Problem 1

In [83]:
```
using MAT
using LinearAlgebra
using SparseArrays
using ProgressMeter
using Plots; pyplot();
using Distributions
```

In [11]:
```
vars = matread("mnist_all.mat");

X0 = Float64.(vars["train0"])
X1 = Float64.(vars["train1"])

y0 = zeros(size(X0)[1])
y1 = ones( size(X1)[1])

X = [X0
     X1]

y = [y0
     y1];
```

In [12]:
```
σ(z) = z >= 0 ? 1 / (1 + exp(-z)) : exp(z) / (1 + exp(z));
```

Problem 1(b)

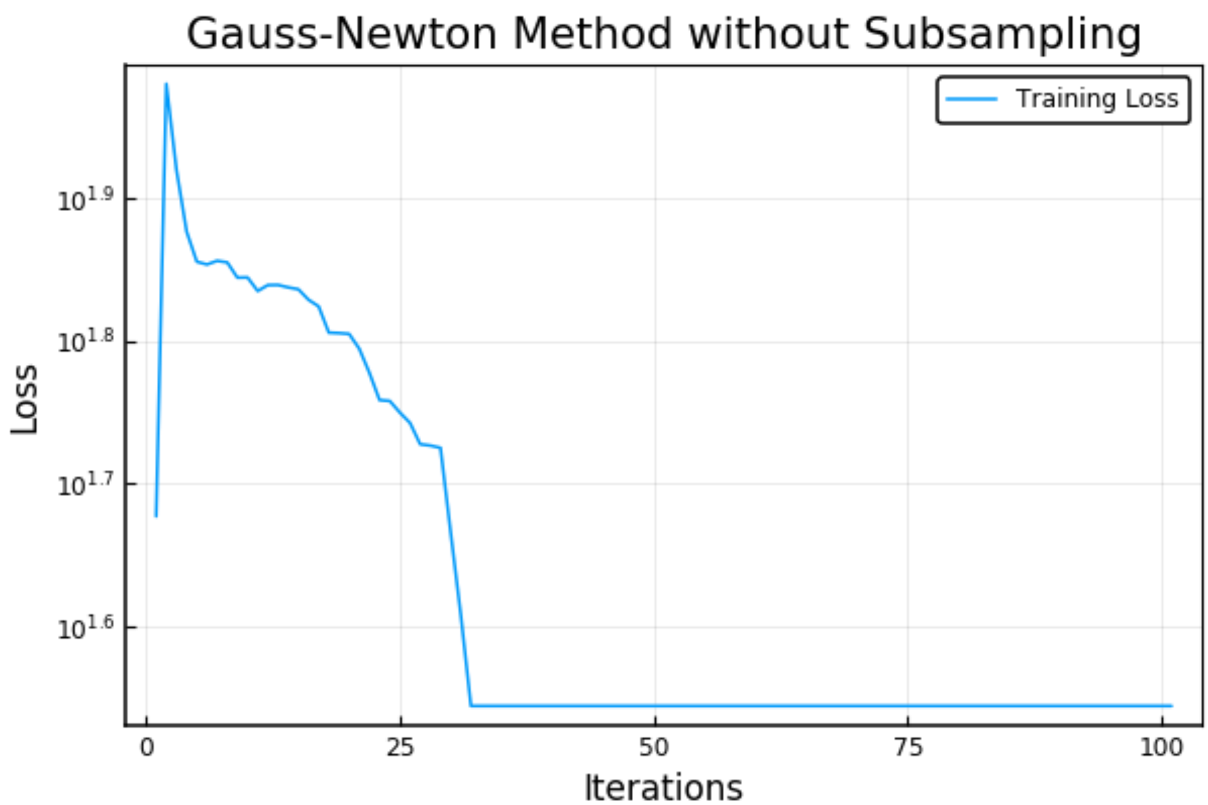
In [101]:
```
function damped_gauss_newton_algorithm(X, y, w0; μ=0.99, k_max=20)

    X = sparse(X)

    hist = []
    loss = []
    push!(hist, w0)
    push!(loss, norm(σ.(X*w0) - y))
    wk = w0

    @showprogress for k in 1:k_max

        σk = σ.(X*wk)
        Σk = spdiagm(0 => σk .* (1 .- σk))
        Jk = Σk*X

        dk = wk + inv(Array(Jk'*Jk) + 1E-5*I)*(Jk'*(y-σk))

        wk1 = (1-μ)*wk + μ*dk

        push!(hist, wk1)
        push!(loss, norm(σ.(X*wk1) - y))
        wk = wk1

    end

    return hist, loss

end;
```

In [207]:
```
hist, loss = damped_gauss_newton_algorithm(X, y, randn(size(X)[2])*1E-
1, μ=0.99, k_max=100);
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:01
:53
```

In [208]:

```
plot(loss, box=:on, yscale=:log10, label="Training Loss", thickness_sc
aling=1.1)
xlabel!("Iterations")
ylabel!("Loss")
title!("Gauss-Newton Method without Subsampling")
savefig("1b.png")
```



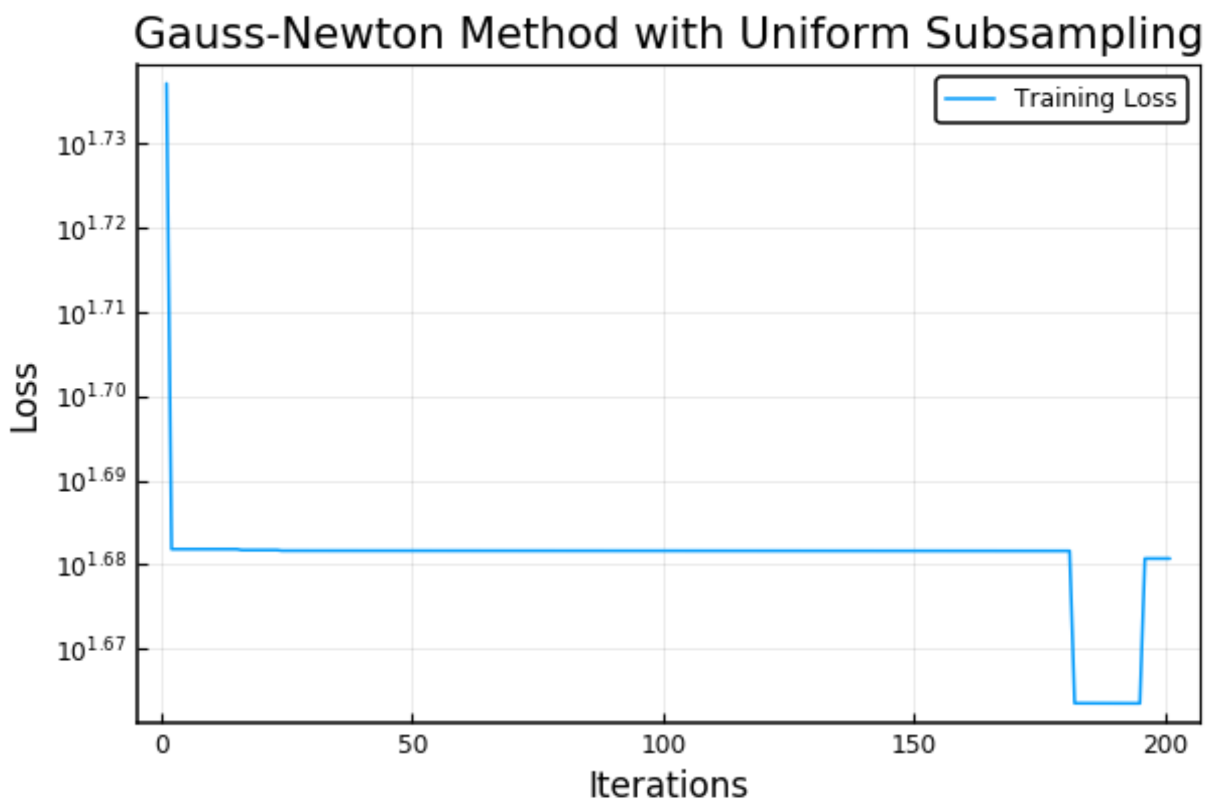
Problem 1(c)

