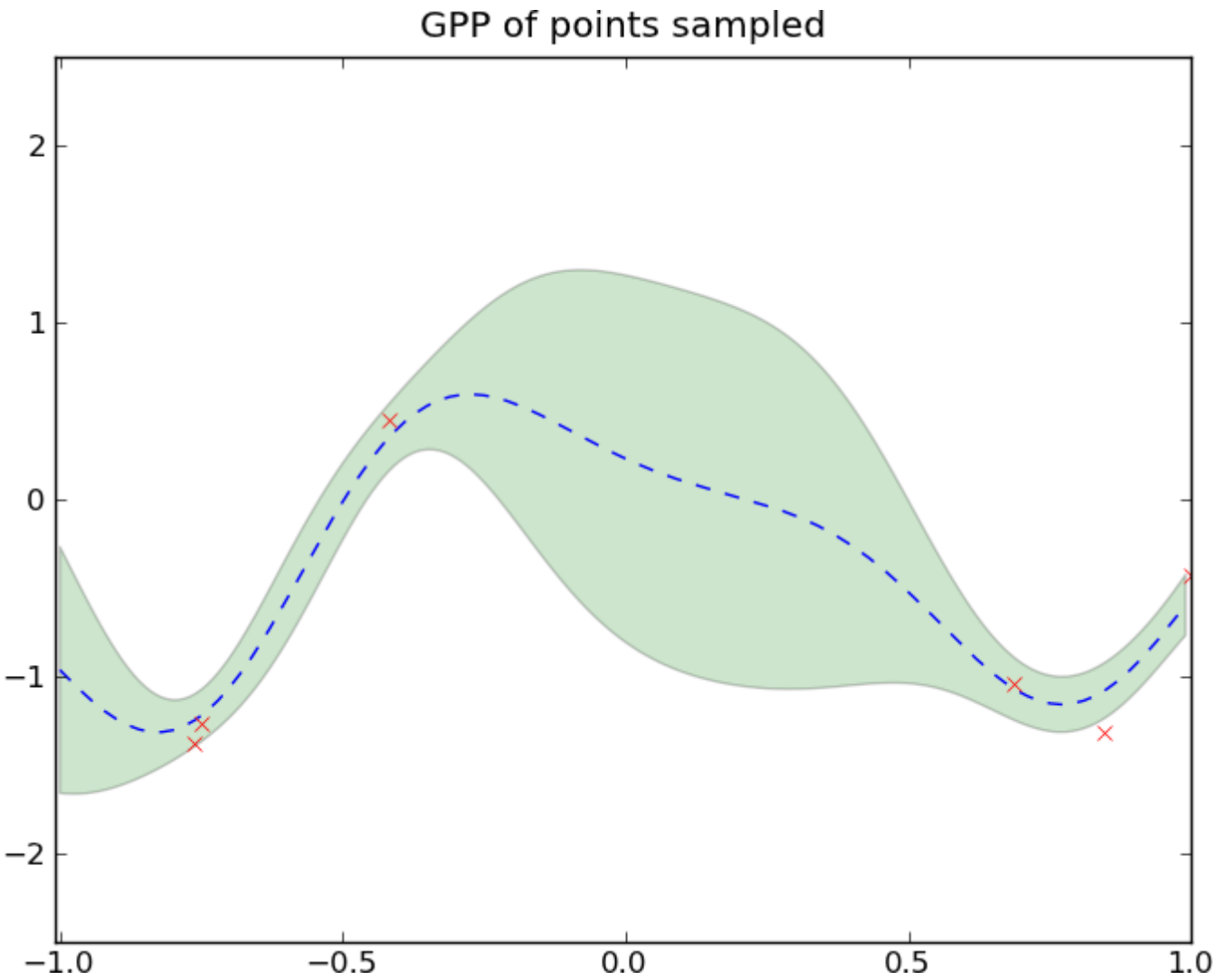
GPP evolving with information



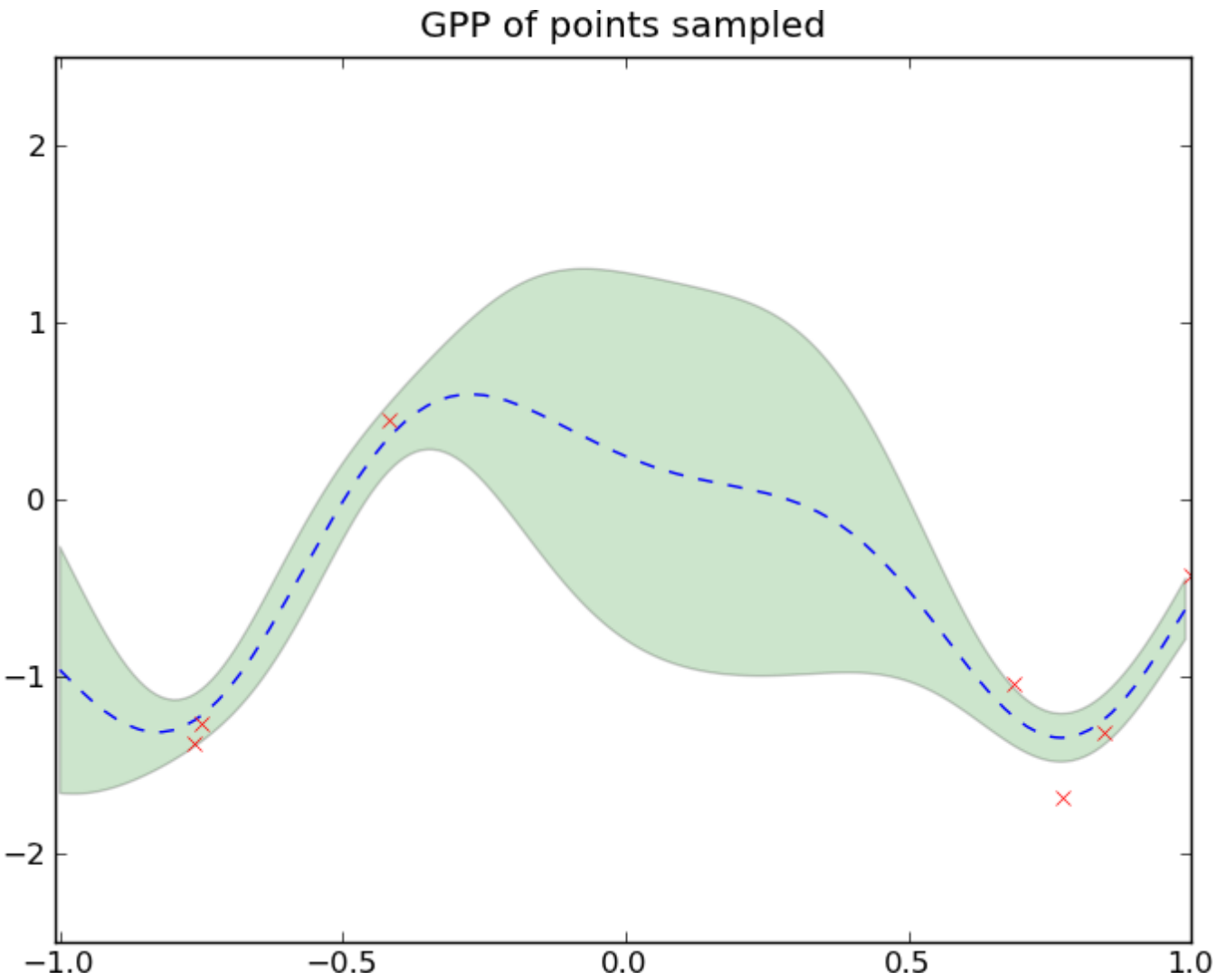
GPP evolving with information



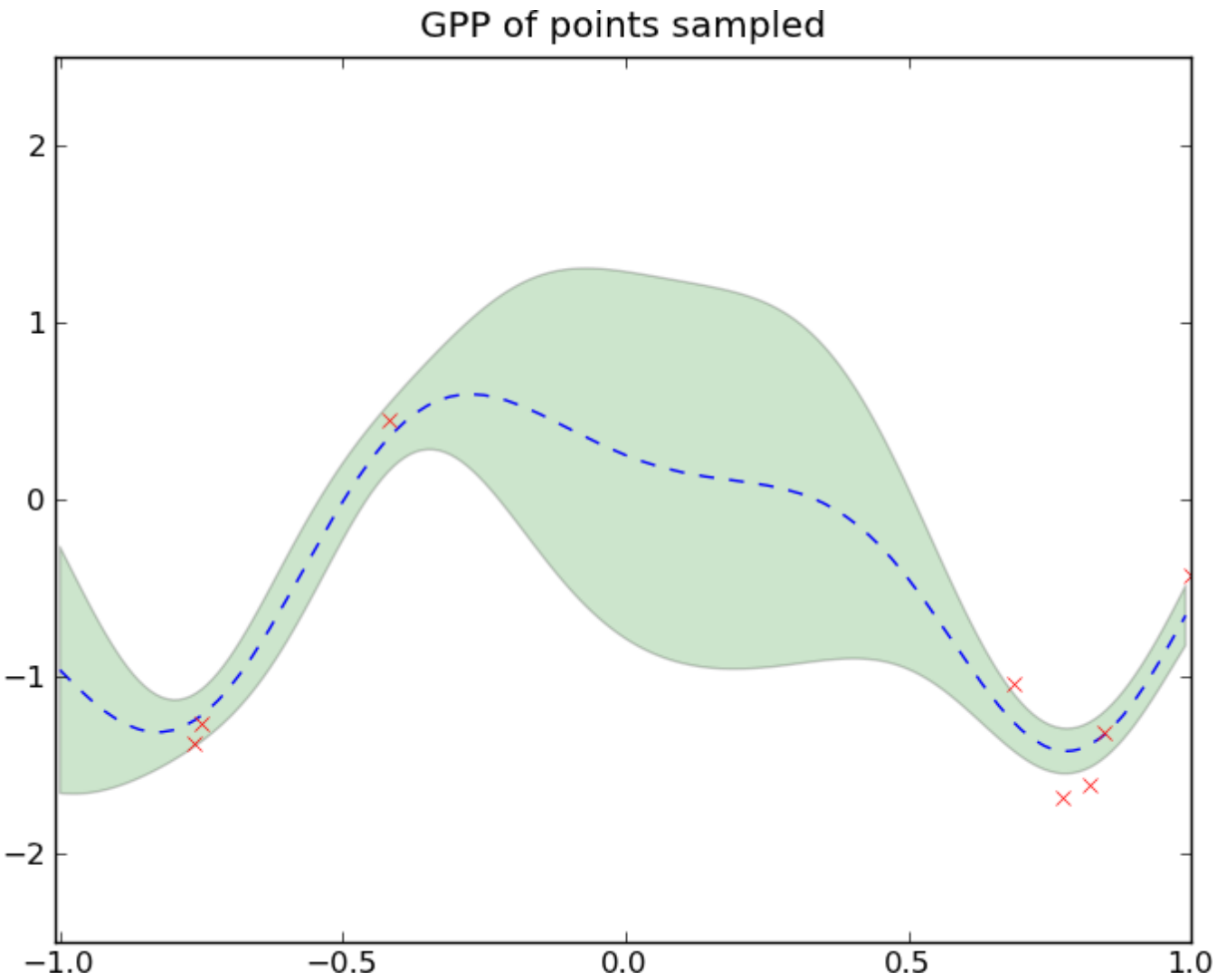
GPP evolving with information



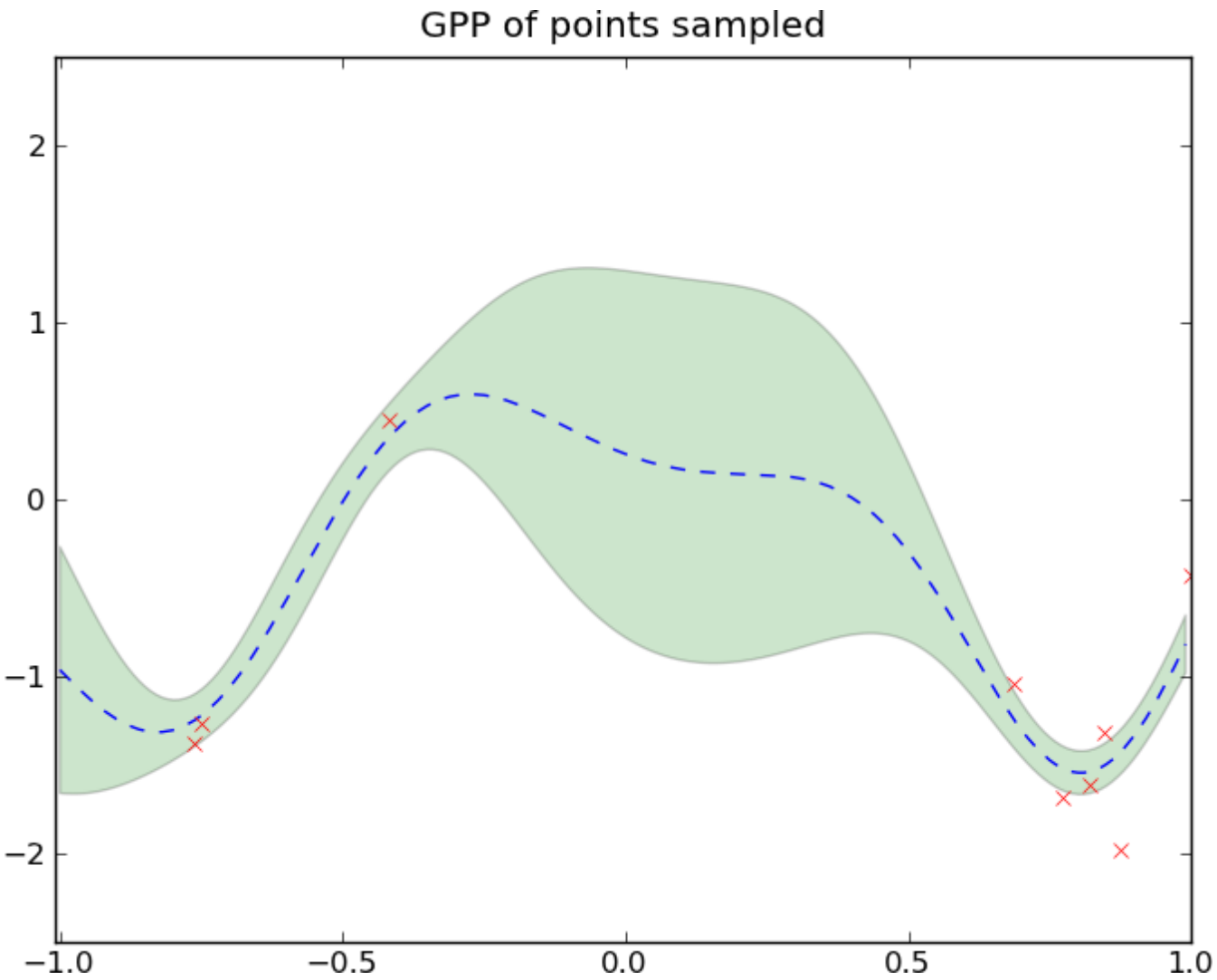
GPP evolving with information



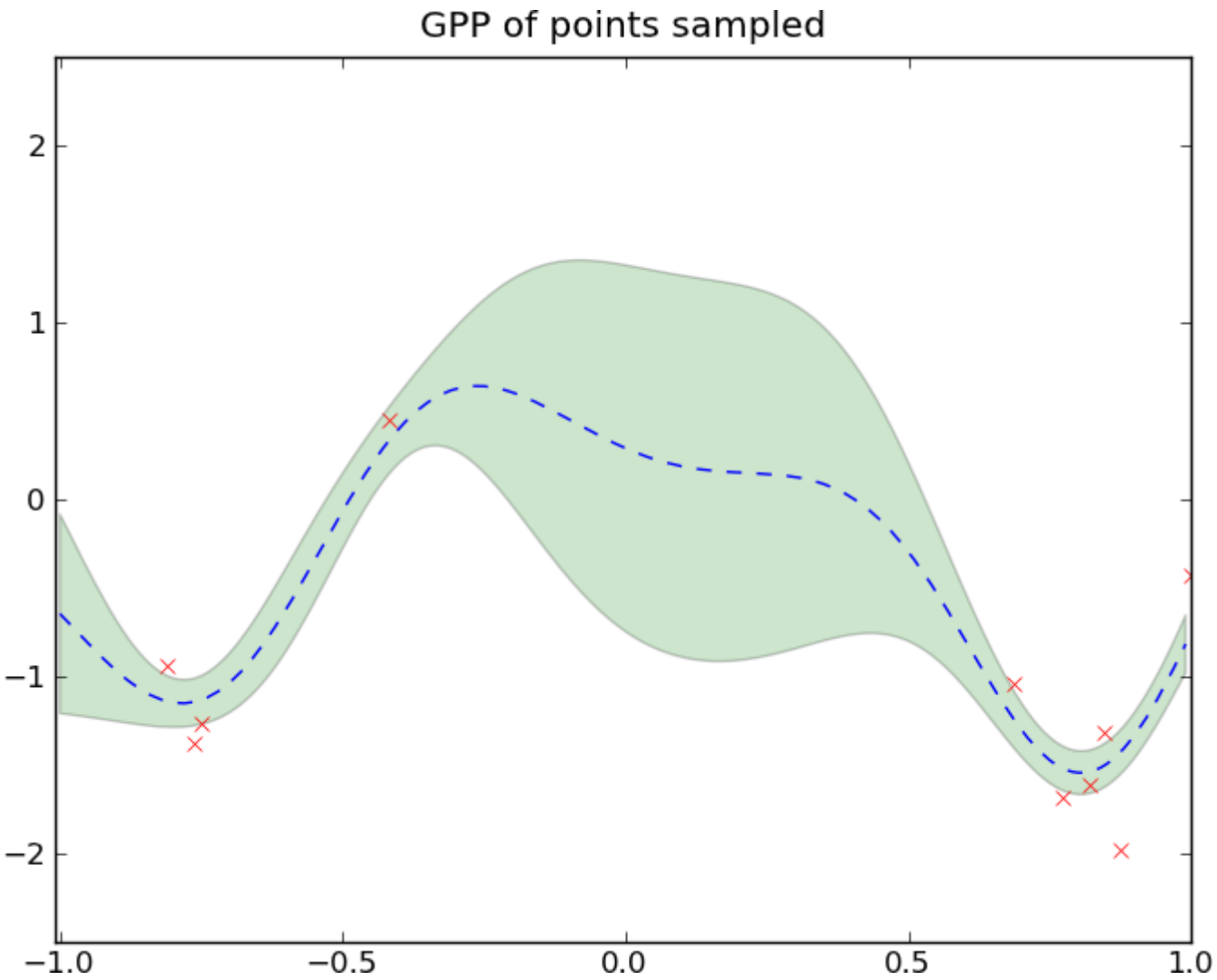
GPP evolving with information



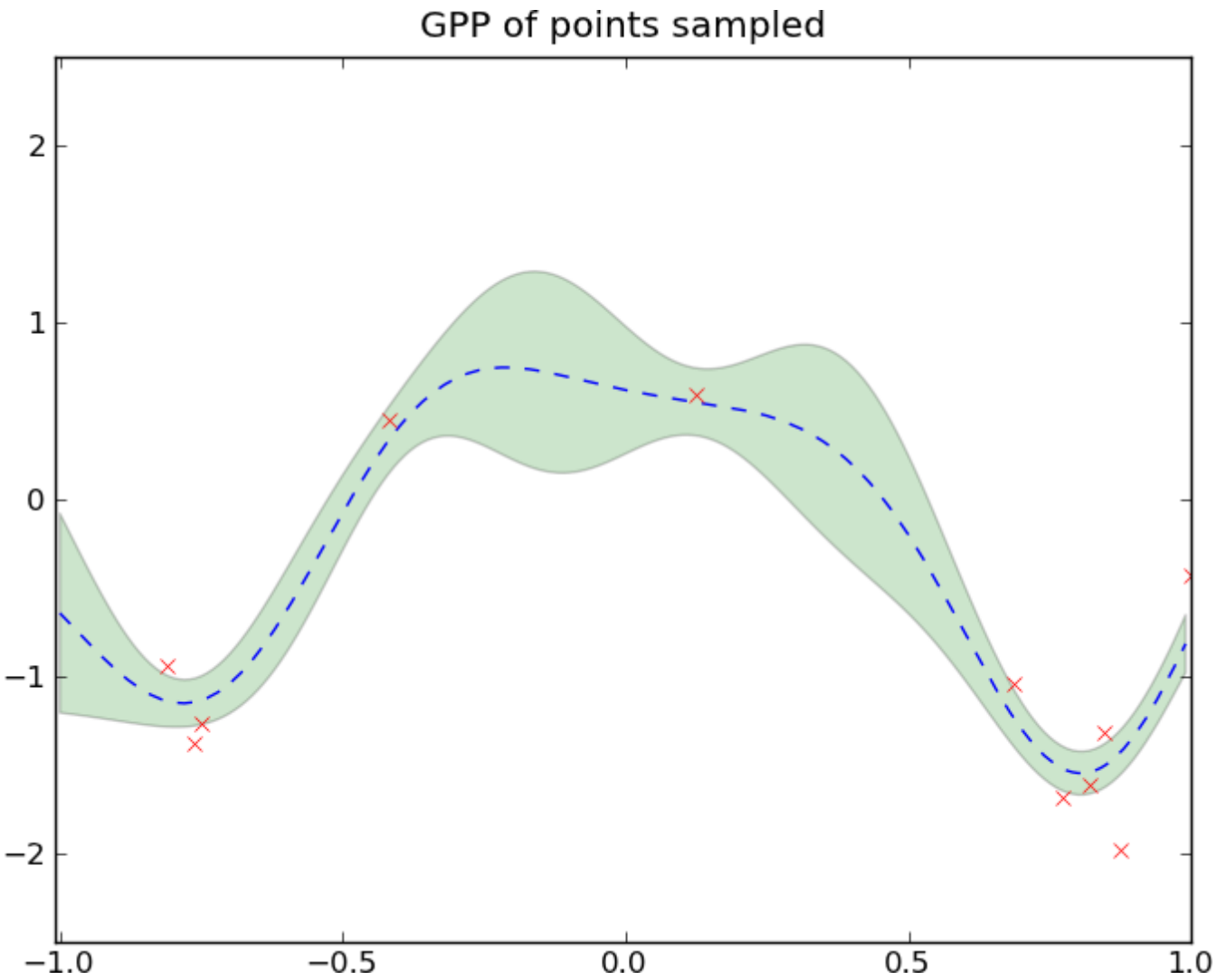
GPP evolving with information



GPP evolving with information



GPP evolving with information



Expected Improvement

$$\mathrm{EI}(\vec{x}_\star) = \mathbb{E}_n\left[[f(\vec{x}_\star) - f_n^\star]^+\right] = \mathbb{E}\left[f_{n+1}^\star - f_n^\star | \vec{x}_{n+1} = \vec{x}_\star\right]$$

