ParaMonte

Plain Powerful **Para**llel
Monte Carlo Fortran Library

for all programming languages: C, C++, Fortran, MATLAB, Python, …

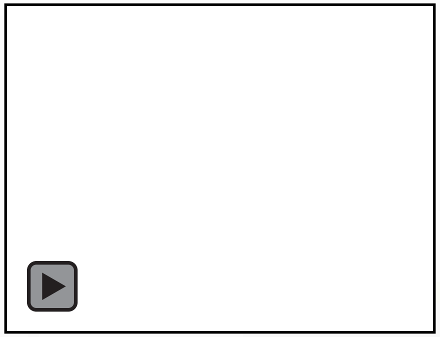
Amir Shahmoradi, Fatemeh Bagheri,
Shashank Kumbhare, Joshua Osborne

Department of Physics / College of Science
Data Science Program / College of Science
The University of Texas
Arlington, Texas

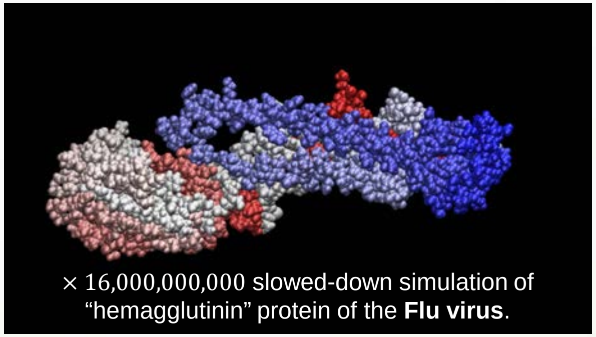
1st International Fortran Conference, July 2020

The Computational and Data Science Lab @ Department of Physics / College of Science / UTA

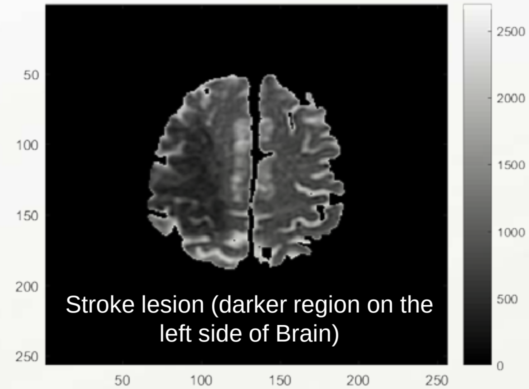
Join us @ **cdslab**.org

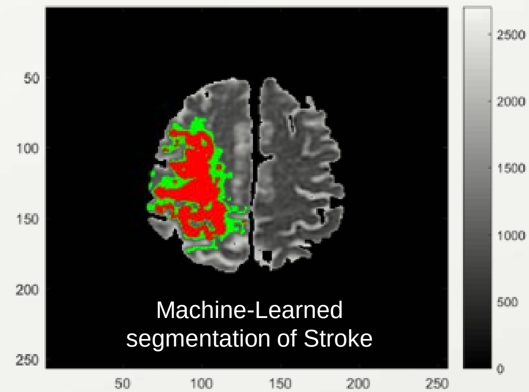


Physics of Gamma-Ray Bursts
The most powerful explosions in the universe



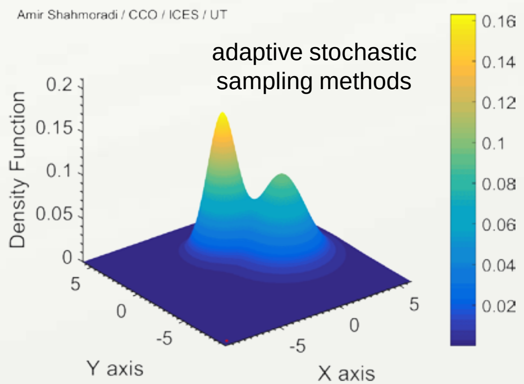
Bioinformatics / Biophysics





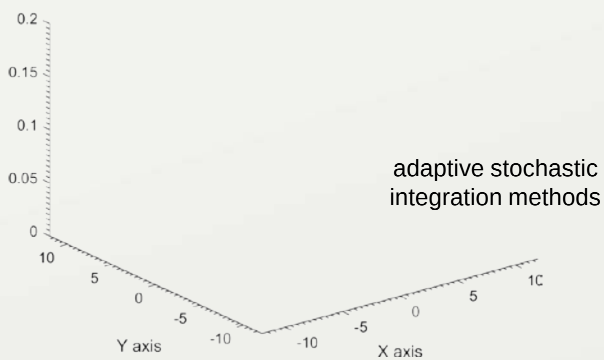
Biomedical Data Science

Open-source software development:
Machine Learning and Monte Carlo Methods



ParaMonte
Plain Powerful Parallel Monte Carlo Library

cdslab.org/pm

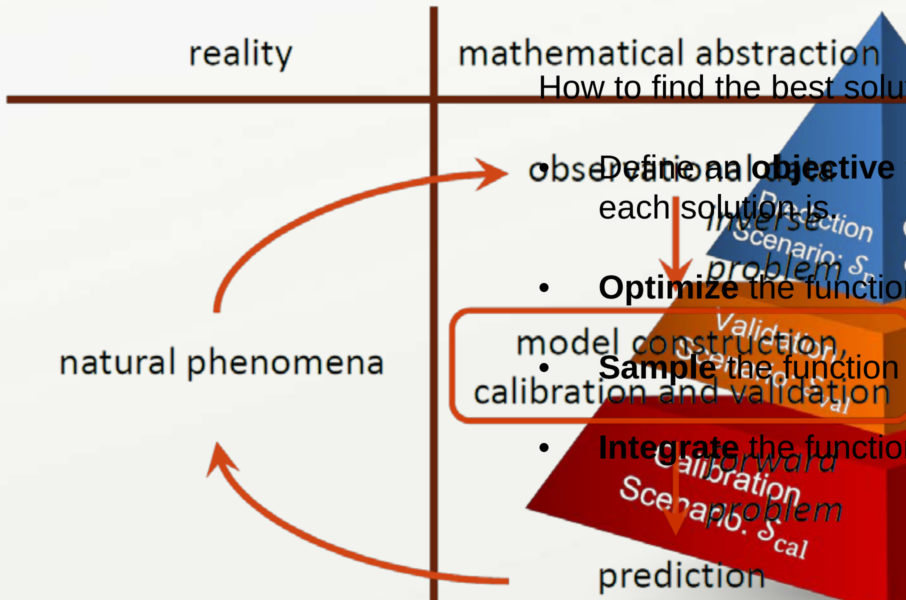


Traffic Engineering

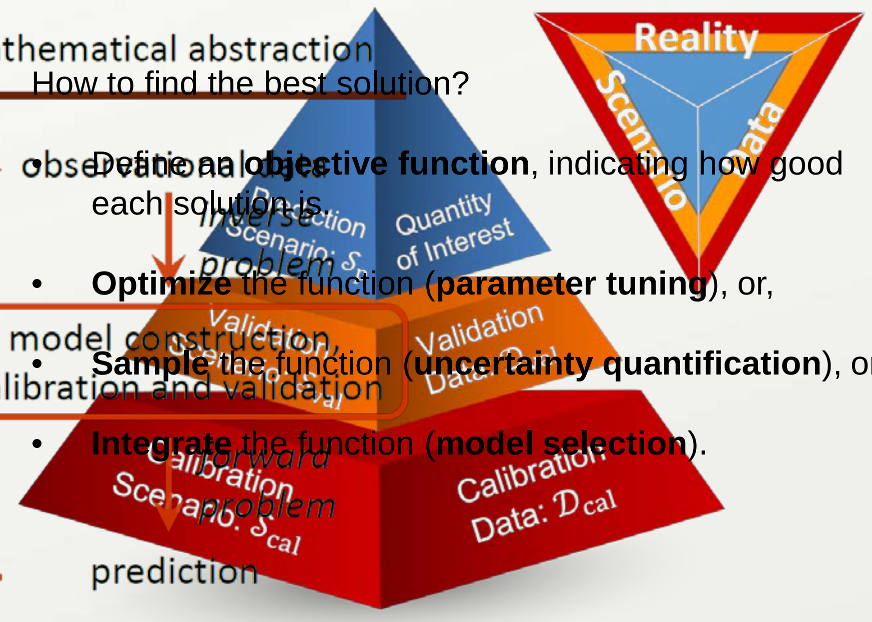
The two classical pillars of science: Experiment and Theory

How do we make a scientific inference?

A very elementary depiction of the scientific method



The prediction pyramid



How to find the best solution?

- Define an **objective function**, indicating how good each solution is.
- **Optimize** the function (**parameter tuning**), or,
- **Sample** the function (**uncertainty quantification**), or,
- **Integrate** the function (**model selection**).

