MA5232 Assignment Part 2

Gaussian Processes

Alon Rafael Landmann

A0276620E

May 17, 2024

Question 1

Part 1 (a)

The answers to 1 (a) are as follows.

	Smaller l	Medium l	Larger l
Smaller λ	(a)	(b)	(f)
Larger λ	(c)	(d)	(e)

Table 1: Curve assignments.

The parameter l determines the scale at which the function is free to vary, and so (a) and (c) share a small scale, (b) and (d) an intermediate scale, and (e) and (f) a much larger scale so that these latter two functions almost appear linear.

A larger λ reduces sensitivity to noise but adds bias, so that we are closer to the mean for (c), (d), and (e), compared to (a), (b), and (f) respectively.

Part 1 (b)

In this part we consider the hidden function $f(x) = \sin(3x) - \frac{1}{2}x^2 + 1$ on the domain $[-2, 2]$. We have sampled nine points with x-coordinates $-2, -1.5, -1, -0.5, -, 0.5, 1, 1.5, 2$ respectively and considered a noise level distributed as $N(0, 0.01)$. The posterior mean has then been calculated using the kernel function $k(x, x') \exp(-\frac{1}{2l^2}|x - x'|^2)$ for different values of l and λ, where λ features in the prediction equation $\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$.

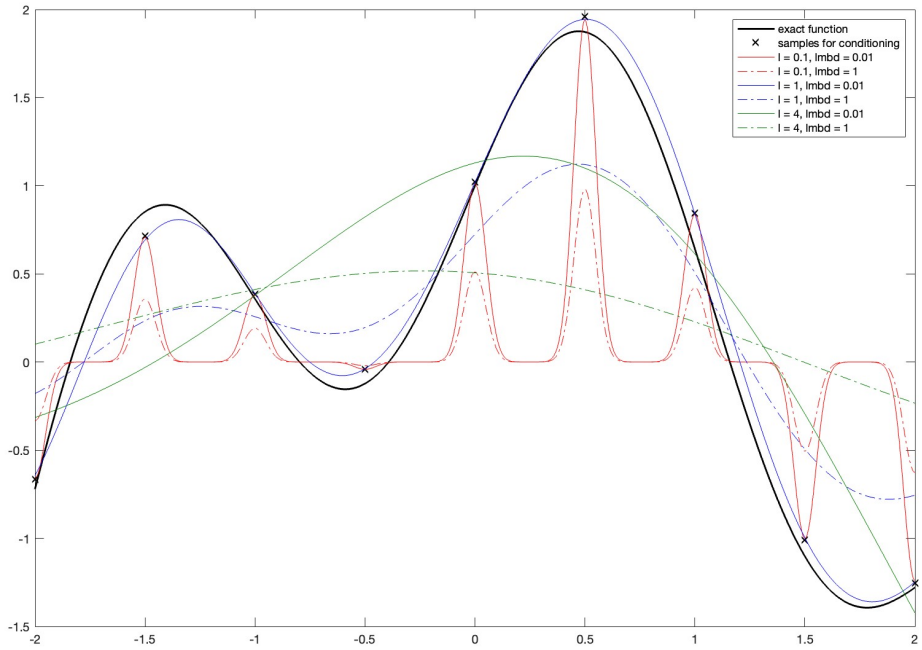

Figure 1: Posterior mean predictions - Part 1 (b) - 1000 grid points.

We can confirm our answers to Part 1 (a) by observing very similar patterns for the different parameters l and λ. Too small of an l leads to over-fitting with return to the mean, and too large of an l doesn't allow the function to vary enough to fit the data. Also, $\lambda = 1$ is clearly too large a value for this parameter in this case, as the reduction in sensitivity to noise is insignificant for the small amount of noise we have, but the bias introduced is very significant.

Question 2

Part 2 (a)

We analyze a similar setup to Part 1 (b) without the sample points at $-0.5, 0$, and 0.5. The parameters l and λ are set to 0.5 and 0.01 respectively. In addition to the posterior mean, the posterior variance has been computed using the formula $\sigma^2(\mathbf{x}) = \mathbf{k}(\mathbf{x},\mathbf{x}) - \mathbf{k}(\mathbf{x})^T(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x})$. In Figure 2, this is shown by showing two different confidence bounds, one for 4 standard deviations, and one for 0.5.

