GaPP Documentation

Version 1.0

Marina Seikel

August 27, 2013

Abstract

GaPP (Gaussian Processes in Python) is a package in Python that provides methods for Gaussian process regression. It allows for reconstruction of a function and its first, second and third derivatives using observational data of the function; optionally also observations of the first derivative of the function can be used. The observations may have individual errors. Updates of GaPP will be available on
`http://www.acgc.uct.ac.za/~seikel/GAPP/index.html`

Contents

1 Getting started

1.1 Requirements and installation

The requirements for GaPP are Python 2.6 as well as the Python packages NumPy and SciPy. If the GaPP modules `mcmcgp` or `mcmcdgp` are used to sample hyperparameters, the packages `emcee` (`http://github.com/dfm/emcee`) and `acor` (`http://github.com/dfm/acor`) are required.

For installation, unpack the file, change into the `GaPP` directory and install `GaPP` with:
`python setup.py install`

If you are not familiar with Gaussian Processes, we suggest that you read our accompanying paper: M. Seikel, C. Clarkson and M. Smith, *Reconstruction of dark energy using Gaussian processes*, JCAP06(2012)036.

1.2 Modules

GaPP contains four modules for Gaussian process regression: `gp`, `dgp`, `mcmcgp` and `mcmcdgp`. All modules can be used to reconstruct a function from observational data `X`, `Y` and `Sigma`. The differences between the modules are the following:

- `dgp` (section 2):
  The inputs can only be one-dimensional, i.e. the elements x_i of `X` must be numbers. `dgp` can reconstruct the function as well as its first, second and third derivatives. Observations of the function and optionally of the first derivative of the function (`dX`, `dY` and `dSigma`) can be used for the reconstruction.
  The hyperparameters are optimized.
  Returns the mean and standard deviation of the reconstructed function.
- `mcmcdgp` (section 3):
  Similar to `dgp`, but the hyperparameters are marginalized over.
  Returns samples of the probability density of the reconstructed function.
- `gp` (section 4):
  The inputs can be multi-dimensional, i.e. the elements x_i of `X` can be vectors or numbers. `gp` can only reconstruct the function, but not its derivatives. Only observations of the function, but not of its derivative can be used.
  The hyperparameters are optimized.
  Returns the mean and standard deviation of the reconstructed function.
- `mcmcgp` (section 5):
  Similar to `gp`, but the hyperparameters are marginalized over.
  Returns samples of the probability density of the reconstructed function.

From a Bayesian point of view, the marginalization over hyperparameters is the correct approach. However, the optimisation of the hyperparameters is usually a good approximation and can be calculated much faster than the marginalization.

1.3 Examples

1.3.1 Basic `dgp` example

The basic usage of `dgp` is described in the example `dgp_example.py`, which can be found in the directory `GaPP/examples/dgp`. If matplotlib is installed, a plot of the reconstructions will be created automatically.

```
from gapp import dgp
from numpy import loadtxt, savetxt

(X, Y, Sigma) = loadtxt("inputdata.txt", unpack = 'True')

xmin = 0.0
xmax = 10.0
nstar = 200
initheta = [2.0,2.0]

g = dgp.DGaussianProcess(X, Y, Sigma, cXstar = (xmin, xmax, nstar))

(rec, theta) = g.gp(theta = initheta)
(drec, theta) = g.dgp(thetatrain = 'False')
(d2rec, theta) = g.d2gp()
(d3rec, theta) = g.d3gp()

savetxt("f.txt", rec)
savetxt("df.txt", drec)
savetxt("d2f.txt", d2rec)
savetxt("d3f.txt", d3rec)
```

The first block imports the necessary modules.

Then the data are loaded from `inputdata.txt`, where the data are arranged in three columns: the first column contains `X`, the second `Y` and the third the measurement errors `Sigma`.

In the next block, `xmin`, `xmax` and `nstar` are defined. The function will be reconstructed at `nstar` evenly distributed points between `xmin` and `xmax`. `initheta` is the initial guess for the hyperparameters $\{\sigma_f, \ell\}$ of the squared-exponential covariance function, which is the default covariance function. σ_f should be the typical change of the data in y-direction, and ℓ the typical length scale in x-direction, along which $f(x)$ changes significantly. It is recommended to run the Gaussian process a few times with different initial values for the hyperparameters.

The Gaussian process `g` is then initialized with the data `X`, `Y` and `Sigma`. `cXstar` creates the locations `Xstar`, where the function will be reconstructed.

The actual reconstruction of the function and its derivatives is performed in the next block. `g.gp(theta=initheta)` first trains the hyperparameters `theta` starting with the initial values `initheta`; then the function $f(x)$ is reconstructed at positions `Xstar`. `rec` and the optimized hyperparameters `theta` are returned. `rec` is a (`nstar`, 3) array, where the first column contains `Xstar`, the second and third column contain the mean and standard deviation of the reconstructed function at positions `Xstar`, respectively. `g.dgp(thetatrain='False')` reconstructs the first derivative $f'(x)$. `thetatrain='False'` fixes the hyperparameters to the current value, so they are not trained again. Note that also in the subsequent reconstructions of the second and third derivatives (using `g.d2gp()` and `g.d3gp()`, respectively), the hyperparameters are fixed. If one wishes to retrain the hyperparameters, one needs to set `thetatrain='True'` in the method call.

Finally the results are saved to files.

1.3.2 More advanced `dgp` example

Slightly more advanced usage of `dgp` is described in the example `dgp2_example.py`, which can be found in the directory `GaPP/examples/dgp2`. Again, if matplotlib is installed, a plot of the reconstructions will be created automatically.

```
from gapp import dgp, covariance
from numpy import loadtxt, savetxt
```

```
(X, Y, Sigma) = loadtxt("inputdata.txt", unpack='True')
(DX, DY, DSigma) = loadtxt("dinputdata.txt", unpack='True')

xmin = 0.0
xmax = 10.0
nstar = 200
initheta = [1.0, 1.0]

g = dgp.DGaussianProcess(X, Y, Sigma, dX = DX, dY = DY, dSigma = DSigma,
        covfunction = covariance.Matern72, cXstar=(xmin, xmax, nstar), grad='False')

(rec, theta) = g.gp(theta = initheta)
(drec, theta) = g.dgp(thetatrain = 'False')
(d2rec, theta) = g.d2gp()
(d3rec, theta) = g.d3gp()

savetxt("f.txt", rec)
savetxt("df.txt", drec)
savetxt("d2f.txt", d2rec)
savetxt("d3f.txt", d3rec)
```

This example is very similar to the basic `dgp` example (see section 1.3.1). The difference is that now also observations of the first derivative of the function are used; and that a different covariance function for the Gaussian process is used.

In order to use the observations of $f'(x)$, first the data `DX`, `DY` and `DSigma` are loaded. When the Gaussian process is initialized, one just needs to add `dX=DX`, `dY=DY`, `dSigma=DSigma` to the parameter list.

The module `covariance` must be imported, if a covariance function other than the default (`SquaredExponential`) is used. The desired covariance function is then added to the parameter list when initializing the Gaussian process. In the example, we use the `Matern72` covariance function; thus we add `covfunction=covariance.Matern72` to the parameter list. The available covariance functions are described in section 6.

Setting `grad='False'` causes GaPP to optimize the hyperparameters using `scipy.optimize.fmin_cobyla` instead of `scipy.optimize.fmin_tnc`, which is used by default.

1.3.3 gp example

The basic usage of `gp` is described in the example `gp_example.py`, which can be found in the directory `GaPP/examples/gp`. If matplotlib is installed, a plot of the reconstruction will be created automatically.

```
from gapp import gp
from numpy import loadtxt, savetxt

X = loadtxt("2d-inputdata.txt", usecols=(0,1))
(Y, Sigma) = loadtxt("2d-inputdata.txt", usecols=(2,3), unpack='True')

xmin = 0.0
xmax = 10.0
nstar = 40

initheta = [2.0, 2.0]

g = gp.GaussianProcess(X, Y, Sigma, cXstar=(xmin, xmax, nstar))
(rec, theta) = g.gp(theta=initheta)
```

```
savetxt("f.txt", rec)
```

This example is very similar to the basic `dgp` example (section 1.3.1). However, now the input data `X` are two-dimensional.

