Homework 2

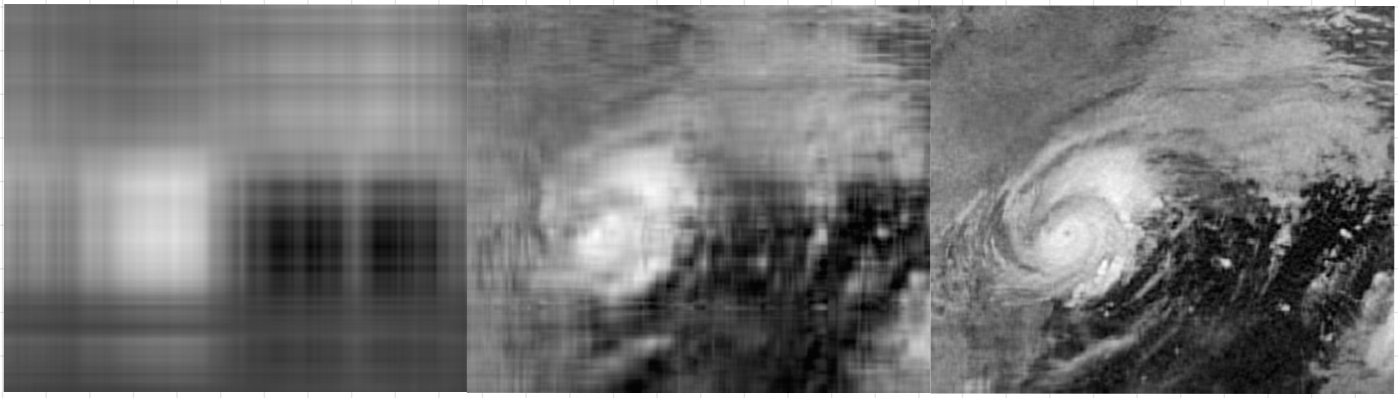
Ross B. Alexander (rbalexan@stanford.edu)

Problem 1 - Singular value decomposition (SVD) for compression

Singular value decomposition (SVD) factorizes an $m \times n$ matrix X as $X = U \Sigma V^T$, where $U \in \mathbb{R}^{m \times m}$, $U^T U = U U^T = I$, $\Sigma \in \mathbb{R}^{m \times n}$ contains non-increasing, non-negative values along its diagonal, and $V \in \mathbb{R}^{n \times n}$, $V^T V = V V^T = I$. Hence, X can be represented as $X = \sum_{i=1}^{d} \sigma_i u_i v_i^T$ where u_i denotes the ith column of U and v_i denotes the ith column of V. Download the Harvey image and load it as the matrix X (in grayscale).

(a) Perform SVD on this matrix and zero out all but the top-k singular values to form an approximation $\tilde{X}$. Specifically, compute $\tilde{X} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$, display the resulting approximation as an image, and report $\frac{\|X - \tilde{X}\|_F}{\|X\|_F}$ for $k \in \{2, 10, 40\}$.

$k = 2$ | $k = 10$ | $k = 40$



$\frac{\|X - \tilde{X}\|_F}{\|X\|_F} \approx 0.2815$ | $\frac{\|X - \tilde{X}\|_F}{\|X\|_F} \approx 0.1588$ | $\frac{\|X - \tilde{X}\|_F}{\|X\|_F} \approx 0.0837$

(b) How many numbers do you need to describe the approximation $\tilde{X} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ for $k \in \{2, 10, 40\}$?

For each outer product, we need 1 term for the singular value σ_i, m terms for the column vector of U (u_i), and n terms for the column vector of V (v_i). In total, the approximation using k outer products requires storing $k(1+m+n)$ numbers.

For our image, which is 1296×1548 ($m \times n$) we require:

- $k = 2 \longrightarrow$ 5690 numbers
- $k = 10 \longrightarrow$ 28450 numbers
- $k = 40 \longrightarrow$ 113800 numbers

SEE ATTACHED CODE

Problem 2 - Singular value decomposition (SVD) and subspaces

Let A be an $m \times n$ singular matrix of rank r with SVD

$$A = U\Sigma V^T = \begin{pmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix} \begin{pmatrix} — & v_1^T & — \\ — & v_2^T & — \\ & \vdots & \\ — & v_n^T & — \end{pmatrix}$$

$$= \begin{pmatrix} \hat{U} & \tilde{U} \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix} \begin{pmatrix} \hat{V}^T \\ \tilde{V}^T \end{pmatrix}$$

where $\sigma_1 \geq \cdots \geq \sigma_r > 0$, $\hat{U}$ consists of the first r columns of U, $\tilde{U}$ consists of the remaining $m-r$ columns of U, $\hat{V}$ consists of the first r columns of V, and $\tilde{V}$ consists of the remaing $n-r$ columns of V. Give bases for the spaces range(A), null(A), range(A^T), and null(A^T) in terms of the components of the SVD of A and give a brief justification.

- the first r columns of U are a basis for range(A)
  - since $A = U\Sigma V^T$, nonzeros in first r elements on the diagonal of Σ induce a space spanned by first r columns of U
- the last $m-r$ columns of U are a basis for null(A)
  - since $A = U\Sigma V^T$, zeros in last $\min(m,n) - r$ elements on the diagonal of Σ induce a nullspace spanned by the last $m-r$ columns of U
- the first r columns of V are a basis for range(A^T)
  - since $A^T = (U\Sigma V^T)^T = V\Sigma U^T$; nonzeros in first r elements on the diagonal of Σ induce a space spanned by first r columns of r
- the last $n-r$ columns of V are a basis for null(A^T)
  - since $A^T = (U\Sigma V^T)^T = V\Sigma U^T$; zeros in last $\min(m,n) - r$ elements on the diagonal of Σ induce a nullspace spanned by the last $n-r$ columns of V.

$$A = U\Sigma V^T = \sum_{i=1}^{\min(m,n)} \sigma_i u_i v_i^T$$

$$= \sum_{i=1}^{r} \sigma_i u_i v_i^T + \sum_{i=r+1}^{\min(m,n)} 0 \, u_i v_i^T$$

Problem 3 - Least-squares

Consider the least-squares problem $\min_x \|Ax-b\|_2$. Which of the following statements are true?

(a) If x is a solution to the least-squares problem, then $Ax=b$.

False, take $A=\begin{bmatrix}0\\0\\1\end{bmatrix}$ $b=\begin{bmatrix}-1\\1\\2\end{bmatrix}$. This leads to the solution $x=2$, however $Ax=\begin{bmatrix}0\\0\\2\end{bmatrix}\neq\begin{bmatrix}-1\\1\\2\end{bmatrix}=b$.

(b) If x is a solution to the least-squares problem, then the residual vector $r=b-Ax$ is in the nullspace of A^T.

True. Using the definition of nullspace

$$\begin{aligned}0 &= A^T r\\ &= A^T(b-Ax)\\ &= A^Tb - A^TAx\\ A^TAx &= A^Tb\end{aligned}$$

which holds regardless of the rank of A (can use inv.^{-1} or $\text{p.inv.}^{\dagger}$)

(c) The solution is unique.

False, let $A=[0]$, $b=[0]$, then any $x\in\mathbb{R}$ is a solution.

(d) A solution may not exist.

False, a solution always exists (either using inv.^{-1} or $\text{p.inv.}^{\dagger}$).

(e) None of the above.

False.

Problem 4 - Binary classification via regression

Download the MNIST dataset. In binary classification, we restrict Y to take on only two values. Suppose $Y \in \{0,1\}$. Now let us use least-squares regression for classification. Let us consider the classification problem of recognizing if a digit is 2 or not using linear regression. Here, let $Y=1$ for all the 2 digits in the training set and $Y=0$ for all other digits.

Build a linear classifier by solving:

$$\underset{w}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}(y_i - w_i^T x)^2 + \lambda \|w\|_2^2$$

using the training set $\{(x_1,y_1),\ldots,(x_N,y_N)\}$. Use appropriate regularization ($\lambda > 0$) as needed. Let x_i be a vector of the pixel values of the image. For the purpose of classification, we can label a digit as 2 if $w^T z$ is larger than some threshold.

(a) Based on your training set, choose a reasonable threshold for classification. On the training set, what is the 0/1 loss? What is the square loss?

With a regularization parameter of $\lambda = 1\times10^{1}$, we choose a threshold of 0.1.
On the training set we achieve:

- 0/1 loss: 4637
- Squared loss: 7.728×10^{-2}

(b) On the test set, what is the 0/1 loss? What is the square loss?

With a regularization parameter of $\lambda = 1\times10^{1}$, we choose a threshold of 0.1.
On the testing set we achieve:

- 0/1 loss: 819
- Squared loss: 8.190×10^{-2}

SEE ATTACHED CODE

Problem 5 - Leave-one-out cross-validation (LOOCV)

Assume that there are n training examples $\{(x_i, y_i)\}_{i=1}^n$, where each input data point x_i has d real-valued features. The goal of regression is to learn to predict y_i from x_i. The linear regression model assumes that the output y is a linear combination of the input features plus Gaussian noise with weights given by w.

The maximum likelihood estimate of the model parameters w (which also happens to minimize the sum of squared prediction errors) is given by the normal equations we saw in class: $(X^TX)^{-1}X^Ty$. Here, the rows of the matrix X are the training examples $x_1^T, \ldots, x_n^T$. Define $\hat{y}$ to be the vector of predictions using $\hat{w}$ if we were to plug in the original training set X:

$$\begin{aligned}\hat{y} &= X\hat{w} \\ &= X(X^TX)^{-1}X^Ty \\ &= Hy\end{aligned}$$

where we define $H = X(X^TX)^{-1}X^T$. Show that H is a projection matrix.