$$\text{where } f_n^\star = \min_{m \le n} f(\vec{x}_m).$$

The core of this idea is that we can calculate the expected improvement for simultaneous evaluation of points $\{\vec{x}_{n+1}, \ldots, \vec{x}_{n+\ell}\} = \mathbf{x}_\star$ as

$$\mathrm{EI}(\mathbf{x}_\star) = \mathbb{E}_n\left[[\min\{f(\vec{x}_{n+1}), \ldots, f(\vec{x}_{n+\ell})\} - f_n^\star]^+\right].$$

The optimization then approximates the solution to

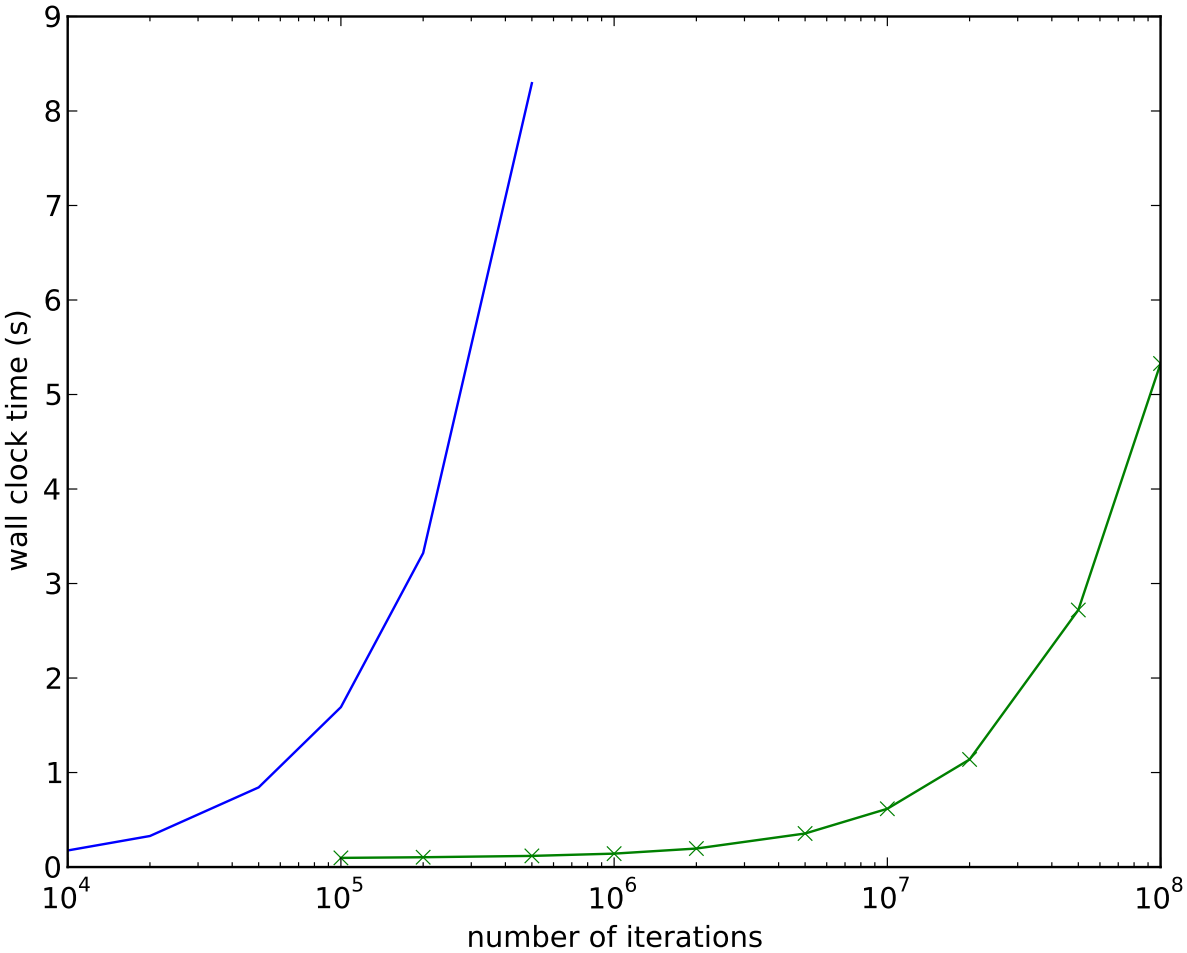
$$\underset{\mathbf{x}_\star \in \mathbb{R}^{d \times \ell}}{\operatorname{argmin}} \mathrm{EI}(\mathbf{x}_\star).$$