(Shahmoradi 2017)

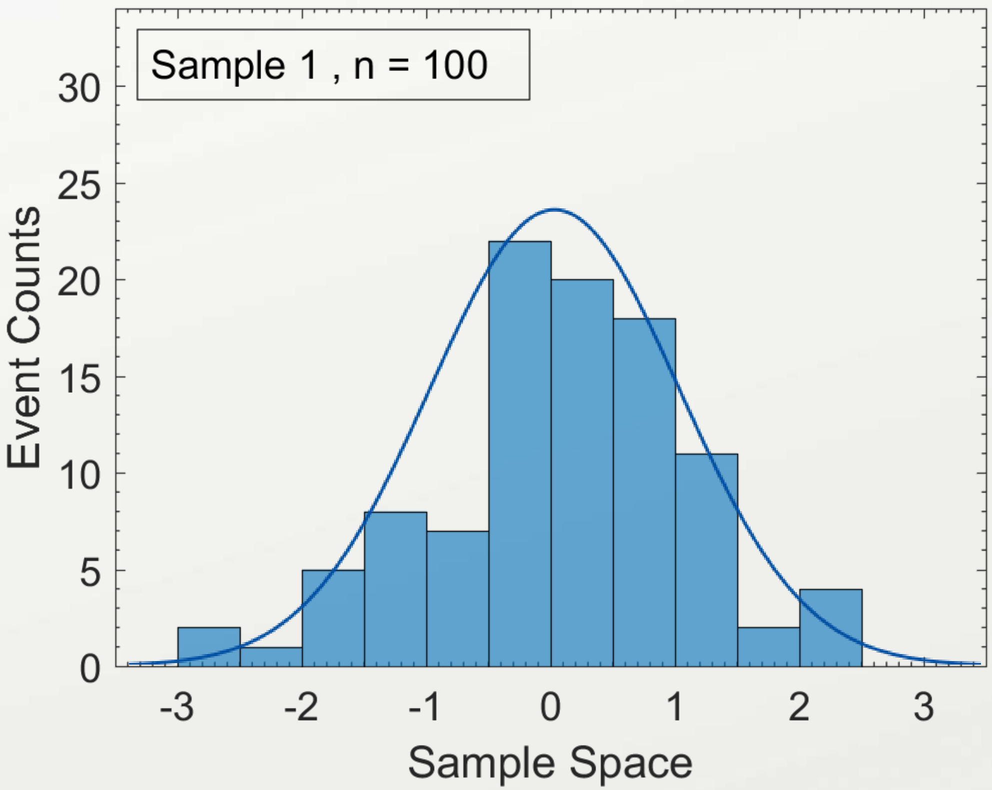
A scientific inference toy problem

Hypothesis: Data comes from a Normal distribution

$$\mathcal{P} = \left\{ P(x|\boldsymbol{\theta} = \{\mu, \sigma\}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \quad : \quad \mu \in \mathbb{R} \, , \; \sigma \in \mathbb{R}^+ \right\}$$

Objective Function:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \equiv$$
$$\pi(\mathcal{D}|\mu, \sigma) =$$
$$\pi(x_1, \dots, x_n|\mu, \sigma) =$$
$$\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} =$$
$$\boxed{\frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}}$$



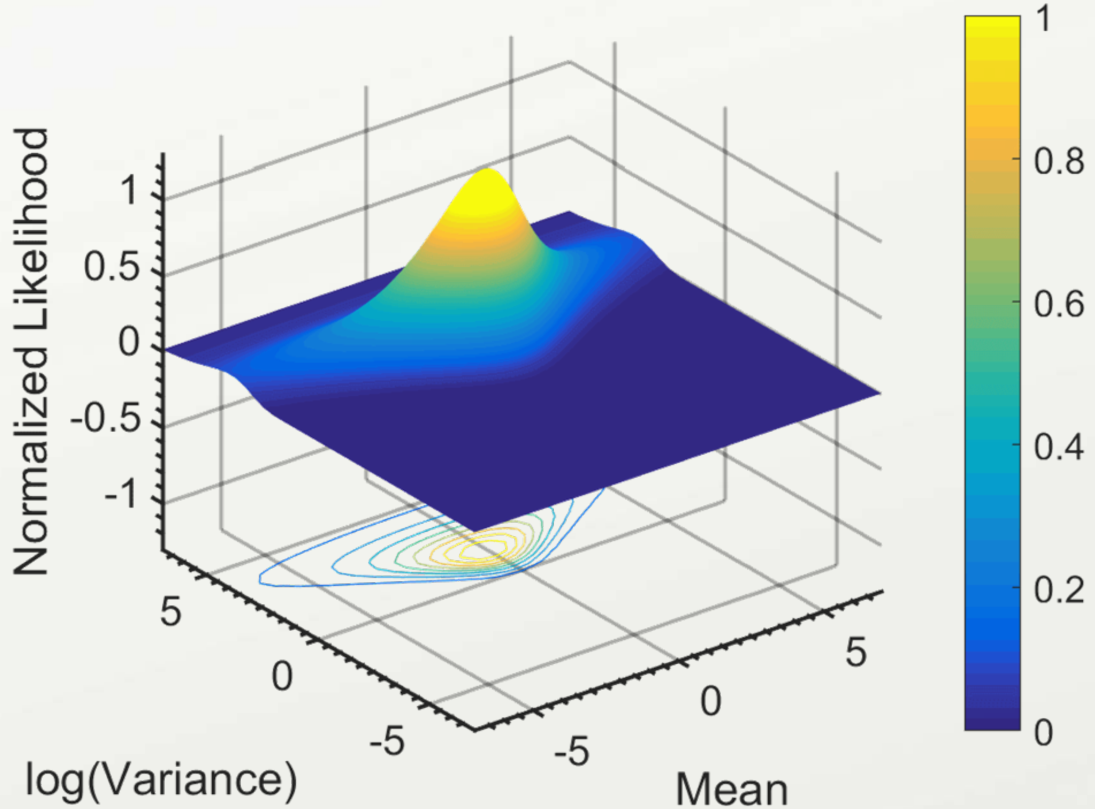
A scientific inference toy problem

Hypothesis: Data comes from a Normal distribution

$$\mathcal{P} = \left\{ \quad P(x|\boldsymbol{\theta} = \{\mu, \sigma\}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(- \frac{(x-\mu)^2}{2\sigma^2} \right) \quad : \quad \mu \in \mathbb{R} \, , \; \sigma \in \mathbb{R}^+ \right\}$$

Objective Function:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \equiv$$
$$\pi(\mathcal{D}|\mu,\sigma) =$$
$$\pi(x_1, \ldots, x_n|\mu,\sigma) =$$
$$\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} =$$
$$\boxed{\frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}}$$



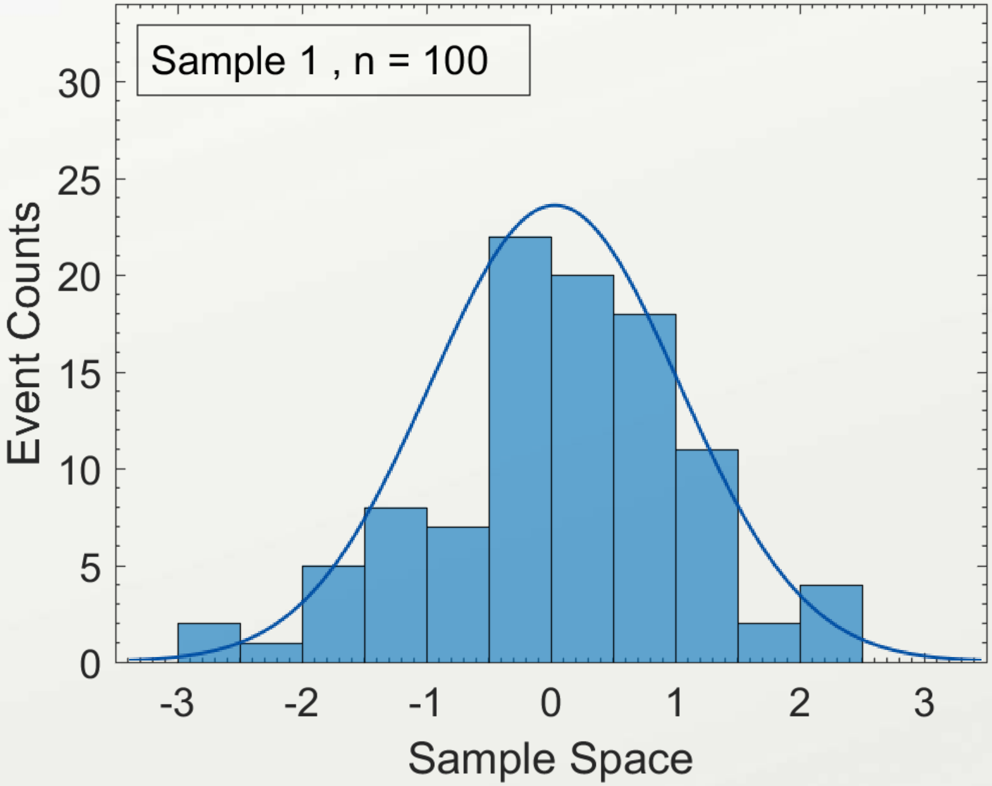
A scientific inference toy problem

Hypothesis: Data comes from a Normal distribution

$$\mathcal{P} = \left\{ P(x|\boldsymbol{\theta} = \{\mu, \sigma\}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \; : \; \mu \in \mathbb{R}\,,\ \sigma \in \mathbb{R}^+ \right\}$$

