Fast Iterative Solvers Project 1

Kenny Yung
Mat# 416070
Due 22.06.2021

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv

# reads csv file for 3 cols
def read_3col(address):
    c1=[]
    c2=[]
    c3=[]
    with open(address) as file:
        read = csv.reader(file, delimiter=',')
        line = 0
        for row in read:
            if (line == 0):
                line += 1
            else:
                c1 += [int(row[0])]
                c2 += [float(row[1])]
                c3 += [float(row[2])]
    return c1,c2,c3

# reads csv file for 2 cols
def read_2col(address):
    k=[]
    r=[]
    with open(address) as file:
        read = csv.reader(file, delimiter=',')
        line = 0
        for row in read:
            if (line == 0):
                line += 1
            else:
                k += [int(row[0])]
                r += [float(row[1])]
    return k,r
```

Full GMRES method

Solving for $Ax = b$, where A is "orsirr_1.mtx" from matrix market, b is vector of 1's, and starting guess x_0 is vector of 0's

Number of Krylov vectors required to achieve relative residual tolerance $\frac{||r_k||}{||r_0||}$ of 1e-8:

Table 1. Runtimes for full GMRES

Method	# Krylove vectors	Runtime (s)

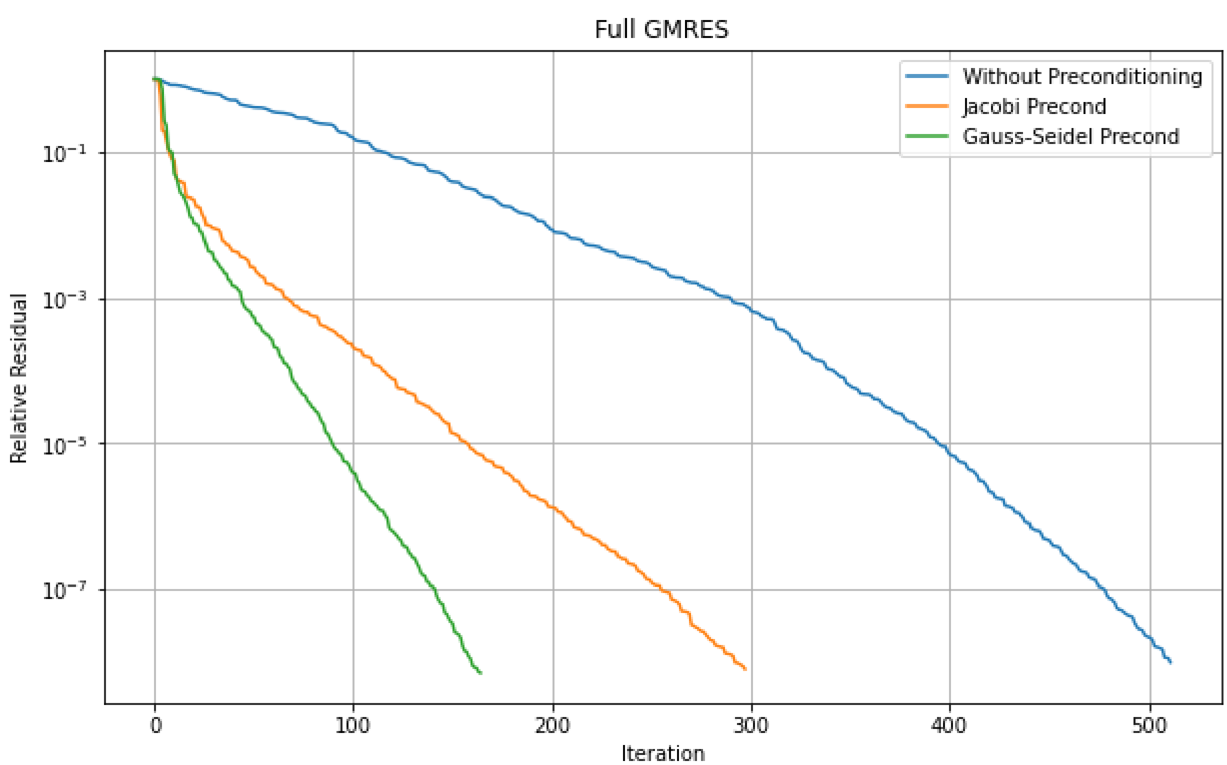
Method	# Krylove vectors	Runtime (s)
Without preconditioning	m = 512	46.6
Jacobi preconditioning	m = 298	17.0
Gauss-Seidel preconditioning	m = 165	5.24

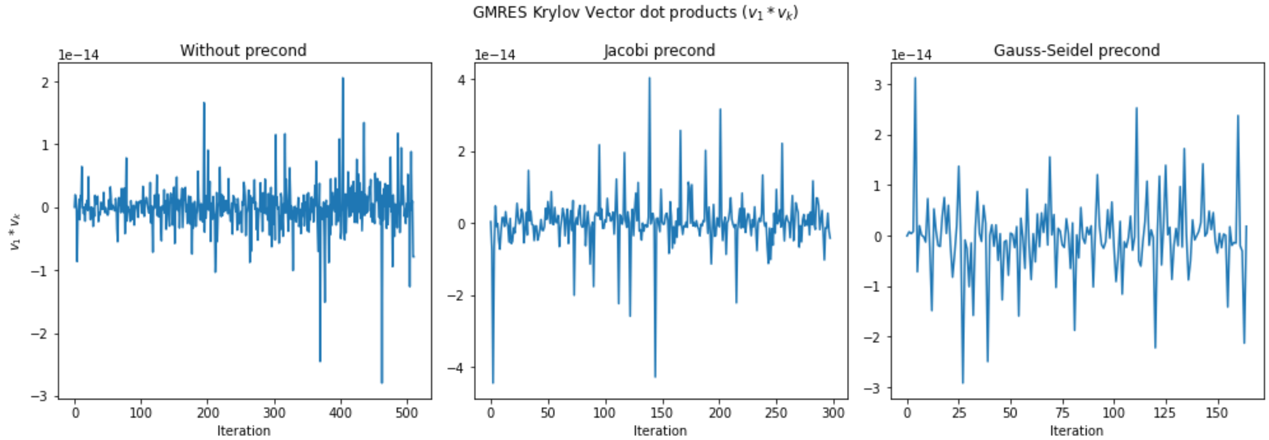
In [2]:

```
k_wo,ortho_wo,r_wo = read_3col('results/gmres_wo.csv')
k_jc,ortho_jc,r_jc = read_3col('results/gmres_jc.csv')
k_gs,ortho_gs,r_gs = read_3col('results/gmres_gs.csv')

plt.figure(figsize=(10,6))
plt.semilogy(k_wo,r_wo,k_jc,r_jc,k_gs,r_gs)
plt.legend(['Without Preconditioning','Jacobi Precond','Gauss-Seidel Precond'])
plt.title('Full GMRES')
plt.xlabel('Iteration')
plt.ylabel('Relative Residual')
plt.grid(True)
plt.show()

fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(14,5))
ax1.plot(k_wo,ortho_wo)
ax1.set_title('Without precond')
ax1.set_ylabel('$v_1*v_k$')
ax1.set_xlabel('Iteration')
ax2.plot(k_jc,ortho_jc)
ax2.set_title('Jacobi precond')
ax2.set_xlabel('Iteration')
ax3.plot(k_gs,ortho_gs)
ax3.set_title('Gauss-Seidel precond')
ax3.set_xlabel('Iteration')
fig.suptitle("GMRES Krylov Vector dot products ($v_1*v_k$)")
plt.tight_layout()
```





As seen from the first plot, preconditioning cuts down a significant amount of iteration steps, with Gauss-Seidel having better performance than Jacobi method.

The second set of plots confirms the orthogonality of all the krylov vectors (which are column vectors in matrix V_m) with $v_1 * v_k = 0 \pm 4\mathrm{e}{-14}$ which is within machine error.

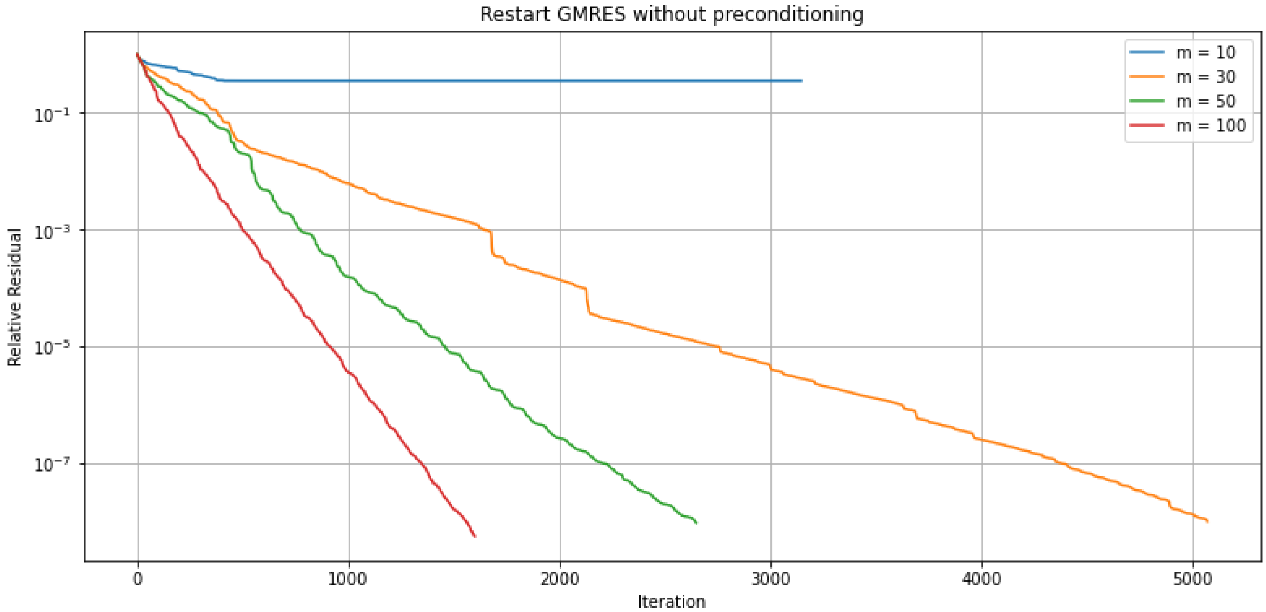
Restart GMRES method

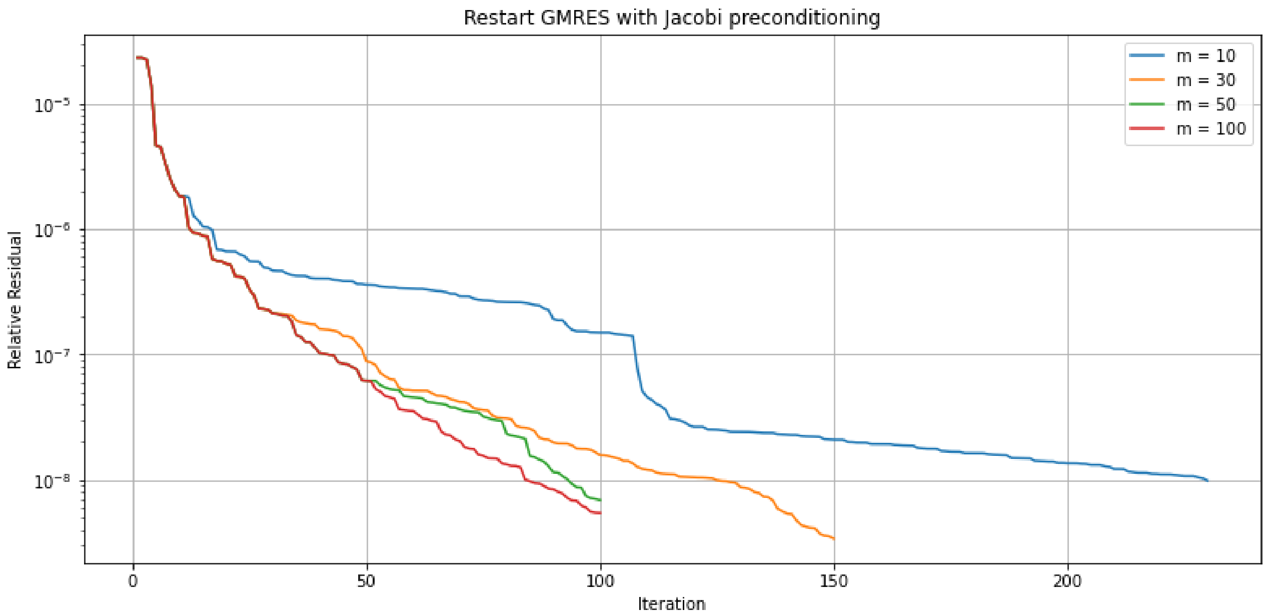
Investigate parameters relationships for restart GMRES method where it repeats full GMRES until convergence condition is achieved

In [3]:

