```
clc; clear variables;
```

QR Decompositions

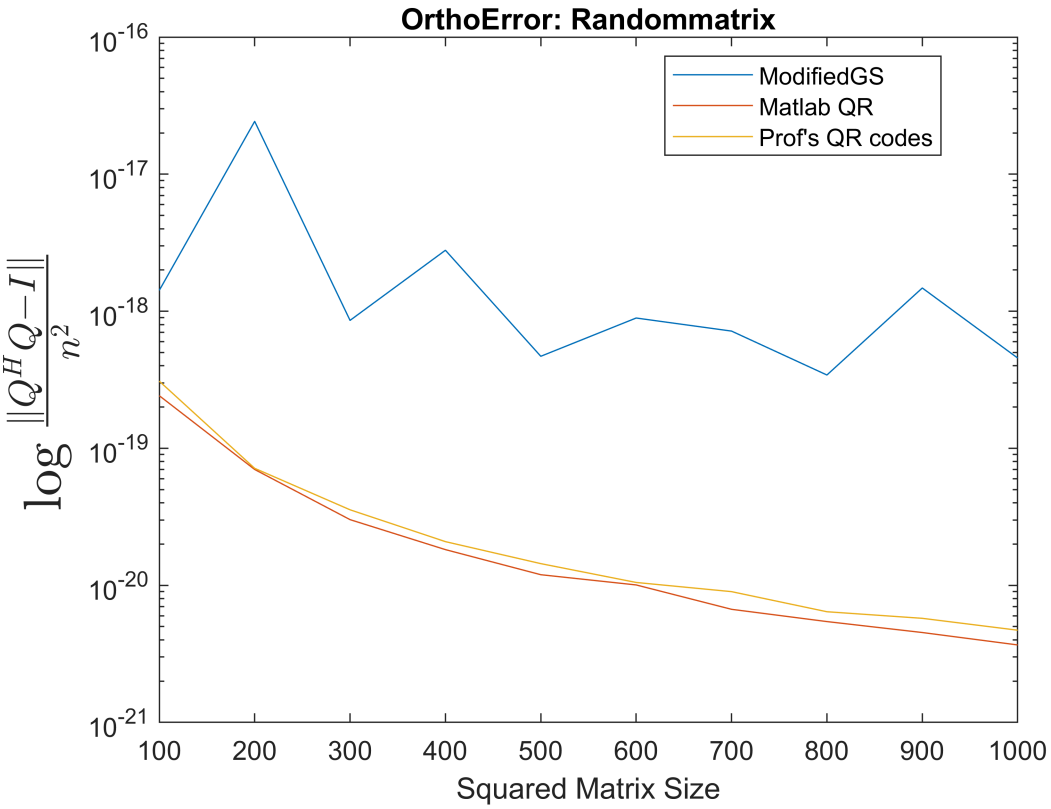
Reconstruction and Orthogonality erros on random matrices in increasing size.

```
import java.util.ArrayList;
Matrices = ArrayList();
MatrixSizes = 100: 100: 1000;
for I = MatrixSizes
    Matrices.add(rand(I));
end
[OrthErrs, RestrucErrs] = PerformenceSubroutine(Matrices,{@ModifiedGS, @qr, @QRFactorFromClass}
```

Plotting the Errors

```
figure;
semilogy(MatrixSizes, RestrucErrs(1, :));
hold on;
semilogy(MatrixSizes, RestrucErrs(2, :));
semilogy(MatrixSizes, RestrucErrs(3, :));
title("ReconstructionError: Randommatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert QR - A \Vert}{mn}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "ReconstructionError Randommatrix", "png");
```

```
figure;
semilogy(MatrixSizes, OrthErrs(1, :));
hold on;
semilogy(MatrixSizes, OrthErrs(2, :));
semilogy(MatrixSizes, OrthErrs(3, :));
title("OrthoError: Randommatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert Q^HQ - I \Vert}{n^2}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "OrthogonalityError Randommatrix", "png");
```

