Problem 2(a)

In [1]:
```
using ProgressMeter
using LinearAlgebra
using MAT
using Plots; pyplot();
```

In [2]:
```
vars = matread("spam_data.mat")

X_train = log.(vars["Xtrain"] .+ 0.1)
X_test  = log.(vars["Xtest"] .+ 0.1)

y_train = vec(vars["ytrain"])
y_test  = vec(vars["ytest"])

y_train[y_train .== 0] .= -1
y_test[ y_test  .== 0] .= -1

n, p = size(vars["Xtrain"]);
```

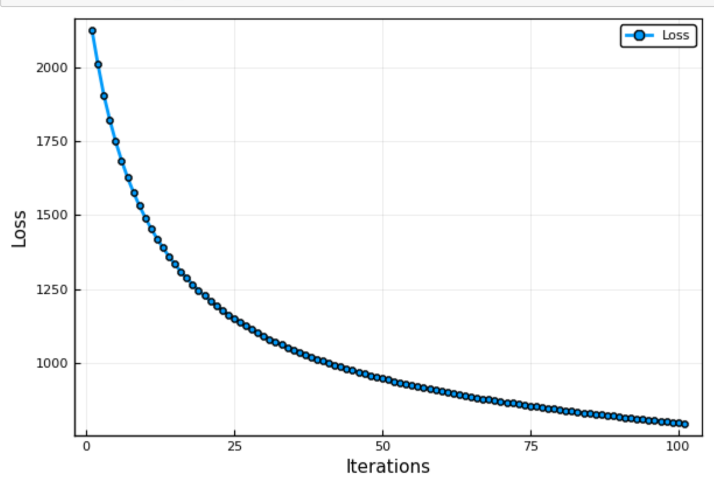
In [3]:
```
σ(a)          = 1 / (1 + exp(-a[1]))
g(w, X, y)    = sum((σ(y[i]*w'*X[i,:]) - 1)*y[i]*X[i,:] for i in 1:size(y)[1])
H(w, X, y)    = sum( σ(y[i]*w'*X[i,:])*(1 - σ(y[i]*w'*X[i,:]))*X[i,:]*X[i,:]' for i in 1:size(y)[1])
loss(w, X, y) = sum(log.(1 .+ exp.(-y[i]*w'*X[i,:])) for i in 1:size(y)[1])[1];
```

In [4]:
```
function gradient_descent(g, loss, X, y, x0; α=1E0, max_iter=1000, ϵ=1E0)

    tol = Inf
    hist, l_hist = [], []
    push!(hist, x0), push!(l_hist, loss(x0, X, y))

    for k in 1:max_iter

        tol < ϵ ? break : nothing;

        xk = hist[end]
        gk = g(xk, X, y)

        xk1 = xk - α*gk

        push!(hist, xk1)
        push!(l_hist, loss(xk1, X, y))

        tol = norm(g(xk1, X, y))

    end

    return hist, l_hist

end;
```

In [5]:
```
x_hist, l_hist = gradient_descent(g, loss, X_train, y_train, zeros(p, 1), α=1E-5, max_iter=100);
```

In [6]:
```
plot(l_hist, w=2, marker=:o, box=:on, xlabel="Iterations", ylabel="Loss", label="Loss")
#savefig("problem_2a.png")
```

Out[6]:



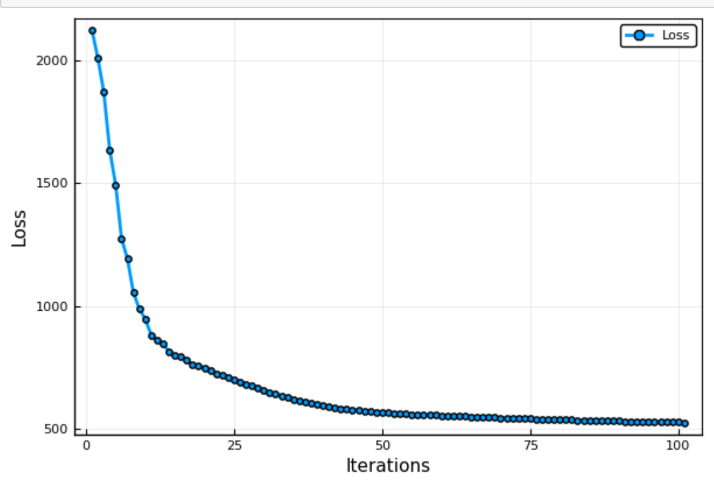
Problem 2(b)

In [7]:
```
function gradient_descent_w_momentum(g, loss, X, y, x0; α=1E0, β=0.9, max_iter=1000, ϵ=1E0)

    tol = Inf
    v0 = zeros(size(x0))
    v_hist, x_hist, l_hist = [], [], []
    push!(v_hist, v0), push!(x_hist, x0), push!(l_hist, loss(x0, X, y))

    for k in 1:max_iter

        tol < ϵ ? break : nothing;

        vk = v_hist[end]
        xk = x_hist[end]

        gk = g(xk, X, y)

        vk1 = β*vk + α*gk
        xk1 = xk - vk1

        push!(v_hist, vk1)
        push!(x_hist, xk1)
        push!(l_hist, loss(xk1, X, y))

        tol = norm(g(xk1, X, y))

    end

    return x_hist, l_hist

end;
```