As mentioned above, $\hat{w}$ also minimizes the sum of squared errors (SSE).

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Leave-one-out cross-validation (LOOCV) score is defined as:

$$LOOCV = \sum_{i=1}^n (y_i - \hat{y}_i^{(-i)})^2$$

where $\hat{y}^{(-i)}$ is the estimator of y after removing the i^{th} observation from the training set, i.e. it minimizes

$$\sum_{j \neq i} (y_j - \hat{y}_j^{(-i)})^2$$

(a) What is the time complexity of computing the LOOCV score naively? (The naive algorithm is to loop through each point, performing a regression on the $n-1$ remaining points at each iteration.) Hint: the complexity of inverting a $k \times k$ matrix is $O(k^3)$ using classical methods.

The time complexity is $O(\underbrace{n}_{CV}(\underbrace{n-1}_{inversion})^3) \rightarrow O(n^4)$.

(b) Write $\hat{y}_i$ in terms of H and y.

$$\begin{aligned}\hat{y}_i &= (Hy)_i \\ &= \sum_{j=1}^n H_{ij}y_j \\ \hat{y}_i &= h_i^T y \quad \text{where } h_i \text{ is the } i\text{th row of } H\end{aligned}$$

(c) Show that $\hat{y}^{(-i)}$ is also the estimator which minimizes SSE for z when

$$z_j = \begin{cases} y_j & j \neq i \\ \hat{y}_i^{(-i)} & j = i \end{cases}$$

- the loss function is

$$\mathcal{L}_z = \|z - Xw\|_2^2$$
$$= \sum_{j=1}^{n} (z_j - \hat{y}_j)^2$$
$$= \sum_{j \neq i} (y_j - \hat{y}_j)^2 + (\hat{y}_i^{(-i)} - \hat{y}_i)$$

($\hat{y}_i^{(-i)} - \hat{y}_i$: constant since y_i isn't seen in $\hat{y}^{(-i)}$ training)

$$\mathcal{L}_z = \mathcal{L}^{(-i)}$$

- since the LOOCV loss function and z loss function are equivalent, they lead to the same predictor $\hat{y}^{(-i)}$

(d) Write $\hat{y}_i^{(-i)}$ in terms of H and z. By definition, $\hat{y}_i^{(-i)} = z_i$, but give an answer that is analogous to (b).

$$\hat{y}_i^{(-i)} = (Hz)_i$$
$$= \sum_{j=1}^{n} H_{ij} z_j$$

$\hat{y}_i^{(-i)} = h_i^T z$ where h_i is the ith row of H

(e) Show that $\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii} y_i - H_{ii} \hat{y}_i^{(-i)}$.

- using our previous results

$$\hat{y}_i - \hat{y}_i^{(-i)} = h_i^T y - h_i^T z$$
$$= h_i^T (y - z)$$
$$= \sum_{j=1}^{n} H_{ij} (y_j - z_j)$$
$$= \sum_{j \neq i} H_{ij} (y_j - y_j) + H_{ii} (y_i - \hat{y}_i^{(-i)})$$

($y_j - y_j = 0$)

$$\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii} (y_i - \hat{y}_i^{(-i)}) \quad \blacksquare$$

(f) Show that

$$\text{LOOCV} = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - H_{ii}} \right)^2.$$

What is the algorithmic complexity of computing the LOOCV score using this formula? Note: We see from this formula that the diagonal elements of H somehow indicate the impact that each particular observation has on the result of the regression.

- we manipulate the previous result to give

$$\begin{aligned}
\hat{y}_i - \hat{y}_i^{(-i)} &= H_{ii} y_i - H_{ii} \hat{y}_i^{(-i)} \\
\hat{y}_i &= H_{ii} y_i - (H_{ii} - 1) \hat{y}_i^{(-i)} \\
\hat{y}_i - y_i &= (H_{ii} - 1) y_i - (H_{ii} - 1) \hat{y}_i^{(-i)} \\
\hat{y}_i - y_i &= (H_{ii} - 1)(y_i - \hat{y}_i^{(-i)})
\end{aligned}$$

$$\therefore \quad y_i - \hat{y}_i^{(-i)} = \frac{\hat{y}_i - y_i}{H_{ii} - 1} = \frac{y_i - \hat{y}_i}{1 - H_{ii}}$$

- by definition, we have

$$\text{LOOCV} = \sum_{i=1}^{n} (y_i - \hat{y}_i^{(-i)})^2$$

- substituting in our result, we achieve

$$\text{LOOCV} = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - H_{ii}} \right)^2 \blacksquare$$

The complexity is $O(n^3)$ since we need to compute $\hat{y}$ from a standard normal equations matrix inversion. (Computing H is also $O(n^3)$.) This is still less than the naive approach which has complexity $O(n^4)$.

Problem 6 - Sketching least squares

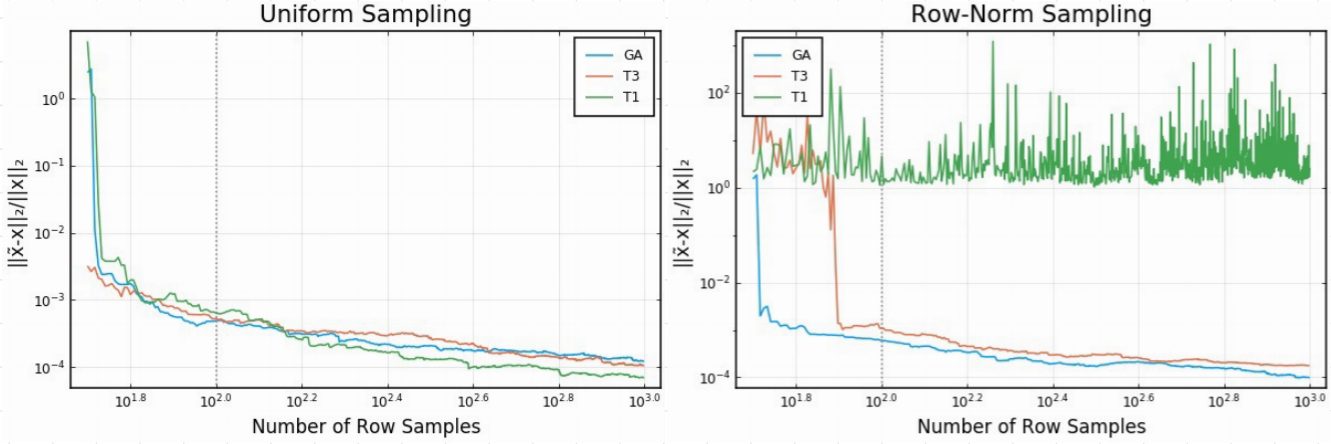
Here, we will consider the empirical performance of random sampling and random projection algorithms for approximating least-squares.

Let $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$, $n \gg d$, and consider approximating the solution to $\min_x \|Ax-b\|_2$. Generate the matrices A from one of the three different classes of distributions introduced below.

GA data: $A \sim \mathcal{N}(1_d, \Sigma)$ where $\Sigma_{ij} = 2 \cdot 0.5^{|i-j|}$
T3 data: $A \sim t(\nu=3, \Sigma)$ where $\Sigma_{ij} = 2 \cdot 0.5^{|i-j|}$
T1 data: $A \sim t(\nu=1, \Sigma)$ where $\Sigma_{ij} = 2 \cdot 0.5^{|i-j|}$

To start, consider matrices of size $n \times d = 500 \times 50$.

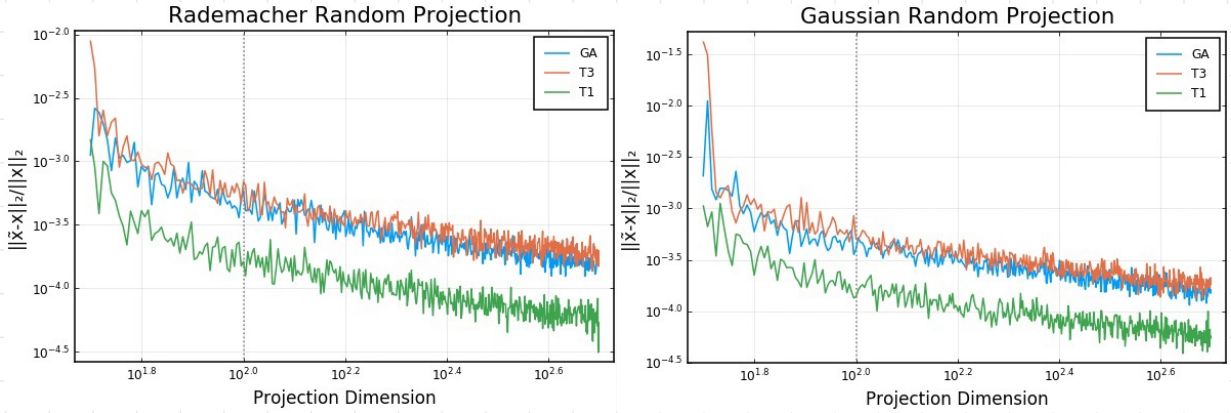
(a) First, for each matrix, consider approximating the solution by randomly sampling a small number r of rows/elements in one of three ways: uniformly at random; according to an importance sampling distribution that is proportional to the Euclidean norms squared of the rows of A; and according to an importance sampling distribution that is proportional to the leverage scores of A. In each case, plot the error as a function of the number r of samples, paying particular attention to the two regimes $r \in \{d, d+1, \cdots, 2d\}$ and $r \in \{2d, 3d, \ldots\}$. Show that the behavior of these 3 procedures is most similar for GA data, intermediate for T3 data, and most different for T1 data; and explain why, and explain similarities and differences.



