

# MODELING COMPUTATION KERNELS WITH STAN

---

Hoël Jalmin

Tutored by Arnaud Legrand and Tom Cornebize

The 21th of June, 2019

With the current need for high performance computing, and the hardware complexity:

- How to predict the duration of computations?
- How to quantify uncertainty?

**For this talk:**

1. Brief presentation of the context
2. Introduction to Bayesian sampling through Stan
3. Examples of application

## Modern context

- HPC systems use thousands of nodes, cache, hyperthreading, etc  
→ makes it difficult to predict performance
- Some functions (like DGEMM in the BLAS library) are used everywhere, and called thousands of times in a program.

## Previous work

- Simulating high performance programs to optimize them at a lesser cost
- Elaborated complex models within a few percent of reality but needed to evaluate and confirm them

**Model** Let's say  $y \sim \mathcal{N}(\alpha * x + \beta, \sigma)$

- $\alpha, \beta, \sigma$ : Model parameters
- $y$ : Dependent data (posterior)
- $x$ : Independent data

We observe some data and need to find model parameters

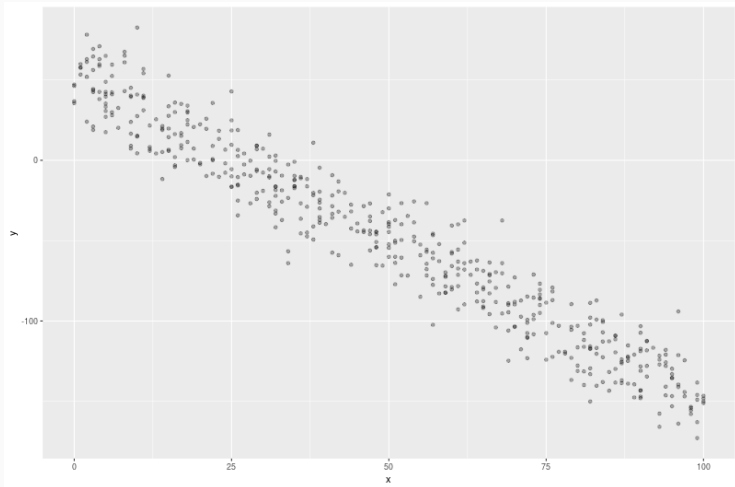
## The vocabulary

- **Posterior**: The distribution of the parameters
- **Likelihood**: A function of the parameters, the model
- **Prior**: Existing knowledge of the system, guesses on the parameters values ( $\sigma > 0$  per example)

# A BAYESIAN SAMPLER, STAN

---

## WITH A SIMPLE EXAMPLE



Using this data, we'll try to find the parameters that were used to generate it.

# THE STAN MODEL

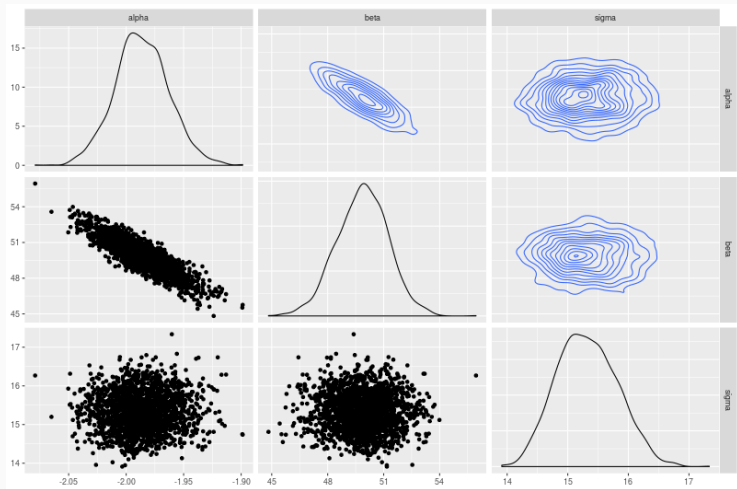
```
library(rstan)

modelString = "data { // the observations
  int<lower=1> N; // number of points
  vector[N] x;
  vector[N] y;
}
parameters { // what we want to find
  real beta;
  real alpha;
  real<lower=0> sigma; // indication: sigma cannot be negative
}
model {
  // We define our priors
  beta ~ normal(0, 10);
  alpha ~ normal(0, 10);
  sigma ~ normal(0, 10);

  // Then, our likelihood function
  y ~ normal(alpha*x + beta, sigma);
}
"

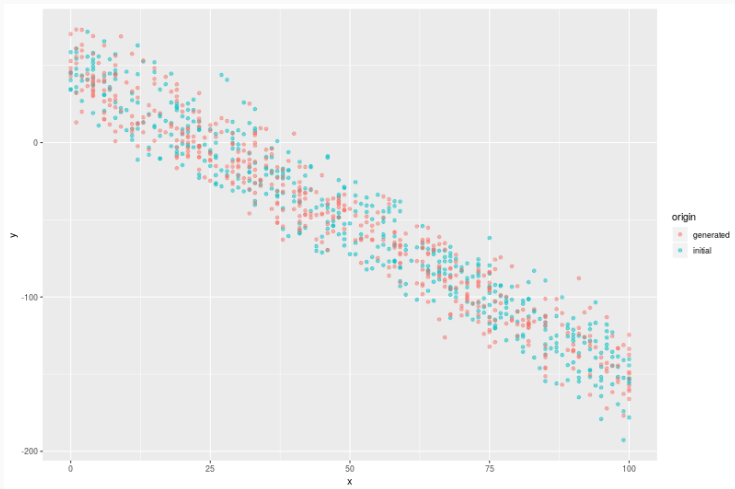
sm = stan_model(model_code = modelString)
```

