Gaussian Process Probabilistic Programming

UNIVERSITY OF CAMBRIDGE

Will Tebbutt, Wessel Bruinsma, and Richard E. Turner

{wct23,wpb23,ret26}@cam.ac.uk

- Write programmes specifying the relationship between a collection of Gaussian processes (GPs)
- Both Julia and Python implementations available: **Stheno.jl** and **Stheno**
- GP perspective: adopt a **process-centric** as opposed to **kernel-centric** approach to model specification
- PP perspective: useful corner-case where **exact** inference is **tractable**

Implementing GPPPs: Maths $\Longleftrightarrow$ Code

Maths	Code
$s \sim \mathcal{GP}(0, k_{\text{quadratic}} + k_{\text{EQ}}),$	`s = GP(Quadratic() + EQ())`
$a \mid s = \mathrm{d}^2 s / \mathrm{d}t^2,$	`a = ∇(∇(s))`
$\varepsilon_s \sim \mathcal{GP}(0, k_{\text{noise}}),$	`ε_s = GP(0, Noise())`
$\varepsilon_a \sim \mathcal{GP}(0, k_{\text{noise}}),$	`ε_a = GP(0, Noise())`
$y_s \mid s, \varepsilon_s = s + \varepsilon_s,$	`y_s = s + ε_s`
$y_a \mid a, \varepsilon_v = a + \varepsilon_a,$	`y_a = a + ε_a`
$p(s \mid y_s(t) = \text{obs\_s}, y_a(t) = \text{obs\_a})?$	`s_posterior = s | (y_s(t) ← obs_s, y_a(t) ← obs_a)`

Kernel and mean function of the posterior: `kernel(s_posterior)` and `mean(s_posterior)`

Background

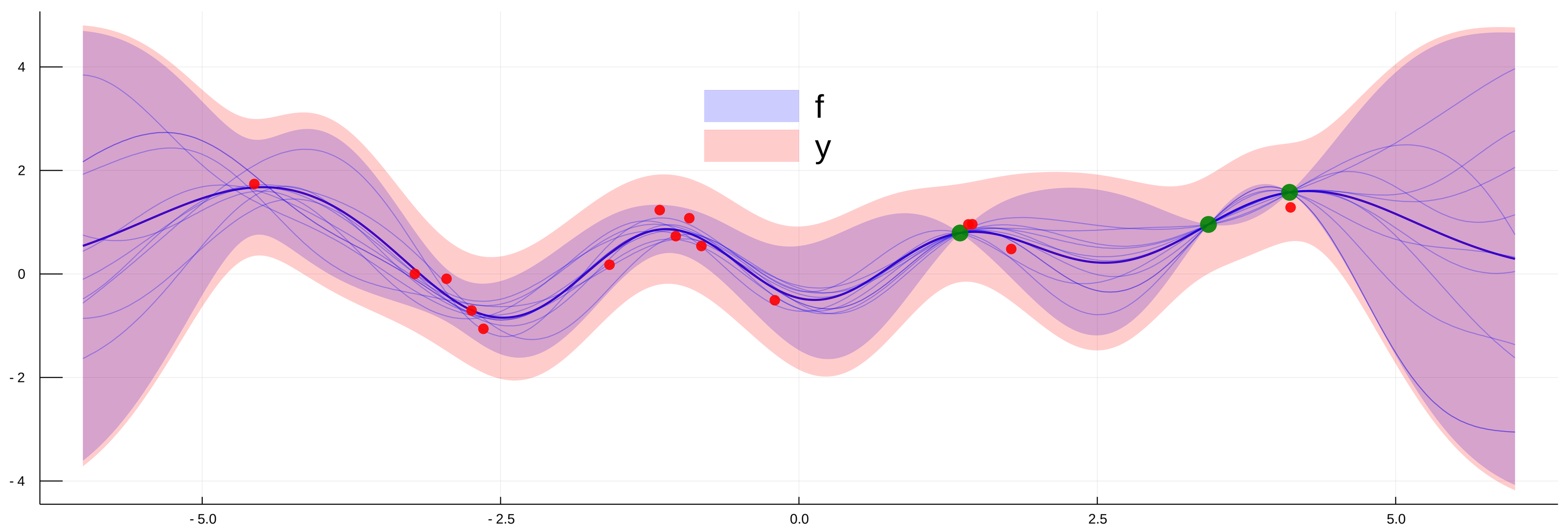
A Gaussian Process Probabilistic Program (GPPP) **is** a GP.