We observe the expected trends - the GA data performs similarly in both sampling methods since the row scores are approximately uniform. We also see that the row score sampling methods perform worse which is likely due to "outlier" samples being selected too often, when, in reality, each of the samples is given equal weight in standard least-squares. With the row sampling method, the GA data converges quickly, the T3 data has delayed convergence, and the T1 data does not converge!

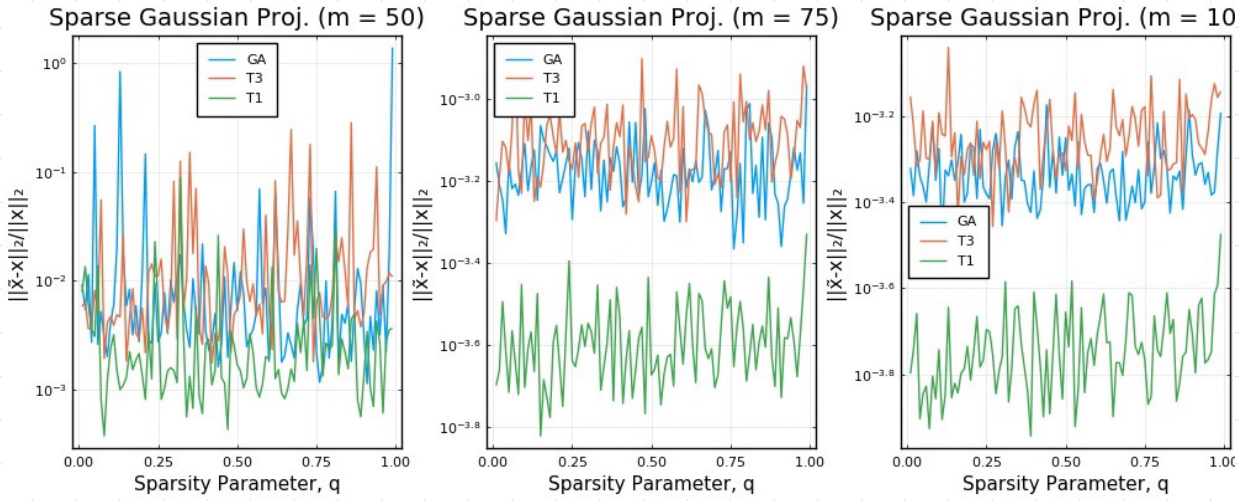
SEE ATTACHED CODE

(b) Next, for each matrix, consider approximating the solution by randomly projecting rows/elements in one of two ways: a random projection matrix, where each entry is i.i.d. $\{\pm 1\}$, with appropriate variance; and a random projection matrix, in which entry is i.i.d. Gaussian, with appropriate variance. In each case, plot the error as a function of the number samples, i.e., dimensions on which the data are projected, paying particular attention to the two regimes $r \in \{d, d+1, \ldots, 2d\}$ and $r \in \{2d, 3d, \ldots\}$. Describe and explain similarities and differences between these two procedures for GA data, T3 data, and T1 data.



Both projection methods appear largely similar, with GA and T3 data converging to higher normalized error than the T1 data.

c)



These results appear mostly similar, but when m=50, the results are similar with high error. This error decreases with increasing projection dim.

Problem 1

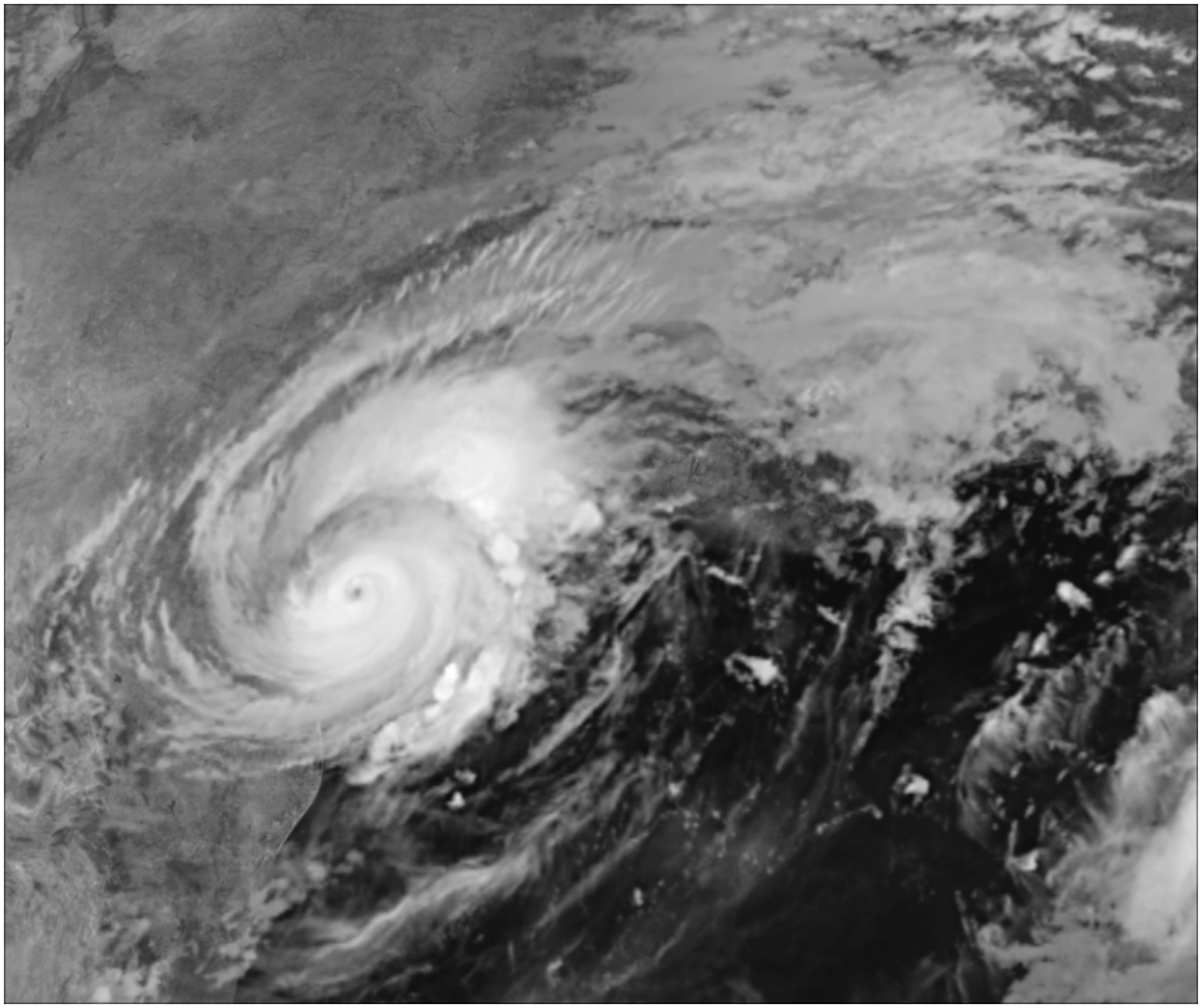
In [1]:
```
using Images
using Plots; pyplot()
using LinearAlgebra
```

In [2]:
```
img_path = "harvey-saturday-goes7am.jpg"
img = load(img_path)
size(img)
```

Out[2]:
```
(1296, 1548)
```

In [3]:
```
X = Gray.(img)
```

Out[3]:



In [4]:
```
X = Float64.(X);
```

In [5]:
```
function svd_compress(X, k)

    U, σ, V = svd(X, full=true); # slow to redo SVD, but not a big deal for homework
    X̃       = sum(σ[i] * U[:, i] * V[:, i]' for i in 1:k)
    err     = norm(X - X̃)/norm(X)

    return X̃, err

end
```