`xmin`, `xmax` and `nstar` are interpreted as two-dimensional vectors `xmin = [0.0, 0.0]`, `xmax = [10.0, 10.0]` and `nstar = [40, 40]`.

The created plot only shows the reconstructed mean function without error bands.

1.3.4 `mcmcdgp` example

The basic usage of `mcmcdgp` is described in the example `mcmcdgp_example.py`, which can be found in the directory `GaPP/examples/mcmcdgp`. Note that the python packages `emcee` and `acor` are required for this example. If matplotlib is installed, a plot of the reconstruction will be created automatically.

```
from gapp import mcmcdgp
from numpy import loadtxt, random, savetxt

(X, Y, Sigma) = loadtxt("inputdata.txt", unpack='True')
(DX, DY, DSigma) = loadtxt("dinputdata.txt", unpack='True')

xmin = 0.0
xmax = 10.0
nstar = 50

nwalker = 20
theta0 = random.normal(2.0, 0.2, (nwalker, 2))

g = mcmcdgp.MCMCDGaussianProcess(X, Y, Sigma, theta0, Niter=50, dX=DX, dY=DY, dSigma=DSigma,
                                 cXstar=(xmin, xmax, nstar), threads=4, reclist=[0, 1])

(Xstar, rec, drec) = g.mcmcdgp()

savetxt("rec.txt", rec)
savetxt("drec.txt", drec)
savetxt("Xstar.txt", Xstar)
```

As in the other examples, we start by importing modules, loading the data and fixing the points where the function will be reconstructed.

`nwalker` is the number of walkers used by the affine invariant MCMC sampler `emcee`. The walkers form an ensemble of positions in parameter space. `theta0` provides the initial positions of the walkers, i.e. the initial values of the hyperparameters. As the squared exponential covariance function depends on two hyperparameters, `theta0` is an array of size `(nwalker, 2)`. As initial values we choose random values with mean 2.0 and standard deviation 0.2.

When initializing the Gaussian process, we provide the number of iterations `Niter` that are performed after the burn-in period. During each iteration, `nwalker` samples of the hyperparameters will be drawn. Thus the total hyperparameter sample size is `nwalker` $\times$ `Niter` $= 1000$. The degree of parallelisation is determined by `threads=4`, i.e. four processors are used for the calculations. `reclist` defines which derivatives of the function are to be reconstructed, 0 stands for the function itself and 1 for the first derivative.

The actual calculations are performed with the command `mcmcdgp()`. First the hyperparameters are sampled. Then this sample will be used to sample the distribution of the reconstructed

function: For each set of hyperparameters, a Gaussian process reconstruction is performed. From the resulting mean and standard deviation of the reconstruction `nsample` sample points are drawn at each point of `Xstar`. (As we have not specified `nsample` in this example, the default value `nsample=50` is used.) Thus we obtain `nwalker` $\times$ `Niter` $\times$ `nsample` $= 50000$ sample points for the reconstruction at each point of `Xstar`.

`mcmcdgp()` returns the positions of the reconstructions `Xstar` and the samples of the reconstructions as (`nstar`, `nwalker` $\times$ `Niter` $\times$ `nsample`) arrays. Note that the samples are always in order from the lowest to the highest derivative, i.e. if we set `reclist=[1, 0]` instead of `reclist=[0, 1]`, the return value is still `(Xstar, rec, drec)`.

The plots that are created for this example use the mean and standard deviation of the samples. This makes only sense if the distribution of the samples is approximately Gaussian. Do not use this approach without confirming Gaussianity first!

2 dgp module – Gaussian Processes with derivatives

The module `dgp` can reconstruct a function as well as its first, second and third derivatives. Measurements of the first derivative $f'(x)$ can be used in addition to measurements of the function $f(x)$. It can only handle one-dimensional inputs x_i. Optimized hyperparameters are used for the reconstruction.

2.1 Initializing a Gaussian process

As a first step, the Gaussian Process needs to be initialized with

`dgp.`**DGaussianProcess**(*X, Y, Sigma, covfunction=covariance.SquaredExponential, theta=None, dX=None,dY=None,dSigma=None, Xstar=None, cXstar=None, mu=None, dmu=None, d2mu=None, d3mu=None, muargs=(), prior=None, gradprior=None, priorargs=(), thetatrain='True', scale=None, scaletrain='True', grad='True'*)

Parameters:

X: array_like,
A vector of length n containing the locations x_i of the observations. n is the number of data points.

Y: array_like
Vector containing the observations of $f(X)$.

Sigma: array_like
Either a vector of length n containing the measurement errors of `Y`, or a $n \times n$ covariance matrix of the data.

covfunction: class or tuple (class, class)
Covariance function. See section 6 for the covariance functions provided by the module `covariance`.
If `covfunction` is given in the form `(covfunction1, covfunction2)`, the sum of `covfunction1` and `covfunction2` will be used as covariance function (see section 6.2 for details).

theta: tuple
(Initial) values of the hyperparameters of the covariance function. For covariance functions with only two hyperparameters, `theta[0]` denotes the signal variance σ_f and `theta[1]` the length scale ℓ. (See section 6 for the meaning of `theta` for the specific covariance functions.)

dX: array_like or number,
A vector of length n' containing the locations x_i of the observations of the first derivative f'. n' is the number of data points. For $n' = 1$, `dX` can be a number.

dY: array_like or number
Vector (or number, if $n' = 1$) containing the observations of f'(X).

dSigma: array_like or number
Either a vector of length n' containing the measurement errors of `dY`, or a $n' \times n'$ covariance matrix of the data. If `dSigma` is a number (if $n' = 1$), it is interpreted as a measurement error and *not* a covariance matrix.

Xstar: array_like
Vector containing the locations, where f(x) is to be reconstructed.

cXstar: tuple (`xmin`, `xmax`, `nstar`)
`Xstar` will be created automatically. `nstar` values are created between `xmin` and `xmax`. If `xmin==None` or `xmax==None`, their respective values will be determined automatically:

$$\texttt{xmin} = \min(\texttt{X}) - 0.1(\max(\texttt{X}) - \min(\texttt{X})), \quad \texttt{xmax} = \max(\texttt{X}) + 0.1(\max(\texttt{X}) - \min(\texttt{X}))$$

If `Xstar` is provided explicitly, `cXstar` will be ignored.

mu: callable `mu(x, *muargs)`
A priori mean function of the Gaussian Process. The first argument of `mu` is `x`.

dmu: callable `dmu(x, *muargs)`
First derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `dmu` will be ignored.

d2mu: callable `d2mu(x, *muargs)`
Second derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `d2mu` will be ignored.

d3mu: callable `d3mu(x, *muargs)`
Third derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `d3mu` will be ignored.

muargs: tuple
Additional arguments passed to `mu`, `dmu`, `d2mu` and `d3mu`.

prior: callable `prior(theta, *priorargs)`
Prior on the hyperparameters `theta`. The first argument of `prior` must be `theta`. Must return a non-negative number. If `prior` depends on additional parameters, they have to be given by `priorargs`.

gradprior: callable `gradprior(theta, *priorargs)`
Function that returns a tuple containing the gradient of `prior` with respect to `theta`. `gradprior` needs to be provided if `prior!=None` and `grad=='True'`.

priorargs: tuple
Additional arguments passed to the functions prior and gradprior.

thetatrain: `'True'`, `'False'` or tuple
Defines which hyperparameters are to be trained.
`'True'`: all hyperparameters will be optimized.
`'False'`: values of the hyperparameters are fixed.
tuple of the length of the number of hyperparameters with entries `0` and `1`: the hyperparameter `theta[i]` will be trained if `thetatrain[i] = 1`, and is fixed for `thetatrain[i] = 0`.