In [209]:
```
function uniform_sampling_damped_gauss_newton_algorithm(X, y, w0; μ=0.
99, k_max=20, m=100)

    X = sparse(X)

    hist = []
    loss = []
    push!(hist, w0)
    push!(loss, norm(σ.(X*w0) - y))
    wk = w0

    @showprogress for k in 1:k_max

        σk = σ.(X*wk)
        Σk = spdiagm(0 => σk .* (1 .- σk))
        Jk = Σk*X

        n = size(Jk)[1]

        S = spzeros(m, n)
        for i in 1:m
            S[i, rand([1:n]...)] = 1
        end

        dk = wk + inv(Array((S*Jk)'*(S*Jk)) + 1E-5*I)*((S*Jk)'*(S*(y-σ
k)))

        wk1 = (1-μ)*wk + μ*dk

        push!(hist, wk1)
        push!(loss, norm(σ.(X*wk1) - y))
        wk = wk1

    end

    return hist, loss

end;
```

In [210]:
```
hist, loss = uniform_sampling_damped_gauss_newton_algorithm(X, y, rand
n(size(X)[2])*1E-1, μ=0.9, k_max=200, m=100);
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:00
:31
```

In [211]:
```
plot(loss, box=:on, yscale=:log10, label="Training Loss", thickness_sc
aling=1.1)
xlabel!("Iterations")
ylabel!("Loss")
title!("Gauss-Newton Method with Uniform Subsampling")
savefig("1c_uniform.png")
```



In [212]:

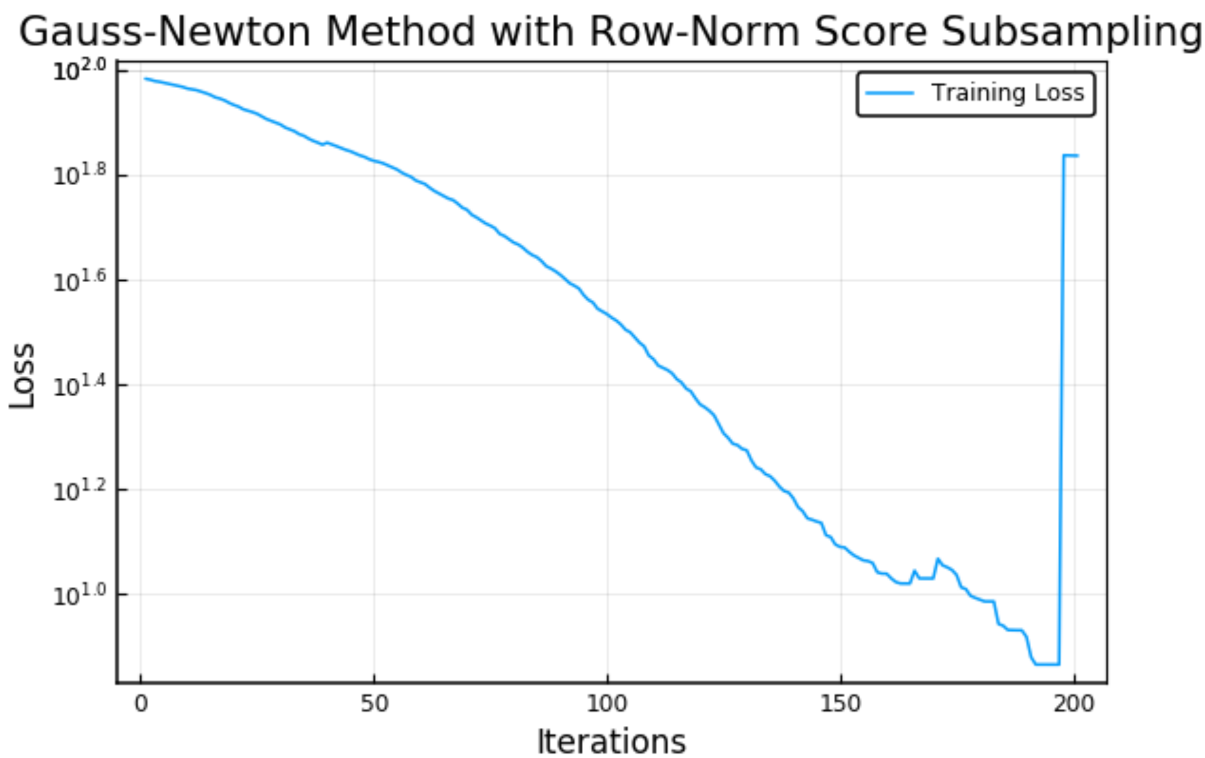
```
function row_norm_score_sampling_damped_gauss_newton_algorithm(X, y, w
0; μ=0.99, k_max=20, m=100)

    X = sparse(X)

    hist = []
    loss = []
    push!(hist, w0)
    push!(loss, norm(σ.(X*w0) - y))
    wk = w0

    @showprogress for k in 1:k_max

        σk = σ.(X*wk)
        Σk = spdiagm(0 => σk .* (1 .- σk))
        Jk = Σk*X

        n = size(Jk)[1]

        row_score_dist = DiscreteNonParametric(1:n, normalize([norm(ro
w)^2 for row in eachrow(Jk)], 1))

        S = spzeros(m, n)
        for i in 1:m
            S[i, rand(row_score_dist)] = 1
        end

        dk = wk + inv(Array((S*Jk)'*(S*Jk)) + 1E-5*I)*((S*Jk)'*(S*(y-σ
k)))

        wk1 = (1-μ)*wk + μ*dk

        push!(hist, wk1)
        push!(loss, norm(σ.(X*wk1) - y))
        wk = wk1

    end

    return hist, loss

end;
```