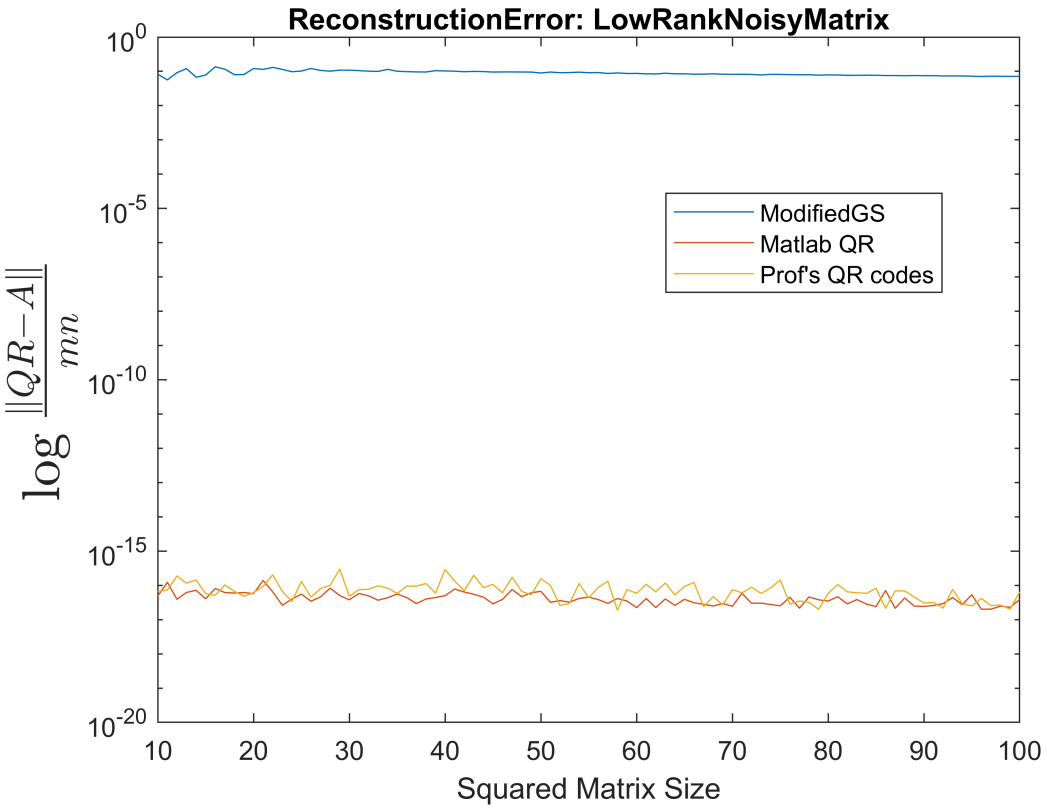


Pertubated Low Rank Matrix for accurancy testing

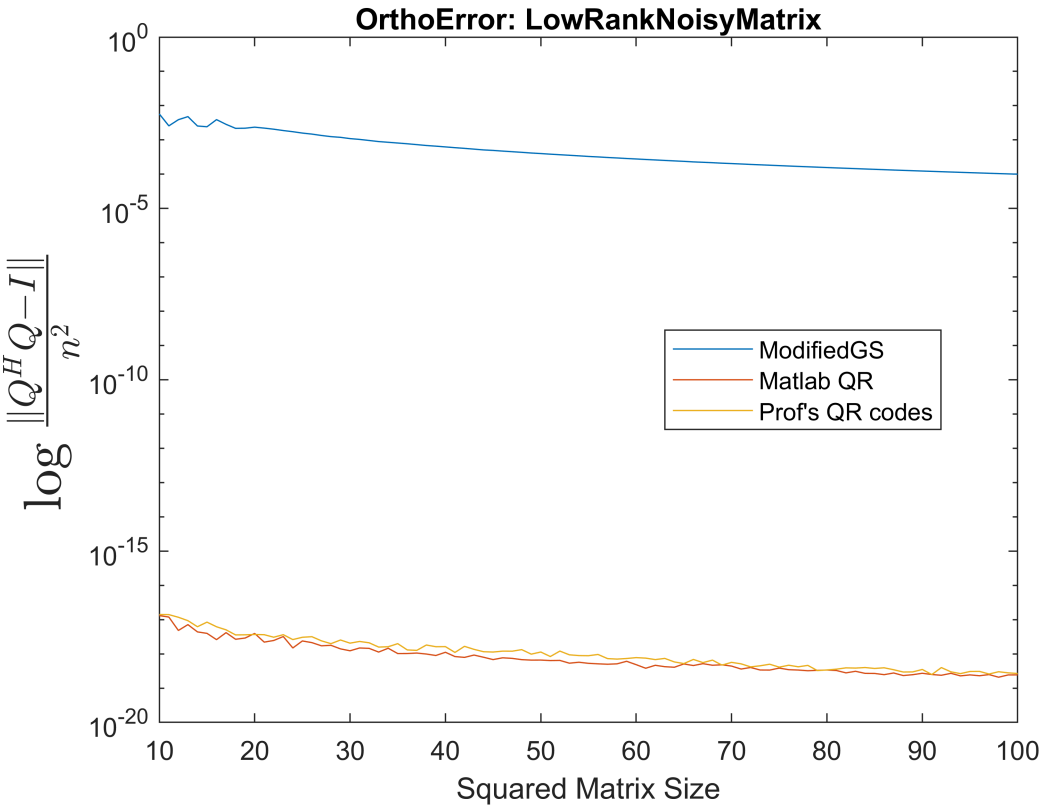
```
import java.util.ArrayList;
Matrices = ArrayList();
MatrixSizes = 10: 100;
for I = MatrixSizes
    Matrices.add(1./(1:I)'*(1:I) + rand(I)*1e-14);
end
[OrthErrs, RestrucErrs] = PerformenceSubroutine(Matrices,{@ModifiedGS, @qr, @QRFactorFromClass}
```

```
figure;
```

```
semilogy(MatrixSizes, RestrucErrs(1, :));
hold on;
semilogy(MatrixSizes, RestrucErrs(2, :));
semilogy(MatrixSizes, RestrucErrs(3, :));
title("ReconstructionError: LowRankNoisyMatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert QR - A \Vert}{mn}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "ReconstructionError LowRankNoisyMatrix", "png");