Perfect application to GPGPUs

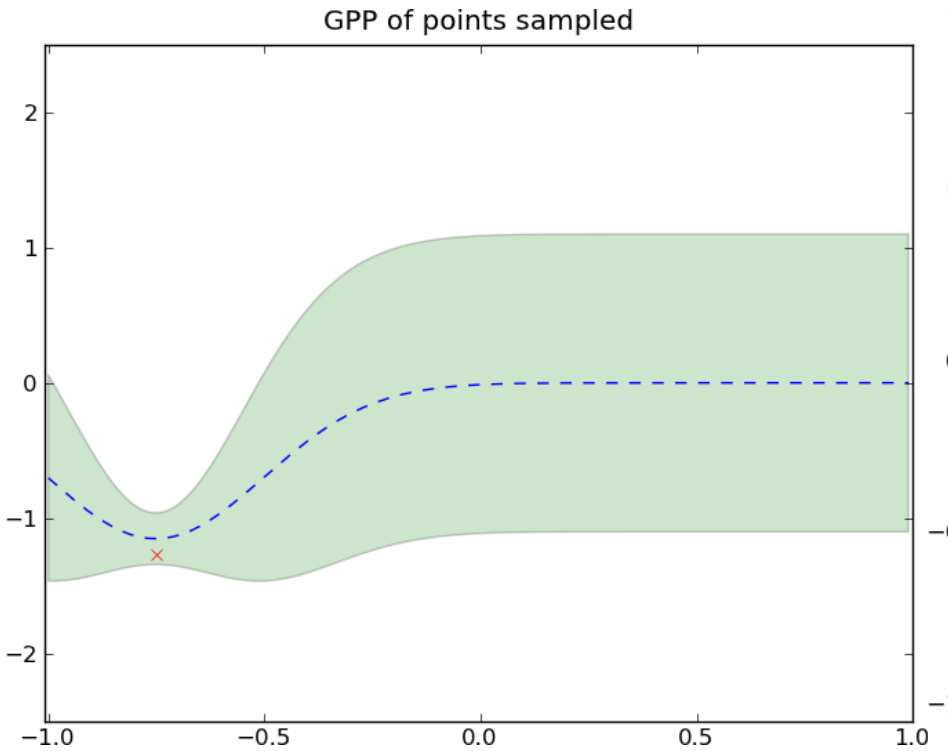
GPGPUs can provide high performance computing on the desktop

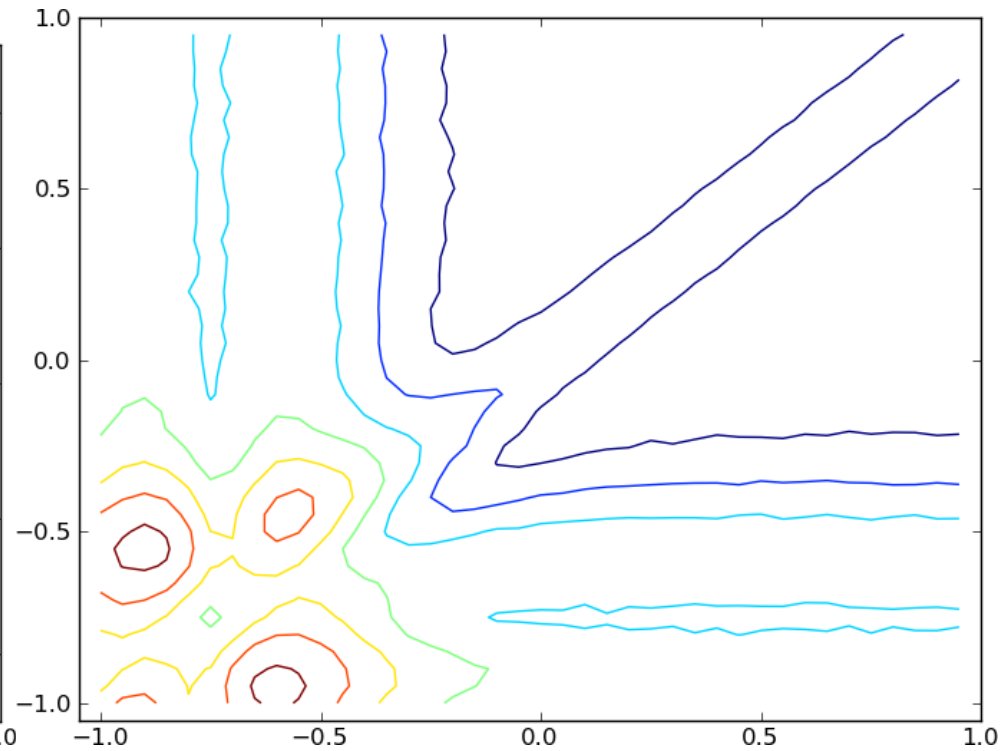
- 1536 cores on a ~$500 chip (many TFLOPs)
- Program in CUDA/openCL (C *like*)
- Tight memory restrictions
  - Low memory per core
  - Transfer across PCI-E bus is expensive