In [213]:

```
hist, loss = row_norm_score_sampling_damped_gauss_newton_algorithm(X,
y, randn(size(X)[2])*1E-1, μ=0.9, k_max=200, m=100);
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:04
:14
```

In [214]:
```
plot(loss, box=:on, yscale=:log10, label="Training Loss", thickness_sc
aling=1.1)
xlabel!("Iterations")
ylabel!("Loss")
title!("Gauss-Newton Method with Row-Norm Score Subsampling")
savefig("1c_row_norm.png")
```



Problem 2

In [160]:
```
using CSV
using DataFrames
using Statistics
using Distributions
```

In [161]:
```
data = CSV.read("YearPredictionMSD.txt", DataFrame, header=false);
```

In [162]:
```
A = Array(data[:, 2:end])
A = (A .- mean(A, dims=1)) ./ std(A, dims=1);
```

In [163]:
```
b = Array(data[:, 1])
b = (b .- minimum(b))/(maximum(b) - minimum(b));
```

In [164]:
```
A_train = A[1:463715, :]
A_test  = A[463716:end, :]
b_train = b[1:463715]
b_test  = b[463716:end];
```

Problem 2(a)

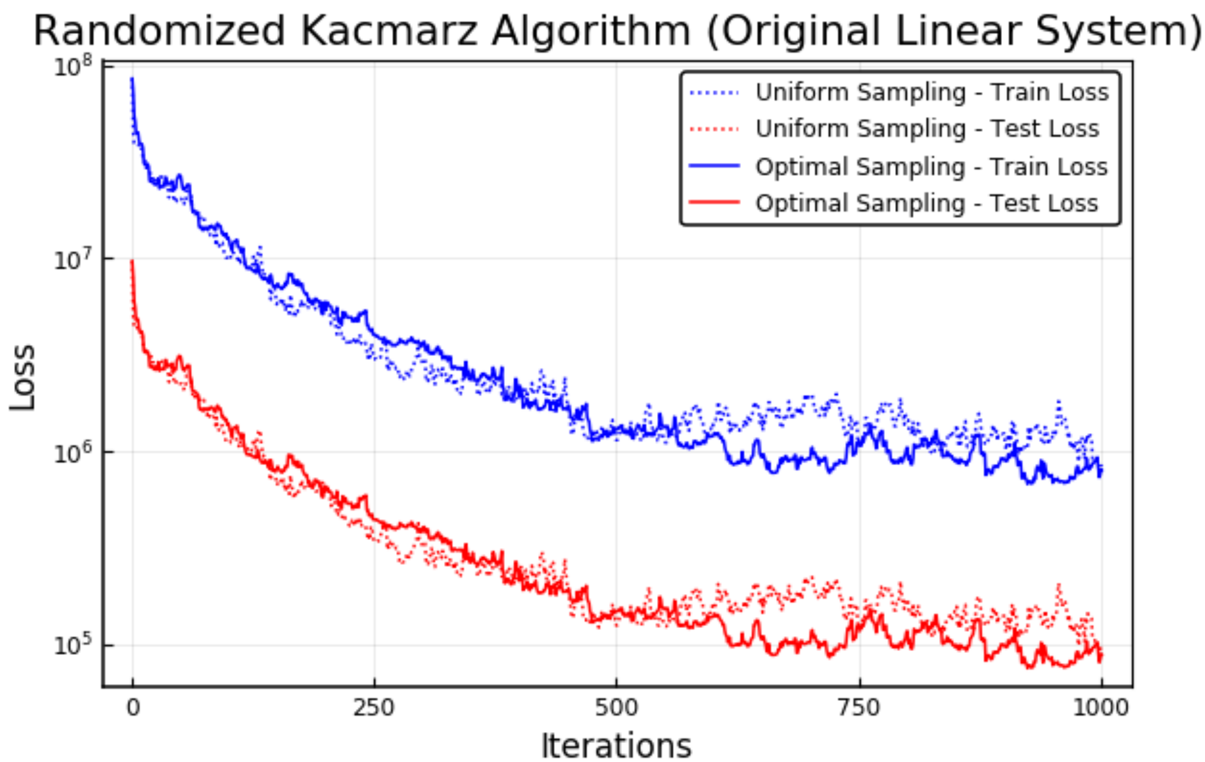
In [167]:
```
function randomized_kacmarz(A_train, b_train, A_test, b_test, x0, dist
; t_max=1000)

    hist       = []
    train_loss = []
    test_loss  = []

    push!(hist, x0)
    push!(train_loss, norm(A_train*x0 - b_train)^2)
    push!(test_loss,  norm(A_test*x0  - b_test )^2)

    x = x0

    @showprogress for t in 1:t_max

        it = rand(dist)
        ait = A_train[it, :]
        bit = b_train[it]
        x = x - ait/(norm(ait)^2) * (ait'*x - bit)

        push!(hist, x)
        push!(train_loss, norm(A_train*x - b_train)^2)
        push!(test_loss,  norm(A_test*x  - b_test )^2)

    end

    return hist, train_loss, test_loss

end;
```

In [168]:
```
n = size(A_train)[1]

unif_dist = DiscreteNonParametric([i for i in 1:n], [1/n for i in 1:n]
)

row_scores = [norm(row)^2 for row in eachrow(A_train)]
probs      = normalize(row_scores, 1)
opt_dist = DiscreteNonParametric([i for i in 1:n], probs)

t_max = 1000
hist, unif_train_loss, unif_test_loss = randomized_kacmarz(A_train, b_
train,
                                                           A_test,  b_
test,
                                                           ones(size(A
_train)[2]), unif_dist, t_max=t_max);
hist, opt_train_loss,  opt_test_loss  = randomized_kacmarz(A_train, b_
train,
                                                           A_test,  b_
test,
                                                           ones(size(A
_train)[2]), opt_dist,  t_max=t_max);
```

```
Progress: 100%|██████████████████████████████████████████| Time: 0:00
:43
Progress: 100%|██████████████████████████████████████████| Time: 0:00
:47
```

In [169]:

```
plot(yscale=:log10, box=:on, thickness_scaling=1.1)
plot!(unif_train_loss, label="Uniform Sampling - Train Loss", color=:b
lue, ls=:dot)
plot!(unif_test_loss,  label="Uniform Sampling - Test Loss",  color=:r
ed,  ls=:dot)
plot!(opt_train_loss,  label="Optimal Sampling - Train Loss", color=:b
lue)
plot!(opt_test_loss,   label="Optimal Sampling - Test Loss",  color=:r
ed)
title!("Randomized Kacmarz Algorithm (Original Linear System)")
xlabel!("Iterations")
ylabel!("Loss")
savefig("2a.png")
```



Problem 2(c)

In [173]:

```
m, n = size(A_train)

A_train = sparse(A_train)
b_train = sparse(b_train)

A_train_aug = [A_train       spdiagm(0 => [-1 for i in 1:m])
               zeros(n, n)   A_train']
b_train_aug = [b_train
               zeros(n)];
```