```




```
figure;
semilogy(MatrixSizes, OrthErrs(1, :));
hold on;
semilogy(MatrixSizes, OrthErrs(2, :));
semilogy(MatrixSizes, OrthErrs(3, :));
title("OrthoError: LowRankNoisyMatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert Q^HQ - I \Vert}{n^2}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "OrthogonalityError LowRankNoisyMatrix", "png");
```



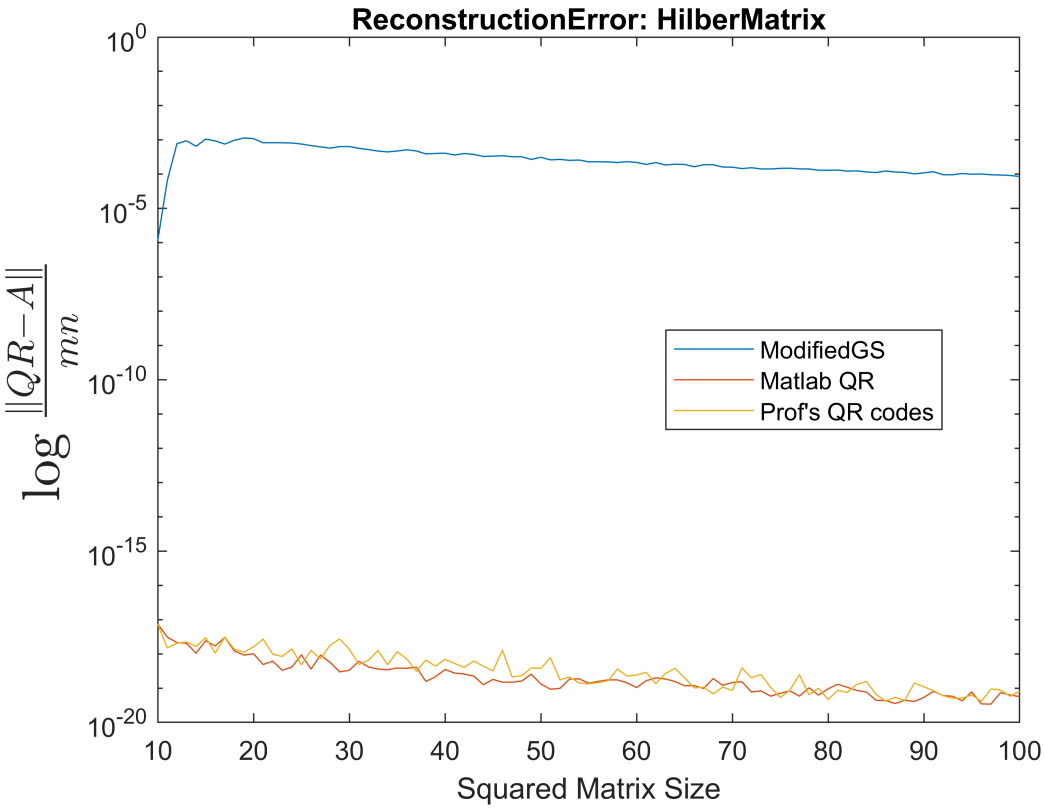
Hilbert Matrix Accuracy Testing

```
import java.util.ArrayList;
Matrices = ArrayList();
MatrixSizes = 10: 100;
for I = MatrixSizes
    [X, Y] = meshgrid(1:I);
    HilbertMatrix = 1./(X + Y);
    Matrices.add(HilbertMatrix);
end
[OrthErrs, RestrucErrs] = PerformenceSubroutine(Matrices,{@ModifiedGS, @qr, @QRFactorFromClass}
```

