Problem Sheet 3

```julia
begin
    using Pkg
    Pkg.activate(".")
    Pkg.add(["Distributions", "Plots", "PlutoUI"])
    using Distributions
    using LinearAlgebra
    using Plots
    using PlutoUI
    default(linewidth = 3.0, legendfontsize= 15.0)
end
```

Table of Contents

1. Bayes inference for the variance of a Gaussian

Use a Bayesian approach to estimate the inverse variance λ of a univariate Gaussian distribution

$$p(x|\lambda) = \sqrt{\frac{\lambda}{2\pi}}\ \exp\left[-\frac{\lambda x^2}{2}\right].$$

Here we have assumed for simplicity that the data has zero mean $\mu = 0$. To apply Bayesian inference we specify a **Gamma** prior distribution for λ,

$$p(\lambda) = \frac{\lambda^{\alpha-1} \exp\left[-\lambda/\beta\right]}{\Gamma(\alpha)\beta^{\alpha}}$$

where the positive numbers α and β, the \emph{hyperparameters} of the model are assumed to be known and $\Gamma(\alpha)$ is Euler's **gamma** function. We then observe a dataset $D = (x_1, x_2, \ldots, x_N)$ comprising N independent random samples from $p(x|\lambda)$.

(a) [MATH] Show that the posterior probability $p(\lambda|D)$ of the inverse variance is also a Gamma distribution with parameters

$$\alpha_p = \alpha + \frac{N}{2}, \qquad \frac{1}{\beta_p} = \frac{1}{\beta} + \frac{1}{2}\sum_{i=1}^{N} x_i^2.$$

Write your answer here or on paper

(b) [MATH] Compute the mean of the posterior distribution of λ. Compare the result with the result from the maximum-likelihood estimation, $\lambda_{\mathrm{ML}} = 1/\sigma^2_{\mathrm{ML}}$ and explain what happens if $N \to \infty$.

Write your answer here or on paper

(c) [MATH] Show that the variance of the posterior distribution

$$V[\lambda_{\mathrm{post}}] = \langle \lambda^2 \rangle - \langle \lambda \rangle^2$$

shrinks to zero as $N \to \infty$. Here we have used the notation $\langle \cdot \rangle$ for posterior expectations.

Write your answer here or on paper

(d) [MATH] Show that the predictive distribution is

$$p(x|D) = \frac{1}{\sqrt{2\pi}} \frac{\Gamma(\alpha_p + 1/2)}{\Gamma(\alpha_p)} \sqrt{\beta_p} \left(1 + \frac{x^2 \beta_p}{2}\right)^{-\alpha_p - 1/2}$$

where α_p and β_p were defined above. Note, this is not a Gaussian!

Write your answer here or on paper

(e) [CODE] Program a function generating samples from a known value λ and compute both the posterior distribution and ML estimator by adding progressively new samples.

```
gamma_params = (α=2.0, β=0.5);
```

```
true_λ = 1.127053974465347
```

```
true_λ = rand(Gamma(gamma_params...)) # We sample a random value λ
```

```
syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. top-level scope @ none:1
```

```
generate_x(λ) = ## !! FILL IN !! Generate a random value x from a give λ
```

```
syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. top-level scope @ none:1
```

```
αp(x, α) = ## !! FILL IN !! Return the posterior alpha paramter
```

```
syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. top-level scope @ none:1
```

```
βp(x, β) = ## !! FILL IN !! Return the posterior beta paramter
```

```
posterior_λ(x, α, β) = Gamma(αp(x, α), βp(x, β)); # Posterior distribution
```

```
syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. top-level scope @ none:1
```

```
λ_ML(x) = ## !! FILL IN !! Return the maximum likelihood estimator
```

```
0.0:0.016722408026755852:5.0
```

```
begin  # Plotting values
    N = 10000
    bins = range(-5, 5, length = 50)
    range_λ = range(0, 5.0, length = 300)
end
```

UndefVarError: generate_x not defined

1. top-level scope @ *Local: 5*

```
begin
    xs = Float64[]
    anim = Animation()
    for i in 1:N
        push!(xs, generate_x(true_λ))
        if i % 10^floor(Int64, log10(i)) == 0
            p_x = histogram(xs, bins = bins, lw = 0.0, label = "", title = "N = $i")
            p_λ = plot(range_λ, x -> pdf(posterior_λ(xs, gamma_params...), x), label =
"p(λ|D)")
            vline!([λ_ML(xs)], label = "λ ML")
            vline!([true_λ], label = "true λ")
            plot(p_x, p_λ, size = (800,400))
            frame(anim)
        end
    end
end
```

ArgumentError: Cannot build empty animations

1. var"#buildanimation#214"(::Int64, ::Int64, ::Bool, ::Bool, ::Bool, ::typeof(Plots.buildanimation), ::Plots.Animation, ::String, ::Bool) @ *animation.jl:79*
2. #gif#210 @ *animation.jl:64* [inlined]
3. top-level scope @ *Local: 1* [inlined]

```
gif(anim, fps = 6)
```

UndefVarError: αp not defined

1. posterior_λ(::Vector{Float64}, ::Float64, ::Float64) @ *Other: 1*
2. top-level scope @ *Local: 1*

```
posterior_λ(xs, gamma_params...)
```

2. Hyperparameter estimation for a generalised linear model

Consider a model for a set of data $D = (y_1, \dots, y_n)$ defined by

$$p(D|\mathbf{w}, \beta) = \left(\frac{\beta}{2\pi}\right)^{N/2} \exp\left[-\sum_{i=1}^{N}\frac{\beta}{2}\left(y_i - \sum_{j=1}^{K} w_j \Phi_j(x_i)\right)^2\right]$$

with a fixed set $\{\Phi_1(x), \ldots, \Phi_k(x)\}$ of K basis functions. The prior distribution on the weights is given by

$$p(\mathbf{w}|\alpha) = \left(\frac{\alpha}{2\pi}\right)^{K/2} \exp\left[-\frac{\alpha}{2}\sum_{j=1}^{K} w_j^2\right].$$

This **generalised linear model** assumes that the observations are generated from a weighted linear combination of the basis functions with additive Gaussian noise.

- ### (a) [MATH] The posterior distribution $p(\mathbf{w}|D)$ of the vector of weights is a Gaussian. Compute the posterior mean vector $E[\mathbf{w}]$ and the posterior covariance in terms of the matrix $\mathbf{X}$ where $X_{lk} = \Phi_k(x_l)$.

Write your answer here or on paper

- ### (b) [MATH] Derive an EM algorithm for optimising the hyperparameter β by maximising the log-evidence

$$p(D|\alpha, \beta) = \int p(D|\mathbf{w}, \beta) p(\mathbf{w}|\alpha)\, d\mathbf{w}$$

Tip

Hint: Treat the weights $\mathbf{w}$ as a set of latent variables similar to the procedure for α given in the lecture. Express your result in terms of the posterior mean and variance.

Write your answer here or on paper

(c) [CODE] Implement the posterior solution of a generalised linear model given an arbitrary base function $\Phi(X) = \{\Phi_1(X), \ldots, \Phi_K(X)\}$

```
Φ (generic function with 2 methods)
```

```
begin
    # Φ(x::Real) = [x] # Linear Case
    # Φ(x::Real) = [1, x, sin(x), cos(x)]
    Φ(x::Real) = [one(x), x, x^2, x^3]#evalpoly(x, ones(4))
    Φ(x::Vector) = mapreduce(Φ, hcat, x) # Create a matrix out of a vector
end
```

syntax: invalid identifier name "..."

1. **top-level scope** @ *none:1*

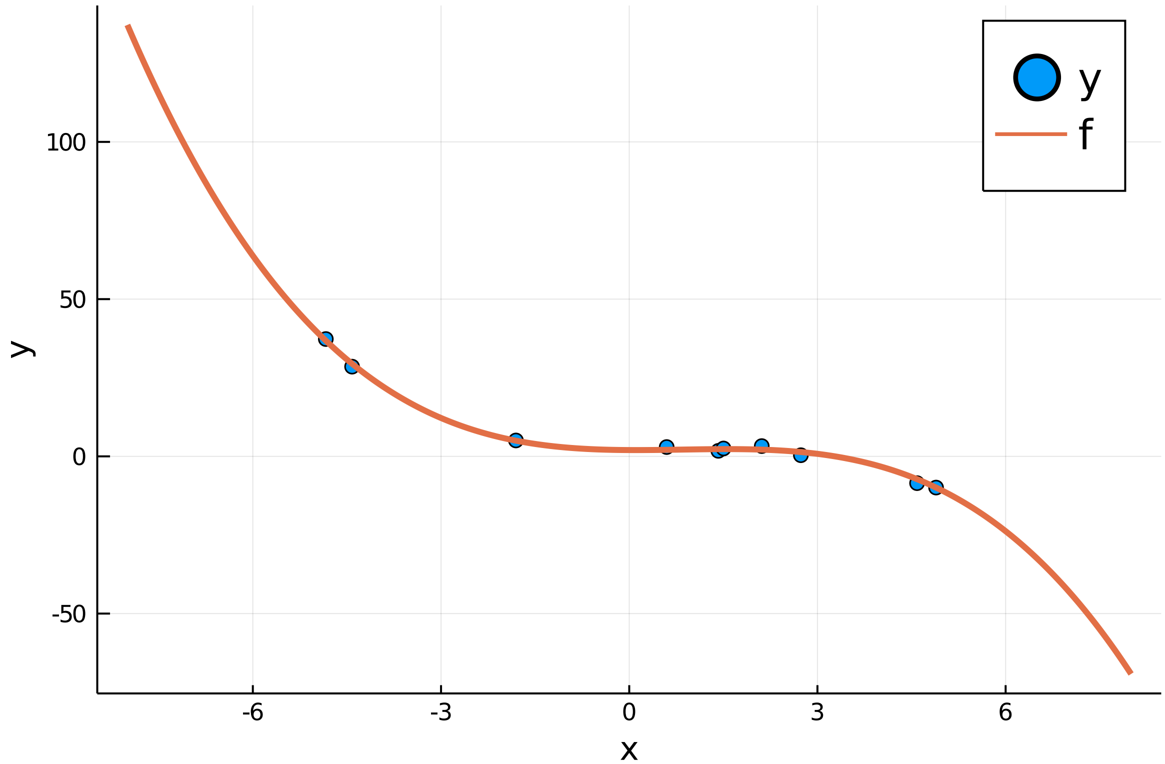
```
function posterior_params(val_Φ, y, α, β)
    ## !! FILL IN !!
    ## This function should return the mean and the covariance of the posterior
    μ = ...
    Σ = ...
    return μ, Σ
end
```

(d) [CODE] Given some random data, implement the EM algorithm to optimize α and β

data points : 10

β = 1.0

```
begin # Create some data
    X = rand(Uniform(-5, 5), Nx); # Sample X uniformly
    offset = 3
    X_test = collect(range(-5 - offset, 5 + offset, length = 500))
    w_true = [2.0, -0.1, 0.5, -0.2]
    f = evalpoly.(X, Ref(w_true)); # return w[0] + w[1] X + w[2] X^2 + ....
    y = f + randn(Nx) / sqrt(β_true); # Add some noise
end;
```



syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. **top-level scope** @ *none:1*

```
update_α(μ, Σ, K) = ## !! FILL IN !! Return the optimal alpha parameter
```

syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

1. **top-level scope** @ *none:1*

```
update_β(val_Φ, y, μ, Σ) = ## !! FILL IN !! Return the optimal beta parameter
```

expectation_step (generic function with 1 method)

```
function expectation_step(val_Φ, y, μ, Σ, K, α, β)
    ## !! FILL IN !! Return the value of L
end
```

UndefVarError: posterior_params not defined

1. **top-level scope** @ *Local: 8* [inlined]
2. **top-level scope** @ *none:0*

```
begin
    α = 1.0 # Initial parameters
    β = 1.0
    val_Φ = Φ(X) # Feature map
    T = 10 # Number of steps
    K = size(val_Φ, 1) # Feature map dimension
    for i in 1:T
        μ, Σ = posterior_params(val_Φ, y, α, β)
        println("i = $i, pre L = $(expectation_step(val_Φ, y, μ, Σ, K, α, β)), α = $α, β = $β")
        α = update_α(μ, Σ, K)
        β = update_β(val_Φ, y, μ, Σ)
```

```
        println("i = $i, post L = $(expectation_step(val_Φ, y, μ, Σ, K, α, β)), α = $α, β = $β")
    end
end
```

UndefVarError: posterior_params not defined

1. top-level scope @ *Local: 2*

```
begin
    μ, Σ = posterior_params(val_Φ, y, α, β)
    plot(
        bar([μ, w_true], label = ["E[w]" "True w"], alpha = 0.5, lw = 0.0),
        heatmap(Σ, title = "Cov(w)", yflip =true),
    )
end
```

UndefVarError: μ not defined

1. top-level scope @ *Local: 3*

```
begin
    scatter(X, y, lab= "y", xlabel="x", ylabel="y")
    plot!(X_test, Φ(X_test)' * μ; lab="Prediction", linewidth=5.0)
    plot!(X_test, evalpoly.(X_test, Ref(w_true)); linestyle=:dash, color=:black, label="f")
end
```

We can also sample from the posterior to visualize all the different possibilities

10

UndefVarError: Σ not defined

1. top-level scope @ *Local: 5* [inlined]
2. top-level scope @ *none:0*

```
begin
    p = scatter(X, y, lab= "y", xlabel="x", ylabel="y")
    S = 100
    for i in 1:S
        w = rand(MvNormal(μ, Symmetric(Σ)))
        plot!(X_test, Φ(X_test)' * w; lab="", color=:black, alpha=0.01)
    end
    p
end
```

```
(α = 1.0, β = 1.0)
```

```
(;α, β)
```

```
1.001
```

```
β_true
```

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