GPU speedup for MC EI

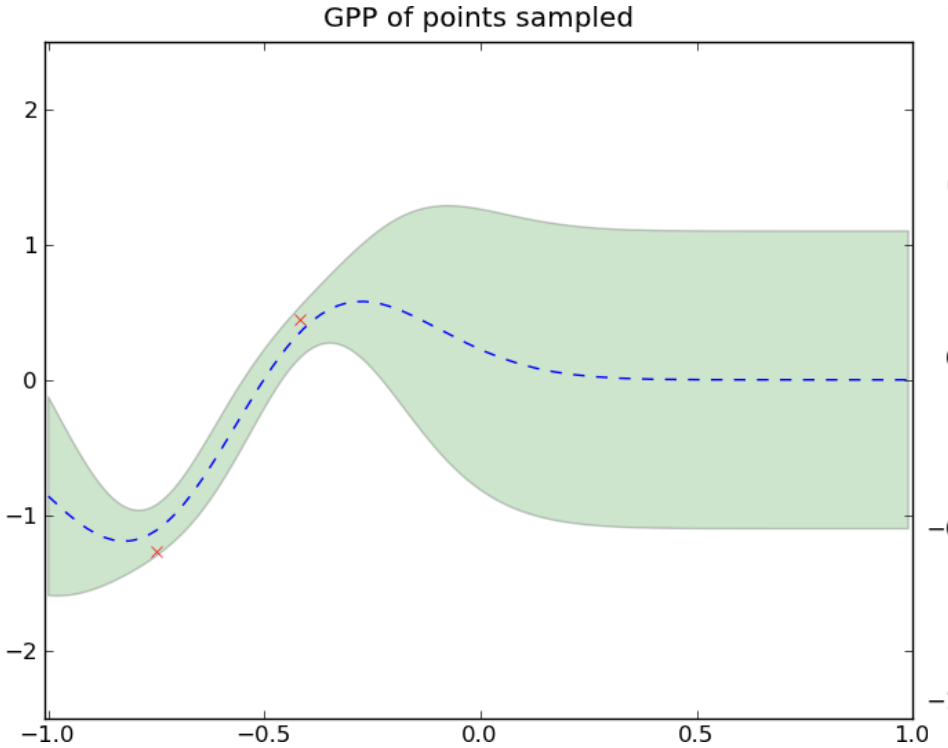


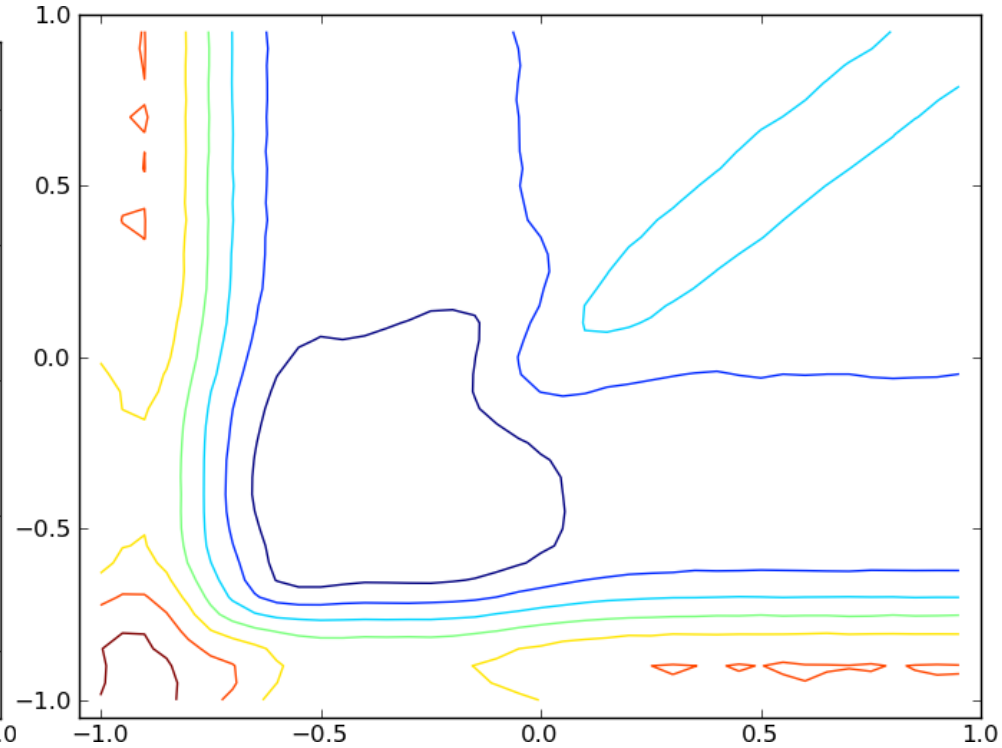
GPP evolving with information



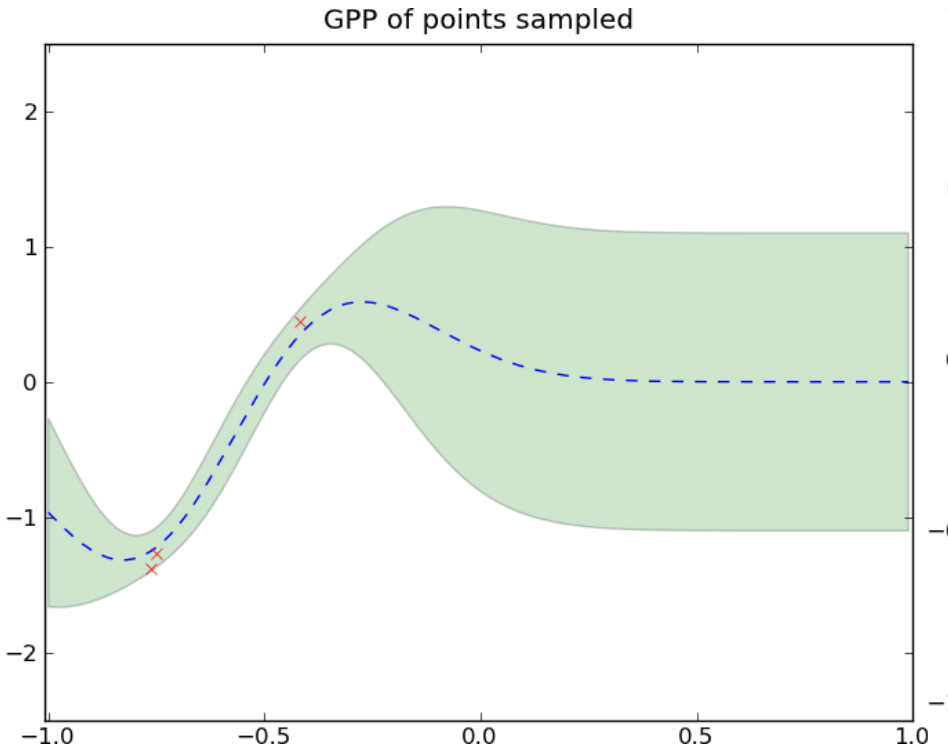


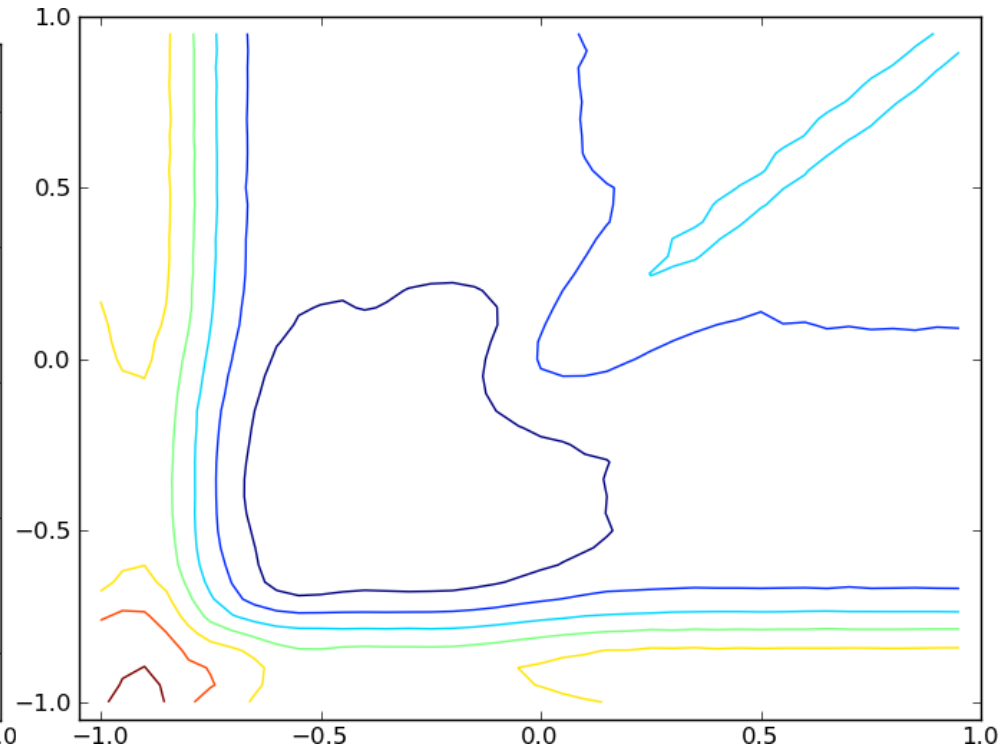
GPP evolving with information



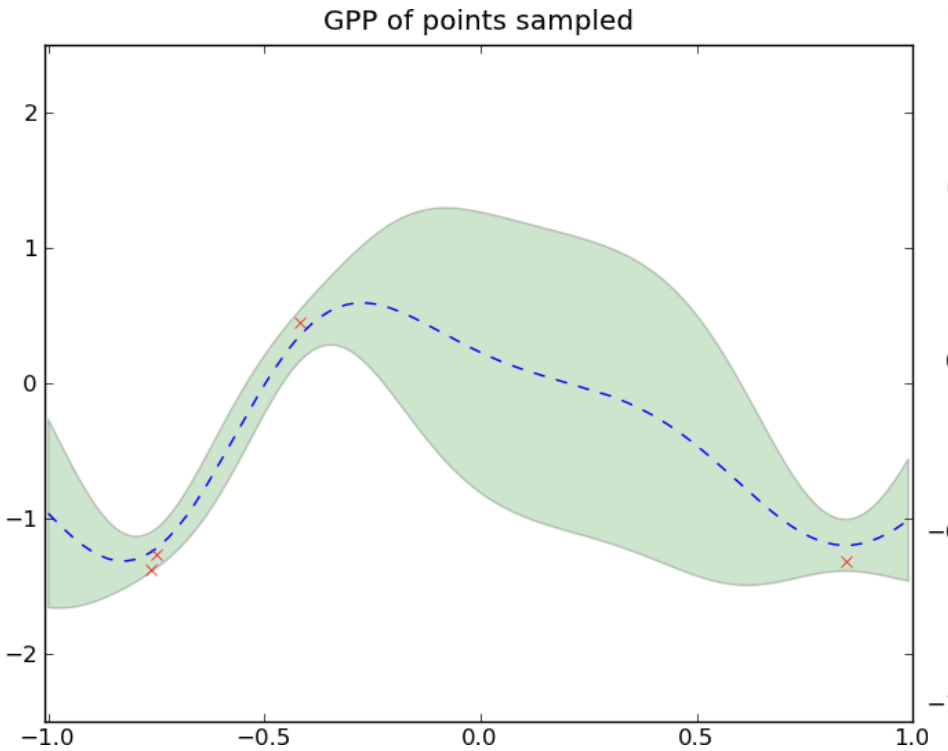


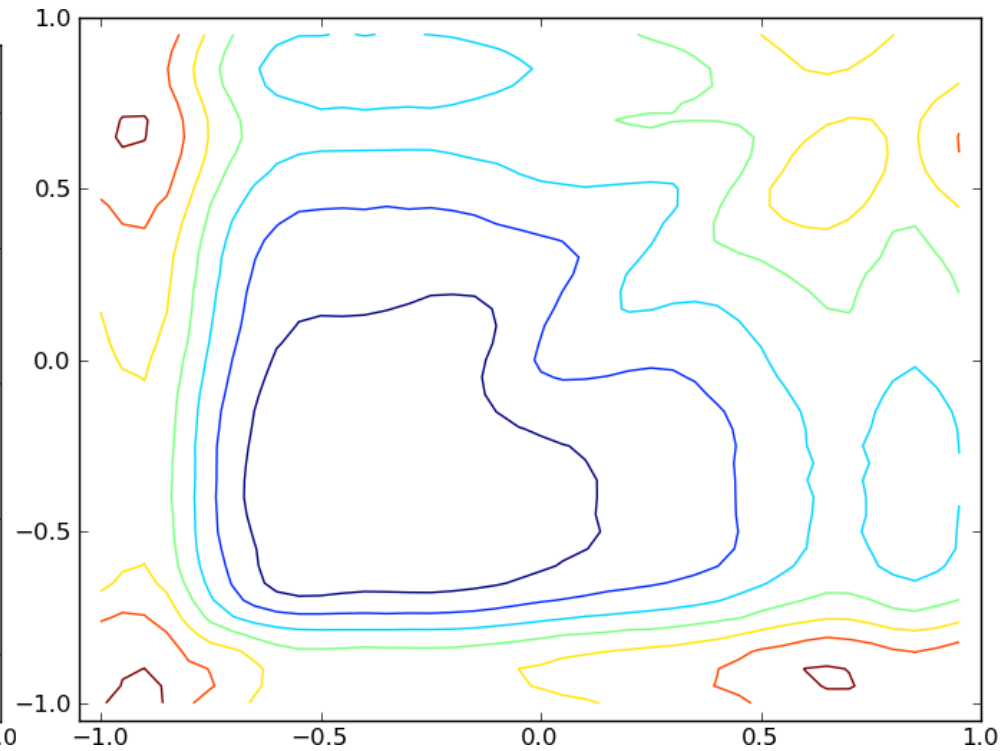
GPP evolving with information



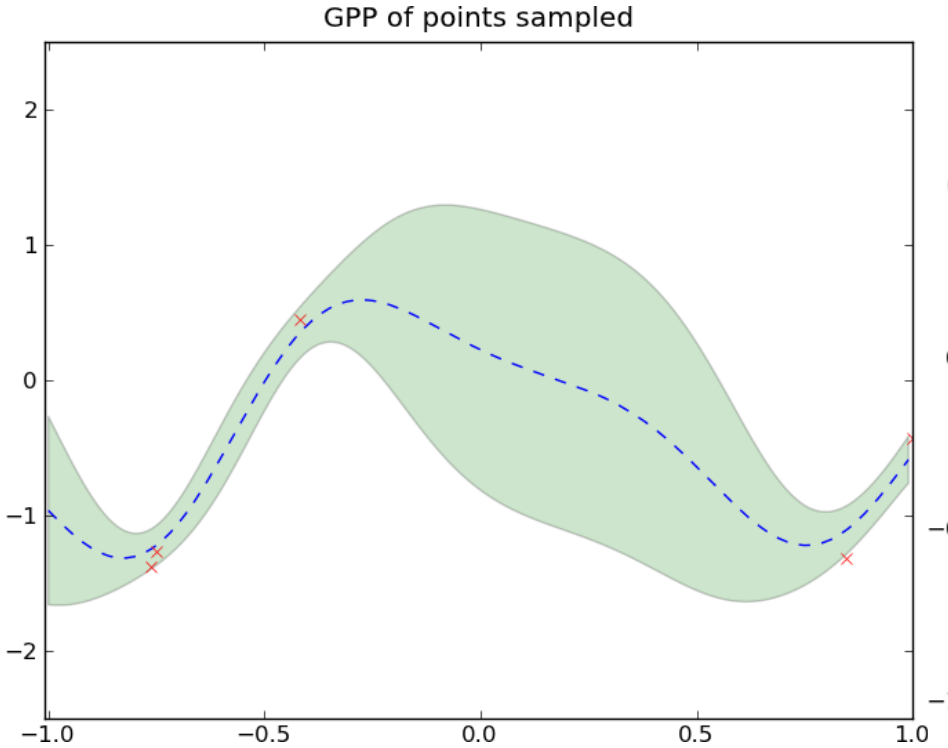


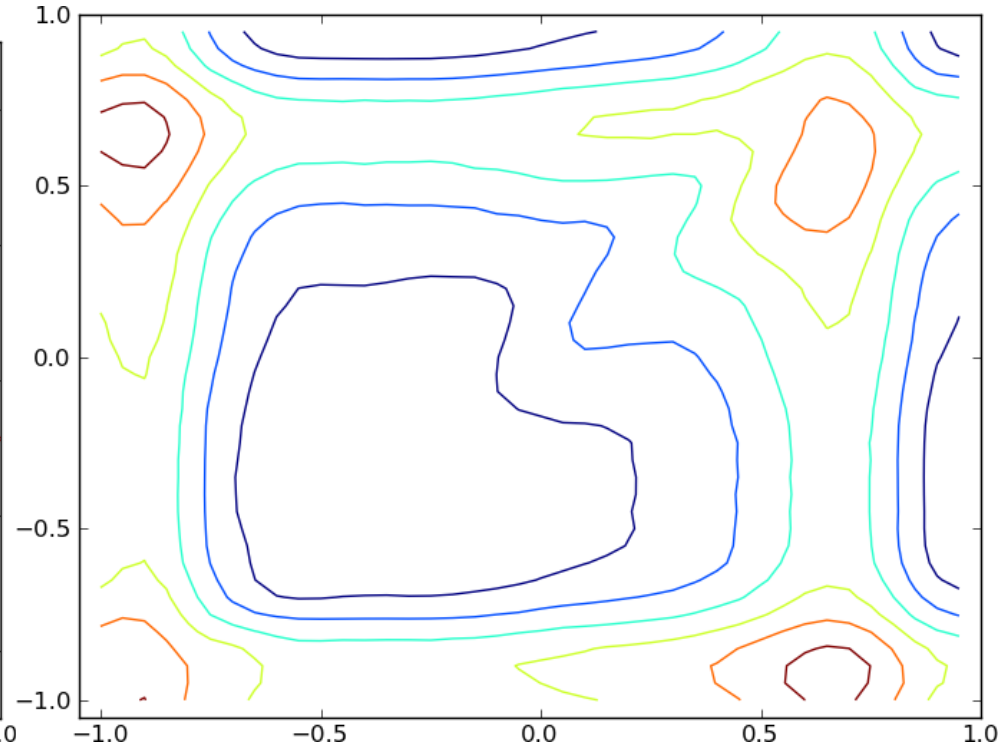
GPP evolving with information



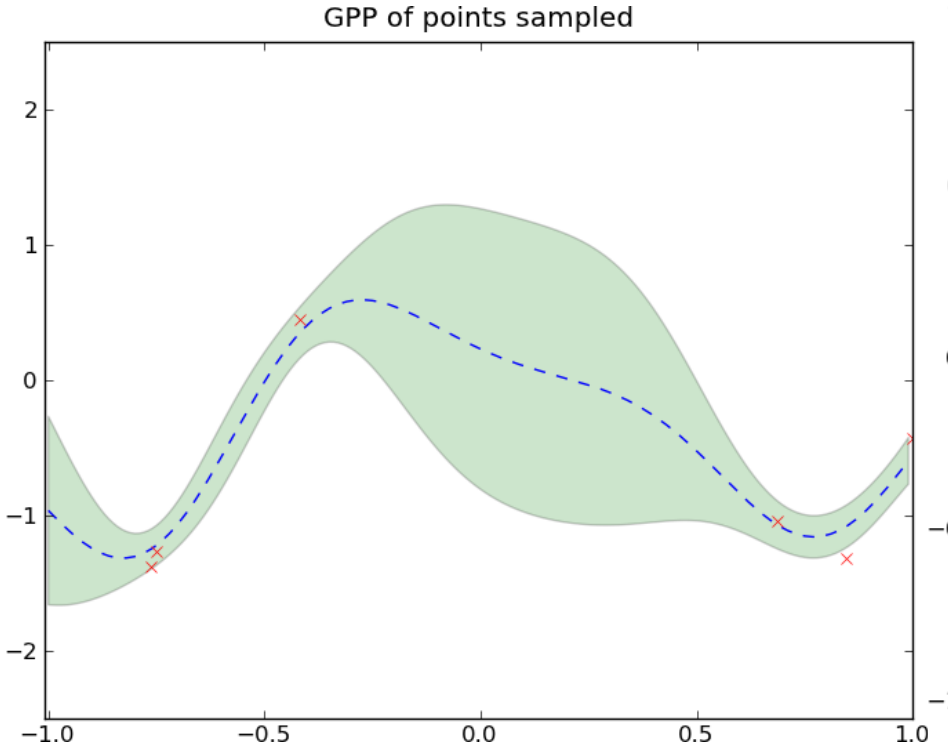


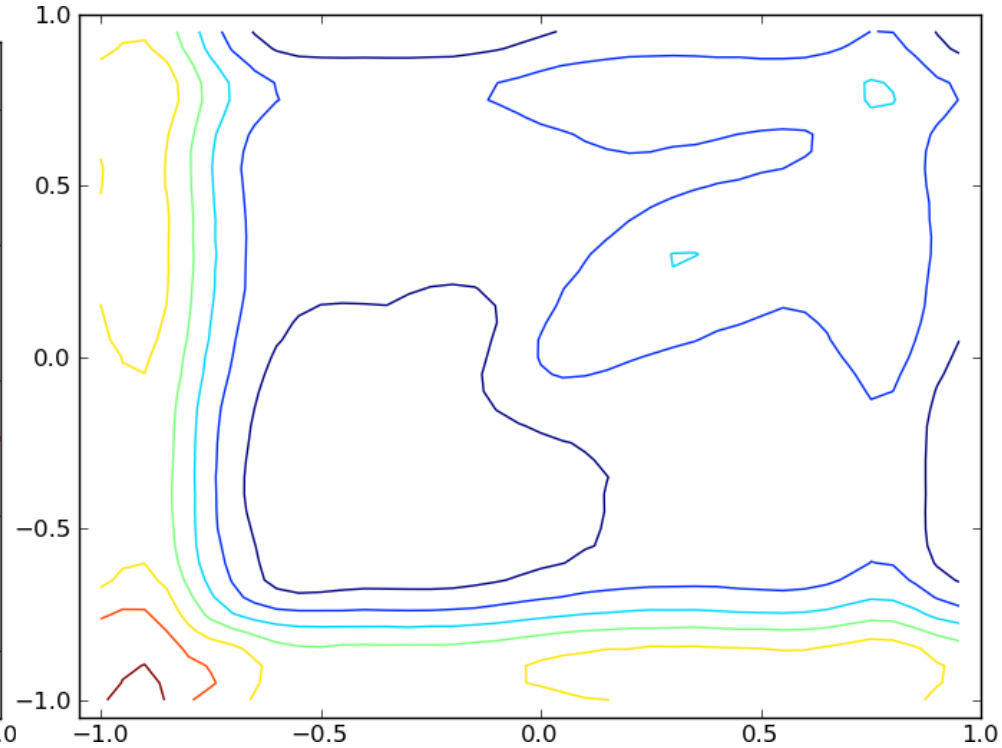
