QUASI-MINIMAL RESIDUAL ALGORITHM

Sarina Heshmati

sarinaheshmatii@gmail.com

INTRODUCTION



INTRODUCTION

- The new world needs new tools in order to solve its problems.
 - The tools need to efficient, precise and simple.
 - QMR is one of the answers.

BRIEF EXPLANATION





Quasi



Minimal



Residual





Iterative method as an improvement over the MINRES algorithm

Minimizing the residual by projecting it onto a Krylov subspace

The Difference with CG algorithm

Can be improved by using the appropriate preconditioners

THE ALGORITHM



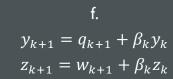
THE STEPS FOR SOLVING:

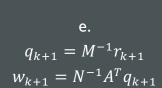
$$Ax = b$$





- a. Krylov subspace vectors
- b. Step sizes
- c. Update the solution
- d. Update the residual
- e. Left & right preconditioned residuals
- f. Update the search directions





$$\alpha_k = \frac{(r_k, z_k)}{(y_k, Ay_k)}$$
$$\beta_k = \frac{(r_k, A^T z_k)}{(y_k, A^T z_k)}$$

e.
$$= M^{-1}r_{k+1}$$

$$= N^{-1}A^{T}q_{k+1}$$

$$x_{k+1} = x_{k}$$

 Ay_k, A^Tz_k

d.
$$r_{k+1} = r_k - \alpha_k A y_k$$

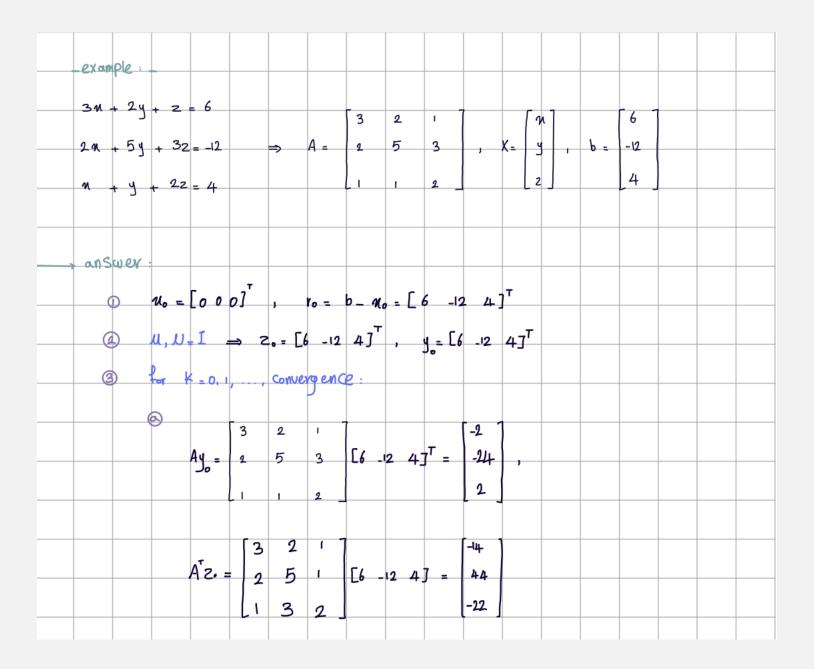
$$x_{k+1} = x_k + \alpha_k y_k$$



THE ALGORITHM 06 of 21

```
import numpy as np
     # define the matrix A
     A = np.array([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
     # define the right-hand side vector b
     b = np.array([1, 0, 1])
     # solve the system Ax = b using QMR
     max_iter = 1000
     tol = 1e-8
11
     x = np.zeros_like(b, dtype=float)
     r = b - A @ x
     p = r.copy()
     q = r.copy()
     for k in range(max_iter):
         y = A @ q
17
18
         z = A.T @ p
19
         alpha = np.dot(r, q) / np.dot(y, z)
         beta = np.dot(r, z) / np.dot(y, z)
20
21
         x += alpha * p
22
         r -= alpha * y
23
         p = r + beta * p
24
         q = q - beta * z
25
         if np.linalg.norm(r) < tol:</pre>
26
             print(f"Converged in {k+1} iterations")
27
             break
28
29
     # Print the solution
     print("Converged in 1000 iterations \n")
     print(f"Solution using QMR: {x}")
```







