Probabilistic Machine Learning with Julia

Lecture 5 | Gaussian Processes

Amirabbas Asadi

amir.asadi78@sharif.edu

2025

Today's Topics

- Multivariate Normal Distribution
- Gaussian Processes
- Kernel Design
- Gaussian Processes in Probabilistic Programs
- Gaussian Random Field, Nonparametric Latent ODE, ...

Nonparametric Models

Every model we have seen so far had been defined with a parametric family of functions

- Bayesian Linear Regression : $\alpha\tau + \beta$
- Bayesian Neural Network : $\sigma(Wx + b)$
- Bayesian Differential Equations : $\frac{dx}{dt} = f(x, \frac{dx}{dt}, t; \theta)$

Gaussian Process

How to define a distribution over functions?

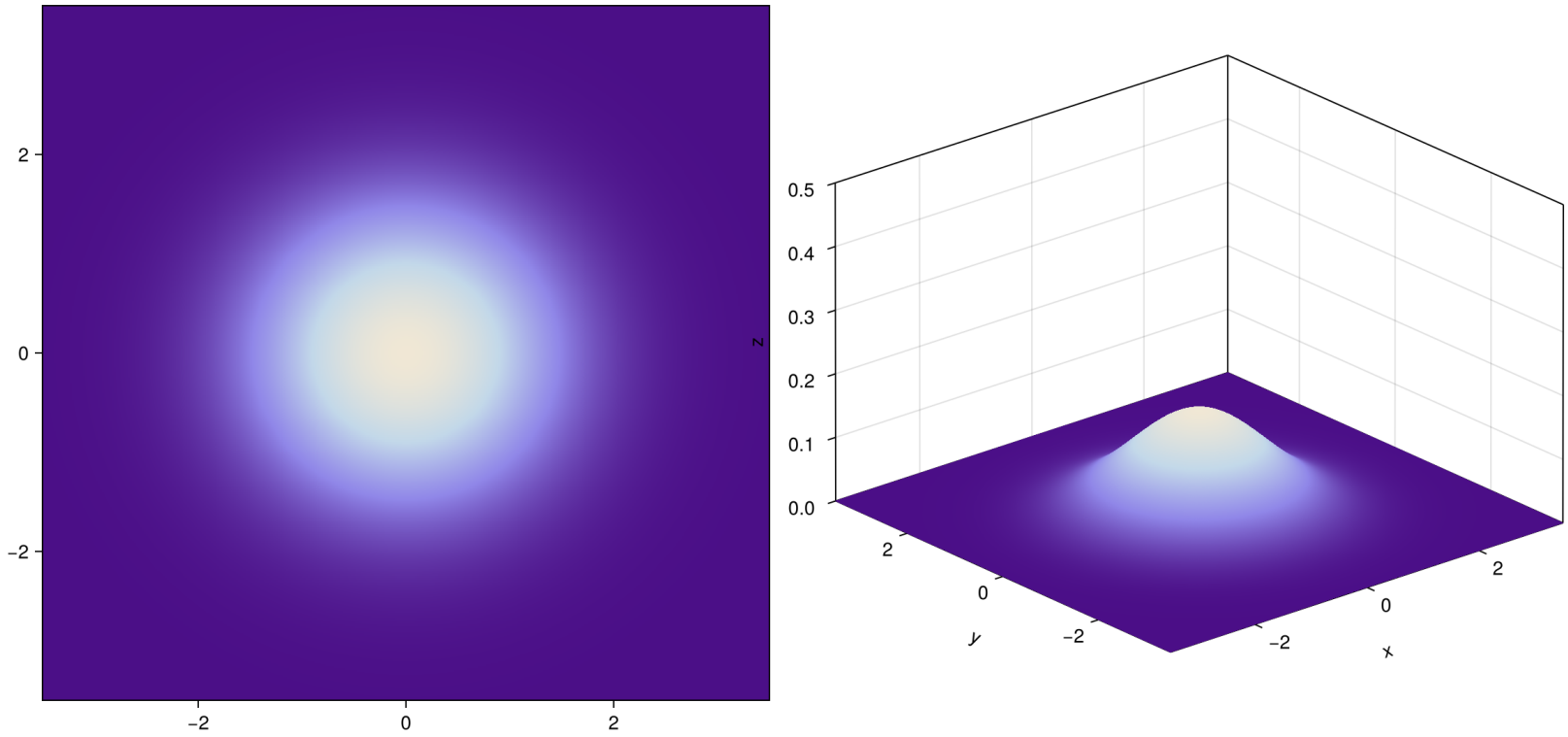
Multivariate Normal Distribution

$$p(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

- $\mu \in \mathbb{R}^d$ is the mean vector
- $\Sigma \in \mathbb{R}^{d \times d}$ is the covarioance matrix so has to be positive-definite matrix.

