In order calculate our exponent p value for each method, we simply have to measure the slope of the lines present in the graph from part 2 (this is possible because we have a loglog plot).

We do this by using the **polyfit** function using our points (we apply the log function to fit our loglog plot) found in part 2. We run the following code:

```
var = polyfit(log(N(1:2)),log(number_of_jacobi_iterations(1:2)),1) % Since we only have 2 data points

var =

   0.897288158668149   1.506189992795032

p_value_for_jacobi_method = var(1)

p_value_for_jacobi_method =

   0.897288158668149

var = polyfit(log(N(1:3)),log(number_of_gauss_iterations(1:3)),1)

var =

   0.915509680934949   0.757689781532053

p_value_for_gauss_method = var(1)

p_value_for_gauss_method =

   0.915509680934949

var = polyfit(log(N),log(number_of_SOR_iterations),1)
```

```
var =

   0.471440639758865   1.096757604547755

p_value_for_SOR_method = var(1)

p_value_for_SOR_method =

   0.471440639758865

var = polyfit(log(N),log(number_of_cg_iterations),1)

var =

   0.523083274658625   0.039103344101659

p_value_for_cg_iterations = var(1)

p_value_for_cg_iterations =

   0.523083274658625

var = polyfit(log(N),log(number_of_pre_cg_iterations),1)

var =

   0.454435549913471   0.437816661433937

p_value_for_pre_cg_method = var(1)
```

p_value_for_pre_cg_method =

0.454435549913471

var = polyfit(log(N),log(number_of_pre_cg_ichol_iterations),1)

var =

0.335036328373983 0.612511526175062

p_value_for_pre_cg_ichol_method = var(1)

p_value_for_pre_cg_ichol_method =

0.335036328373983

In conclusion, the approximate exponent p for each method is:

Jacobi: 0.897288158668149
Gauss-Seidel: 0.915509680934949
SOR: 0.471440639758865
Conjugate Gradient: 0.523083274658625
Preconditioned CG: 0.454435549913471
Preconditioned CG w/ ichol: 0.335036328373983

As to why Jacobi might have a higher p value, it could be due to having more points.