Home work sheet 3
Report

Mohammad Mahdi Fallah
01428941

In this report we take a look at the runtime and convergence rate of my implemented (preconditioned) conjugate gradient and we see a comparison with the standard octave pcg().

Implementation:
For this homework I have used octave v.6.

First I implemented residual() method in order to get relative residuals, but I used octave normest() instead of norm() to make the functions run quicker.
Then I implemented standard conjugate gradient in cg.m and a preconditioned version in p_cg.m.
For preconditions that p_cg() uses I have Implemented different additional functions:

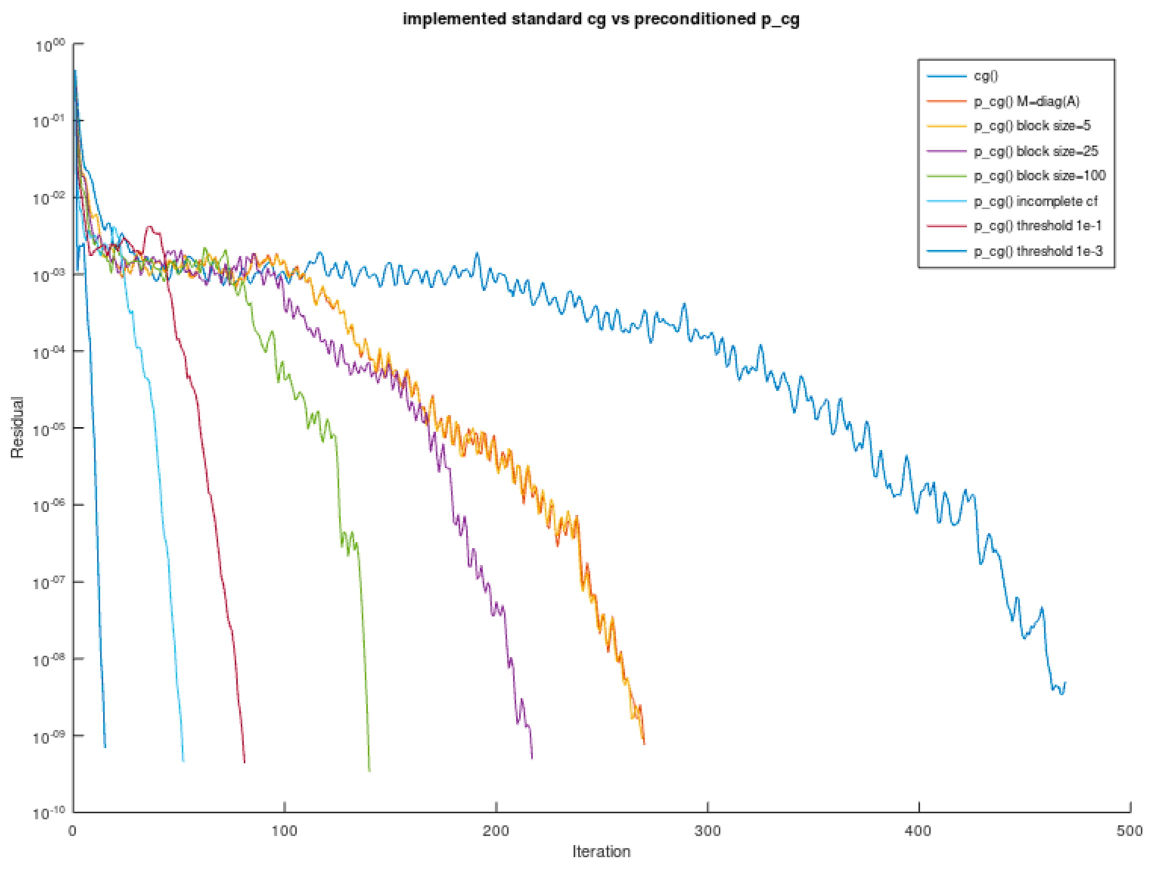
1.Diagonal precondition: I implemented it in Jacobi.m with a simple diag() function.