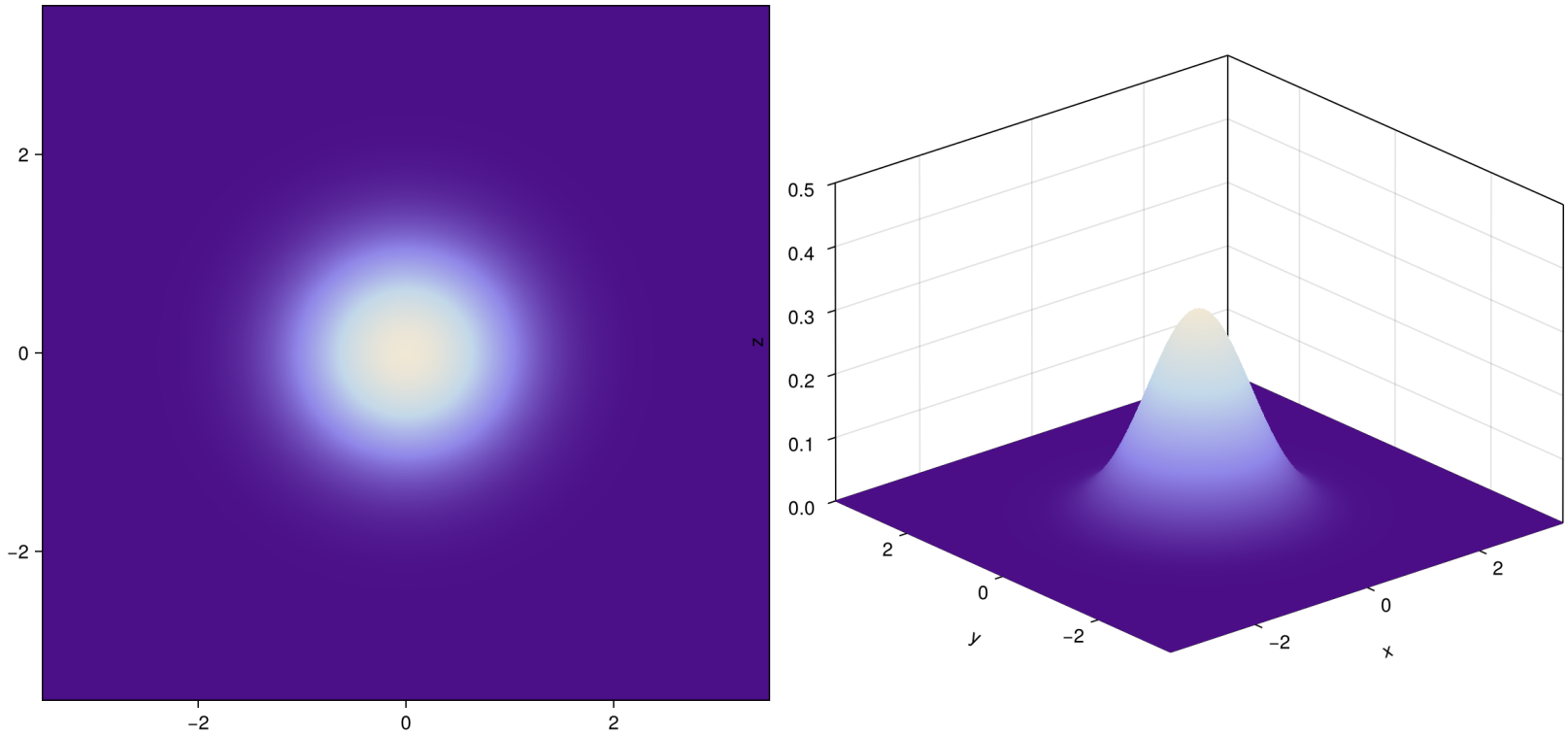
Multivariate Normal Distribution

$$\mu = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$



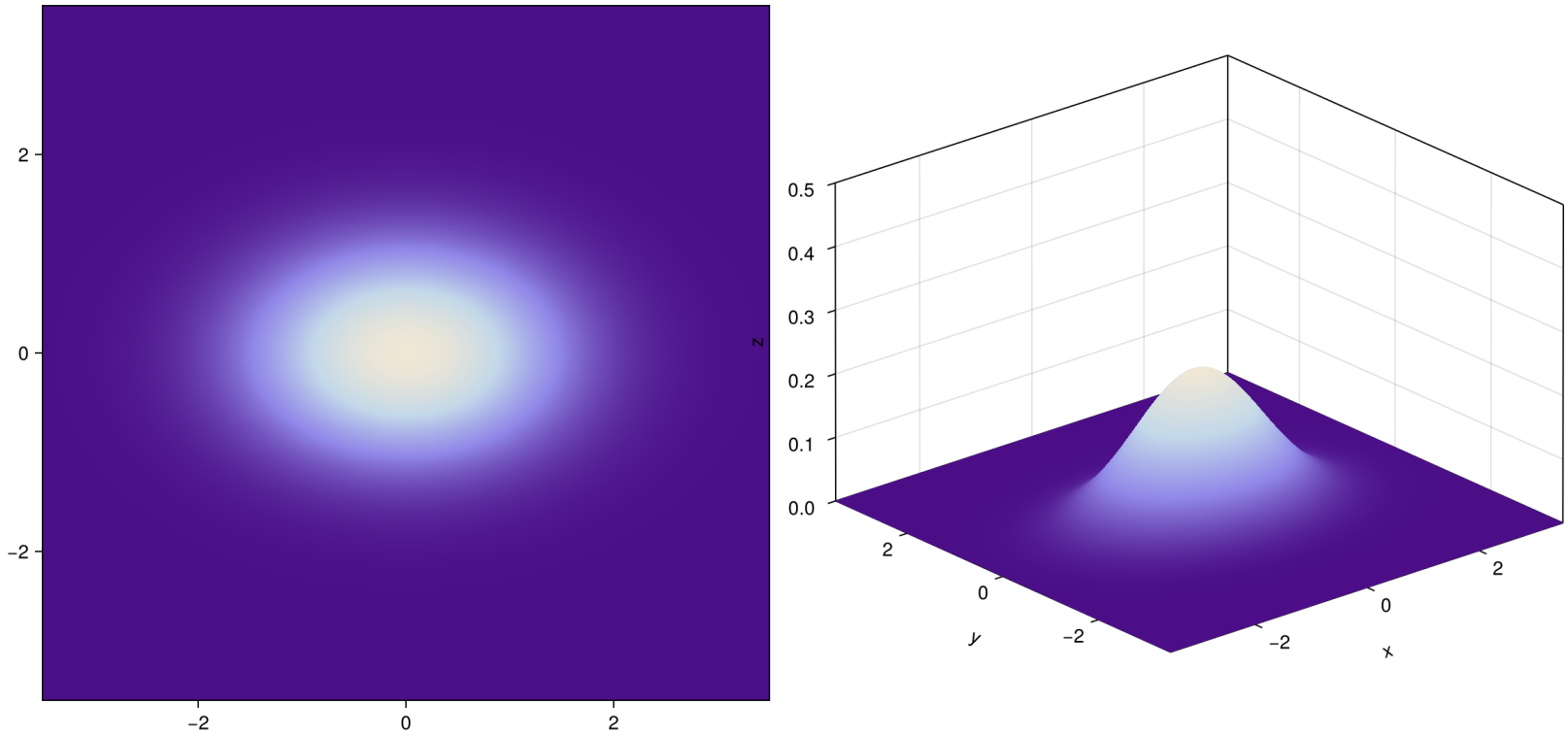
Multivariate Normal Distribution

$$\mu = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix}$$



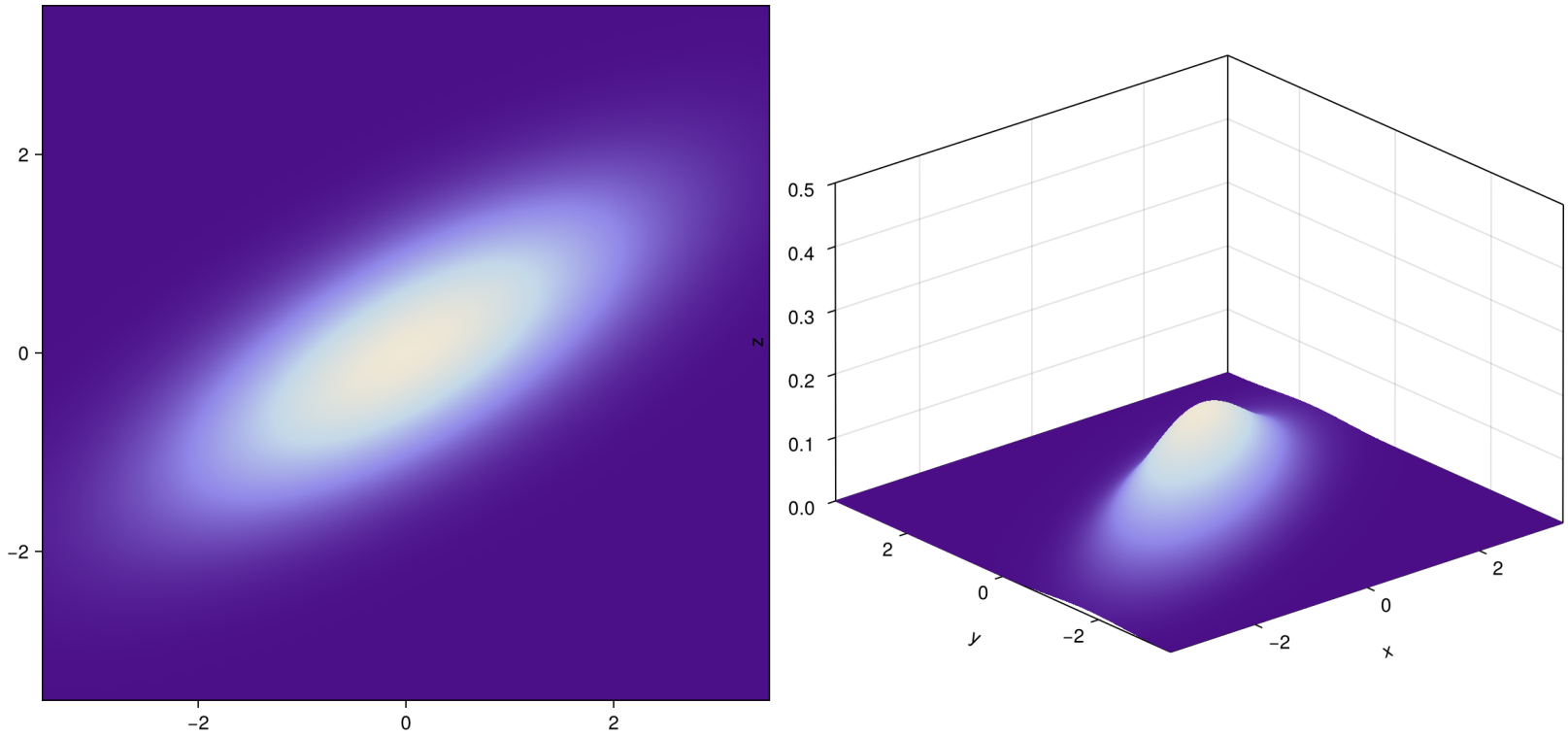
Multivariate Normal Distribution

$$\mu = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 0.5 \end{bmatrix}$$

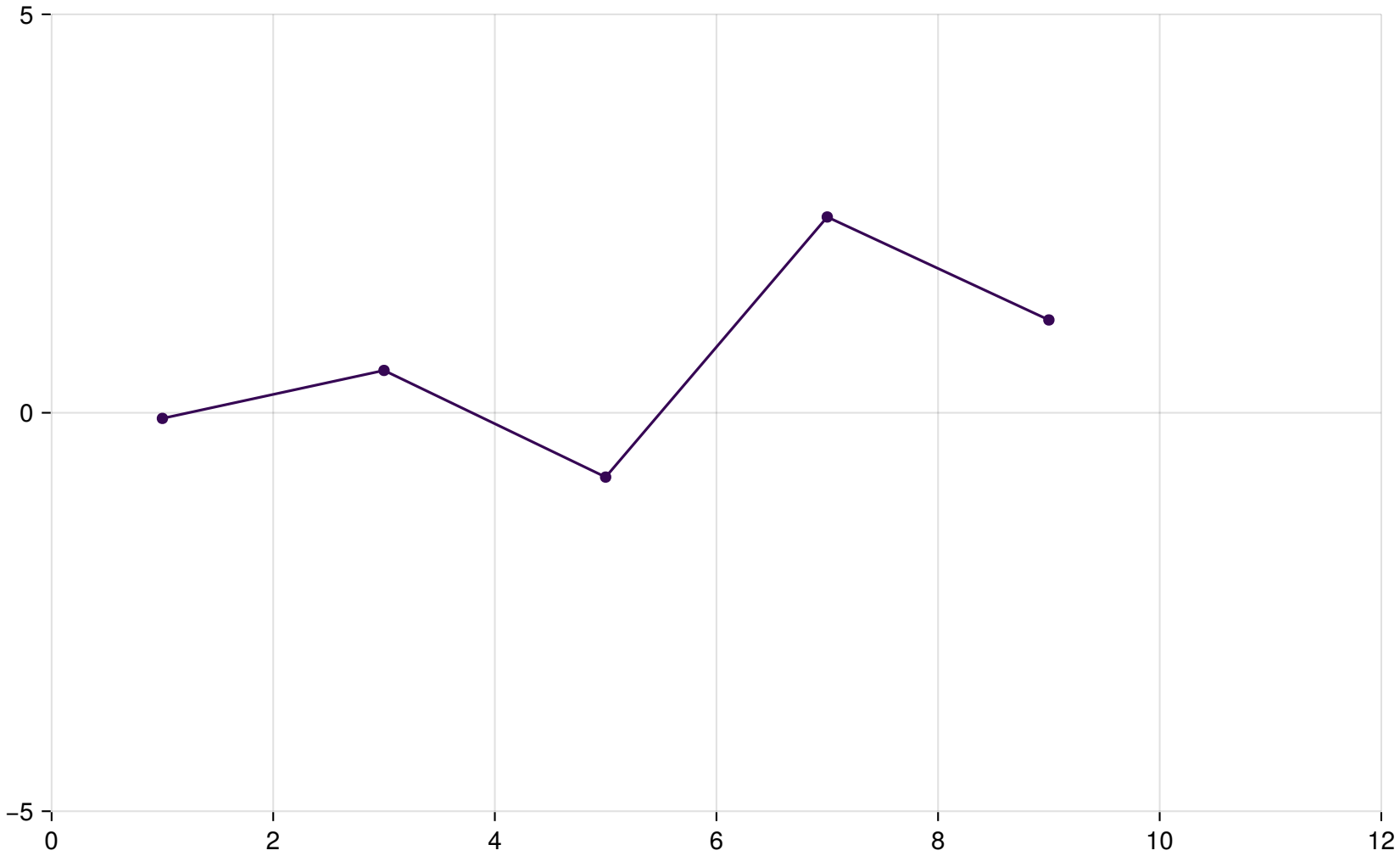


Multivariate Normal Distribution

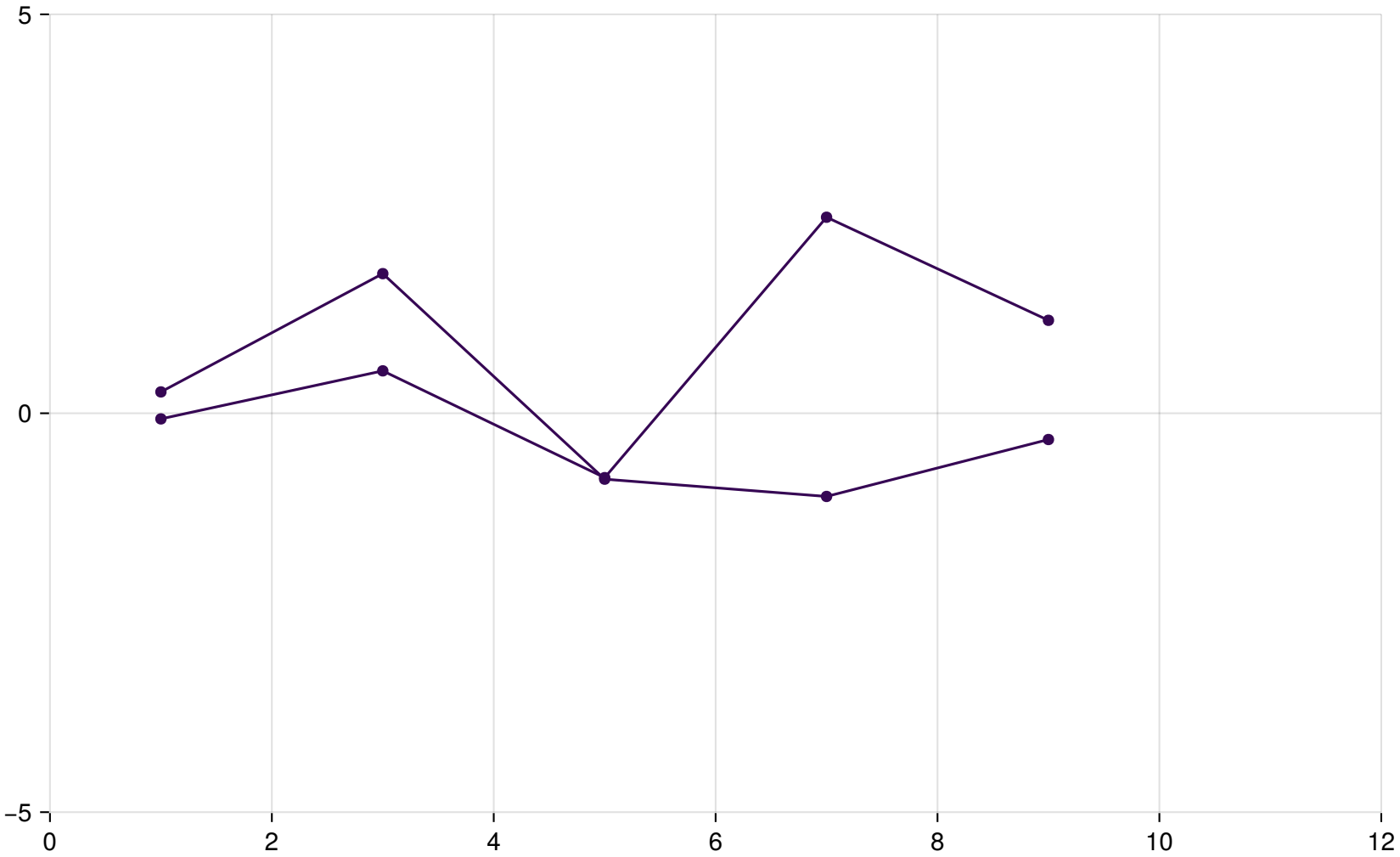
$$\mu = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$



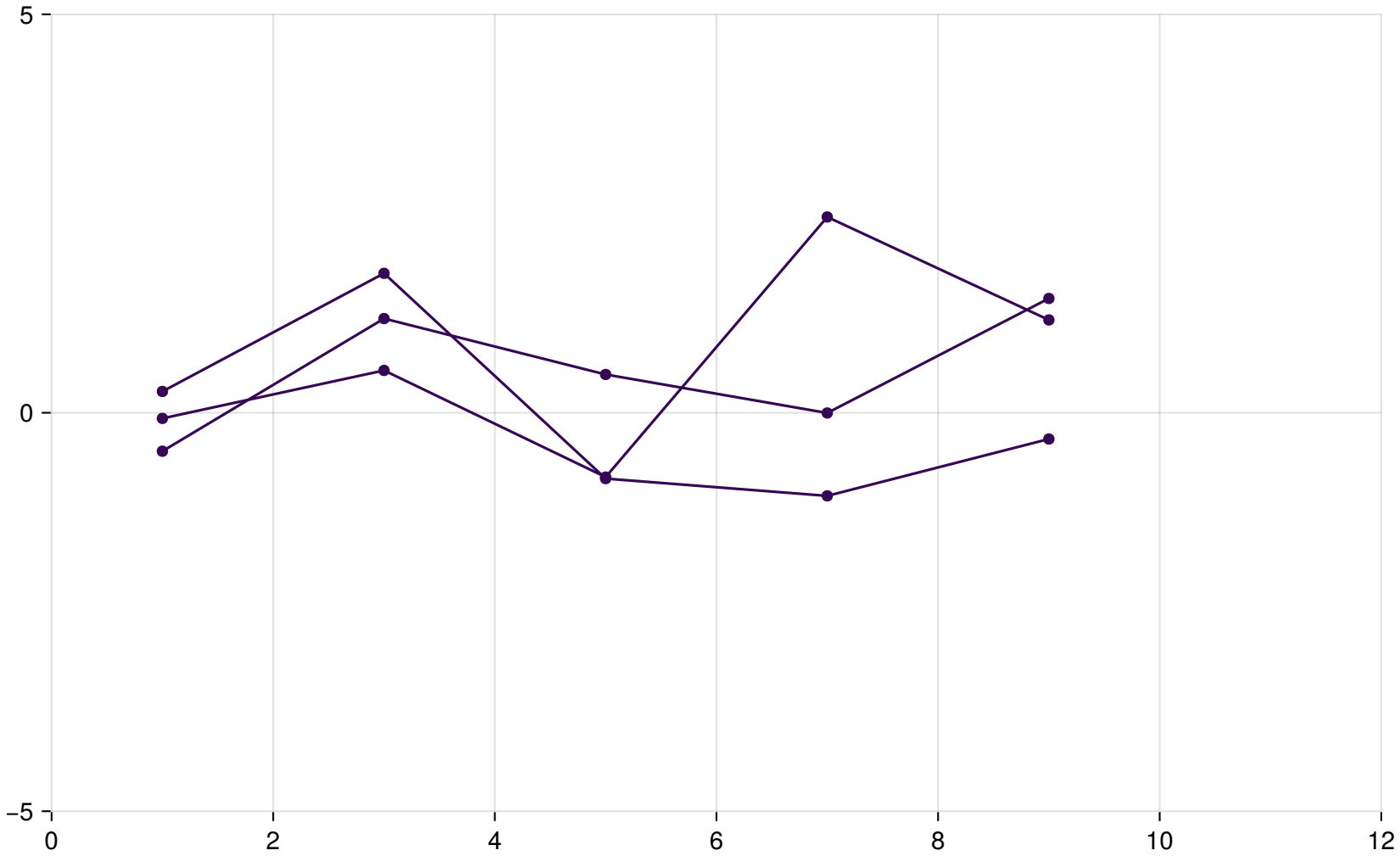
Gaussian Process



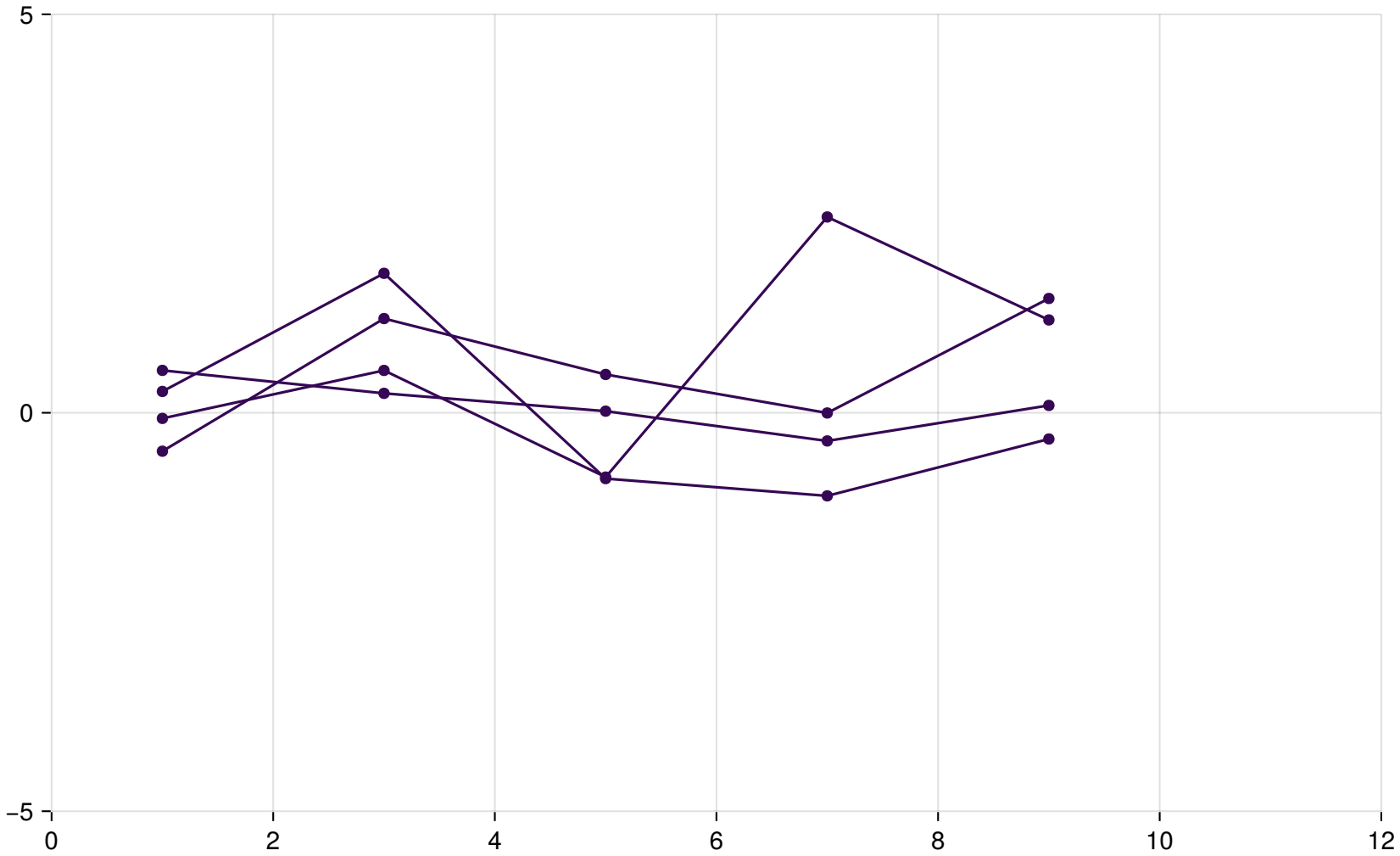
Gaussian Process



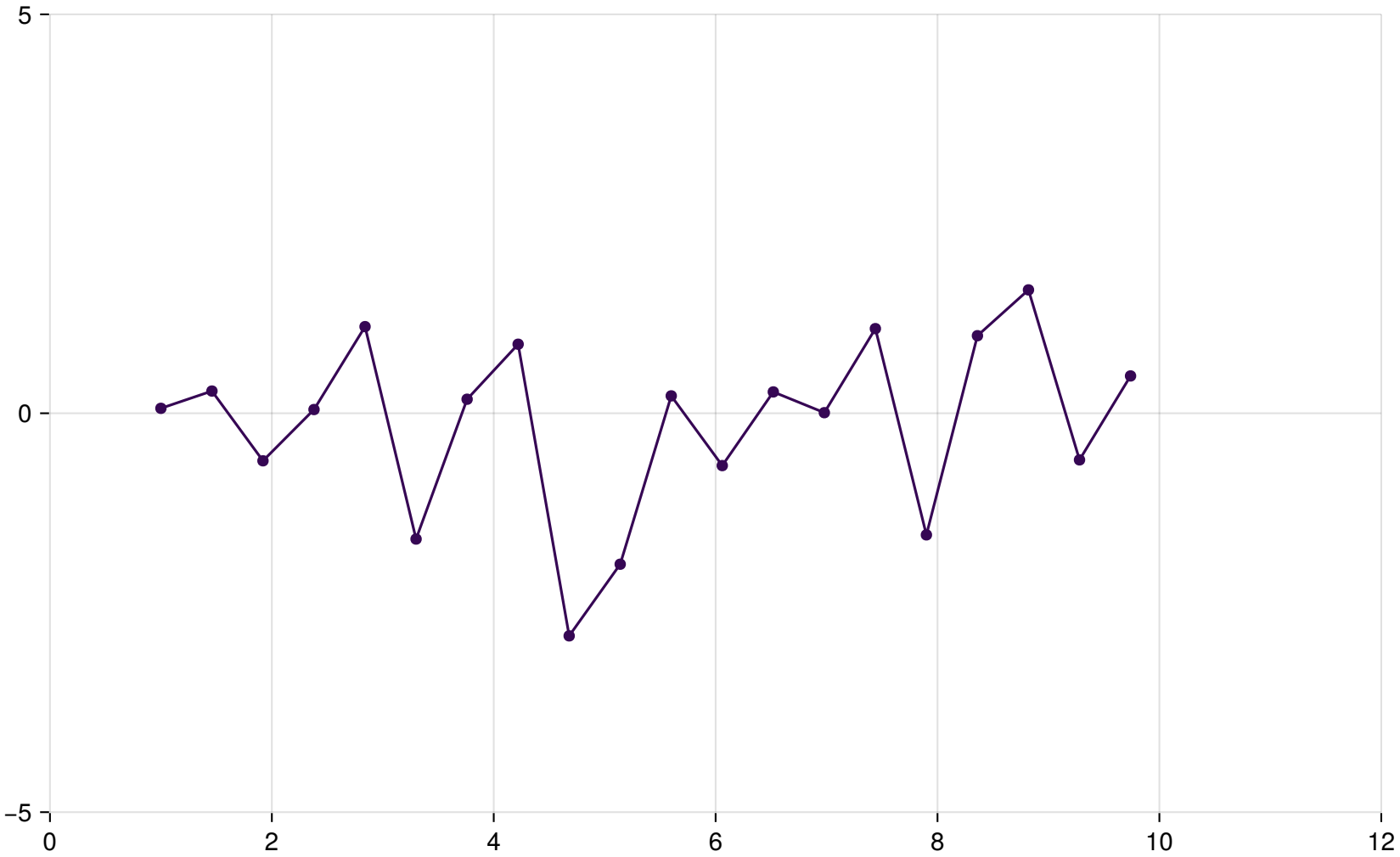
Gaussian Process



Gaussian Process



Gaussian Process

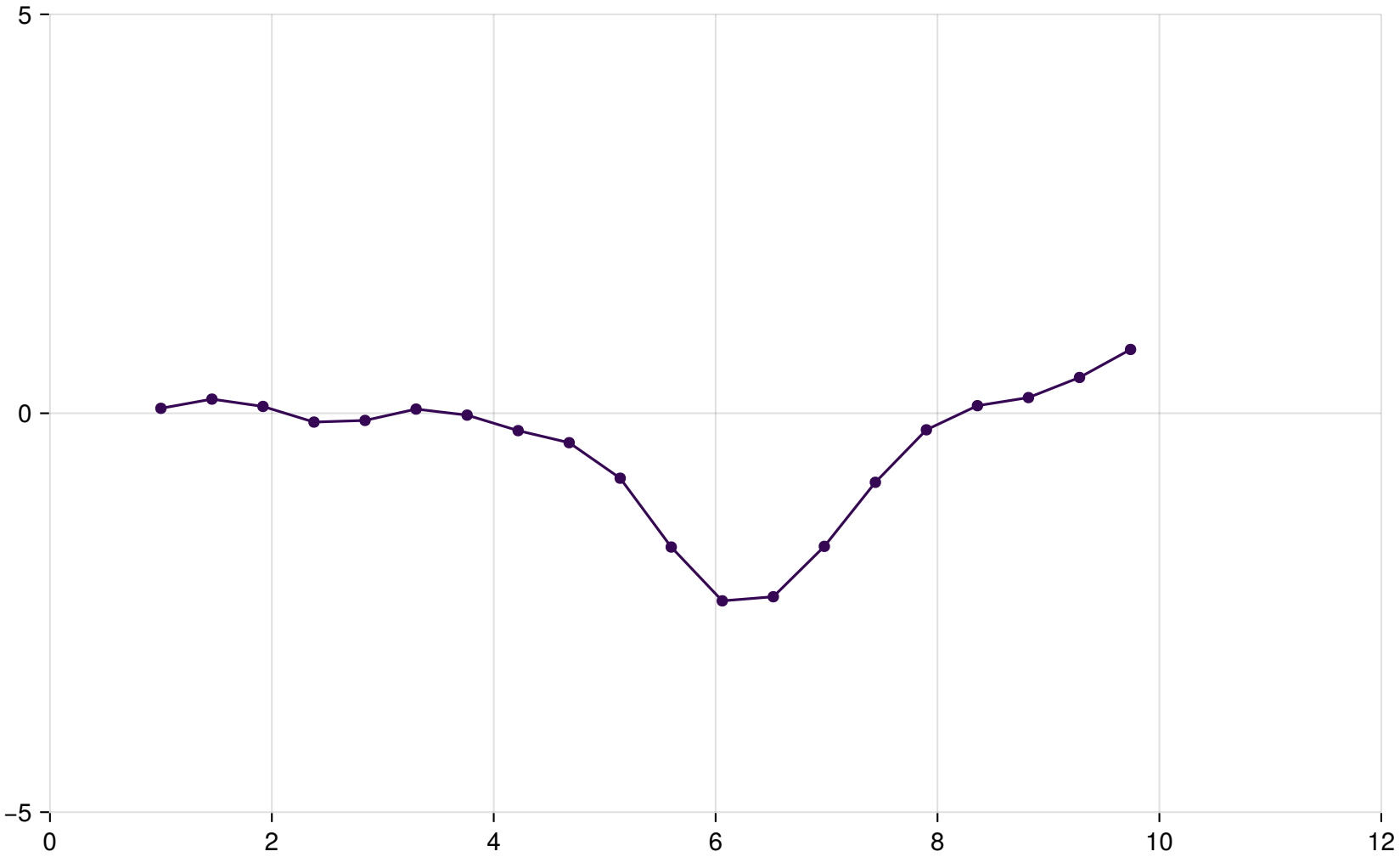


Gaussian Process

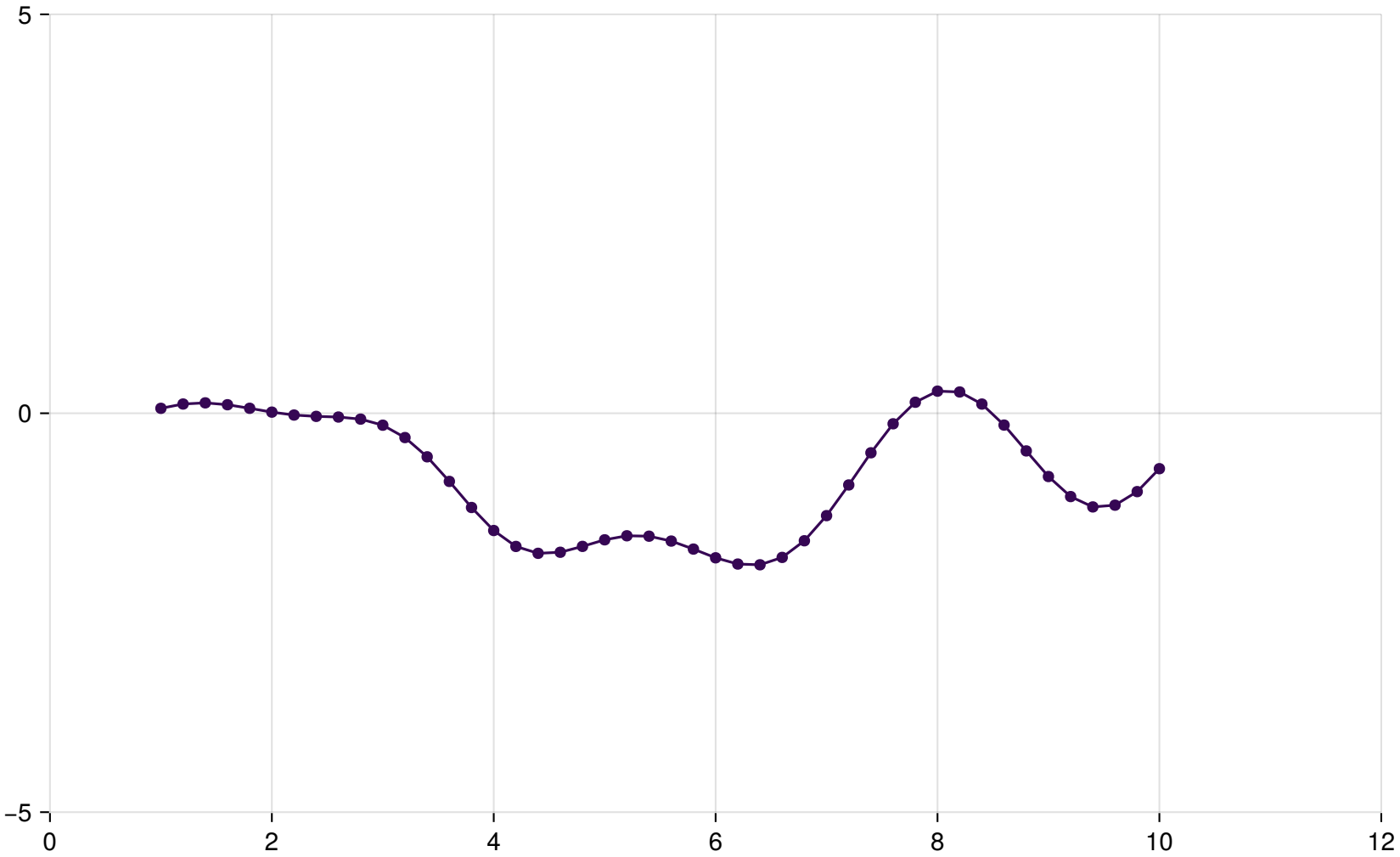
Instead of using $\Sigma = I$, we can assume closer points should have similar outputs to achieve continuous functions

$$\Sigma(x, x') = k(x, x') = \exp(-||x - x'||^2)$$

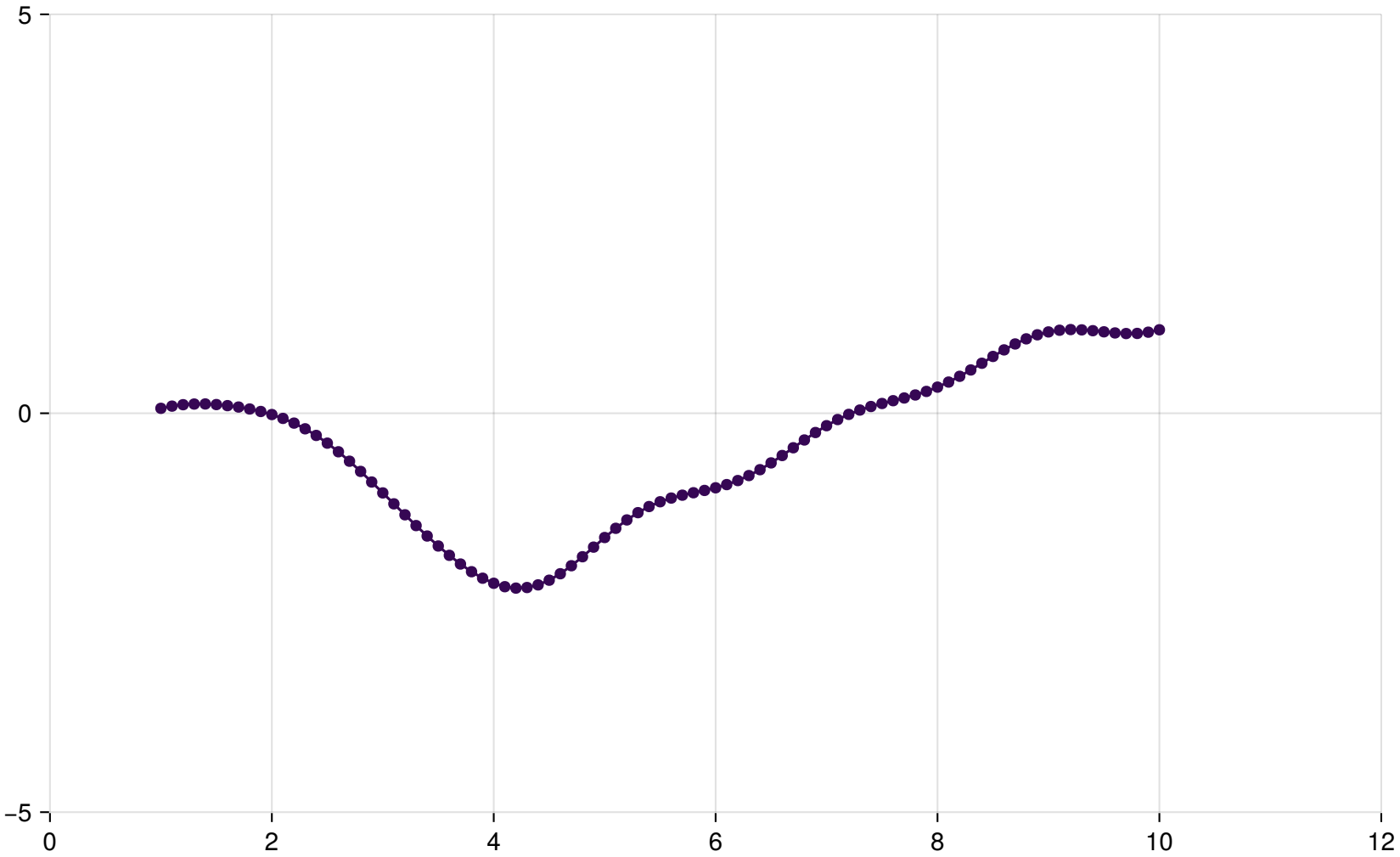
Gaussian Process



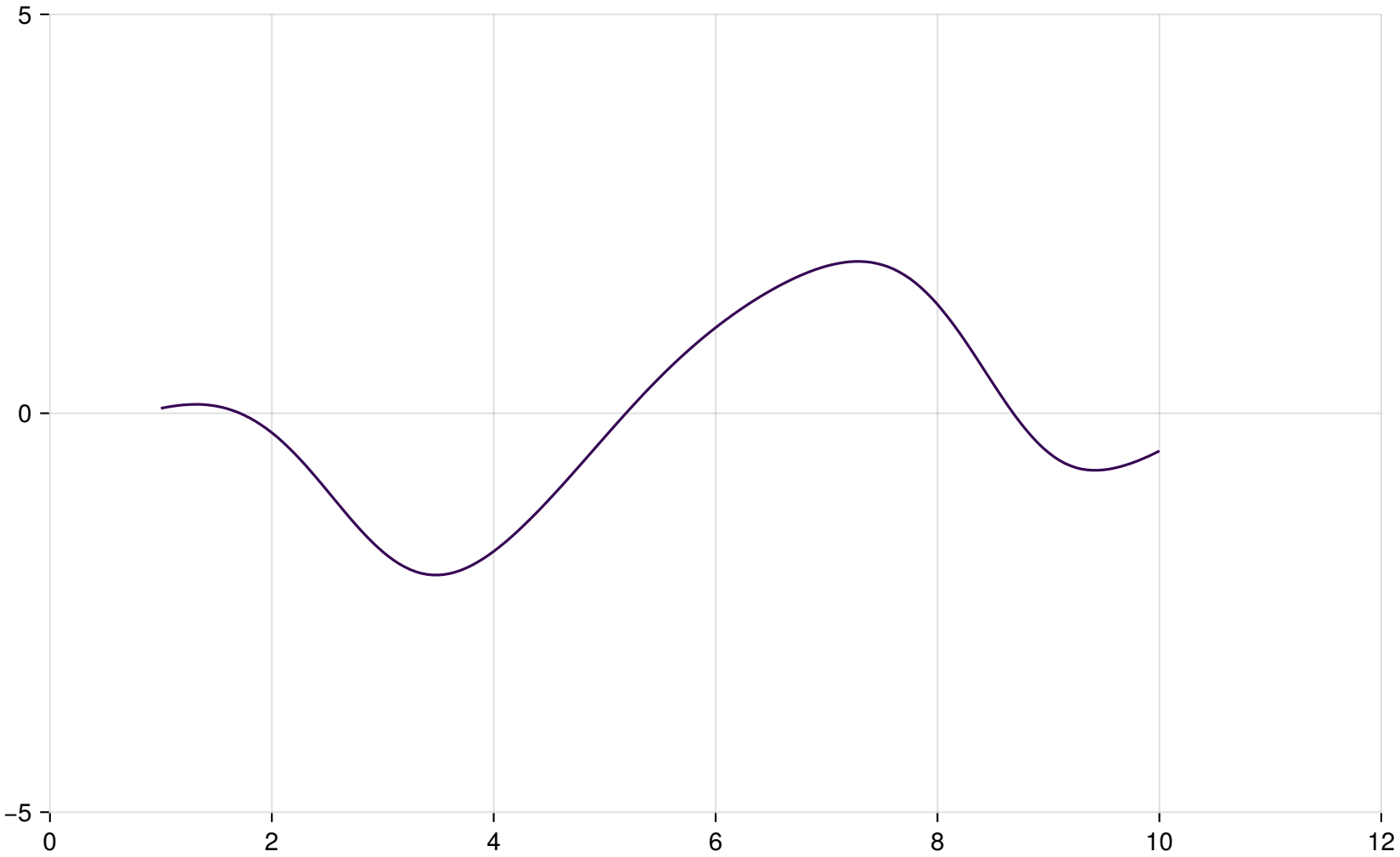
Gaussian Process



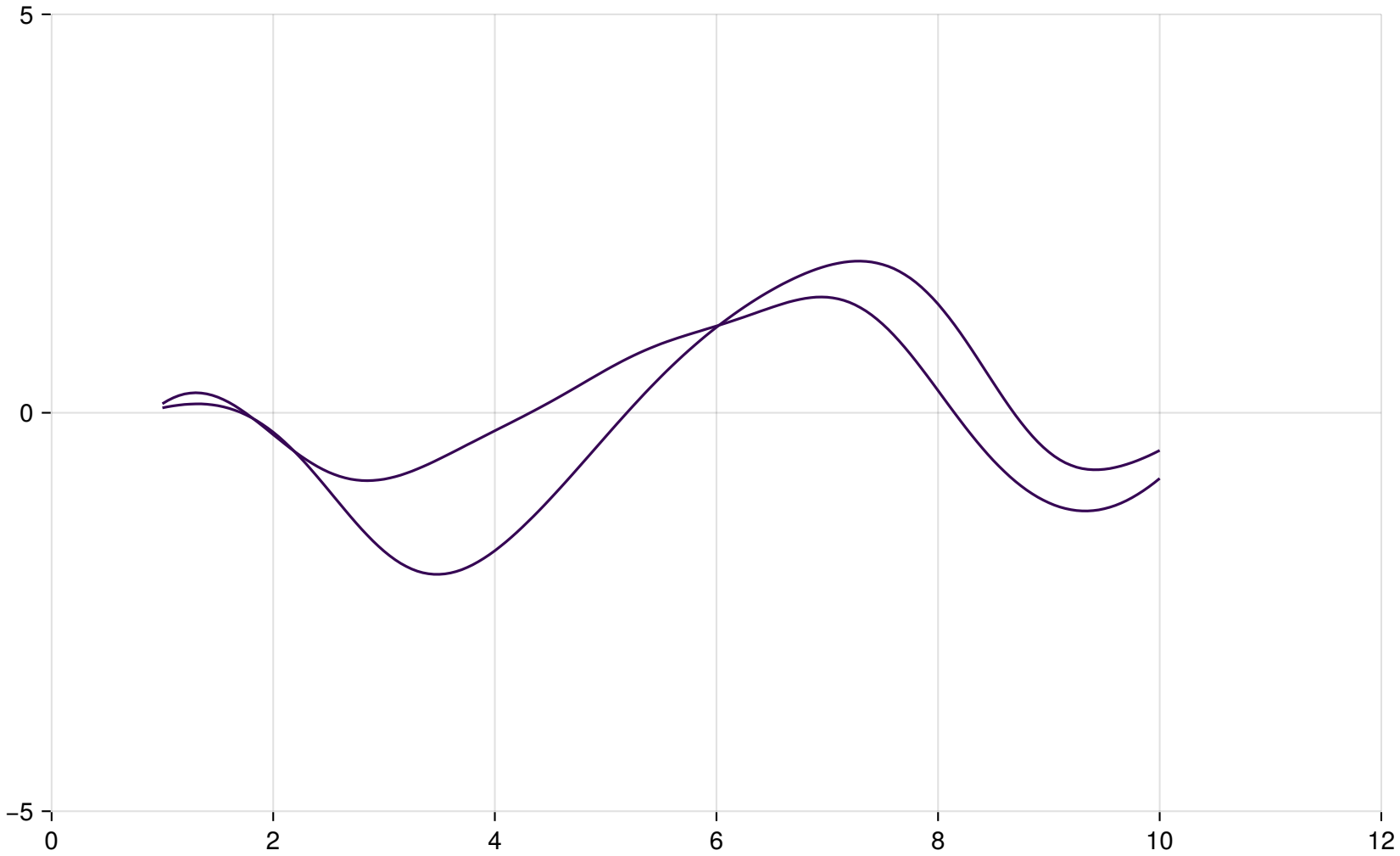
Gaussian Process



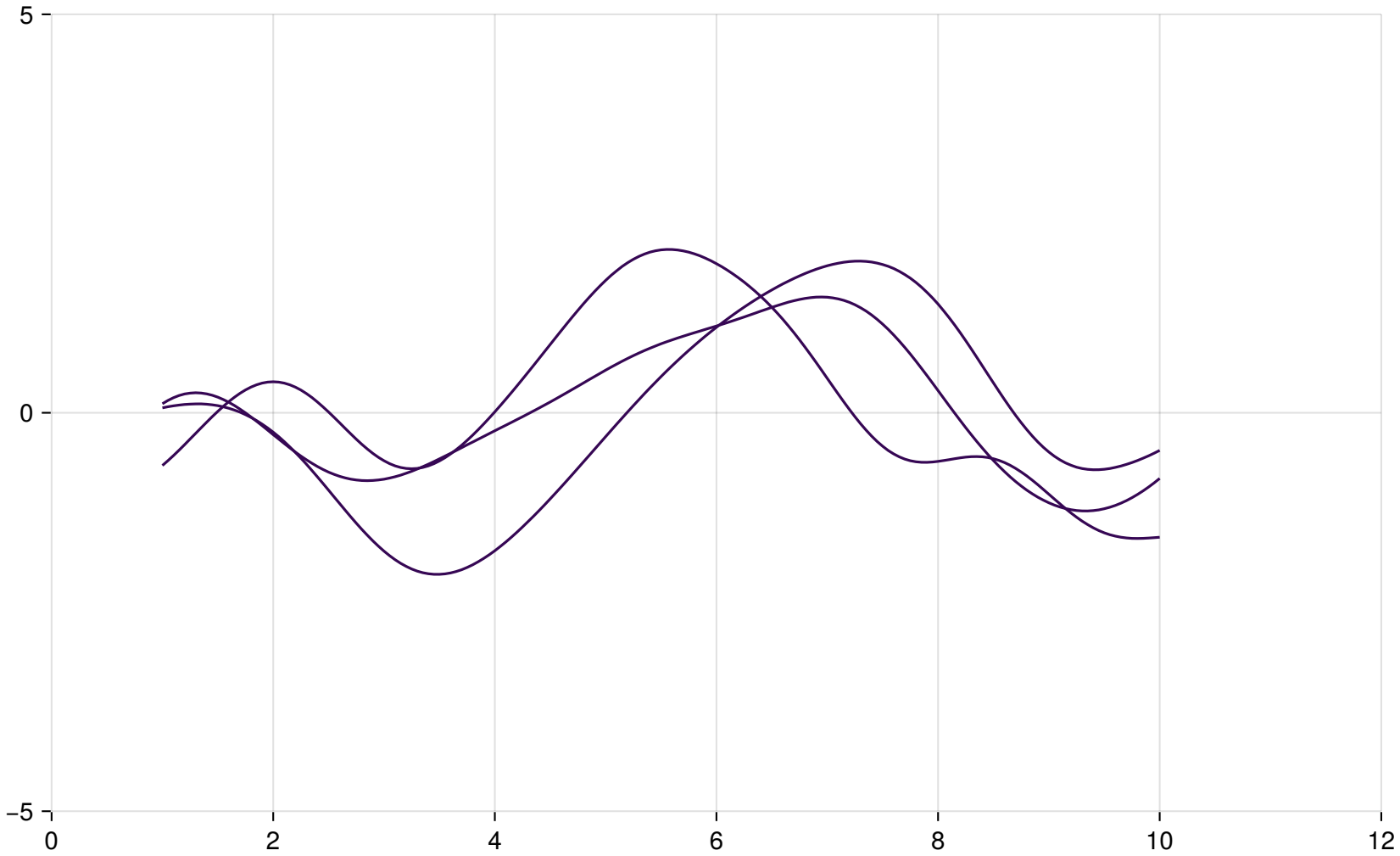
Gaussian Process



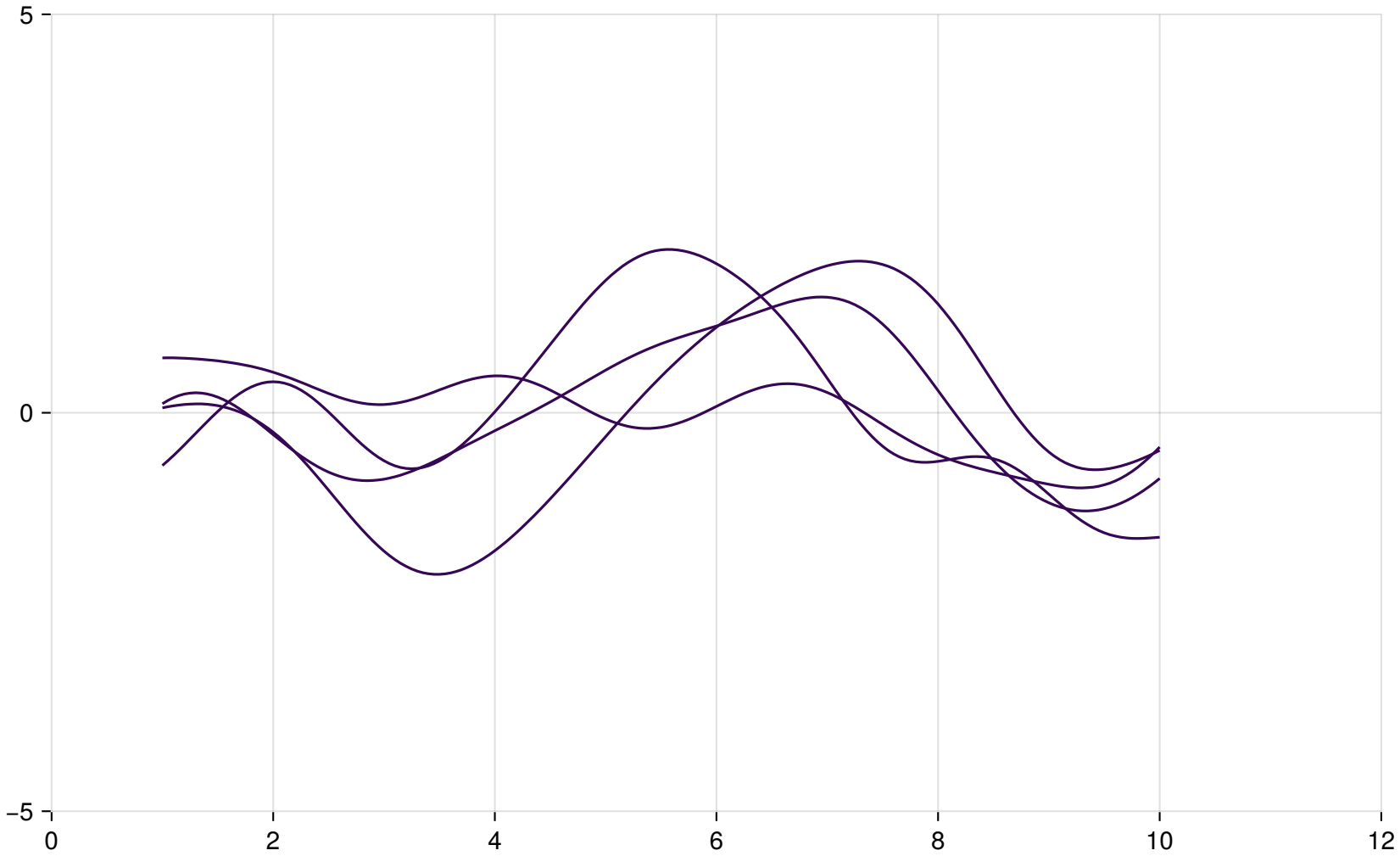
Gaussian Process



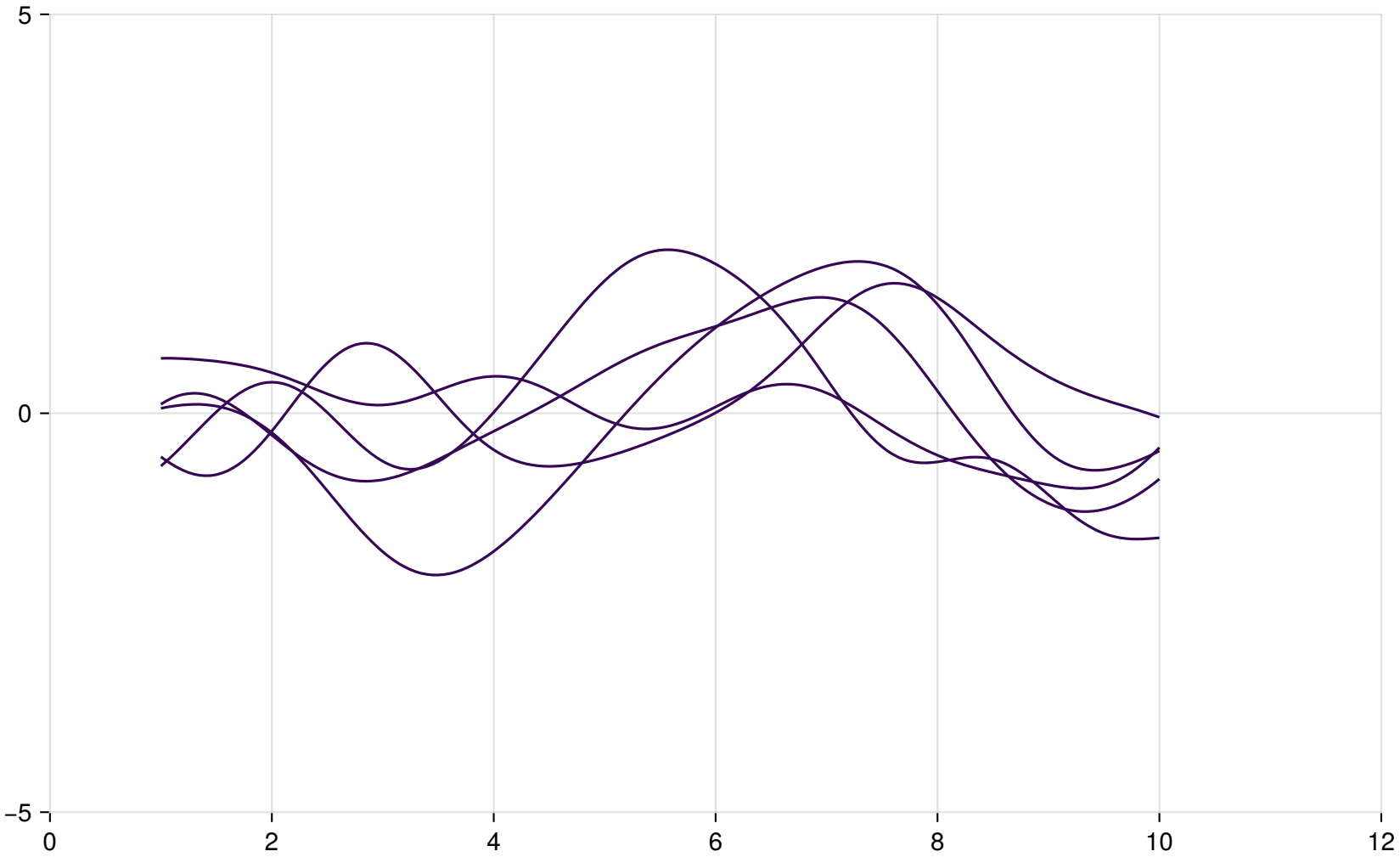
Gaussian Process



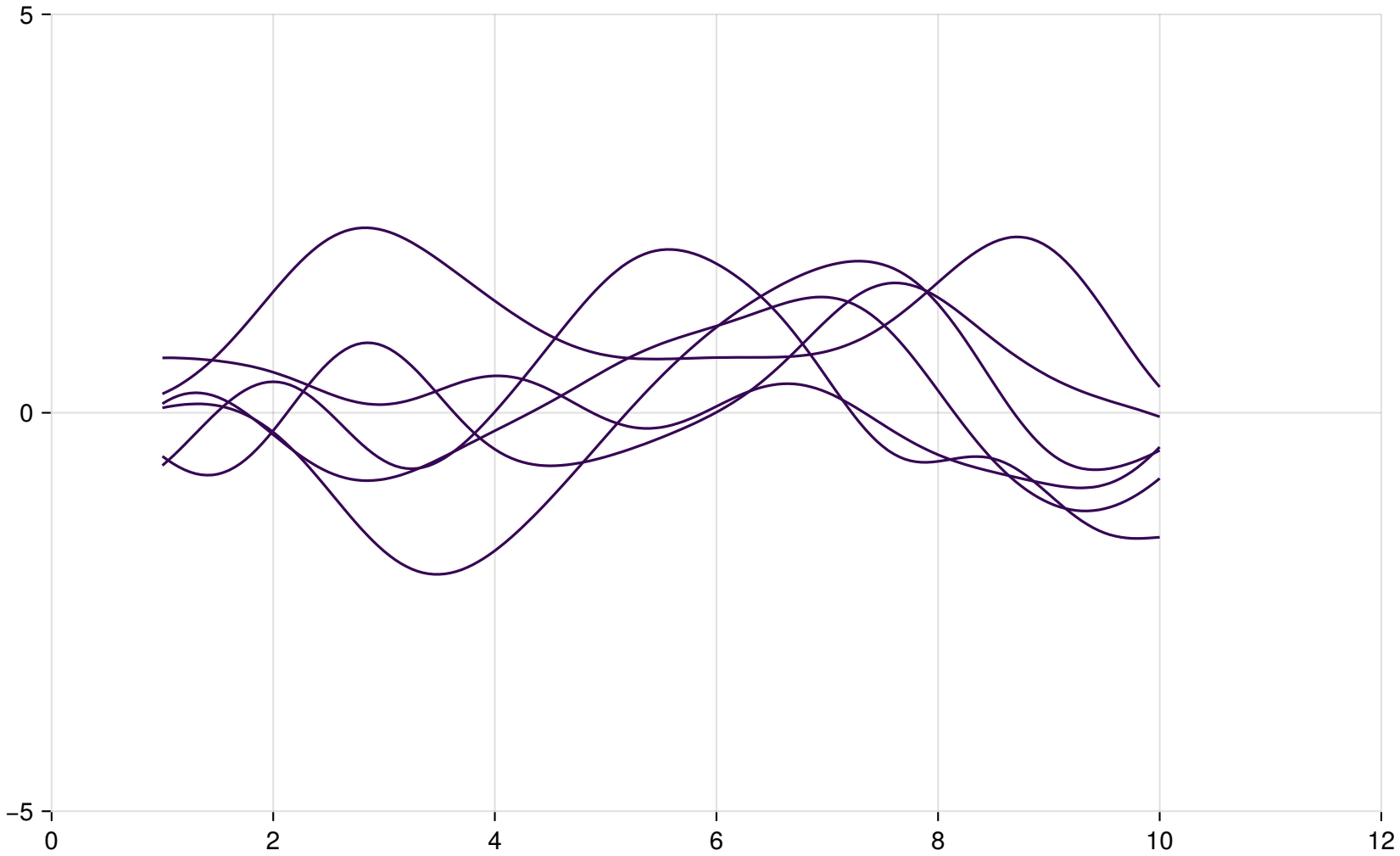
Gaussian Process



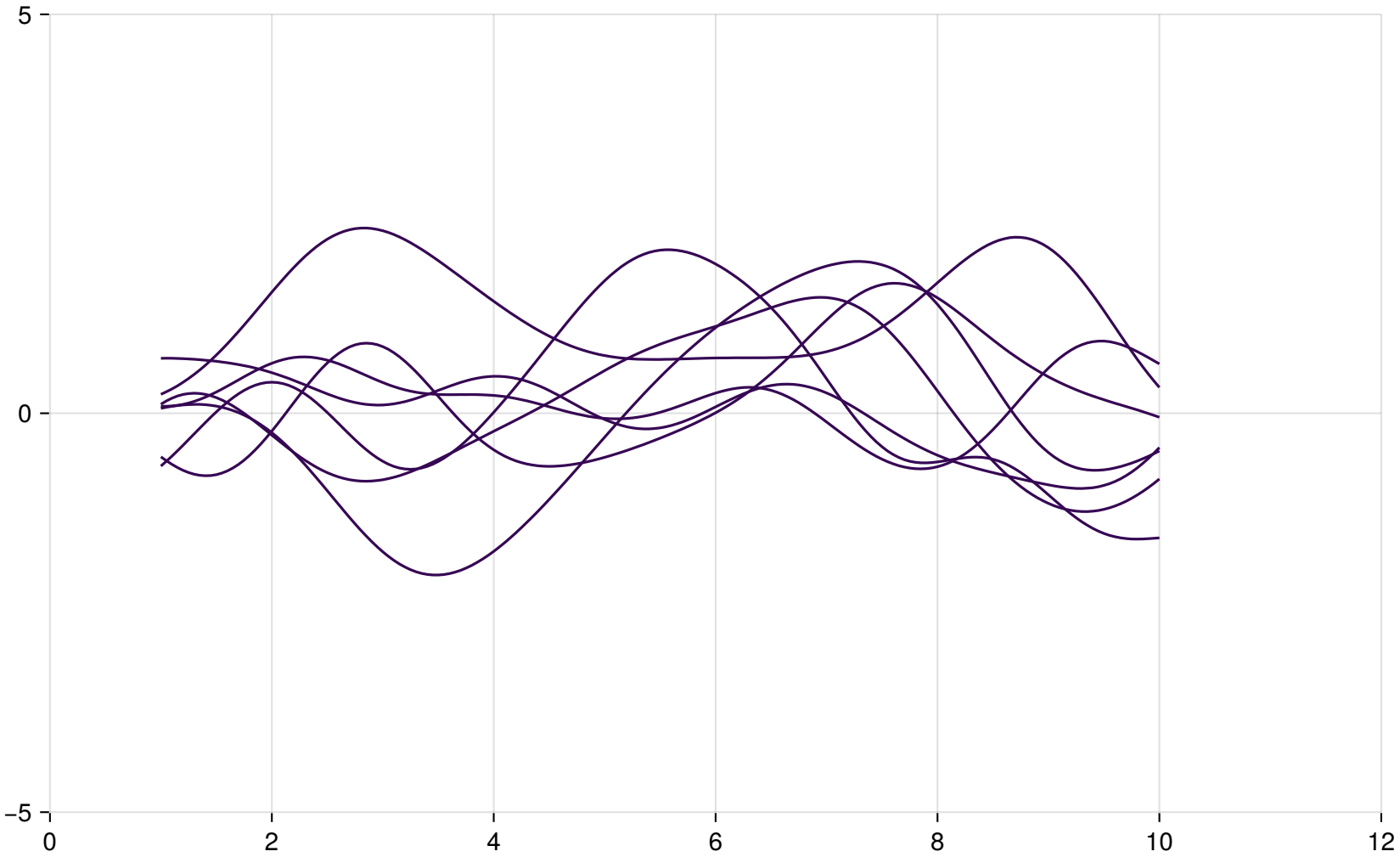
Gaussian Process



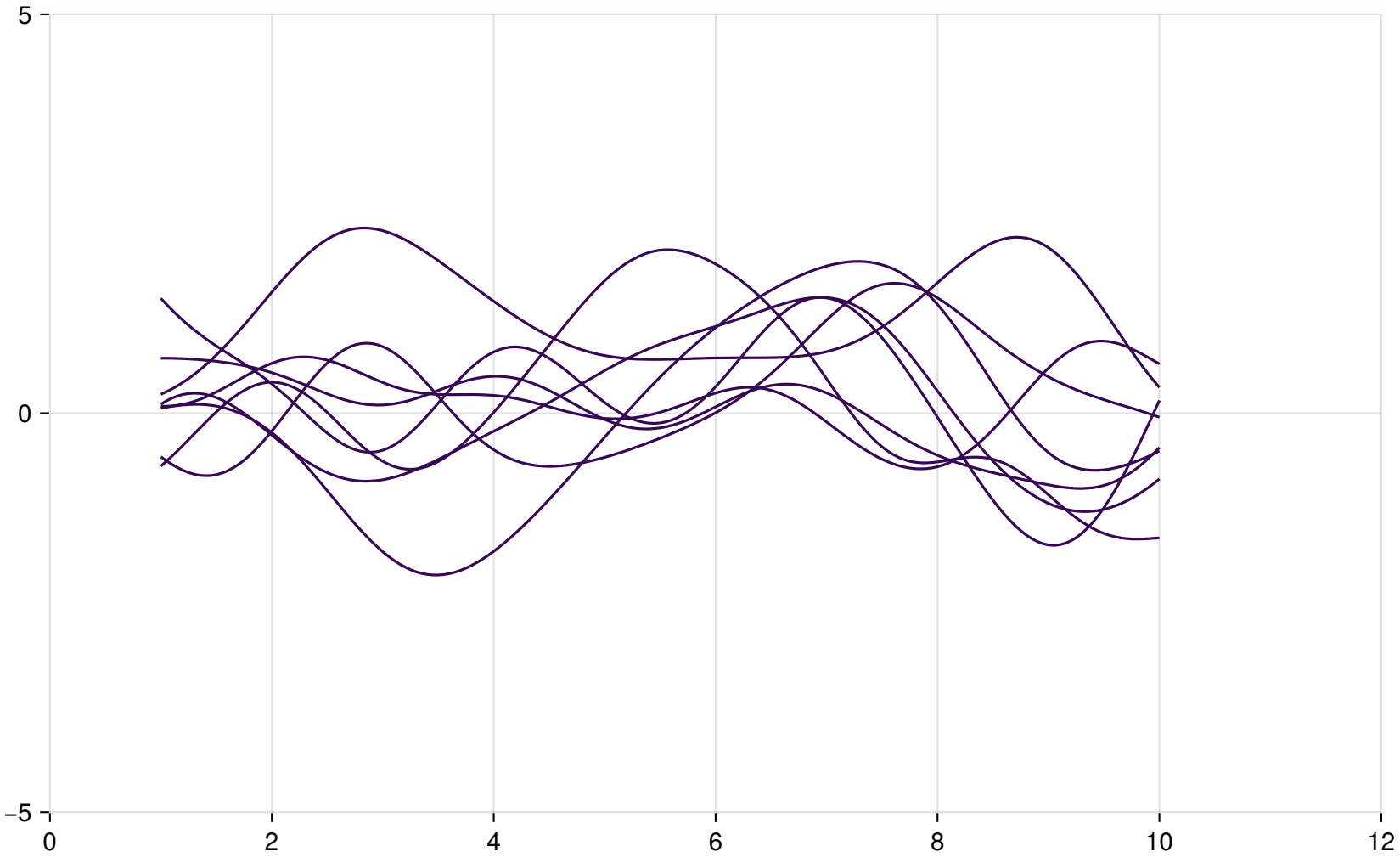
Gaussian Process



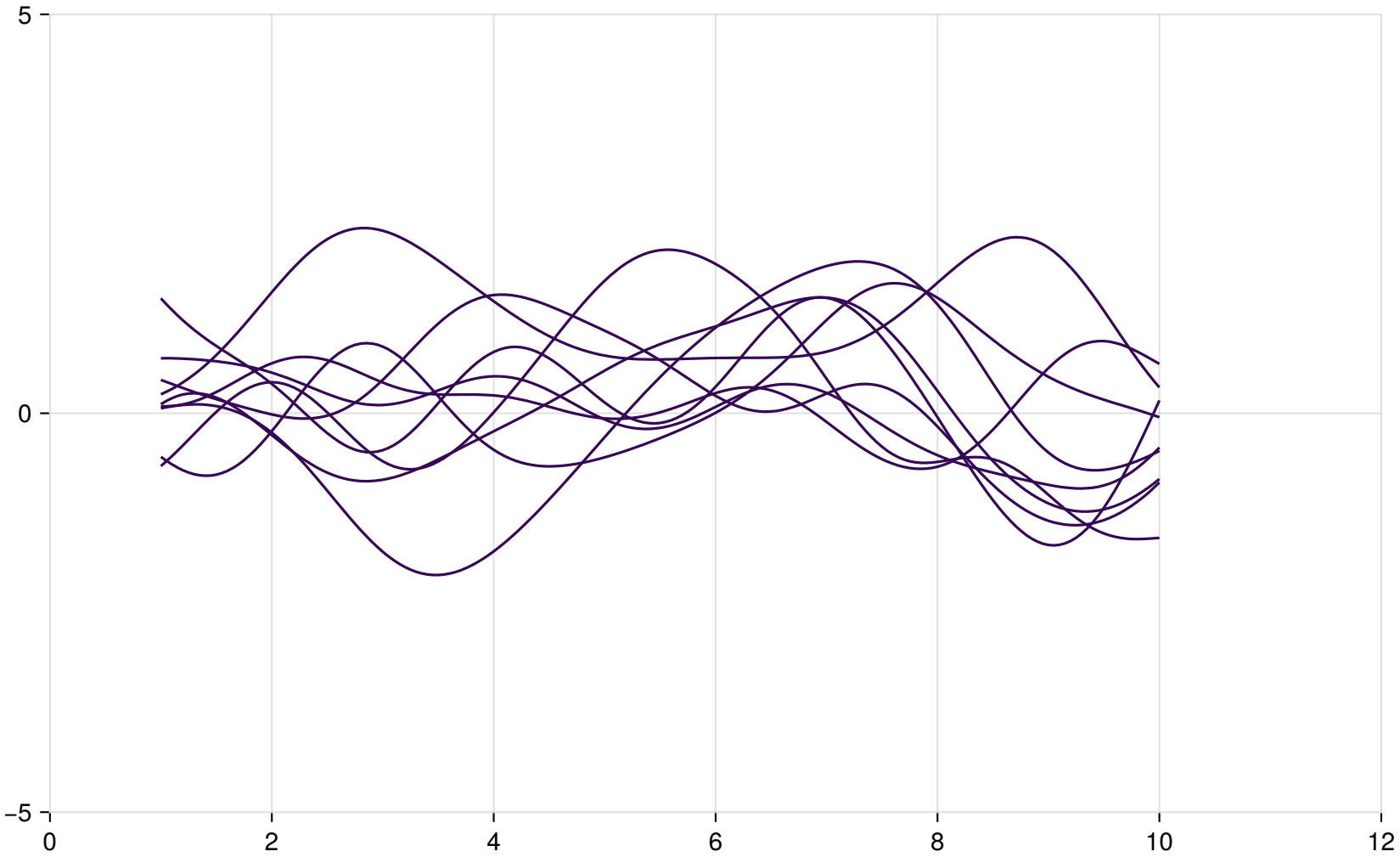
Gaussian Process



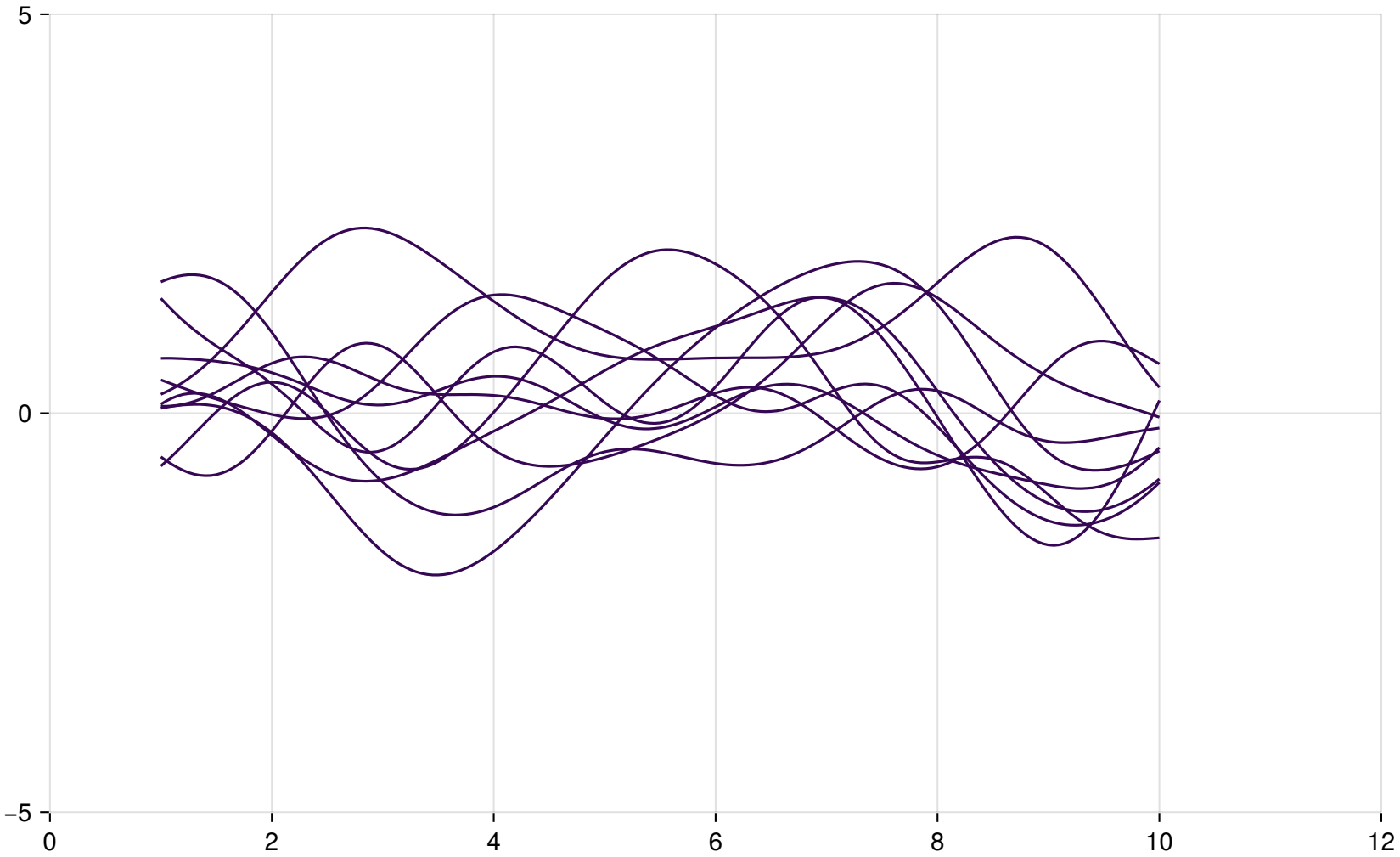
Gaussian Process



Gaussian Process



Gaussian Process



Gaussian Process Inference

$$f \sim \mathrm{Gp}(\mathbf{0}, \Sigma)$$

How to find the posterior given a set of observed data points?

Gaussian Process Inference

The basic inference method for GP is simply as same as conditioning a multivariate normal distribution:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathrm{N}(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix})$$

Gaussian Process Inference

The basic inference method for GP is simply as same as conditioning a multivariate normal distribution:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathrm{N}(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix})$$

$$\begin{aligned} p(y_1 | y_2 = a) &= \mathrm{N}(\hat{\mu}, \hat{\Sigma}) \\ \hat{\mu} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(a - \mu_2) \\ \hat{\Sigma} &= \underbrace{\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}}_{\text{Schur complement}(\Sigma/\Sigma_{22})} \end{aligned}$$

Gaussian Process Inference

We can use the conditioning formula for GP

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathrm{N}(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix})$$

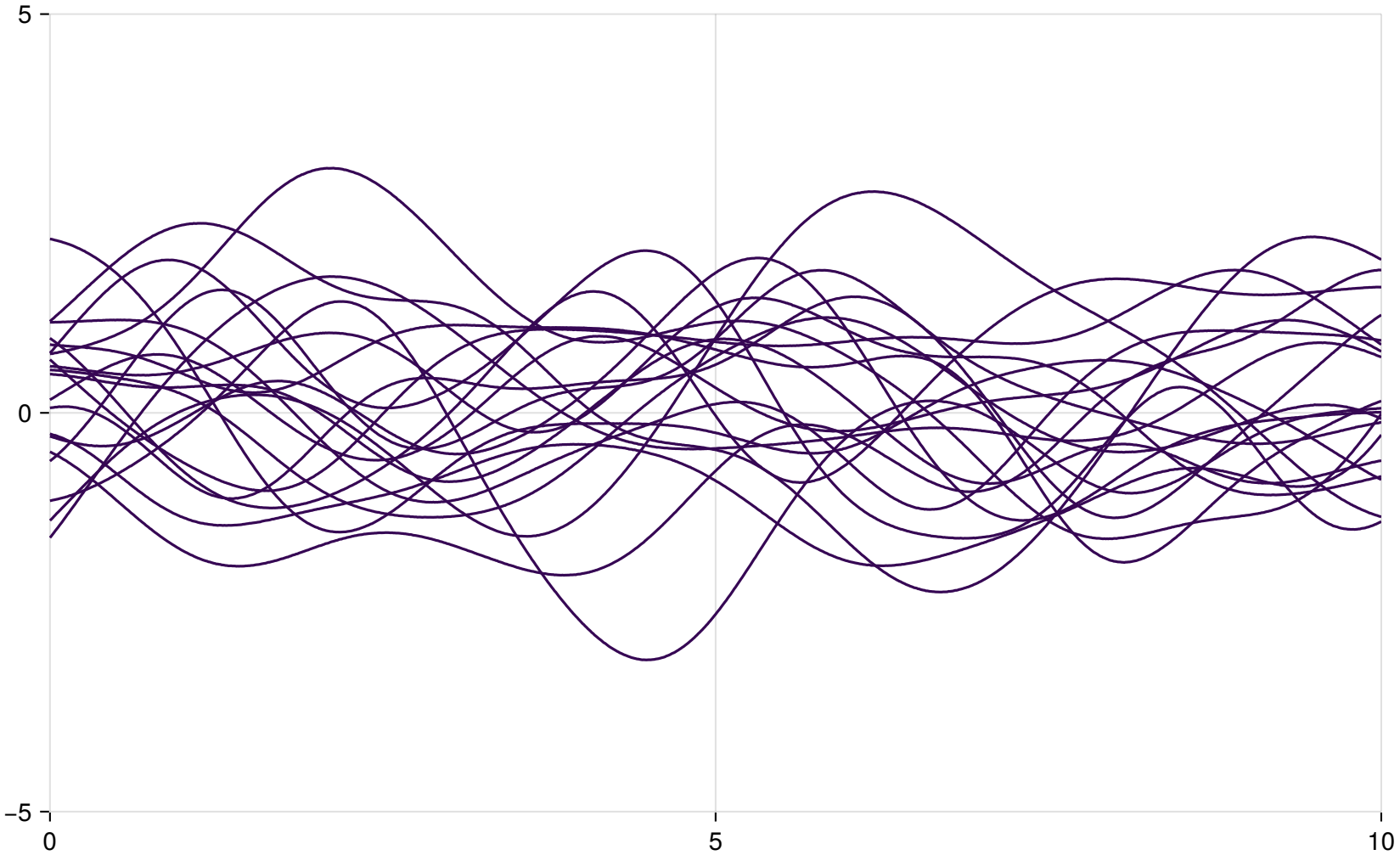
Gaussian Process Inference

We can use the conditioning formula for GP

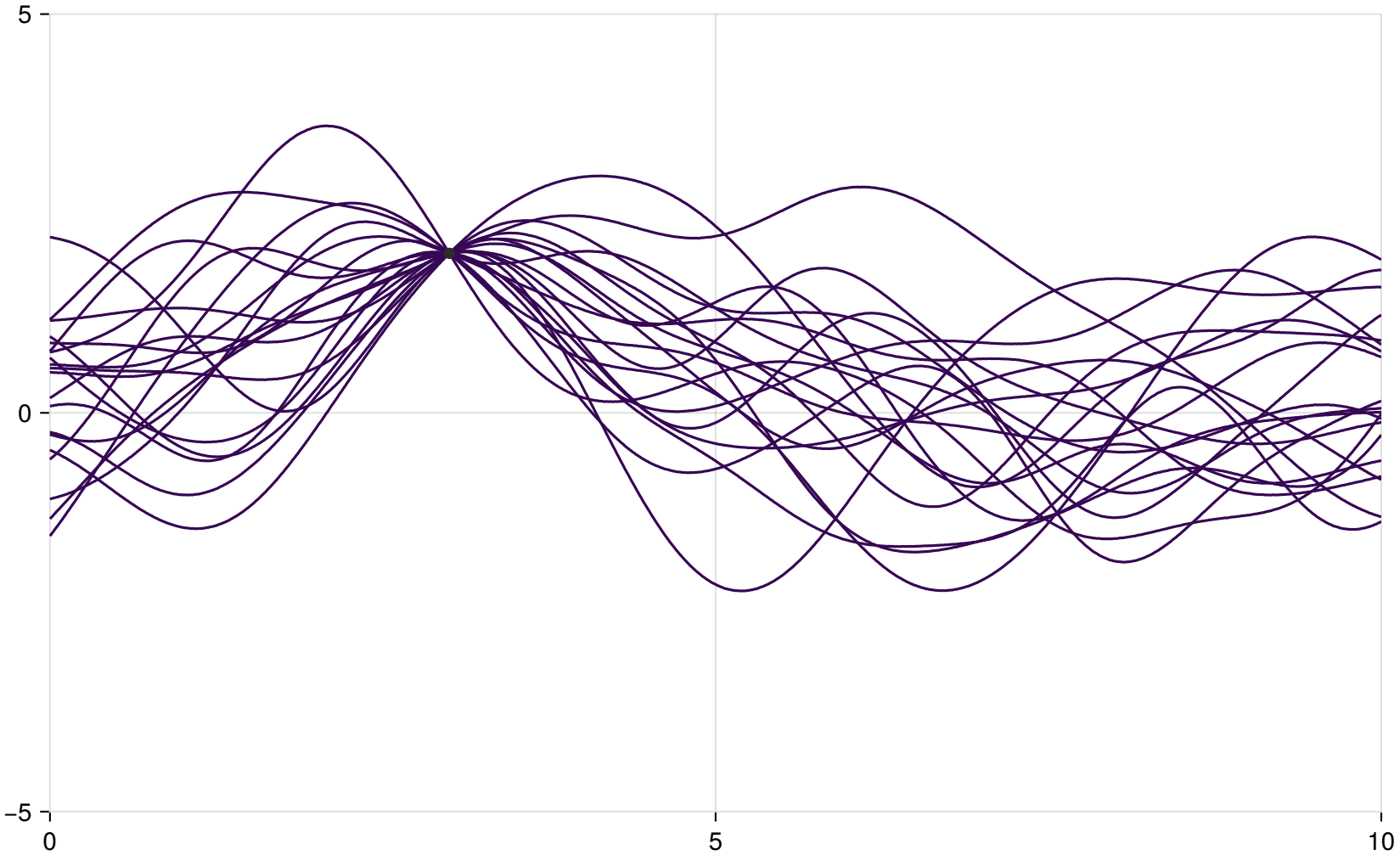
$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathrm{N}(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix})$$

$$\begin{aligned} p(f_*|X_*, X, f) &= \mathrm{N}(f_*|\mu_*, \Sigma_*) \\ \mu_* &= \mu(X_*) + K_*^T K^{-1}(f - \mu(X)) \\ \Sigma_* &= K_{**} - K_*^T K^{-1} K_* \end{aligned}$$

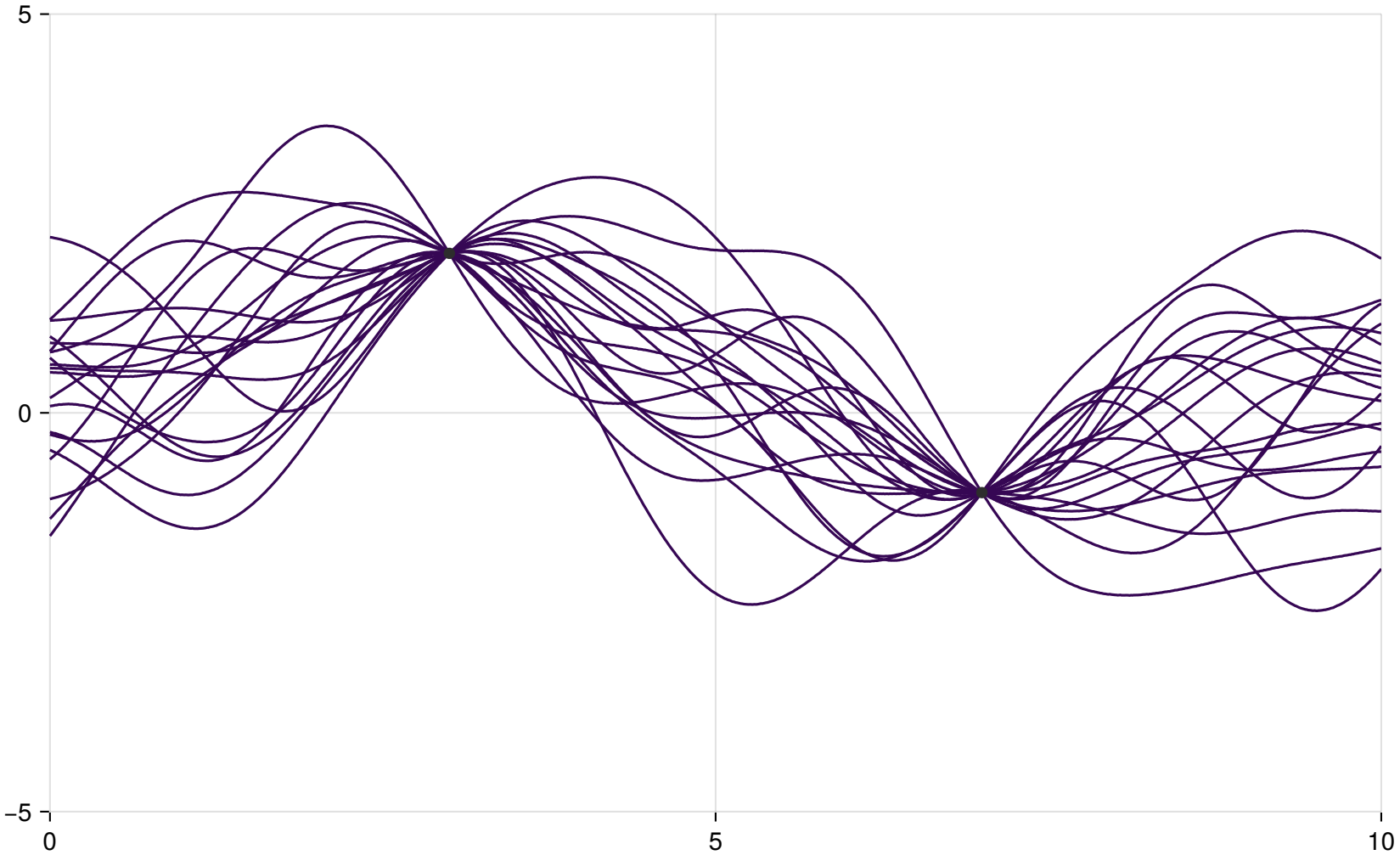
Gaussian Process Inference



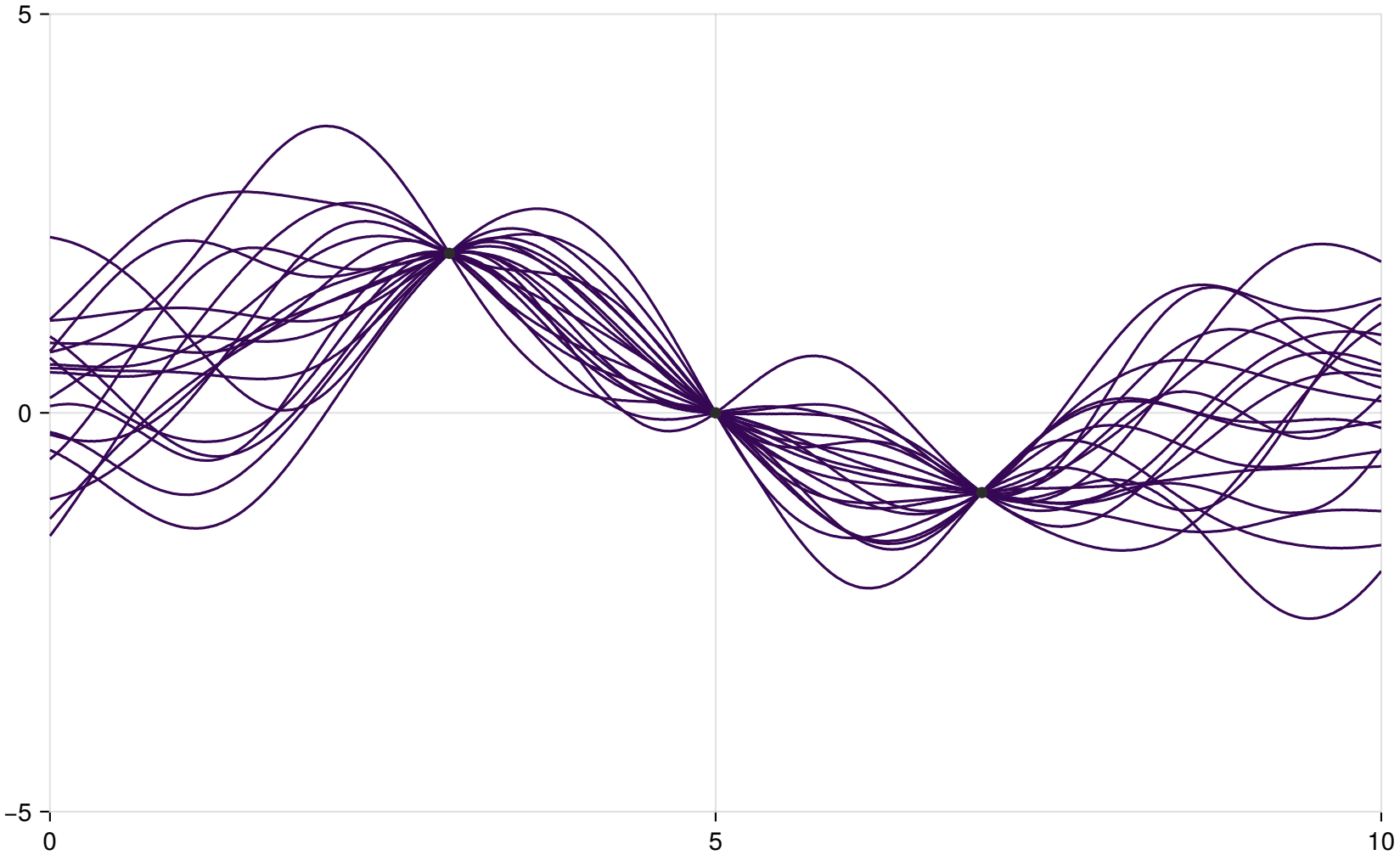
Gaussian Process Inference



Gaussian Process Inference



Gaussian Process Inference



Kernel Design

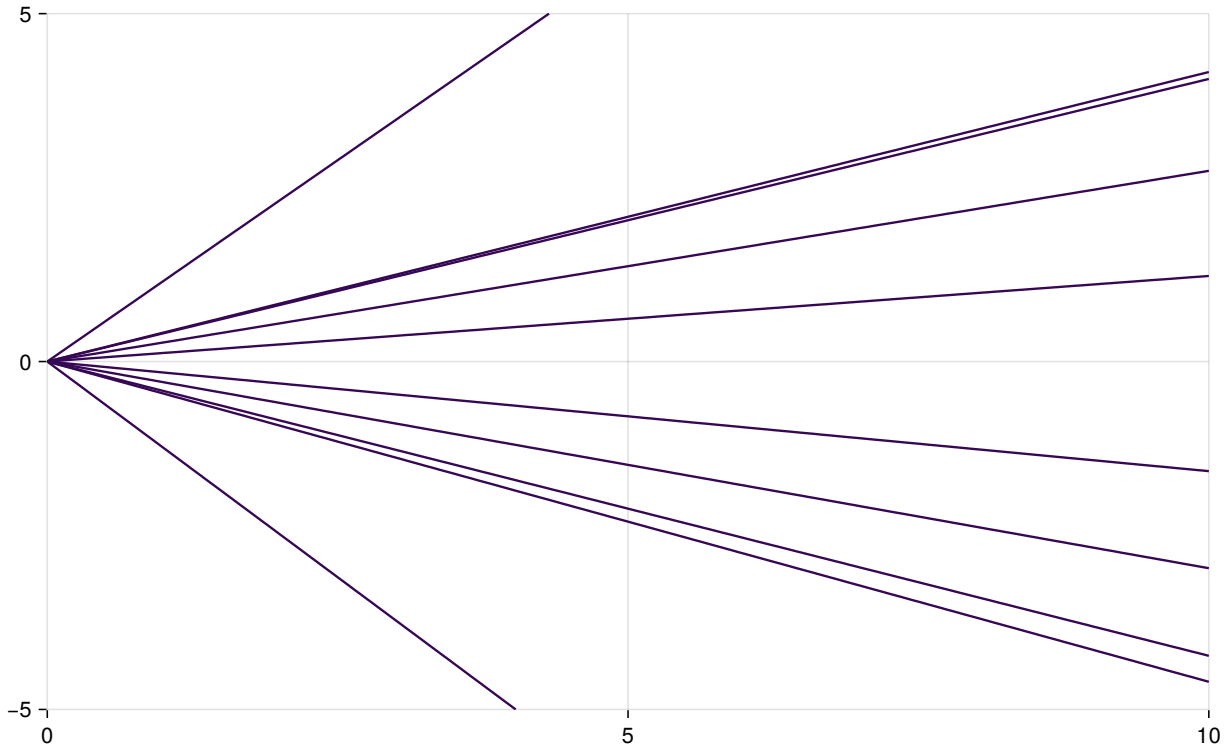
Linear kernel

$$k(x, x'; c) = x^\top x' + c$$

Kernel Design

Linear kernel

$$k(x, x'; c) = x^\top x' + c$$



Kernel Design

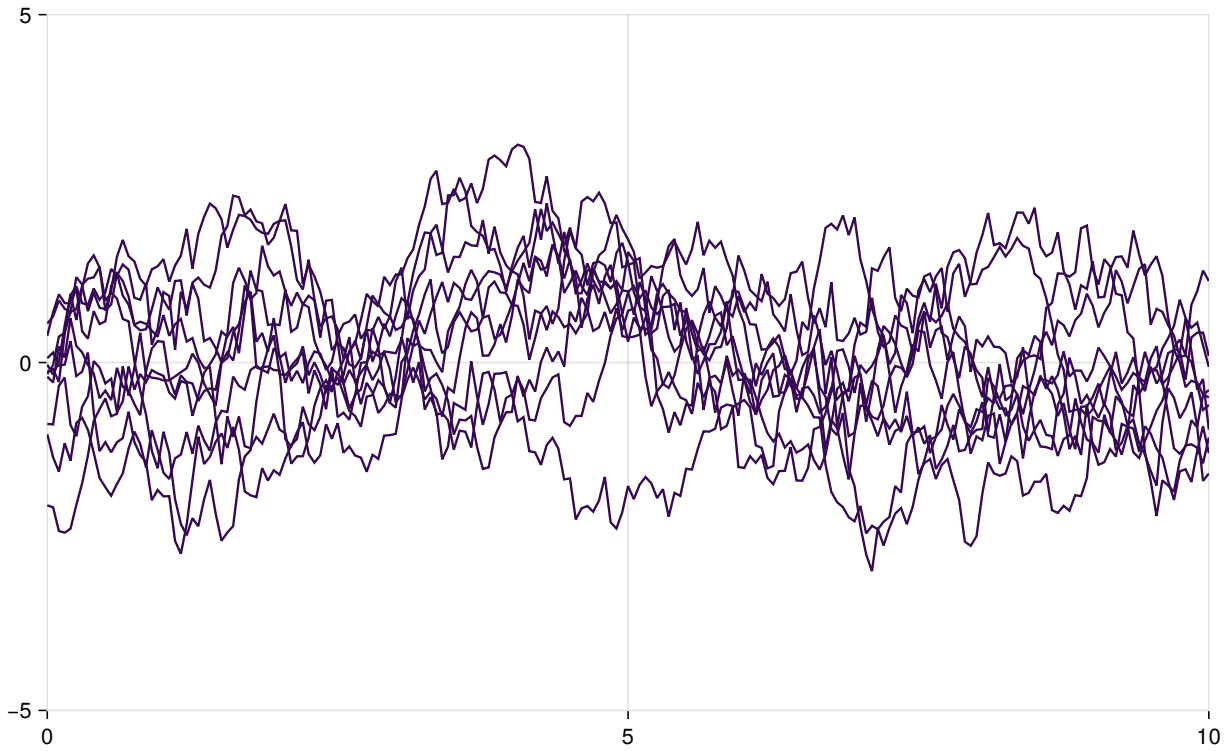
Exponential kernel

$$k(x, x') = \exp(x^\top x')$$

Kernel Design

Exponential kernel

$$k(x, x') = \exp(x^\top x')$$



Kernel Design

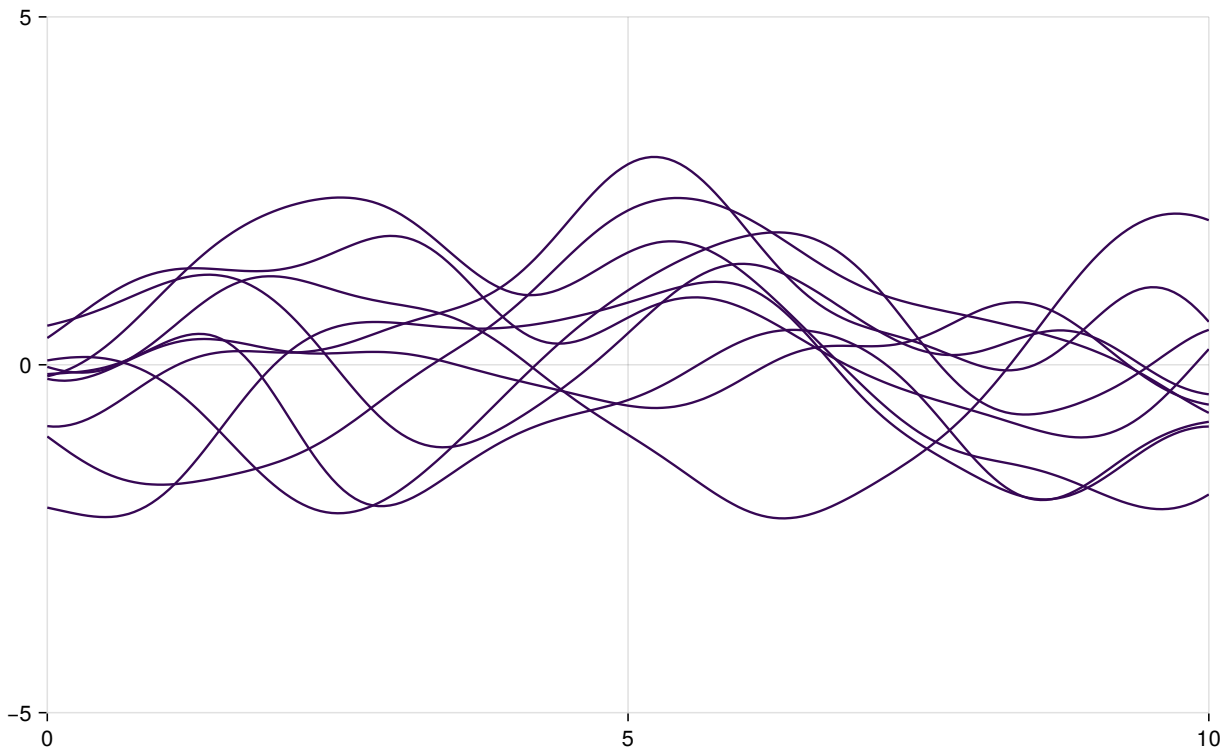
Squared-exponential kernel

$$k(x, x') = \exp\left(-\frac{d(x, x')^2}{2}\right)$$

Kernel Design

Squared-exponential kernel

$$k(x, x') = \exp\left(-\frac{d(x, x')^2}{2}\right)$$



Kernel Design

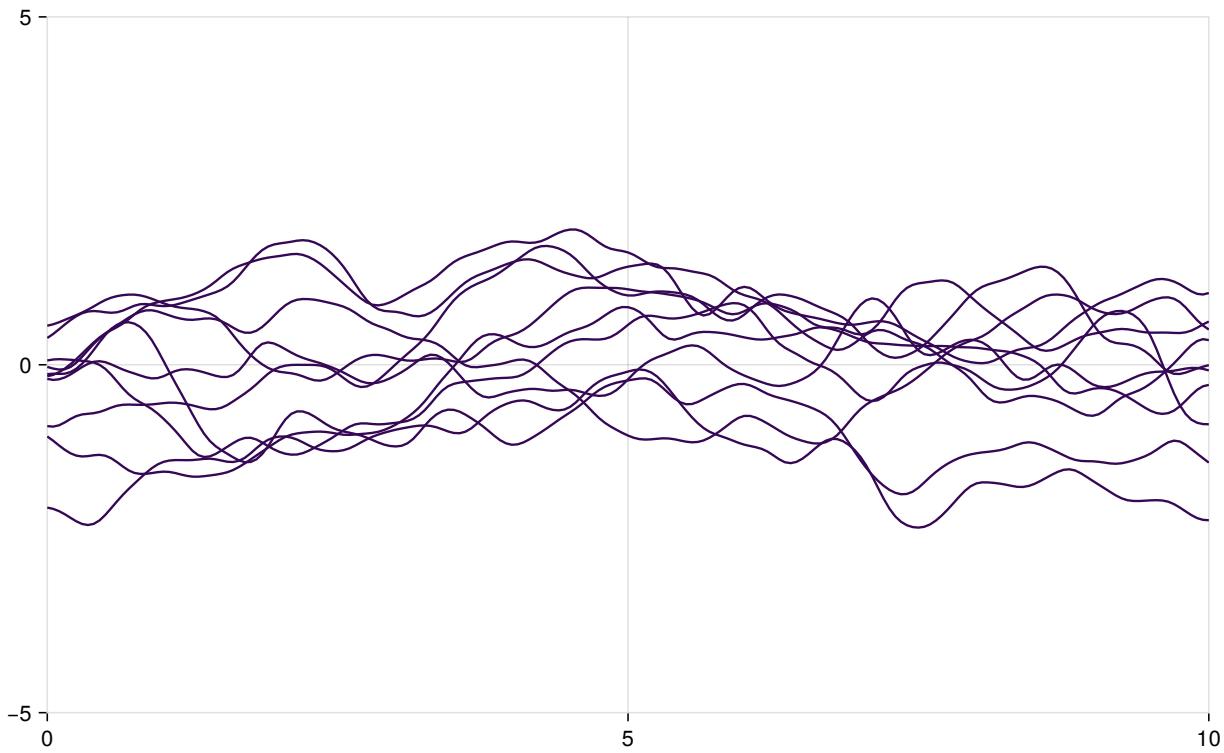
Rational-quadratic kernel

$$k(x, x'; \alpha) = \left(1 + \frac{d(x, x')^2}{2\alpha}\right)^{-\alpha}$$

Kernel Design

Rational-quadratic kernel

$$k(x, x'; \alpha) = \left(1 + \frac{d(x, x')^2}{2\alpha}\right)^{-\alpha}$$



Kernel Design

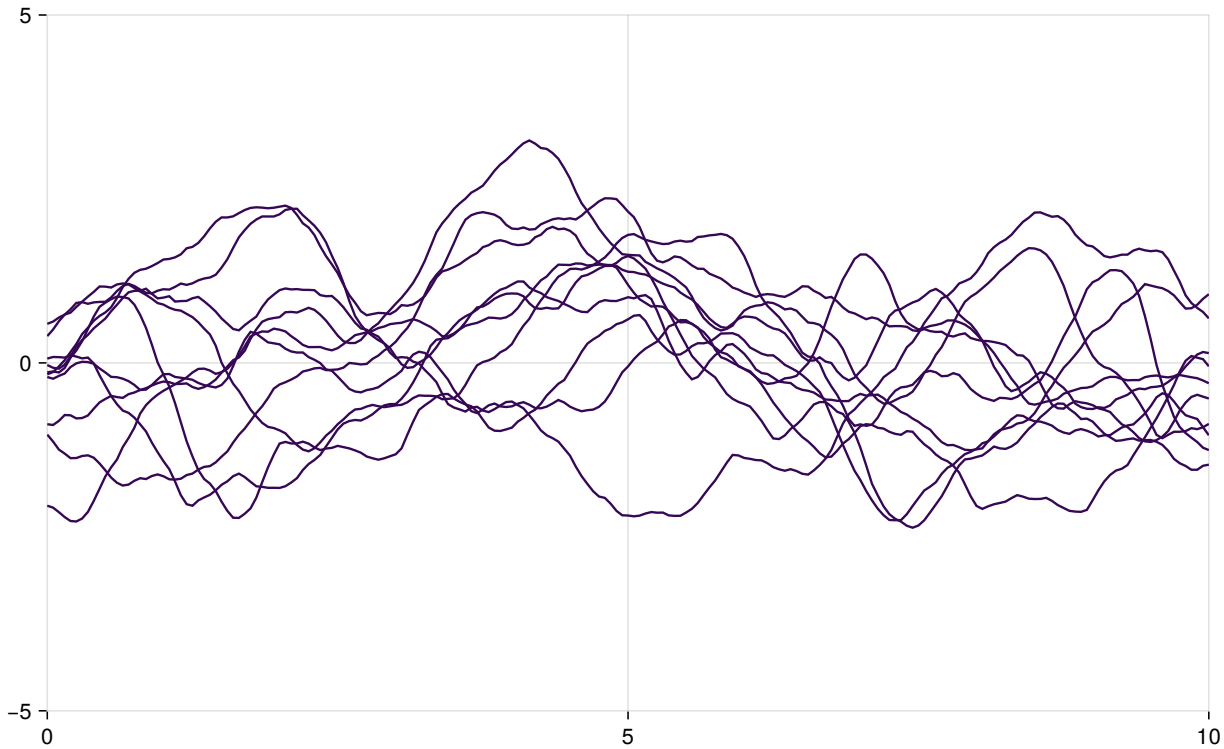
Matérn kernel

$$k(x, x'; \nu) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}d(x, x'))K_\nu(\sqrt{2\nu}d(x, x'))$$

Kernel Design

Matérn kernel

$$k(x, x'; \nu) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}d(x, x'))K_\nu(\sqrt{2\nu}d(x, x'))$$



Kernel Design

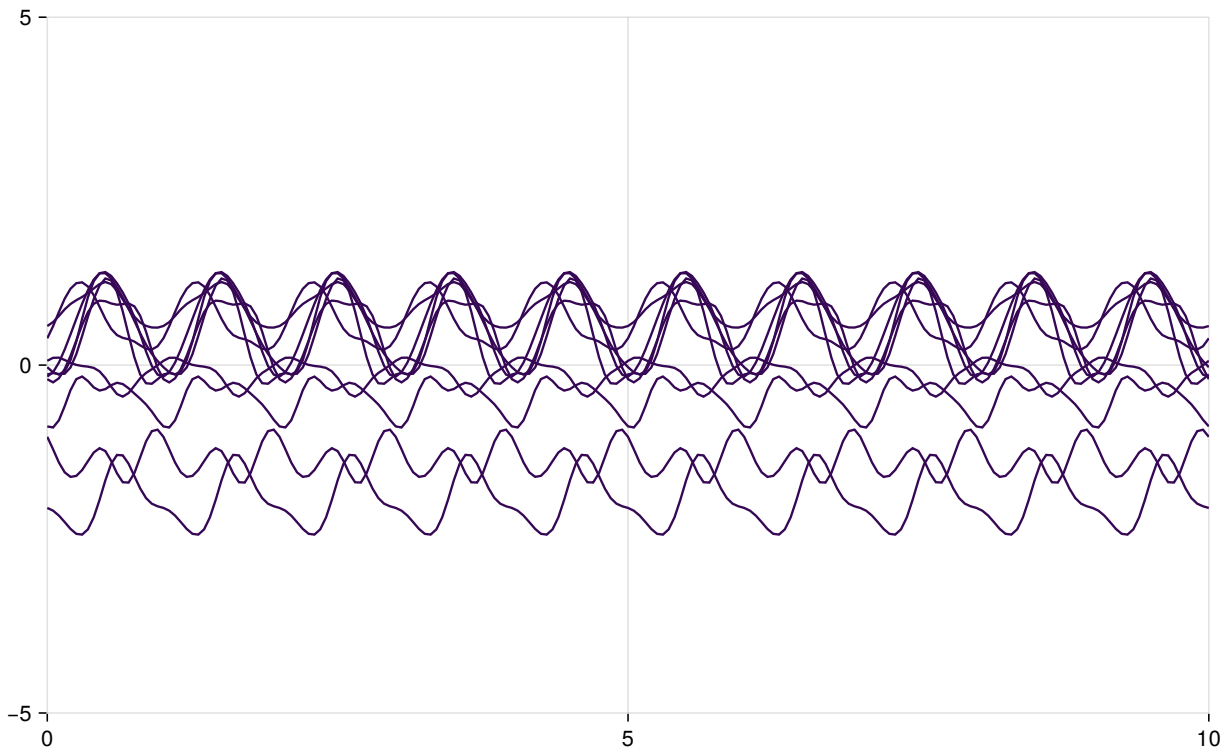
Periodic kernel

$$k(x, x'; r) = \exp\left(-\frac{1}{2} \sum_{i=1}^{d} \left(\frac{\sin\left(\pi(x_i - x_i')\right)}{r_i} \right)^2 \right)$$

Kernel Design

Periodic kernel

$$k(x, x'; r) = \exp\left(-\frac{1}{2} \sum_{i=1}^{d} \left(\frac{\sin\left(\pi(x_i - x_i')\right)}{r_i} \right)^2 \right)$$



Kernel Design

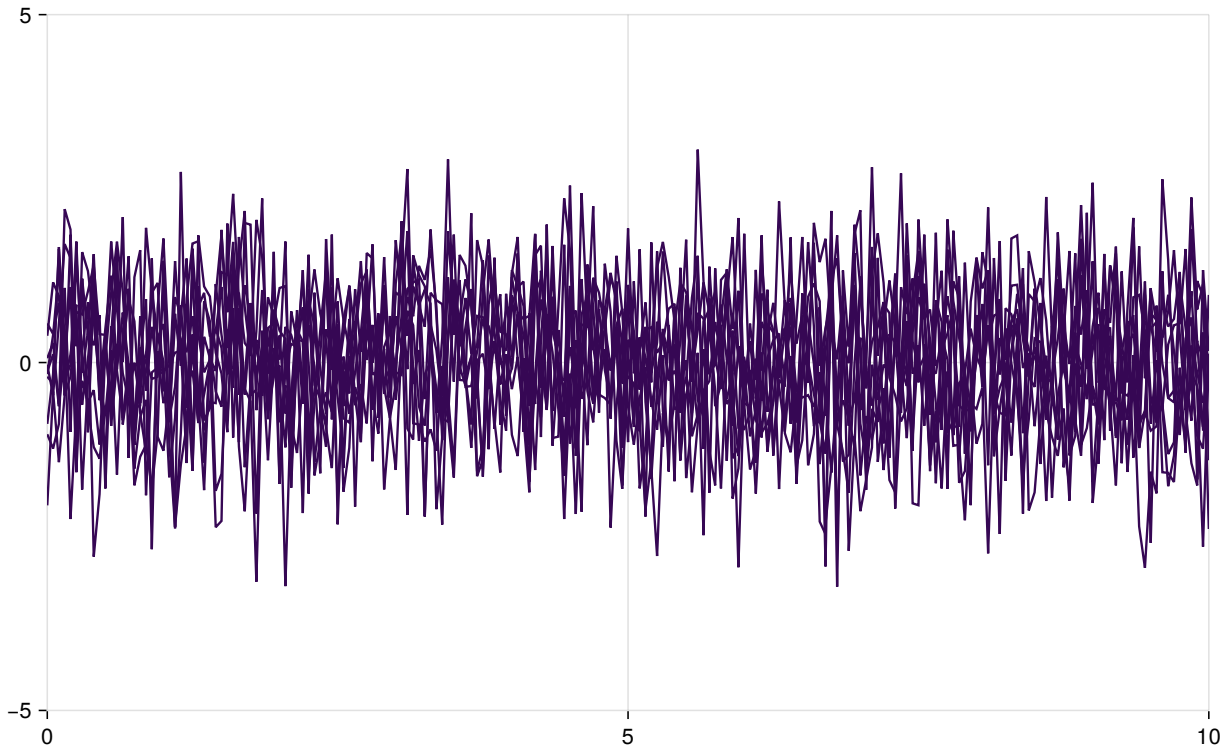
White noise kernel

$$k(x, x') = \min(x, x')$$

Kernel Design

White noise kernel

$$k(x, x') = \min(x, x')$$



Kernel Design

Deriving new kernels using the existing ones

- Kernel transformation
- Sum of kernels
- Product of kernels

Kernel Design

In Julia you can use `KernelFunctions.jl` to use famous kernels and to combine them

```
using KernelFunctions
```

Define a kernel

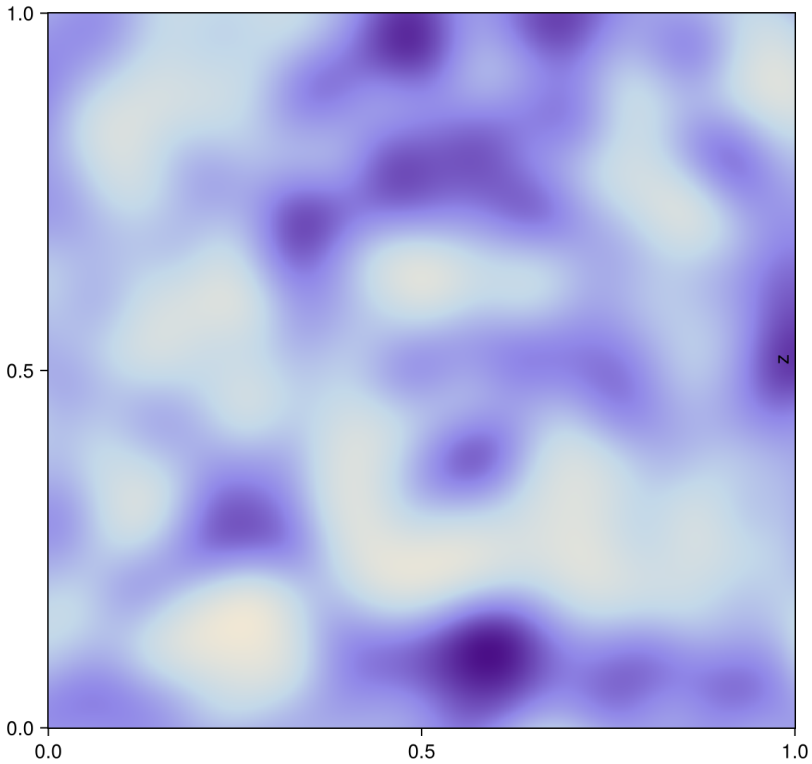
```
k1 = SqExponentialKernel ∘ ScaleTransform(2.0)
k2 = PeriodicKernel()
k = k1 + k2
```

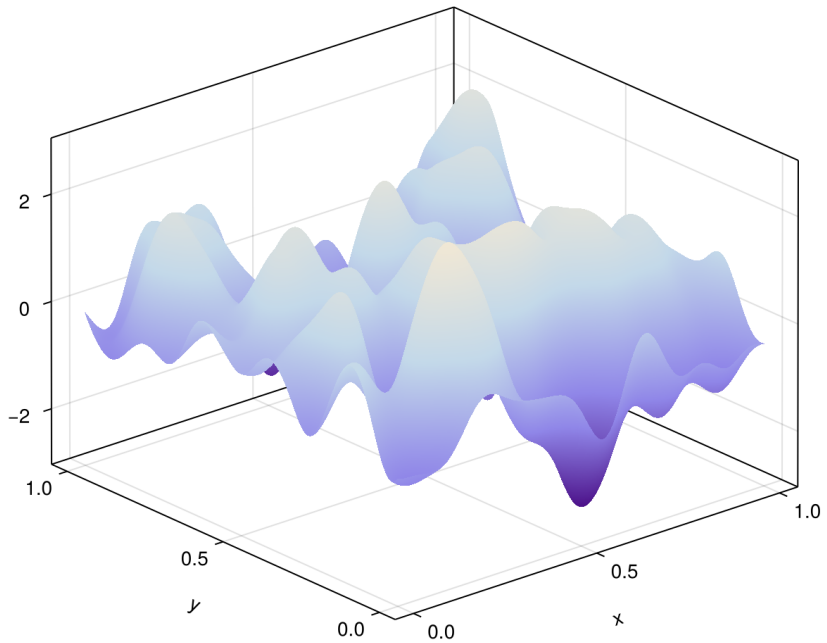
Compute the covariance matrix

```
Σ = kernelmatrix(k, x)
```

Example: Gaussian Random Fields

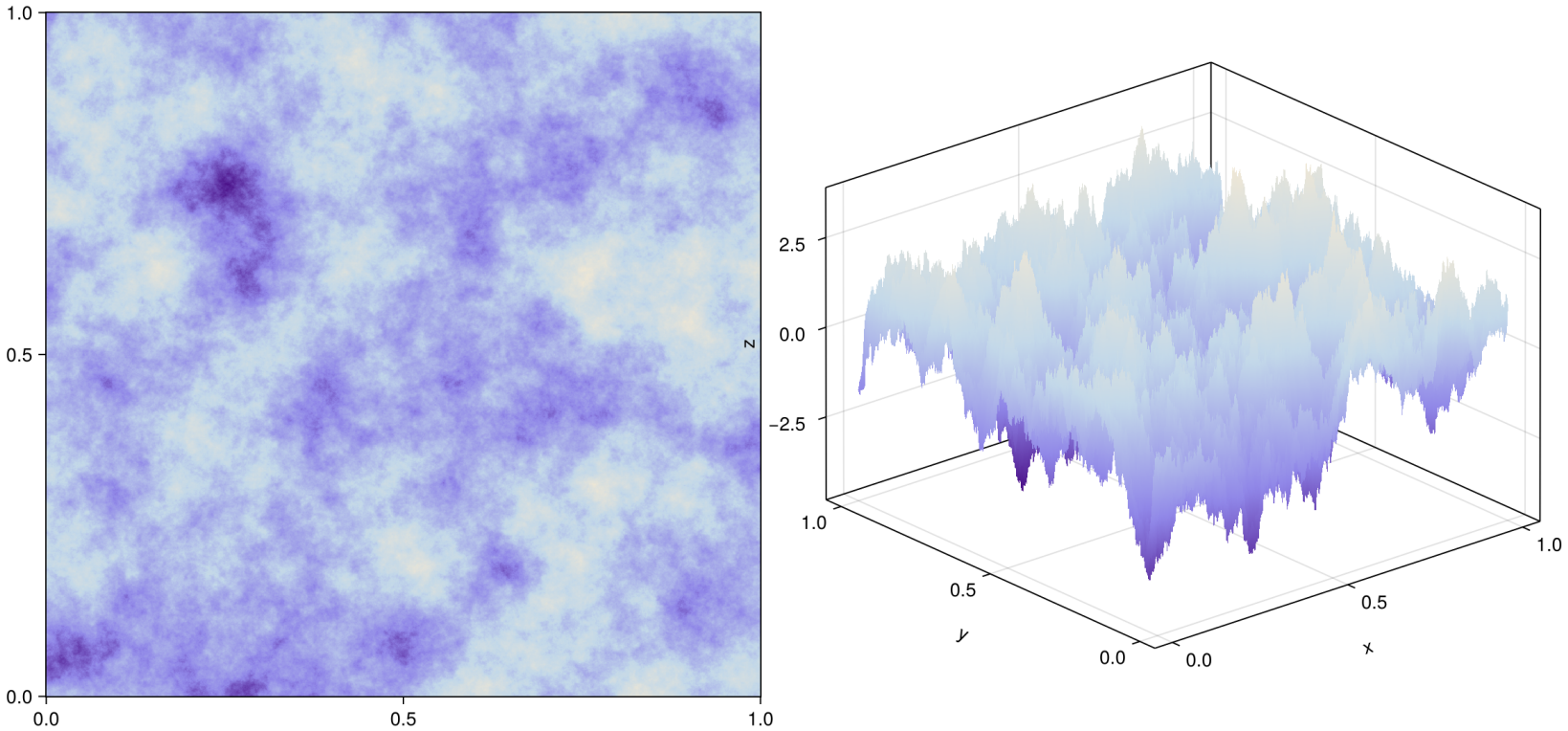
Squared-exponential kernel





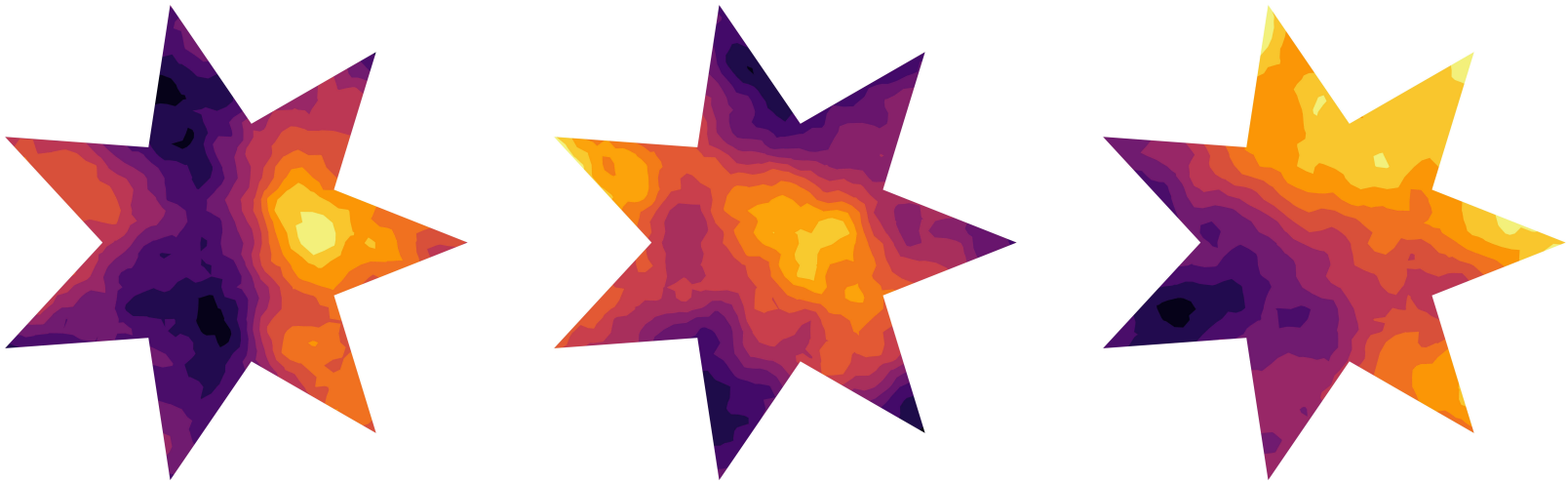
Example: Gaussian Random Fields

Exponential kernel



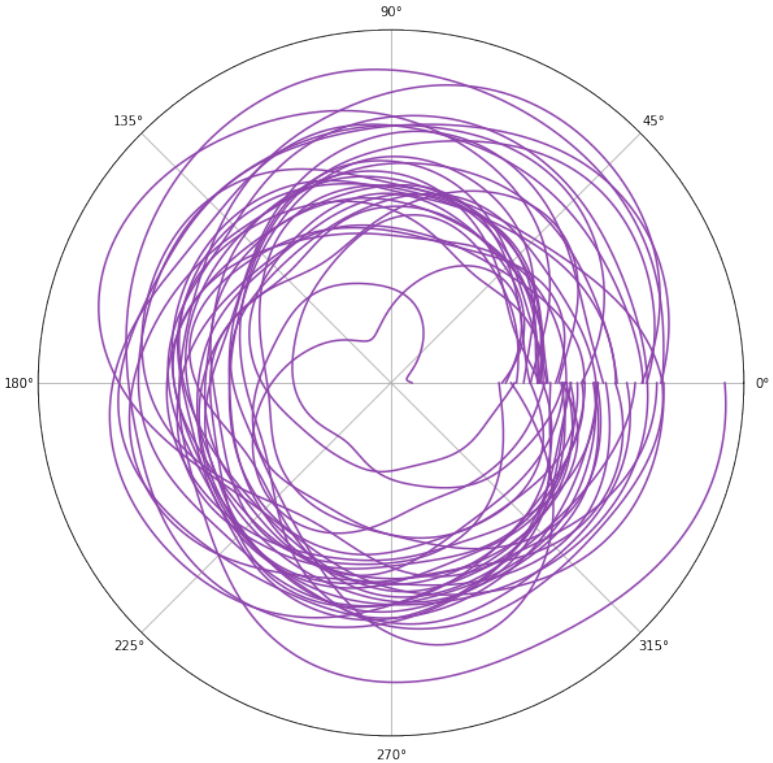
Example: Gaussian Random Fields

A GRF with Matern kernel defined on a star-shaped mesh [1]

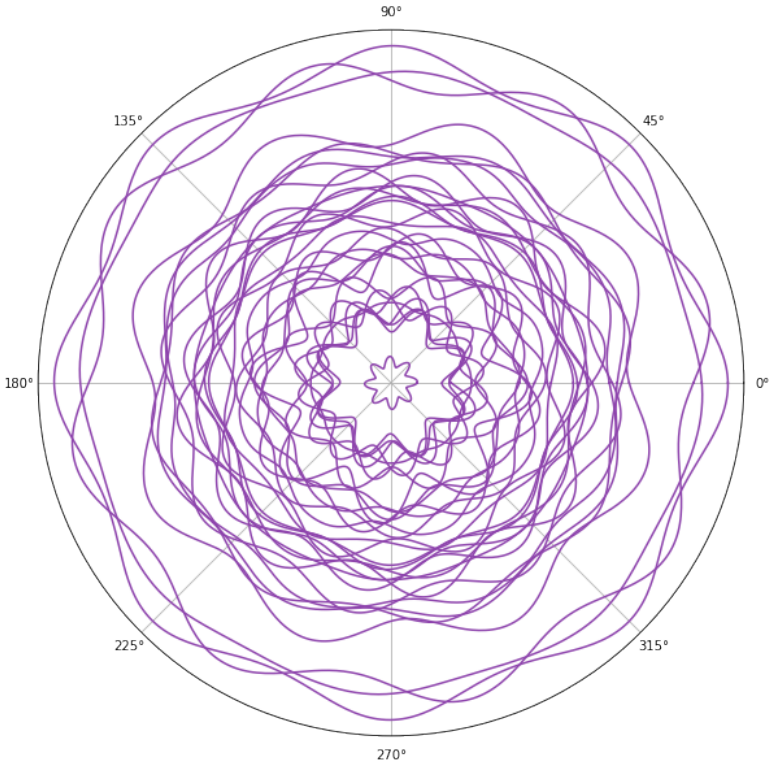


[1] For this example see `https://pieterjanrobbe.github.io/GaussianRandomFields.jl`

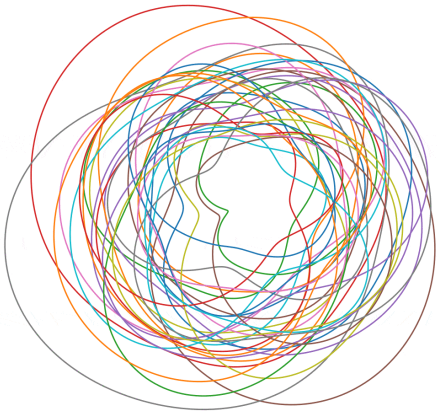
Example : Probabilistic shape modeling with GP



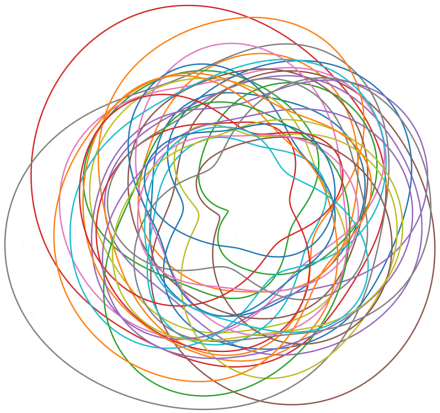
Example : Probabilistic shape modeling with GP

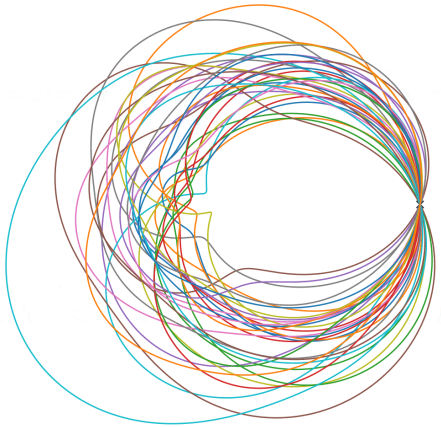


Example : Probabilistic shape modeling with GP

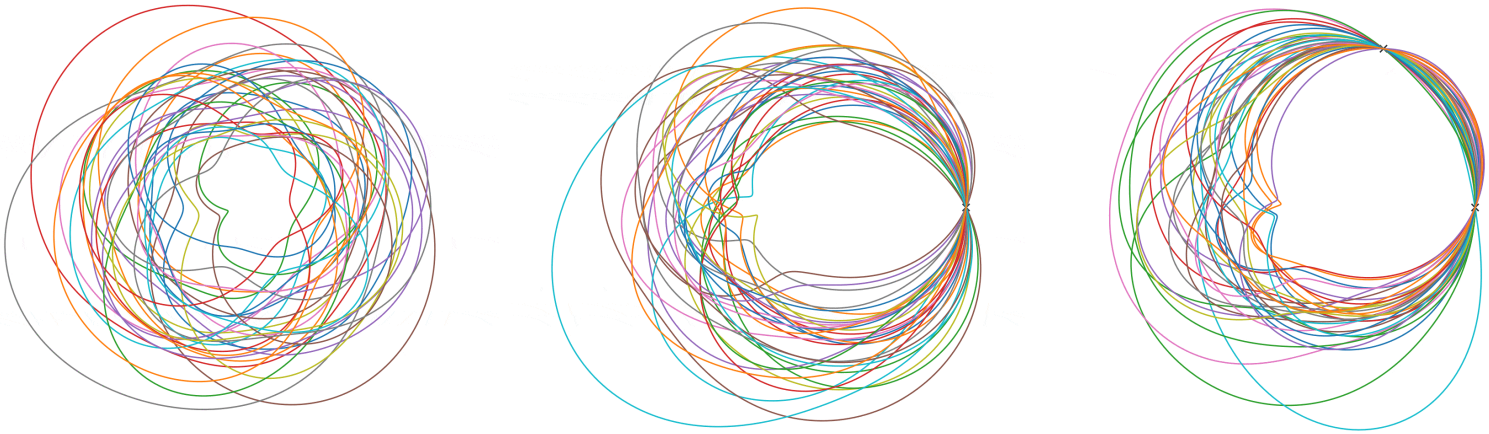


Example : Probabilistic shape modeling with GP

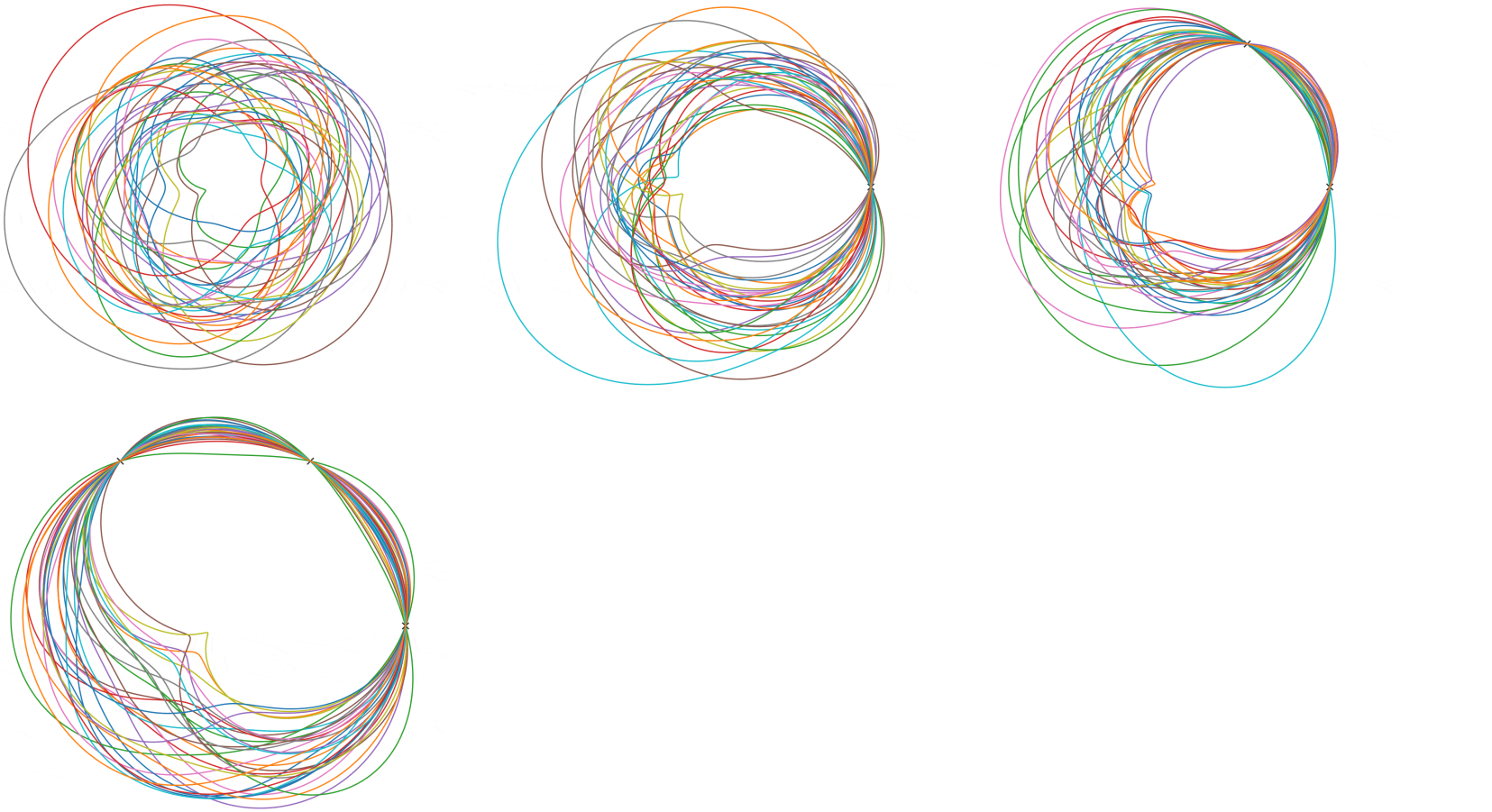




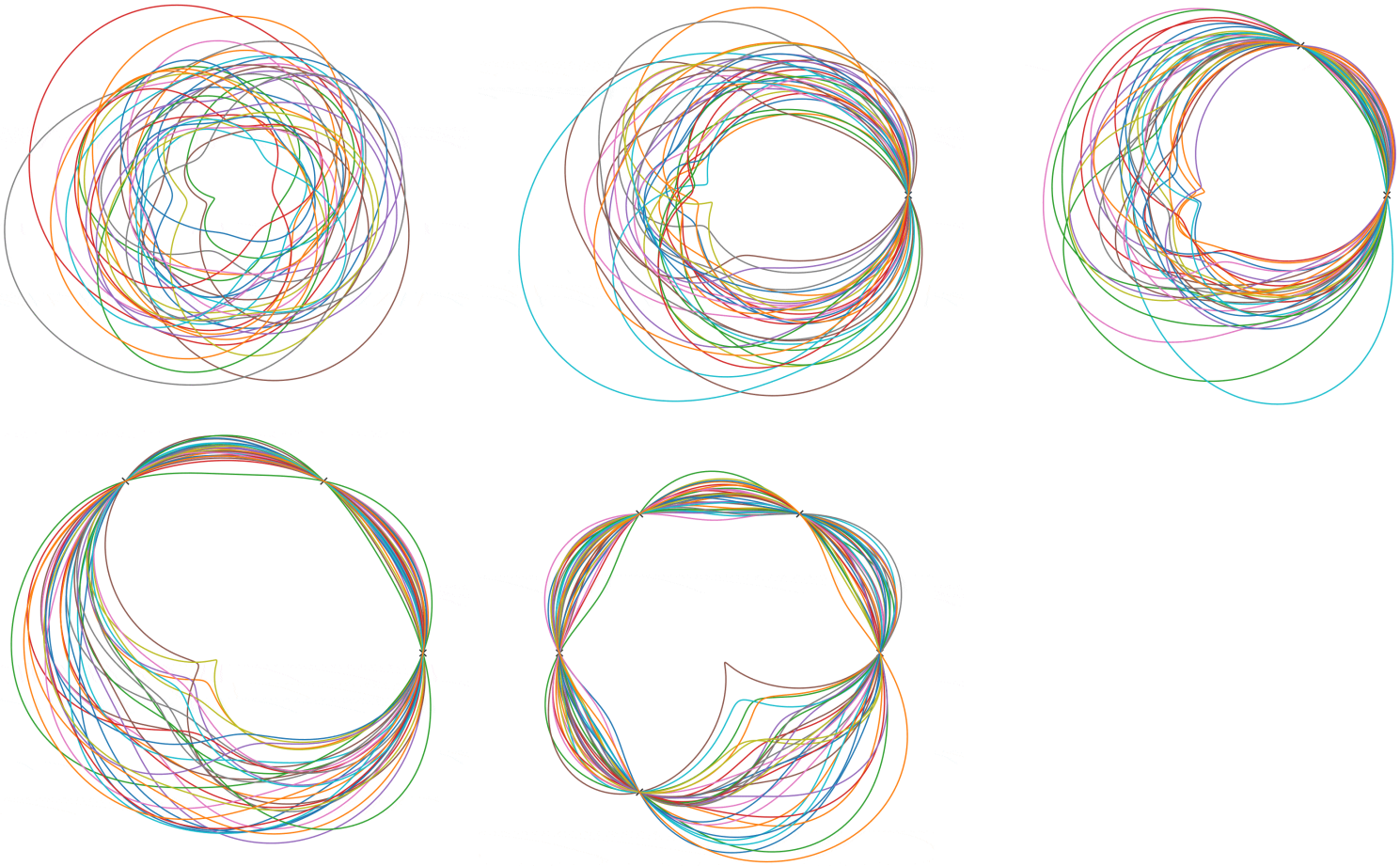
Example : Probabilistic shape modeling with GP



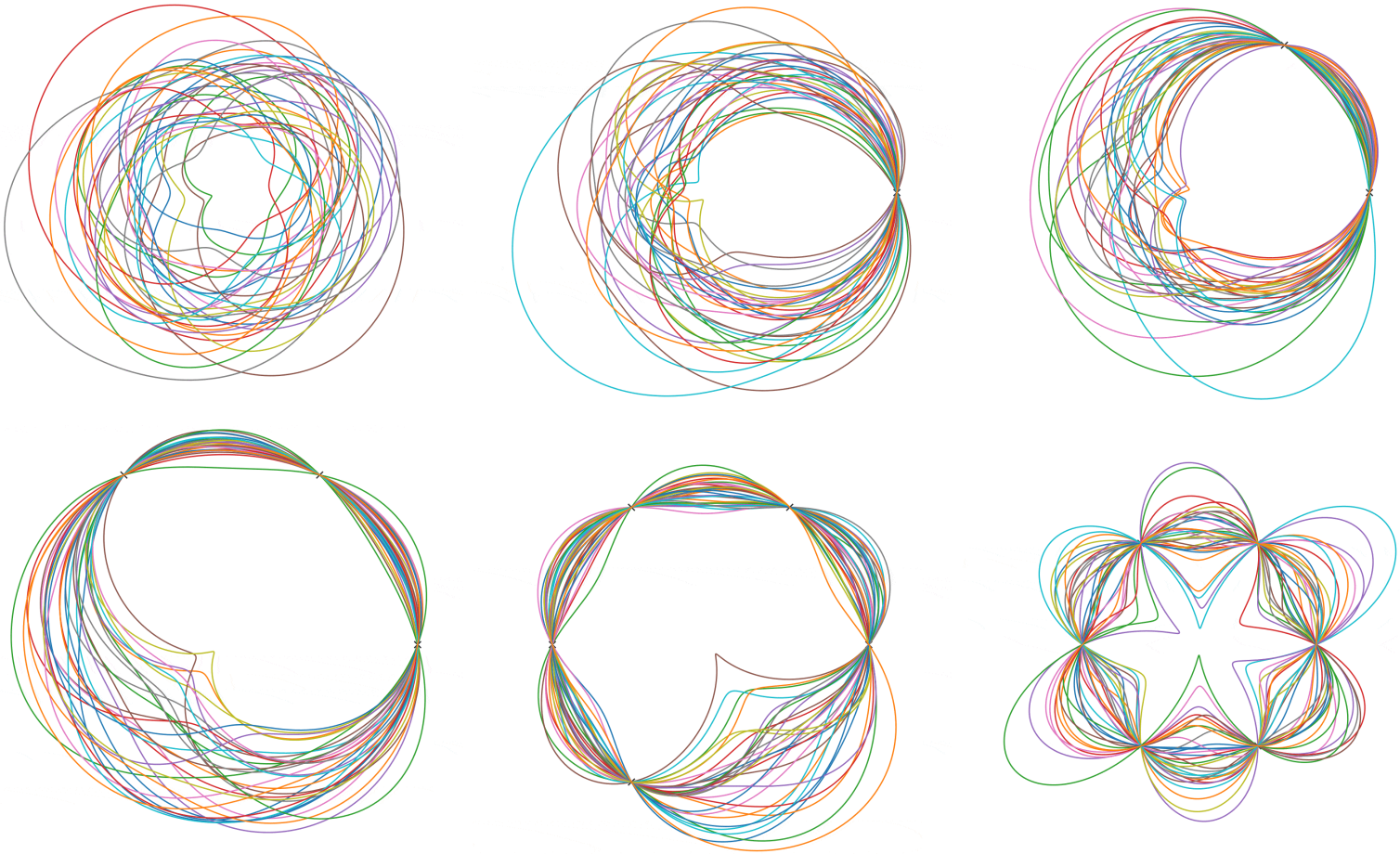
Example : Probabilistic shape modeling with GP



Example : Probabilistic shape modeling with GP



Example : Probabilistic shape modeling with GP



Gaussian Process in Probabilistic Programs

We can use GPs in probabilistic programs to define latent functions in a nonparametric way. Parameters of GP kernel can be latent variables too. [2]

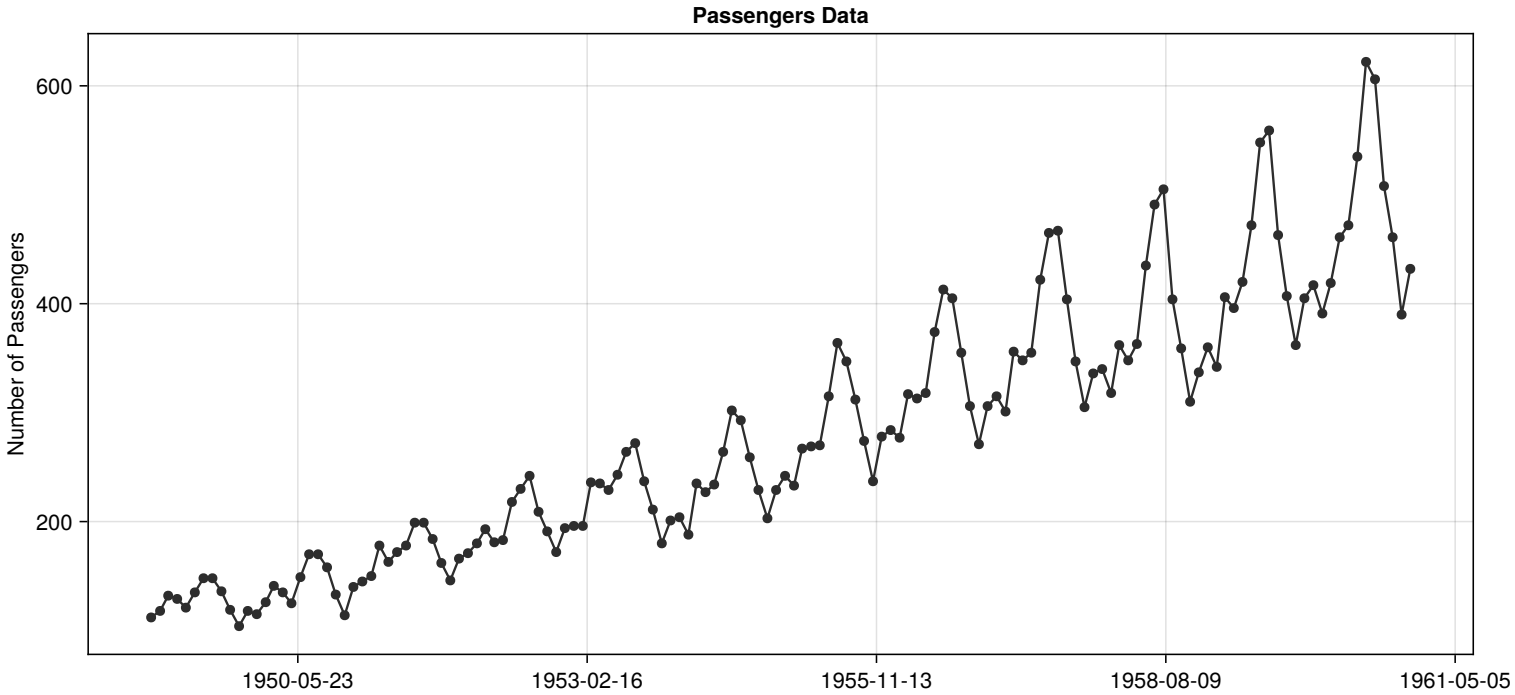
$$f \sim \text{GP}(0, k)$$
$$y_j \mid f(d_j) \sim \text{Binomial}(n_j, g(f(d_j)))$$
$$g(x) := \frac{1}{1 + e^{-x}}$$

[2] Source : `https://turinglang.org/docs/tutorials/gaussian-processes-introduction/`

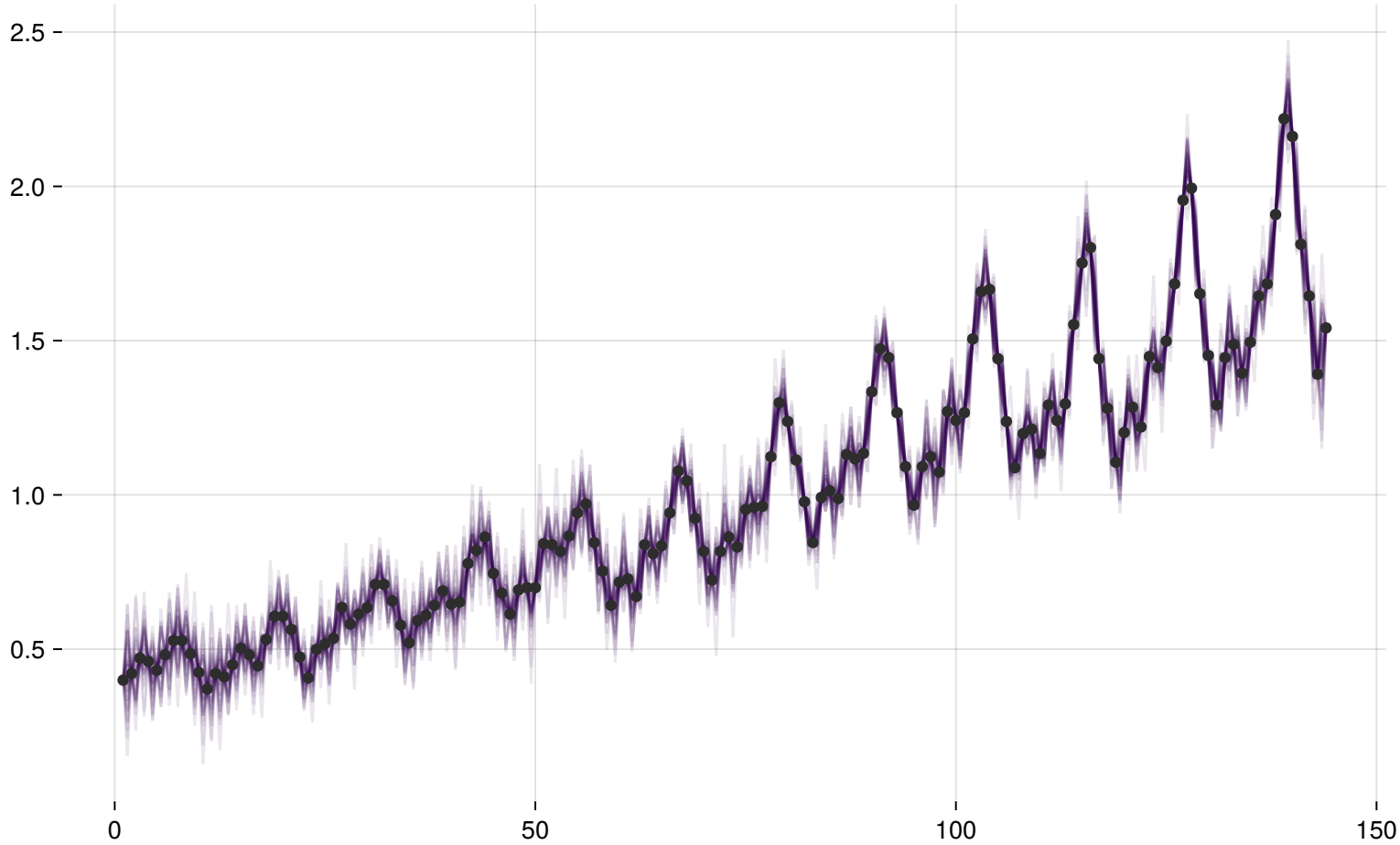
Gaussian Process in Probabilistic Programs

```julia
@model function GP_program(d, n; ϵ=1e-4)
    v ~ Gamma(2, 1)
    l ~ Gamma(4, 1)
    f = GP(v * with_lengthscale(SEKernel(), l))
    f_latent ~ f(d, ϵ)
    y ~ product_distribution(Binomial.(n, logistic.(f_latent)))
end
```

Example: Time Series modeling with Gaussian Processes



Example: Time Series modeling with Gaussian Processes



Example: Nonparametric Latent ODE

Using GP to formulate latent ODEs

$$\frac{dx}{dt} = f(.)$$

$$f \sim GP(k)$$

Research Trends

- Scalable Gaussian Process

Research Trends

- Scalable Gaussian Process
- Deep Gaussian Process

Research Trends

- Scalable Gaussian Process
- Deep Gaussian Process
- Constrained Gaussian Process

Learning Resources

- Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K. I. Williams
- Recorded lectures from Gaussian Process Summer School (available on YouTube)