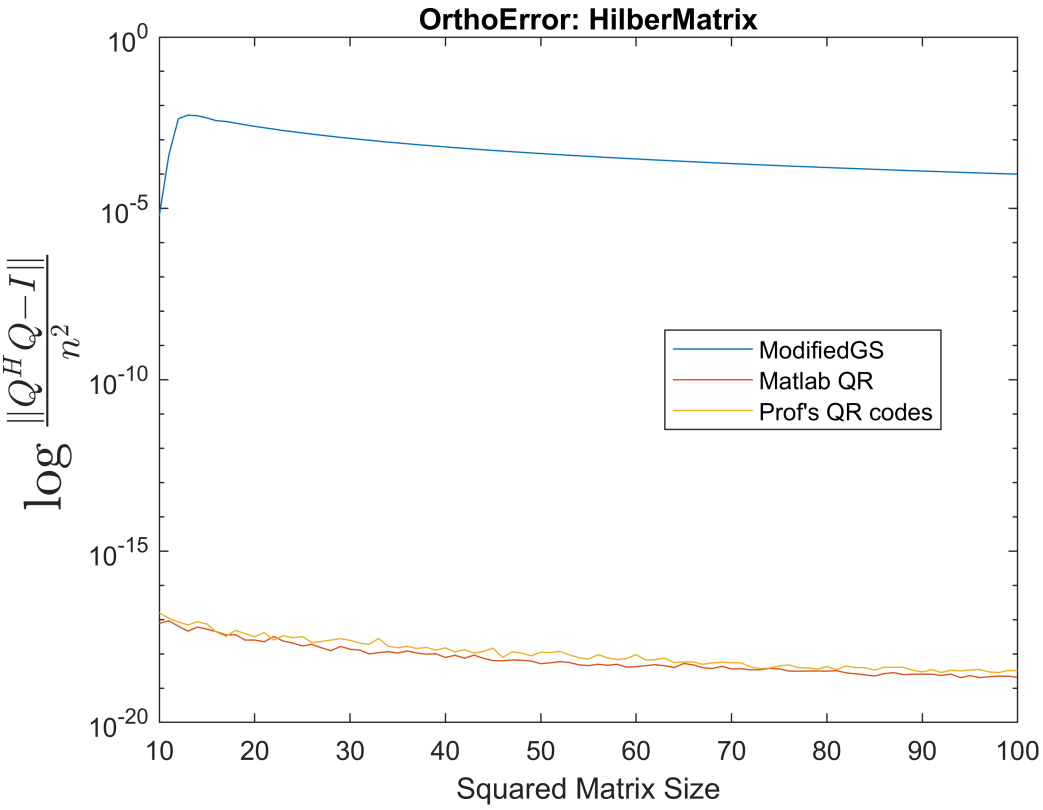
Plotting all of them out:

```
figure;
semilogy(MatrixSizes, RestrucErrs(1, :));
hold on;
semilogy(MatrixSizes, RestrucErrs(2, :));
semilogy(MatrixSizes, RestrucErrs(3, :));
title("ReconstructionError: HilberMatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert QR - A \Vert}{mn}}$", "interpreter", "latex", "FontSize", 20);
```

```
saveas(gcf, "ReconstructionError HilberMatrix", "png");
```




```
figure;
semilogy(MatrixSizes, OrthErrs(1, :));
hold on;
semilogy(MatrixSizes, OrthErrs(2, :));
semilogy(MatrixSizes, OrthErrs(3, :));
title("OrthoError: HilberMatrix");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert Q^HQ - I \Vert}{n^2}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "OrthogonalityError HilberMatrix", "png");
```