In [8]:
```
x_hist, l_hist = gradient_descent_w_momentum(g, loss, X_train, y_train, zeros(p, 1), α=1E-5, β=0.9, max_iter=100);
```

In [9]:
```
plot(l_hist, w=2, marker=:o, box=:on, xlabel="Iterations", ylabel="Loss", label="Loss")
#savefig("problem_2b.png")
```

Out[9]:



Problem 2(c)

In [10]:
```
# Armijo line search
```

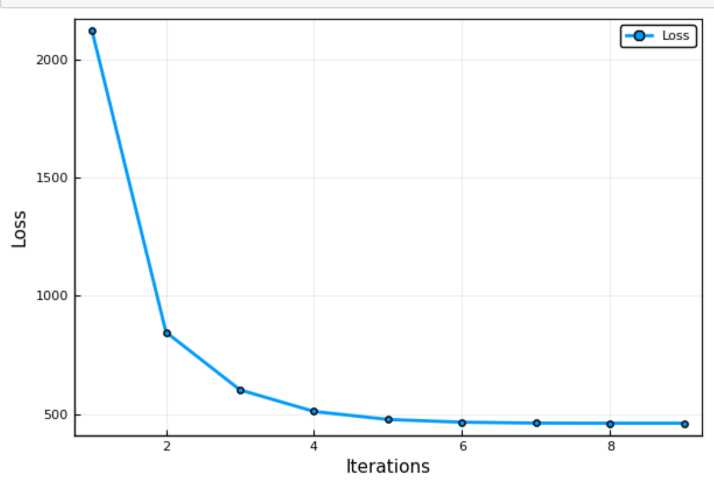
Problem 4(a)

In [11]:
```
function newtons_method(g, H, loss, X, y, x0; α=1E0, max_iter=100, ϵ=1E0)

    tol = Inf
    hist, l_hist = [], []
    push!(hist, x0), push!(l_hist, loss(x0, X, y))

    for i in 1:max_iter

        tol < ϵ ? break : nothing;

        xk = hist[end]
        gk = g(xk, X, y)
        Hk = H(xk, X, y)

        xk1 = xk - α*inv(Hk)gk

        push!(hist, xk1)
        push!(l_hist, loss(xk1, X, y))

        tol = norm(g(xk1, X, y))

    end

    return hist, l_hist

end;
```

In [12]:
```
x_hist, l_hist = newtons_method(g, H, loss, X_train, y_train, zeros(p, 1), α=1);
```

In [13]:
```
plot(l_hist, w=2, marker=:o, box=:on, xlabel="Iterations", ylabel="Loss", label="Loss")
#savefig("problem_4a.png")
```

Out[13]:



Problem 4(b)

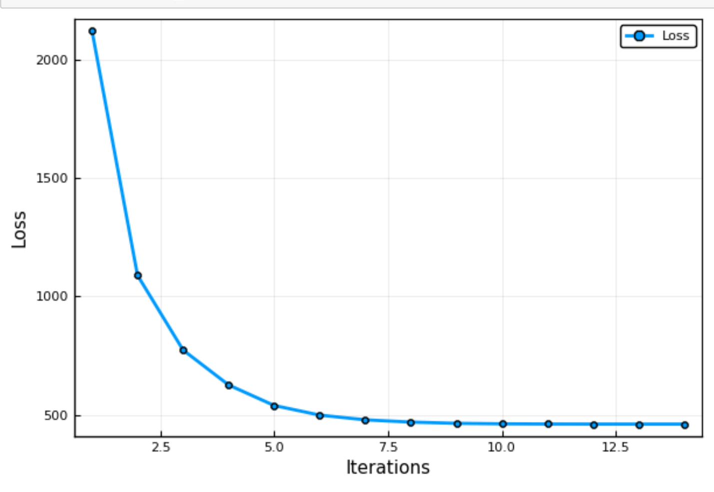
In [14]:
```
Asqrt(w, X, y) = H(w, X, y)^(1/2);
```

In [15]:
```
function randomized_newtons_method(g, A, d, loss, X, y, x0; α=1E0, max_iter=100, ϵ=1E0)

    tol = Inf
    hist, l_hist = [], []
    push!(hist, x0), push!(l_hist, loss(x0, X, y))

    for i in 1:max_iter

        tol < ϵ ? break : nothing;

        xk = hist[end]
        gk = g(xk, X, y)
        Ak = A(xk, X, y)

        n = size(Ak)[1]

        Sk = zeros(d, n)
        for i in 1:d
            Sk[i, rand([1:n...])] = 1/sqrt(n/d)
        end

        Hk = Ak'*Sk'*Sk*Ak

        xk1 = xk - α*inv(Hk)*gk

        push!(hist, xk1)
        push!(l_hist, loss(xk1, X, y))

        tol = norm(g(xk1, X, y))

    end

    return hist, l_hist

    end;
```

In [16]:
```
d = 1000
x_hist, l_hist = randomized_newtons_method(g, Asqrt, d, loss, X_train, y_train, zeros(p, 1), α=2E2, ϵ=1E0, max_iter=50
);
```

In [17]:
```
plot(l_hist, w=2, marker=:o, box=:on, xlabel="Iterations", ylabel="Loss", label="Loss")
#savefig("problem_4b.png")
```

Out[17]:



Problem 5(a)

In [18]:
```
m = 128
n = 1024;
```

In [19]:
```
function hadamard(d)

    if d == 1
        return [[1, 1] [1, -1]]
    else
        return hcat(vcat(hadamard(d-1), hadamard(d-1)), vcat(hadamard(d-1), -hadamard(d-1)))
    end

end
```

Out[19]:
```
hadamard (generic function with 1 method)
```

In [20]:
```
function fjlt(d, m, n)

    H = hadamard(d)
    D = diagm([rand([-1, +1]) for _ in 1:n])
    P = zeros(m, n)
    for i in 1:m
        j = rand([1:n...])
        P[i, j] = 1
    end
    P *= sqrt(n/m)

    S = 1/sqrt(n)*P*H*D;

    return S

end
```

Out[20]:
```
fjlt (generic function with 1 method)
```

In [21]:
```
S = fjlt(10, m, n);
```

Problem 5(b)

In [22]:
```
@show norm(S'*S - I);
```

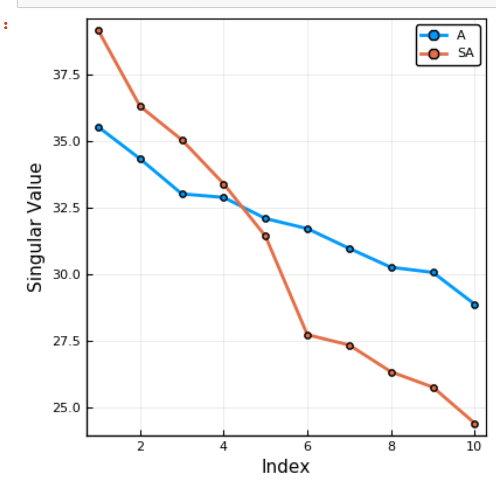
```
norm(S' * S - I) = 89.7997772825746
```

Problem 5(c)

In [23]:
```
A = [randn() for i in 1:1024, j in 1:10];
```

In [24]:
```
UA,  ΣA,  VA  = svd(A)
USA, ΣSA, VSA = svd(S*A)

plot( ΣA,  w=2, marker=:o, label="A",  box=:on, xlabel="Index", ylabel="Singular Value", size=(400,400))
plot!(ΣSA, w=2, marker=:o, label="SA")
#savefig("problem_5c.png")
```

Out[24]:



Problem 5(d)

In [25]:
```
function jl_epsilon(A, S)

    A_dists  = zeros(10,10)
    SA_dists = zeros(10,10)

    for i in 1:10, j in 1:10
        A_dists[i, j]  = norm(A[:, i] - A[:, j])^2
        SA_dists[i, j] = norm(S*(A[:, i] - A[:, j]))^2
    end

    ϵ = abs.(SA_dists ./ A_dists .- 1)
    ϵ[isnan.(ϵ)] .= 0

    return maximum(ϵ)

    end;
```

In [26]:
```
@show jl_epsilon(A, S);
```

```
jl_epsilon(A, S) = 0.29705755463730754
```

In [27]:
```
using ProgressMeter
using Statistics

ϵs = []

@showprogress for i in 1:100
    S = fjlt(10, m, n)
    push!(ϵs, jl_epsilon(A, S))
end

@show minimum(ϵs)
@show mean(ϵs)
@show maximum(ϵs);
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:00:48

minimum(ϵs) = 0.17793509316963618
mean(ϵs) = 0.2960596802090196
maximum(ϵs) = 0.6668913151454343
```

Problem 5(e)

In [28]:
```
A = [randn() for i in 1:1024, j in 1:10]
b = [randn() for i in 1:1024, j in 1:1]

x̂ = (A'*A)\A'*b

SA = S*A
Sb = S*b

x̄ = (SA'*SA)\SA'*Sb;
```

In [29]:
```
@show norm(x̄ - x̂)
@show norm(A*x̄ - A*x̂)
@show norm(A*x̄ - b)^2 / norm(A*x̂ - b)^2;
```

```
norm(x̄ - x̂) = 0.2828462927729973
norm(A * x̄ - A * x̂) = 9.240542031042262
norm(A * x̄ - b) ^ 2 / norm(A * x̂ - b) ^ 2 = 1.0870096195456453
```