```
# restart plot
def plot_rgmres(title,k1,r1,k2,r2,k3,r3,k4,r4):
    plt.figure(figsize=(13,6))
    plt.semilogy(k_10,r_10,k_30,r_30,k_50,r_50,k_100,r_100)
    plt.legend(['m = 10','m = 30','m = 50','m = 100'])
    plt.title(title)
    plt.xlabel('Iteration')
    plt.ylabel('Relative Residual')
    plt.grid(True)
    plt.show()

k_10,r_10 = read_2col("results/rgmres_wo_10.csv")
k_30,r_30 = read_2col("results/rgmres_wo_30.csv")
k_50,r_50 = read_2col("results/rgmres_wo_50.csv")
k_100,r_100 = read_2col("results/rgmres_wo_100.csv")
plot_rgmres('Restart GMRES without preconditioning',k_10,r_10,k_30,r_30,k_50,r_5
k_10,r_10 = read_2col("results/rgmres_jc_10.csv")
k_30,r_30 = read_2col("results/rgmres_jc_30.csv")
k_50,r_50 = read_2col("results/rgmres_jc_50.csv")
k_100,r_100 = read_2col("results/rgmres_jc_100.csv")
plot_rgmres('Restart GMRES with Jacobi preconditioning',k_10,r_10,k_30,r_30,k_50
k_10,r_10 = read_2col("results/rgmres_gs_10.csv")
k_30,r_30 = read_2col("results/rgmres_gs_30.csv")
k_50,r_50 = read_2col("results/rgmres_gs_50.csv")
k_100,r_100 = read_2col("results/rgmres_gs_100.csv")
plot_rgmres('Restart GMRES with Gauss-Seidel preconditioning',k_10,r_10,k_30,r_3
```





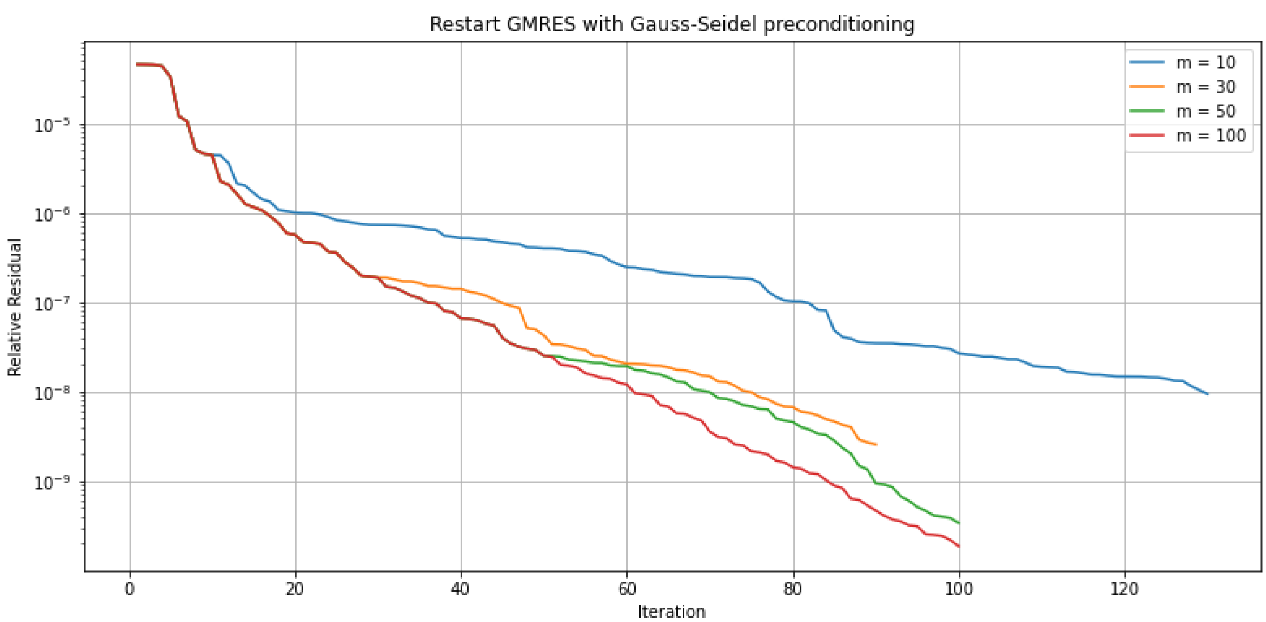


Table 2. Runtimes for Restart GMRES with various parameters

# Krylov vectors	Runtime without precond (s)	Runtime w Jacobi precond (s)	Runtime w Gauss-Seidel precond (s)
m = 10	did not converge	0.896 (23 iter)	0.561 (13 iter)
m = 30	32.082 (169 iter)	0.980 (5 iter)	1.048 (3 iter)
m = 50	28.821 (53 iter)	1.091 (2 iter)	1.262 (2 iter)
m = 100	32.661 (16 iter)	2.200 (1 iter)	2.215 (1 iter)

Seeing that the unpreconditioned restart GMRES doesn't converge for $m = 10$ krylov vectors, there is a minimum m requirement for each matrix problem in order to converge. Even though choosing a higher m will lead to convergence with less cumulative steps, the runtime is roughly the same. I suspect it is because each restart adds a "big" calculation of the large sparse matrix vector, which dominates in computing time than the lessor FLOPs.