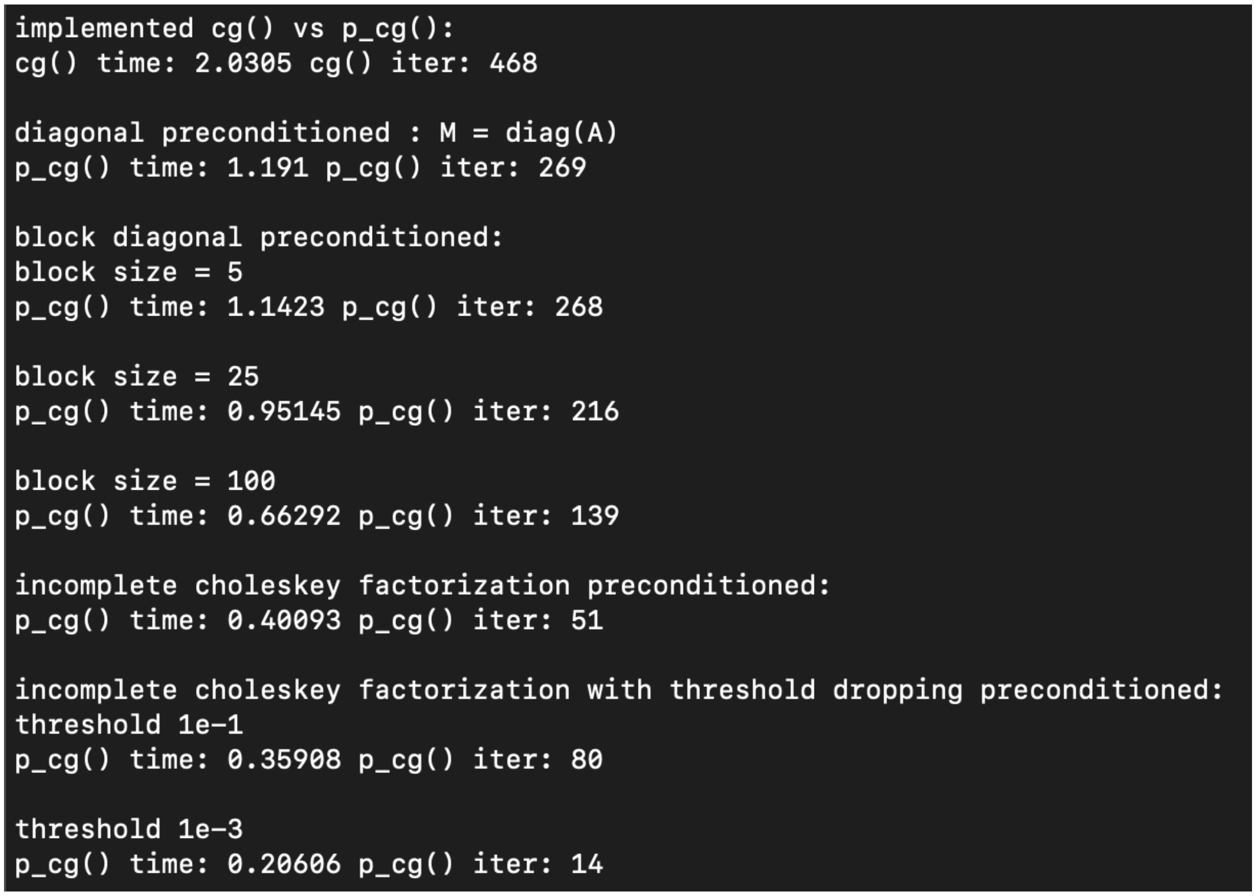
2.Block diagonal precondition: I implemented a function that computes the block diagonal of a matrix I'm block_jacobi.m

At the end in assignment3.m I wrote a script that test the three given matrix using cg() and p_cg() with all possible preconditions and the results of each matrix are in the following pages.
The first part of the results is the graph showing the convergence histories, And the second part is the runtime and iteration of each conjugate gradient method which is available through terminal.
Note that the computing of preconditions is included in the runtimes.

