Homework 3

Ross B. Alexander (rbalexan@stanford.edu)

Problem 1 – Logistic regression

$X \in \mathbb{R}^{n \times p}$, $y \in \mathbb{R}^n$, $w \in \mathbb{R}^p$

$$\sigma(a) \triangleq \frac{1}{1+e^{-a}} = \frac{e^a}{1+e^a}$$

$$y_i = \begin{cases} +1 & \text{with prob. } \sigma(w^T x_i) \\ -1 & \text{with prob } 1-\sigma(w^T x_i) \end{cases}$$

- our probability is

$$p(y_i \mid w, x_i) = \sigma(y_i w^T x_i) \quad \text{since} \quad \sigma(a) = 1-\sigma(-a)$$
$$p(y \mid w, X) = \prod_{i=1}^n p(y_i \mid w, x_i)$$

- the negative log-likelihood is

$$\begin{aligned} l(w) &\triangleq -\log p(y \mid w, X) \\ &= -\log \prod_{i=1}^n \sigma(y_i w^T x_i) \\ &= -\sum_{i=1}^n \log \sigma(y_i w^T x_i) \\ &= -\sum_{i=1}^n \log\left(\frac{1}{1+\exp(-y_i w^T x_i)}\right) \\ &= \sum_{i=1}^n \log\left(1+\exp(-y_i w^T x_i)\right) \end{aligned}$$

(a) Derive $\nabla l(w)$

$$\begin{aligned} \nabla l(w) &= \nabla_w \sum_{i=1}^n \log\left(1+\exp(-y_i w^T x_i)\right) \\ &= \sum_{i=1}^n \nabla_w \log\left(1+\exp(-y_i w^T x_i)\right) \\ &= \sum_{i=1}^n \frac{\nabla_w\left[1+\exp(-y_i w^T x_i)\right]}{1+\exp(-y_i w^T x_i)} \\ &= \sum_{i=1}^n \frac{\exp(-y_i w^T x_i)}{1+\exp(-y_i w^T x_i)} \nabla_w\left[-y_i w^T x_i\right] \\ &= \sum_{i=1}^n \sigma(-y_i w^T x_i)(-y_i x_i) \end{aligned}$$

- $\sigma(-a) = \sigma(a) - 1$

$$\boxed{\nabla l = \sum_{i=1}^n \left(\sigma(y_i w^T x_i) - 1\right) y_i x_i}$$

(b) Derive $\nabla^2 l(w)$

$$\begin{aligned} \nabla^2 l(w) &= \nabla_w^T\left[\nabla l(w)\right] \\ &= \nabla_w^T \sum_{i=1}^n \left(\sigma(y_i w^T x_i) - 1\right) y_i x_i \\ &= \sum_{i=1}^n y_i x_i \nabla_w^T\left[\sigma(y_i w^T x_i) - 1\right] \\ &= \sum_{i=1}^n y_i x_i \nabla_w^T\left[\frac{1}{1+\exp(-y_i w^T x_i)}\right] \\ &= \sum_{i=1}^n y_i x_i \left[\frac{\left(1+\exp(-y_i w^T x_i)\right)\nabla_w^T[1] - (1)\nabla_w^T\left[1+\exp(-y_i w^T x_i)\right]}{\left(1+\exp(-y_i w^T x_i)\right)^2}\right] \\ &= \sum_{i=1}^n y_i x_i \left[\frac{-\exp(-y_i w^T x_i)\nabla_w^T\left[-y_i w^T x_i\right]}{\left(1+\exp(-y_i w^T x_i)\right)^2}\right] \end{aligned}$$

$$= \sum_{i=1}^{n} \frac{\exp(-y_i w^T x_i)}{(1+\exp(-y_i w^T x_i))^2} (y_i x_i)(y_i x_i^T)$$

$$= \sum_{i=1}^{n} \left[\frac{\exp(-y_i w^T x_i)}{1+\exp(-y_i w^T x_i)}\right]\left[\frac{1}{1+\exp(-y_i w^T x_i)}\right] y_i^2 x_i x_i^T$$

$$= \sum_{i=1}^{n} \sigma(-y_i w^T x_i)\,\sigma(y_i w^T x_i)\, x_i x_i^T$$

$y_i^2 = 1 \quad \forall \quad y_i \in \{\pm 1\}$

$\sigma(-a) = 1 - \sigma(a)$

$$\boxed{\nabla^2 \ell = \sum_{i=1}^{n} \sigma(y_i w^T x_i)\left(1-\sigma(y_i w^T x_i)\right) x_i x_i^T}$$

(c) Is $\ell(w)$ convex?