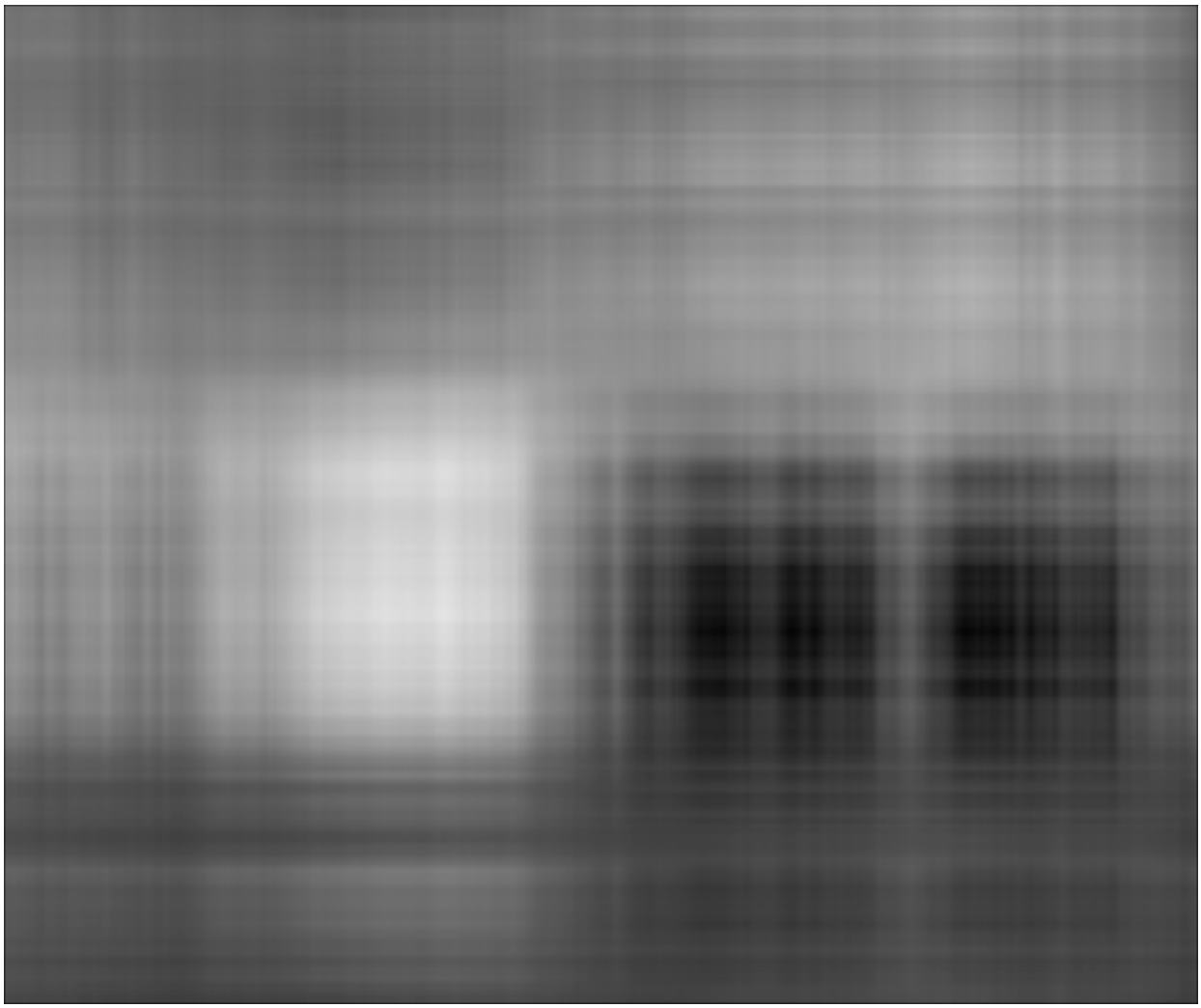
Out[5]:
```
svd_compress (generic function with 1 method)
```

Problem 1a | $k = 2$

In [6]:
```
X̃, err = svd_compress(X, 2)
print(err)
#save("svd_k2.png", Gray.(X̃))
Gray.(X̃)
```

```
0.2815103158754801
```

Out[6]:

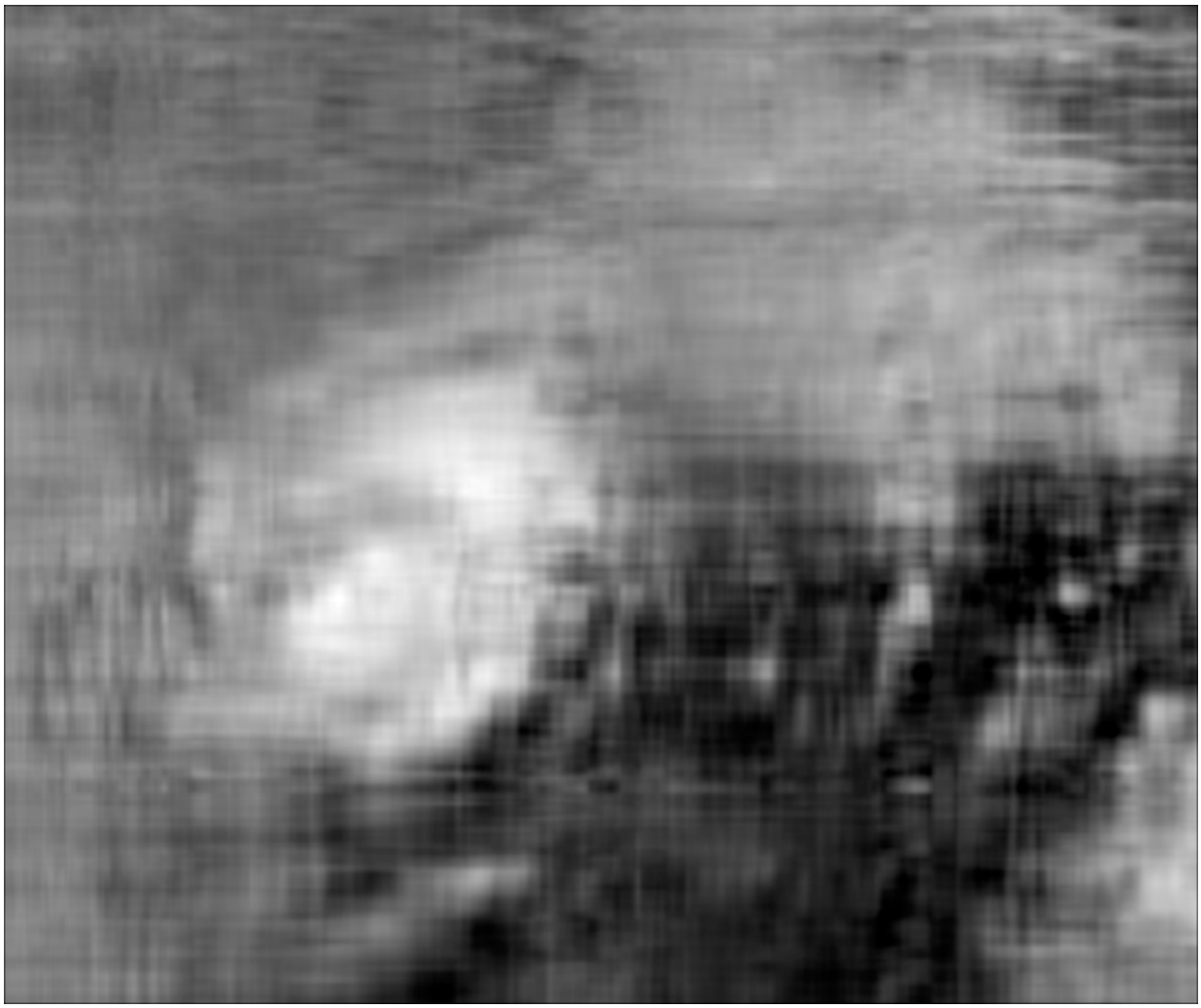


Problem 1a | $k = 10$

In [7]:
```
X̃, err = svd_compress(X, 10)
print(err)
#save("svd_k10.png", Gray.(X̃))
Gray.(X̃)
```

```
0.15876586555633745
```

Out[7]:

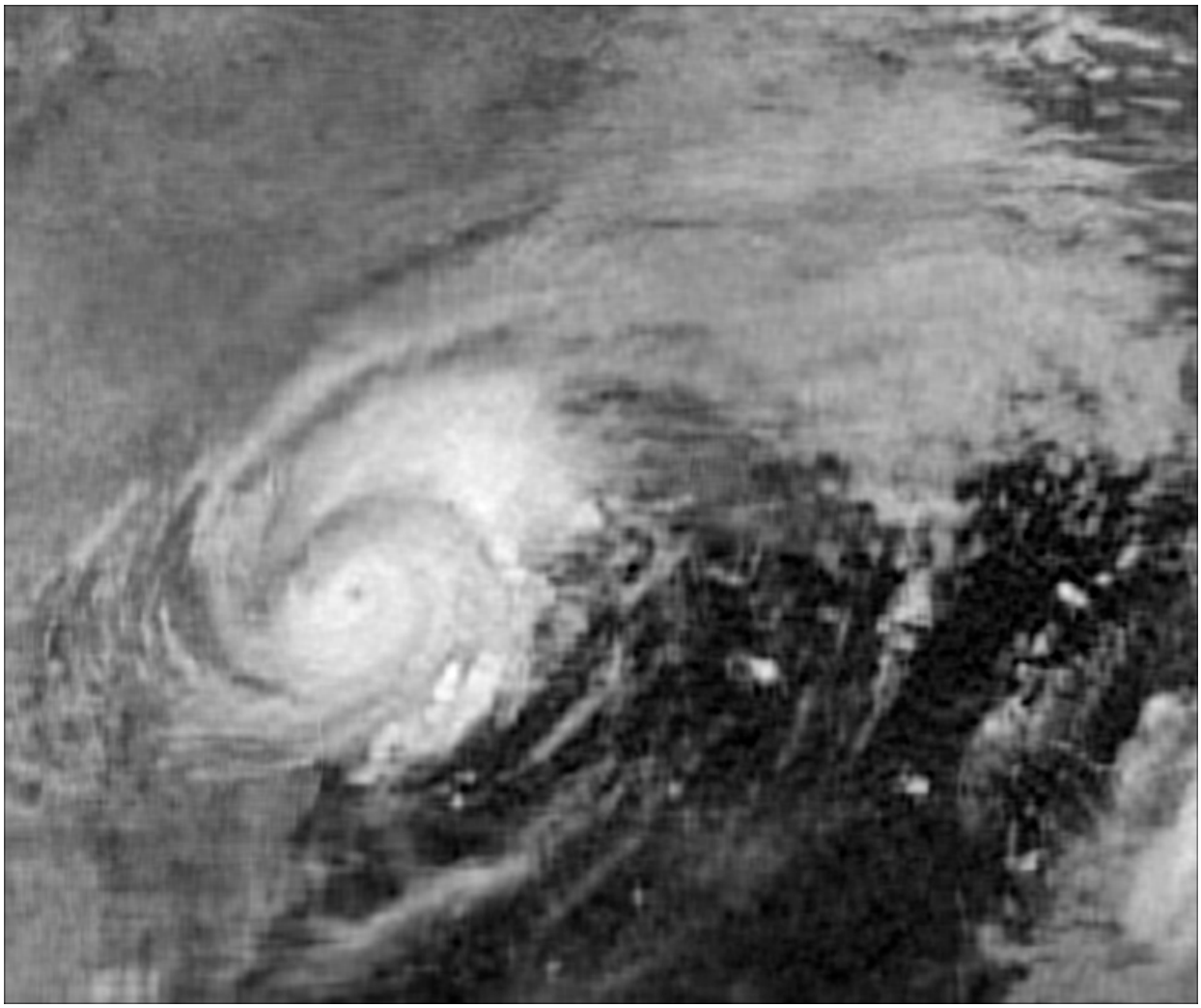


Problem 1a | $k = 40$

In [8]:
```
X̃, err = svd_compress(X, 40)
print(err)
#save("svd_k40.png", Gray.(X̃))
Gray.(X̃)
```

```
0.08368458635729387
```