scale: number, tuple of length 2, or `None`
If `scale!=None`, the data covariance matrix `Sigma` is multiplied by $\texttt{scale}^2$ if `scale` is a number, or by $\texttt{scale[0]}^2$ if `scale` is a tuple. Thus for uncorrelated data, σ_i is multiplied by `scale` or `scale[0]`, respectively.
If `scale` is a tuple, the data covariance matrix of the observations of the first derivative `dSigma` is multiplied by $\texttt{scale[1]}^2$.
If `scale!=None`, the value of `scale` is optimized by default.

scaletrain: `'True'`, `'False'` or tuple of length 2
Defines if `scale` is optimized.
`'True'`: `scale` will be optimized.
`'False'`: `scale` will not be optimized.
tuple of length 2 with entries 0 and 1: If `scale` is a tuple, `scale[i]` will be optimized if `scaletrain[i]==1`, and is fixed for `scaletrain==0`.

grad: `'True'` or `'False'`

`'True'` if the gradient of the covariance function is to be used to train the hyperparameters. This training method is faster than the alternative method. If `grad=='True'` and `prior!=None`, one needs to provide the gradient of the prior `gradprior`.

The only required parameters are the measurements `X`, `Y` and `Sigma`. By default, the squared exponential covariance function is used.

If no initial values for the hyperparameters `theta` are provided, the programm guesses somewhat reasonable initial values based on the range of `X` and `Y` values. It is, however, recommended that you choose the initial values yourself. If e.g. the function seems to be oscillating at a high frequency, you should choose a rather small initial value for the length scale ℓ to avoid a covariance matrix that is not positive definite. Should the program end with the error message "Matrix is not positive definite - Cholesky decomposition cannot be computed", try adjusting the initial values of `theta`.

The hyperparameters are trained by maximizing the marginal likelihood $p(\texttt{Y}|\texttt{X}, \texttt{theta})$. As this likelihood might have multiple local maxima, you should try different initial values for `theta` to avoid ending up in the wrong local maximum. This can also be avoided by using suitable priors on the hyperparameters.

If neither `Xstar` nor `cXstar` are provided, `Xstar` is created automatically with 200 values between `xmin` and `xmax`, which are determined from the range of `X` values.

Example: Inverse Gamma prior on `theta[0]`

```
from gapp import dgp
from numpy import exp
from scipy.special import gamma
X = [6.04, 0.86, 2.11, 9.54, 8.58]
Y = [2.29, 0.39, -0.56, -3.12, -2.12]
Sigma = [0.28, 0.36, 0.22, 0.19, 0.23]
def invgamma(theta, a, b):
    s = theta[0]
    p = b**a/gamma(a) * s**(-1 - a) * exp(-b/s)
    return p
g = dgp.DGaussianProcess(X, Y, Sigma, theta=[1, 1], prior = invgamma,
                         priorargs=(0.2, 1), grad = 'False')
```

2.2 Gaussian Process Regression

2.2.1 Full Gaussian Process Run

Once the Gaussian Process has been initialized, the reconstructions of the function and its derivatives are performed with:

Function:

gp(*theta=default, dX=default, dY=default, dSigma=default, Xstar=default, cXstar=default, mu=default, dmu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, scaletrain=default, grad=default, unpack='False'*)

First derivative:

dgp(*theta=default, dX=default, dY=default, dSigma=default, Xstar=default, cXstar=default, mu=default, dmu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, scaletrain=default, grad=default, unpack='False'*)

Second derivative:

d2gp(*theta=default, dX=default, dY=default, dSigma=default, Xstar=default, cXstar=default, mu=default, dmu=default, d2mu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, scaletrain=default, grad=default, unpack='False'*)

Third derivative:

d3gp(*theta=default, dX=default, dY=default, dSigma=default, Xstar=default, cXstar=default, mu=default, dmu=default, d3mu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, scaletrain=default, grad=default, unpack='False'*)

Parameters:

By default almost all parameters (exceptions are mentioned below) are set to the current attribute values. These values can be changed in the function call of `gp`, which changes the attribute value permanently (the only exception is `unpack`, which always defaults to `'False'`). It is not possible to change the observational values in the function call, as the data are considered to be fixed for a certain Gaussian Process. If for some reason, you need to change these data, use `set_data(X,Y,Sigma)` as described in section 2.3.

dX, dY, dSigma:

It is not possible to change only one of the parameters `dX`, `dY` and `dSigma`, i.e. all three parameters have to be changed in the same method call. The only exception is setting one of the parameters to `None`; then the other two parameters are also automatically set to `None`.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

scale, scaletrain:

If you set a new `scale`, but do not provide `scaletrain`, `scaletrain` will be set to the initial default values, i.e. `scaletrain = 'False'` if `scale == None`, and else `scaletrain = 'True'`.

There is one additional parameter that determines the form of the function return:

unpack: 'True' or 'False'

Defaults to `'False'`, even if it has been set to `'True'` in a previous function call.
`'True'`: function returns `(Xstar, fmean, fstd, theta)` if `scale==None` or `(Xstar, fmean, fstd, theta, scale)` if `scale!=None`. `fmean` and `fstd` are vectors containing the mean and standard deviation of the reconstructed function at locations `Xstar`, respectively.
`'False'`: function returns `(rec, theta)` if `scale==None` or `(rec, theta, scale)` if `scale!=None`. `rec` is an array containing `Xstar`, `fmean` and `fstd`.

Returns:

`(Xstar, fmean, fstd, theta)` if `unpack=='True'` and `scale==None`
`(Xstar, fmean, fstd, theta, scale)` if `unpack=='True'` and `scale!=None`
`(rec, theta)` if `unpack=='False'` and `scale==None`
`(rec, theta, scale)` if `unpack=='False'` and `scale!=None`

If the hyperparameters are trained, messages from the optimisation and the optimised hyperparameters are displayed.

Example:

Assuming you have initialized the Gaussian Process according to the example in the previous section, you can perform the reconstruction using the attribute values specified in the initialization, or you can change the attributes in the function call.

```
(Xstar,fmean,fstd,theta1) = g.gp(unpack='True')
(rec,theta2) = g.gp(prior=None)
(drec,theta2) = g.dpg(thetatrain='False')
```

In the second line, the attribute `prior` of `g` is permanently set to None. In the third line, the hyperparameters are fixed. Training the hyperparameters would again result in `theta2` and is thus not necessary.

2.2.2 Training the hyperparameters

If one is not interested in a whole Gaussian Process run, but only in training the hyperparameters, this can be done with:

hypertrain(*theta=default, dX=default, dY=default, dSigma=default, mu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, grad=default*)

Parameters:

By default all parameters are set to the current attribute values. These values can be changed in the function call of `hypertrain`, which changes the attribute value permanently.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

scale, scaletrain:

If you set a new `scale`, but do not provide `scaletrain`, `scaletrain` will be set to the initial default values, i.e. `scaletrain = 'False'` if `scale == None`, and else `scaletrain = 'True'`.

Returns:

`theta` if `scale==None`
`(theta, scale)` if `scale!=None`

2.2.3 Log Likelihood

The log likelihood `logp` of observing `Y` and `dY`, given the locations and measurement errors `X`, `Sigma`, `dX` and `dSigma`, and the hyperparameters `theta`

$$\ln p(\boldsymbol{Y}, \boldsymbol{dY} | \boldsymbol{X}, \boldsymbol{\sigma}, \boldsymbol{dX}, \boldsymbol{d\sigma}, \boldsymbol{\theta})$$

can be calculated with:

log_likelihood(*theta=default, dX=default, dY=default, dSigma=default, mu=default, dmu=default, muargs=default, prior=default, priorargs=default, scale=default*)

This is the log likelihood that is maximised to train the hyperparameters.