In [174]:
```
function randomized_kacmarz_aug(A_train_aug, b_train_aug, A_test, b_te
st, x0, dist; t_max=1000)

    hist       = []
    train_loss = []
    test_loss  = []

    m = size(A_test)[2]

    push!(hist, x0)
    push!(train_loss, norm(A_train_aug*x0 - b_train_aug)^2)
    push!(test_loss,  norm(A_test*x0[1:m] - b_test     )^2)

    x = x0

    @showprogress for t in 1:t_max

        it = rand(dist)
        ait = A_train_aug[it, :]
        bit = b_train_aug[it]
        x = x - ait/(norm(ait)^2) * (ait'*x - bit)

        push!(hist, x)
        push!(train_loss, norm(A_train_aug*x - b_train_aug)^2)
        push!(test_loss,  norm(A_test*x[1:m] - b_test     )^2)

    end

    return hist, train_loss, test_loss

end;
```

In [179]:
```
n = size(A_train_aug)[1]

unif_dist = DiscreteNonParametric(1:n, [1/n for i in 1:n]);
```

In [182]:
```
row_scores = vcat([norm([row, -1])^2 for row in eachrow(A_train)],
                  [norm(row)^2 for row in eachrow(A_train')])
opt_dist  = DiscreteNonParametric(1:n, normalize(row_scores, 1));
```

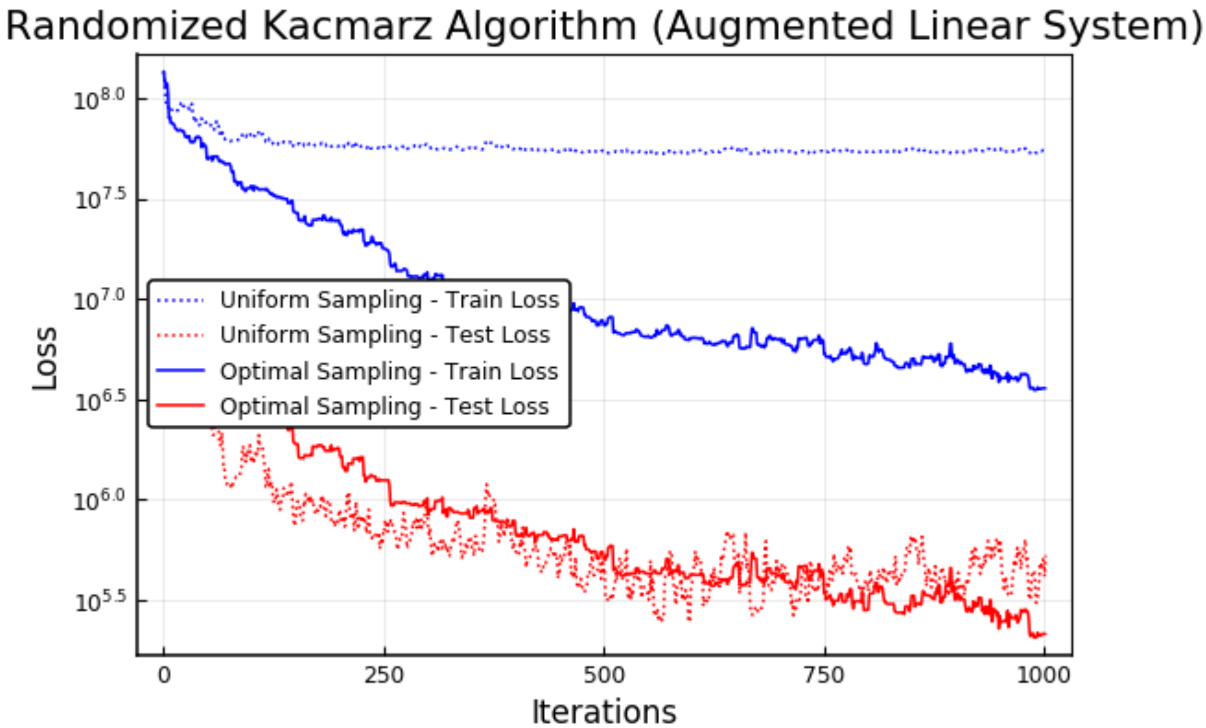
In [186]:

```
t_max = 1000
hist, unif_train_loss, unif_test_loss = randomized_kacmarz_aug(A_train
_aug, b_train_aug,
                                                               A_test,
b_test,
                                                               ones(si
ze(A_train_aug)[2]), unif_dist, t_max=t_max);
hist, opt_train_loss,  opt_test_loss  = randomized_kacmarz_aug(A_train
_aug, b_train_aug,
                                                               A_test,
b_test,
                                                               ones(si
ze(A_train_aug)[2]), opt_dist,  t_max=t_max);
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:05
:38
Progress: 100%|█████████████████████████████████████████| Time: 0:07
:02
```

In [187]:

```
plot(yscale=:log10, box=:on, thickness_scaling=1.1)
plot!(unif_train_loss, label="Uniform Sampling - Train Loss", color=:b
lue, ls=:dot)
plot!(unif_test_loss,  label="Uniform Sampling - Test Loss",  color=:r
ed,  ls=:dot)
plot!(opt_train_loss,  label="Optimal Sampling - Train Loss", color=:b
lue)
plot!(opt_test_loss,   label="Optimal Sampling - Test Loss",  color=:r
ed)
title!("Randomized Kacmarz Algorithm (Augmented Linear System)")
xlabel!("Iterations")
ylabel!("Loss")
savefig("2c.png")
```


Randomized Kacmarz Algorithm (Augmented Linear System)
Loss
Iterations
Uniform Sampling - Train Loss
Uniform Sampling - Test Loss
Optimal Sampling - Train Loss
Optimal Sampling - Test Loss
$10^{8.0}$
$10^{7.5}$
$10^{7.0}$
$10^{6.5}$
$10^{6.0}$
$10^{5.5}$
0
250
500
750
1000

Problem 2(d)

In [190]:

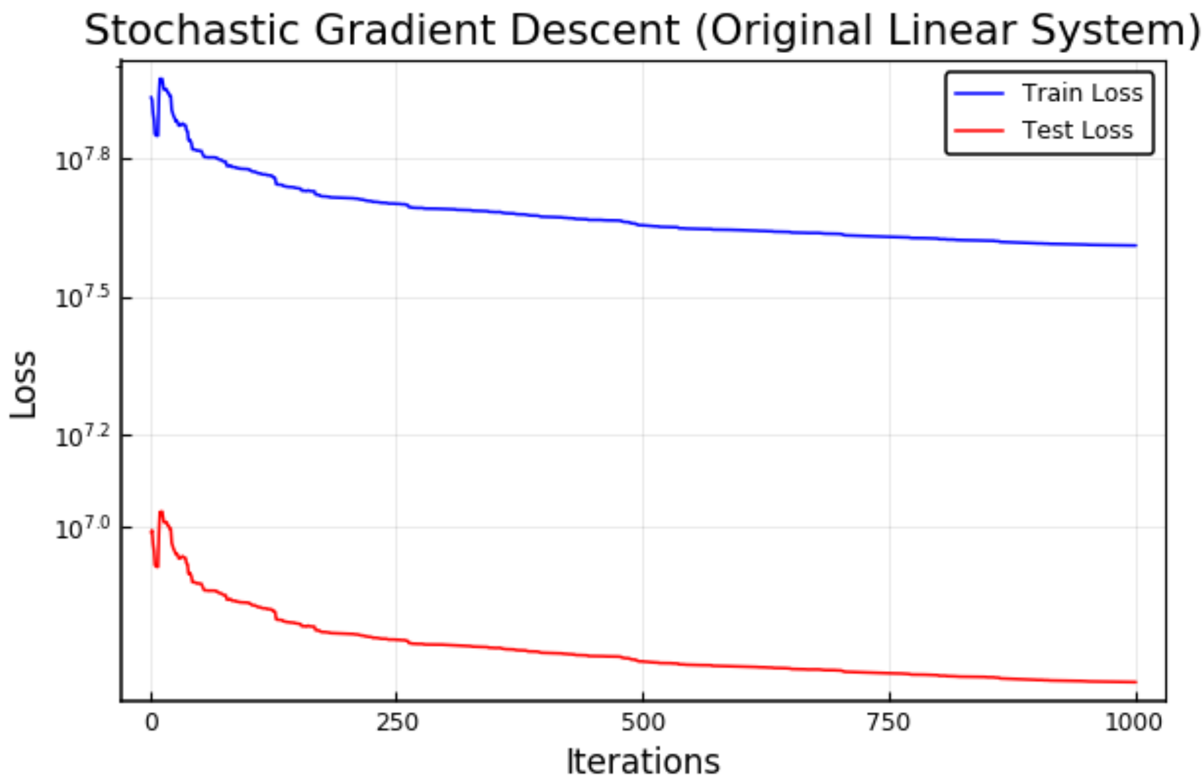
```julia
function sgd(A_train, b_train, A_test, b_test, x0, μ0; t_max=1000)

    hist       = []
    train_loss = []
    test_loss  = []

    push!(hist, x0)
    push!(train_loss, norm(A_train*x0 - b_train)^2)
    push!(test_loss,  norm(A_test*x0  - b_test )^2)

    x = x0
    n = size(A_train)[1]

    @showprogress for t in 1:t_max

        it = rand([1:n]...)
        ait = A_train[it, :]
        bit = b_train[it]
        x = x - μ0/t * ait * (ait'*x - bit)

        push!(hist, x)
        push!(train_loss, norm(A_train*x - b_train)^2)
        push!(test_loss,  norm(A_test*x  - b_test )^2)

    end

    return hist, train_loss, test_loss

end;
```

In [205]:

```julia
t_max = 1000
μ0 = 5E-2

hist, train_loss, test_loss = sgd(A_train, b_train,
                                  A_test,  b_test,
                                  ones(size(A_train)[2]), μ0, t_max=t_
max);
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:02
:47
```

In [206]:
```
plot(yscale=:log10, box=:on, thickness_scaling=1.1)
plot!(train_loss,  label="Train Loss", color=:blue)
plot!(test_loss,   label="Test Loss",  color=:red)
title!("Stochastic Gradient Descent (Original Linear System)")
xlabel!("Iterations")
ylabel!("Loss")
savefig("2d.png")
```



Problem 3

In [226]:
```
n = 5000
d = 1000

A = randn(n, d)
U, Σ, V = svd(A);

Σ̂ = diagm([i^-2 for i in 1:d])

A = U*Σ̂*V';
```

Problem 3(a)

In [267]:
```
ks = round.(Int, 10 .^ range(0, 2, length=11));
```

In [244]:
```
function rank_k_approximation(A, S)

    C = A*S
    Ãk = C*pinv(C)*A

    rel_error    = norm(A*A' - C*C') / norm(A*A')
    approx_error = opnorm(A - Ãk)^2

    return approx_error

end;
```

In [245]:
```
function rank_k_approximation_uniform(A, k)

    n = size(A)[2]

    S = spzeros(n, k)
    for i in 1:k
        S[rand([1:n...]), i] = 1
    end

    return rank_k_approximation(A, S)

end;
```

In [246]:
```
approx_errors_uniform = []
@showprogress for k in ks
    push!(approx_errors_uniform, rank_k_approximation_uniform(A, k))
end
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:01
:03
```

In [247]:
```
function rank_k_approximation_column(A, k)

    n = size(A)[2]

    dist = DiscreteNonParametric(1:n, normalize([norm(col, 2)^2 for co
l in eachcol(A)], 1))

    S = spzeros(n, k)
    for i in 1:k
        S[rand(dist), i] = 1
    end

    return rank_k_approximation(A, S)

end;
```

In [248]:
```
approx_errors_column = []
@showprogress for k in ks
    push!(approx_errors_column, rank_k_approximation_column(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:01
:05
```

In [249]:
```
function rank_k_approximation_gaussian(A, k)

    n = size(A)[2]

    s = [randn() for _ in 1:n*k]
    S = 1/sqrt(k)*reshape(s, (n, k))

    return rank_k_approximation(A, S)

end;
```

In [250]:
```
approx_errors_gaussian = []
@showprogress for k in ks
    push!(approx_errors_gaussian, rank_k_approximation_gaussian(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:01
:02
```

In [251]:
```
function rank_k_approximation_rademacher(A, k)

    n = size(A)[2]

    s = [rand() >= 0.5 ? 1 : -1 for _ in 1:n*k]
    S = reshape(s, (n, k))

    return rank_k_approximation(A, S)

end;
```

In [252]:
```
approx_errors_rademacher = []
@showprogress for k in ks
    push!(approx_errors_rademacher, rank_k_approximation_rademacher(A,
k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:59
```

