Exercise Set 2 | PML Fall 2022

Theory

Kernel Properties and Random Processes

1.)

Show that the variation of the Wiener Process marginal $f_0 = 0, f_{i+1} = f_i + w_i, w \sim \mathcal{N}(0, \sqrt{x_{i+1} - x_i})$ is not a random process.

Let $f = (f_T, f_C)$ and from the definition of random processes be $p(f_T|T) = \int p(f_T, f_C|S) df_C = \int p(0, w_1, \dots, \sum_i w_i, f_C|S) df_C$

$$\Rightarrow W \sim \mathcal{N}\left(0, \begin{bmatrix} \sqrt{x_1 - x_0} & 0 & & \dots & 0 \\ 0 & \sqrt{x_2 - x_1} & 0 & \dots & 0 \\ 0 & \dots & \dots & & 0 \\ 0 & \dots & \dots & & \sqrt{x_C - x_{C-1}} \end{bmatrix}\right)$$

then by $f \sim \mathcal{N}(0, ADA^T) \Leftrightarrow f \sim \begin{bmatrix} f_1 \\ . \\ . \\ . \\ f_C \end{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ . & . & \dots & 0 \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ . \\ . \\ . \\ w_C \end{bmatrix}$. Then eventually we get

$$ADA^T = \begin{bmatrix} \sqrt{x_1 - x_0} & \sqrt{x_1 - x_0} & \dots & \sqrt{x_1 - x \\ \sqrt{x_1 - x_0} & \sqrt{x_2 - x_1} + \sqrt{x_2 - x_1} & \sqrt{x_2 - x_1} + \sqrt{x_2 - x_1} & \sqrt{x_2 - x_1} + \sqrt{\ } \\ . & . & \dots & \dots \\ \sqrt{x_1 - x_0} & \sqrt{x_2 - x_1} + \sqrt{x_2 - x_1} & \dots & \sum_i \sqrt{x_i - } \end{bmatrix}$$

From the resulting matrix we can see a dependency on previous iterations through the sum of square-roots. This is not a reducible (c.f. Wiener Process) sum. Thus the KCT property is not met and the resulting process not a random process.

2.)

Show that $ak_1 + bk_2$ is a kernel for $a, b > 0$.

First, we have $k = k_1 + k_2$ as a kernel by the additive property of kernels. Second, we have $k' = ak_1$ is a kernel for any real vector a. Thus, $k = ak_1 + ak_2$ is a kernel from the definitions in our script.

Furthermore: we show symmetry and the pos. def. property such that given any N-vector α, we have for any set $\{x\}_{i=1}^{N} : \alpha^T(k_1(x, x') + k_2(x, x'))\alpha = \alpha^T K_1 \alpha + \alpha^T K_2 \alpha \geq 0$, where K is defined as the matrix which is obtained by applying the kernel function to all input-pairs. For scaling: $K = aK_1 \Rightarrow \alpha^T K \alpha = a\alpha^T K_1 \alpha \geq 0$. Given that k_1, k_2 give rise to symmetric matrices this property is preserved for the scaling and additive operation.

3.)

Given two kernels with finite feature representations, show that $k_1 \times k_2$ is a kernel.

Let k1, k2 be two kernels with finite $|N|$ feature maps $\phi^{(1)}, \phi^{(2)}$, then by Mercer's theorem.:
$k_1(x, x') \cdot k_2(x, x') = \sum_i^{|N|} \phi_i^{(1)}(x)\phi_i^{(1)}(x') \cdot \sum_j^{|N|} \phi_j^{(2)}(x)\phi_j^{(2)}(x')$
$= \sum_i \sum_j [\phi_i^{(1)}(x)\phi_j^{(2)}(x)][\phi_i^{(1)}(x')\phi_j^{(2)}(x')]$. Now let $\psi(\cdot) = \phi^{(1)}(\cdot)\phi^{(2)}(\cdot)$, then
$k_1(x, x') \cdot k_2(x, x') = \sum_{i,j} \psi_{i,j}(x)\psi_{i,j}(x') = \psi(x)\psi(x')$, which is a kernel.