Parameters:

By default all parameters are set to the current attribute values. These values can be changed in the function call of `hypertrain`, which changes the attribute value permanently.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

Returns:

`logp`

2.2.4 Covariances between f(x) and its derivatives

The covariances between $f(x)$, $f'(x)$, $f''(x)$ and $f'''(x)$ at points `Xstar` can be calculated with:

f_covariances(*fclist=[0,1,2,3]*)

Parameters:

fclist:

Defines which covariances are calculated. 0 stands for $f(x)$, 1 for $f'(x)$, 2 for $f''(x)$ and 3 for $f'''(x)$.

Returns:

(nstar, m, m) array, where m is the length of `fclist`, i.e. for each point in `Xstar` a m×m covariance matrix is returned.

Example:

For a Gaussian process initialized as `g`,
`fcov = g.f_covariances(fclist = [1, 2])`
would calculate the covariances between $f'(x_i)$ and $f''(x_i)$ at points x_i. For each i in $[0, \texttt{nstar}]$, `fcov[i, :, :]` contains a covariance matrix

$$\begin{pmatrix} \mathrm{var}(f'(x_i)) & \mathrm{cov}(f'(x_i), f''(x_i)) \\ \mathrm{cov}(f'(x_i), f''(x_i)) & \mathrm{var}(f''(x_i)) \end{pmatrix}$$

Note that the order of the numbers in `fclist` does not change the output, i.e. `f_covariances(fclist=[2,1])` would give the same result as `f_covariances(fclist=[1,2])`.

2.3 Changing parameters of the Gaussian process

Parameters can also be changed without performing a reconstruction or training the hyperparameters. For more detailed descriptions of the parameters see section 2.1.

set_data(*X, Y, Sigma*)
Change the observational data (measurements of $f(x)$).

set_ddata(*dX, dY, dSigma*)
Change the observational data (measurements of $f'(x)$).

set_theta(*theta*)
Set the hyperparameters `theta`.

set_Xstar(*Xstar*)
Set locations `Xstar`, where $f(x)$ is to be reconstructed.

create_Xstar(*xmin, xmax, nstar*)
Create `Xstar` with `nstar` values between `xmin` and `xmax`.

set_mu(*mu, muargs=()*)
Set a priori mean function `mu` of the Gaussian Process. `muargs` is a list of additional arguments passed to `mu`. For `mu=None`, no mean function will be used in the Gaussian Process.

set_dmu(*dmu*)
Set the first derivative of the a priori mean function `dmu` of the Gaussian Process.

set_d2mu(*d2mu*)
Set the second derivative of the a priori mean function `d2mu` of the Gaussian Process.

set_d3mu(*d3mu*)
Set the third derivative of the a priori mean function `d3mu` of the Gaussian Process.

unset_mu()
Sets `mu`, `dmu`, `d2mu` and `d3mu` to `None`, i.e. no mean function will be used in the Gaussian Process.

set_prior(*prior, gradprior=None, priorargs=()*)
Set prior on the hyperparameters `theta`. If `grad=='True'`, you also need to provide the gradient of the prior, `gradprior`. `priorargs` is a list of additional parameters passed to `prior`. For `prior=None`, no prior will be used on the hyperparameters.

unset_prior()
Same effect as `set_theta(None)`.

set_scale(*scale*)
Set scale to a number or to `None`. Note that setting a new `scale`, resets `scaletrain` to the initial default values, i.e. `scaletrain = 'False'` if `scale == None` and `scaletrain = 'True'` else.

unset_scale()
Same effect as `set_scale(None)`.

set_scaletrain(*scaletrain*)
Set parameter `scaletrain`.

set_thetatrain(*thetatrain*)
Set parameter `thetatrain`.

set_grad(*grad='True'*)
Set the parameter `grad` to `'True'` if the gradient of the covariance function is to be used for the training of the hyperparameters, and to `'False'` else.

unset_grad()
Same effect as `set_grad('False')`, i.e. the gradient of the covariance function will not be used for the training of the hyperparameters.

Example:

If the Gaussian Process has been initialized as `g`, parameters can be changed in the following way:

```
g.create_Xstar(0, 10, 500)
g.set_thetatrain('False')
```

3 mcmcdgp module

The module `mcmcdgp` can be used to sample the hyperparameters using the MCMC package `emcee` – a Python implementation of Goodman & Weare's affine invariant MCMC ensemble sampler. The sample of hyperparameters can be used to sample the probability distribution of the reconstructed function. The function itself as well as the first, second and third derivatives can be reconstructed. Measurements of the first derivative $f'(x)$ can be used in addition to measurements of the function $f(x)$.

3.1 Initializing a Gaussian process

As a first step, the Gaussian Process needs to be initialized with

`mcmcdgp.`**MCMCDGaussianProcess**(*X, Y, Sigma, theta0, Niter=100, reclist=[0], covfunction=covariance.SquaredExponential, dX=None, dY=None, dSigma=None, Xstar=None, cXstar=None, mu=None, dmu=None, d2mu=None, d3mu=None, muargs=(), prior=None, priorargs=(), scale0=None, a=2.0, threads=1, nacor=10, nsample=50, sampling='True'*)

Parameters:

X: array_like,
A vector of length n containing the locations x_i of the observations. n is the number of data points.

Y: array_like
Vector containing the observations of $f(X)$.

Sigma: array_like
Either a vector of length n containing the measurement errors of `Y`, or a $n \times n$ covariance matrix of the data.

theta0: array_like
(`nwalker`, `ntheta`) array, where `ntheta` is the number of hyperparameters and `nwalker` is the number of walkers for the MCMC. `theta0` contains either the initial distribution of hyperparameters if `sampling='True'`, or the hyperparameter sample that is used for the reconstruction if `sampling='False'`.
`theta0` (plus `scale0`) corresponds to the initial position of the walkers `p0` in `emcee`.

Niter: int
Number of iterations of the MCMC sampling, i.e. code will return `Niter`×`nwalker` samples for `theta`.

reclist: tuple
Defines which reconstructions are performed. 0 stands for $f(x)$, 1 for $f'(x)$, 2 for $f''(x)$ and 3 for $f'''(x)$.

covfunction: class or tuple (class, class)
Covariance function. See section 6 for the covariance functions provided by the module `covariance`.
If `covfunction` is given in the form `(covfunction1, covfunction2)`, the sum of `covfunction1` and `covfunction2` will be used as covariance function (see section 6.2 for details).

dX: array_like or number,
A vector of length n' containing the locations x_i of the observations of the first derivative f'. n' is the number of data points. For $n' = 1$, `dX` can be a number.

dY: array_like or number
Vector (or number, if $n' = 1$) containing the observations of f'(X).

dSigma: array_like or number
Either a vector of length n' containing the measurement errors of `dY`, or a $n' \times n'$ covariance matrix of the data. If `dSigma` is a number (if $n' = 1$), it is interpreted as a measurement error and *not* a covariance matrix.