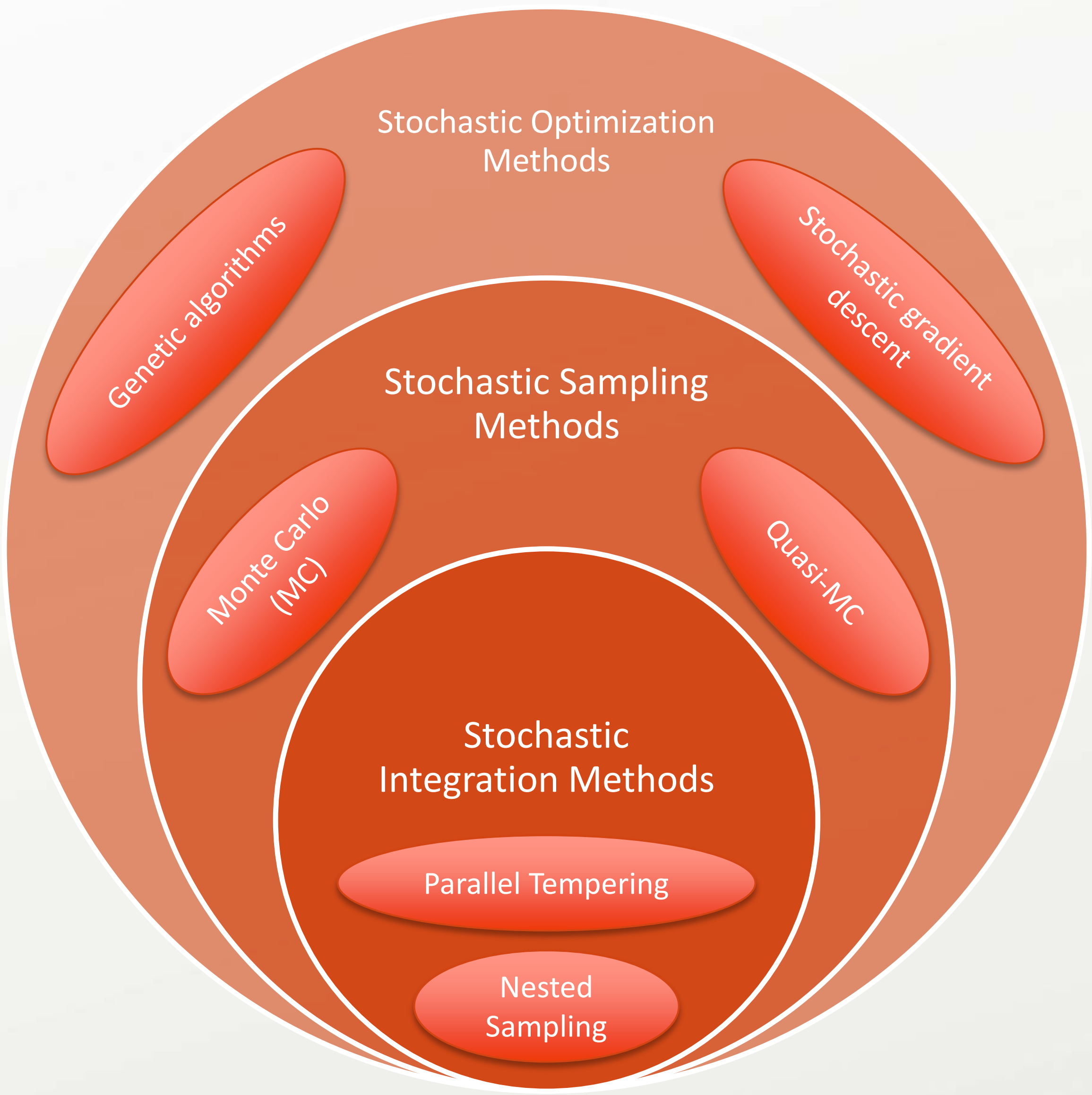
Objective Function:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \equiv \pi(\mathcal{D}|\mu, \sigma) = \pi(x_1, \ldots, x_n|\mu, \sigma) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} = \boxed{\frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}}$$

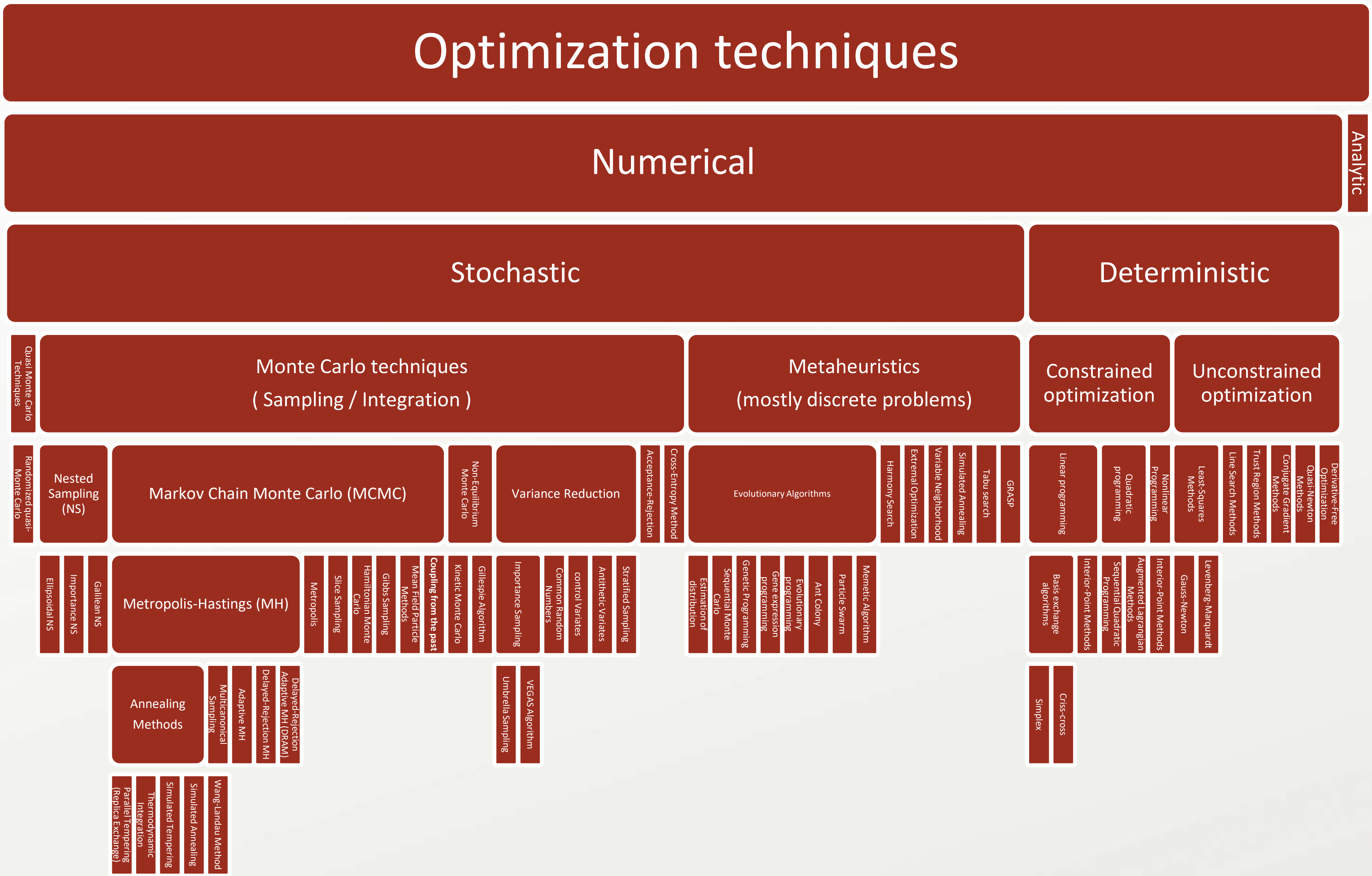


Hierarchical diagram of optimization, sampling, and integration methods

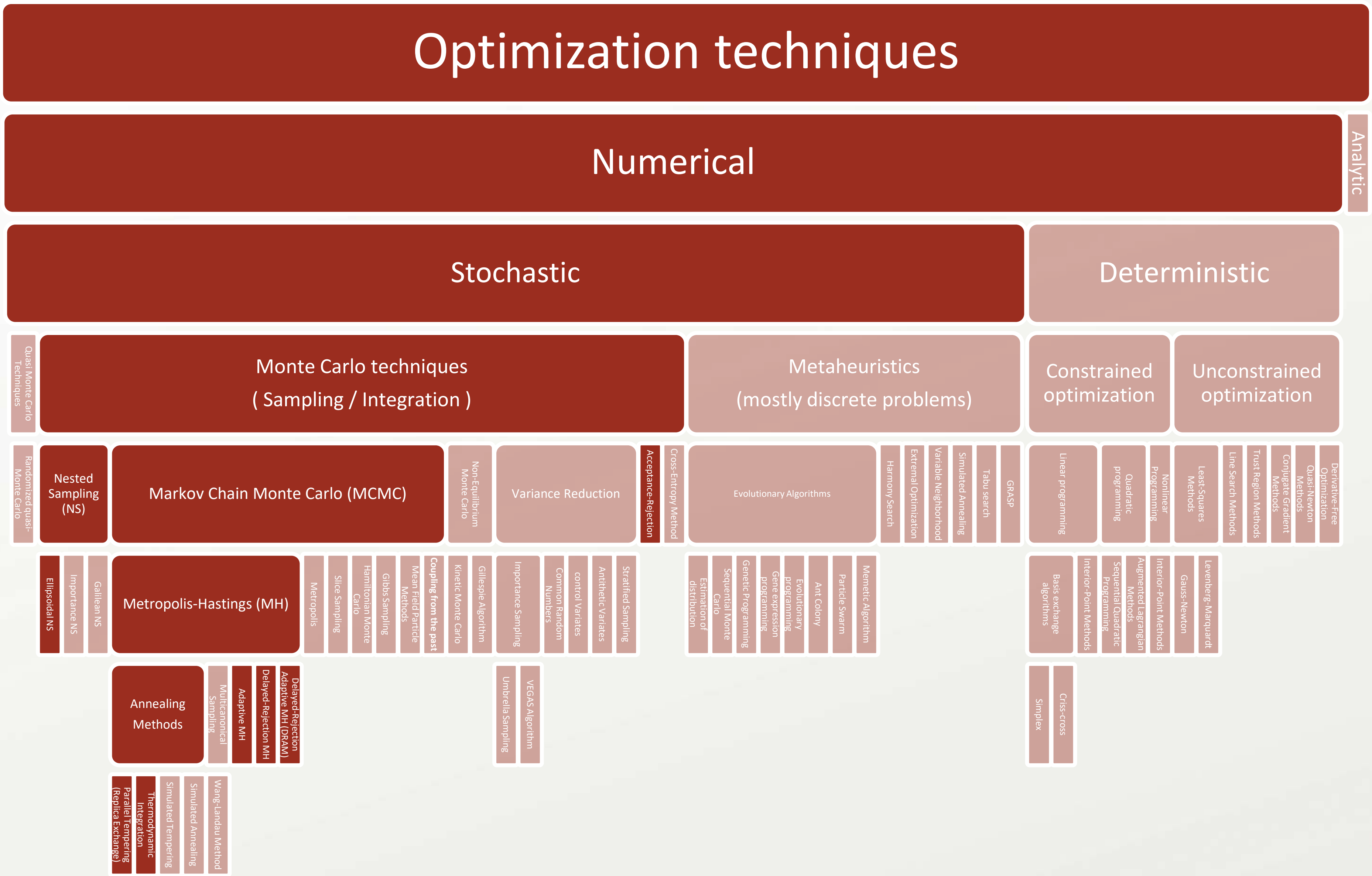
The onion-like structure of stochastic optimization algorithms