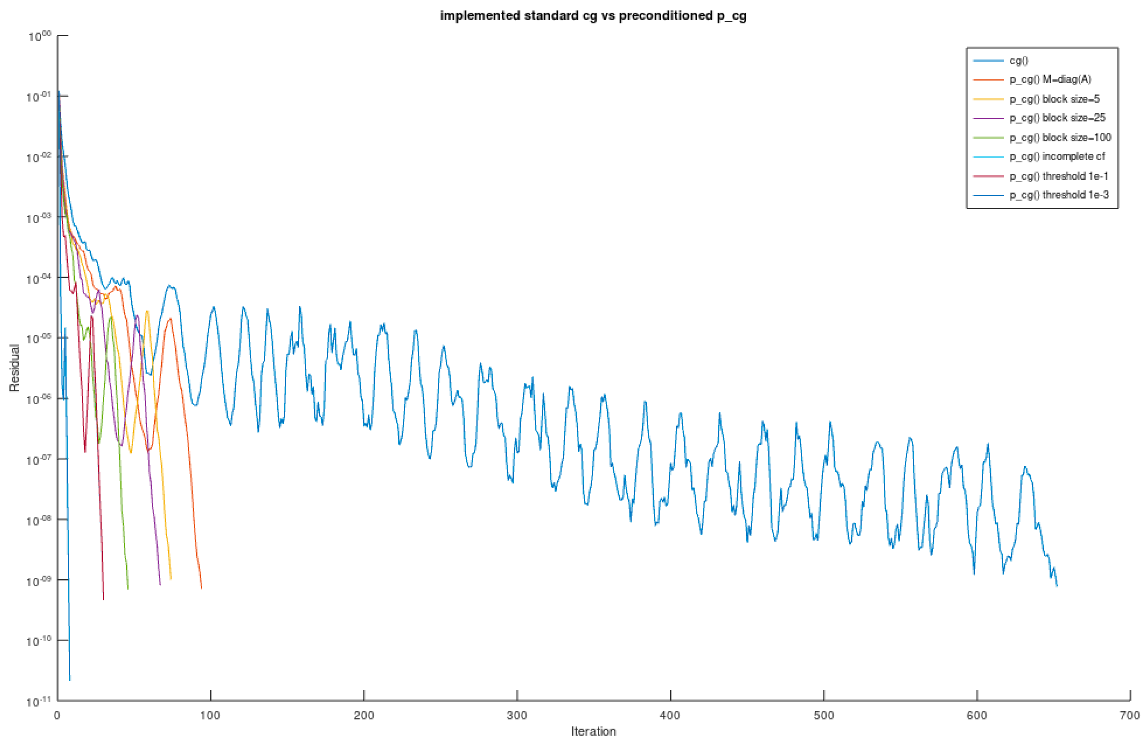
For reading the sparse format matrix I have used mmread.m file.