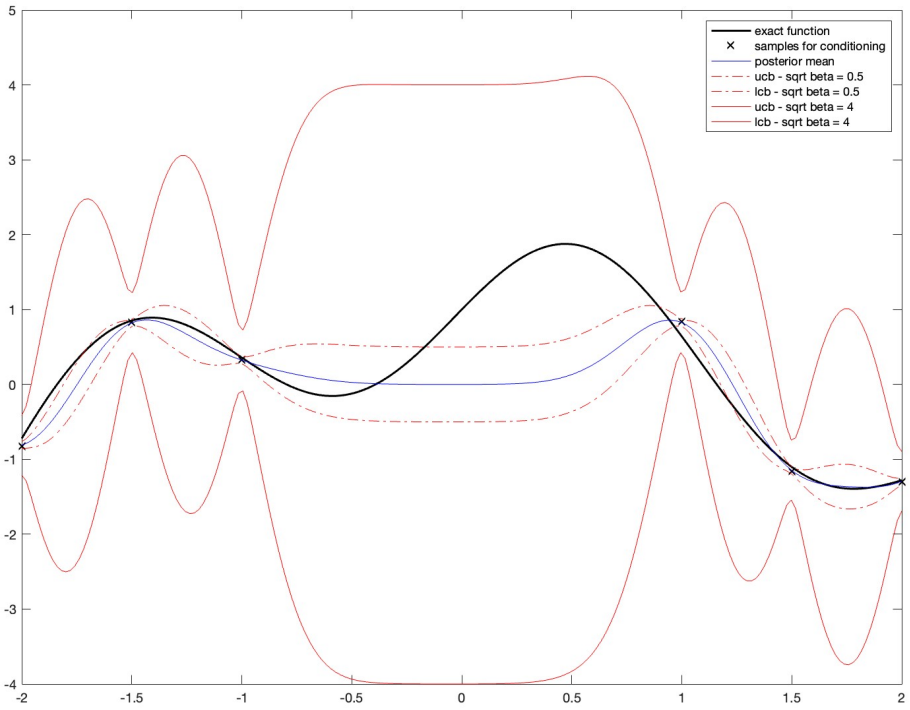

Figure 2: Posterior mean and variance - Part 2 (a) - 200 grid points.

We can see that the actual function (shown with a thick line) lies well within the confidence bound given by 4 standard deviations, but does not lie within the one given by 0.5.

If we were to assume that this prediction had arisen as part of the GP-UCB optimization algorithm, a point near $x = 0.6$ would be chosen next, as this is the maximizer of the upper confidence bound (assuming the 4 standard deviations case).

Part 2 (b)

The parameter $\sqrt{\beta}$ determines the number of standard deviations considered when creating the confidence bounds of the posterior. In implementing the GP-UCB algorithm, we need to consider the following.

(i) If β is too small, the predicted mean dominates the algorithm, as the upper confidence bound closely follows it. This means that exploration is not encouraged enough, and a local maximum might be pursued and exploited too soon.

(ii) On the other hand, if β is too large, then the variance term dominates the selection of the next point, and this leads to continued exploration of unknown regions for quite a long time until the variance has reduced everywhere, and a local maximum can be pursued. If we have a finite

number of iterations, and a large β, we therefore run the risk of never exploiting any of the good regions we found.

Part 2 (c)

Here, we have implemented the GP-UCB algorithm with $\sqrt{\beta}$ varying between $0.1, 2$, and 5. The first point to explore was chosen uniformly at random from the domain.

In Figures 3-6, the true curve is shown in black with a thicker line, the final posterior mean is shown in blue, and the final confidence bounds are shown in red. The data points sampled are marked with crosses and the darker the points are the younger they are, meaning that the points corresponding to iterations 1, 2, and 3 are bright, and the points corresponding to iterations 17, 18, and 19, are dark. The final point has been marked green.

We have two plots (Figures 3 and 4) for the case where $\sqrt{\beta} = 0.1$. As discussed in Part 2 (b), this choice of β is too small, and the algorithm performs subobtimally. Exploration is lacking, and as is the case for the second run (Figure 4), we may end up in a local maximum that isn't global.