Hierarchical diagram of optimization, sampling, and integration methods



Hierarchical diagram of optimization, sampling, and integration methods



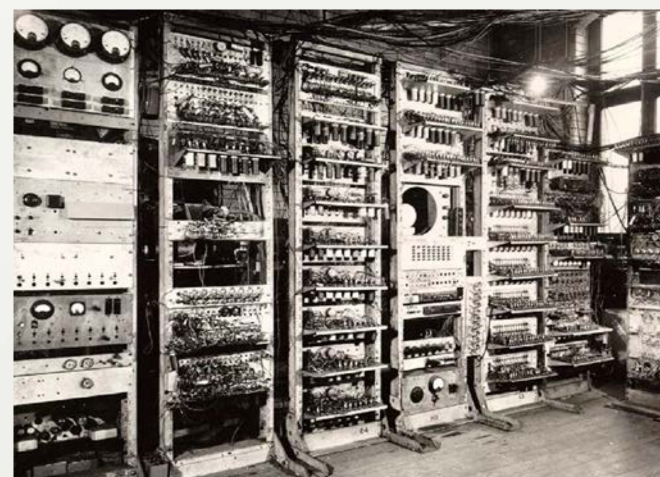
Monte Carlo methods: A brief history

Enrico Fermi
Physicist (1901-1954)

Stanislaw Ulam
Mathematician (1909-1984)

John von Neumann
Mathematician (1903-1957)

Fermi's Analog Computer (**FERMIAC**)



Electronic **N**umerical **I**ntegrator **A**nd **C**omputer (**ENIAC**)

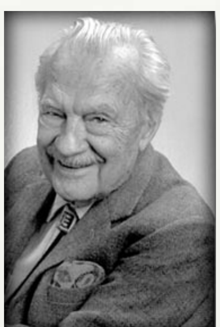
Monte Carlo methods: A brief history

Enrico Fermi
Physicist (1901-1954)

Stanislaw Ulam
Mathematician (1909-1984)

John von Neumann
Mathematician (1903-1957)

Metropolis-Hastings Markov Chain Monte Carlo Technique



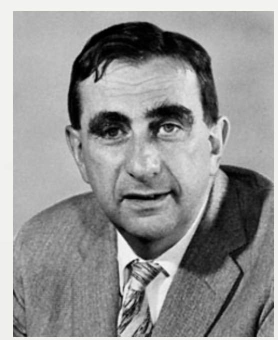
N. Metropolis
(1915 – 1999)
Physicist

?

A. Rosenbluth
(19?? – ????)
Programmer

M. Rosenbluth
(1927 – 2003)
Physicist

Augusta Teller
(19?? – ????)
Programmer(?)



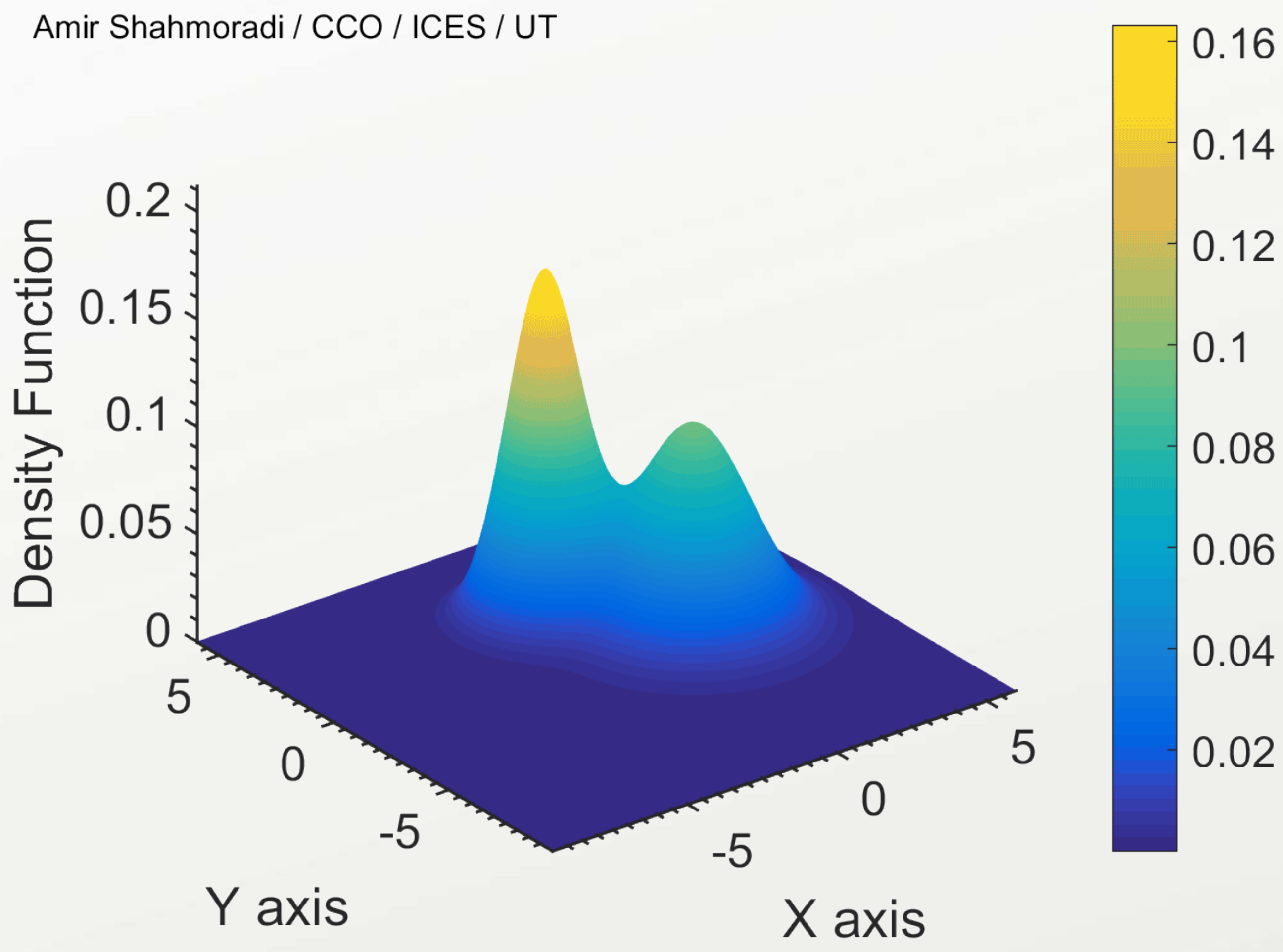
Edward Teller
(1908 – 2003)
Physicist

Wilfred Hastings
(1930 – 2016)
Statistician

The Metropolis-Hastings Algorithm

Setup a random walker such that visits every point in the domain of the objective function proportional to the height of the point.