# LOOKING AT THE POSTERIOR





# LOOKING AT THE GENERATED DATA



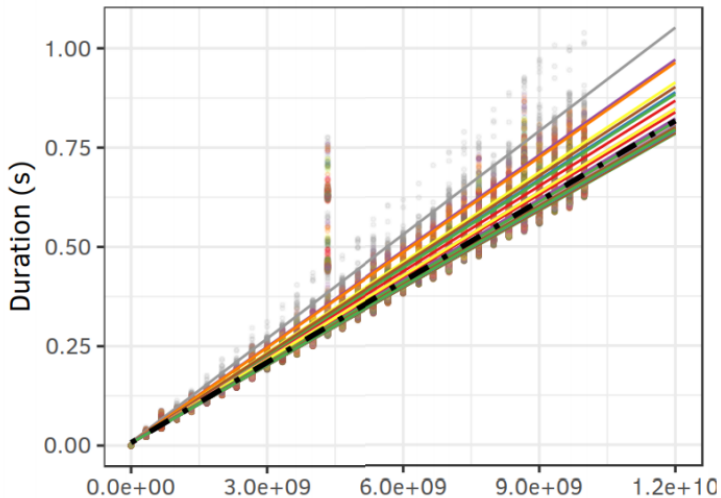
- The priors are necessary to have convergence in the fit
- Non-informative prior vs informative (careful not to have a falsely informative one and introduce bias)
- A little bit of precision is better, but initialisation values can do the trick

# THE DIFFERENT MODELS FOR DGEMM

---

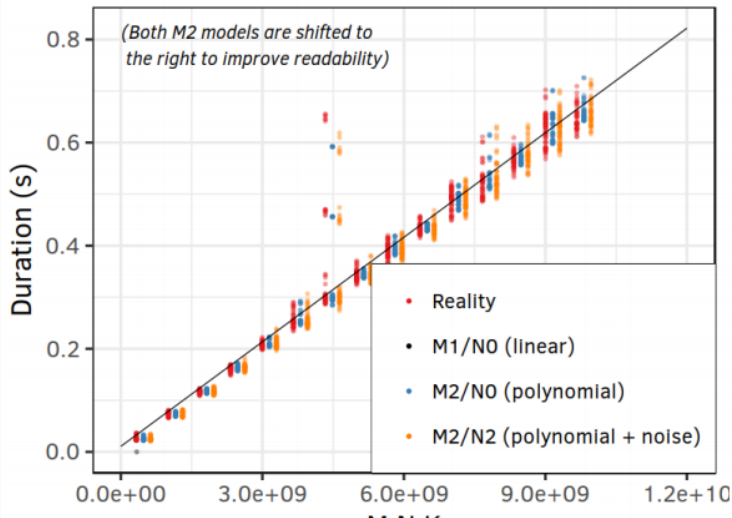
## SPATIAL AND TEMPORAL VARIABILITY

- DGEMM's duration depends on the matrix size, on the CPU used to run it, and on residual noise coming from the system.



# THE POSSIBLE MODELS

(Source: Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study)



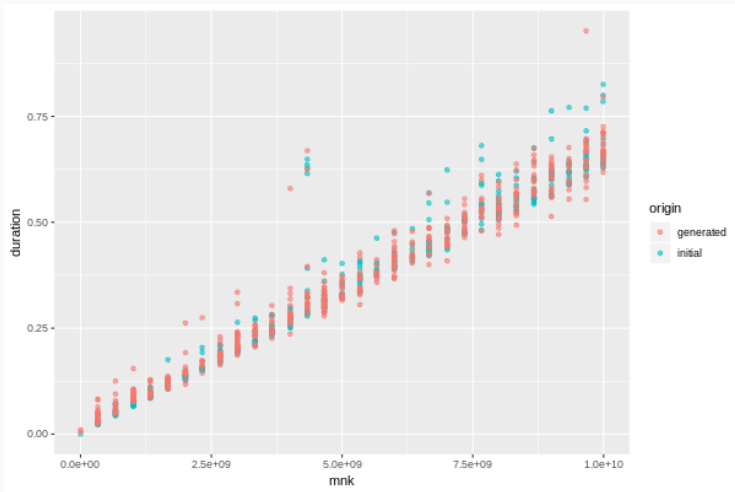
## MODEL 1 : A POLYNOMIAL MODEL WITH NOISE DEPENDING ON X

A redo of the last model presented before, using Stan. Like a linear model but with more parameters (in this case 10).