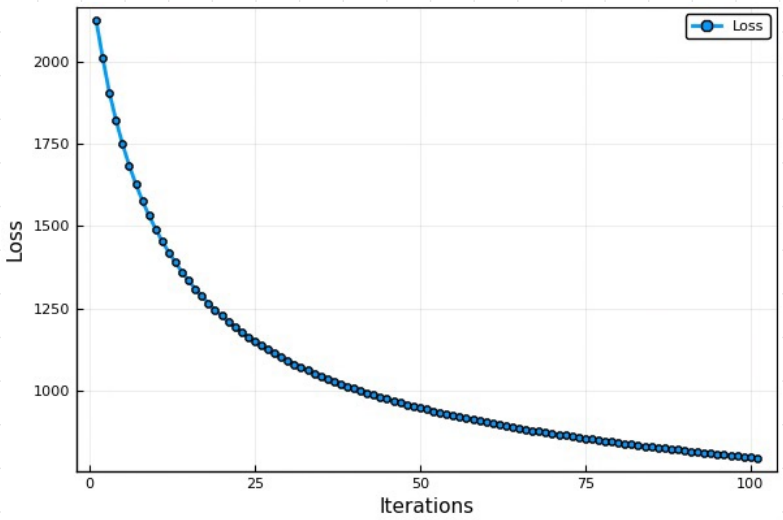
Since $\nabla^2 \ell = \sum_{i=1}^{n} \sigma(y_i w^T x_i)\left(1-\sigma(y_i w^T x_i)\right) x_i x_i^T$, which is analogous to an eigenvalue decomposition and $\sigma: \mathbb{R} \to [0,1]$, then all of the eigenvalues of the Hessian, $\lambda_i = \sigma(y_i w^T x_i)\left(1-\sigma(y_i w^T x_i)\right) \geq 0$, and therefore $\nabla^2 \ell \succeq 0$, which is a sufficient condition for convexity.

Problem 2 - Logistic regression for spam classification

(a) Run gradient descent with a fixed step size. Plot the value of the cost function at each step and find a reasonable step size for fast convergence.

SEE ATTACHED CODE

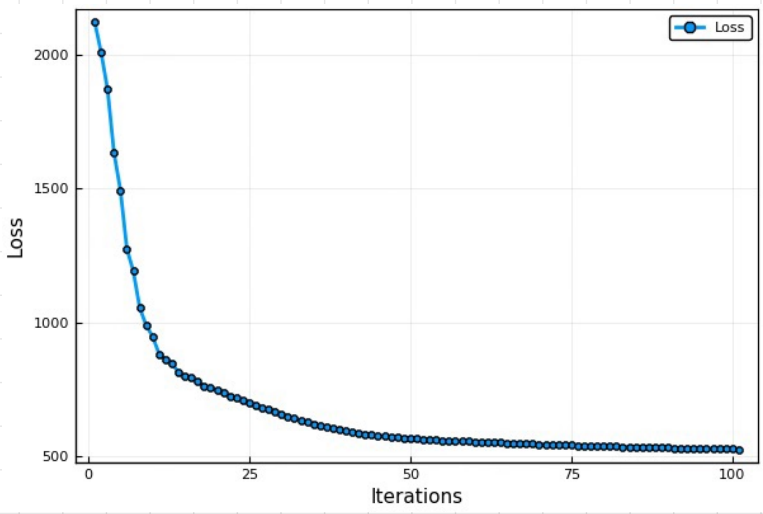
learning rate = 1×10^{-5}



(b) Repeat (a) using gradient descent with momentum.

SEE ATTACHED CODE

learning rate = 1×10^{-5}, momentum decay = 0.9



(c) Implement gradient descent with Armijo line search.

Problem 3 – Newton's method affine invariance

Let $f: \mathbb{R}^n \to \mathbb{R}$ be convex. Consider an affine transform $y \to Ay+b$, $A \in \mathbb{R}^{n \times n}$ invertible, $y \in \mathbb{R}^n$, $b \in \mathbb{R}^n$. Define $g: \mathbb{R}^n \to \mathbb{R}$ by $g(y) = f(Ay+b)$. Denote $x^{(k)}$ as the k^{th} iterate of Newton's method performed on f. Denote $y^{(k)}$ as the kth iterate of Newton's method performed on g.