GPP evolving with information



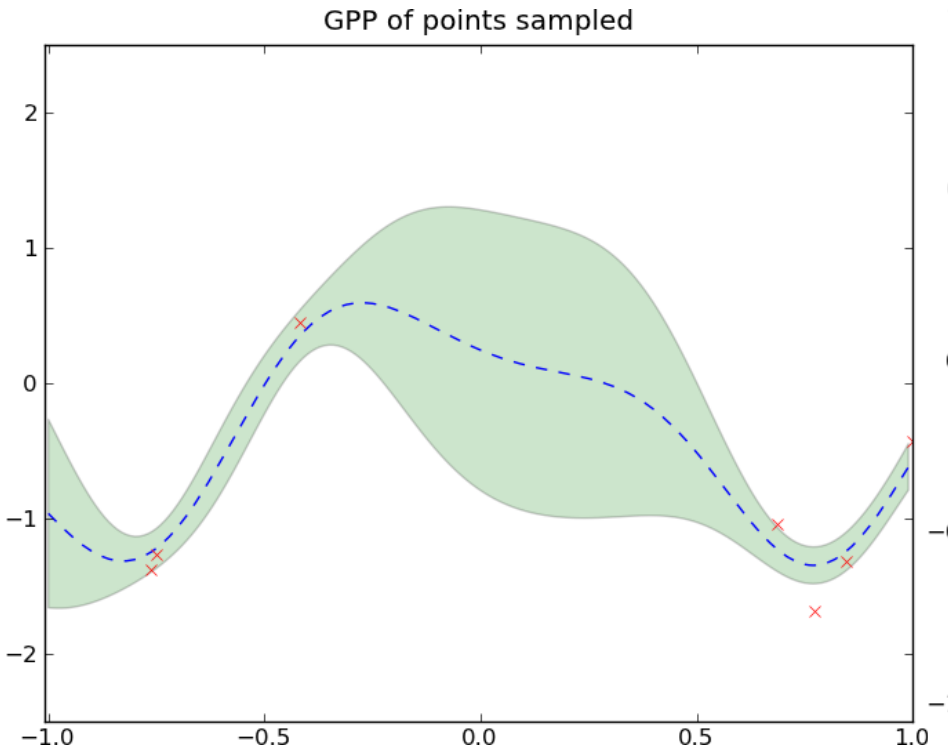


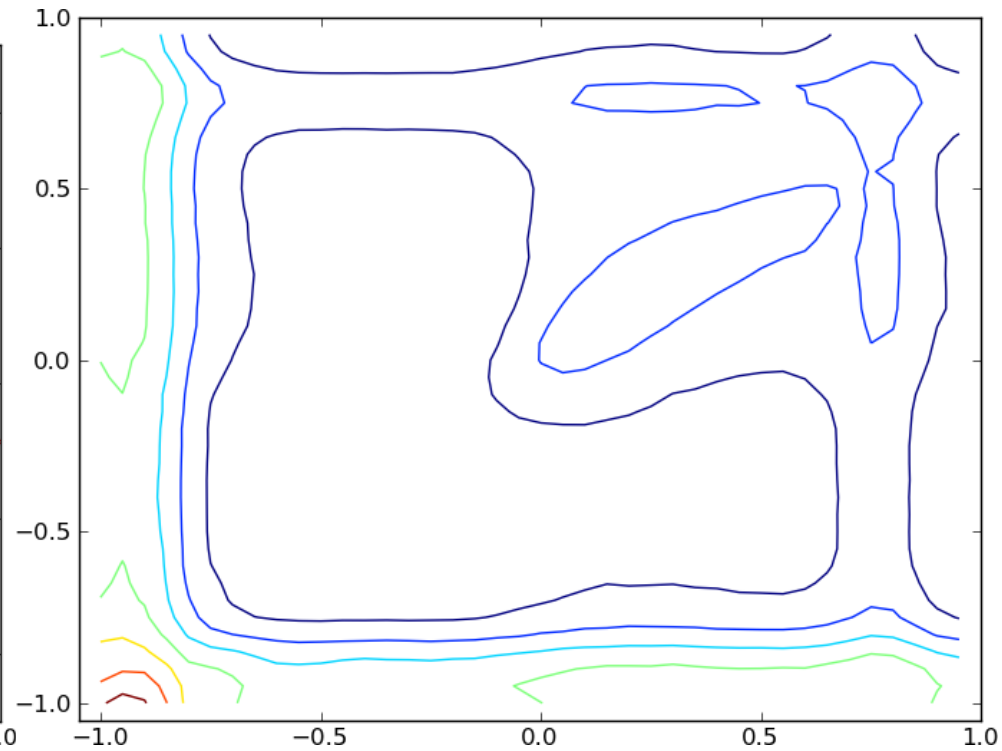
GPP evolving with information



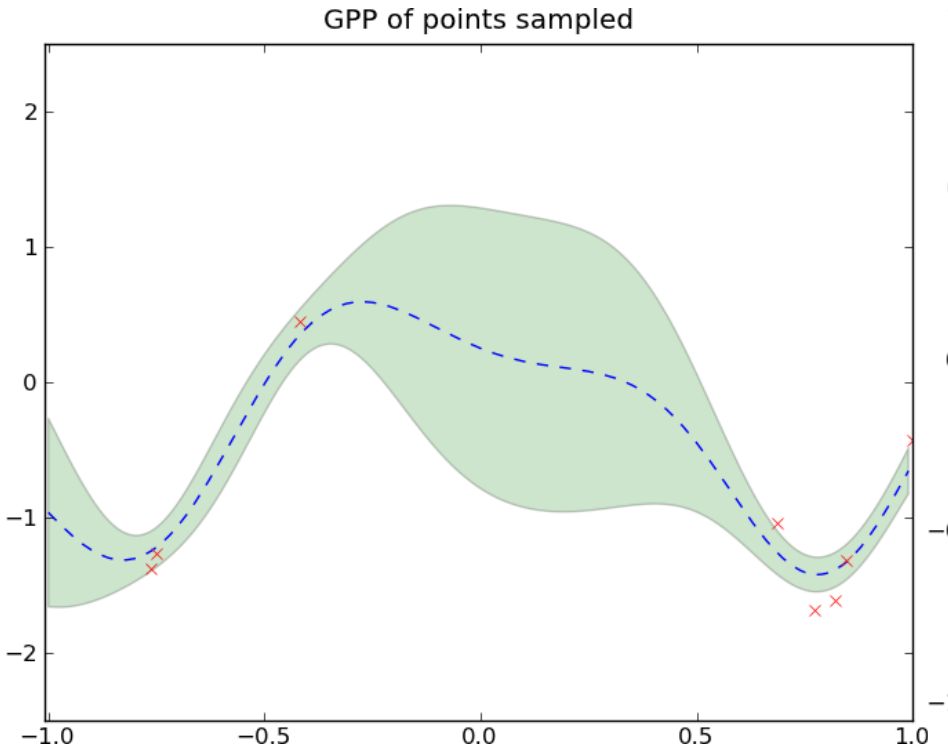


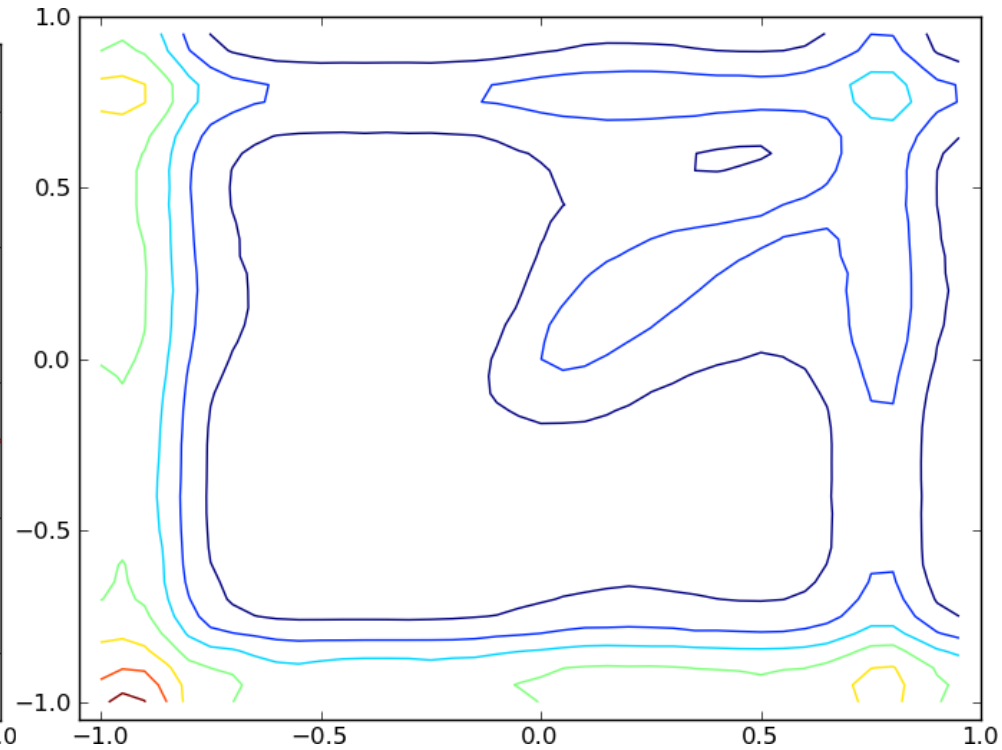
GPP evolving with information



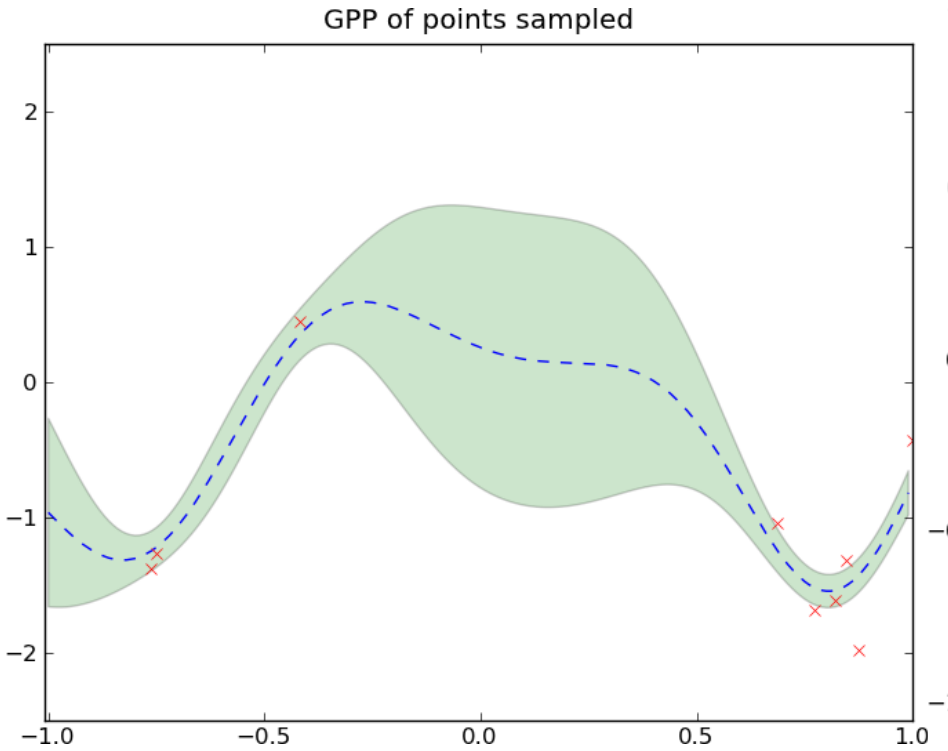


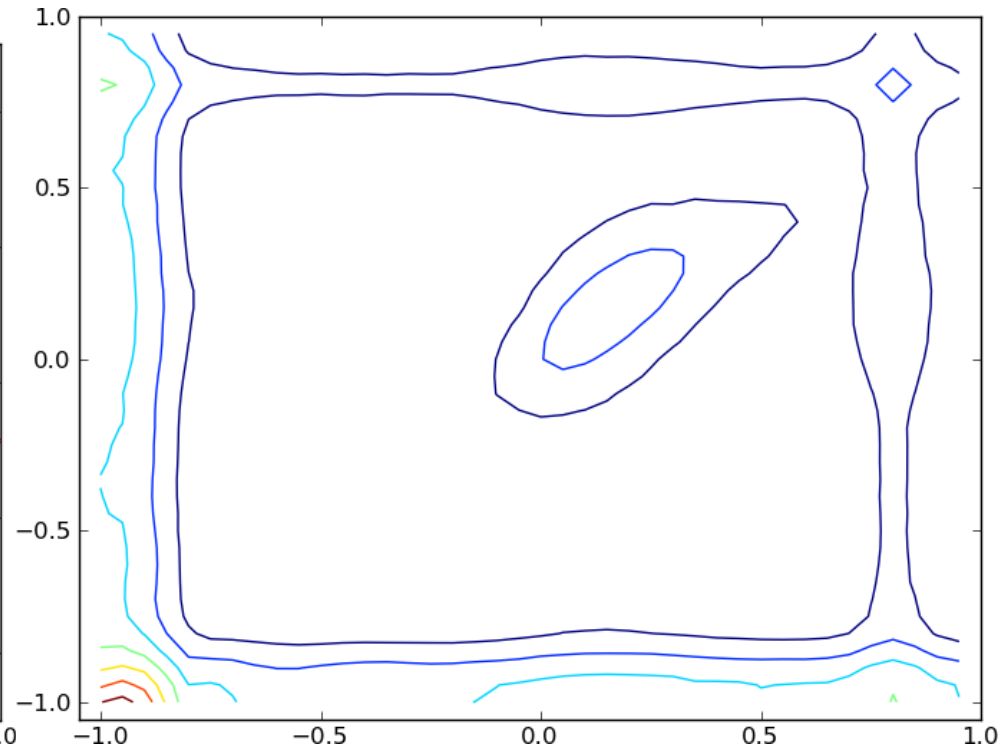
GPP evolving with information



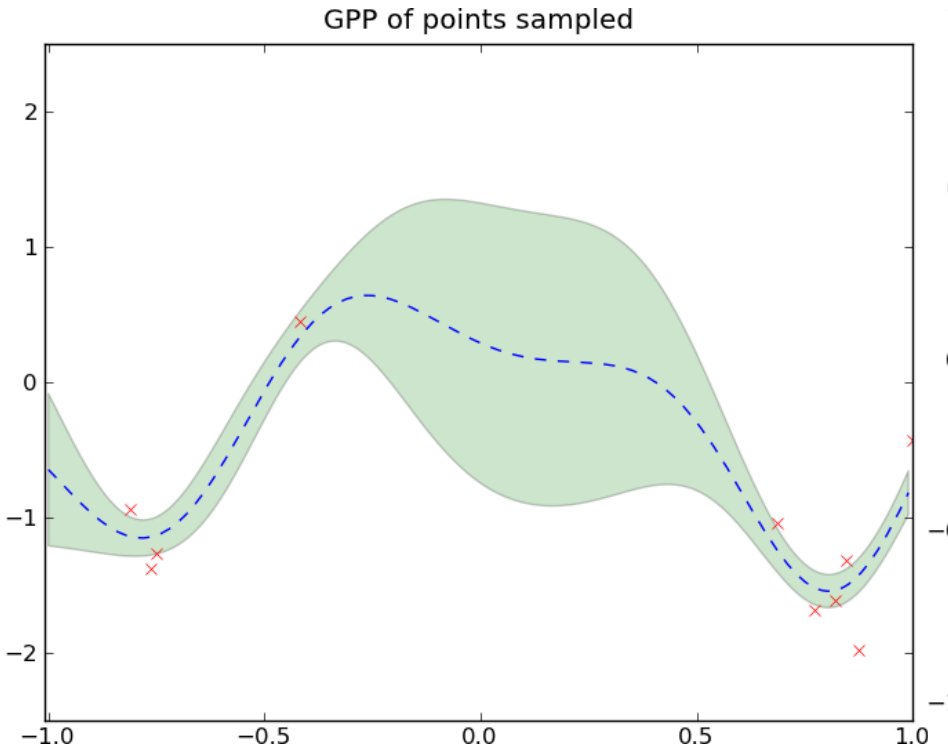


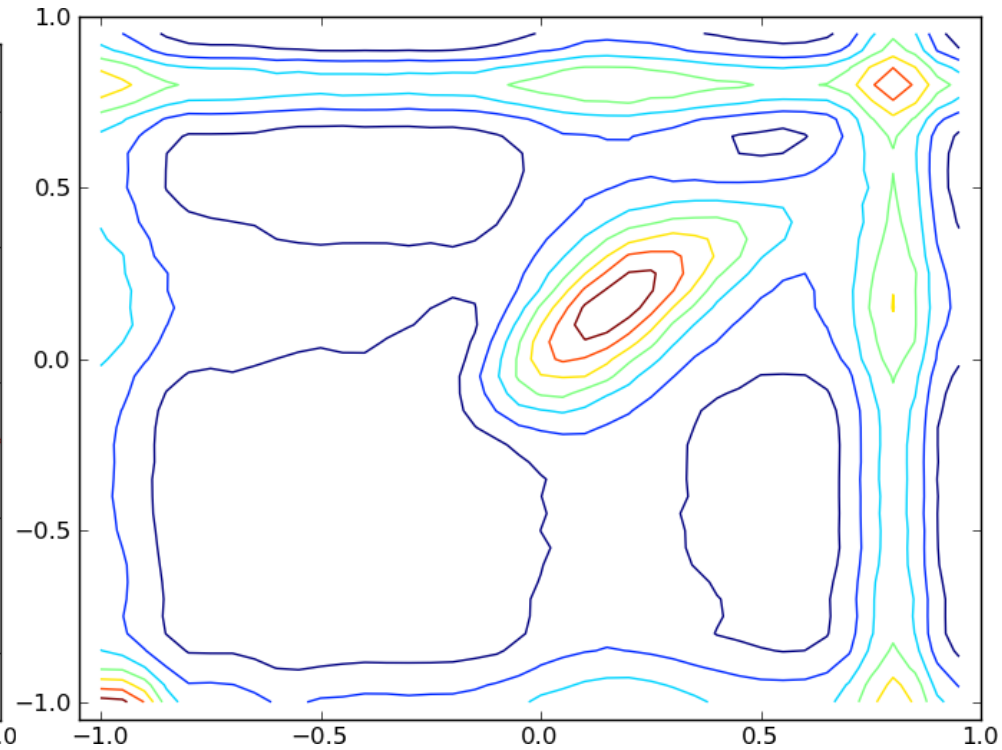
GPP evolving with information



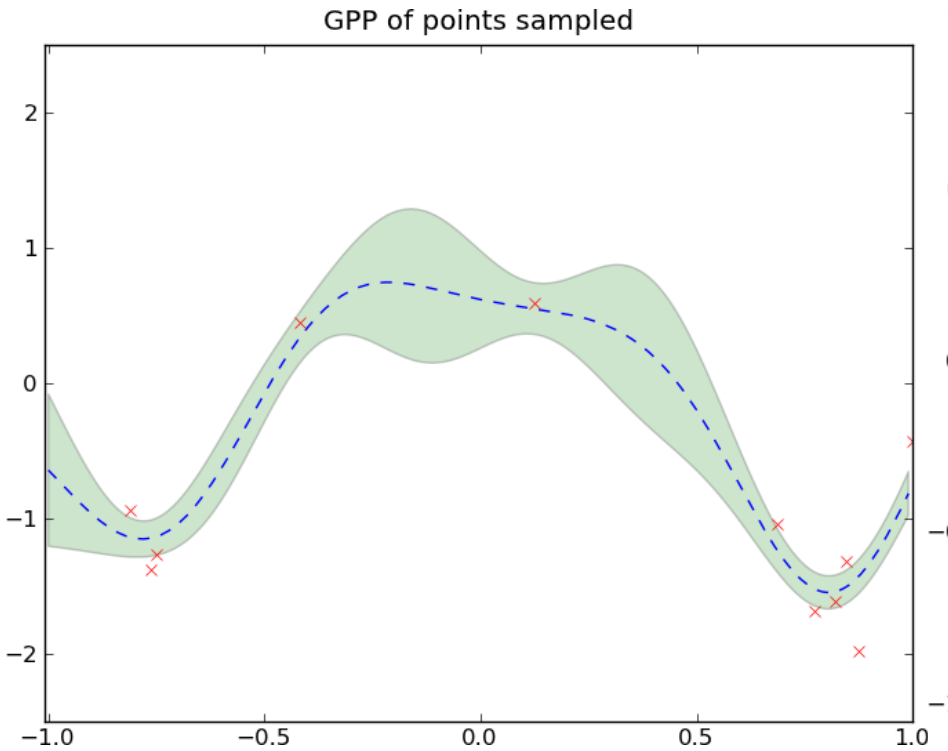


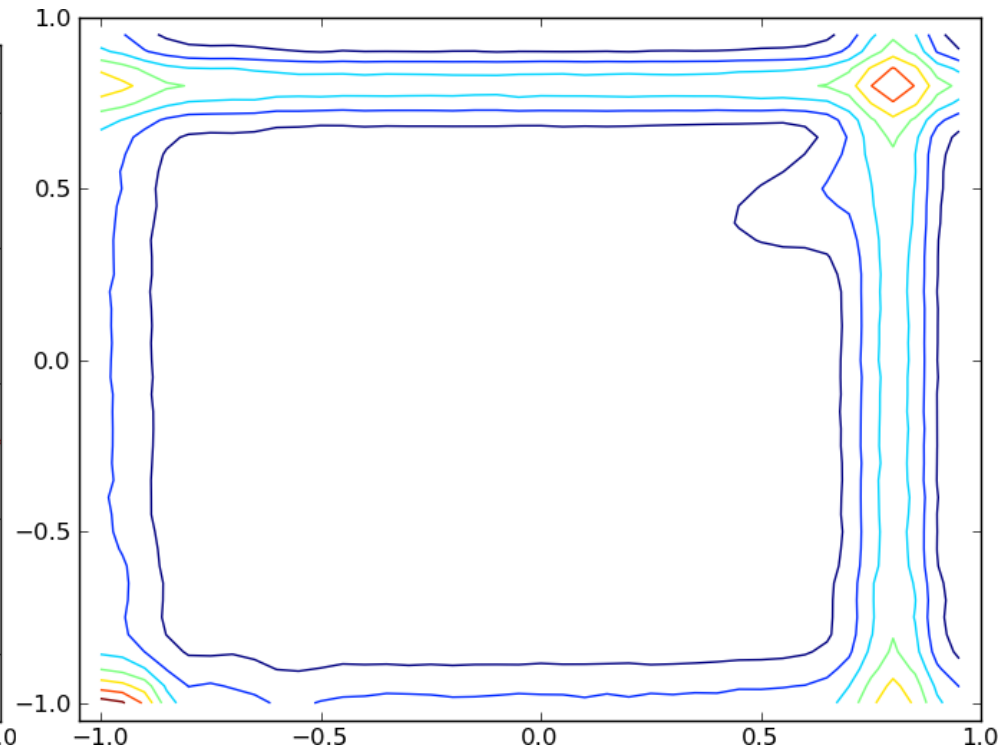