Xstar: array_like
Vector containing the locations, where f(x) is to be reconstructed.

cXstar: tuple (`xmin`, `xmax`, `nstar`)
`Xstar` will be created automatically. `nstar` values are created between `xmin` and `xmax`. If `xmin==None` or `xmax==None`, their respective values will be determined automatically:

$$\texttt{xmin} = \min(\texttt{X}) - 0.1(\max(\texttt{X}) - \min(\texttt{X})), \quad \texttt{xmax} = \max(\texttt{X}) + 0.1(\max(\texttt{X}) - \min(\texttt{X}))$$

If `Xstar` is provided explicitly, `cXstar` will be ignored.

mu: callable `mu(x, *muargs)`
A priori mean function of the Gaussian Process. The first argument of `mu` is `x`.

dmu: callable `dmu(x, *muargs)`
First derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `dmu` will be ignored.

d2mu: callable `d2mu(x, *muargs)`
Second derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `d2mu` will be ignored.

d3mu: callable `d3mu(x, *muargs)`
Third derivative of the a priori mean function of the Gaussian Process. If `mu==None`, `d3mu` will be ignored.

muargs: tuple
Additional arguments passed to `mu`, `dmu`, `d2mu` and `d3mu`.

prior: callable `prior(theta, *priorargs)`
Prior on the hyperparameters `theta`. The first argument of `prior` must be `theta`. Must return a non-negative number. If `prior` depends on additional parameters, they have to be given by `priorargs`.

priorargs: tuple
Additional arguments passed to the functions prior and gradprior.

scale0: array_like or `None`
Vector of length `nwalker` or (`nwalker`, 2) array, where `nwalker` is the number of walkers for the MCMC, or `None`.
If `scale0` is a vector, it contains either a sample or the initial positions of `scale`. For uncorrelated data, the errors σ_i of the observations of $f(x)$ are multiplied by `scale`. For correlated data, the covariance matrix of the observations is multiplied by $\texttt{scale}^2$.
If `scale0` is a (`nwalker`, 2) array, the second column is used to scale the errors of the observations of $f'(x)$.
`theta0` plus `scale0` corresponds to the initial position of the walkers `p0` in `emcee`.

a: number
The proposal scale parameter of `emcee`. This parameter probably does not need to be changed.

threads: int
Number of threads used for the parallelization.

nacor: int ≥ 10

`nacor`×`acor` burn-in steps are used for the MCMC, where `acor` is the autocorrelation time as estimated by the `acor` package.

nsample: int

Number of samples of the function distribution for each set of hyperparameters `theta` and each point in `Xstar`, i.e. at each point one obtains `nsample`×`Niter`×`nwalker` samples of the probability distribution of the reconstructed function.

sampling: 'True' or 'False'

'True': The hyperparameters are sampled using `emcee` starting from initial positions `theta0` (and `scale0`). The result is then used to sample the function distribution.
'False': `theta0` (and `scale0`) are directly used to sample the function distribution.

The number of walkers `nwalker` in `theta0` and `scale0` needs to be the same.

3.2 Gaussian Process Regression

Once the Gaussian process has been initialized, the reconstructions of the function and its derivatives are performed with:

mcmcdgp()

The parameters of the Gaussian process cannot be changed in this step. The parameters from the initialization are used.

`mcmcdgp()` performs the sampling of the distribution of the reconstructed function as follows: First the hyperparameters (and the scale) are sampled (if `sampling='True'`) using the affine invariant MCMC sampler `emcee` and using `theta0` (and `scale0`) as initial positions for the walkers. This step is equivalent to running `mcmc_sampling()` (see section 3.3).

The sampling of the hyperparameters can be skipped by setting `sampling='False'`. In this case, `theta0` (and `scale0`) are not interpreted as initial positions, but as the actual sample of the hyperparameters (and the scale).

For each set of values of the hyperparameters (and the scale) in the sample, a Gaussian process regression is performed, leading to a mean and standard deviation of the reconstructed function at points `Xstar`. At each point of `Xstar`, `nsample` function values are drawn from the Gaussian distribution given by that mean and standard deviation, thus sampling the probability distribution of the reconstructed function.

Returns:

`Xstar` and samples of the reconstruction as (`nstar`, `nwalker` × `Niter` × `nsample`) arrays. Which samples are returned depends on the parameter `reclist`. The order of the samples is always from the lowest to the highest derivative, independent of the order of the numbers in `reclist`.

Examples of return values:

`(Xstar, fsample, dfsample, d2fsample, d3fsample)`	for `reclist=[0,1,2,3]`
`(Xstar, fsample, d2fsample)`	for `reclist=[0,2]`
`(Xstar, fsample, dfsample, d3fsample)`	for `reclist=[3,0,1]`

Warning: $f(x_i^*)$, $f'(x_i^*)$, $f''(x_i^*)$ and $f'''(x_i^*)$ are not independent quantities. The covariances between these quantities are taken into consideration when the samples are drawn. These covariances are important if f and its derivatives are used to reconstruct a function $h(f, f', f'', f''')$. While performing independent reconstructions using `reclist=[0]`, `reclist=[1]`, ..., will give correct results for f, f', f'' and f''', this approach treats f and its derivatives as independent quantities and thus leads to an incorrect reconstruction of h.

If you are planning to reconstruct a function h, always perform the required reconstructions of f, f', f'' and f''' in one go. Thus the covariances between these quantities are correctly taken into account.

Attributes:

After the reconstructions are performed, the samples of the hyperparameters (and the scale) are available:

thetasample: (`nwalker` × `Niter` × `nsample`, `ntheta`) array
Sample of the hyperparameters.

scalesample: (`nwalker` × `Niter` × `nsample`) or (`nwalker` × `Niter` × `nsample`, 2) array
Sample of the scale.

3.3 Sampling of the hyperparameters

The sampling of the hyperparameters (and the scale) can be performed without the subsequent sampling of the function distribution:

mcmc_sampling()

No values are directly returned, but the samples of the hyperparameters (and the scale) are then available as attributes:

thetasample: (`nwalker` × `Niter` × `nsample`, `ntheta`) array
Sample of the hyperparameters.

scalesample: (`nwalker` × `Niter` × `nsample`) or (`nwalker` × `Niter` × `nsample`, 2) array
Sample of the scale.

4 gp module – Gaussian Processes with multi-dimensional inputs

This section describes the module `gp`. It can handle multi-dimensional inputs x_i, but cannot be used to reconstruct derivatives of a function. Optimized hyperparameters are used for the reconstruction. If you are interested in reconstructing derivatives, use the module `dgp` (section 2).

4.1 Initializing a Gaussian process

As a first step, the Gaussian Process needs to be initialized with

`gp.`**GaussianProcess**(*X, Y, Sigma, covfunction=covariance.SquaredExponential, theta=None, Xstar=None, cXstar=None, mu=None, muargs=(), prior=None, gradprior=None, priorargs=(), thetatrain='True', scale=None, scaletrain='True', grad='True'*)

Parameters:

X: array_like,
A vector or matrix containing the locations x_i of the observations. Its shape is $(n,)$ or (n, d), where n is the number of data points and d is the number of dimensions.

Y: array_like
Vector containing the observations of $f(X)$.

Sigma: array_like
Either a vector of length n containing the measurement errors of `Y`, or a $n \times n$ covariance matrix of the data.

covfunction: class or tuple (class, class)
Covariance function. See section 6 for the covariance functions provided by the module `covariance`.
If `covfunction` is given in the form `(covfunction1, covfunction2)`, the sum of `covfunction1` and `covfunction2` will be used as covariance function (see section 6.2 for details).

theta: tuple
(Initial) values of the hyperparameters of the covariance function. For covariance functions with only two hyperparameters, `theta[0]` denotes the signal variance σ_f and `theta[1]` the length scale ℓ. (See section 6 for the meaning of `theta` for the specific covariance functions.)