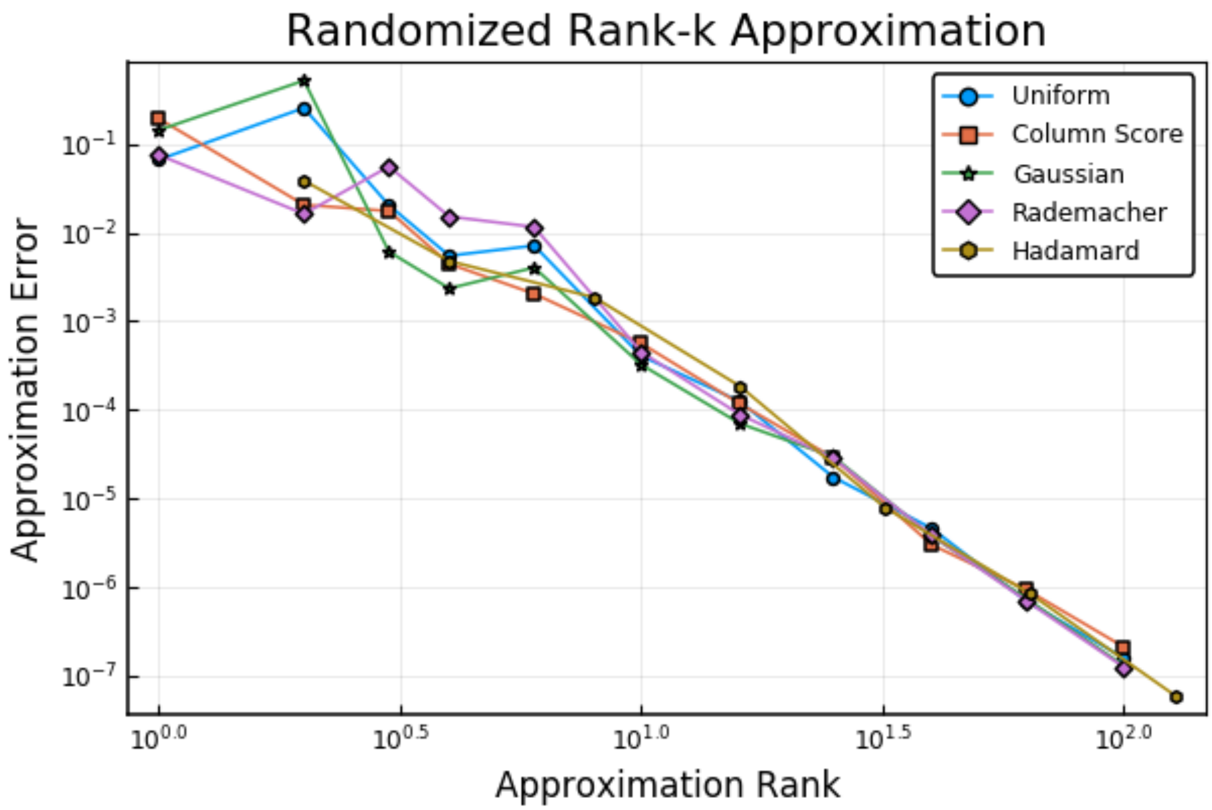
In [257]:
```
function hadamard(d)

    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;
```

In [268]:
```
function rank_k_approximation_hadamard(A, k)

    n_prev = size(A)[2]
    n = 2^ceil(Int, log2(n_prev)) # round n to nearest larger power of
2

    # pad A with zeros
    A = hcat(A, zeros(size(A)[1], n-n_prev))

    k = 2^ceil(Int, log2(k)) # round k to nearest power of 2

    H = hadamard(Int(log2(k)))
    D = diagm([rand([-1, +1]) for _ in 1:k])
    P = spzeros(n, k)
    for i in 1:k
        P[rand([1:n...]), i] = 1
    end
    P *= sqrt(k/n)
    S = 1/sqrt(k)*P*H*D;

    return rank_k_approximation(A, S)

end;
```

In [276]:
```
approx_errors_hadamard = []
ks_hadamard = 2 .^ (1:7);
@showprogress for k in ks_hadamard
    push!(approx_errors_hadamard, rank_k_approximation_hadamard(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:39
```

In [283]:
```
plot(xscale=:log10, yscale=:log10, box=:on, thickness_scaling=1.1)
plot!(ks, approx_errors_uniform,    marker=:auto, label="Uniform")
plot!(ks, approx_errors_column,     marker=:auto, label="Column Score"
)
plot!(ks, approx_errors_gaussian,   marker=:auto, label="Gaussian")
plot!(ks, approx_errors_rademacher, marker=:auto, label="Rademacher")
plot!(ks_hadamard, approx_errors_hadamard, marker=:auto, label="Hadama
rd")
title!("Randomized Rank-k Approximation")
xlabel!("Approximation Rank")
ylabel!("Approximation Error")
savefig("3a.png")
```



Problem 3(b)

In [306]:
```
function exact_svd_error(A, k)

    U, Σ, V = svd(A)
    Σ[k+1:end] .= 0
    Ak = U*diagm(Σ)*V'

    return opnorm(A - Ak)^2

end;
```

In [307]:
```
exact_errors = []
@showprogress for k in ks
    push!(exact_errors, exact_svd_error(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:41
```

In [328]:
```
function randomized_svd(A, S)

    C = A*S
    Q, R = qr(C)
    U, Σ, V = svd(Q'*A)
    Ãk = (Q*U)*diagm(Σ)*V'

    return opnorm(A - Ãk)^2

end;
```

In [329]:
```
function randomized_svd_uniform(A, k)

    n = size(A)[2]

    S = spzeros(n, k)
    for i in 1:k
        S[rand([1:n...]), i] = 1
    end

    return randomized_svd(A, S)

end;
```

In [330]:
```
approx_errors_uniform = []
@showprogress for k in ks
    push!(approx_errors_uniform, randomized_svd_uniform(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:58
```

In [331]:

```
function randomized_svd_column(A, k)

    n = size(A)[2]

    dist = DiscreteNonParametric(1:n, normalize([norm(col, 2)^2 for co
l in eachcol(A)], 1))

    S = spzeros(n, k)
    for i in 1:k
        S[rand(dist), i] = 1
    end

    return randomized_svd(A, S)

end;
```

In [332]:

```
approx_errors_column = []
@showprogress for k in ks
    push!(approx_errors_column, randomized_svd_column(A, k))
end
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:00
:44
```

In [333]:

```
function randomized_svd_gaussian(A, k)

    n = size(A)[2]

    s = [randn() for _ in 1:n*k]
    S = 1/sqrt(k)*reshape(s, (n, k))

    return randomized_svd(A, S)

end;
```

In [334]:

```
approx_errors_gaussian = []
@showprogress for k in ks
    push!(approx_errors_gaussian, randomized_svd_gaussian(A, k))
end
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:00
:36
```

In [335]:
```
function randomized_svd_rademacher(A, k)

    n = size(A)[2]

    s = [rand() >= 0.5 ? 1 : -1 for _ in 1:n*k]
    S = reshape(s, (n, k))

    return randomized_svd(A, S)

end;
```

In [336]:
```
approx_errors_rademacher = []
@showprogress for k in ks
    push!(approx_errors_rademacher, randomized_svd_rademacher(A, k))
end
```