GPP evolving with information





GPP evolving with information





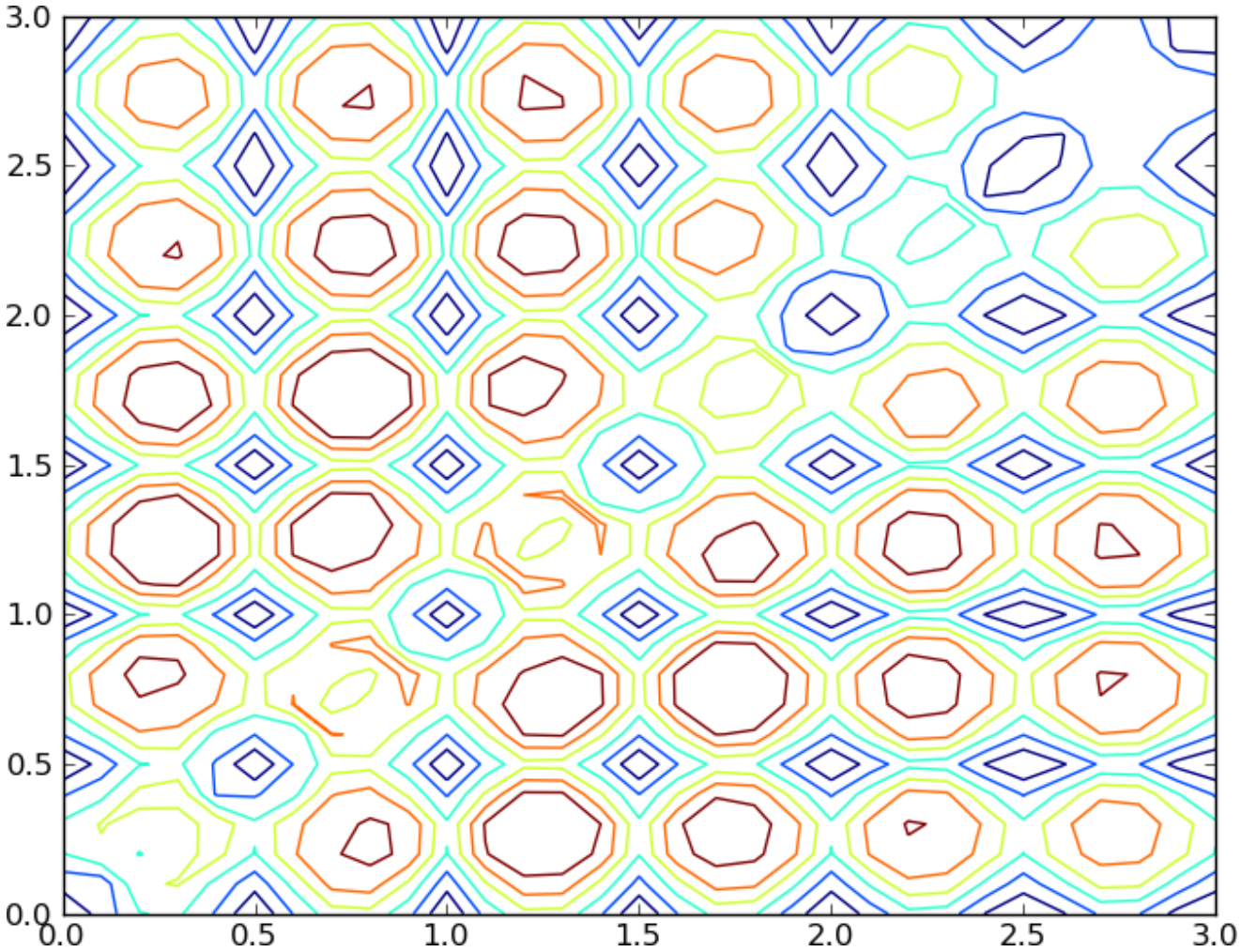
Expected Improvement 2-D

To optimize $\mathrm{EI}(\vec{x})$, we calculate stochastic gradients

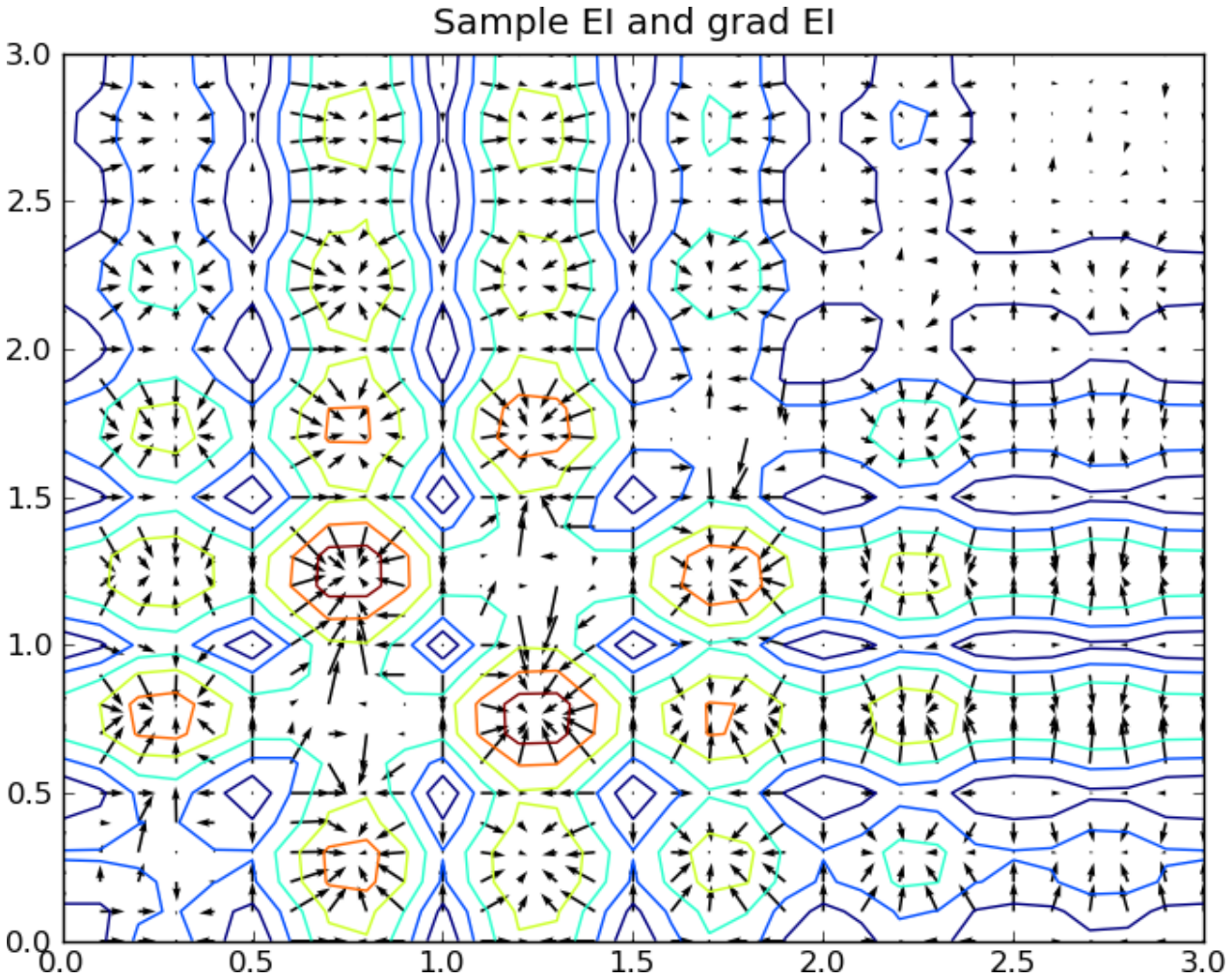
$$g(\vec{x}) = \bigtriangledown \mathrm{EI}(\vec{x})$$

- Use Infinitesimal Perturbation Analysis (IPA, Fu 1995)
- Exchange the gradient and the expectation
  - get an unbiased estimator of $g(\mathbf{x}_\star)$
- use for multistart gradient descent

Expected Improvement 2-D



Stochastic Gradient of the Expected Improvement