Figure 5 demonstrates how the algorithm should behave for a well chosen parameter and kernel function. Here, $\sqrt{\beta} = 2$ turned out to be a good choice. Quite a few points have been explored before the best region has been exploited further.

Finally, Figure 6 showcases $\sqrt{\beta} = 25$ creating confidence intervals that are clearly too large. The variance of the posterior dominates the selection algorithm, and the result is continued exploration without exploitation.

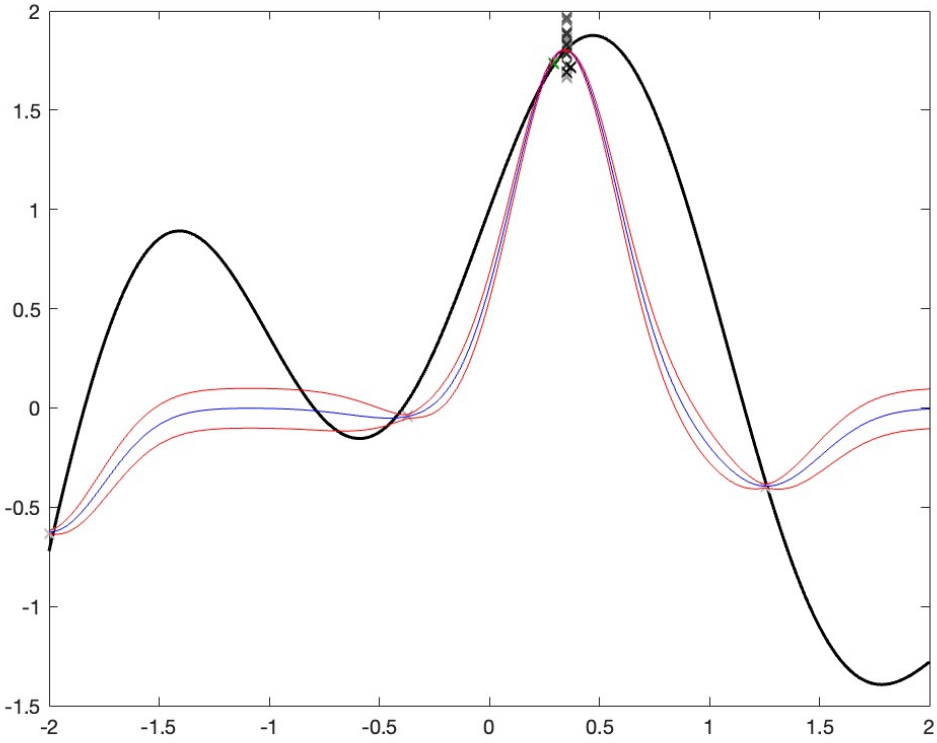

Figure 3: GP-UCB optimization with $\sqrt{\beta} = 0.1$, run 1.

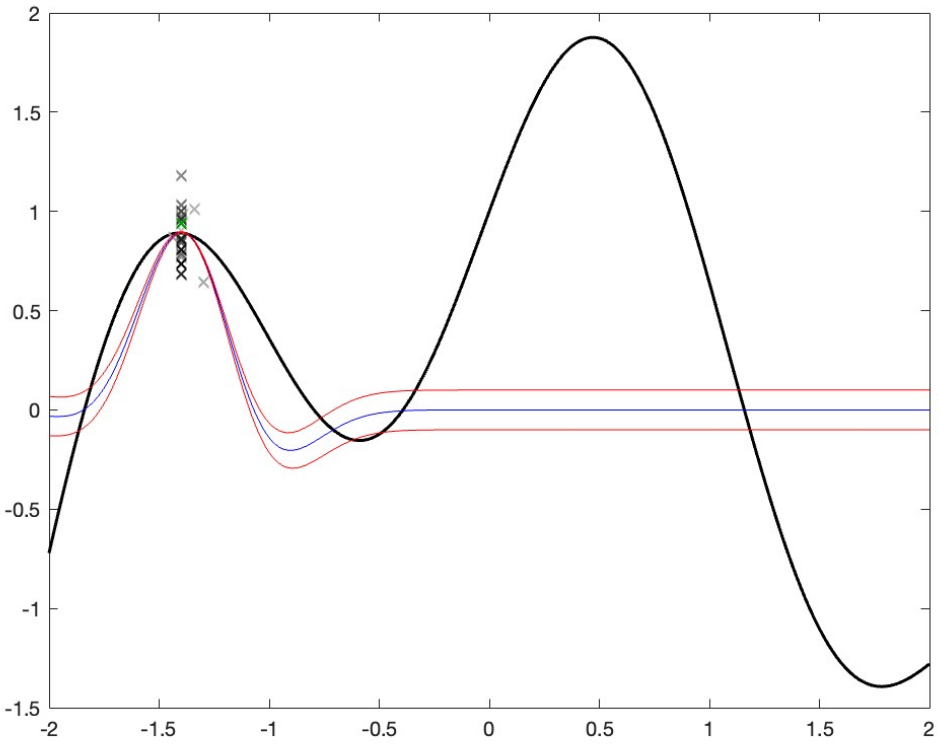

Figure 4: GP-UCB optimization with $\sqrt{\beta} = 0.1$, run 2.

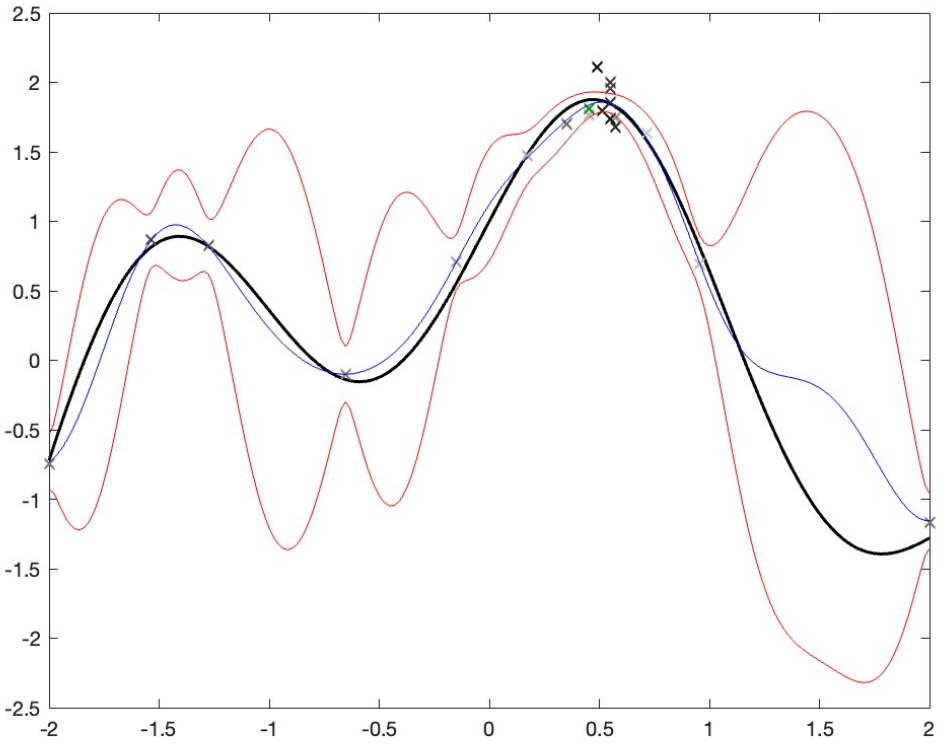

Figure 5: GP-UCB optimization with $\sqrt{\beta} = 2$.

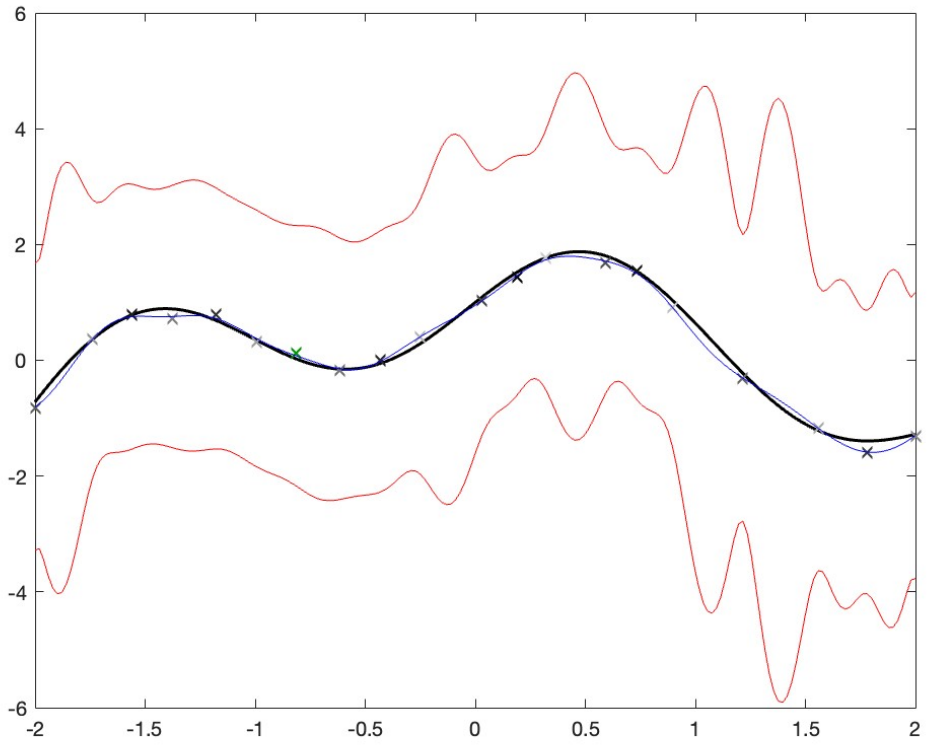

Figure 6: GP-UCB optimization with $\sqrt{\beta} = 25$.

Appendix

This appendix contains the MATLAB code that was used to generate the various figures of the corresponding sections.

Part 1 (b)

```
% parameters
p = 1000;       % grid resolution
n = 9;          % number of sample points
sigma = 0.1;    % standard deviation of the noise

% grid cells
x = linspace(-2,2,p);

% exact function
f = @(x) sin(3*x) - 0.5*x.^2 + 1;
plot(x, f(x), 'black', LineWidth=1.5); hold on;

% samples for conditioning
x_sample = linspace(-2,2,n)';
z_sample = sigma * randn(n,1);
y_sample = f(x_sample) + z_sample;
scatter(x_sample, y_sample, 'x', 'MarkerEdgeColor', '#000', LineWidth=1.5);


for l = [0.1 1 4]
  % kernel function based on l
  k = @(a,b) exp(-((b-a).^2)/(0.5*l^2));

  % kernel matrix
  kXX = zeros(n);

  for i = 1:n
    for j = 1:n
      kXX(i,j) = k(x_sample(i), x_sample(j));
    end
  end

  for lambda = [0.01 1]
    % posterior mean memory allocation
    mu_post = zeros(p,1);

    for t = 1:p
      % kernel vector
      kxX = k(x(t), x_sample);

      % posterior mean
      mu_post(t) = kxX'*((kXX + lambda*eye(n))\y_sample);
    end

    % plotting the posterior mean
    if l == 0.1
      col = 'red';
    elseif l == 1
      col = 'blue';
    else
      col = [0 0.5 0];
    end
```

```
    if lambda == 0.01
      linestyle = '-';
    else
      linestyle = '-.';
    end

    plot(x, mu_post, 'Color', col, 'LineStyle', linestyle);
  end
end

% legend
legend( ...
  'exact function', ...
  'samples for conditioning', ...
  'l = 0.1, lmbd = 0.01', ...
  'l = 0.1, lmbd = 1', ...
  'l = 1, lmbd = 0.01', ...
  'l = 1, lmbd = 1', ...
  'l = 4, lmbd = 0.01', ...
  'l = 4, lmbd = 1');
```

Part 2 (a)

```
% parameters
p = 200;         % grid resolution
n = 6;           % number of sample points
sigma = 0.1;     % standard deviation of the noise
l = 0.5;         % kernel parameter
lambda = 0.01;   % noise sensitivity parameter

% grid cells
x = linspace(-2,2,p);

% exact function
f = @(x) sin(3*x) - 0.5*x.^2 + 1;
plot(x, f(x), 'black', LineWidth=1.5); hold on;

% samples for conditioning
x_sample = [-2; -1.5; -1; 1; 1.5; 2];
z_sample = sigma * randn(n,1);
y_sample = f(x_sample) + z_sample;
scatter(x_sample, y_sample, 'x', 'MarkerEdgeColor', '#000', LineWidth=1.5);

% kernel function based on l
k = @(a,b) exp(-((b-a).^2)/(0.5*l^2));

% kernel matrix
kXX = zeros(n);

for i = 1:n
  for j = 1:n
    kXX(i,j) = k(x_sample(i), x_sample(j));
  end
end

% posterior mean and variance memory allocation
mu_post = zeros(p,1);
var_post = zeros(p,1);

for t = 1:p
```

```
  % kernel vector
  kxX = k(x(t), x_sample);

  % kernel inner product
  kxx = k(x(t), x(t));

  % posterior mean and variance
  mu_post(t) = kxX'*((kXX + lambda*eye(n))\y_sample);
  var_post(t) = kxx - kxX'*((kXX + lambda*eye(n))\kxX);
end

% plotting
plot(x, mu_post, 'Color', 'blue');

for sqrtbeta = [0.5 4]
  ucb_post = mu_post + sqrtbeta * sqrt(var_post);
  lcb_post = mu_post - sqrtbeta * sqrt(var_post);

  if sqrtbeta == 0.5
    linestyle = '-.';
  else
    linestyle = '-';
  end

  plot(x, ucb_post, 'Color', 'red', 'LineStyle', linestyle);
  plot(x, lcb_post, 'Color', 'red', 'LineStyle', linestyle);
end

% legend
legend( ...
  'exact function', ...
  'samples for conditioning', ...
  'posterior mean', ...
  'ucb - sqrt beta = 0.5', ...
  'lcb - sqrt beta = 0.5', ...
  'ucb - sqrt beta = 4', ...
  'lcb - sqrt beta = 4');
```

Part 2 (c)

```
% parameters
p = 200;          % grid resolution
sigma = 0.1;      % standard deviation of the noise
l = 0.5;          % kernel parameter
lambda = 0.01;    % noise sensitivity parameter
t_max = 20;       % max number of sampled points
sqrtbeta = 2;     % number of standard deviations added to form the ucb

% exact function
f = @(x) sin(3*x) - 0.5*x.^2 + 1;

% kernel function based on l
k = @(a,b) exp(-((b-a).^2)/(0.5*l^2));

% grid
x = linspace(-2,2,p);

% preallocating memory
x_sample = zeros(t_max,1);
z_sample = zeros(t_max,1);
```

```
y_sample = zeros(t_max,1);

% initial sample
x_sample(1) = 4*rand() - 2;
z_sample(1) = sigma * randn();
y_sample(1) = f(x_sample(1)) + z_sample(1);

% main iterations
for t = 1:t_max
  kXX = zeros(t);

  for i = 1:t
    for j = 1:t
      kXX(i,j) = k(x_sample(i), x_sample(j));
    end
  end

  mu_post = zeros(p,1);
  var_post = zeros(p,1);

  for i = 1:p
    kxX = k(x(i), x_sample(1:t));
    kxx = k(x(i), x(i));
    mu_post(i) = kxX'*((kXX + lambda*eye(t))\y_sample(1:t));
    var_post(i) = kxx - kxX'*((kXX + lambda*eye(t))\kxX);
  end

  ucb_post = mu_post + sqrtbeta * sqrt(var_post);
  lcb_post = mu_post - sqrtbeta * sqrt(var_post);

  if t < t_max
    [M, i_max] = max(ucb_post);

    x_sample(t+1) = x(i_max);
    z_sample(t+1) = sigma * randn();
    y_sample(t+1) = f(x_sample(t+1)) + z_sample(t+1);
  end
end


% plotting
plot(x, f(x), 'black', LineWidth=1.5); hold on;

for t = 1:t_max-1
  b = (t_max - t)/t_max;
  scatter(x_sample(t), y_sample(t), 'x', 'MarkerEdgeColor', [b b b]/1.2,
      LineWidth=1.5);
end

scatter(x_sample(t_max), y_sample(t_max), 'x', 'MarkerEdgeColor', [0 0.6
    0], LineWidth=1.5);

plot(x, mu_post, 'Color', 'blue');
plot(x, ucb_post, 'Color', 'red');
plot(x, lcb_post, 'Color', 'red');
```