4.)

For a kernel $k(x, x') = (1 + x^T x')$ compute a feature vector ϕ such that $k(x, x') = \phi(x)\phi(x')$.

Let $\phi(x, x') = (1 + x^T x')$
$k(x, x') = (1 + x^T x')^2 = (1 + x^T x')(1 + x^T x') = \phi(x, x')\phi(x, x') = \hat{\phi}(x)\hat{\phi}(x')$ by using the previous results. For n dimensions we get the feature map:
$\hat{\phi}(x) := (1, x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_{n-1}x_{n-2}, \sqrt{2}x_2 x_1, \ldots, \sqrt{2}x_1)^T$

Or simpler: $\phi(x) = [1, x_1x_1, x_1x_2, \ldots, x_2x_2, \ldots, x_nx_n, \sqrt{2}x_1, \ldots, \sqrt{2}x_n]$.

Programming

Gaussian Processes

The Observatory CO2 data-set.

Track CO_2 concentration over time.

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8
1	1958	3	1958.2	315.7	314.43	-1	-9.99	-0.99
2	1958	4	1958.29	317.45	315.16	-1	-9.99	-0.99
3	1958	5	1958.37	317.51	314.71	-1	-9.99	-0.99
4	1958	6	1958.45	317.24	315.14	-1	-9.99	-0.99
5	1958	7	1958.54	315.86	315.18	-1	-9.99	-0.99
6	1958	8	1958.62	314.93	316.18	-1	-9.99	-0.99
7	1958	9	1958.71	313.2	316.08	-1	-9.99	-0.99
8	1958	10	1958.79	312.43	315.41	-1	-9.99	-0.99
9	1958	11	1958.87	313.33	315.2	-1	-9.99	-0.99
10	1958	12	1958.96	314.67	315.43	-1	-9.99	-0.99
more								
776	2022	10	2022.79	415.78	419.12	30	0.27	0.09