The Algorithm

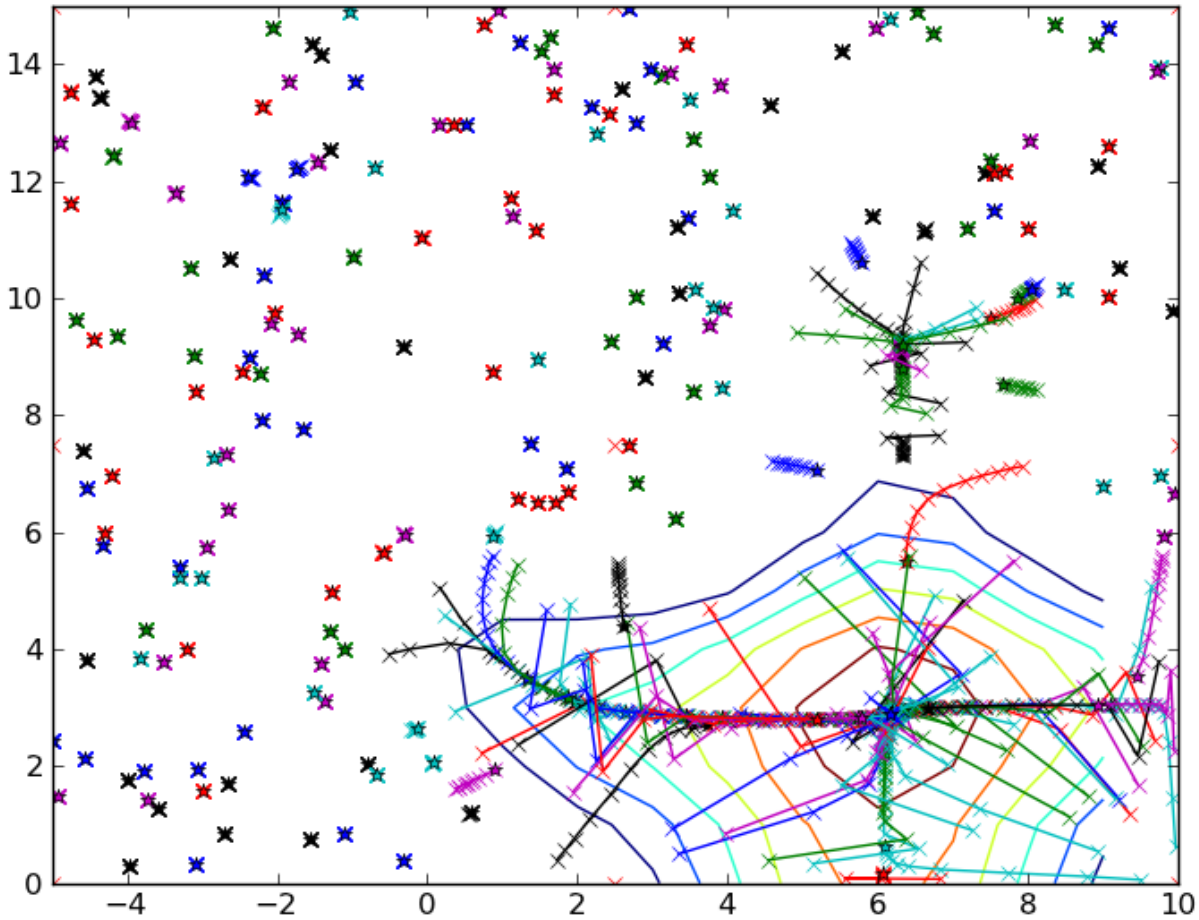
The Algorithm

1. Choose the set of points with the highest EI
2. As nodes return values search for new best EI along the space constrained by running simulations

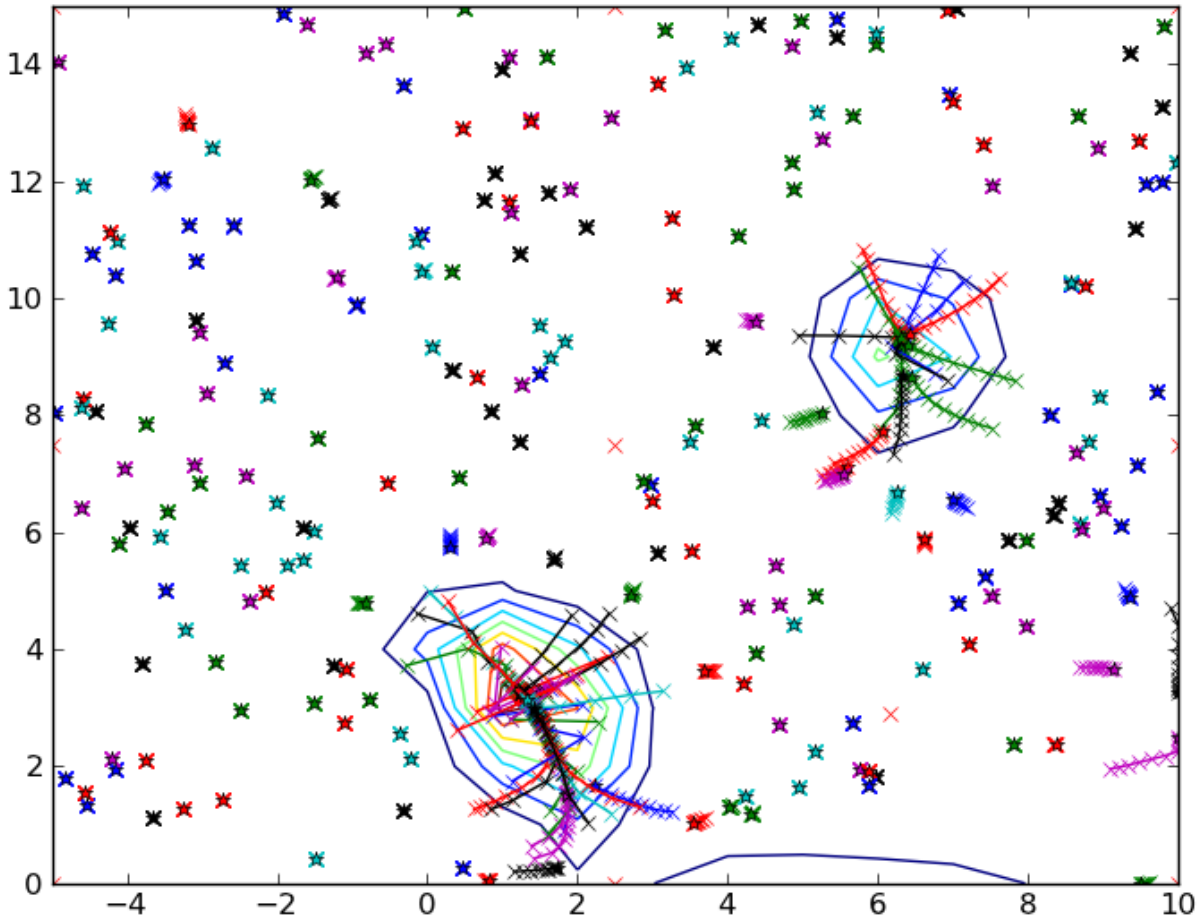
$$\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)} + \frac{a}{t^\gamma} \nabla_{\vec{x}_i} \mathrm{EI}\left(\mathbf{x}_\star{}^{(t)} | \vec{X}\right)$$

$$\left| \vec{x}_i^{(t+1)} - \vec{x}_i^{(t)} \right| < \epsilon$$

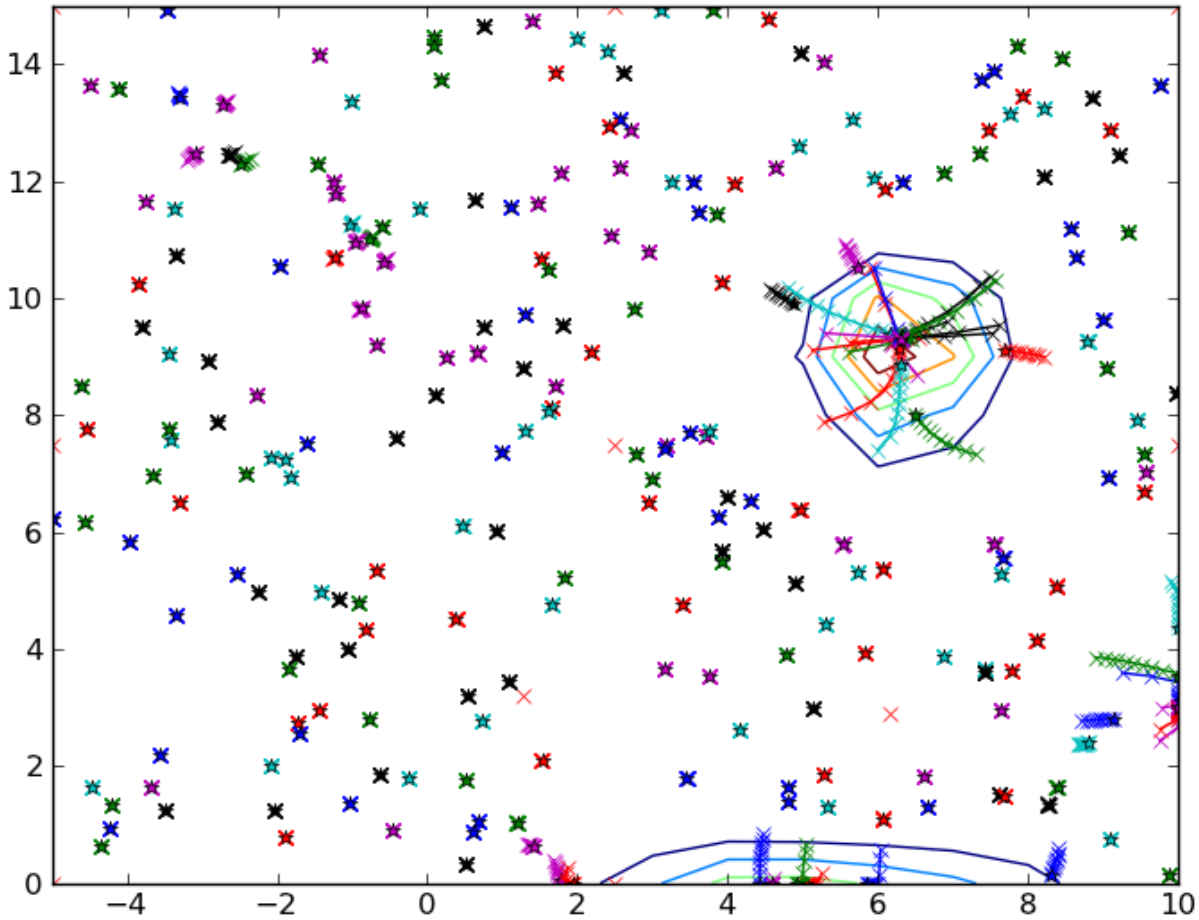
Multistart paths (1/5)