Ь	$\alpha_{\circ} = \frac{(v_{\circ}, z_{\circ})}{(y_{\circ}, Ay_{\circ})} = \frac{196}{284} = 0.69$, $\beta_{\circ} = \frac{(v_{\circ}, A^{T}z_{\circ})}{(y_{\circ}, A^{T}z_{\circ})} = \frac{-700}{-700} = 1$
	$\mathcal{H}_{1} = [0000]^{T} + 0.69[6 -12 4]^{T} = [4.14 -8.28 2.76]^{T}$
	$r_1 = \begin{bmatrix} 6 & -12 & 4 \end{bmatrix}^T = \begin{bmatrix} 0.69 & \boxed{1} & -24 & 2 \end{bmatrix}^T = \begin{bmatrix} 7.38 & 4.56 & 2.62 \end{bmatrix}^T$
e	3 2 1 77.38 7 33.88
(P)	$9, = [7.38 \ 4.56 \ 2.62]^{T}, \omega_{1} = [3 \ 2 \ 1] [7.38] = [33.88] = 40.18$
	$y_1 = [7.38 4.56 2.62]^{T} + [x [6 12 4]^{T} = [13.38 7.44 6.62]^{T}$ $2_1 = \begin{bmatrix} 33.88 \\ 40.18 \end{bmatrix} + [x [6 -12 4]^{T} = [39.88 29.18 30.3]^{T}$
	[26-3]



EACH ITERATION'S ANSWER (TOL = 1e - 6)

k	$\dfrac{ r_k }{ b }$
0	1.732
I I	0.843
2	0.218
3	0.031
4	0.008
5	0.0003
6	0.00002



EACH ITERATION'S ANSWER (TOL = 1e-6) USING IC METHOD FOR PRECONDITIONERS

k	$rac{ig r_k ig }{ m{b} }$
0	1.732
I	0.196
2	2.018e-08



COMPARISON, NUMBER OF ITERATIONS

QMR without any preconditioners	QMR with preconditioners
7	3



TIME AND SPACE COMPLEXITY

time	space
$O(n^2)$	O(n)

The given time complexity is in general and the actual amount is very dependence on the matrix properties.

STRENGTHS AND WEAKNESSES



STRENGTHS AND WEAKNESSES

STRENGTHS

- Non-symmetric
- 2. Indefinite

WEAKNESSES

- III-conditioned
- 2. Highly singular

NON-SYMMETRIC

```
import numpy as np
import time
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import qmr
from scipy.sparse.linalg import cg
# Define the matrix A and vector b
A = csc_{matrix}([[10, 1, -2], [2, 8, -3], [1, -7, 1]], dtype=float)
b = np.array([1, 1, 1], dtype=float)
start = time.time()
x, exitCode = qmr(A, b)
end = time.time()
print("answer: ", x)
print("exit code: ", exitCode)
print("time: ", end - start)
start = time.time()
x, exit_code = cg(A, b)
end = time.time()
print("answer: ", x)
print("exit code: ", exitCode)
print("time: ", end - start)
```



NON-SYMMETRIC

QMR

answer: [-0.14285714 -0.36263736 -1.3956044]

exit code: 0

time: 0.000881195068359375

answer: [-0.13971585 -0.36901004 -1.41650883]

exit code: 0

time: 0.000997304916381836

ILL-CONDITIONED

STRENGTHS AND WEAKNESSES 16 of 21

```
import numpy as np
import time
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import qmr
from scipy.sparse.linalg import cg
# Define the matrix A and vector b
A = csc_matrix([[1.001, 2], [2, 4.001]], dtype=float)
b = np.array([1, 1], dtype=float)
start = time.time()
x, exitCode = qmr(A, b)
end = time.time()
print("answer: ", x)
print("exit code: ", exitCode)
print("time: ", end - start)
start = time.time()
x, exit\_code = cg(A, b)
end = time.time()
print("answer: ", x)
print("exit code: ", exitCode)
print("time: ", end - start)
```



ILL-CONDITIONED

QMR

answer: [400.119976 -199.76004799]

exit code: 0

time: 0.0011818408966064453

answer: [400.119976 -199.76004799]

exit code: 0

time: 0.0011818408966064453



OTHER STRENGTHS & WEAKNESSES

STRENGTHS

 Can be implemented in a way that allows for parallelization, hence it is well-suited for large matrices.

WEAKNESSES

 Requires more memory space since it is a bi-orthogonal method.

CONCLUSION



CONCLUSION

- Iterative bi-orthogonal Krylov subspace method.
- Well-suited for indefinite and non-symmetric matrices.
- Can be improved with appropriate preconditioners.
- More precision given it's a bi-orthogonal method.
- High memory cost.



SOURCES

- 1. Yousef Saad "Iterative Methods for Sparse Linear Systems"
- 2. https://mathworld.wolfram.com/Quasi-MinimalResidualMethod.html
- 3. C.T.Kelley"Iterativemethodsforlinearandnonlinearequations"



THANKS FOR YOUR ATTENTION. CONTACT INFO:

sarinaheshmatii@gmail.com

sarinaheshmati@aut.ac.ir

+989102460274