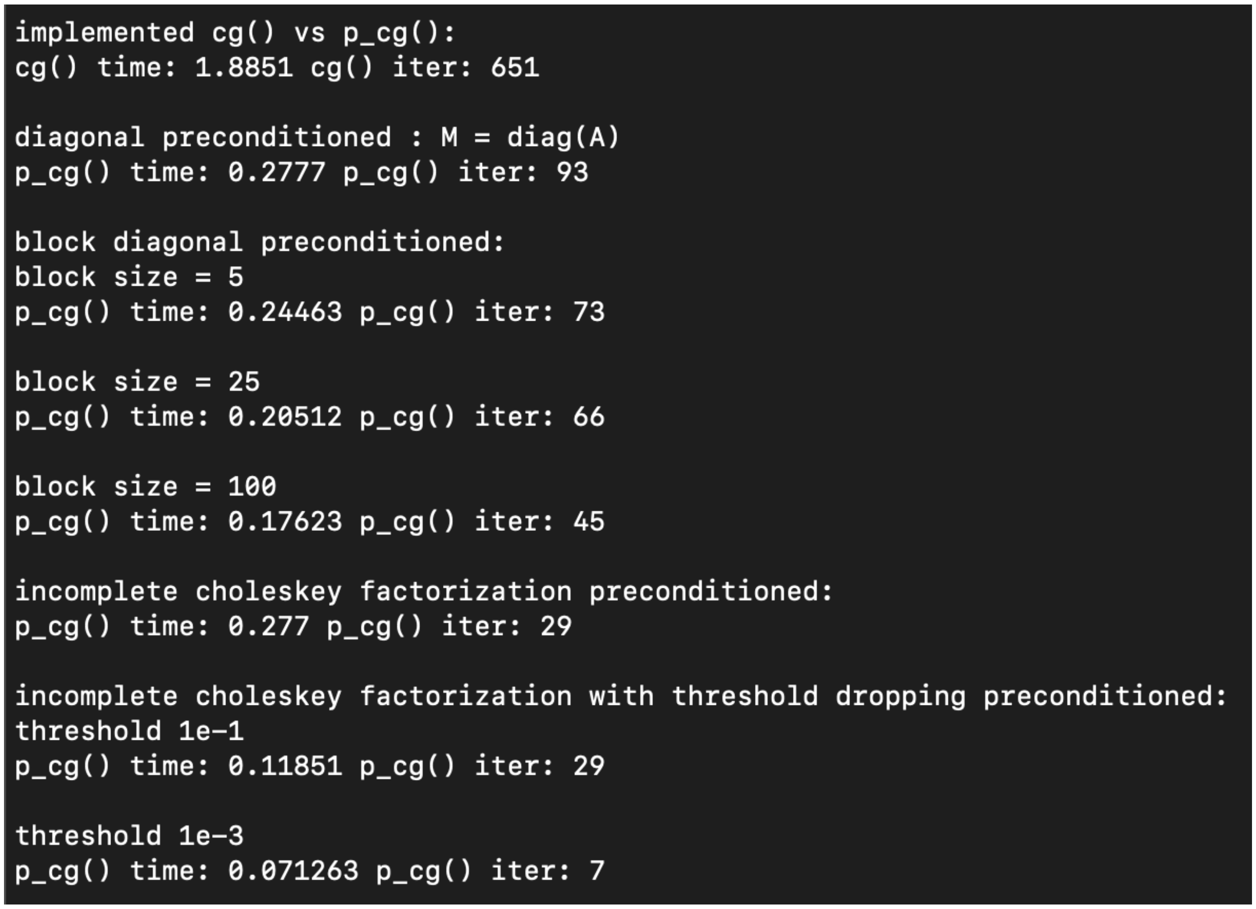
Results of nos5.mtx




```
implemented cg() vs p_cg():
cg() time: 2.0305 cg() iter: 468

diagonal preconditioned : M = diag(A)
p_cg() time: 1.191 p_cg() iter: 269

block diagonal preconditioned:
block size = 5
p_cg() time: 1.1423 p_cg() iter: 268

block size = 25
p_cg() time: 0.95145 p_cg() iter: 216

block size = 100
p_cg() time: 0.66292 p_cg() iter: 139

incomplete choleskey factorization preconditioned:
p_cg() time: 0.40093 p_cg() iter: 51

incomplete choleskey factorization with threshold dropping preconditioned:
threshold 1e-1
p_cg() time: 0.35908 p_cg() iter: 80

threshold 1e-3
p_cg() time: 0.20606 p_cg() iter: 14
```