Xstar: array_like
Vector or matrix containing the locations, where f(x) is to be reconstructed.

cXstar: tuple (xmin, xmax, nstar)
`Xstar` will be created automatically.
In a one dimensional setting, `nstar` values are created between `xmin` and `xmax`.
In a multi-dimensional setting, `xmin` and `xmax` are vectors of length d. (If `xmin` and `xmax` are given as scalars, they are interpreted as vectors of length d with entries `xmin` and `xmax`, respectively.) `nstar` is either a number or a vector of length d. `nstar`^{d} or $\prod_{i=1}^{d}$ `nstar`$_i$ values are created.
If `xmin==None` or `xmax==None`, their respective values will be determined automatically:

$$\texttt{xmin} = \min(\texttt{X}) - 0.1(\max(\texttt{X}) - \min(\texttt{X})), \quad \texttt{xmax} = \max(\texttt{X}) + 0.1(\max(\texttt{X}) - \min(\texttt{X}))$$

If `Xstar` is provided explicitly, `cXstar` will be ignored.

mu: callable `mu(x, *muargs)`
A priori mean function of the Gaussian Process. The first argument of `mu` is `x`.

muargs: tuple
Additional arguments passed to `mu`.

prior: callable `prior(theta, *priorargs)`
Prior on the hyperparameters `theta`. The first argument of `prior` must be `theta`. Must return a non-negative number. If `prior` depends on additional parameters, they have to be given by `priorargs`.

gradprior: callable `gradprior(theta, *priorargs)`
Function that returns a tuple containing the gradient of `prior` with respect to `theta`. `gradprior` needs to be provided if `prior!=None` and `grad=='True'`.

priorargs: tuple
Additional arguments passed to the functions prior and gradprior.

thetatrain: `'True'`, `'False'` or tuple
Defines which hyperparameters are to be trained.
`'True'`: all hyperparameters will be optimized.
`'False'`: values of the hyperparameters are fixed.
tuple of the length of the number of hyperparameters with entries `0` and `1`: the hyperparameter `theta[i]` will be trained if `thetatrain[i] = 1`, and is fixed for `thetatrain[i] = 0`.

scale: number or None
If `scale!=None`, the data covariance matrix `Sigma` is multiplied by $\texttt{scale}^2$, i.e. for uncorrelated data, σ_i is multiplied by `scale`.
If `scale!=None`, the value of `scale` is optimized by default.

scaletrain: `'True'` or `'False'`
Defines if `scale` is optimized.
`'True'`: `scale` will be optimized.
`'False'`: `scale` will not be optimized.

grad: `'True'` or `'False'`
'True' if the gradient of the covariance function is to be used to train the hyperparameters. This training method is faster than the alternative method. If `grad=='True'` and `prior!=None`, one needs to provide the gradient of the prior `gradprior`.

The only required parameters are the measurements `X`, `Y` and `Sigma`. By default, the squared exponential covariance function is used.

If no initial values for the hyperparameters `theta` are provided, the programm guesses somewhat reasonable initial values based on the range of `X` and `Y` values. It is, however, recommended that you choose the initial values yourself. If e.g. the function seems to be oscillating at a high frequency, you should choose a rather small initial value for the length scale ℓ to avoid a covariance matrix that is not positive definite. Should the program end with the error message "Matrix is not positive definite - Cholesky decomposition cannot be computed", try adjusting the initial values of `theta`.

The hyperparameters are trained by maximizing the marginal likelihood $p(\texttt{Y}|\texttt{X}, \texttt{theta})$. As this likelihood might have multiple local maxima, you should try different initial values for `theta` to avoid ending up in the wrong local maximum. This can also be avoided by using suitable priors on the hyperparameters.

If neither `Xstar` nor `cXstar` are provided, `Xstar` is created automatically with 200 values between `xmin` and `xmax`, which are determined from the range of `X` values.

Example: Inverse Gamma prior on `theta[0]`

```
from gapp import gp
from numpy import exp
from scipy.special import gamma
X = [6.04, 0.86, 2.11, 9.54, 8.58]
Y = [2.29, 0.39, -0.56, -3.12, -2.12]
Sigma = [0.28, 0.36, 0.22, 0.19, 0.23]
def invgamma(theta,a,b):
    s = theta[0]
    p = b**a/gamma(a) * s**(-1-a) * exp(-b/s)
    return p
g = gp.GaussianProcess(X, Y, Sigma, theta=[1,1], prior=invgamma,
                       priorargs=(0.2,1), grad='False')
```

4.2 Gaussian Process Regression

4.2.1 Full Gaussian Process Run

Once the Gaussian Process has been initialized, the actual function reconstruction is performed with:

gp(*theta=default, Xstar=default, cXstar=default, mu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, grad=default, unpack='False'*)

Parameters:

By default almost all parameters (exceptions are mentioned below) are set to the current attribute values. These values can be changed in the function call of `gp`, which changes the attribute value permanently (the only exception is `unpack`, which always defaults to `'False'`). It is not possible to change the observational values in the function call, as the data are considered to be fixed for a certain Gaussian Process. If for some reason, you need to change these data, use `set_data(X,Y,Sigma)` as described in section 2.3.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

scale, scaletrain:

If you set a new `scale`, but do not provide `scaletrain`, `scaletrain` will be set to the initial default values, i.e. `scaletrain = 'False'` if `scale == None`, and else `scaletrain = 'True'`.

There is one additional parameter that determines the form of the function return:

unpack : 'True' or 'False'

Defaults to `'False'`, even if it has been set to `'True'` in a previous function call.
`'True'`: function returns `(Xstar, fmean, fstd, theta)` if `scale==None` or `(Xstar, fmean, fstd, theta, scale)` if `scale!=None`. `fmean` and `fstd` are vectors containing the mean and standard deviation of the reconstructed function at locations `Xstar`, respectively.
`'False'`: function returns `(rec, theta)` if `scale==None` or `(rec, theta, scale)` if `scale!=None`. `rec` is an array containing `Xstar`, `fmean` and `fstd`.

Returns:

`(Xstar, fmean, fstd, theta)` if `unpack=='True'` and `scale==None`
`(Xstar, fmean, fstd, theta, scale)` if `unpack=='True'` and `scale!=None`
`(rec, theta)` if `unpack=='False'` and `scale==None`
`(rec, theta, scale)` if `unpack=='False'` and `scale!=None`

If the hyperparameters are trained, messages from the optimisation and the optimised hyperparameters are displayed.

Example:

Assuming you have initialized the Gaussian Process according to the example in the previous section, you can perform the reconstruction using the attribute values specified in the initialization, or you can change the attributes in the function call.

```
(Xstar,fmean,fstd,theta1) = g.gp(unpack='True')
(rec,theta2) = g.gp(prior=None)
```

The attribute `prior` of `g` is now permanently set to None, i.e. when subsequently performing `g.gp()`, no prior will be used.

4.2.2 Training the hyperparameters

If one is not interested in a whole Gaussian Process run, but only in training the hyperparameters, this can be done with:

hypertrain(*theta=default, mu=default, muargs=default, prior=default, gradprior=default, priorargs=default, thetatrain=default, scale=default, grad=default*)

Parameters:

By default all parameters are set to the current attribute values. These values can be changed in the function call of `hypertrain`, which changes the attribute value permanently.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

scale, scaletrain:

If you set a new `scale`, but do not provide `scaletrain`, `scaletrain` will be set to the initial default values, i.e. `scaletrain = 'False'` if `scale == None`, and else `scaletrain = 'True'`.

Returns:

`theta` if `scale==None`
`(theta, scale)` if `scale!=None`

4.2.3 Log Likelihood

The log likelihood `logp` of observing `Y` and `dY`, given the locations and measurement errors `X`, `Sigma`, `dX` and `dSigma`, and the hyperparameters `theta`

$$\ln p(\boldsymbol{Y}, \boldsymbol{dY} | \boldsymbol{X}, \boldsymbol{\sigma}, \boldsymbol{dX}, \boldsymbol{d\sigma}, \boldsymbol{\theta})$$

can be calculated with:

log_likelihood(*theta=default, mu=default, dmu=default, muargs=default, prior=default, priorargs=default, scale=default*)

This is the log likelihood that is maximised to train the hyperparameters.

Parameters:

By default all parameters are set to the current attribute values. These values can be changed in the function call of `hypertrain`, which changes the attribute value permanently.

mu, muargs:

If you set a new `mu`, then `muargs=()` by default. Thus `muargs` has to be provided if `mu` depends on additional parameters.

prior, gradprior, priorargs:

If you set a new `prior`, but do not provide `gradprior`, `gradprior` will be set to `None`. `priorargs=()` by default. Thus `priorargs` has to be provided if `prior` depends on additional parameters.

Returns:

`logp`

4.3 Changing parameters of the Gaussian process

Parameters can also be changed without performing a reconstruction or training the hyperparameters. For more detailed descriptions of the parameters see section 4.1.

set_data(*X, Y, Sigma*)
Change the observational data.

set_theta(*theta*)
Set the hyperparameters `theta`.

set_Xstar(*Xstar*)
Set locations `Xstar`, where $f(x)$ is to be reconstructed.

create_Xstar(*xmin, xmax, nstar*)
Create `Xstar` with `nstar` values between `xmin` and `xmax`.

set_mu(*mu, muargs=()*)
Set a priori mean function `mu` of the Gaussian Process. `muargs` is a list of additional arguments passed to `mu`. For `mu=None`, no mean function will be used in the Gaussian Process.

unset_mu()
Same effect as `set_mu(None)`, i.e. no mean function will be used in the Gaussian Process.

set_prior(*prior, gradprior=None, priorargs=()*)
Set prior on the hyperparameters `theta`. If `grad=='True'`, you also need to provide the gradient of the prior, `gradprior`. `priorargs` is a list of additional parameters passed to `prior`. For `prior=None`, no prior will be used on the hyperparameters.

unset_prior()
Same effect as `set_theta(None)`.

set_scale(*scale*)
Set scale to a number or to `None`. Note that setting a new `scale`, resets `scaletrain` to the initial default values, i.e. `scaletrain = 'False'` if `scale == None` and `scaletrain = 'True'` else.

unset_scale()
Same effect as `set_scale(None)`.

set_scaletrain(*scaletrain*)
Set parameter `scaletrain`.

set_thetatrain(*thetatrain*)
Set parameter `thetatrain`.

set_grad(*grad='True'*)
Set the parameter `grad` to `'True'` if the gradient of the covariance function is to be used for the training of the hyperparameters, and to `'False'` else.

unset_grad()
Same effect as `set_grad('False')`, i.e. the gradient of the covariance function will not be used for the training of the hyperparameters.

Example:

If the Gaussian Process has been initialized as `g`, parameters can be changed in the following way:

```
g.create_Xstar(0, 10, 500)
g.set_thetatrain('False')
```

5 mcmcgp module

The module `mcmcgp` can be used to sample the hyperparameters using the MCMC package `emcee` – a Python implementation of Goodman & Weare's affine invariant MCMC ensemble sampler. The sample of hyperparameters can be used to sample the probability distribution of the reconstructed function. The module can handle multi-dimensional inputs x_i, but cannot be used to reconstruct derivatives of a function.

5.1 Initializing a Gaussian process

As a first step, the Gaussian Process needs to be initialized with

`mcmcgp.`**MCMCGaussianProcess**(*X, Y, Sigma, theta0, Niter=100, covfunction=covariance.SquaredExponential, Xstar=None, cXstar=None, mu=None, muargs=(), prior=None, priorargs=(), scale0=None, a=2.0, threads=1, nacor=10, nsample=50, sampling='True'*)

Parameters:

X: array_like,
A vector or matrix containing the locations x_i of the observations. Its shape is $(n,)$ or (n, d), where n is the number of data points and d is the number of dimensions.

Y: array_like
Vector containing the observations of $f(X)$.

Sigma: array_like
Either a vector of length n containing the measurement errors of `Y`, or a $n \times n$ covariance matrix of the data.

theta0: array_like
(`nwalker`, `ntheta`) array, where `ntheta` is the number of hyperparameters and `nwalker` is the number of walkers for the MCMC. `theta0` contains either the initial distribution of hyperparameters if `sampling='True'`, or the hyperparameter sample that is used for the reconstruction if `sampling='False'`.
`theta0` (plus `scale0`) corresponds to the initial position of the walkers `p0` in `emcee`.

Niter: int
Number of iterations of the MCMC sampling, i.e. code will return `Niter`×`nwalker` samples for `theta`.

covfunction: class or tuple (class, class)
Covariance function. See section 6 for the covariance functions provided by the module `covariance`.
If `covfunction` is given in the form `(covfunction1, covfunction2)`, the sum of `covfunction1` and `covfunction2` will be used as covariance function (see section 6.2 for details).

Xstar: array_like
Vector containing the locations, where f(x) is to be reconstructed.

cXstar: tuple (`xmin`, `xmax`, `nstar`)
`Xstar` will be created automatically.
In a one dimensional setting, `nstar` values are created between `xmin` and `xmax`.
In a multi-dimensional setting, `xmin` and `xmax` are vectors of length d. (If `xmin` and `xmax` are given as scalars, they are interpreted as vectors of length d with entries `xmin` and `xmax`, respectively.) `nstar` is either a number or a vector of length d. `nstar`^{d} or

$\prod_{i=1}^{d}$ `nstar`$_i$ values are created.
If `xmin==None` or `xmax==None`, their respective values will be determined automatically:

$$\texttt{xmin} = \min(\texttt{X}) - 0.1(\max(\texttt{X}) - \min(\texttt{X})), \quad \texttt{xmax} = \max(\texttt{X}) + 0.1(\max(\texttt{X}) - \min(\texttt{X}))$$

If `Xstar` is provided explicitly, `cXstar` will be ignored.

mu: callable `mu(x, *muargs)`
A priori mean function of the Gaussian Process. The first argument of `mu` is `x`.

muargs: tuple
Additional arguments passed to `mu`.

prior: callable `prior(theta, *priorargs)`
Prior on the hyperparameters `theta`. The first argument of `prior` must be `theta`. Must return a non-negative number. If `prior` depends on additional parameters, they have to be given by `priorargs`.

priorargs: tuple
Additional arguments passed to the functions prior and gradprior.

scale0: array_like or `None`
Vector of length `nwalker`, where `nwalker` is the number of walkers for the MCMC, or `None`.
`scale0` contains either a sample or the initial positions of `scale`. For uncorrelated data, the errors σ_i of the observations of $f(x)$ are multiplied by `scale`. For correlated data, the covariance matrix of the observations is multiplied by `scale`^{2}.
If `scale0` is a (`nwalker`, 2) array, the second column is used to scale the errors of the observations of $f'(x)$.
`theta0` plus `scale0` corresponds to the initial position of the walkers `p0` in `emcee`.

a: number
The proposal scale parameter of `emcee`. This parameter probably does not need to be changed.

threads: int
Number of threads used for the parallelization.

nacor: int ≥ 10
`nacor`×`acor` burn-in steps are used for the MCMC, where `acor` is the autocorrelation time as estimated by the `acor` package.

nsample: int
Number of samples of the function distribution for each set of hyperparameters `theta` and each point in `Xstar`, i.e. at each point one obtains `nsample`×`Niter`×`nwalker` samples of the probability distribution of the reconstructed function.

sampling: 'True' or 'False'
'True': The hyperparameters are sampled using `emcee` starting from initial positions `theta0` (and `scale0`). The result is then used to sample the function distribution.
'False': `theta0` (and `scale0`) are directly used to sample the function distribution.

The number of walkers `nwalker` in `theta0` and `scale0` needs to be the same.

5.2 Gaussian Process Regression

Once the Gaussian process has been initialized, the reconstruction of the function is performed with:

mcmcgp()

The parameters of the Gaussian process cannot be changed in this step. The parameters from the initialization are used.

`mcmcgp()` performs the sampling of the distribution of the reconstructed function as follows: First the hyperparameters (and the scale) are sampled (if `sampling='True'`) using the affine invariant MCMC sampler `emcee` and using `theta0` (and `scale0`) as initial positions for the walkers. This step is equivalent to running `mcmc_sampling()` (see section 5.3).

The sampling of the hyperparameters can be skipped by setting `sampling='False'`. In this case, `theta0` (and `scale0`) are not interpreted as initial positions, but as the actual sample of the hyperparameters (and the scale).

For each set of values of the hyperparameters (and the scale) in the sample, a Gaussian process regression is performed, leading to a mean and standard deviation of the reconstructed function at points `Xstar`. At each point of `Xstar`, `nsample` function values are drawn from the Gaussian distribution given by that mean and standard deviation, thus sampling the probability distribution of the reconstructed function.

Returns:

`(Xstar, fsample)`

where `fsample` is the sample of the reconstruction of the function. It is a (`nstar`, `nwalker` × `Niter` × `nsample`) array.