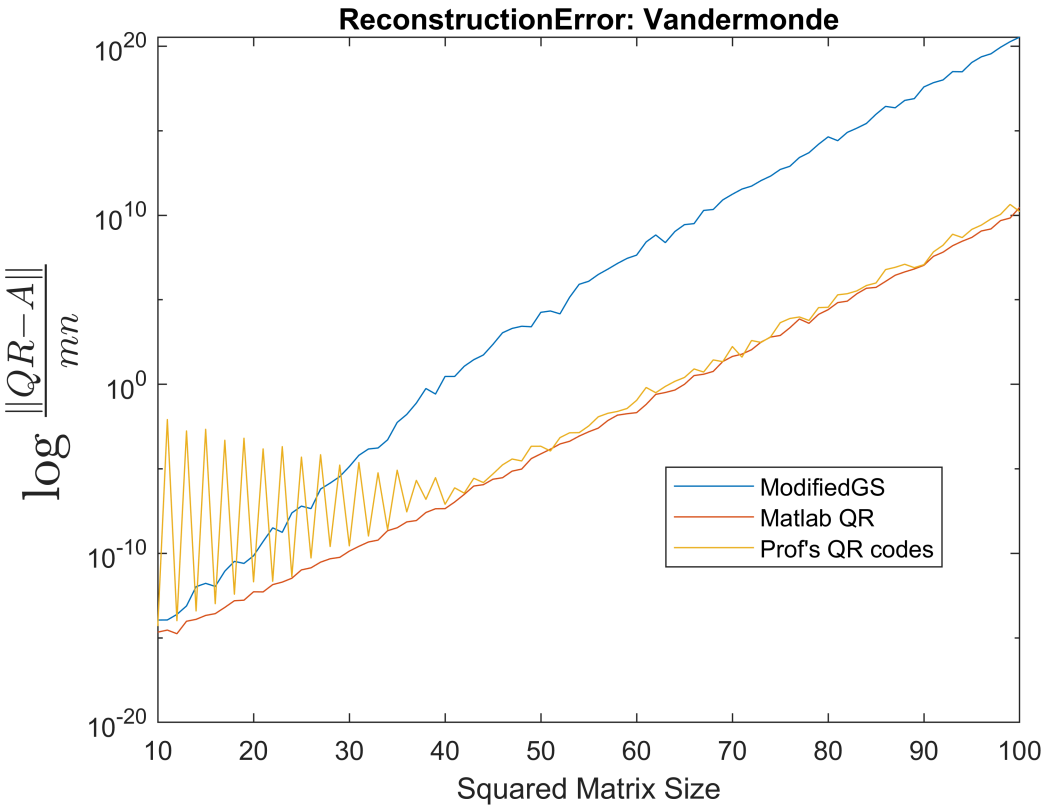


Vandermonde Matrix Accuracy Testing

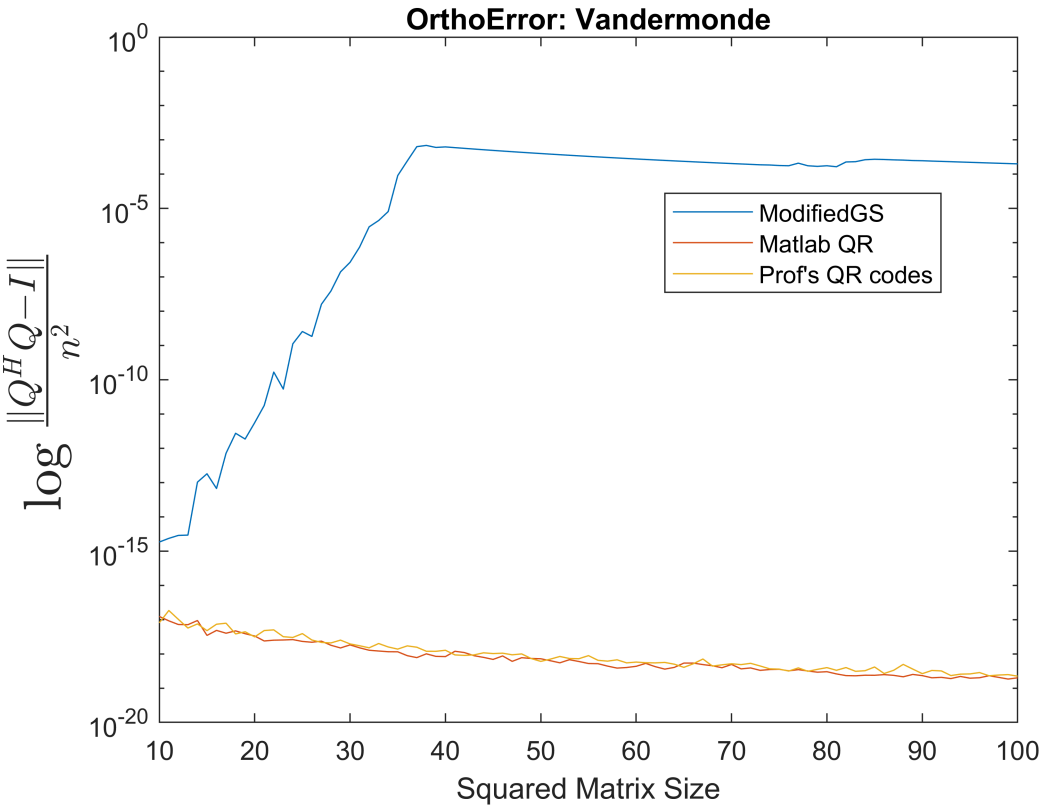
```
import java.util.ArrayList;
Matrices = ArrayList();
MatrixSizes = 10: 100;
for I = MatrixSizes
    Matrices.add(vander(linspace(-2, 2, I)));
end
[OrthErrs, RestrucErrs] = PerformenceSubroutine(Matrices,{@ModifiedGS, @qr, @QRFactorFromClass}
```

Plotting things out:

```
figure;
semilogy(MatrixSizes, RestrucErrs(1, :));
hold on;
semilogy(MatrixSizes, RestrucErrs(2, :));
semilogy(MatrixSizes, RestrucErrs(3, :));
title("ReconstructionError: Vandermonde");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert QR - A \Vert}{mn}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "ReconstructionError Vandermonde", "png");
```




```
figure;
semilogy(MatrixSizes, OrthErrs(1, :));
hold on;
semilogy(MatrixSizes, OrthErrs(2, :));
semilogy(MatrixSizes, OrthErrs(3, :));
title("OrthoError: Vandermonde");
legend("ModifiedGS", "Matlab QR", "Prof's QR codes", "location", "best");
xlabel("Squared Matrix Size");
ylabel("$\log{\frac{\Vert Q^HQ - I \Vert}{n^2}}$", "interpreter", "latex", "FontSize", 20);
saveas(gcf, "OrthogonalityError Vandermonde", "png");
```



Polynomials Conditioning

II (a), Plotting the badly conditioned polynomials:

```
Xs = 1.920: 0.001: 2.08;
plot(Xs, BadPolynomial(Xs))
saveas(gcf, "BadPolynomial.png", "png")
```

II (b), Plotting the goodly conditioned polynomials:

```
Xs = 1.920: 0.001: 2.08;
plot(Xs, GoodPolynomial(Xs))
saveas(gcf, "GoodPolynomial.png", "png")
```

Matrix Conditioning

Condition number of a random matrix as the size increases.

```
MatrixSizes = 1:10:1000
ConditionNumbers = zeros(1, length(MatrixSizes))
for I = length(MatrixSizes)
    ConditionNumbers(I) = cond(rand(MatrixSizes(I)));
```

```
end
```

Plotting it out:

```
```