Out[8]:



Problem 4

In [9]:
```
using MAT
using Statistics
```

In [10]:
```
vars = matread("mnist_all.mat");
```

In [11]:
```
test_3_9 = vars["test3"][9,:]
img_as_mat(example) = permutedims(reshape(example, (28, 28)), [2, 1])
Gray.(img_as_mat(test_3_9)/255)
```

Out[11]:

In [12]:
```
function get_dataset(vars, classifier_digit)

    X_train = zeros()
    y_train = zeros()

    X_test  = zeros()
    y_test  = zeros()

    for digit in 0:9

        label = digit == classifier_digit ? 1 : 0

        X_train = digit == 0 ? vars["train$(digit)"] : vcat(X_train, v
ars["train$(digit)"])
        y_train = digit == 0 ? label*ones(size(vars["train$(digit)"])[
1], 1) : vcat(y_train, label*ones(size(vars["train$(digit)"])[1], 1))

        X_test  = digit == 0 ? vars["test$(digit)"]  : vcat(X_test,  v
ars["test$(digit)"])
        y_test  = digit == 0 ? label*ones(size(vars["test$(digit)"])[1
], 1)  : vcat(y_test,  label*ones(size(vars["test$(digit)"])[1], 1))

    end

    X_train = Float64.(X_train)
    y_train = Float64.(y_train)

    X_test  = Float64.(X_test)
    y_test  = Float64.(y_test);

    return X_train, y_train, X_test, y_test

end
```

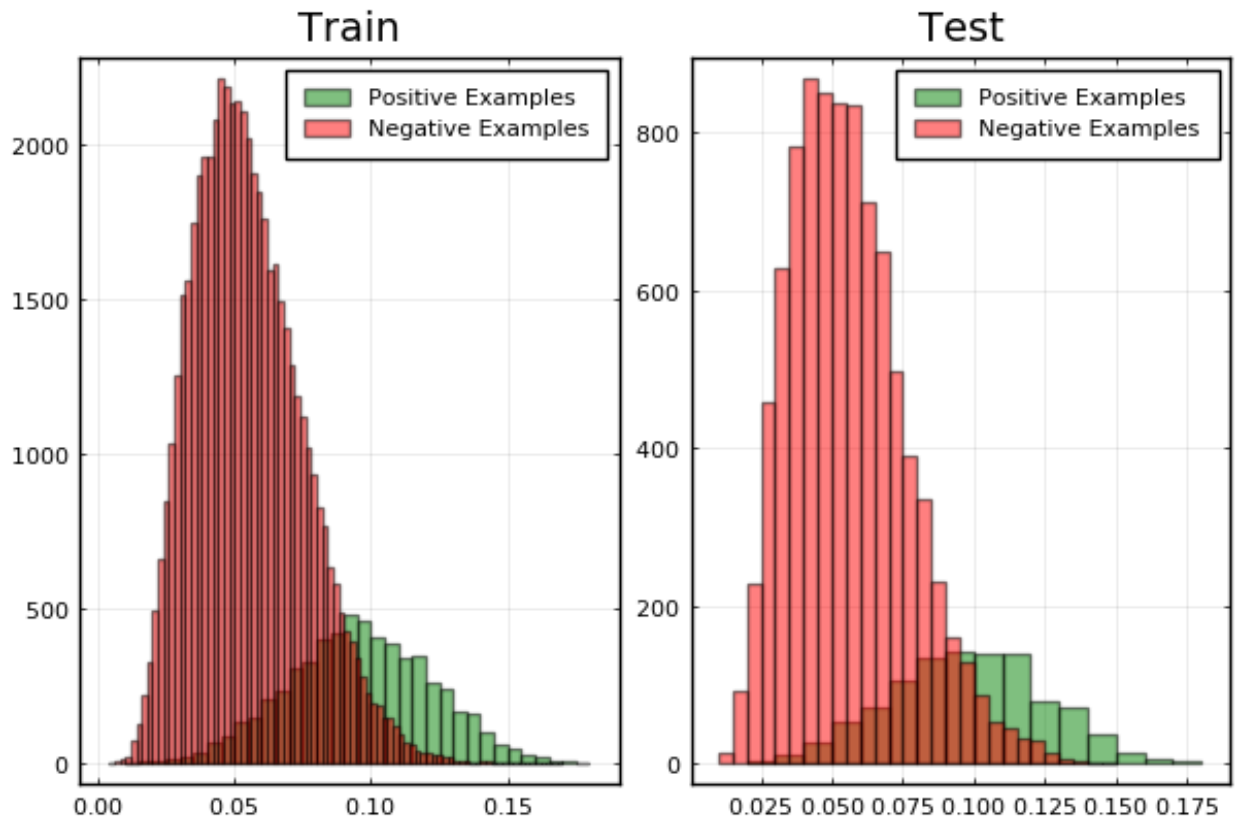
Out[12]:
```
get_dataset (generic function with 1 method)
```

In [13]:
```
binarize(vec, threshold) = [v >= threshold ? 1 : 0 for v in vec]
zero_one_loss(ŷ, y) = sum(  ŷ .≠  y)
accuracy(     ŷ, y) = mean( ŷ .== y)
squared_loss( ŷ, y) = mean((ŷ .-  y).^2)
```

Out[13]:
```
squared_loss (generic function with 1 method)
```

In [14]:
```
digit = 2

X_train, y_train, X_test, y_test = get_dataset(vars, digit)

λ = 1E11

w = inv(X_train'*X_train + λ*I)*X_train'*y_train

ŷ_train = X_train * w
ŷ_test  = X_test  * w

p1 = histogram( ŷ_train[y_train .== 1], alpha=0.5, title="Train", label="Positive Examples", c=:green, box=:on)
     histogram!(ŷ_train[y_train .== 0], alpha=0.5, label="Negative Examples", c=:red)

p2 = histogram( ŷ_test[ y_test  .== 1], alpha=0.5, title="Test", label="Positive Examples", c=:green, box=:on)
     histogram!(ŷ_test[ y_test  .== 0], alpha=0.5, label="Negative Examples", c=:red)

plot(p1, p2)
```

Out[14]:



In [15]:
```
w = abs.(w)
w = (w .- minimum(w)) / (maximum(w) - minimum(w))
attention_map = Gray.(permutedims(reshape(w, (28, 28)), [2, 1]))
```

Out[15]:

In [16]:
```
threshold = 0.1

ŷ_train_b = binarize(ŷ_train, threshold)
ŷ_test_b  = binarize(ŷ_test,  threshold)

@show train_zero_one_loss = zero_one_loss(ŷ_train_b, y_train)
@show train_squared_loss  = squared_loss( ŷ_train_b, y_train)
@show train_accuracy      = accuracy(     ŷ_train_b, y_train)

@show test_zero_one_loss  = zero_one_loss(ŷ_test_b,  y_test)
@show test_squared_loss   = squared_loss( ŷ_test_b,  y_test)
@show test_accuracy       = accuracy(     ŷ_test_b,  y_test);
```

```
train_zero_one_loss = zero_one_loss(ŷ_train_b, y_train) = 4637
train_squared_loss = squared_loss(ŷ_train_b, y_train) = 0.0772833333
3333333
train_accuracy = accuracy(ŷ_train_b, y_train) = 0.9227166666666666
test_zero_one_loss = zero_one_loss(ŷ_test_b, y_test) = 819
test_squared_loss = squared_loss(ŷ_test_b, y_test) = 0.0819
test_accuracy = accuracy(ŷ_test_b, y_test) = 0.9181
```

Problem 6

In [17]:
```
using Distributions
```

In [18]:
```
function rand_matrix_from_row_dist(dist, n, d)

    A = zeros(n, d)

    for i in 1:n
        A[i, :] = rand(dist)
    end

    return A

end
```

Out[18]:
```
rand_matrix_from_row_dist (generic function with 1 method)
```

In [19]:
```
n = 500
d = 50

μ = ones(d)
Σ = [2*0.5^abs(i-j) for i in 1:d, j in 1:d]

μt = zeros(d)

GA = MvNormal(μ, Σ)
T1 = MvTDist(1, μt, Σ)
T3 = MvTDist(3, μt, Σ)

A_GA = rand_matrix_from_row_dist(GA, n, d)
A_T3 = rand_matrix_from_row_dist(T3, n, d)
A_T1 = rand_matrix_from_row_dist(T1, n, d)

x_star = randn(d)

b_GA = A_GA * x_star + 0.01*rand(n)
b_T3 = A_T3 * x_star + 0.01*rand(n)
b_T1 = A_T1 * x_star + 0.01*rand(n);
```

Problem 6(a)

In [20]:
```
function uniform_distribution(A)

    n    = size(A)[1]
    dist = DiscreteNonParametric([i for i in 1:n], [1/n for i in 1:n])

end

function row_score_distribution(A)

    row_scores = [norm(row, 2)^2 for row in eachrow(A)]
    probs      = normalize(row_scores, 1)

    dist = DiscreteNonParametric([i for i in 1:size(A)[1]], probs)

end
```

