

# Notes on Gaussian process regression

Mark Blyth

---

## Week's goal

Last week. . .

- ✂ Kernel choice is critical to getting usable results from a GP (more so than hyperparameters?)
- ✂ Stationary covariance functions aren't able to fully capture neuron dynamics
- ✂ Periodic kernels are harder to fit, but better represent our data

This week. . .

- ✂ Decide on a suitable kernel (something non-stationary, non-monotonic)
- ✂ Learn how it works
- ✂ Implement it

---

## Week's activities

- ✿ Read about modelling non-stationarity in GPs
  - ▶ Appropriate kernel choice allows us to model non-stationarity
  - ▶ There's also more general GPs that allow us to do that
  - ▶ These generalised GPs also have other nice features such as heteroscedasticity, and non-Gaussian noise
- ✿ Considered experimental issues with GPs
  - ▶ Fitting time
  - ▶ Noise type
- ✿ Wrote up notes about everything I've done with GPs so far
  - ▶ Covers some of the key developments in relevant areas of the literature

---

## Presentation points

1. Highlight what the requirements are for *in silico* experiments
  - ▶ Goal: identify what we want from a GP
2. Consider how experimental requirements may differ from *in silico*
  - ▶ Goal: identify what we want from a GP
3. Suggest some possible approaches
  - ▶ Goal: find some possible modelling strategies that fit our requirements
4. Compare approaches to help decide which approach is best
  - ▶ Goal: choose the strategy to work with

---

## Why GPs?

Always useful to remember why we're doing things!

- ✿ All existing CBC methods require us to discretise the system behaviours (outputs, control signals)
  - ▶ Neurons spike fast, so this is hard
- ✿ Discretisations are necessary in the continuation procedure either to...
  - ▶ predict and correct the control target,
  - ▶ or to iteratively zero the control action
- ✿ I'm hoping we can avoid discretisation by using simple transformations of continuous functions, rather than discretised vectors

This usage case defines our requirements of the Gaussian processes

---

---

## The best GP

The best Gaussian process model satisfies the following:

- ✿ Easy to use and understand
  - ▶ No need to re-invent the wheel
  - ▶ Simplicity is a virtue!
- ✿ Gives a *sufficiently* good model
  - ▶ Doesn't necessarily have to be perfect, depending on how we use the corrector step
- ✿ Easy to train
  - ▶ Hyperparameters are either quick and easy to tune, or the model works well even with bad hyperparameters

How simple we can go depends on the data we expect to see

---

---

## Data characteristics for *in silico* CBC

With computer experiments, we have...

- ✂ Reliable results with only small amounts of data
  - ▶ GP training speed doesn't matter
- ✂ Negligible noise
  - ▶ Only noise is numerical errors
- ✂ 'Nice' artificial noise
  - ▶ We know exactly what the noise distribution is
  - ▶ Noise can be exactly Gaussian
  - ▶ Noise can have constant variance

---

## Choice of GPs for *in silico* CBC

- ✂ Small amounts of data mean we don't need any clever speedup methods
- ✂ Constant-variance noise means we don't need to try modelling heteroscedastic behaviours
- ✂ Gaussian noise means we don't have warp data into a Gaussian process
- ✂ Multiple-timescale dynamics mean we can't rely on stationarity

The only non-standard requirement is that the covariance function must be non-stationary.



---

## Data characteristics for *in vitro* CBC

With real experiments, we have...

- ✂ Potentially lots of data, if we assume *KHz* sample rates
  - ▶ GPs must train quickly
- ✂ Unavoidable noise
  - ▶ Noise might not be Gaussian, especially for measurement-precision errors
  - ▶ Noise variance might change with signal amplitude (eg. multiplicative noise)

---

## Issues with GPs for *in vitro* CBC

### 🔥 Lots of data

- ▶ GPs are  $\mathcal{O}(n^3)$  to train, so they become impractical with more than a few thousand datapoints

### 🔥 Non-Gaussian noise

- ▶ GPs are a collection of random variables, whose finite joint distribution is Gaussian
- ▶ This means GPs only let us model Gaussian noise

### 🔥 Non-constant signal noise

- ▶ GPs are heteroscedastic – the noise is assumed to be constant across the signal
- ▶ This might not be true for our experiments

Luckily there's a range of solutions to all these problems!

---

---

## Nice GPs

A 'nice' Gaussian process is stationary:

- ✦ Strong stationarity: moments (hyperparameters) remain constant across the signal
- ✦ Weak stationarity: mean, variance remain constant across the signal
- ✦ Standard kernels assume stationarity
- ✦ Stationary GP models are analytically tractable, with simple closed-form solutions

---

## Practical GPs

Realistic data aren't stationary; there's two main approaches to handle this:

- ✦ Learn a transformation of the data, so that the transformed data are stationary
- ✦ Learn a kernel that can handle the non-stationarity observed in the signal

Non-stationary models are not always analytically tractable, and require more advanced solution methods.

---

## Warping GPs

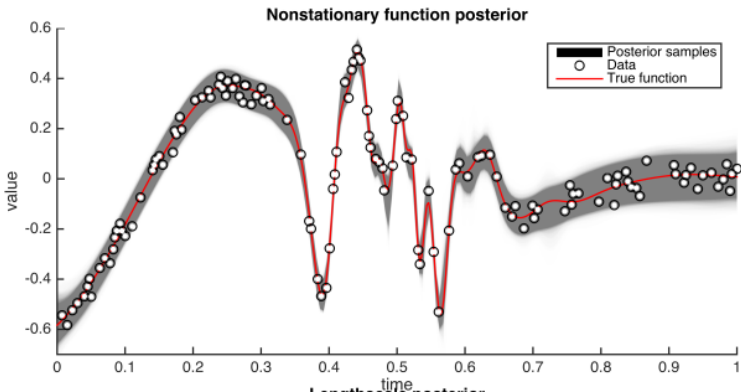
- ✿ Gaussian processes assume observations are distributed Gaussian'ly about a true function value
- ✿ When this isn't true, we can try to learn a transformation from the original data to some latent variables, such that the latent variables *are* Gaussian
- ✿ A 'nice' GP can then be fitted to the latent variables
- ✿ This allows us to model non-Gaussian noise
- ✿ Only works when it's possible to transform the signal into a stationary GP

---

## Nonstationary Gaussian processes

- ✦ Take the standard square-exponential kernel
- ✦ Replace the hyperparameters with latent functions (while retaining PD)
- ✦ Model the latent functions as GPs
- ✦ Design the kernel by fitting those GPs
- ✦ There's clever optimisation techniques, **but they're not necessarily fast, and they require good hyperpriors**
  - ▶ Since all neuron data will look similar (in some respects), it's probably possible to train a kernel on a representative dataset; using it on novel data will then only require small optimisations

# Nonstationary GPs work well on biological data



Source: Heinonen, Markus, et al. "Non-stationary gaussian process regression with hamiltonian monte carlo." Artificial Intelligence and Statistics. 2016.

*This also shows how important kernel choice is – couldn't do that with an SE kernel!*

---

## Spectral kernels

- ✂ Bochner's theorem relates the power spectrum of a signal to its covariance
  - ✂ A custom kernel can be designed by fitting a GP to the signal power spectrum, and inverse-Fourier-transforming the result
    - ▶ This means we only have to fit one latent GP
    - ▶ Derives the kernel directly from the data, so presumably these methods will give the most reasonable kernel for the given problem
  - ✂ The resulting kernel can model non-monotonic covariance (long-term trends, eg. periodicity), and can be designed to be non-stationary
  - ✂ They seem to be exactly the same as the previous non-stationary method, but designed in a perhaps easier-to-compute way
  - ✂ Spectral kernels have also been developed with sparse methods in mind. . .
-



---

## Sparse Gaussian processes

- ✿ Gaussian processes train in  $\mathcal{O}(n^3)$ ; this is too slow for  $n > \mathcal{O}(10^3)$  datapoints
- ✿ When faced with big data, we could train a GP by selecting a subset of data to work with
  - ▶ This throws away useful information
- ✿ Alternative: learn a set of representative latent variables, and train on those
  - ▶ For  $m$  latent vars, we get  $\mathcal{O}(nm^2)$  complexity
- ✿ Sparse GPs let us train on a smaller number of variables, while minimising loss of information
  - ▶ KL divergence gives a measure of the difference between PDFs
  - ▶ Variational Bayesian methods give an upper bound on the KL divergence between true posterior, and sparse posterior
  - ▶ Gradient descent can then be used to minimise this upper bound

---

## Choice of GPs

- ✦ Spectral kernels and the pictured nonstationary GP method will both work well for neuron data
- ✦ The best method is the one that's
  - ▶ quickest to train
  - ▶ easiest to implement
- ✦ More advanced learning methods are harder to understand and implement
  - ▶ Instead of spending weeks reinventing the wheel, existing codes should be used

Which one of these two I choose therefore depends on what software implementations are available.

---

## Next steps

- ✂ Days / immediate: Go back and make changes to the continuation paper
- ✂ Week / medium-term: (Once the paper is finished), find code for, implement, and test the chosen GP scheme
- ✂ Weeks / longer-term: code the predictor-corrector methods, giving a completed CBC code