

# analysis

February 21, 2025

## 1 Outlab 6

- Author: Kirill Shumilov

For all analysis done here SOR relaxation factor is 1.8

### 1.1 Setup

```
[1]: import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy as np

import json
```

```
[2]: with open('timing_info.json') as f:
    data = json.load(f)
    df_all = pd.DataFrame.from_records(data)

df_all['time'] /= 1000
df_all['algo'] = pd.Categorical(df_all['algo'])

O = lambda x, n, C, S: C * (x ** n) + S
```

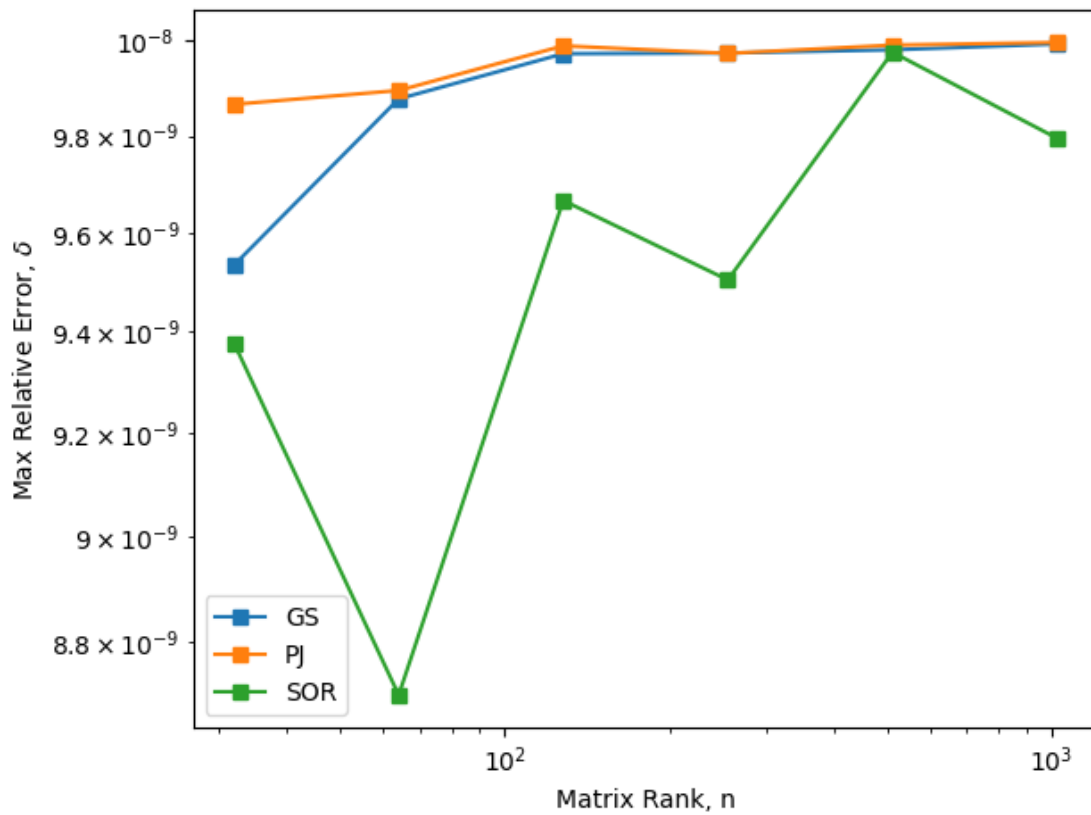
### 1.2 Relative Error, $\delta^\mu$

```
[3]: with pd.option_context('display.precision', 5, "display.expand_frame_repr",
    ↪False, 'display.float_format', lambda f: f'{f: 12.6e}'):
    print(pd.pivot(df_all[df_all.algo != 'LUP'], columns='n', index='algo',
    ↪values='relative_error'))
```

n	32	64	128	256	512
1024					
algo					
PJ	9.865678e-09	9.894863e-09	9.989114e-09	9.973093e-09	9.990600e-09
	9.996239e-09				
GS	9.534645e-09	9.877176e-09	9.972400e-09	9.973878e-09	9.980651e-09
	9.992983e-09				

SOR    9.377436e-09   8.702079e-09   9.666169e-09   9.504398e-09   9.975606e-09  
9.794969e-09

```
[4]: fig, ax = plt.subplots(1, 1, layout='tight', sharex=True)
for i, (algo, dfa) in enumerate(df_all[df_all.algo != 'LUP'].groupby('algo',
    observed=True)):
    ax.loglog('n', 'relative_error', 's-', data=dfa, label=algo)
ax.legend()
ax.set_xlabel('Matrix Rank, n')
ax.set_ylabel('Max Relative Error,  $\delta$ ')
plt.show()
```



It can be observed that the error is below tolerance error of  $1e-4$ . It is smallest for SOR and largest for PJ. The kinks in SOR plot can be explained by the fact that the error is dependent on relaxation factor, which itself depends on structure of the matrix, and potentially was not optimal for all  $n$ .