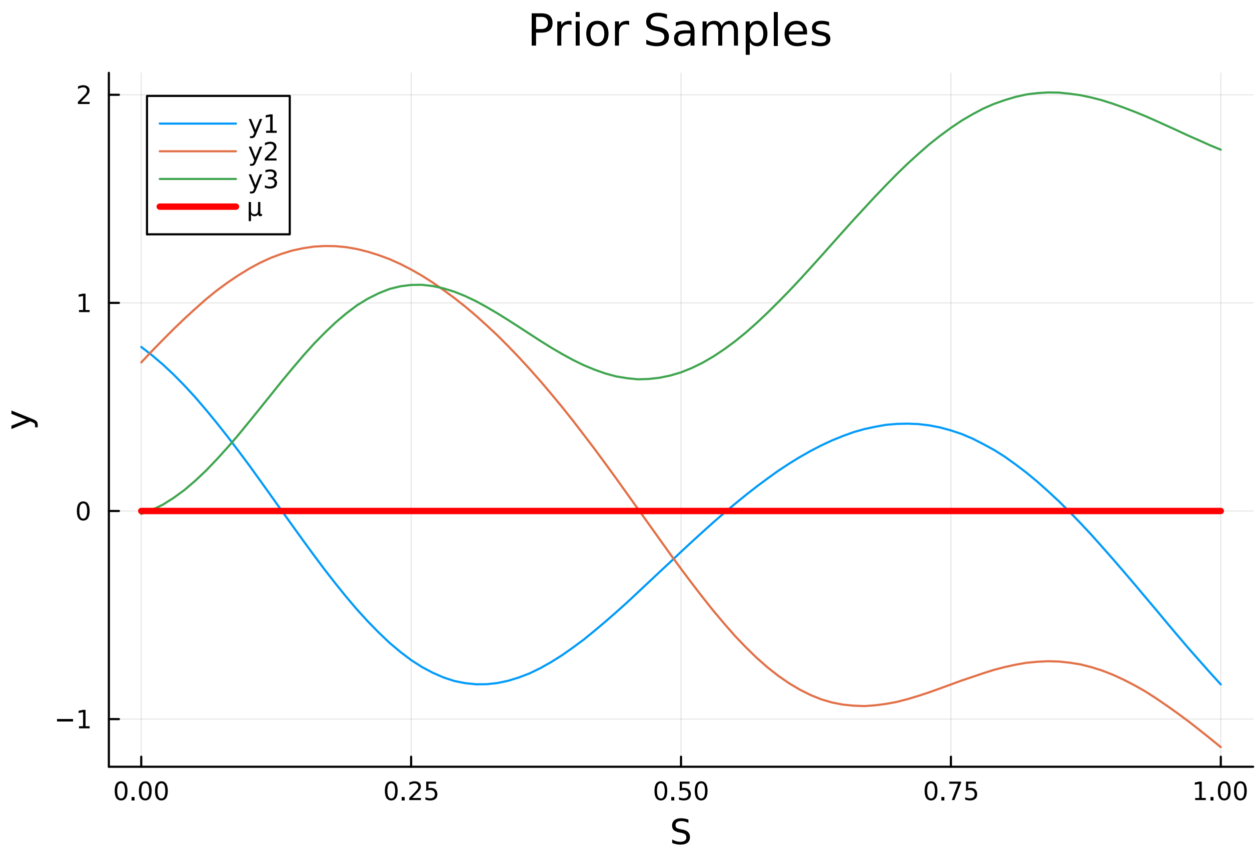
```julia
df
```

([0.2027, 0.2877, 0.3699, 0.4548, 0.537, 0.6219, 0.7068, 0.789, 0.874, more ,10.12

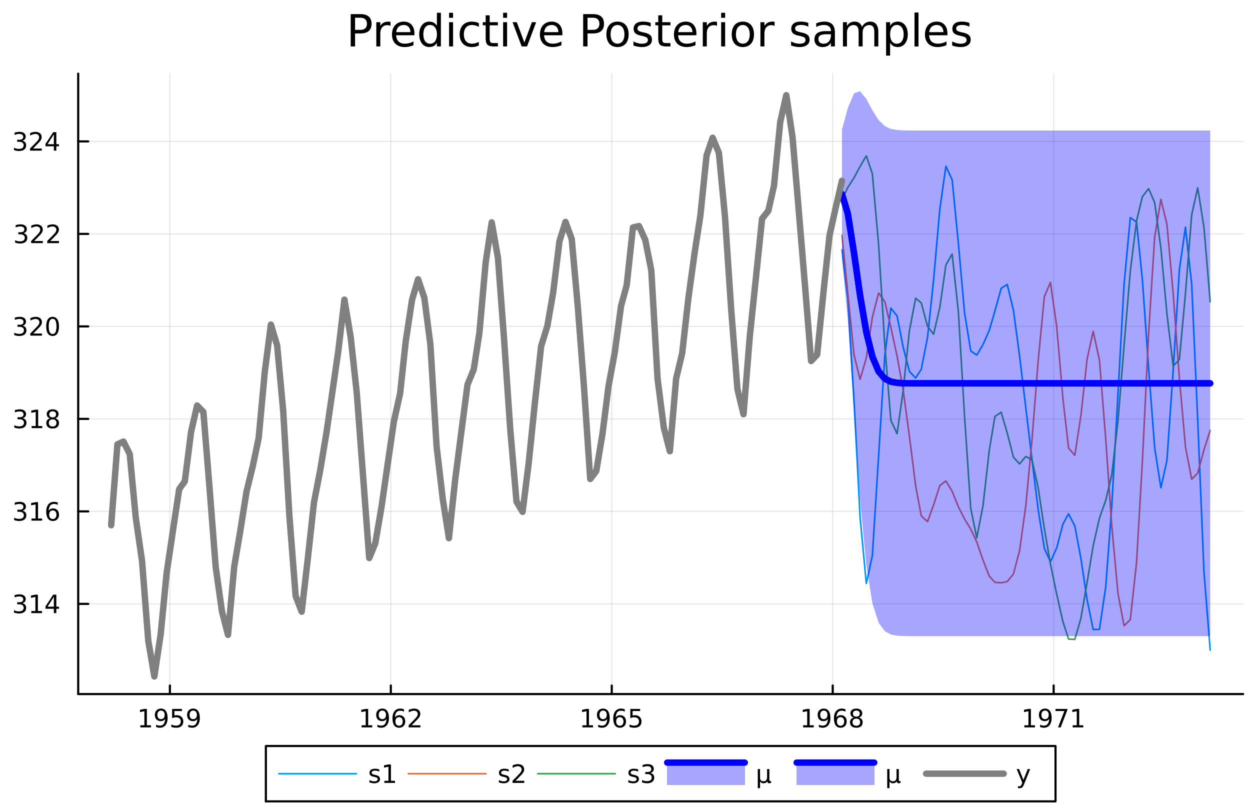
```julia
begin
    # load and prep data
    using PlutoUI
    using DataFrames
    using CSV
    using Statistics
    df = DataFrame(CSV.File("./data/co2_mm_mlo.csv", header=0))
    YEAR = 1958
    X = df[1:120, :Column3].-YEAR
    X_new = df[120:180, :Column3].-YEAR
    _y = df[1:120, :Column4]
    _y_new = df[120:180, :Column4]
    y_μ = mean(_y)
    y_std = std(_y)
    y = (_y .- y_μ)./y_std
    # check the properties after standardization
    @assert isapprox(mean(y), 0, atol=10e-12) && std(y) == 1.
    X, y
end
```