(a) Show that if $x^{(k)} = Ay^{(k)} + b$, then $x^{(k+1)} = Ay^{(k+1)} + b$

- from Newton's method, we have

$$x^{(k+1)} = x^{(k)} - \nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)})$$
$$y^{(k+1)} = y^{(k)} - \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})$$

- we will show $x^{(k+1)} = Ay^{(k+1)} + b$

$$\begin{aligned} y^{(k+1)} &= y^{(k)} - \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)}) \\ &= y^{(k)} - (A^T \nabla^2 f(x^{(k)}) A)^{-1} A^T \nabla f(x^{(k)}) \\ &= y^{(k)} - A^{-1} \nabla^2 f(x^{(k)})^{-1} A^{-T} A^T \nabla f(x^{(k)}) \\ &= y^{(k)} - A^{-1} \nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)}) \\ &= y^{(k)} - A^{-1}(x^{(k)} - x^{(k+1)}) \\ A^{-1} x^{(k+1)} &= y^{(k+1)} - y^{(k)} + A^{-1} x^{(k)} \\ A A^{-1} x^{(k+1)} &= Ay^{(k+1)} - Ay^{(k)} + x^{(k)} \\ x^{(k+1)} &= Ay^{(k+1)} + b - \cancel{Ay^{(k)}} - \cancel{b} + \cancel{x^{(k)}} \end{aligned}$$

- using $x^{(k)} = Ay^{(k)} + b$, $g(y) = f(Ay+b)$

$$\nabla f(Ay^{(k)} + b) = A^T \nabla f(x^{(k)})$$
$$\nabla^2 f(Ay^{(k)} + b) = A^T \nabla^2 f(x^{(k)}) A$$

- using Newton's method on $f(x)$

$$\boxed{x^{(k+1)} = Ay^{(k+1)} + b} \quad \blacksquare$$

(b) Show that Newton's decrement does not depend on the coordinates, i.e. show that $\lambda(x^{(k)}) = \lambda(y^{(k)})$, where $\lambda(x) = \left(\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)\right)^{\frac{1}{2}}$

- we have

$$\lambda(x^{(k)}) = \lambda(y^{(k)})$$
$$\left(\nabla f(x^{(k)})^T \nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)})\right)^{\frac{1}{2}} = \left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}}$$
$$\left(\nabla f(Ay^{(k)}+b)^T \nabla^2 f(Ay^{(k)}+b)^{-1} \nabla f(Ay^{(k)}+b)\right)^{\frac{1}{2}} = \left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}}$$
$$\left((A^T \nabla g(y^{(k)}))^T (A^T \nabla^2 g(y^{(k)}) A)^{-1} (A^T \nabla g(y^{(k)}))\right)^{\frac{1}{2}} = \left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}}$$
$$\left(\nabla g(y^{(k)})^T A A^{-1} \nabla^2 g(y^{(k)})^{-1} A^{-T} A^T \nabla g(y^{(k)})\right)^{\frac{1}{2}} = \left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}}$$

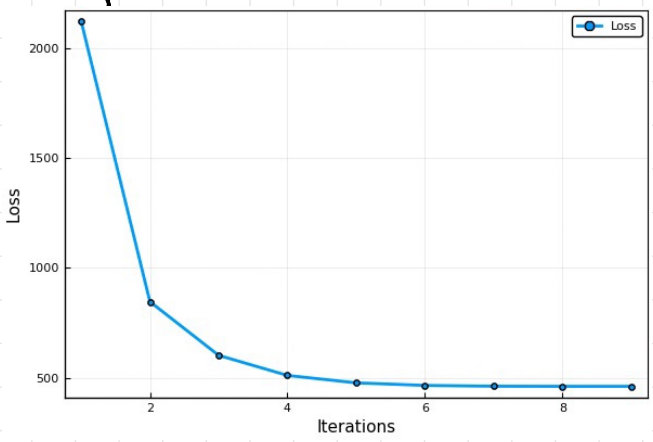
$$\boxed{\left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}} = \left(\nabla g(y^{(k)})^T \nabla^2 g(y^{(k)})^{-1} \nabla g(y^{(k)})\right)^{\frac{1}{2}}} \quad \blacksquare$$

Problem 4 - Newton's method for convex optimization

(a) Implement Newton's method for the logistic regression in Problem 2. Plot the value of the cost function at each iteration and find a reasonable step size for convergence.

SEE ATTACHED CODE

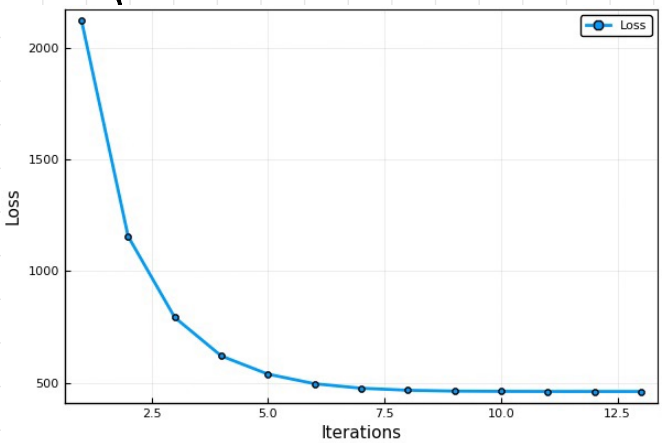
learning rate = 1



(b) Implement randomized Newton's method with uniform sampling sketch, i.e. sampling rows of $H^{1/2}$ uniformly at random. Plot the value of the cost function at each iteration and find a reasonable step size & sketch size for convergence.

SEE ATTACHED CODE

learning rate = 2×10^2, sketch dimension = 1000



Problem 5 - Fast Johnson-Lindenstrauss transform (FJLT) using Hadamard matrices

(a) Construct a 128×1024 FJLT matrix

SEE ATTACHED CODE

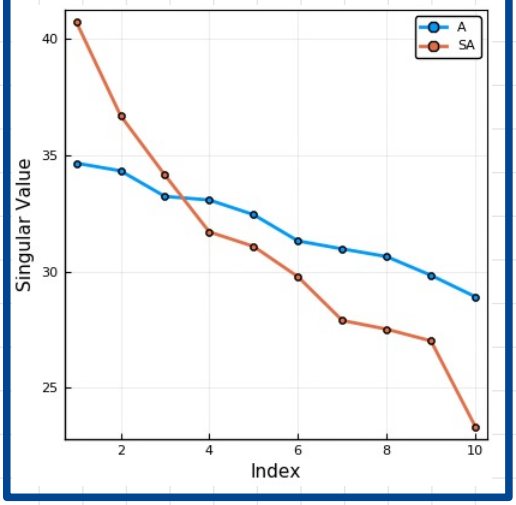
(b) Verify that $S^T S \approx I$

SEE ATTACHED CODE

```
norm(S' * S - I) = 92.6066952223218
```

(c) Generate a 1024×10 data matrix A with iid Gaussian entries. Plot the singular values of A & SA.

SEE ATTACHED CODE



(d) Since SA is a JL embedding of A, verify that pairwise distances are approximately preserved, i.e. $\exists \epsilon > 0$ s.t.

$$(1-\epsilon)\|A_i - A_j\|_2^2 \leq \|S(A_i - A_j)\|_2^2 \leq (1+\epsilon)\|A_i - A_j\|_2^2 \quad \forall\ i,j$$

Find the smallest ϵ that satisfies the above equation for a single realization and the minimum, mean, and maximum ϵ over 100 realizations

SEE ATTACHED CODE

```
jl_epsilon(A, S) = 0.3439616619671395
minimum(εs)      = 0.18904810352785817
mean(εs)         = 0.29887169094967453
maximum(εs)      = 0.4805564370933668
```

(e) Generate a 1024×10 data matrix A and 1024×1 target vector b using iid Gaussians. Solve

$$x^* = \arg\min_x \|Ax - b\|_2^2$$
$$\tilde{x} = \arg\min_x \|SAx - Sb\|_2^2$$

Compute the Euclidean distance between the solutions $\|\hat{x} - x^*\|_2$, the predictions $\|A(\tilde{x} - x^*)\|_2$, and the approximation ratio $(\|A\tilde{x} - b\|_2^2 / \|Ax^* - b\|_2^2)$.

SEE ATTACHED CODE

```
norm(x̃ - x̂)                                = 0.2946934264556728
norm(A * x̃ - A * x̂)                        = 9.240428869445797
norm(A * x̃ - b) ^ 2 / norm(A * x̂ - b) ^ 2 = 1.0853115942400169
```

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"]);
```

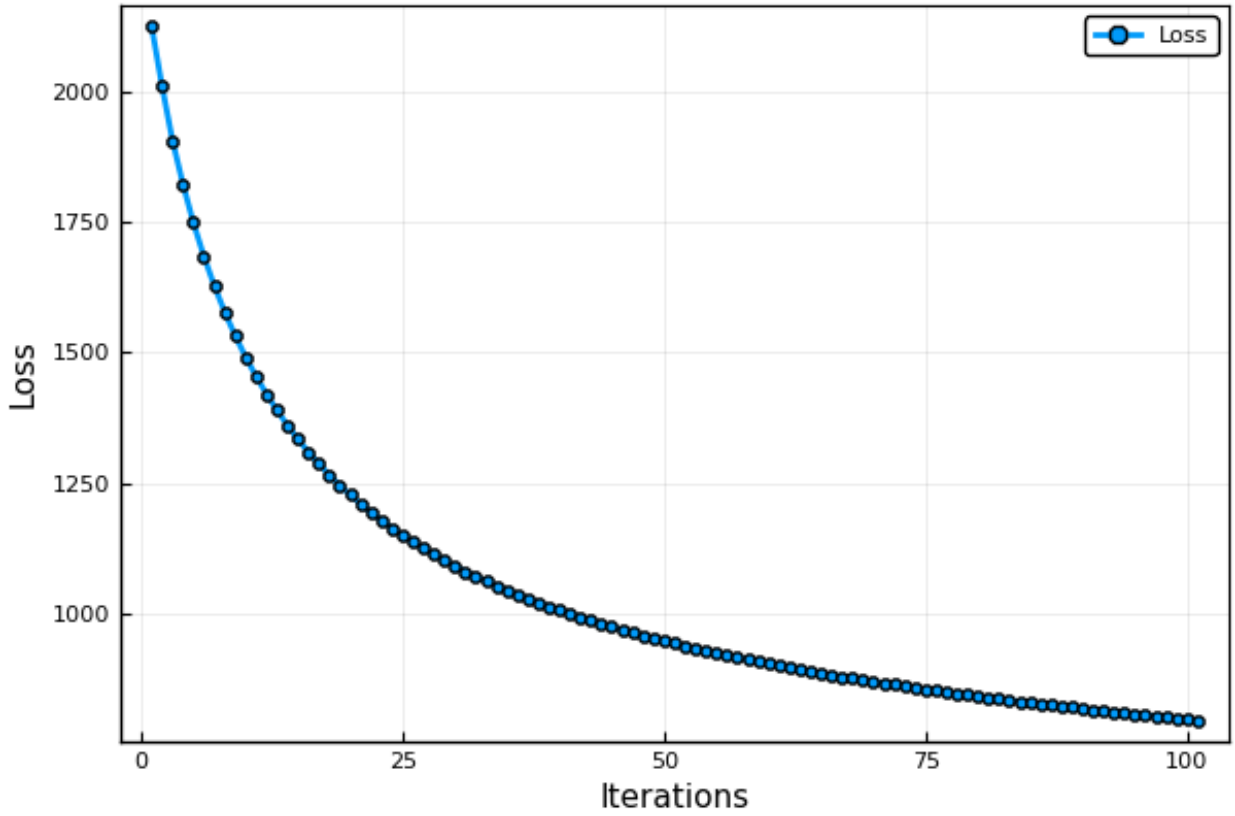
In [3]:
```
σ(a)          = 1 / (1 + exp(-a[1]))
g(w, X, y)    = sum((σ(y[i]*w'*X[i,:]) - 1)*y[i]*X[i,:] for i in 1:siz
e(y)[1])
H(w, X, y)    = sum( σ(y[i]*w'*X[i,:])*(1 - σ(y[i]*w'*X[i,:]))*X[i,:]*
X[i,:]' for i in 1:size(y)[1])
loss(w, X, y) = sum(log.(1 .+ exp.(-y[i]*w'*X[i,:])) for i in 1:size(y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



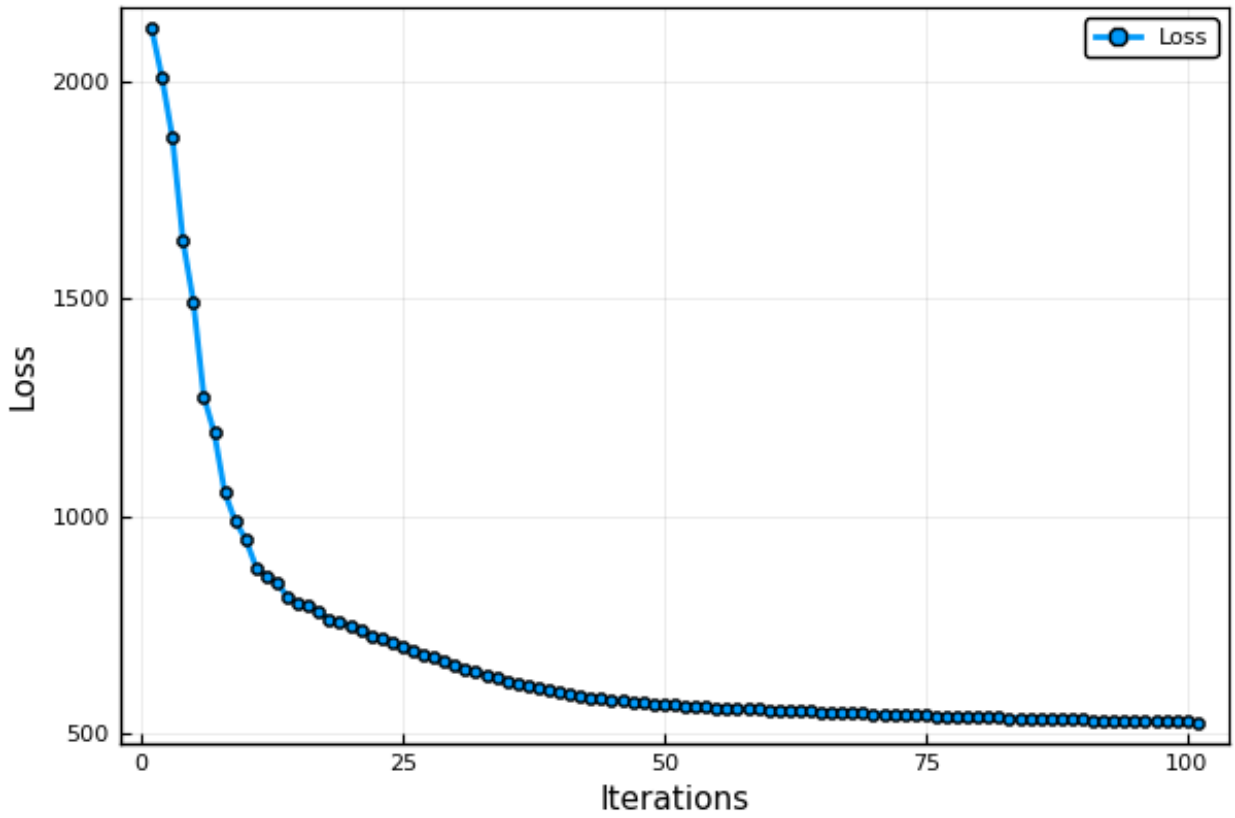
Problem 2(b)

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
```

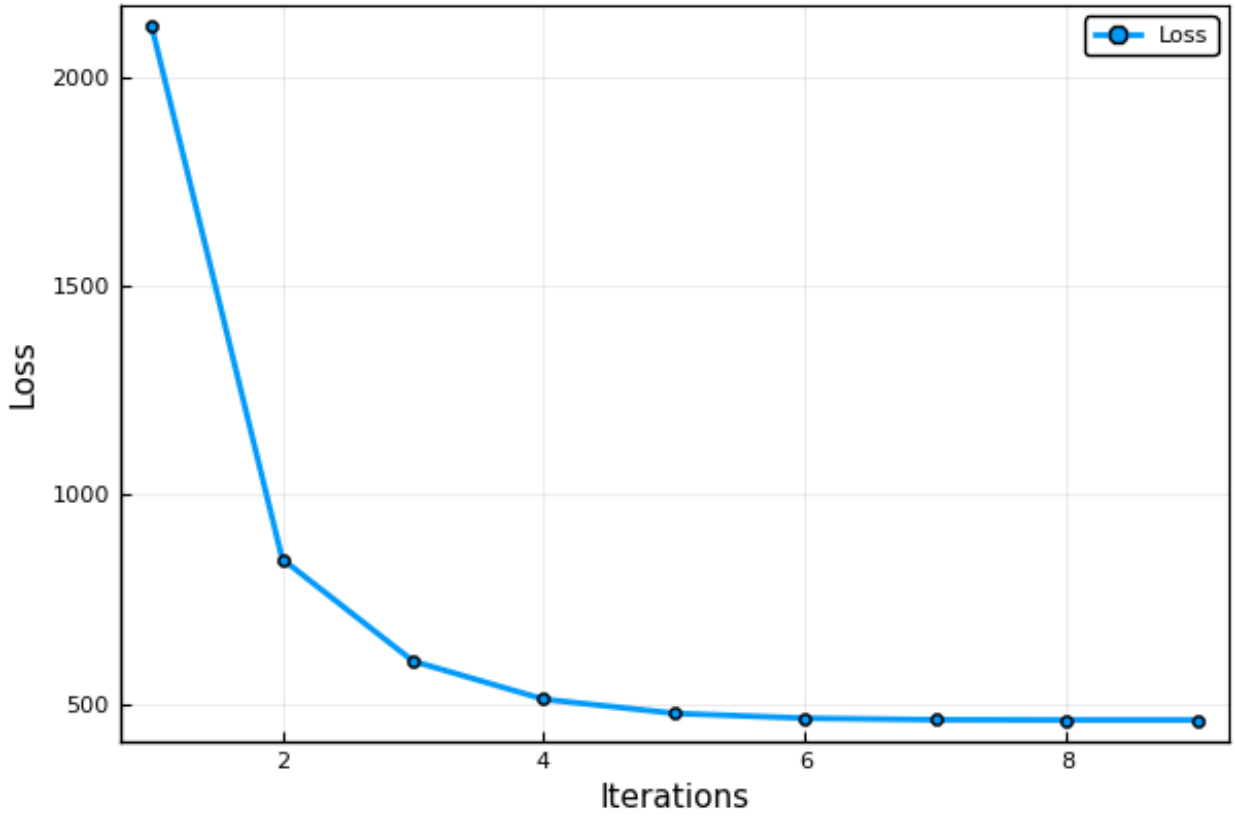
Problem 4(a)

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)

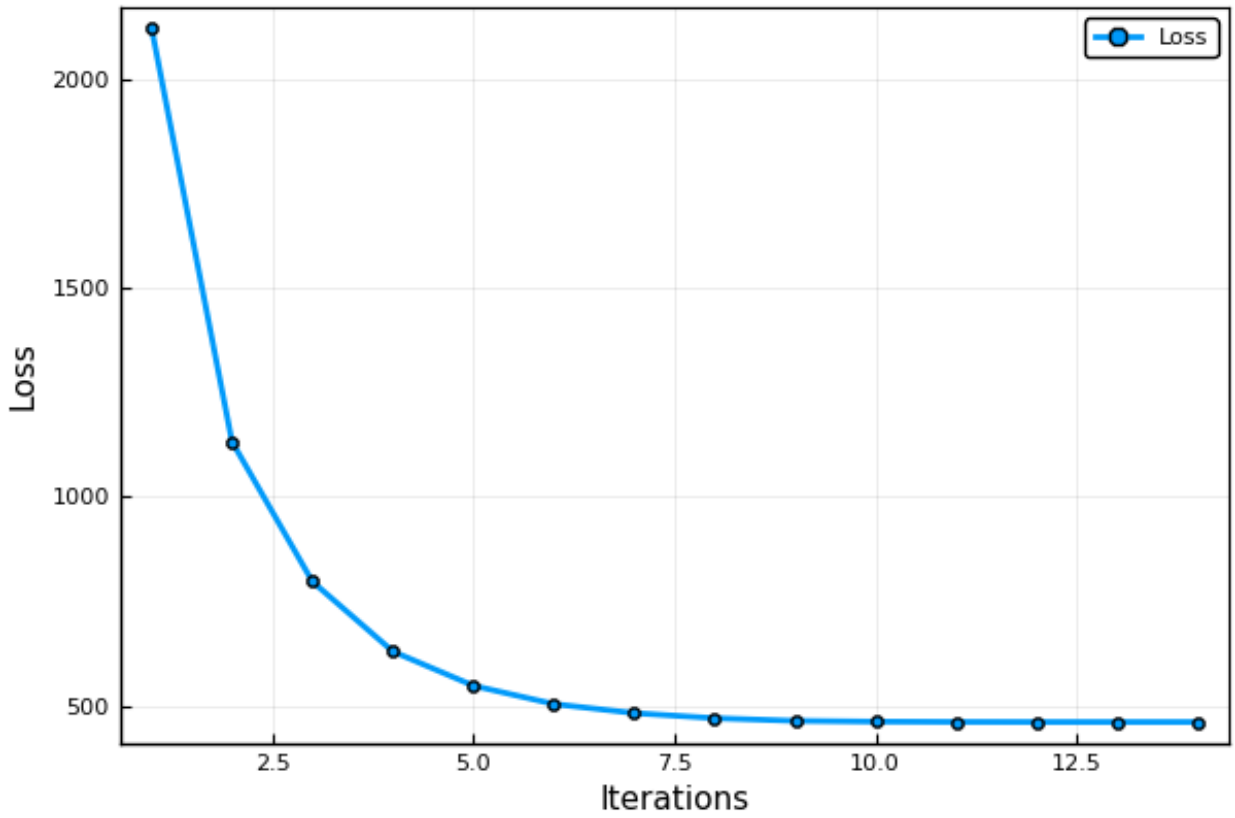
In [14]:
```
Asqrt(w, X, y) = H(w, X, y)^(1/2);
```

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="Los
s", 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) = 90.50966799187809
```

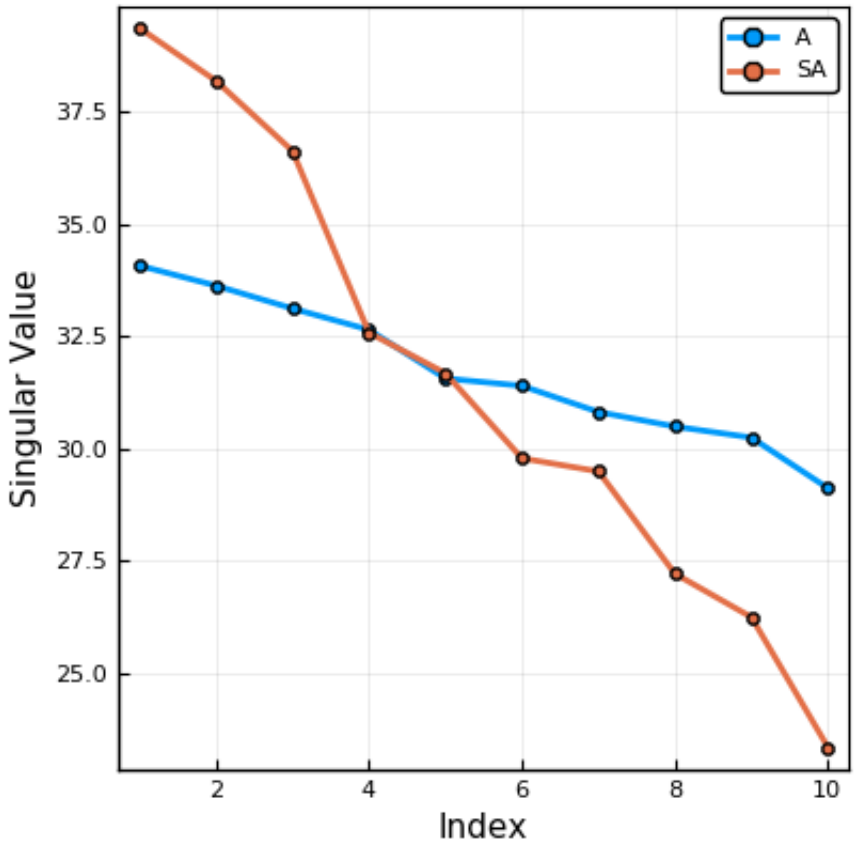
Out[22]:
```
90.50966799187809
```

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;
```

Out[25]:
```
jl_epsilon (generic function with 1 method)
```

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

```
jl_epsilon(A, S) = 0.27278541464284056
```

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
:53

minimum(ϵs) = 0.19199664097167368
mean(ϵs) = 0.31058823564990584
maximum(ϵs) = 0.5110472130475483
```

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

x̃ = (SA'*SA)\SA'*Sb;
```

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

```
norm(x̃ - x̂) = 0.21136553345447284
norm(A * x̃ - A * x̂) = 6.587023953860569
norm(A * x̃ - b) ^ 2 / norm(A * x̂ - b) ^ 2 = 1.0459242514997966
```