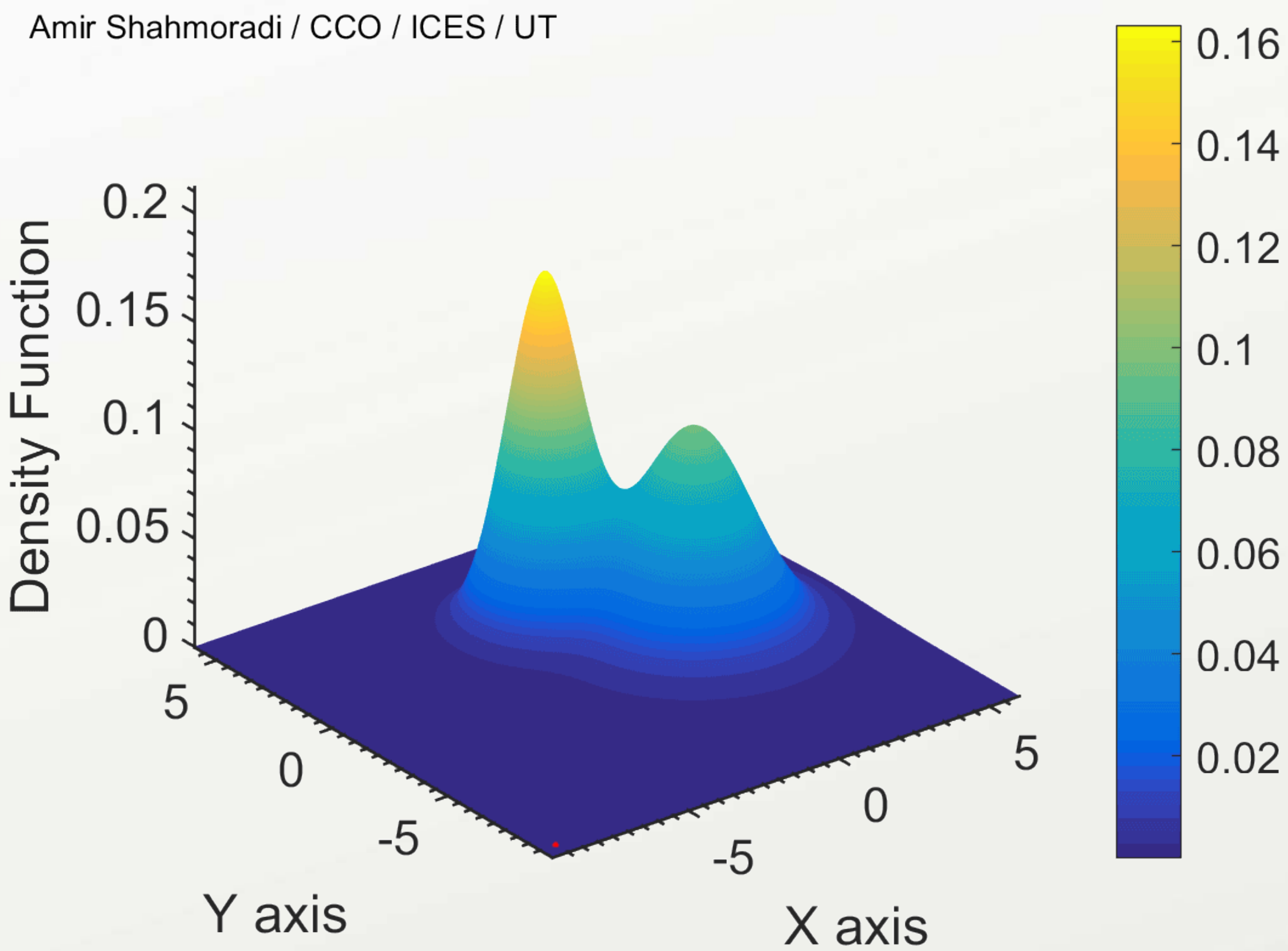
Example Metropolis random walker with **small** steps



The Metropolis-Hastings Algorithm

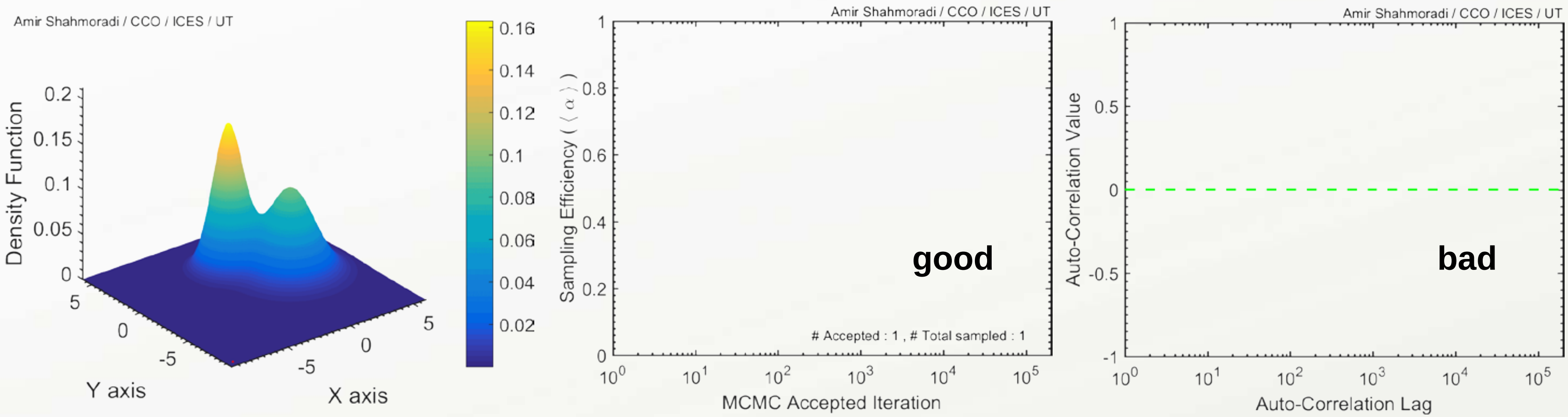
Setup a random walker such that visits every point in the domain of the objective function proportional to the height of the point.

Example Metropolis random walker with **large** steps

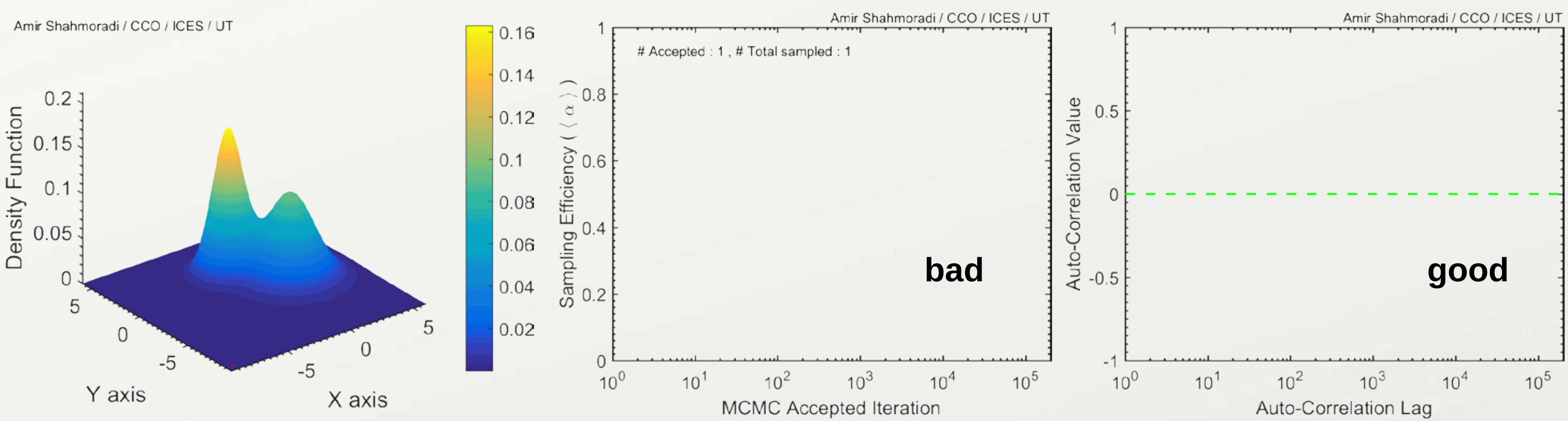


The Art of MH-MCMC Sampling – *Finding the optimal comprise between efficiency and mixing*

Small Steps → High–Efficiency Sampler, Bad Mixing Results (not *i.i.d.* samples)



Large Steps → Low–Efficiency Sampler, Good Mixing Results (almost *i.i.d.* samples)

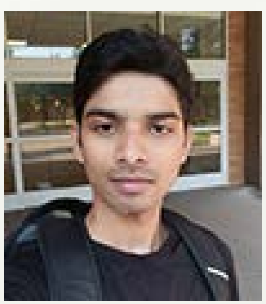


Existing Monte Carlo software

- **C/C++**: MCSim, QUESO, TensorFlow
- **MATLAB**: mcmcstat,
- **Python**: pymcmcstat, PyMC3, emcee, pystan, Zeus
- **R**: FME, mcmc, MCMCpack, greta
- **Java**: Keanu
- **Julia**: Turing.jl, Mamba.jl
- **Fortran**: DREAM, mcmcf90 (by the original author of the Adaptive Metropolis algorithm)

Fatemeh Bagheri
Physicist, UTA

Travis Driver
Aerospace, Georgia Tech



Shashank Kumbhare
Physics, UTA

Joshua Osborne
Physics, UTA

cdslab.org/pm

ParaMonte

Plain Powerful Parallel Monte Carlo Library

The ParaMonte library features and design goals

- Open-source, currently comprised of ~130,000 lines of code in
  - Fortran (50%)
  - MATLAB (25%)
  - Python (15%)
  - Other (10% - C, Cmake, Shell, Batch, …)
- A package addressing some of the major weaknesses of the existing Monte Carlo software.
- The design philosophy of the ParaMonte library:
  - **Full automation** of all Monte Carlo simulations to the highest levels possible to ensure the highest level of **user-friendliness** of the library and **minimal time investment requirements for building, running, and post-processing of simulation models**.
  - **Interoperability** of the core library with as many programming languages as currently possible, including C/C++, Fortran, MATLAB, Python, with ongoing efforts to support other popular programming languages.
  - **High-Performance** meticulously-low-level implementation of the library to ensure the fastest-possible Monte Carlo simulations.
  - **Parallelizability** of all simulations via two-sided and one-sided MPI/Coarray communications while **requiring zero-parallel-coding efforts by the user**.
  - **Zero-dependence** on external libraries to ensure hassle-free ParaDRAM simulation builds and runs.
  - **Fully-deterministic reproducibility** and automatically-enabled restart functionality for all simulations up to 16 digits of precision if requested by the user.
  - **Comprehensive-reporting and post-processing** of each simulation and its results, as well as their automatic compact storage in external files to ensure the simulation results will be comprehensible and reproducible at any time in the distant future.