The model follows this:

$$\text{duration} \sim \mathcal{N}(\alpha_1 * mnk + \alpha_2 * mn + \alpha_3 * mk + \alpha_4 * nk + \beta, \\ \gamma_1 * mnk + \gamma_2 * mn + \gamma_3 * mk + \gamma_4 * nk + \delta)$$

# THE GENERATED DATA



## MODEL 2 : A MODEL WITH PARAMETERS DEPENDING ON THE HOST

- Much like the previous model, but with different observations for each host
- Added a variable for the number of hosts, and used matrices instead of vectors for all the parameters.

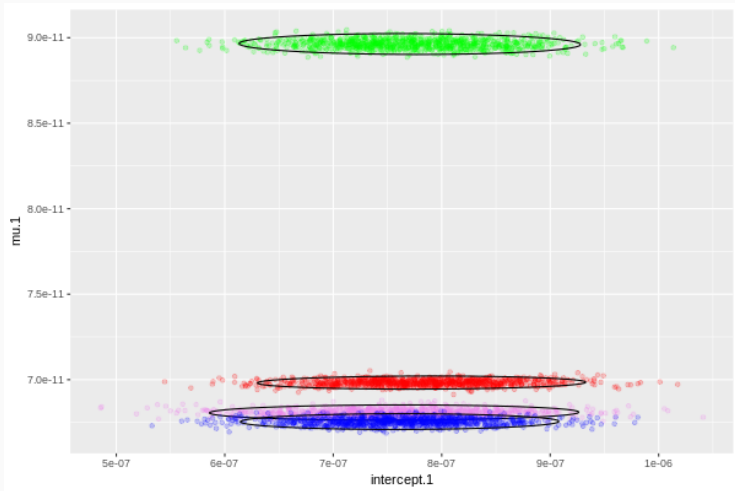
For this model we have:

$$\text{duration}[i] \sim \mathcal{N}(\alpha_1[i] * mnk + \alpha_2[i] * mn + \alpha_3[i] * mk + \alpha_4[i] * nk + \beta[i], \\ \gamma_1[i] * mnk + \gamma_2[i] * mn + \gamma_3[i] * mk + \gamma_4[i] * nk + \delta[i])$$



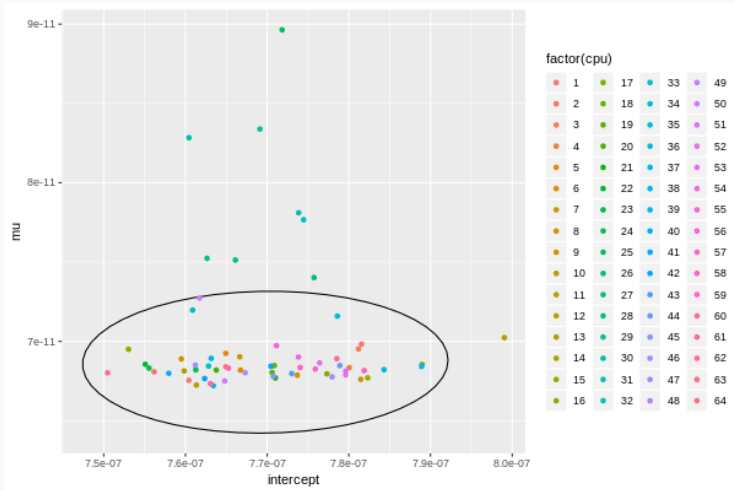
# POSTERIOR VISUALISATION

The posterior with models depending on the host shows a lot of difference between hosts (here we have 3 "average" CPU and a slow one):



# POSTERIOR VISUALISATION

If we look at the means of the parameters' values for each host, we get a range of values in which most hosts are.



## MODEL 3 : A HIERARCHICAL LINEAR MODEL

- Useful to find the value of hyperparameters from which we get the parameters
- From this we could calculate new parameters for new CPUs
- Here  $\mu_{\alpha}$  and  $\sigma_{\alpha}$  are the hyperparameters for  $\alpha$ , and the same goes for the other parameters

$\mu_{\alpha} \sim \mathcal{N}(\alpha_{\text{moy}}, \alpha_{\text{sd}})$  with  $\alpha_{\text{moy}}$  and  $\alpha_{\text{sd}}$  the priors

$\sigma_{\alpha} \sim \mathcal{N}(0, 1)$

$\alpha[i] \sim \mathcal{N}(\mu_{\alpha}, \sigma_{\alpha})$

$\text{duration}[i] \sim \mathcal{N}(\alpha[i] * mnk + \beta[i], \gamma[i] * mnk + \delta[i])$

## My contribution

- Created several models to represent the performance of a computation kernel within a few percent of reality
- Model adaptable to changes (addition/removal of CPUs)

## Following up work

- Implementing this work in Simgrid research (other computation kernels, network communications, etc)
- Novelty detection and non regression performance tests