Attributes:

After the reconstructions are performed, the samples of the hyperparameters (and the scale) are available:

thetasample: (`nwalker` × `Niter` × `nsample`, `ntheta`) array
Sample of the hyperparameters.

scalesample: (`nwalker` × `Niter` × `nsample`) array
Sample of the scale.

5.3 Sampling of the hyperparameters

The sampling of the hyperparameters (and the scale) can be performed without the subsequent sampling of the function distribution:

mcmc_sampling()

No values are directly returned, but the samples of the hyperparameters (and the scale) are then available as attributes:

thetasample: (`nwalker` × `Niter` × `nsample`, `ntheta`) array
Sample of the hyperparameters.

scalesample: (`nwalker` × `Niter` × `nsample`) array
Sample of the scale.

6 Covariance functions

6.1 General covariance functions

The following covariance functions are implemented in GaPP:

`SquaredExponential:` Squared exponential

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left(-\frac{(x-\tilde{x})^2}{2\ell^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`.

`DoubleSquaredExponential:` Sum of two squared exponentials

$$k(x,\tilde{x}) = \sigma_{f1}^2 \exp\left(-\frac{(x-\tilde{x})^2}{2\ell_1^2}\right) + \sigma_{f2}^2 \exp\left(-\frac{(x-\tilde{x})^2}{2\ell_2^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_{f1}`, `ℓ_1`, `σ_{f2}`, `ℓ_2`]`.

`Matern32:` Matérn ($\nu = 3/2$)

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\frac{\sqrt{3}\,|x-\tilde{x}|}{\ell}\right]\left(1+\frac{\sqrt{3}\,|x-\tilde{x}|}{\ell}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`. Can only be used for reconstruction of $f(x)$ and $f'(x)$.

`Matern52:` Matérn ($\nu = 5/2$)

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\frac{\sqrt{5}\,|x-\tilde{x}|}{\ell}\right]\left(1+\frac{\sqrt{5}\,|x-\tilde{x}|}{\ell}+\frac{5(x-\tilde{x})^2}{3\ell^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`. Can only be used for reconstruction of $f(x)$, $f'(x)$ and $f''(x)$.

`Matern72:` Matérn ($\nu = 7/2$)

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\frac{\sqrt{7}\,|x-\tilde{x}|}{\ell}\right]\left(1+\frac{\sqrt{7}\,|x-\tilde{x}|}{\ell}+\frac{14(x-\tilde{x})^2}{5\ell^2}+\frac{7\sqrt{7}\,|x-\tilde{x}|^3}{15\ell^3}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`.

`Matern92:` Matérn ($\nu = 9/2$)

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\frac{3\,|x-\tilde{x}|}{\ell}\right]\left(1+\frac{3\,|x-\tilde{x}|}{\ell}+\frac{27(x-\tilde{x})^2}{7\ell^2}+\frac{18\,|x-\tilde{x}|^3}{7\ell^3}+\frac{27(x-\tilde{x})^4}{35\ell^4}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`.

`Cauchy:` Cauchy

$$k(x,\tilde{x}) = \sigma_f^2 \frac{\ell}{(x-\tilde{x})^2+\ell^2}$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`]`.

`RationalQuadratic`: Rational quadratic

$$k(x,\tilde{x}) = \sigma_f^2 \left(1 + \frac{(x-\tilde{x})^2}{2\alpha\ell^2}\right)^{-\alpha}$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ`, `α`]`.

Note: The Matérn covariance functions can only be used for the reconstruction of the kth derivative of $f(x)$ if $k < \nu$ as they are k-times mean square differentiable.

6.2 Sum of two covariance functions

The sum of two covariance functions is again a covariance function. You can use the sum of any two covariance functions by initializing the Gaussian process with the parameter `covfunction=(covfunction1, covfunction2)`. `theta` is a tuple containing the hyperparameters of `covfunction1` and `covfunction2`.

Example: If we want to use the sum of a squared exponential and a rational quadratic covariance function, we would initialize the Gaussian process as
`g = dgp.DGaussianProcess(X, Y, Sigma, covfunction=(SquaredExponential, RationalQuadratic), theta=[sigmaf1, l1, sigmaf2, l2, alpha])`
where `sigmaf1` and `l1` are the hyperparameters of the squared exponential, and `sigmaf2`, `l2` and `alpha` are the hyperparameters of the rational quadratic.

Note: When using the sum of two covariance functions, you should fix some of the hyperparameters and/or use priors. It might for example be useful to constrain `l1` to small values and `l2` to large values. Thus different frequencies of the function can be captured. Not using any constraints at all will probably fail.

6.3 Covariance functions for multi-dimensional inputs

These covariance functions allow for an individual characteristic length scale ℓ for each dimension of the input space.

`MultiDSquaredExponential`: Squared exponential for inputs with dimension d

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left(-\sum_{i=1}^{d} \frac{(x_i-\tilde{x_i})^2}{2\ell_i^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`,...,` ℓ_d`]`.

`MultiDDoubleSquaredExponential`: Sum of two squared exponentials for inputs with dimension d

$$k(x,\tilde{x}) = \sigma_{f1}^2 \exp\left(-\sum_{i=1}^{d} \frac{(x_i-\tilde{x_i})^2}{2\ell_{1_i}^2}\right) + \sigma_{f2}^2 \exp\left(-\sum_{i=1}^{d} \frac{(x_i-\tilde{x_i})^2}{2\ell_{2_i}^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_{f1}`, `ℓ_{1_1}`,...,` ℓ_{1_d}`, `σ_{f2}`, `ℓ_{2_1}`,...,` ℓ_{2_d}`]`.

`MultiDMatern32`: Matérn ($\nu = 3/2$) for inputs with dimension d

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\sqrt{3\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}\right]\left(1+\sqrt{3\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`,...,` ℓ_d`]`.

`MultiDMatern52`: Matérn ($\nu = 5/2$) for inputs with dimension d

$$k(x,\tilde{x}) = \sigma_f^2 \exp\left[-\sqrt{5\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}\right]\left(1+\sqrt{5\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}+\frac{5}{3}\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2\right)$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`, ..., `ℓ_d`]`.
Can only be used for reconstruction of $f(x)$, $f'(x)$ and $f''(x)$.

`MultiDMatern72`: Matérn ($\nu = 7/2$) for inputs with dimension d

$$\begin{aligned} k(x,\tilde{x}) &= \sigma_f^2 \exp\left[-\sqrt{7\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}\right]\times \\ &\left(1+\sqrt{7\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}+\frac{14}{5}\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2+\frac{7\sqrt{7}}{15}\left[\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2\right]^{\frac{3}{2}}\right) \end{aligned}$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`, ..., `ℓ_d`]`.

`MultiDMatern92`: Matérn ($\nu = 9/2$) for inputs with dimension d

$$\begin{aligned} k(x,\tilde{x}) &= \sigma_f^2 \exp\left[-3\sqrt{\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}\right]\times \\ &\left(1+3\sqrt{\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2}+\frac{27}{7}\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2+\frac{18}{7}\left[\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2\right]^{\frac{3}{2}}+\frac{27}{35}\left[\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2\right]^2\right) \end{aligned}$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`, ..., `ℓ_d`]`.

`MultiDCauchy`: Cauchy for inputs with dimension d

$$k(x,\tilde{x}) = \sigma_f^2\left[\sqrt{\sum_{i=1}^{d}\ell_i^2}\left(1+\sum_{i=1}^{d}\left(\frac{x_i-\tilde{x_i}}{\ell_i}\right)^2\right)\right]^{-1}$$

The parameter `theta` is a tuple containing the hyperparameters: `theta = [`σ_f`, `ℓ_1`, ..., `ℓ_d`]`.

Note: The covariance functions for multi-dimensional inputs cannot be used to reconstruct derivatives of $f(x)$.

Acknowledgements

This work is funded by the NRF (South Africa) and the South African Square Kilometre Array Project.