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

In [21]:
```
unif_dist_GA = uniform_distribution(A_GA)
unif_dist_T3 = uniform_distribution(A_T3)
unif_dist_T1 = uniform_distribution(A_T1)

rs_dist_GA   = row_score_distribution(A_GA)
rs_dist_T3   = row_score_distribution(A_T3)
rs_dist_T1   = row_score_distribution(A_T1);
```

In [22]:
```
using ProgressMeter
```

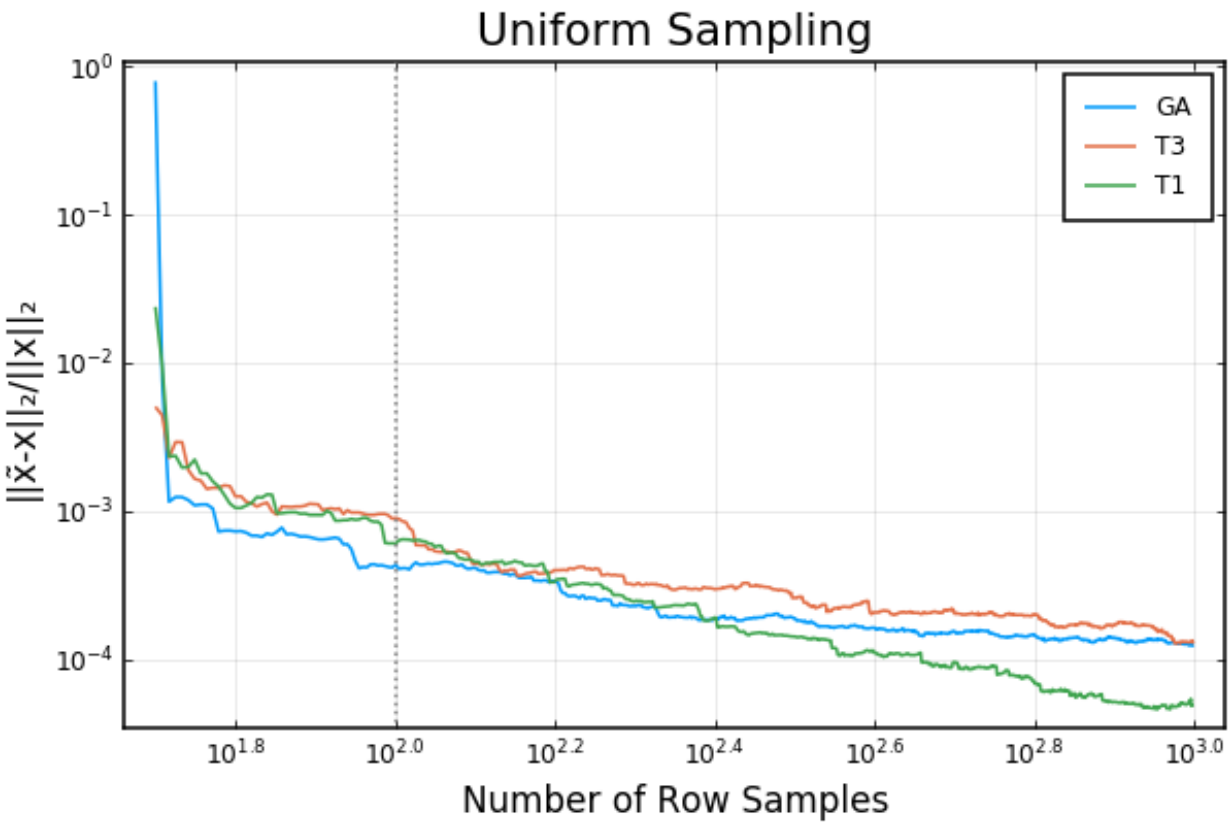
In [23]:
```
function random_sampling_approximate_least_squares(A, b, dist, r_max)

    d = size(A)[2]
    x = inv(A'*A)*A'*b
    errors = []

    Ã = zeros(1, d)
    b̃ = zeros(1, 1)

    @showprogress for k in 1:r_max

        ik = rand(dist)

        if k == 1
            Ã[k, :] = A[ik, :]
            b̃[k]    = b[ik]
        else
            Ã = vcat(Ã, reshape(A[ik, :], (1, d)))
            b̃ = vcat(b̃, b[ik])
        end

        if k ≥ d
            x̃ = inv(Ã'*Ã)*Ã'*b̃
            push!(errors, norm(x̃ .- x, 2) / norm(x, 2))
        end

    end

    return errors

end
```

Out[23]:
```
random_sampling_approximate_least_squares (generic function with 1 m
ethod)
```

In [24]:
```
r_max = 20d

unif_errs_GA = random_sampling_approximate_least_squares(A_GA, b_GA, u
nif_dist_GA, r_max)
unif_errs_T3 = random_sampling_approximate_least_squares(A_T3, b_T3, u
nif_dist_T3, r_max)
unif_errs_T1 = random_sampling_approximate_least_squares(A_T1, b_T1, u
nif_dist_T1, r_max)

plot( d:r_max, unif_errs_GA, label="GA", xscale=:log10, yscale=:log10,
box=:on, thickness_scaling=1.1,
        title="Uniform Sampling", xlabel="Number of Row Samples", ylab
el="||x̃-x||₂/||x||₂")
plot!(d:r_max, unif_errs_T3, label="T3")
plot!(d:r_max, unif_errs_T1, label="T1")
vline!([2d], c=:gray, ls=:dot, label=:none)

#png("als_error_unif_sampling.png")
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:01
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:01
```

Out[24]:

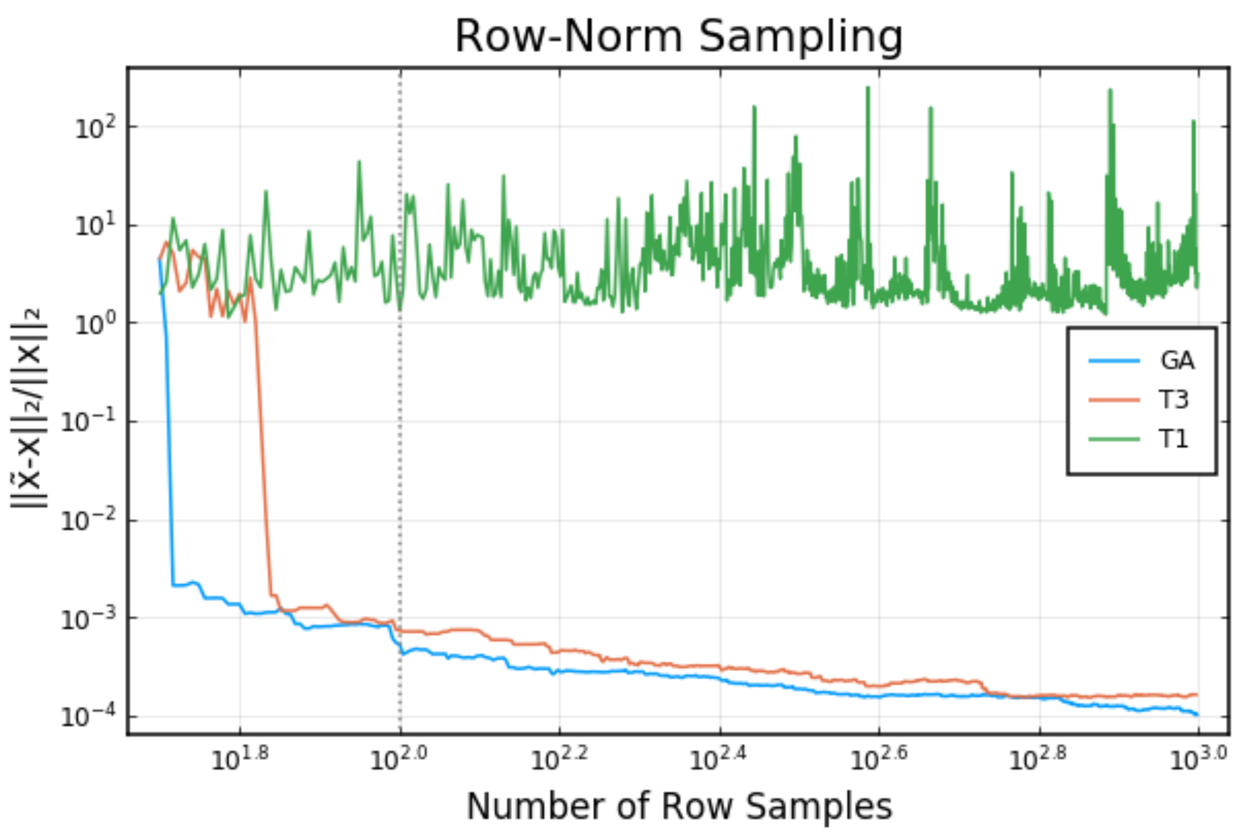


In [25]:

```
r_max = 20d

rs_errs_GA = random_sampling_approximate_least_squares(A_GA, b_GA, rs_dist_GA, r_max)
rs_errs_T3 = random_sampling_approximate_least_squares(A_T3, b_T3, rs_dist_T3, r_max)
rs_errs_T1 = random_sampling_approximate_least_squares(A_T1, b_T1, rs_dist_T1, r_max)

plot( d:r_max, rs_errs_GA, label="GA", xscale=:log10, yscale=:log10, box=:on, thickness_scaling=1.1,
        title="Row-Norm Sampling", xlabel="Number of Row Samples", ylabel="||x̃-x||₂/||x||₂")
plot!(d:r_max, rs_errs_T3, label="T3")
plot!(d:r_max, rs_errs_T1, label="T1")
vline!([2d], c=:gray, ls=:dot, label=:none)

#png("als_error_rs_sampling.png")
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00:01
Progress: 100%|████████████████████████████████████████| Time: 0:00:00
Progress: 100%|████████████████████████████████████████| Time: 0:00:00
```