### 1.3 Execution Time

```
[5]: fig, ax = plt.subplots(1, 1, layout='tight', sharex=True)
for i, (algo, dfa) in enumerate(df_all.groupby('algo', observed=True)):
```

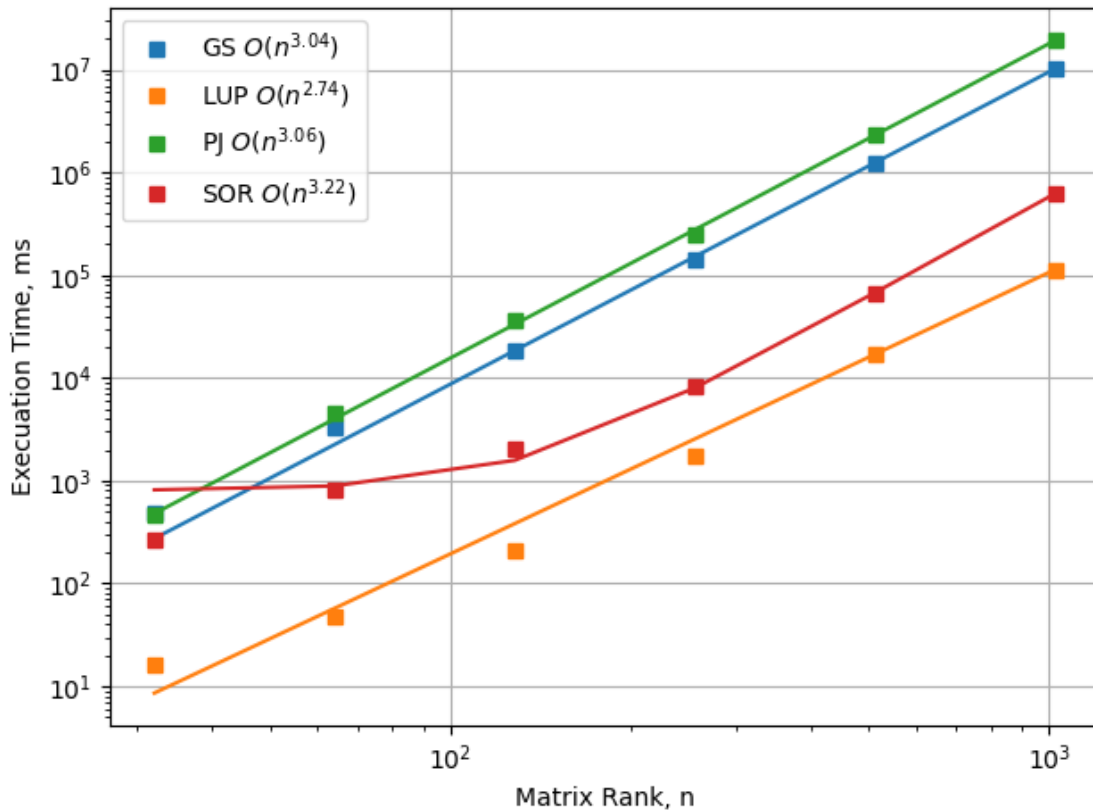
```

popt, pcov = curve_fit(0, dfa['n'], dfa['time'], p0=(2, 1e-2, 0),
↳ bounds=([0, 0, 0], [np.inf, np.inf, np.inf]))
ax.loglog('n', 'time', 's', data=dfa, label=' '.join([f'{algo}', r'$O(n^{\%3.
↳ 2f})$' % (popt[0])]), color=f'C{i}')
ax.loglog(dfa['n'], 0(dfa['n'], *popt), '-', color=f'C{i}')

ax.legend()
ax.set_xlabel('Matrix Rank, n')
ax.set_ylabel('Execution Time, ms')
ax.grid(True)

plt.show()

```



We can see that execution time scales roughly as  $O(n^3)$ , give or take, for all methods, as is expected, given that all methods rely on matrix multiplies. The approximately the same for all iterative methods. The primary difference comes from the constant in front of the scaling. It is smallest for LUP and largest for PJ.