The ParaMonte library features and design goals

- Why (Modern) Fortran?
  - Reliable backward-compatible language for decades.
  - High performance (as opposed to higher-level programming languages).
  - High-level easy-to-learn (as opposed to C/C++) modularized Object-Oriented language.
  - Native support for scalable parallelism (via Coarrays), mature support for MPI/OpenMP.
  - Native support for many frequently needed numerical objects (arrays), kinds, types, and functions.
  - Excellent standardized interoperability features enabling seamless interactions with other programming languages.
  - All of the above enables the development of one Application Programming Interface (API) for access from all programming languages.
    - The library's kernel routines are all implemented in pure Fortran.
    - Wrappers in C/C++, Fortran, MATLAB, Python, … provide virtually identical interfaces to the library.

Inspired by

SOURCERY INSTITUTE

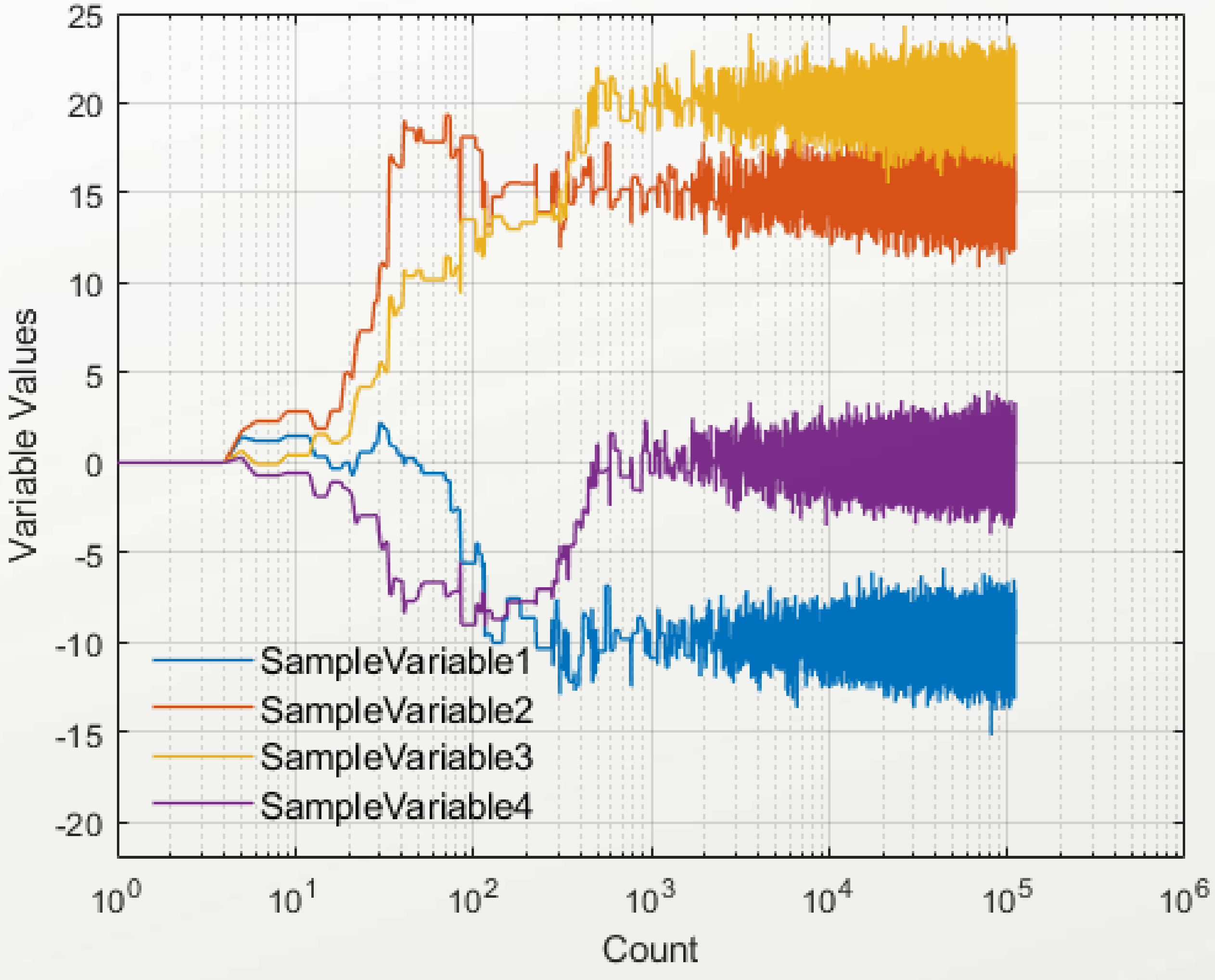
OpenCoarrays

A good example is worth a thousand lines of documentation

DEMO

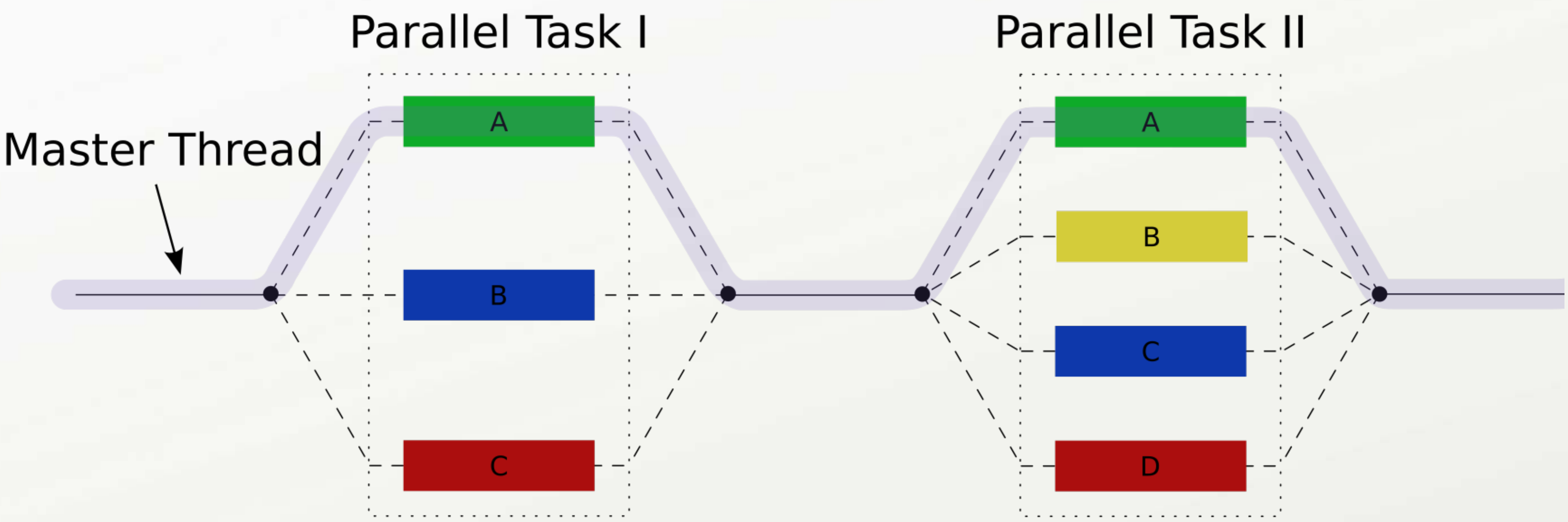
Performance comparison of the parallel versions of ParaMonte

Many Monte Carlo algorithms (e.g., the Markov Chain Monte Carlo) are inherently sequential.