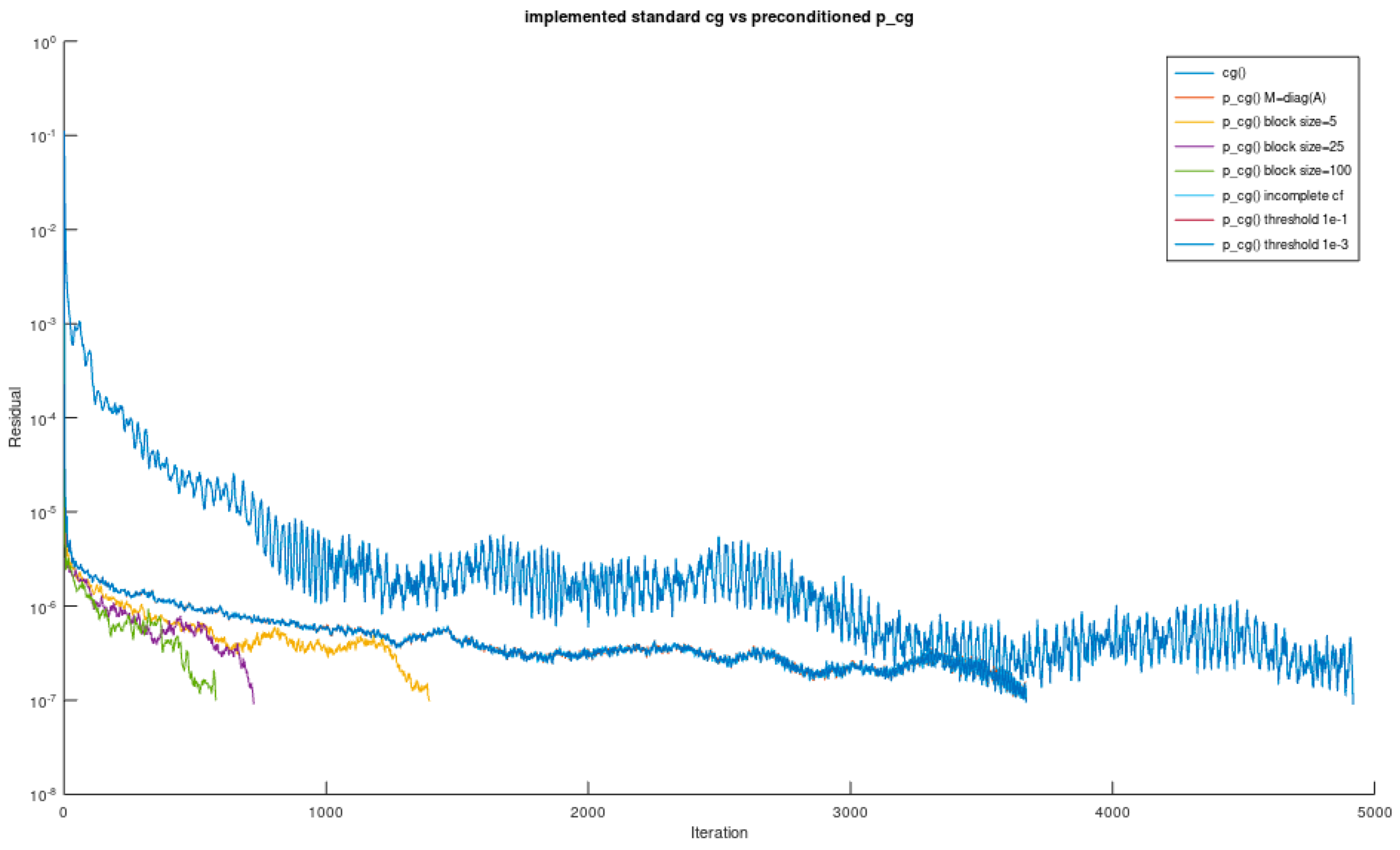
Results of nos6.mtx

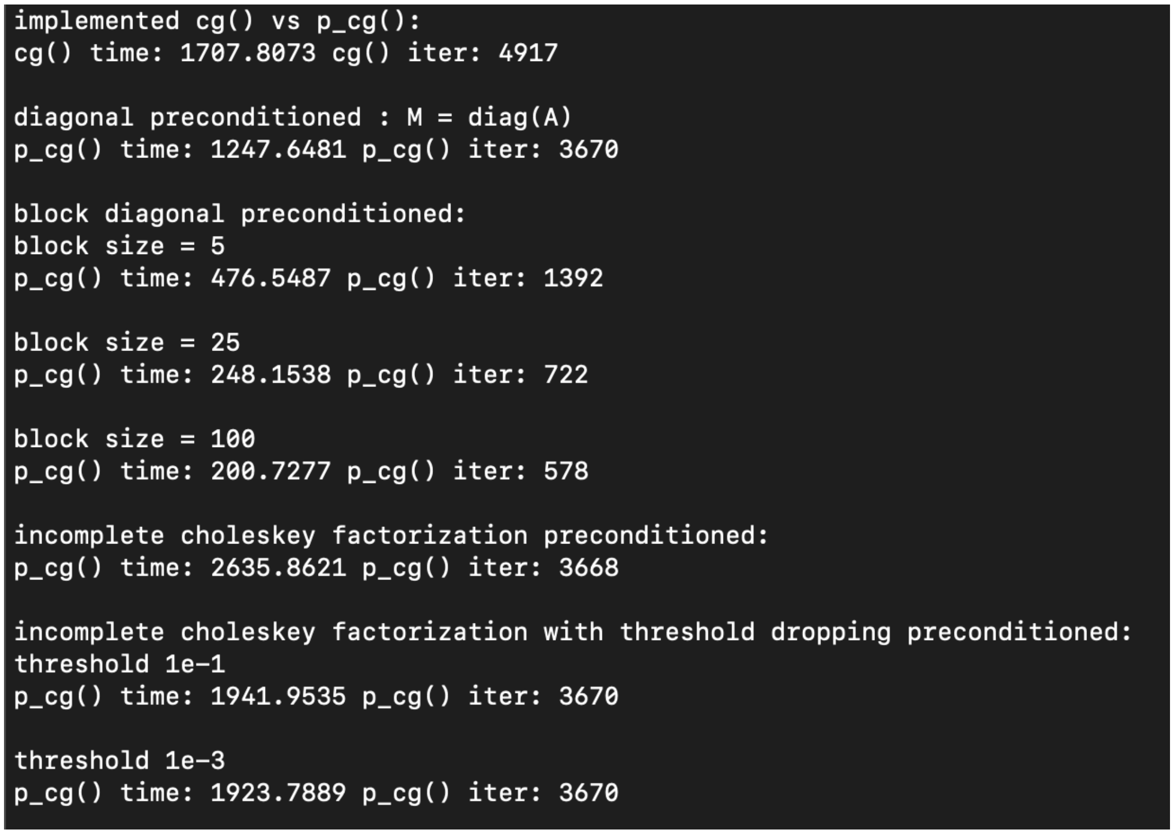




```
implemented cg() vs p_cg():
cg() time: 1.8851 cg() iter: 651

diagonal preconditioned : M = diag(A)
p_cg() time: 0.2777 p_cg() iter: 93

block diagonal preconditioned:
block size = 5
p_cg() time: 0.24463 p_cg() iter: 73

block size = 25
p_cg() time: 0.20512 p_cg() iter: 66

block size = 100
p_cg() time: 0.17623 p_cg() iter: 45

incomplete choleskey factorization preconditioned:
p_cg() time: 0.277 p_cg() iter: 29

incomplete choleskey factorization with threshold dropping preconditioned:
threshold 1e-1
p_cg() time: 0.11851 p_cg() iter: 29

threshold 1e-3
p_cg() time: 0.071263 p_cg() iter: 7
```

Results of s3rmt3m3.mtx





```
implemented cg() vs p_cg():
cg() time: 1707.8073 cg() iter: 4917

diagonal preconditioned : M = diag(A)
p_cg() time: 1247.6481 p_cg() iter: 3670

block diagonal preconditioned:
block size = 5
p_cg() time: 476.5487 p_cg() iter: 1392

block size = 25
p_cg() time: 248.1538 p_cg() iter: 722

block size = 100
p_cg() time: 200.7277 p_cg() iter: 578

incomplete choleskey factorization preconditioned:
p_cg() time: 2635.8621 p_cg() iter: 3668

incomplete choleskey factorization with threshold dropping preconditioned:
threshold 1e-1
p_cg() time: 1941.9535 p_cg() iter: 3670

threshold 1e-3
p_cg() time: 1923.7889 p_cg() iter: 3670
```

Review of the results:
As it is expected the cg() function consistently needs the most time and the most iterations until convergence, and precondition conjugate gradients are always faster, but they have a great degree of difference among themselves.

Diagonal matrix as precondition shortens the runtime and iterations compared to standard method, but next we can see that block diagonal of size 5 is actually really comparable with the normal diagonal, and only when block sizes of 25, and 100 are used we can see big improvement.

Next we take a look at the results of standard incomplete choleskey factorization and a version with threshold dropping.
For this assignment I chose 2 thresholds of 1e-1 and 1e-3, the results as you can see are comparable for standard incomplete choleskey factorization and threshold drop of 1e-1, but when 1e-3 is used runtime and iterations are shortened by quite a bit.

For computing choleskey factorizations of the last matrix I ran to a problem of negative pivot since the matrix was not diagonally dominant, so as a solution I defined an alpha and used ‘diagcomp' according to the documentation of ichol() (https://www.mathworks.com/help/matlab/ref/ichol.html)
Other interesting result that only happens for the last matrix is the closeness of runtime and iteration of all form of choleskey factorizations.

For the last matrix since it took so much longer for computation I changed too from 1e-10 to 1e-7.

I set the max iteration to number of rows of each matrix.