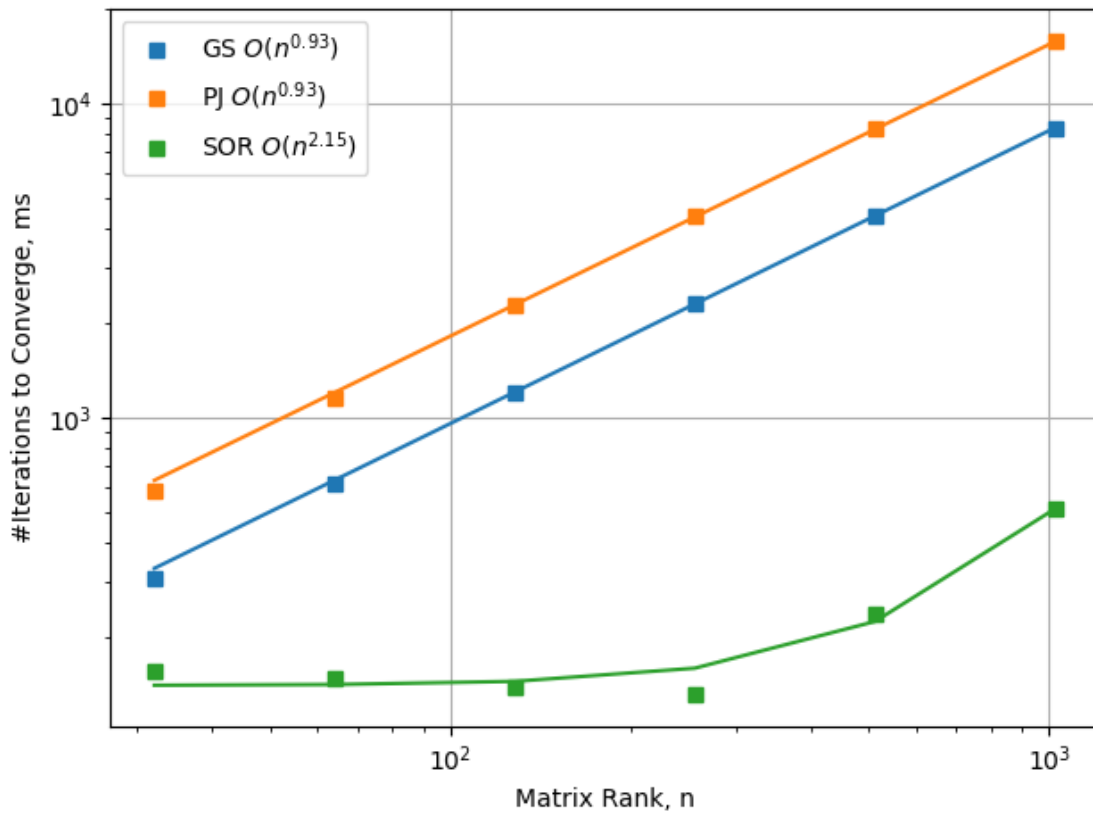
Fits are done using function:  $f(n) = C \times n^x + S$ , where  $C$  and  $S$  are constant.

## 1.4 Number of Iterations

```
[6]: fig, ax = plt.subplots(1, 1, layout='tight', sharex=True)
for i, (algo, dfa) in enumerate(df_all[df_all.algo != 'LUP'].groupby('algo',
    observed=True)):
    popt, pcov = curve_fit(0, dfa['n'], dfa['iterations'], p0=(2, 1e-2, 0),
    bounds=([0, 0, 0], [np.inf, np.inf, np.inf]))
    ax.loglog('n', 'iterations', 's', data=dfa, label=' '.join([f'{algo}',
    r'$O(n^{\%3.2f})$' % (popt[0]]), color=f'C{i}'))
    ax.loglog(dfa['n'], 0(dfa['n'], *popt), '-', color=f'C{i}')

ax.legend()
ax.set_xlabel('Matrix Rank, n')
ax.set_ylabel('#Iterations to Converge, ms')
ax.grid(True)

plt.show()
```



For GS and PJ, the scaling is essentially linear. For SOR, the scaling starts due to successful selection of relaxation factor, but turns linear as the system size grows.

Fits are done using function:  $f(n) = C \times n^x + S$ , where  $C$  and  $S$  are constant.