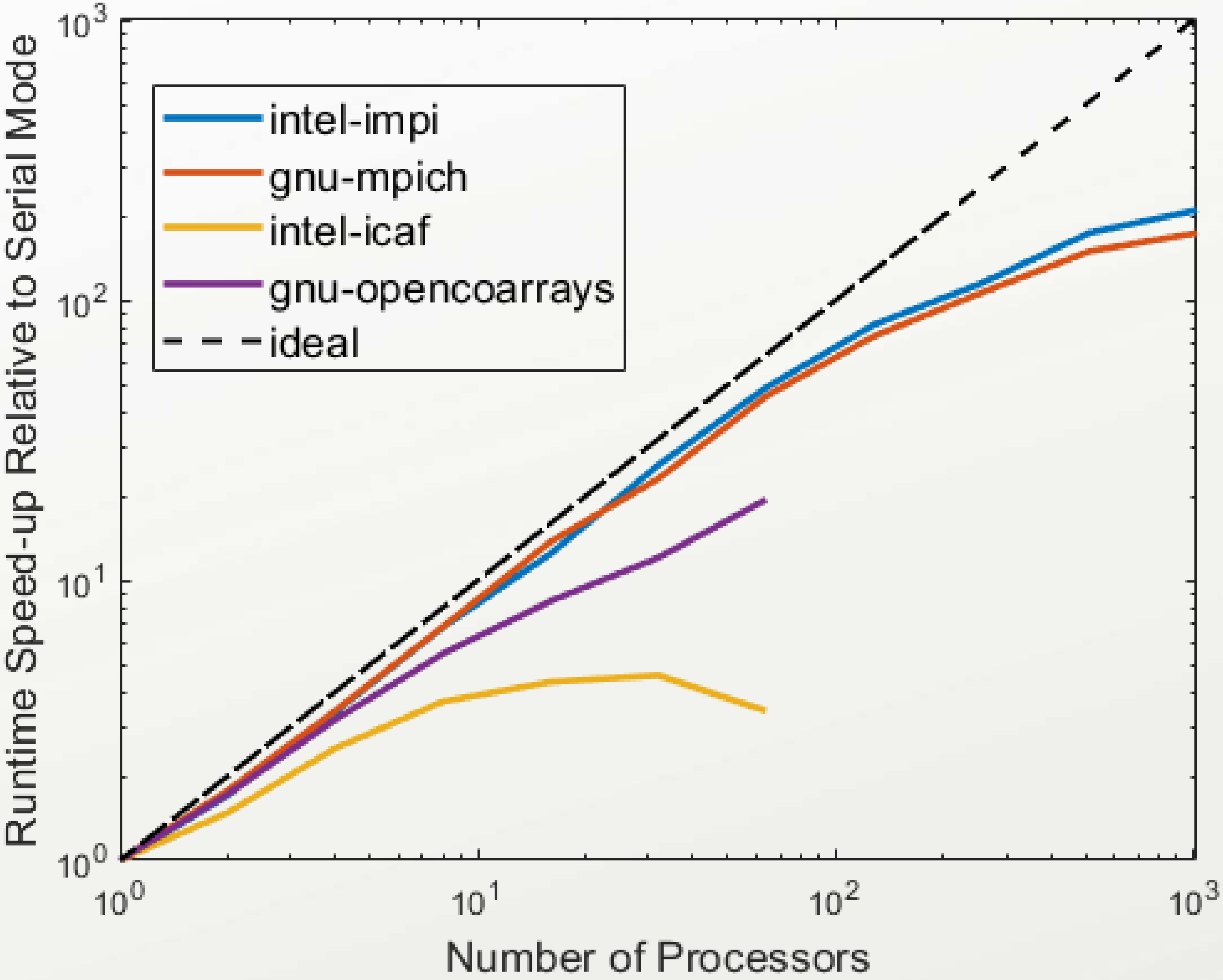


Performance comparison of the parallel versions of ParaMonte

The Fork-Join parallelism



Performance comparison of the parallel versions of ParaMonte



The ParaMonte library development design and goals

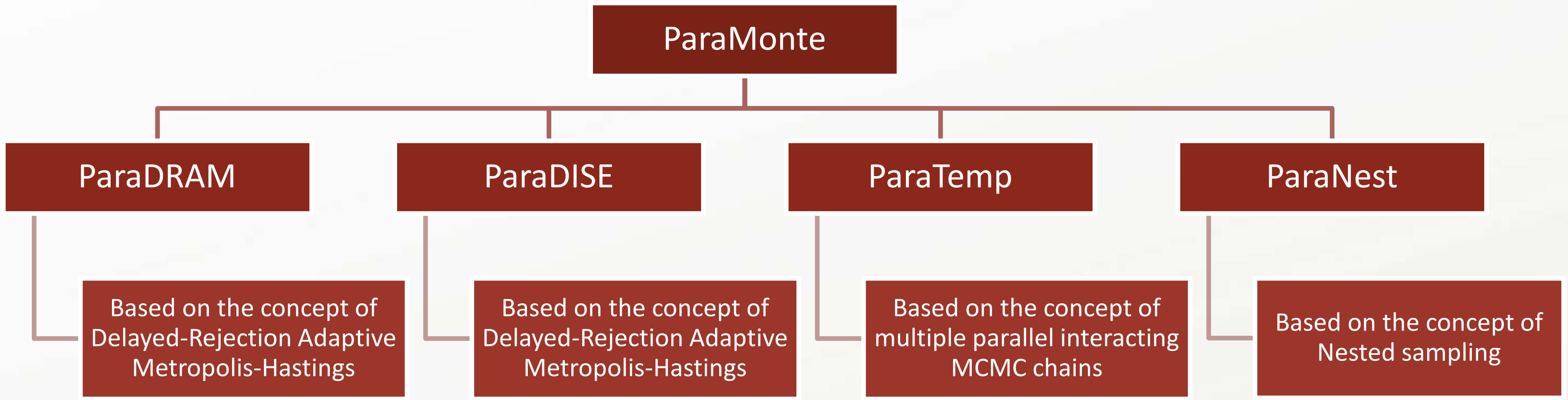
- Strict semantic compliance with the latest Fortran standard (2018).
- Strict source-code compliance with the latest Fortran standard. too restrictive: max line length limit 132 chars
- Strict parallelism compliance with the Fortran standard (via coarrays).
  1. poor compiler support for advanced features,
  2. incompatibility with DLL / shared library packaging
- Strict naming convention enforced within the entire library.
  - Source code should be self-explanatory with minimal need for comments
  - **camelCase** enforced within the entire library (except constants): **outputUnit**
    - Naturally distinguishes Fortran's intrinsic entities from the developer's (**output_unit**).
    - Allows extremely long multi-segment variable names within the 63 character limit of Fortran.
  - Functions / subroutines always begin with a verb: **getCovarianceMatrix**
    - Logical functions always begin with **is**: **isDigit()**
  - All scalar variables begin with lower-case character, otherwise upper-case: **MyMatrix, myScalar** incompatibility with template metaprogramming via Fortran/C preprocessor.
  - All logical variables must be English propositions that evaluate to true or false: **inputFileHasPriority**
  - All Coarray variables must begin with **co_** : **co_LogFunc**
  - All module variables must begin with **mv_** : **mv_State, comv_LogFuncState**
  - All constants (parameters) are upper-case separated by underscore: **FILE_EXT = '.txt'**
  - All module names must end with **_mod**: **ParaMCMC_mod**
  - All derived type names must end with **_type**: **ParaMCMC_type**

```fortran
use ParaMonte_mod, only: ParaMonte_type
ParaMonte = ParaMonte_type()
```

Modern Fortran features used in the ParaMonte library

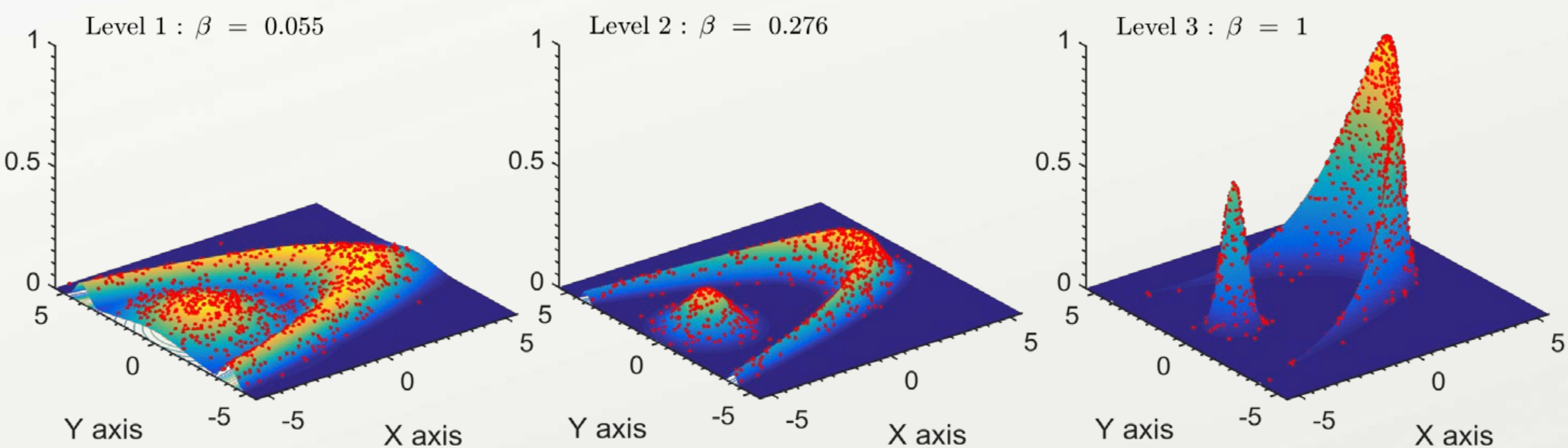
- object-oriented features of **F2003**.
- automatic allocation/reallocation of arrays in **F2003**.
- iso_fortran_env (int8, int16, int32, in64, real32, real64, real128, compiler_version(), compiler_options(), output_unit, IO errors, ...) of **F2008**.
- do-concurrent of **F2008**.
- coarray one-sided parallelism paradigm of **F2008** and **F2018**.
- block-construct of **F2008** (enabling declaration of new variables at any line of the code).
- the g0 edit descriptor of F2008, a true time-saver for simplifying I/O, in particular, CSV file I/O.
- automatically (re)allocatable characters and (re)allocatable components of **F2008**.
- the new array constructor style of **F2003** ([ ] vs. old-style (/ /)).
- the remarkable new C-interoperability features of **F2003**, **F2008**, **F2018**, such as iso_c_binding, bind(), contiguous attribute, C-interoperable optional procedure arguments, ... Without these, communication with other languages, including C/C++, Julia, MATLAB, Python, R, … would have been almost impossible.
- the new attributes of the allocate statement (mold, source, ...) of **F2008.**
- intrinsic support for mathematical functions (Bessel, erf, erfc, log_gamma, norm2, ...) in **F2008**.
- type-bound procedures of **F2003**, and its enhancements in **F2008** and beyond. extremely useful.
- get_environment_variable(), execute_command_line(), command_argument_count(), get_command_argument(), get_command(), and new array searching features like findloc(), ... in **F2008**.
- maximum variable length increase to 63 characters in **F2003**.
- ieee_exceptions, ieee_arithmetic modules for exception handling.
- move_alloc() in **F2003.**
- namelist IO.
- …