Out[25]:



Problem 6(b)

In [26]:
```
function rademacher_random_projection_matrix(m, d)

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

    return S

end

function gaussian_random_projection_matrix(m, d)

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

    return S

end
```

Out[26]:
```
gaussian_random_projection_matrix (generic function with 1 method)
```

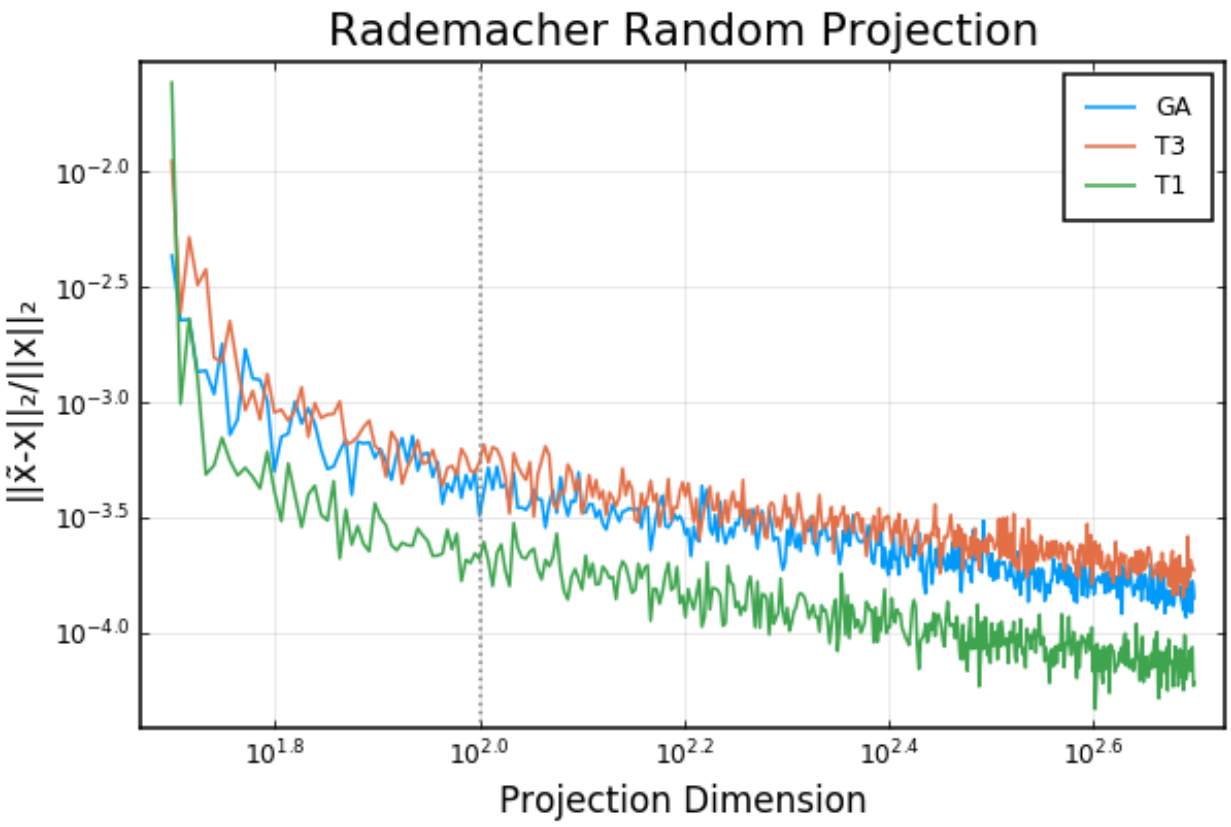
In [27]:
```
function random_projection_approximate_least_squares(A, b, projection_
method, m_max)

    d = size(A)[2]
    x = inv(A'*A)*A'*b
    errors = []

    @showprogress for m in d:m_max

        S = projection_method(m, n)

        SA = S*A
        Sb = S*b

        x̃ = inv(SA'*SA)*SA'*Sb
        push!(errors, norm(x̃ .- x, 2) / norm(x, 2))

    end

    return errors

end
```

Out[27]:
```
random_projection_approximate_least_squares (generic function with 1
method)
```

In [28]:
```
m_max = 10d

rade_errs_GA = random_projection_approximate_least_squares(A_GA, b_GA,
rademacher_random_projection_matrix, m_max)
rade_errs_T3 = random_projection_approximate_least_squares(A_T3, b_T3,
rademacher_random_projection_matrix, m_max)
rade_errs_T1 = random_projection_approximate_least_squares(A_T1, b_T1,
rademacher_random_projection_matrix, m_max)

plot( d:m_max, rade_errs_GA, label="GA", xscale=:log10, yscale=:log10,
box=:on, thickness_scaling=1.1,
        title="Rademacher Random Projection", xlabel="Projection Dimen
sion", ylabel="||x̃-x||₂/||x||₂")
plot!(d:m_max, rade_errs_T3, label="T3")
plot!(d:m_max, rade_errs_T1, label="T1")
vline!([2d], c=:gray, ls=:dot, label=:none)

#png("als_error_rade_projection.png")
```

```
Progress: 100%|████████████████████████████████████| Time: 0:00
:07
Progress: 100%|████████████████████████████████████| Time: 0:00
:04
Progress: 100%|████████████████████████████████████| Time: 0:00
:04
```

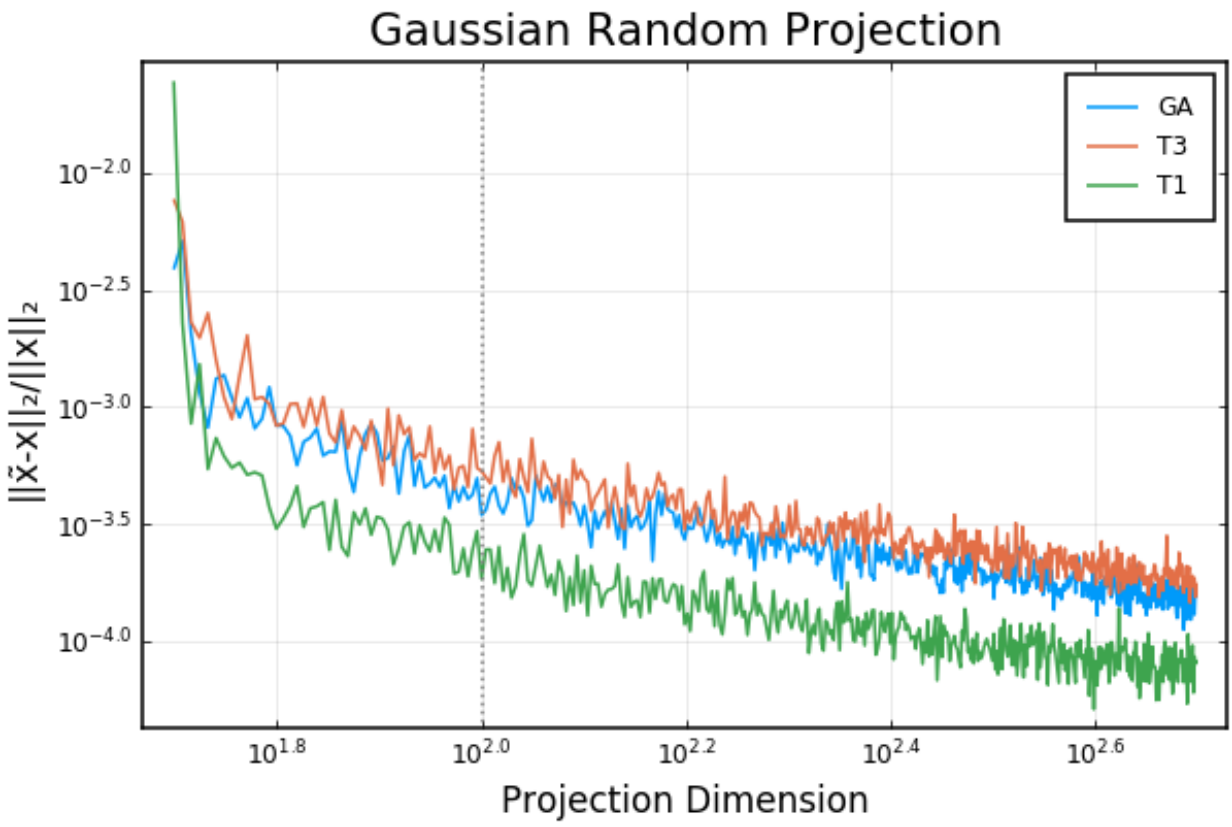
Out[28]:



In [29]:
```
m_max = 10d

gaussian_errs_GA = random_projection_approximate_least_squares(A_GA, b
_GA, gaussian_random_projection_matrix, m_max)
gaussian_errs_T3 = random_projection_approximate_least_squares(A_T3, b
_T3, gaussian_random_projection_matrix, m_max)
gaussian_errs_T1 = random_projection_approximate_least_squares(A_T1, b
_T1, gaussian_random_projection_matrix, m_max)

plot( d:m_max, gaussian_errs_GA, label="GA", xscale=:log10, yscale=:lo
g10, box=:on, thickness_scaling=1.1,
        title="Gaussian Random Projection", xlabel="Projection Dimensi
on", ylabel="||x̃-x||₂/||x||₂")
plot!(d:m_max, gaussian_errs_T3, label="T3")
plot!(d:m_max, gaussian_errs_T1, label="T1")
vline!([2d], c=:gray, ls=:dot, label=:none)

#png("als_error_gaussian_projection.png")
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:02
Progress: 100%|████████████████████████████████████████| Time: 0:00
:02
Progress: 100%|████████████████████████████████████████| Time: 0:00
:02
```