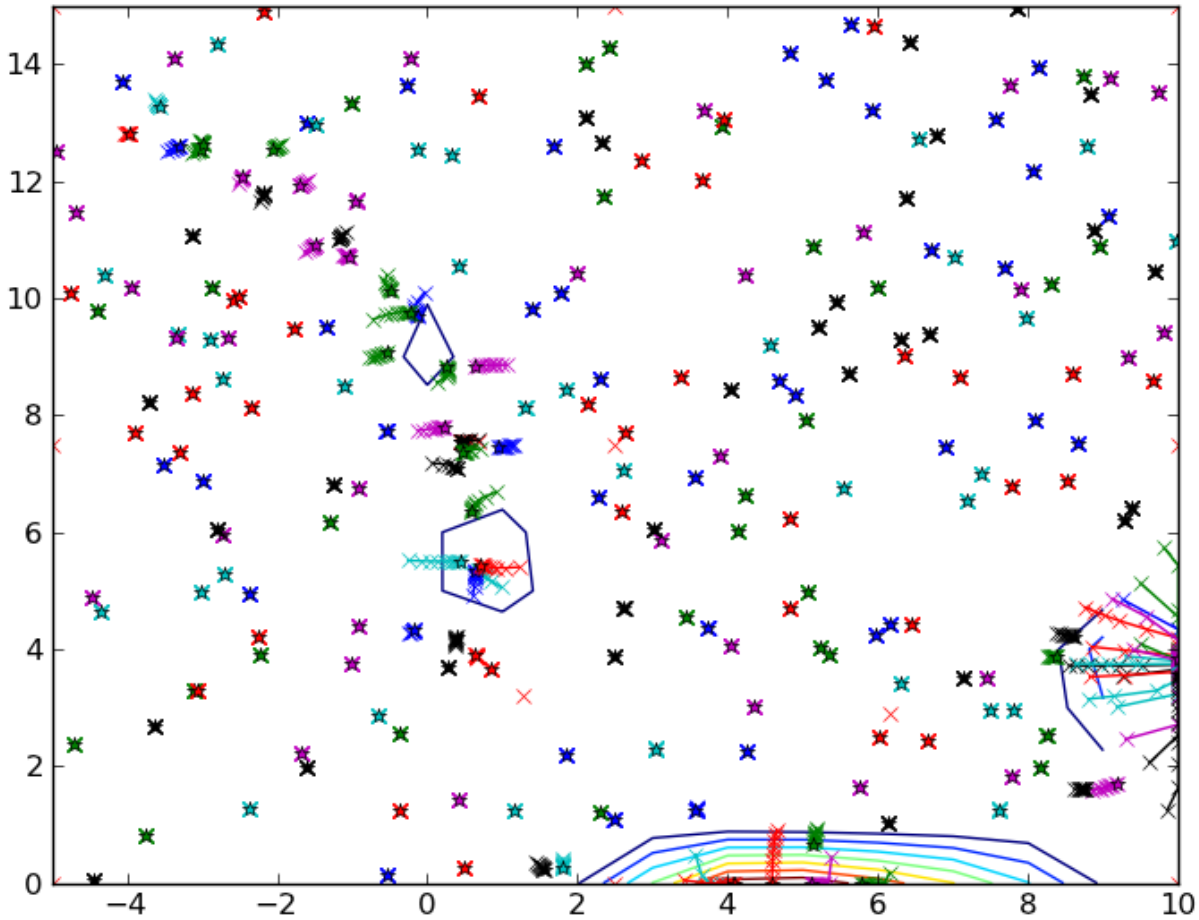
Multistart paths (2/5)



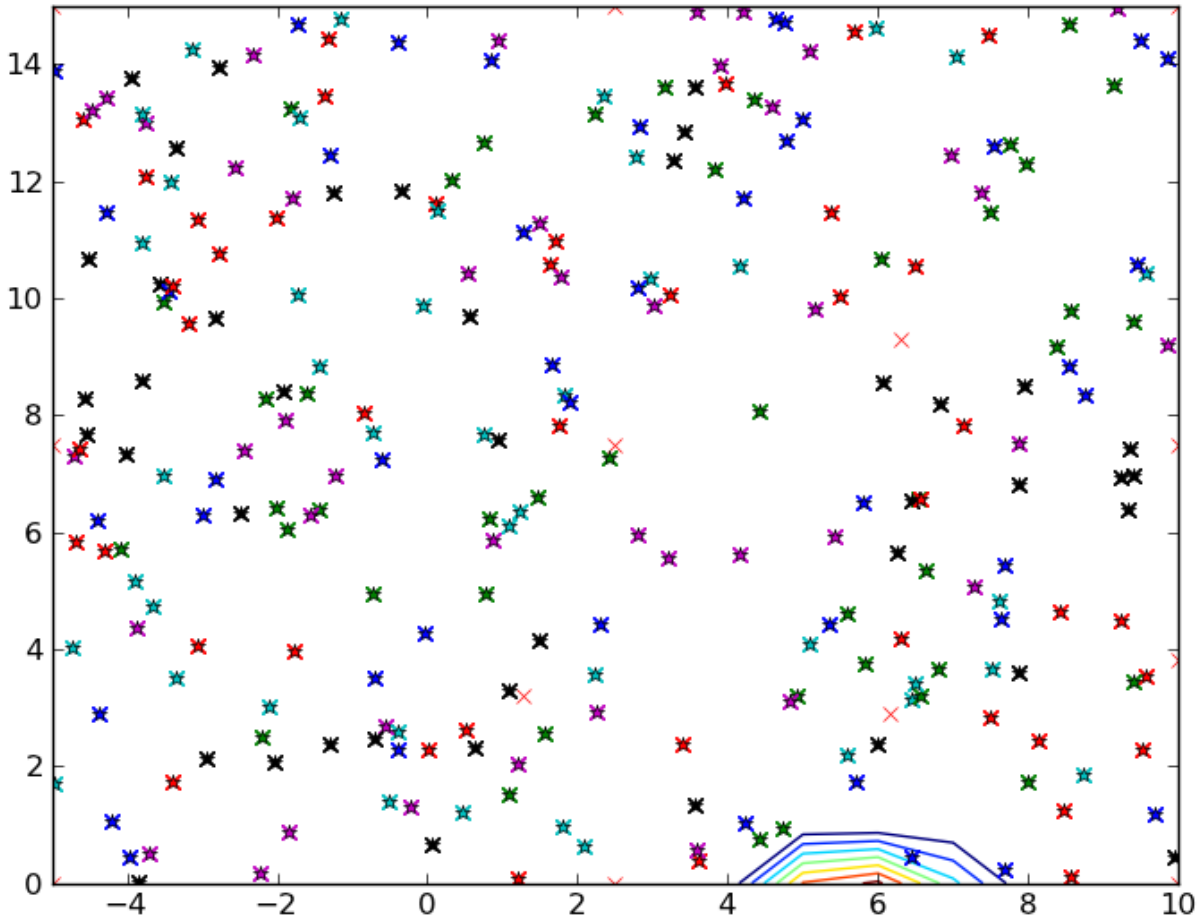
Multistart paths (3/5)



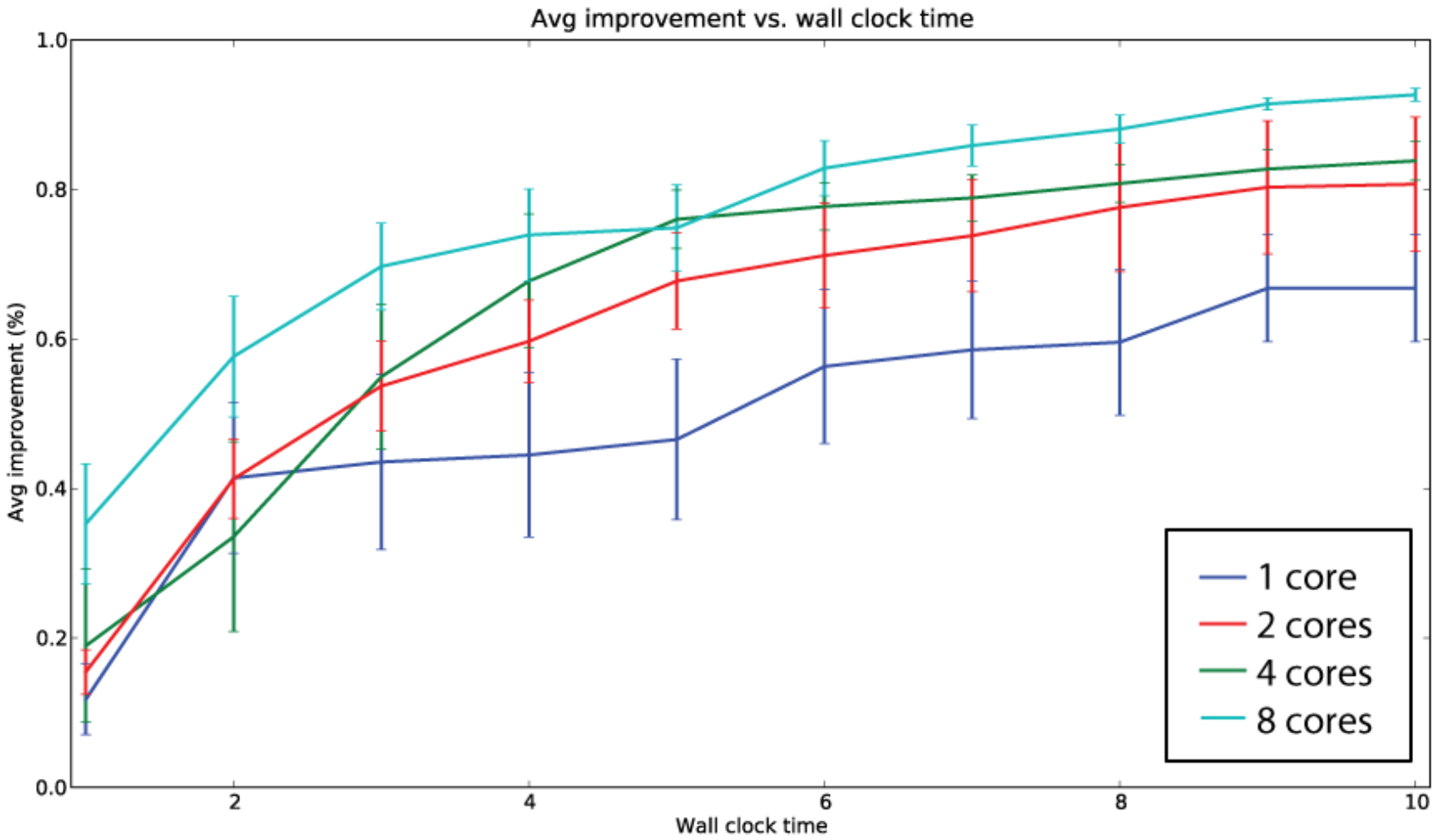
Multistart paths (4/5)



Multistart paths (5/5)



Speedup


Avg improvement vs. wall clock time
Avg improvement (%)
Wall clock time
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
1 core
2 cores
4 cores
8 cores

EPI Conclusion

Developed an algorithm and software package that

- Finds next best **set** of points to sample
- Allows for efficient use of parallel resources
- Is faster than serial methods
- Is general: global, black-box, derivative-free, non-convex
- Is open source and easy use python (+CUDA) and/or C

Velvetrope (Part III of thesis)

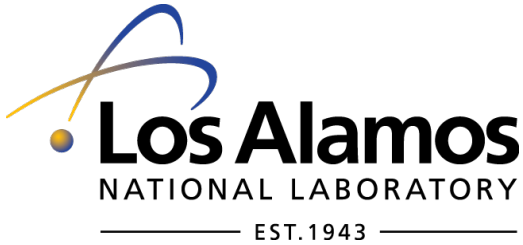
Developed an algorithm and software package that

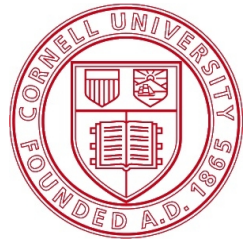
- Finds motifs in sequences using a novel statistical model
- Is faster than the current methods (GPU implementation, CUDA)
- Can handle cases other methods cannot
- Natively allows for quick analysis of aggregate data
- Is open source and easy to fit into a pipeline

Project status

ALE	Software:	github.com/sc932/ALE
		(UoI/CNSA open source license)
	Presented:	SC11, SIAM CSE11, DOE CSGF11
		INFORMS10, JGI Seminar
	Paper:	submitted to Bioinformatics
EPI	Software:	github.com/sc932/EPI
		(UoI/CNSA open source license)
	Presented:	SC12, DOE CSGF12
	Paper:	in preperation
Velvetrope	Software:	github.com/sc932/Velvetrope
		(GPL v2 open source license)
	Presented:	SC10, DOE CSGF10
	Paper:	profiled in DIEXIS

Acknowledgements





- DOE Computational Science Graduate Fellowship
- Advisor: Peter Frazier (Cornell ORIE)
- DOE Joint Genome Institute (JGI) Genome Analysis Group
  - Zhong Wang (Mentor)
  - Rob Egan (Co-Mentor)
- Los Alamos National Laboratory (LANL) Metagenomics Group
  - Nick Hengartner (Mentor)
  - Joel Berendzen (Co-Mentor)
- Committee
  - Steve Strogatz (Math)
  - Bart Selman (CS)
  - Jim Renegar (Math proxy)

DOE Grants
DE-FG02-97ER25308
DE-AC02-05CH11231