Bayesian Optimization and Gaussian Processes

Liang DING

Sept 4, 2020

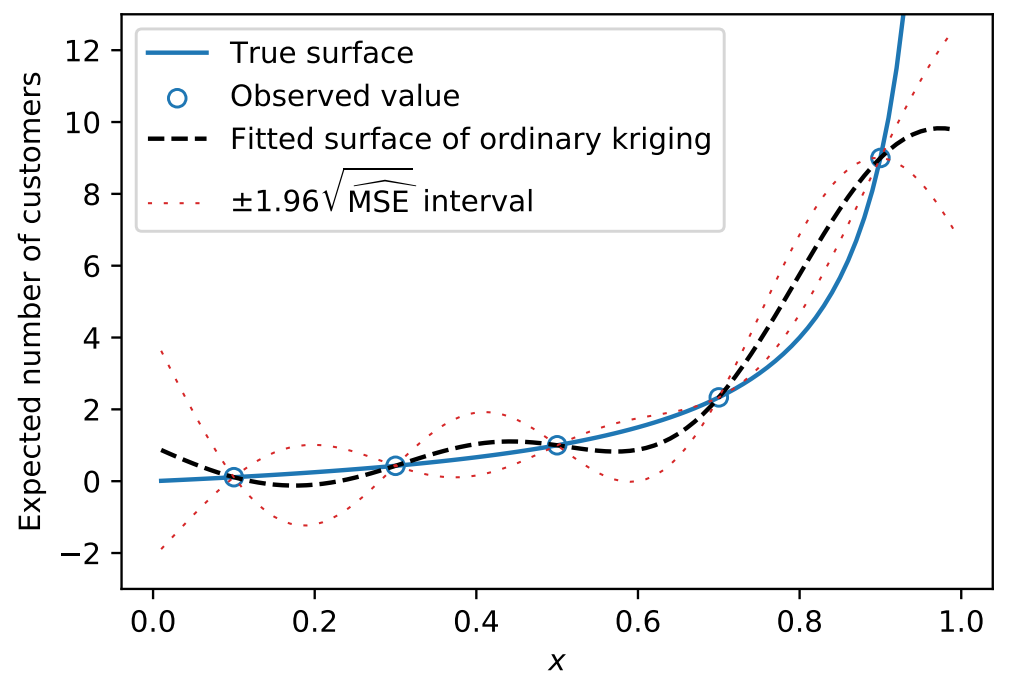
Texas A&M

Gaussian Process (GP)

Gaussian Process

- Generalization of multivariate normal distribution
  - $(\mathrm{M}(\boldsymbol{x}_1), \ldots, \mathrm{M}(\boldsymbol{x}_n))$ is multivariate normal
- Completely characterized by covariance function $k(\boldsymbol{x}, \boldsymbol{y})$
- Unknown function surface is viewed as a GP realization

$$\mathrm{Z}(\boldsymbol{x}) = \underbrace{\mu(\boldsymbol{x})}_{\text{mean}} + \underbrace{\mathrm{M}(\boldsymbol{x})}_{\text{GP}} + \underbrace{\varepsilon(\boldsymbol{x})}_{\text{noise}}, \quad \boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$$



GPML toolbox

- Download GPML (Rasmussen and Williams) toolbox at: `www.gaussianprocess.org/gpml/code/matlab/doc/`
- Run **startup** in Matlab command window to setup
- GPML toolbox is simple, there is only one single function to call – the **gp** function

```
1  function [varargout] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys)
2
3  % Two modes are possible: training or prediction: if no test cases are
4  % supplied, then the negative log marginal likelihood and its partial
5  % derivatives w.r.t. the hyperparameters is computed; this mode is used to fit
6  % the hyperparameters. If test cases are given, then the test set predictive
7  % probabilities are returned. Usage:
8  %
9  %   training: [nlZ dnlZ          ] = gp(hyp, inf, mean, cov, lik, x, y);
10 % prediction: [ymu ys2 fmu fs2   ] = gp(hyp, inf, mean, cov, lik, x, y, xs);
11 %         or: [ymu ys2 fmu fs2 lp] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys);
```

The gp function – argument

- hyp: a struct with the fields mean, cov and lik to specify the their hyperparameters respectively
- inf: inference methods specify how to compute the posterior (we use regression with Gaussian likelihood @infGaussLik)
- mean: mean function $\mu(\boldsymbol{x})$
- cov: covariance function $k(\boldsymbol{x}, \boldsymbol{y})$
- lik: likelihood function specifies the probability of the observations given the GP (we use the Gaussian likelihood)
- x: training inputs
- y: (noisy) training target
- xs: test input
- ys: test target

The gp function – output

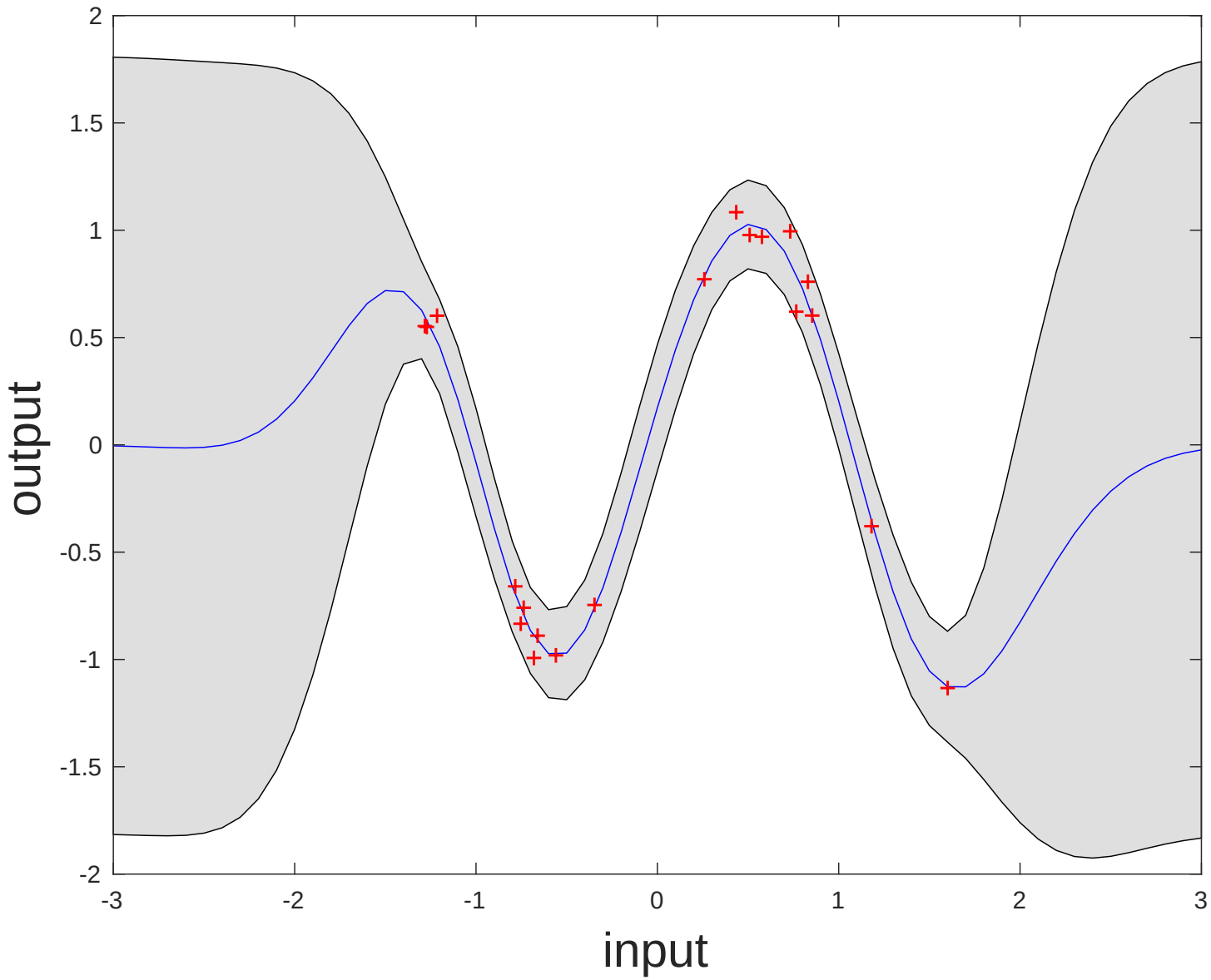
- nlz: negative log marginal likelihood
- dnlz: a struct for the partial derivatives of nlz wrt the hyperparameters
- ymu: test output mean (data mean)
- ys2: test output variance (data variance)
- fmu: latent GP mean
- fs2: latent GP variance
- lp: log predictive probabilities.

Demo

Script for an example:

```
x = gpml_randn(0.8, 20, 1);                 % 20  training inputs
y = sin(3*x) + 0.1*gpml_randn(0.9, 20, 1);  % 20 noisy training targets
xs = linspace(-3, 3, 61)';                  % 61 test inputs

% Specify the mean, covariance and likelihood functions
meanfunc = [];                    % empty: don't use a mean function
covfunc = @covSEiso;              % Squared Exponental covariance function
likfunc = @likGauss;              % Gaussian likelihood

% Initialize the hyperparameter struct
hyp = struct('mean', [], 'cov', [0 0], 'lik', -1);

% If hyperparameters are not known a priori, we can set hyperparameters by
% optimizing the (log) marginal likelihood:
hyp2 = minimize(hyp, @gp, -100, @infGaussLik, meanfunc, covfunc, likfunc, x, y);

% Make Prediction:
[mu s2] = gp(hyp2, @infGaussLik, meanfunc, covfunc, likfunc, x, y, xs);

% Make Plot:
f = [mu+2*sqrt(s2); flipdim(mu-2*sqrt(s2),1)];
fill([xs; flipdim(xs,1)], f, [7 7 7]/8)
hold on; plot(xs, mu); plot(x, y, '+')
```

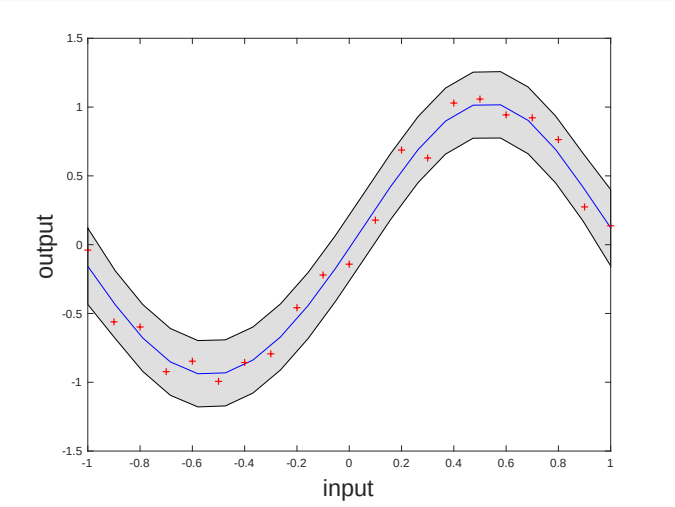
Demo

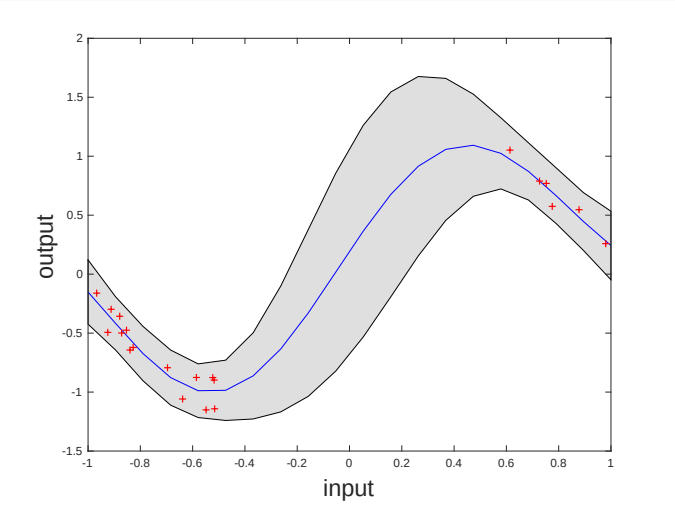


Experimental Design

Why Experimental Design

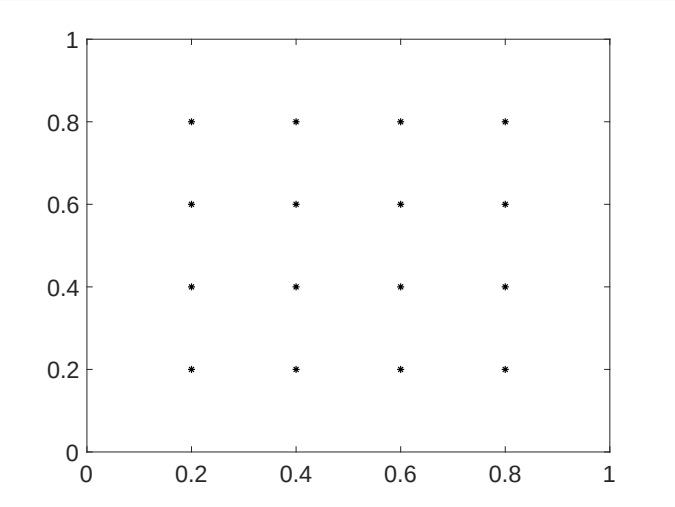
- Experiments with controllable input and physical response
- View the true response as a GP realization
- Experimental design improves overall prediction accuracy
- Good vs Bad:





Full Grid

- Make observations on equally spaced points:



- Advantage: computationally efficient (fast)
- Disadvantage: poor performance when input dimension is high (grid point number grows exponentially)

Full Grid

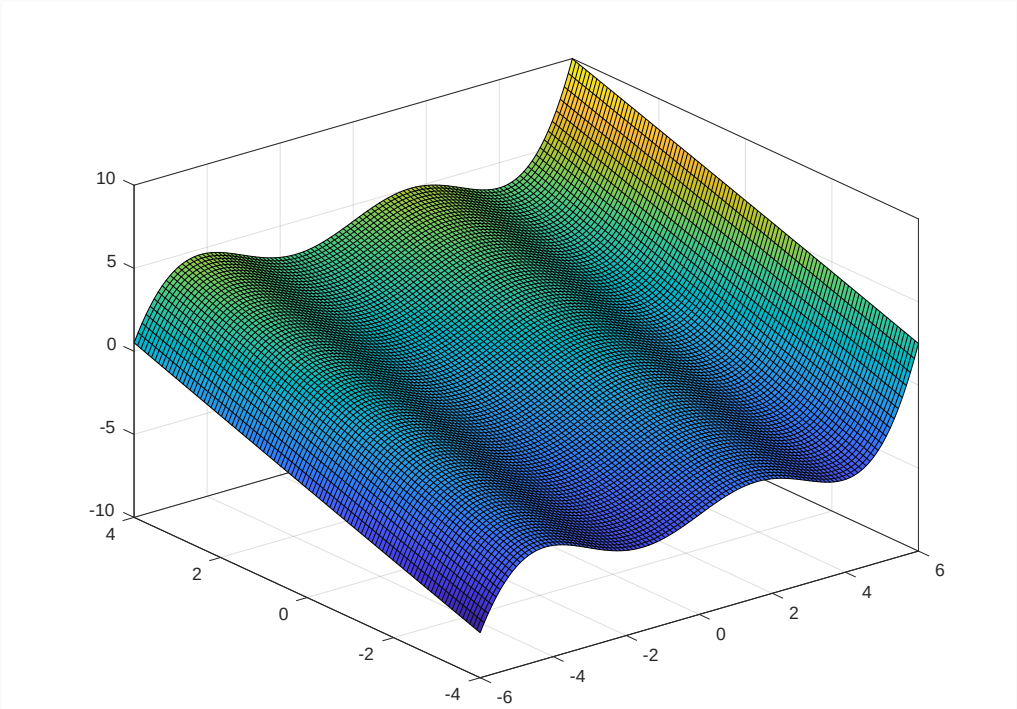
- Use kernel function of the form $k(\boldsymbol{x}, \boldsymbol{y}) = \prod_{j=1}^{d} k_j(x_j, y_j)$
- covariance matrix is the Kronecker product $K = \bigotimes_{j=1}^{d} K_d$
- Create full grid using GPML toolbox:

```
% A [-2,2]x[-3,3] full grid with 120*130 pixels
x1 = linspace(-2,2,120); x2 = linspace(-3,3,130);
x = apxGrid('expand',{x1',x2'});                          %15600
      training inputs
y = sin(x(:,2)) + x(:,1) + 0.1*gpml_randn(1,[size(x,1),1]);  %15600 noisy
      training tragets

% A [-4,4]x[-6,6] full grid to see the result of the extrapolation
xs1 = linspace(-4,4,100); xs2 = linspace(-6,6,110);
xs = apxGrid('expand',{xs1',xs2'});          %11000 test points

% Create a full grid covers both training data x and the test data xs
xg = apxGrid('create',[x;xs],true,[50,70]);
```

Full Grid

- search hyperparameters and make prediction:

```
% Hyperparameter optimisation with specified maximum number of iterations
% and the convergence threshold
  opt.cg_maxit = 200;                  %set maximum number of iterations
  opt.cg_tol = 5e-3;                   %set convergence thereshold
  infg = @(varargin) infGaussLik(varargin{:},opt);
  hyp = minimize(hyp,@gp,-50,infg,[],covg,[],x,y);

 % Compute predictions
  [post,nlZ,dnlZ] = infGrid(hyp,{@meanZero},covg,{@likGauss},x,y,opt);
  [fm,fs2,ym,ys2] = post.predict(xs);
```



SG & LHD

- SG & LHD no as fast as FG but accurate
- Latin Hypercube Design: **lhsdesign** (Matlab function)
- function to generate d-dimensional LHD of n points:

lhsdesign(n,d)

- Download Matlab sparse grid package (Plumlee) at `https://www.mathworks.com/matlabcentral/fileexchange/45668-sparse-grid-designs`
- function to generate d-dimensional SG of level n:

```
1  function sg = gensg(n,d,varargin)
2  % GENSG Create a Sparse Grid Design
3  %
4  % sg=gensg(n,d) returns a structure that constains information about a d
5  % demensional sparse grid design with level of constrcution n.
6  % The component designs are prechosen in an ad-hoc manor.
7  % sg = gensg(n,d,X) does the same operation using component designs in
8  % the structure of different sized vectors X of size d x k (k>n-d+1) where
       X{i+1,j} must contain X{i,j}.
```

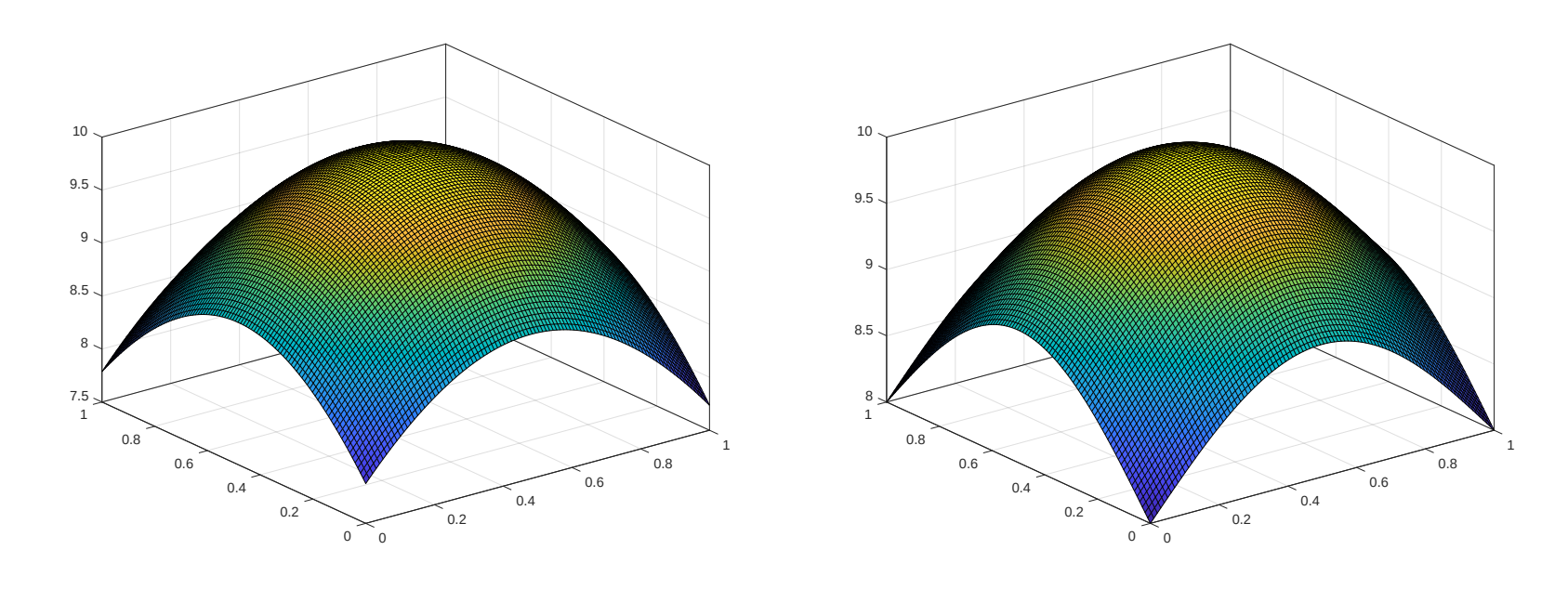
Sparse Grid & LHD

- A 10-dimensional example:

```
% Generate a 10-d sparse grid of level 3
x=gensg(13,10);x=x.design;      %2001 training inputs
% or x=lhsdesign(n,d)           %a LHD of n d-dimensional sample points
y = sum(sin(pi*x),2)+.1*gpml_randn(0.9, 2001, 1);  %2001 noisy training
     targets

% A 2-d full grid projection to see the result
xs1 = linspace(0,1,100); xs2 = linspace(0,1,100);
xs = [apxGrid('expand',{xs1',xs2'}) .5*ones(10000,8)];  %10000 test points


meanfunc = [];                      % empty: don't use a mean function
covfunc = @covSEiso;                % Squared Exponental covariance function
likfunc = @likGauss;                % Gaussian likelihood

% Initialize the hyperparameter struct
hyp = struct('mean', [], 'cov', zeros(2,1), 'lik', -1);

% Optimizing the (log) marginal likelihood:
hyp2 = minimize(hyp, @gp, -100, @infGaussLik, meanfunc, covfunc, likfunc,
     x, y);

% Make prediction
[mu s2] = gp(hyp2, @infGaussLik, meanfunc, covfunc, likfunc, x, y, xs);
```

SG & LHD

- 2-d projection of the prediction and the true function:



Prediction (left) and true function (right)

Bayesian Optimization

Bayesopt

- **bayesopt**: attempts to find values of vars that minimize the target function

$$\text{result} = \text{bayesopt(fun,vars)}$$

- **fun**: function target function (function handle)
- **vars**: created by Matlab function **optimizableVariable** with the specified name and range of values:

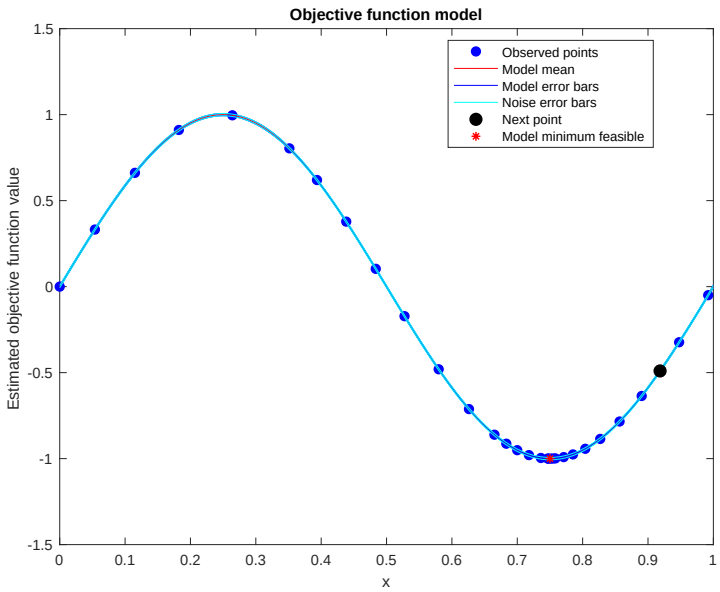
$$\text{vars} = \text{optimizableVariable(Name,Range))}$$

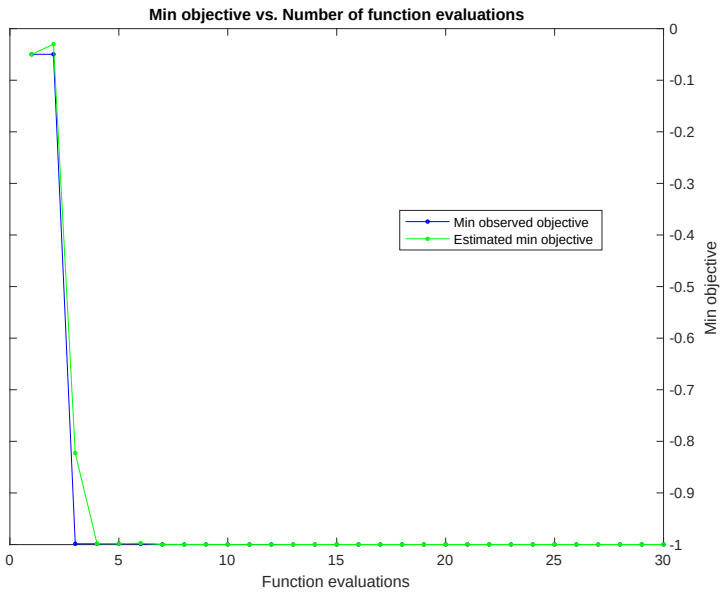
- **result:** an object contains the result of bayesopt

Bayesopt

- A toy example:

```
1 x= optimizableVariable('x',[0,1]);
2 func=@(x) sin(2*pi*x.x);
3 result=bayesopt(func,x);
```





More Details

- Acquisition Function (way of sampling):
  1. 'expected-improvement-per-second[1]-plus[2]' (default)
  2. 'expected-improvement'
  3. 'expected-improvement-plus'
  4. 'expected-improvement-per-second'
  5. 'lower-confidence-bound'
  6. 'probability-of-improvement'
- performance of acquisition function depends on the true function

[1] per-second: run time is varying for different region

[2] plus: add noise to escape local traps

More Details

- NumSeedPoints : number of initial *randomly selected* evaluation points
- InitialX : deterministic initial evaluation points X
- Example: run EI with initial designed $X = [.2; .4; .6; .8]$:

```
1 x= optimizableVariable('x',[0,1]);
2 func=@(x) sin(2*pi*x.x);
3 result=bayesopt(func,x,'InitialX',array2table
      ([.2;.4;.6;.8]));
```

More Details

- Stopping criteria
  1. By number of iterations. Example:

     'MaxObjectiveEvaluations',60

  2. By time (second). Example:

     'MaxTime',3600

Let's do it together

Use bayesopt to estimate the hyperparameters of a GP

Data Generation

```
% Generate some data from a Gaussian process

  ell = 1/4; sf = 1;  %true value
  covfunc = {@covMaterniso, 3};  hyp.cov = log([ell; sf
      ]);
  likfunc = @likGauss; sn = 0.1; hyp.lik = log(sn);

  n = 20;
  x = gpml_randn(0.3, n, 1);
  K = feval(covfunc{:}, hyp.cov, x);
  mu = feval(meanfunc{:}, hyp.mean, x);
  y = chol(K)'*gpml_randn(0.15, n, 1) + mu + exp(hyp.lik
      )*gpml_randn(0.2, n, 1);

save x x
save y y
```

Target Function

```
function [val] = target(x)
% create a target function that takes ell and sf as
    inputs
% and output the negative log likelihood
 load X
 load Y

 covfunc = {@covMaterniso, 3};
 likfunc = @likGauss;
 val=gp(struct('cov',log([x.ell ,x.sf]),'lik',log(.1)),
     @infGaussLik, [], covfunc, likfunc, X, Y);
end
```

Run bayesopt

```
ell=optimizableVariable('ell',[.1,1]);
sf=optimizableVariable('sf',[.1,1]);

f=@(x) target(x);

lhdX=array2table(lhsdesign(8,2));
bayesopt(f,[ell, sf],'InitialX',lhdX)
```