In my implementation, runtimes for restart GMRES are faster than full GMRES for all 3 conditions and m values. I believe the reason is because as m parameter increases, it adds FLOPs in a nonlinear, quadratic way. We can see that the i and k loop for the GetKrylov and GivensRotation operations adds an order of m within each j iteration through m.

Other than runtime, additional motivation to use restart as opposed to full GMRES might include memory management, as full GMRES with a much larger m stores much more in memory for the Hessenberg matrix and Krylov vectors. Therefore, restart GMRES can be ran both faster and more efficiently on a machine with more memory contraints.

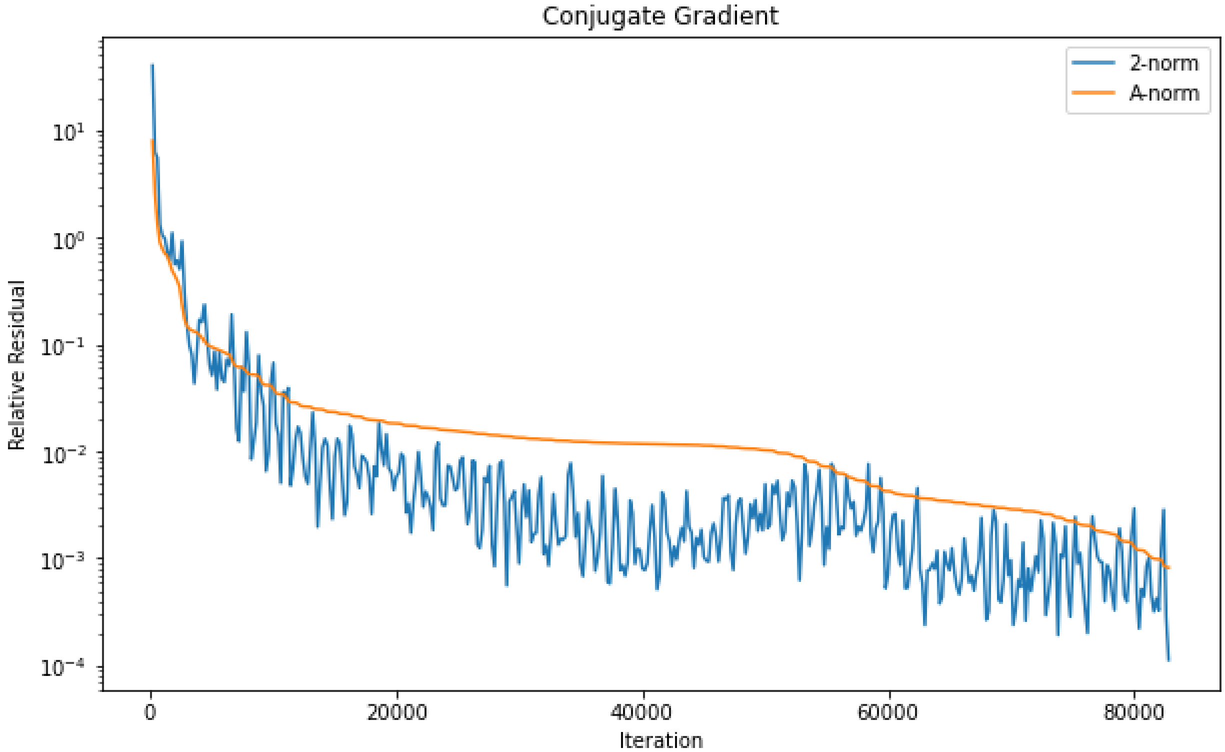
Conjugate Gradient method

Solving for $Ax = b$, where A is "s3rmt3m3.mtx" from matrix market, b is vector of 1's, and starting guess x_0 is vector of 0's

In [4]:

```
# error: 2-norm vs A-norm plot
k,two_norm,a_norm = read_3col('results/cg.csv')

plt.figure(figsize=(10,6))
plt.semilogy(k,two_norm,k,a_norm)
plt.legend(['2-norm','A-norm'])
plt.title('Conjugate Gradient')
plt.xlabel('Iteration')
plt.ylabel('Relative Residual')
plt.show()
```



Convergence requirement: relative residual tolerance $\frac{||r_k||}{||r_0||}$ of 1e-8

Runtime: 1041.06s (1467.96*)

Iteration required: 82801 (137080*)

*for absolute residual

As seen from the plot, the A-norm and the 2-norm follows a similar decreasing trajectory. However, the 2-norm fluctuates up and down significantly more than the A-norm as they decrease in residual. I suspect the reason is because CG method searches for solution that minimizes the A norm of the solution. Therefore the method is A optimal, and we can see solutions x_m that decrease in A-norm while increasing the 2-norm.