Problem Sheet 5

```
begin
    using Pkg; Pkg.activate("."); Pkg.add(["Distributions", "Plots", "PyPlot",
"PlutoUI"])
    using Distributions, LinearAlgebra
    using PlutoUI
    using Plots, StatsPlots; pyplot();
    default(lw = 3.0, legendfontsize= 15.0)
end
```

```
UndefVarError: TableOfContents not defined

1. top-level scope @ Local: 1
```

```
TableOfContents()
```

1. Variational inference

Assume we have n observations $D = (x_1, \ldots, x_n)$ generated independently from a Gaussian density $\mathcal{N}(x|\mu, 1/\tau)$, i.e.

$$p(D|\mu,\tau) = \left(\frac{\tau}{2\pi}\right)^{n/2} \exp\left[-\frac{\tau}{2}\sum_{i=1}^{n}(x_i-\mu)^2\right]$$

We also assume prior densities $p(\mu|\tau) = \mathcal{N}(\mu|\mu_0, (\lambda_0\tau)^{-1})$ and $p(\tau) = \mathrm{Gamma}(\tau|a_0, b_0)$. λ_0 and μ_0 as well as a_0, b_0 are given hyper parameters.

Our goal is to approximate the posterior density $p(\mu, \tau|D)$ by a **factorising density** $q(\mu,\tau) = q_1(\mu)q_2(\tau)$ which minimises the variational free energy

$$F[q] = \int q(\mu,\tau) \ln \frac{q(\mu,\tau)}{p(\mu,\tau,D)}\, d\mu\, d\tau$$

(a) [MATH] Show that the optimal $q_1(\mu)$ is a *Gaussian density* and give expressions for the mean and variance in terms of expectations with respect to q_2.

(b) [MATH] Show that the optimal $q_2(\tau)$ is a *Gamma density* and give expressions for the parameters in terms of expectations with respect to q_1.

Tip

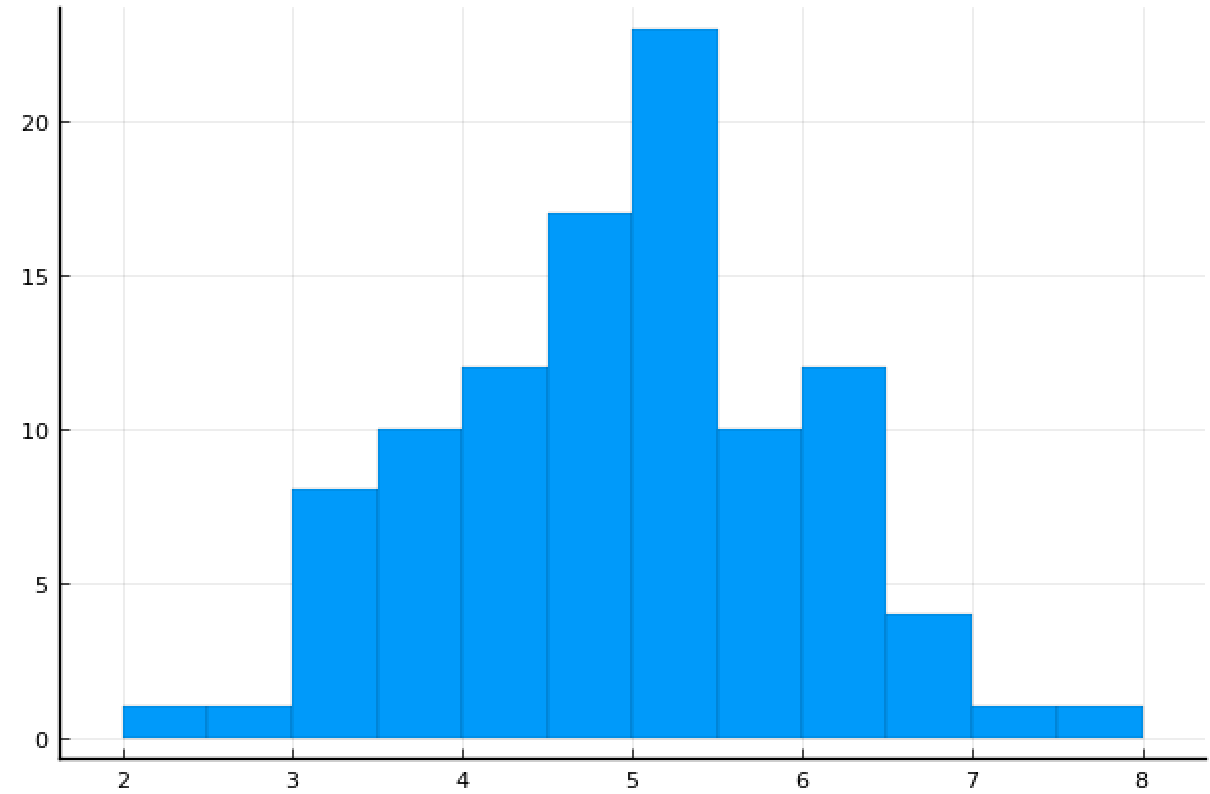
You can use the following results which follow from the derivations given in the lecture

$$q_1(\mu) \propto \exp\left[E_\tau[\ln p(\mu, \tau, D)]\right]$$
$$q_2(\tau) \propto \exp\left[E_\mu[\ln p(\mu, \tau, D)]\right]$$

Fill in your answer here or on paper

c) [CODE] From the generated dataset implement a coordinate ascent scheme, updating variational parameters of τ and μ in an alternated way.

```
begin
    # We create some data
    N = 100
    μ₀ = 5.0
    λ₀ = 2.0
    a₀ = 1.0
    b₀ = 2.0
    τ = rand(Gamma(a₀, b₀))
    μ = rand(Normal(μ₀, inv(sqrt(τ * λ₀))))
    x = rand(Normal(μ, inv(sqrt(τ))), N)
end;
```




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

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

```julia
expec_μ(x, λ₀, μ₀, n) = ## FILL IN ## Return E_q[mu]
```

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

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

```julia
var_μ(e_τ, λ₀, n) =  ## FILL IN ## Return Var_q[mu]
```

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

```julia
expec_τ(a, b) = a / b
```

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

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

```julia
aₙ(a₀, n) = ## FILL IN ## Return first parameter of q(tau)
```

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

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

```julia
bₙ(b₀, x, e_mu, var_mu, λ₀, μ₀) = ## FILL IN ## Return second parameter of q(tau)
```

coordinate_ascent (generic function with 5 methods)

```
function coordinate_ascent(T, a = 1.0, b = 1.0, μ = 1.0, σ² = 1.0)
    as = vcat(a, zeros(T))
    bs = vcat(b, zeros(T))
    μs = vcat(μ, zeros(T))
    σs = vcat(σ², zeros(T))
    for i in 2:T+1
        a = aₙ(a₀, N); as[i] = a;
        b = bₙ(b₀, x, μ, σ², λ₀, μ₀); bs[i] = b
        e_τ = expec_τ(a, b)
        μ = expec_μ(x, λ₀, μ₀, N); μs[i] = μ
        σ² =var_μ(e_τ, λ₀, N); σs[i] = σ²
    end
    return as, bs, μs, σs
end
```

UndefVarError: aₙ not defined

1. **coordinate_ascent**(::Int64, ::Float64, ::Float64, ::Float64, ::Float64) @ ***Other: 7***
2. **coordinate_ascent**(::Int64) @ ***Other: 2***
3. **top-level scope** @ ***Local: 3***

```
begin
    T = 20
    as, bs, mus, σs = coordinate_ascent(T)
    plt1 = plot([as bs], label = ["a" "b"])
    plt2 = plot([mus σs], label = ["μ" "σ²"])
    plot(plt1, plt2)
end
```

UndefVarError: aₙ not defined

1. **coordinate_ascent**(::Int64, ::Int64, ::Int64, ::Int64, ::Float64) @ ***Other: 7***
2. **top-level scope** @ ***Local: 3***

```
begin
    T_hard = 20
    as_hard, bs_hard, mus_hard, σs_hard = coordinate_ascent(T_hard, 1000, 2000, 300, 
1e3)
    p1 = plot([as_hard bs_hard], label = ["a" "b"], yaxis = :log)
    p2 = plot([mus_hard σs_hard], label = ["μ" "σ²"], yaxis = :log)
    plot(p1, p2)
end
```