A)

Compute prior samples for GP with gaussian kernel, for S ∈ {0, 1} with γ=10. .




```
begin
    using Distributions
    using LinearAlgebra
    using Base.Iterators
    using Random
    using Plots
    Random.seed!(42)
    d = size(X)
    ϵ = 1.e-7 # counter POS.DEF Error Cholesky factorization bug in MvNormal
    S = range(start=0, step=0.01, stop=1)
    γ = 10.
    σ_d = 0.1
    k_gauss(r, γ=γ) = exp(-γ * (r[1]-r[2])^2) # Gauss Kernel
    k_S = k_gauss.(product(S, S), γ)
    # A) Sample from prior s ∈ (0, 0.01, ..., 1)
    GP_prior = MvNormal(zeros(size(S)), Hermitian(k_S)+I*ϵ)
    prior_samples = rand(GP_prior, 3)
    plot(S, [prior_samples[:,1], prior_samples[:,2], prior_samples[:,3]],
        title="Prior Samples", xlabel="S", ylabel="y")
    plot!(S, zeros(size(S)), label="μ", lw=3, c="red")
end
```




```
begin
    # B) Compute Posterior
    function gp_posterior(k, X, X_new, y, σ_d)
        k_XX = k.(product(X, X))
        k_xX = k.(product(X_new, X))
        #@show view(k_xX, 1:10, 1:10)
        k_xx = k.(product(X_new, X_new))
        μ_star = k_xX * inv(k_XX+σ_d*I)*y
        Σ_star = k_xx - k_xX*inv(k_XX+σ_d*I)*k_xX'
        return μ_star, Σ_star
    end
    μ_star, Σ_star = gp_posterior(k_gauss, X, X_new, y, σ_d)
    var_star = diag(Σ_star)*y_std^2 # unscale predictive variance
    ci_95 = 1.95*sqrt.(var_star)
    GP_posterior = MvNormal(μ_star, Hermitian(Σ_star)+ϵ*I)
    posterior_samples = rand(GP_posterior, 3)
    unscale(y, y_std=y_std, y_μ=y_μ) = y.*y_std .+ y_μ
    # undo scaling
    y_pred = unscale(μ_star)
    plot(X_new .+ YEAR, [unscale(posterior_samples[:,1]),
    unscale(posterior_samples[:,2]), unscale(posterior_samples[:, 3])],
    title="Predictive Posterior samples", label=["s1" "s2" "s3"], lw=0.75)
    plot!(X_new .+ YEAR, [y_pred y_pred], fillrange=[y_pred.-ci_95
    y_pred.+ci_95], fillalpha=0.35, label="μ", lw=3, c="blue")
    plot!(X .+ YEAR, _y, label="y", lw=3, c="grey")
    plot!(legend=:outerbottom, legendcolumns=6)
end
```

```
nll (generic function with 1 method)
```

```julia
begin
    # C) Implement Special Kernel
    function k_spec(r::Tuple, η::Vector)
        x, y = r
        a, b = η
        return (1+x'*y)^2 + a*sin(2π*x+b)*sin(2π*y+b)
    end
    function nll(y, k, S, η, σ_d)
        n = length(S)
        X = product(S, S)
        k_close(x) = k(x, η)
        k_η_S = k_close.(X) # NOTE: this should be tested for p.s.d. property
        val = 0.
        try
            val = -0.5*y'*inv(k_η_S+σ_d*I)*y - 0.5*logdet(k_η_S+σ_d*I) -
            (n/2)*log(2π) # Eq.2.30 in GPML
        catch DomainError
            val = 1000 # catch inf on logdet computation or neg OOB and assign
            high value
        end
        return val
    end
end
```