The ParaMonte library roadmap



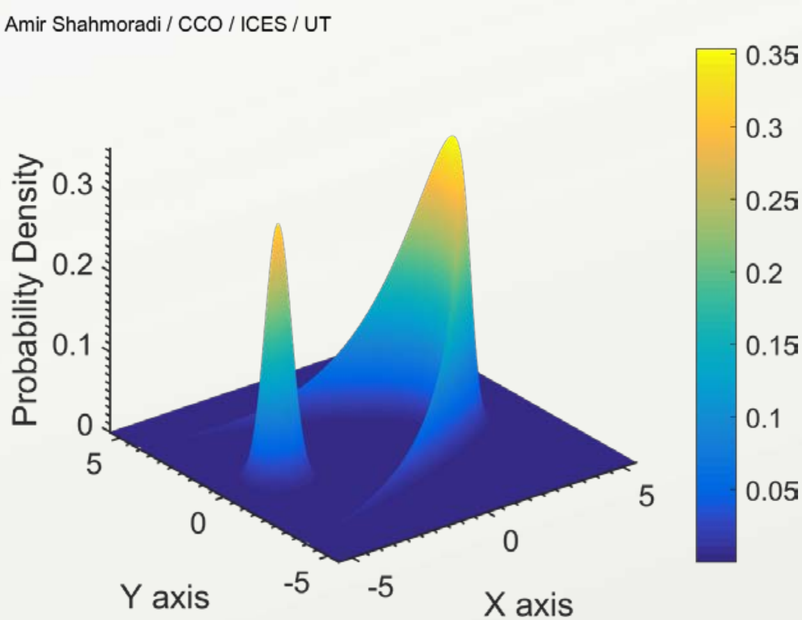
The ParaMonte library roadmap

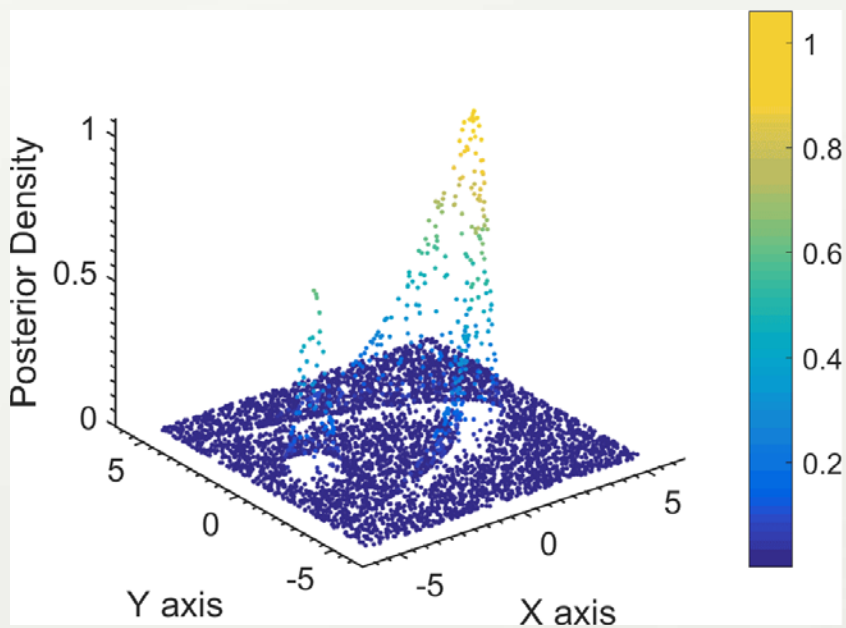
- **ParaMonte**
  - **ParaDRAM**: Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
  - **ParaDISE**: Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
  - **ParaTemp**: Based on the concept of multiple parallel interacting MCMC chains
  - **ParaNest**: Based on the concept of Nested sampling



The ParaMonte library roadmap

- **ParaMonte**
  - **ParaDRAM**
    - Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
  - **ParaDISE**
    - Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
  - **ParaTemp**
    - Based on the concept of multiple parallel interacting MCMC chains
  - **ParaNest**
    - Based on the concept of Nested sampling

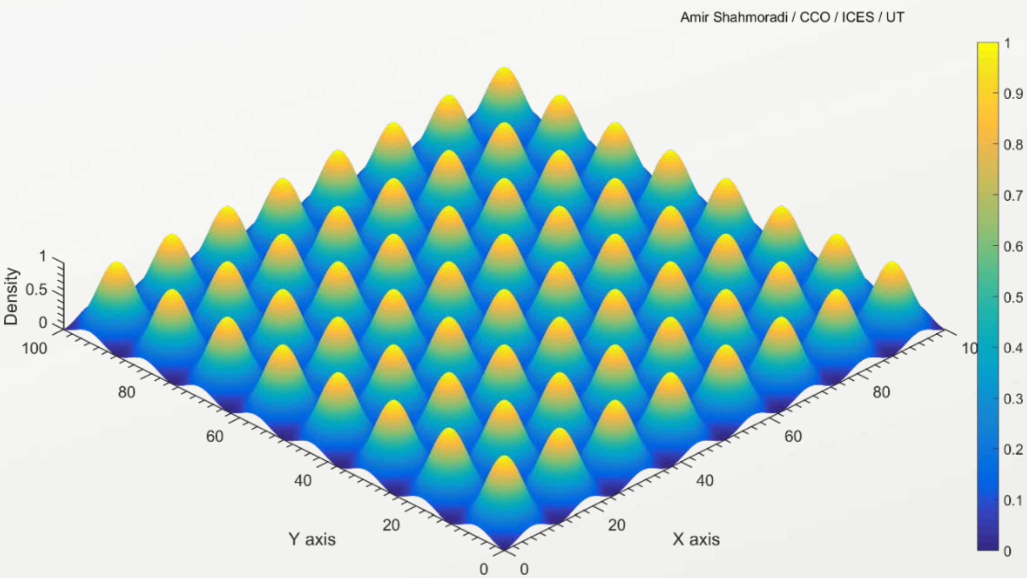


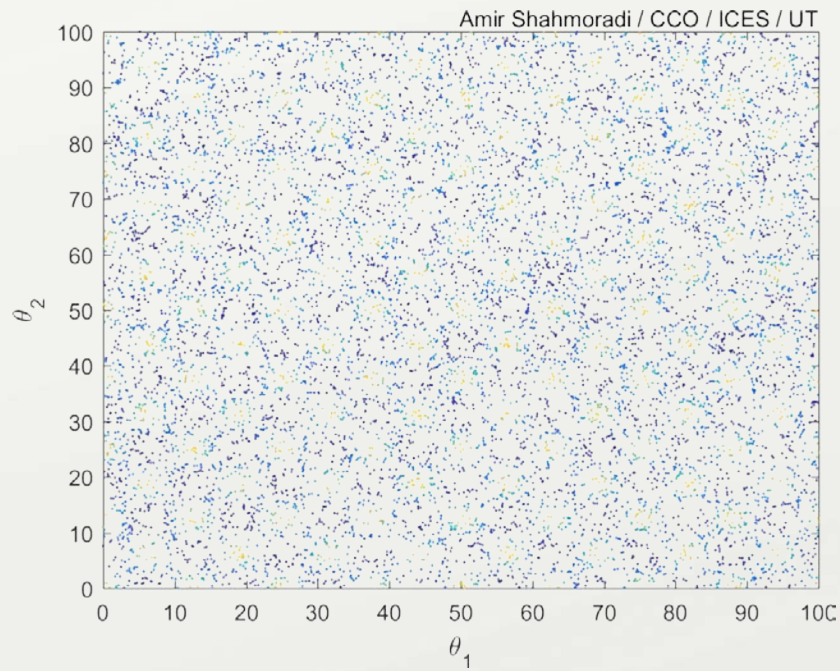


The ParaMonte library roadmap

ParaMonte

- ParaDRAM
  - Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
- ParaDISE
  - Based on the concept of Delayed-Rejection Adaptive Metropolis-Hastings
- ParaTemp
  - Based on the concept of multiple parallel interacting MCMC chains
- ParaNest
  - Based on the concept of Nested sampling





Desired enhancements to the Fortran language

- **Coarray interoperation:** the ability to use CAF in shared (**no_main**) library files (**DLL, so, dylib**).
- Coarray slicing (**MPI_gather**)

```fortran
real :: co_Vector[*]
real, allocatable :: LocalVector(:)
...
LocalVector = co_Vector[ 1 : num_images() ]
```

- enhanced module readability and usage (Python-style module usage).

```fortran
use paramonte as pm
pmpd = pm%ParaDRAM()
```

- the ability to use the non-present optional arguments without defining a substitute (Python-style).

```fortran
function runSampler(chainSize)
    integer, intent(in), optional :: chainSize
    if (.not.present(chainSize)) chainSize = 10000
    ...
```

- **standardized support** for a minimal healthy subset of the **C/Fortran preprocessing** features.
- further support for **template metaprogramming**.

Summary

- ParaMonte is a pure standard-complaint Fortran library with the following design philosophy:
  - **open-source** available at: https://github.com/cdslaborg/paramonte
  - Documentation available at: cdslab.org/pm
  - **Full automation** of all Monte Carlo simulations (to ensure **user-friendliness**).
  - **Interoperability** of the core library with C/C++, Fortran, MATLAB, Python, (Java, Julia, Mathematica, R).
  - **High-Performance** meticulously-low-level implementation of the library to ensure the fastest-possible Monte Carlo simulations.
  - **Parallelizability** via **MPI** / **Coarray** while **requiring zero-parallel-coding efforts by the user**.
  - **Zero-dependence** on external libraries.
  - **Fully-deterministic reproducibility into the future** and automatically-enabled restart functionality.
  - **Comprehensive-reporting and post-processing** of each simulation and its results.
- The next major future release(s) will include:
  - **ParaDISE** (Parallel Delayed-Rejection Adaptive Markov Chain Monte Carlo **on steriod**).
  - **ParaTemp** (Parallel Tempering for stochastic integration and Bayesian model selection)
  - **ParaNest** (Parallel Nested sampling for stochastic integration and Bayesian model selection)
  - **ParaHDMC** (Parallel Hamiltonian Dynamics Markov Chain Monte Carlo)
- **Join us in this effort! Email: shahmoradi@utexas.edu**