Out[29]:



Problem 6(c)

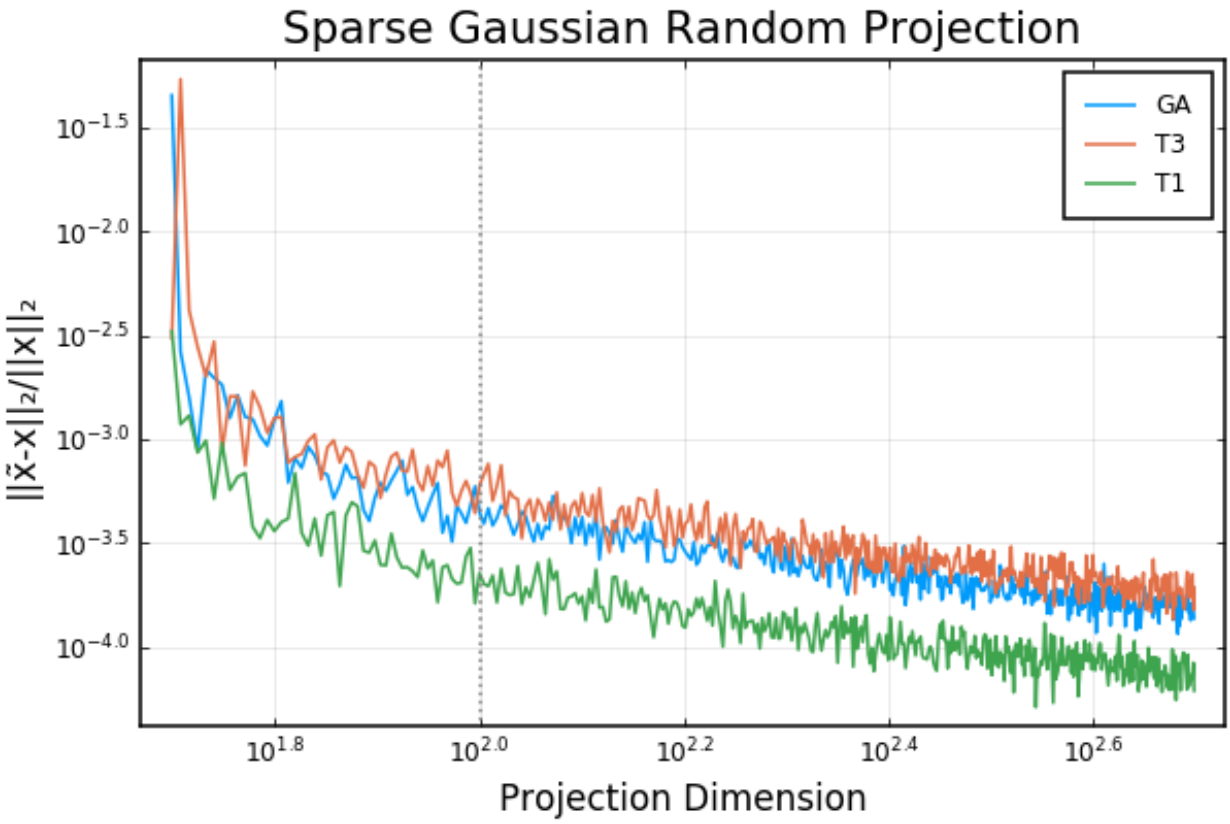
In [30]:
```
function sparse_gaussian_random_projection_matrix(m, d, q=0.5)

    s = [rand() <= q ? 0 : (1-q)*randn() for _ in 1:m*d]
    S = 1/sqrt(m)*reshape(s, (m, d))

    return S

end
```

Out[30]:
```
sparse_gaussian_random_projection_matrix (generic function with 2 me
thods)
```

In [31]:
```
m_max = 10d

sparse_gaussian_errs_GA = random_projection_approximate_least_squares(
A_GA, b_GA, sparse_gaussian_random_projection_matrix, m_max)
sparse_gaussian_errs_T3 = random_projection_approximate_least_squares(
A_T3, b_T3, sparse_gaussian_random_projection_matrix, m_max)
sparse_gaussian_errs_T1 = random_projection_approximate_least_squares(
A_T1, b_T1, sparse_gaussian_random_projection_matrix, m_max)

plot( d:m_max, sparse_gaussian_errs_GA, label="GA", xscale=:log10, ysc
ale=:log10, box=:on, thickness_scaling=1.1,
        title="Sparse Gaussian Random Projection", xlabel="Projection
Dimension", ylabel="||x̃-x||₂/||x||₂")
plot!(d:m_max, sparse_gaussian_errs_T3, label="T3")
plot!(d:m_max, sparse_gaussian_errs_T1, label="T1")
vline!([2d], c=:gray, ls=:dot, label=:none)

#png("als_error_sparse_gaussian_projection.png")
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:16
Progress: 100%|████████████████████████████████████████| Time: 0:00
:14
Progress: 100%|████████████████████████████████████████| Time: 0:00
:17
```

Out[31]:



In [32]:
```
function random_projection_approximate_least_squares_vary_q(A, b, proj
ection_method, m)

    d = size(A)[2]
    x = inv(A'*A)*A'*b
    qvals = []
    errors = []

    @showprogress for q in 0.01:0.01:0.99

        S = projection_method(m, n, q)

        SA = S*A
        Sb = S*b

        try
            x̃ = inv(SA'*SA)*SA'*Sb
            push!(qvals, q)
            push!(errors, norm(x̃ .- x, 2) / norm(x, 2))
        catch
        end

    end

    return qvals, errors

end
```

Out[32]:
```
random_projection_approximate_least_squares_vary_q (generic function
with 1 method)
```

In [33]:
```
m = d

q_GA, sparse_gaussian_errs_q_GA = random_projection_approximate_least_
squares_vary_q(A_GA, b_GA, sparse_gaussian_random_projection_matrix, m
)
q_T3, sparse_gaussian_errs_q_T3 = random_projection_approximate_least_
squares_vary_q(A_T3, b_T3, sparse_gaussian_random_projection_matrix, m
)
q_T1, sparse_gaussian_errs_q_T1 = random_projection_approximate_least_
squares_vary_q(A_T1, b_T1, sparse_gaussian_random_projection_matrix, m
)

p1 = plot(q_GA, sparse_gaussian_errs_q_GA, label="GA", yscale=:log10,
box=:on,
        title="Sparse Gaussian Proj. (m = $m)", xlabel="Sparsity Param
eter, q", ylabel="||x̃-x||₂/||x||₂")
plot!(q_T3, sparse_gaussian_errs_q_T3, label="T3")
plot!(q_T1, sparse_gaussian_errs_q_T1, label="T1")

m = Int(round(1.5d))
```

```
q_GA, sparse_gaussian_errs_q_GA = random_projection_approximate_least_
squares_vary_q(A_GA, b_GA, sparse_gaussian_random_projection_matrix, m
)
q_T3, sparse_gaussian_errs_q_T3 = random_projection_approximate_least_
squares_vary_q(A_T3, b_T3, sparse_gaussian_random_projection_matrix, m
)
q_T1, sparse_gaussian_errs_q_T1 = random_projection_approximate_least_
squares_vary_q(A_T1, b_T1, sparse_gaussian_random_projection_matrix, m
)

p2 = plot(q_GA, sparse_gaussian_errs_q_GA, label="GA", yscale=:log10,
box=:on,
        title="Sparse Gaussian Proj. (m = $m)", xlabel="Sparsity Param
eter, q", ylabel="||x̃-x||₂/||x||₂")
plot!(q_T3, sparse_gaussian_errs_q_T3, label="T3")
plot!(q_T1, sparse_gaussian_errs_q_T1, label="T1")

m = 2d

q_GA, sparse_gaussian_errs_q_GA = random_projection_approximate_least_
squares_vary_q(A_GA, b_GA, sparse_gaussian_random_projection_matrix, m
)
q_T3, sparse_gaussian_errs_q_T3 = random_projection_approximate_least_
squares_vary_q(A_T3, b_T3, sparse_gaussian_random_projection_matrix, m
)
q_T1, sparse_gaussian_errs_q_T1 = random_projection_approximate_least_
squares_vary_q(A_T1, b_T1, sparse_gaussian_random_projection_matrix, m
)

p3 = plot(q_GA, sparse_gaussian_errs_q_GA, label="GA", yscale=:log10,
box=:on,
        title="Sparse Gaussian Proj. (m = $m)", xlabel="Sparsity Param
eter, q", ylabel="||x̃-x||₂/||x||₂")
plot!(q_T3, sparse_gaussian_errs_q_T3, label="T3")
plot!(q_T1, sparse_gaussian_errs_q_T1, label="T1")

plot(p1, p2, p3, layout=(1,3), size=(1000, 400))
#png("als_error_sparse_gaussian_projection.png")
```

```
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:00
Progress: 100%|████████████████████████████████████████| Time: 0:00
:01
Progress: 100%|████████████████████████████████████████| Time: 0:00
:02
Progress: 100%|████████████████████████████████████████| Time: 0:00
:01
Progress: 100%|████████████████████████████████████████| Time: 0:00
:01
```

Out[33]: