Ariadne

Optimization

This section shows how to optimize hyperparameters of covariance functions in Ariadne. The package currently implements two methods:

- Metropolis-Hastings algorithm for sampling from the posterior. This method requires specifying a prior distribution for each hyperparameter.
- Simple gradient descent algorithm. The implementation includes only a very basic form of gradient descent.

Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is a simple Monte Carlo method for sampling from the posterior distribution. Let θ represent hyperparameters of a covariance function. Then the posterior distribution is defined as

$$p(\theta \mid X, Y) \propto p(Y \mid X, \theta) p(\theta)$$

where X are locations of observations and Y are the observed function values. Then $p(Y \mid X, \theta)$ is the Gaussian process likelihood of observed data.

To be able to sample from the posterior distribution, first we need to specify a prior distribution over hyperparameters $p(\theta)$. For the squared covariance function, hyperparameters are the lengthscale l, signal variance σ^2 and noise variance σ^2_{noise}.

$$\theta = \left(l, \sigma^2, \sigma^2_{\text{noise}}\right)$$

Specifying prior distribution

Because the hyperparameters must be positive, we will use a simple log-normal prior distribution for each hyperparameter.

```
#r "Ariadne.dll"
open Ariadne.GaussianProcess
open Ariadne.Kernels

#r "MathNet.Numerics.dll"
open MathNet.Numerics

let rnd = System.Random(3)
let lengthscalePrior = Distributions.LogNormal.WithMeanVariance(2.0, 4.0, rnd)
let variancePrior = Distributions.LogNormal.WithMeanVariance(1.0, 5.0, rnd)
let noisePrior = Distributions.LogNormal.WithMeanVariance(0.5, 1.0, rnd)
```

Ariadne includes a simplified method to work with squared exponential kernels. We can define a prior distribution over squared covariance hyperparameters and directly sample squared covariance kernels from it.

```
let prior = SquaredExp.Prior(lengthscalePrior, variancePrior, noisePrior)
let kernel = prior.Sample()
let gp = kernel.GaussianProcess()
```

Now that we specified a prior distribution, we can use Metropolis-Hastings algorithm to get samples from the posterior distribution of hyperparameters given training data.

```
 // Optimization
 open Ariadne.Optimization

 let data =
   [{Locations = (...)
     Observations = (...) }]
```

Covariance functions

Optimization

Quick how-to

Example: Modeling temperatures

DOCUMENTATION

API Reference

To sample from the posterior distribution with squared exponential kernel, we can use a built-in function `optimizeMetropolis`. This function runs Metropolis-Hastings algorithm with symmetric proposal distribution, i.e. basic Metropolis algorithm.

```
// Metropolis sampling for squared exponential with default settings
let newKernel1 =
    kernel
    |> SquaredExp.optimizeMetropolis data MetropolisHastings.defaultSettings prior

// Specify custom parameters
open Ariadne.Optimization.MetropolisHastings

let settings =
    { Burnin = 500     // Number of burn-in iterations
      Lag = 5           // Thinning of posterior samples
      SampleSize = 100 } // Number of thinned posterior samples

let newKernel2 =
    kernel
    |> SquaredExp.optimizeMetropolis data settings prior
```

The Metropolis sampler takes a specified total number of samples, 50 by default, and returns the posterior mean estimate for each hyperparameter.

```
Squared exponential, l = 0.52, σ² = 22.41, σ²_{noise} = 0.26
```

We can compare difference in fit of the randomly sampled covariance parameters `kernel` and sampled values in `newKernel2`. You might need to re-run the sampler for more iterations or with a different starting location to arrive to a good posterior estimate of hyperparameter values. Note that each iteration of the Metropolis-Hastings sampler has $\mathcal{O}(N^3)$ time complexity.

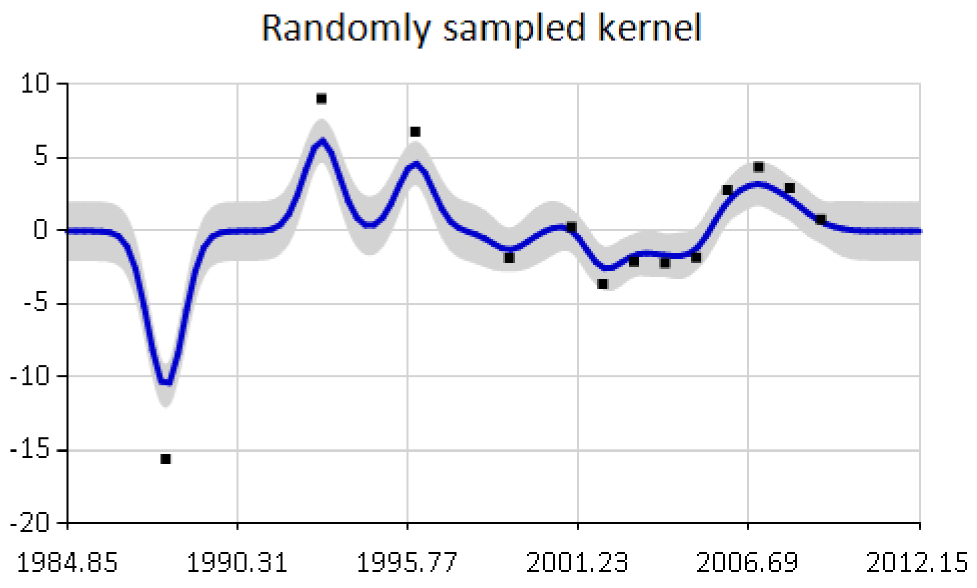
```
let gpInit = kernel.GaussianProcess()
let gpOptim = newKernel2.GaussianProcess()

let loglikInit = gpInit.LogLikelihood data
let loglikOptim = gpOptim.LogLikelihood data
printfn "Initial log likelihood: %f \nFinal log likelihood: %f" loglikInit loglikOptim
```

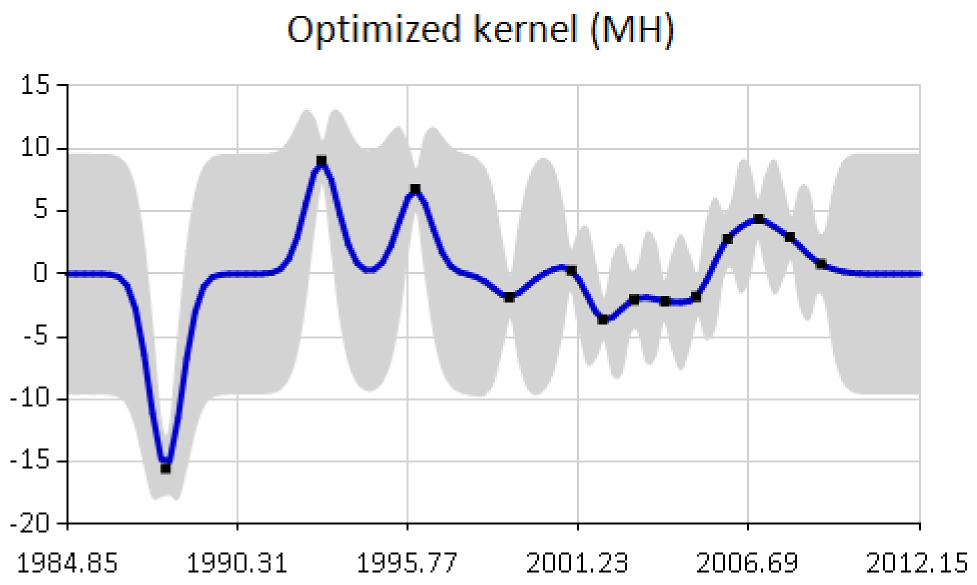
```
Initial log likelihood: -231.973587
Final log likelihood: -41.512115
```

```
open FSharp.Charting

kernel.GaussianProcess() |> plot data
|> Chart.WithTitle("Randomly sampled kernel", InsideArea=false)
|> Chart.WithMargin(0.0,10.0,0.0,0.0)
```



```
newKernel2.GaussianProcess() |> plot data
|> Chart.WithTitle("Optimized kernel (MH)", InsideArea=false)
|> Chart.WithMargin(0.0,10.0,0.0,0.0)
```



Running general Metropolis-Hastings

It is also possible to run general Metropolis-Hastings algorithm to obtain samples from the posterior distribution by calling `sampleMetropolisHastings` function. The function operates over an array of parameter values. As parameters, it takes functions that compute log likelihood, transition probability, Markov Chain settings and a proposal sampler. See the source code for details.

Gradient descent algorithm

It is also possible to fit hyperparameters of covariance functions using any gradient-based optimization algorithm. Ariadne currently implements only basic gradient descent.

The `gradientDescent` function operates over an array of parameter values. It requires a function that computes gradient of log likelihood with respect to all hyperparameters. There is an implementation of gradient for squared exponential covariance kernel. Because there are local maxima in the likelihood function, gradient descent might require several restarts with different initial locations and step sizes.

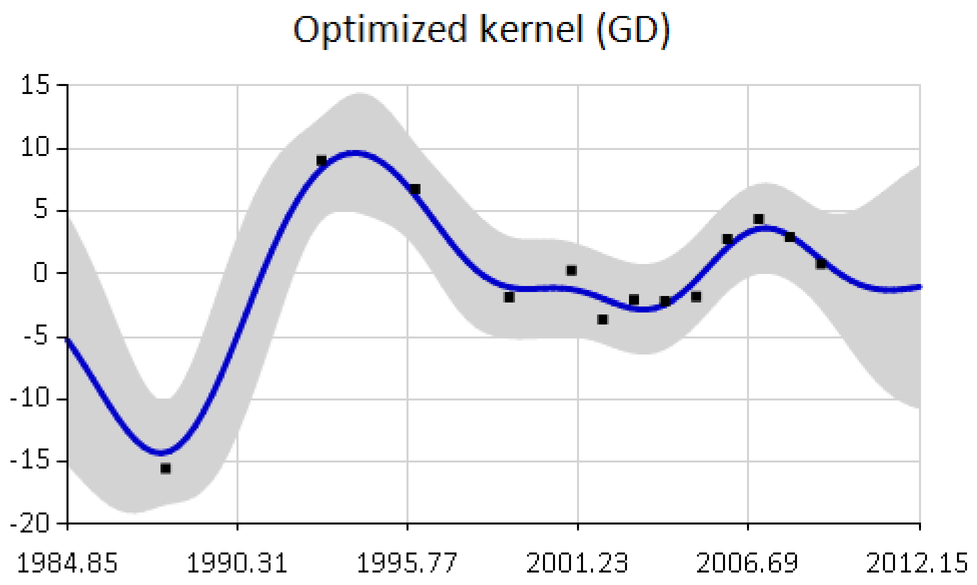
```
// Gradient descent settings
let gdSettings =
    { GradientDescent.Iterations = 10000;
      GradientDescent.StepSize = 0.01}

let gradientFunction parameters = SquaredExp.fullGradient data parameters
// Run gradient descent
let kernelGD =
    gradientDescent gradientFunction gdSettings (kernel.Parameters)
    |> SquaredExp.ofParameters

let gpGD = kernelGD.GaussianProcess()
printfn "Original Gaussian process likelihood: %f" (gpInit.LogLikelihood data)
printfn "Optimized Gaussian process likelihood: %f" (gpGD.LogLikelihood data)
```

```
Original Gaussian process likelihood: -231.973587
Optimized Gaussian process likelihood: -37.614942
```

```
gpGD |> plot data
|> Chart.WithTitle("Optimized kernel (GD)", InsideArea=false)
|> Chart.WithMargin(0.0,10.0,0.0,0.0)
```


Optimized kernel (GD)
15
10
5
0
-5
-10
-15
-20
1984.85
1990.31
1995.77
2001.23
2006.69
2012.15