Implement Basic Grid Search on GP-NLL

```
-27402.423553825938
```

```julia
begin
    # basic Grid Search constrained
    σ_d_range = 0.001:0.01:0.5 # assumption: max-noise is 0.5
    a_range = 0.01:0.1:5
    b_range = 0.01:0.1:10
    search_space = product(a_range, b_range, σ_d_range)
    grid_task = η -> nll(y, k_spec, X, [η[1], η[2]], η[3])
    evaluated_grid = map(grid_task, search_space)
    evaluated_grid[argmin(evaluated_grid)]
end
```

```
(4.91, 4.31, 0.001)
```

```julia
collect(search_space)[argmin(evaluated_grid)]
```

```
-27406.230063702696
```

```julia
nll(y, k_spec, X, [4.9, 4.3], 0.001)
```

Optimization Run on GP-NLL

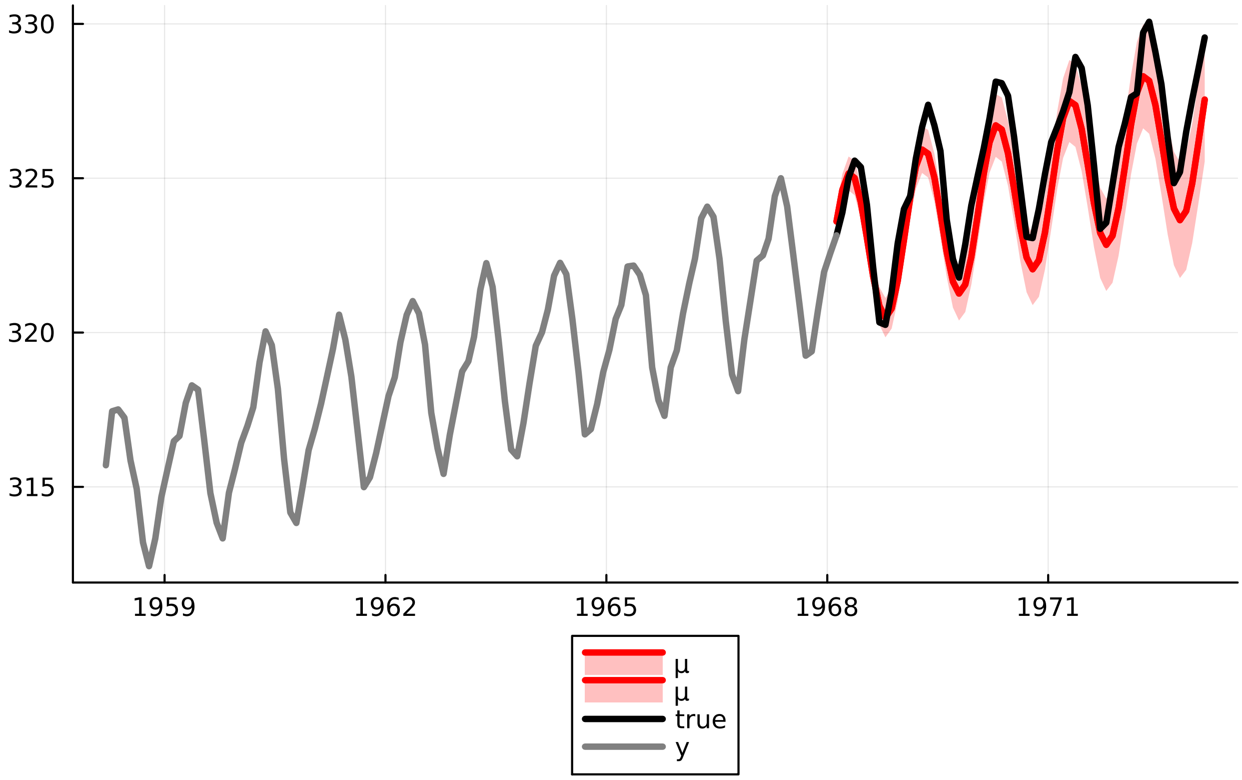
unconstrained parameter optimization, iterating over σ. If σ unconstrained logistic results in noise=1.0 for minimizing NLL value.

```
(-27405.9, [[1.94016, 1.15582], 0.001])
```

```
begin
    using Optim
    using LogExpFunctions    # used to constrain noise in 0<x<1 range
    #opt_closure = η -> nll(y, k_spec, X, [η[1], η[2]], logistic(η[3]))
    _res = 0.
    _minimizer = Nothing
    for _σ ∈ 0.001:0.01:0.5
        global _res # refer to out of loop scope
        global _minimizer
        opt_closure = η -> nll(y, k_spec, X, [max(η[1], 0), max(η[2], 0)], _σ) #
        clipping parameters to positive range
        temp_res = optimize(opt_closure, [2., 1.], LBFGS())
        _minimizer = temp_res.minimum < _res ? [temp_res.minimizer, _σ] :
        _minimizer
        _res = temp_res.minimum < _res ? temp_res.minimum : _res
    end
    _res, _minimizer
end
```

Plot Posterior

after optimization - take grid search parameters.




```
begin
    k(r) = k_spec(r, [2, 2.8])
    # compute posterior predictive with optimized kernel parameters:
    μ_pred, Σ_pred = gp_posterior(k, X, X_new, y, 0.1)
    #GP_pred_posterior = MvNormal(μ_pred, Hermitian(Σ_pred)+ϵ*I)
    # compute variance and CI:95%
    var_pred = diag(Σ_pred)*y_std^2
    ci_pred = 1.95*sqrt.(var_pred)
    # undo scaling
    y_final = unscale(μ_pred)
    # plot posterior
    plot(X_new .+ YEAR, [y_final y_final], fillrange=[y_final.-ci_pred
    y_final.+ci_pred], fillalpha=0.25, label="μ", lw=3, c="red")
    plot!(X_new .+ YEAR, _y_new, label="true", lw=3, c="black")
    plot!(X .+ YEAR, _y, label="y", lw=3, c="grey")
    plot!(legend=:outerbottom)
end
```