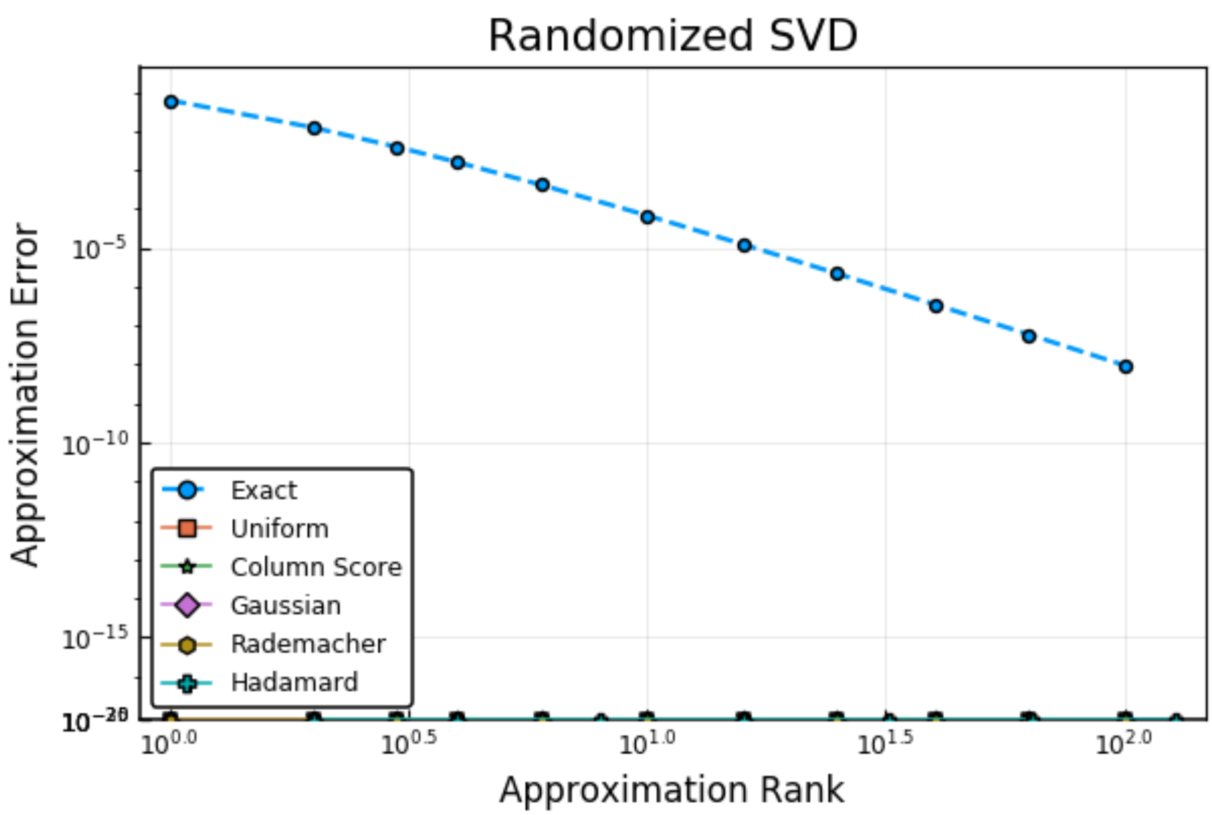
```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:41
```

In [337]:
```
function randomized_svd_hadamard(A, k)

    n_prev = size(A)[2]
    n = 2^ceil(Int, log2(n_prev)) # round n to nearest larger power of
2

    # pad A with zeros
    A = hcat(A, zeros(size(A)[1], n-n_prev))

    k = 2^ceil(Int, log2(k)) # round k to nearest power of 2

    H = hadamard(Int(log2(k)))
    D = diagm([rand([-1, +1]) for _ in 1:k])
    P = spzeros(n, k)
    for i in 1:k
        P[rand([1:n...]), i] = 1
    end
    P *= sqrt(k/n)
    S = 1/sqrt(k)*P*H*D;

    return randomized_svd(A, S)

end;
```

In [338]:
```
approx_errors_hadamard = []
@showprogress for k in ks_hadamard
    push!(approx_errors_hadamard, randomized_svd_hadamard(A, k))
end
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:33
```

In [345]:
```
plot(xscale=:log10, yscale=:log10, box=:on, thickness_scaling=1.1)
plot!(ks, exact_errors,                 marker=:auto, label="Exact", lw=1.
5, ls=:dash)
plot!(ks, approx_errors_uniform,     marker=:auto, label="Uniform")
plot!(ks, approx_errors_column,      marker=:auto, label="Column Score"
)
plot!(ks, approx_errors_gaussian,    marker=:auto, label="Gaussian")
plot!(ks, approx_errors_rademacher, marker=:auto, label="Rademacher")
plot!(ks_hadamard, approx_errors_hadamard, marker=:auto, label="Hadama
rd")
title!("Randomized SVD")
xlabel!("Approximation Rank")
ylabel!("Approximation Error")
savefig("3b.png")
```



Problem 4

Problem 4(a), (b), (c)

In [346]:
```
data = Array(CSV.read("u.data", DataFrame, header=false));
```

In [347]:
```
usermovies = spzeros(943, 1682)

for row in eachrow(Array(data))
    usermovies[row[1], row[2]] = row[3]
end;
```

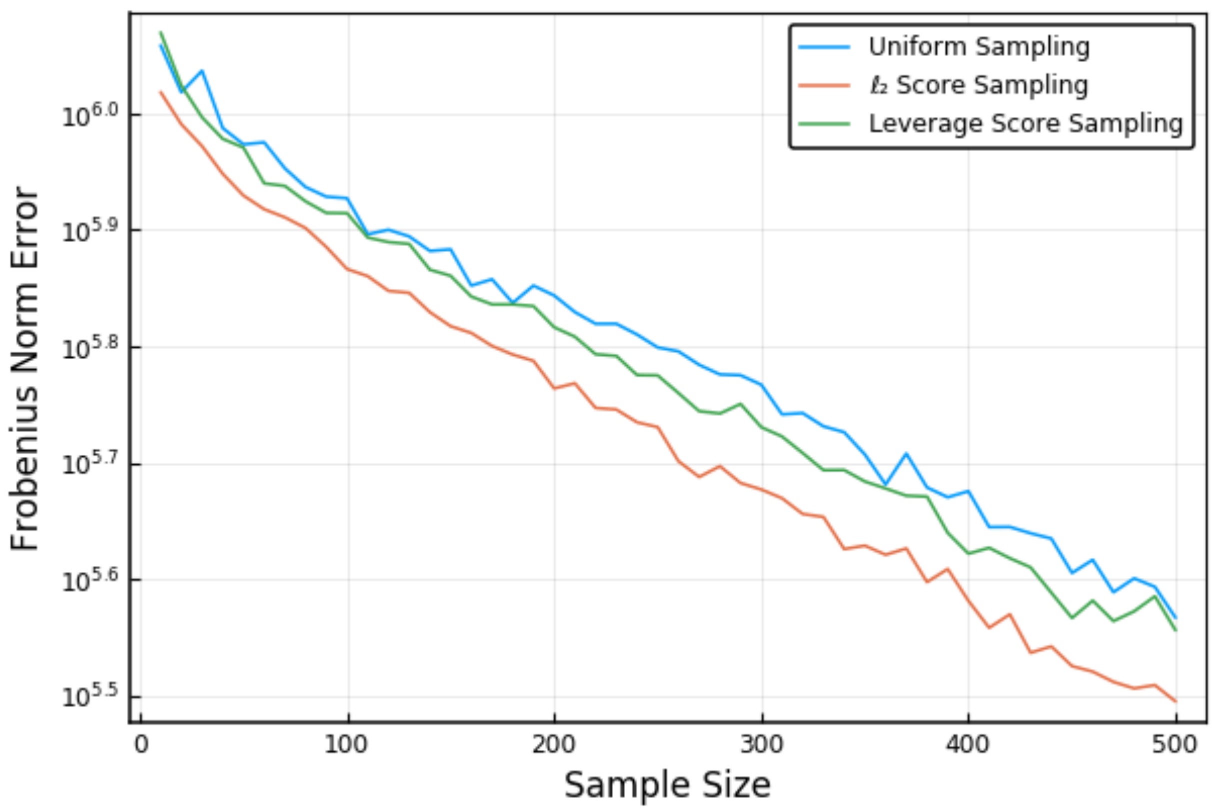
In [348]:
```
function cur_decomposition(A, Ms, row_dist, col_dist)

    approx_frob_error = []
    approx_spec_error = []

    @showprogress for M in Ms

        IS = [rand(row_dist) for i in 1:M]
        JS = [rand(col_dist) for i in 1:M]

        R = A[IS, :]
        C = A[:, JS]

        U = pinv(C)*A*pinv(R)

        Ã = C*U*R

        push!(approx_frob_error, norm(  A - Ã)^2)
        push!(approx_spec_error, opnorm(A - Ã)^2)

    end

    return approx_frob_error, approx_spec_error

end;
```