- GP: useful prior for real-valued functions, closed under affine transformation [1]
- A programme comprises a collection of **primitive** GPs and **affine transforms** thereof
- This collection defines a GP on an **augmented domain**
- Mean function and kernel on augmented domain deduced automatically from programme structure
- Strictly more general than traditional GP packages (e.g. [2]), enables inspection of distribution of any component of the model
- Usable as a primitive distribution in a general PPL

[1] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[2] D. G. Matthews, G. Alexander, M. Van Der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman, "Gpflow: A gaussian process library using tensorflow", *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1299–1304, 2017.

Partially-Noisy Nonlinear Regression



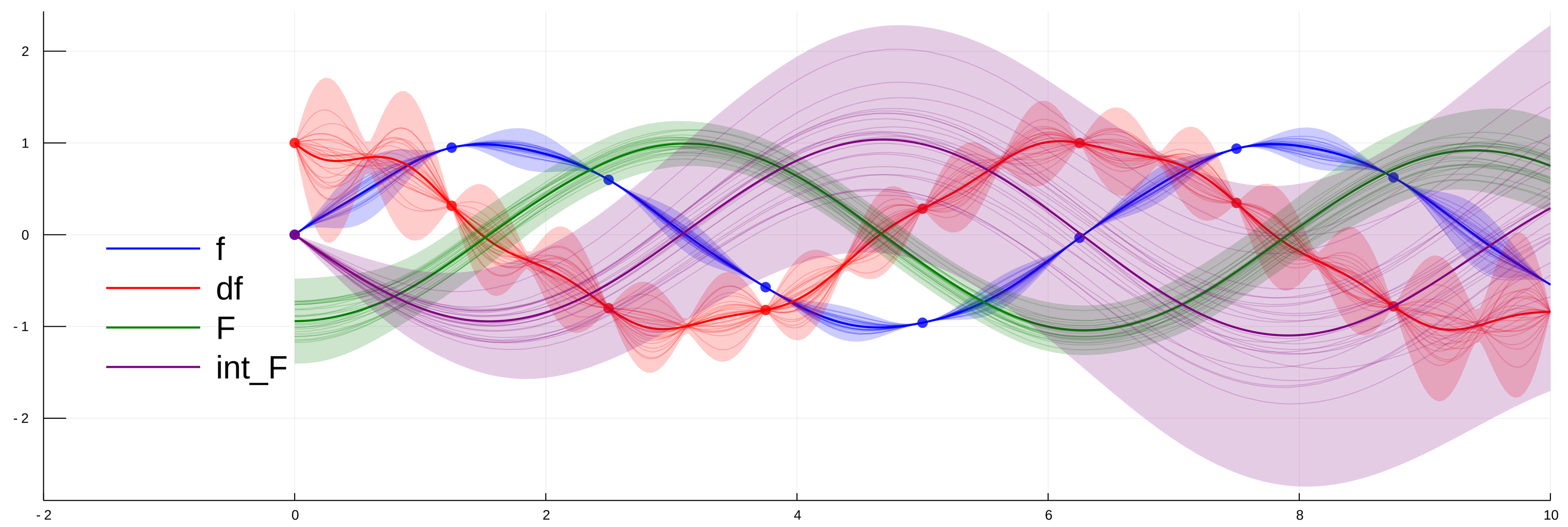

```julia
# Specify generative model.
@model function gp(σ²)
    f = 1.5 * GP(EQ())
    ε = GP(Noise(σ²))
    y = f + ε
    return f, y
end
f, y = gp()

# Sample from the prior.
Xf = rand(Uniform(-5, 5), 3)
Xy = rand(Uniform(-5, 5), 15)
f_, y_ = rand([f(Xf), y(Xy)])

# Compute posterior processes.
f′, y′ = (f, y) |
    (f(Xf) ← f_, y(Xy) ← y_)
```

- Small number of observations f_ of latent process f
- Larger number of observations y_ of noisy process y
- Condition on both to infer posterior over f and y

Probabilistic Integration

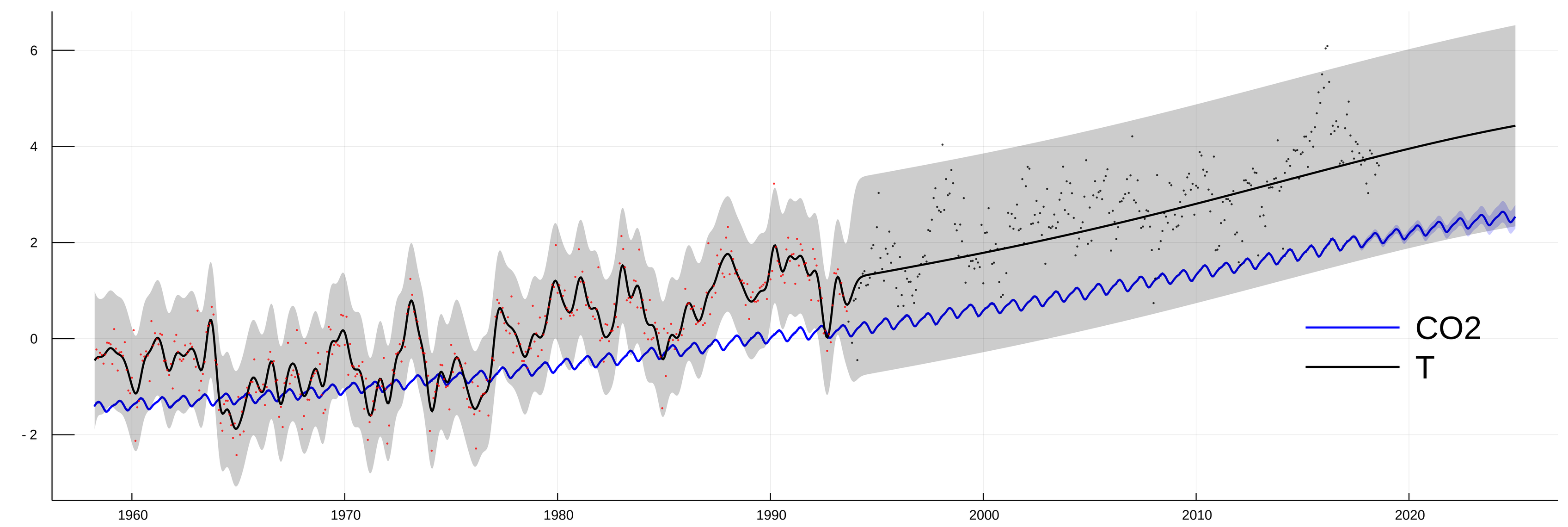



```julia
@model function gp()
    ∫F = GP(EQ())
    F = ∇(∫F)
    f = ∇(F)
    df = ∇(f)
    return ∫F, F, f, df
end
∫F, F, f, df = gp()

x = range(0, stop=10, length=9)
f_, df_ = sin.(x), cos.(x)
∫F′, F′, f′, df′ = (∫F, F, f, df) |
    (∫F([0.0]) ← [0.0],
     f(x) ← f_,
     df(x) ← df_)
```

- Infer the double integral of f
- Make observations of f and its gradient, df
- Nestable AD enables multiple integration
- Take care to fix the integration constants

Toy Climate Problem




```julia
@model function climate(θ)

  # Joint latent trend.
  trend = GP(...)

  # Specify model for CO2.
  co2_trend = trend * θ[1]
  co2_wiggle = GP(...)
  co2_period = GP(...)
  co2 = co2_trend + co2_wiggle +
    co2_period + GP(Noise(...))

  # Specify model for T.
  T_trend = f_trend * θ[2]
  T_wiggle = GP(...)
  T = T_trend + T_wiggle +
    GP(Noise(...))

  return co2, T
end
```

- Goal: predict T (temperature) given co_2 (CO_2 concentration)
- Idea: jointly model co_2 and T
  - Infer long term trend in both co_2 and T
  - Account for short-term fluctuations separately
- Trend recovered; irreducible variation in T accounted for

Check out Stheno on GitHub: **https://github.com/willtebbutt/Stheno.jl** **https://github.com/wesselb/stheno**