In [349]:
```
A = Array(usermovies)
m, n = size(A)
Ms = 10:10:500

unif_row_dist = DiscreteNonParametric(1:m, 1/m*ones(m))
unif_col_dist = DiscreteNonParametric(1:n, 1/n*ones(n))

l2_score_row_dist = DiscreteNonParametric(1:m, normalize!([norm(row, 2
)^2 for row in eachrow(A)], 1))
l2_score_col_dist = DiscreteNonParametric(1:n, normalize!([norm(col, 2
)^2 for col in eachcol(A)], 1))

U1, Σ, V = svd(A)
U2, Σ, V = svd(A')

lev_score_row_dist = DiscreteNonParametric(1:m, normalize!([norm(row,
2)^2 for row in eachrow(U1)], 1))
lev_score_col_dist = DiscreteNonParametric(1:n, normalize!([norm(row,
2)^2 for row in eachrow(U2)], 1));
```

In [350]:
```
unif_frob_error,      unif_spec_error      = cur_decomposition(A, Ms, unif_row_dist,      unif_col_dist)
l2_score_frob_error,  l2_score_spec_error  = cur_decomposition(A, Ms, l2_score_row_dist,  l2_score_col_dist)
lev_score_frob_error, lev_score_spec_error = cur_decomposition(A, Ms, lev_score_row_dist, lev_score_col_dist);
```

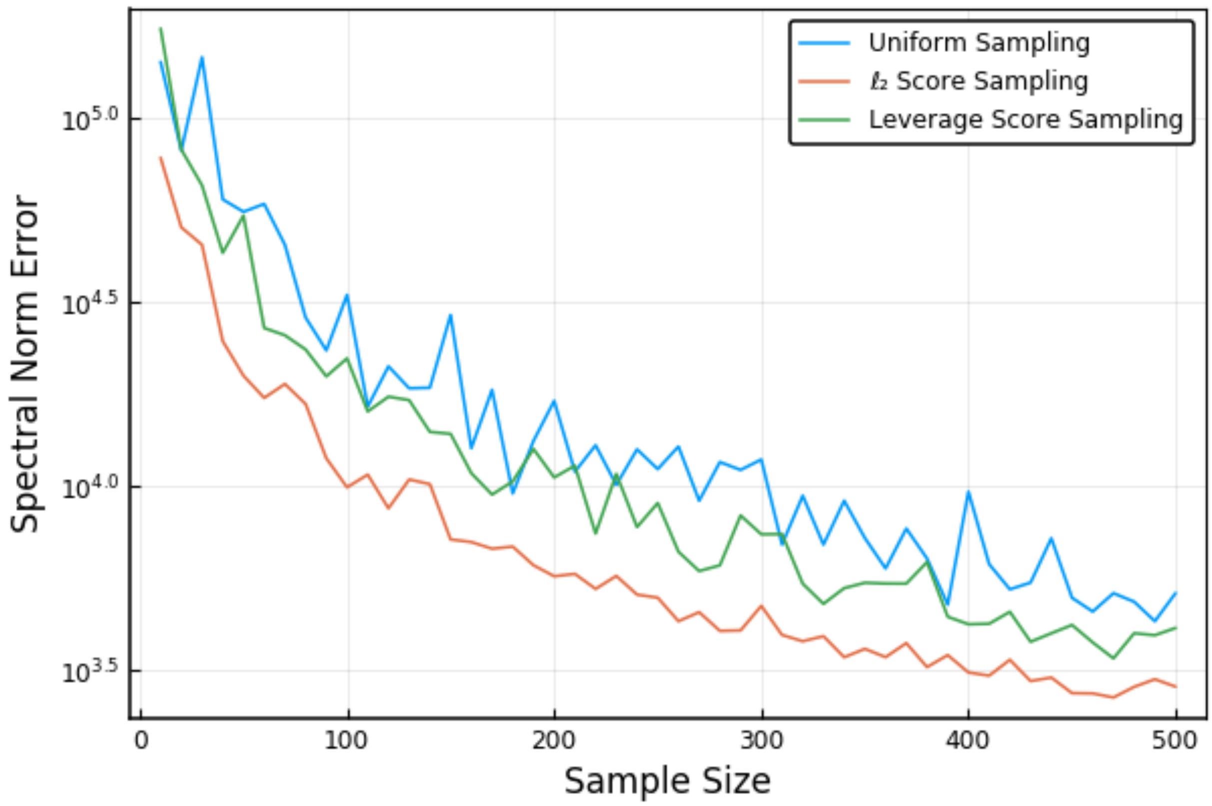
```
Progress: 100%|██████████████████████████████████████████| Time: 0:01:25
Progress: 100%|██████████████████████████████████████████| Time: 0:01:11
Progress: 100%|██████████████████████████████████████████| Time: 0:01:03
```

In [224]:
```
plot(box=:on, yscale=:log10, thickness_scaling=1.1)
plot!(Ms, unif_frob_error,      label="Uniform Sampling")
plot!(Ms, l2_score_frob_error,  label="ℓ₂ Score Sampling")
plot!(Ms, lev_score_frob_error, label="Leverage Score Sampling")
xlabel!("Sample Size")
ylabel!("Frobenius Norm Error")
savefig("4_1.png")
```



In [225]:

```
plot(box=:on, yscale=:log10, thickness_scaling=1.1)
plot!(Ms, unif_spec_error,      label="Uniform Sampling")
plot!(Ms, l2_score_spec_error,  label="ℓ₂ Score Sampling")
plot!(Ms, lev_score_spec_error, label="Leverage Score Sampling")
xlabel!("Sample Size")
ylabel!("Spectral Norm Error")
savefig("4_2.png")
```



In [ ]: