

# NaviFlow Analysis Script

This notebook analyzes NaviFlow simulation results from HDF5 files.

## Imports

```
import h5py
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scienceplots
from matplotlib.cm import coolwarm
import os

# Set up scienceplots style
plt.style.use(['science', 'ieee'])
```

## Helper functions

```
# Function to extract metadata from H5 files
def extract_metadata(file_path):
    with h5py.File(file_path, 'r') as f:
        metadata = {}

        # Get algorithm information
        if 'algorithm' in f:
            for key in f['algorithm'].attrs.keys():
                metadata[f'algorithm_{key}'] = f['algorithm'].attrs[key]

        # Get simulation information
        if 'simulation' in f:
            for key in f['simulation'].attrs.keys():
                metadata[f'simulation_{key}'] = f['simulation'].attrs[key]
            if 'mesh_size' in f['simulation']:
                for key in f['simulation']['mesh_size'].attrs.keys():
                    metadata[f'mesh_{key}'] = f['simulation']['mesh_size'].attrs[key]

        # Get solver information
        if 'momentum_solver' in f:
            for key in f['momentum_solver'].attrs.keys():
                metadata[f'momentum_solver_{key}'] = f['momentum_solver'].attrs[key]

        if 'pressure_solver' in f:
            for key in f['pressure_solver'].attrs.keys():
                metadata[f'pressure_solver_{key}'] = f['pressure_solver'].attrs[key]
```

```

# Get performance information
if 'performance' in f:
    for key in f['performance'].attrs.keys():
        metadata[f'performance_{key}'] = f['performance'].attrs[key]

# Add solver type based on file path
if 'pyamg' in file_path:
    metadata['solver_type'] = 'PyAMG'
elif 'preconditioned_cg' in file_path:
    metadata['solver_type'] = 'Preconditioned CG'

# Add file path
metadata['file_path'] = file_path

return metadata

# Function to plot convergence metrics
def plot_convergence_metric(files, metric, title, ylabel):
    plt.figure(figsize=(10, 6))

    # Get colors from coolwarm colormap
    colors = [coolwarm(i) for i in np.linspace(0, 1, len(files))]

    for idx, (file_path, color) in enumerate(zip(files, colors)):
        with h5py.File(file_path, 'r') as f:
            if 'residual_history' in f:
                hist_group = f['residual_history']
                iterations = hist_group['iteration'][:, :]

                # Get solver type from file path
                if 'pyamg' in file_path:
                    solver = 'PyAMG'
                elif 'preconditioned_cg' in file_path:
                    solver = 'Preconditioned CG'
                else:
                    solver = 'Unknown'

                # Plot the metric with error checking
                if metric in hist_group:
                    data = hist_group[metric][:]
                    print(f"Plotting {metric} for {solver}:")
                    print(f"  First value: {data[0]}")
                    print(f"  Last value: {data[-1]}")
                    print(f"  Min value: {np.min(data)}")
                    print(f"  Max value: {np.max(data)}")

                # Ensure data is not empty or all zeros

```

```

        if len(data) > 0 and not np.all(data == 0):
            plt.loglog(iterations, data, label=solver, color=color,
marker='o', markersize=4, markevery=5)
        else:
            print(f"Warning: {metric} data is empty or all zeros for
{solver}")
    else:
        print(f"Warning: {metric} not found in {file_path}")

plt.title(title)
plt.xlabel('Iteration')
plt.ylabel(ylabel)
plt.grid(True, which='both', ls='--', alpha=0.2)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Set y-axis limits if needed
plt.ylim(bottom=1e-10) # Adjust this value based on your data

return plt.gcf()

```

## Data loading and Processing

```

# Define file paths
file_paths = [
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/04 gauss_seidel/
results/SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/03 jacobi/
results/SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/06 AMG/results/
SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/07 AMG_CG/
results/SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/02 Conjugate
Gradient/results/SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/05
geo_multigrid/results/SIMPLE_Re100_mesh15x15_profile.h5',
    '/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/08 CG Matrix/
results/SIMPLE_Re100_mesh15x15_profile.h5'
]

# Extract metadata from all files
metadata_list = []
for file_path in file_paths:
    metadata = extract_metadata(file_path)
    metadata_list.append(metadata)

# Create DataFrame

```

```
df = pd.DataFrame(metadata_list)
```

```
# Display DataFrame
```

```
print("\nMetadata DataFrame:")
```

```
print(df)
```

```
Metadata DataFrame:
```

	algorithm_alpha_p	algorithm_alpha_u	simulation_algorithm	\
0	0.3	0.7	SimpleSolver	
1	0.3	0.7	SimpleSolver	
2	0.3	0.7	SimpleSolver	
3	0.3	0.7	SimpleSolver	
4	0.3	0.7	SimpleSolver	
5	0.3	0.7	SimpleSolver	
6	0.3	0.7	SimpleSolver	

	simulation_reynolds_number	simulation_timestamp	mesh_x	mesh_y	\
0	100	2025-04-09 16:50:23	15	15	
1	100	2025-04-09 16:49:17	15	15	
2	100	2025-04-09 16:51:12	15	15	
3	100	2025-04-09 16:51:34	15	15	
4	100	2025-04-09 17:08:33	15	15	
5	100	2025-04-09 17:43:39	15	15	
6	100	2025-04-09 16:51:53	15	15	

	momentum_solver_type	pressure_solver_max_iterations	\
0	StandardMomentumSolver	5000000	
1	StandardMomentumSolver	500000	
2	StandardMomentumSolver	100000	
3	StandardMomentumSolver	100000	
4	StandardMomentumSolver	1000000	
5	StandardMomentumSolver	100	
6	StandardMomentumSolver	100000	

	pressure_solver_tolerance	pressure_solver_type	\
0	1.000000e-07	GaussSeidelSolver	
1	1.000000e-07	JacobiSolver	
2	1.000000e-07	PyAMGSolver	
3	1.000000e-07	PreconditionedCGSolver	
4	1.000000e-05	MatrixFreeCGSolver	
5	1.000000e-07	MultiGridSolver	
6	1.000000e-07	ConjugateGradientSolver	

	performance_avg_time_per_iteration	performance_cpu_time	\
0	0.202733	21.346084	
1	0.257171	26.950761	

2	0.004262	0.439738
3	0.006030	0.625720
4	0.004763	0.495689
5	0.183480	19.316477
6	0.001980	0.204989

	performance_iterations	performance_total_time	\
0	106	21.489706	
1	106	27.260111	
2	106	0.451810	
3	106	0.639218	
4	106	0.504928	
5	106	19.448919	
6	106	0.209876	

	file_path	\
0	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
1	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
2	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
3	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
4	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
5	/Users/philipnickel/Documents/GitHub/NaviFlow/...	
6	/Users/philipnickel/Documents/GitHub/NaviFlow/...	

	pressure_solver_matrix_free
0	NaN
1	NaN
2	True
3	NaN
4	NaN
5	NaN
6	NaN

## LaTeX Table generation

```
# Create a clean DataFrame with all relevant columns
metadata_df = df[[
    'simulation_algorithm',
    'simulation_reynolds_number',
    'algorithm_alpha_p',
    'algorithm_alpha_u',
    'mesh_x',
    'mesh_y',
    'momentum_solver_type',
    'pressure_solver_type',
    'pressure_solver_tolerance',
    'performance_avg_time_per_iteration',
    'performance_cpu_time',
```

```

        'performance_iterations',
        'performance_total_time'
    ]].copy()

# Rename columns to be more readable
metadata_df.columns = [
    'Algorithm',
    'Re',
    ' $\alpha_p$ ',
    ' $\alpha_u$ ',
    'Mesh X',
    'Mesh Y',
    'Momentum Solver',
    'Pressure Solver',
    'Pressure Tolerance',
    'Time/Iter (s)',
    'CPU Time (s)',
    'Iterations',
    'Total Time (s)'
]

# Generate the LaTeX table with proper formatting
print("\begin{table}[htbp]\n\\centering\n\\resizebox{\\textwidth}{!}{\n" +
      metadata_df.to_latex(index=False, float_format=lambda x: f"{x:.2e}" if x
< 0.0001 or x > 10000 else f"{x:.3f}") +
      "\n\\caption{Solver Comparison}\n\\label{tab:solver_comparison}\n\n" +
      "\end{table}")

```

```

\begin{table}[htbp]
\centering
\resizebox{\textwidth}{!}{\begin{tabular}{llrrrrllrrrrr}
\toprule
Algorithm & Re &  $\alpha_p$  &  $\alpha_u$  & Mesh X & Mesh Y & Momentum Solver & Pressure Solver & Pressure Tolerance & Time/Iter (s) & CPU Time (s) & Iterations & Total Time (s) \\
\midrule
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver & GaussSeidelSolver & 1.00e-07 & 0.203 & 21.346 & 106 & 21.490 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver & JacobiSolver & 1.00e-07 & 0.257 & 26.951 & 106 & 27.260 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver & PyAMGSolver & 1.00e-07 & 0.004 & 0.440 & 106 & 0.452 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver & PreconditionedCGSolver & 1.00e-07 & 0.006 & 0.626 & 106 & 0.639 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver & MatrixFreeCGSolver & 1.00e-05 & 0.005 & 0.496 & 106 & 0.505 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver &

```

```

MultiGridSolver & 1.00e-07 & 0.183 & 19.316 & 106 & 19.449 \\
SimpleSolver & 100 & 0.300 & 0.700 & 15 & 15 & StandardMomentumSolver &
ConjugateGradientSolver & 1.00e-07 & 0.002 & 0.205 & 106 & 0.210 \\
\bottomrule
\end{tabular}
}
\caption{Solver Comparison}
\label{tab:solver_comparison}
\end{table}

```

metadata\_df

	Al- gorithm	Re αp	αu	Mesh X	Mesh Y	Momen- tum Solver	Pres- sure Solver	Pres- sure Tol- er- ance	Time Iter (s)	CPU Time (s)	Iter- ations	To- tal Time (s)
0	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- del Solver	Gauss- Sei- del Solver	1.000000e-07	207732	21.34608106	21.489706
1	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- bi Solver	Ja- co- bi Solver	1.000000e-07	257172	26.95074106	27.260111
2	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- tum Solver	PyAMG Solver	1.000000e-07	907262	2.439738106	0.451810
3	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- tioned- CG Solver	Pre- con- ditioned- CG Solver	1.000000e-07	907030	0.62572106	0.639218
4	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- tum Solver	MatrixFree Solver	1.000000e-07	907030	3.49568106	0.504928
5	SimpleSolver	100	0.3	0.7	15	15	Stan- dard- Grid- Solver	Multi- Grid- Solver	1.000000e-07	183480	19.31647106	19.448919

Al- gorithm	Re	$\alpha$ p	$\alpha$ u	Mesh X	Mesh Y	Momen- tumSolver	Pres- sure Solver	Pres- sure Tol- er- ance	Time Iter (s)	CPU Time (s)	Iter- ational	To- tal Time (s)
						Momen- tumSolver						
Sim- pleSolver	100	0.3	0.7	15	15	Stan- dard- Momen- tumSolver	Con- jugate- Gra- dientSolver	1.000000e-06	0.001980	0.204981	106	0.209876
6												

## Residual plots

```
# Create a figure with three subplots side by side
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

# Get colors from coolwarm colormap
colors = [coolwarm(i) for i in np.linspace(0, 1, len(file_paths))]

for idx, (file_path, color) in enumerate(zip(file_paths, colors)):
    with h5py.File(file_path, 'r') as f:
        # Get algorithm information
        if 'residual_history' in f:
            hist_group = f['residual_history']
            iterations = hist_group['iteration'][:, :]

        # Get solver type from file path
        if '06 amg' in file_path.lower():
            solver = 'AMG'
        elif '04 gauss_seidel' in file_path.lower():
            solver = 'Gauss_Seidel'
        elif '03 jacobi' in file_path.lower():
            solver = 'Jacobi'
        elif '01 basic_cavity' in file_path.lower():
            solver = 'Basic Cavity'
        elif '07 amg_cg' in file_path.lower():
            solver = 'AMG_CG'
        elif '02 conjugate gradient' in file_path.lower():
            solver = 'Conjugate Gradient'
        elif '05 geo_multigrid' in file_path.lower():
            solver = 'MultiGrid'
        elif '08 cg matrix' in file_path.lower():
            solver = 'CG Matrix'
```



```

else:
    solver = 'Unknown'

# Plot total residual
total_residual = hist_group['total_residual'][:]
ax1.loglog(iterations, total_residual, label=solver, color=color)

# Plot momentum solver residual
momentum_residual = hist_group['momentum_residual'][:]
ax2.loglog(iterations, momentum_residual, label=solver, color=color)

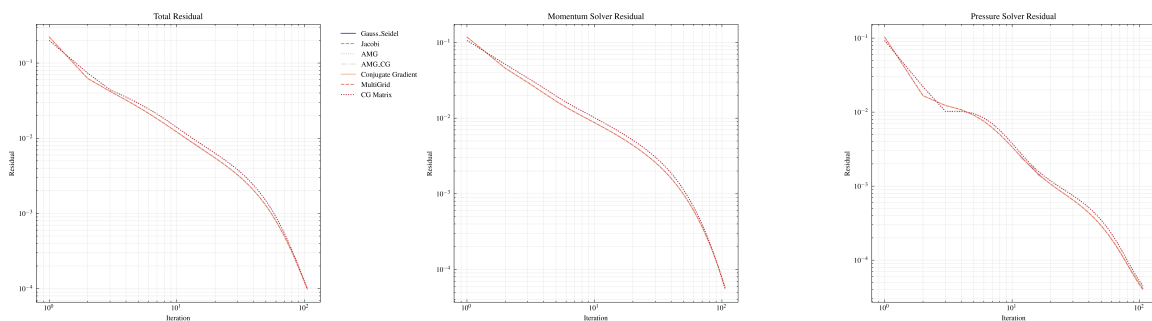
# Plot pressure solver residual
pressure_residual = hist_group['pressure_residual'][:]
ax3.loglog(iterations, pressure_residual, label=solver, color=color)

# Configure each subplot
for ax, title, ylabel in [(ax1, 'Total Residual', 'Residual'),
                          (ax2, 'Momentum Solver Residual', 'Residual'),
                          (ax3, 'Pressure Solver Residual', 'Residual')]:
    ax.set_title(title)
    ax.set_xlabel('Iteration')
    ax.set_ylabel(ylabel)
    ax.grid(True, which='both', ls='--', alpha=0.2)

# Add legend only to the first subplot
ax1.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.savefig('combined_residuals.pdf', bbox_inches='tight', dpi=300)
plt.show()

```



## Infinity norm plot

```

# First, let's check the data
for file_path in file_paths:
    with h5py.File(file_path, 'r') as f:
        if 'residual_history' in f:

```

```

hist_group = f['residual_history']
print(f"\nChecking {file_path}:")
print("Available metrics:", list(hist_group.keys()))
if 'infinity_norm_error' in hist_group:
    data = hist_group['infinity_norm_error'][:,]
    print(f"Infinity norm data shape: {data.shape}")
    print(f"First value: {data[0]}")
    print(f>Last value: {data[-1]}")
    print(f"Min value: {np.nanmin(data)}") # Use nanmin instead of
min
    print(f"Max value: {np.nanmax(data)}") # Use nanmax instead of
max

    print(f"Number of NaN values: {np.isnan(data).sum()}")
else:
    print("No infinity norm data found!")

# Now let's create a new plot
plt.figure(figsize=(12, 6))
# Plot the data
for idx, file_path in enumerate(file_paths):
    with h5py.File(file_path, 'r') as f:
        if 'residual_history' in f:
            hist_group = f['residual_history']
            iterations = hist_group['iteration'][:,]
            data = hist_group['infinity_norm_error'][:,]

            # Get solver type from the DataFrame based on index
            # Since we're processing files in the same order as the DataFrame
            solver_type = df.iloc[idx]['pressure_solver_type']

            # Map solver types to display names
            solver_map = {
                'DirectPressureSolver': 'Direct',
                'GaussSeidelSolver': 'Gauss-Seidel',
                'JacobiSolver': 'Jacobi',
                'PyAMGSolver': 'PyAMG'
            }

            solver_name = solver_map.get(solver_type, solver_type)

            # Get algorithm from filename
            filename = os.path.basename(file_path)
            algorithm = filename.split('_')[0] # e.g., 'SIMPLE'

            # Create label combining algorithm and solver
            label = f"{algorithm} - {solver_name}"

            # Remove NaN values and corresponding iterations

```

```

mask = ~np.isnan(data)
data = data[mask]
iterations = iterations[mask]

# Plot with a different color for each solver
color = coolwarm(idx / (len(file_paths) - 1))
plt.loglog(iterations, data, label=label, color=color, marker='o',
markersize=4, markevery=5)

# Configure the plot
plt.title('Infinity Norm Error')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.grid(True, which='both', ls='--', alpha=0.2)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Set y-axis limits if needed
#plt.ylim(bottom=1e-10) # Adjust this value based on your data

# Save and show the plot
plt.savefig('infinity_norm_error.pdf', bbox_inches='tight', dpi=300)
plt.show()

```

```

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/04
gauss_seidel/results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.35687709767804354
Min value: 0.35687709767804354
Max value: 0.56272529183987
Number of NaN values: 84

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/03_jacobi/
results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.3568771448917909
Min value: 0.3568771448917909
Max value: 0.5627252300469385
Number of NaN values: 84

```

```

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/06 AMG/
results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.40193712843224527
Min value: 0.40193712843224527
Max value: 0.48497207255615193
Number of NaN values: 95

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/07 AMG_CG/
results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.40193881663168396
Min value: 0.40193881663168396
Max value: 0.48497303758824284
Number of NaN values: 95

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/02
Conjugate Gradient/results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.3568632977684305
Min value: 0.3568632977684305
Max value: 0.4614326346339954
Number of NaN values: 95

Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/05
geo_multigrid/results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']
Infinity norm data shape: (106,)
First value: nan
Last value: 0.3568773793233655
Min value: 0.3568773793233655
Max value: 0.5627251769728479
Number of NaN values: 84

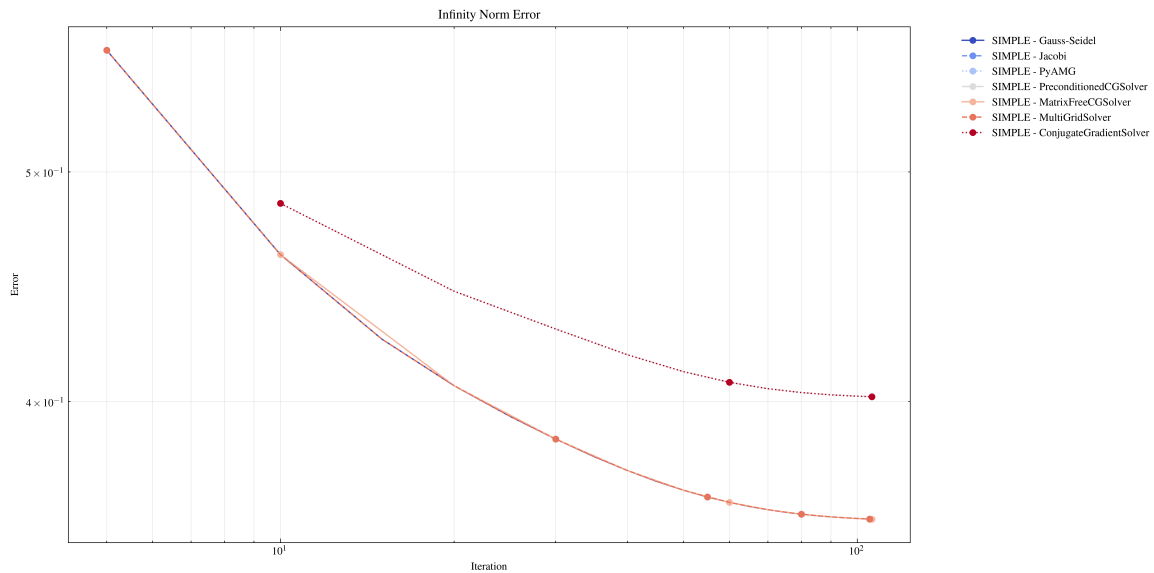
Checking /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/08 CG
Matrix/results/SIMPLE_Re100_mesh15x15_profile.h5:
Available metrics: ['cpu_time', 'infinity_norm_error', 'iteration',
'momentum_residual', 'pressure_residual', 'total_residual', 'wall_time']

```

```

Infinity norm data shape: (106,)
First value: nan
Last value: 0.4019350866642273
Min value: 0.4019350866642273
Max value: 0.4849693907323127
Number of NaN values: 95

```



```

# First, print the DataFrame to see what's in it
print("DataFrame contents:")
print(df)

# Print the file paths we're using
print("\nFile paths:")
for file_path in file_paths:
    print(file_path)

# Plot the data
for idx, file_path in enumerate(file_paths):
    with h5py.File(file_path, 'r') as f:
        if 'residual_history' in f:
            hist_group = f['residual_history']
            iterations = hist_group['iteration'][: ]
            data = hist_group['infinity_norm_error'][: ]

    # Debug: Print the file path and check if it's in the DataFrame
    print(f"\nProcessing file: {file_path}")
    matching_rows = df[df['file_path'] == file_path]
    print(f"Matching rows in DataFrame: {len(matching_rows)}")

```

```

if len(matching_rows) > 0:
    solver_name = matching_rows['solver_type'].iloc[0]
    print(f"Found solver name: {solver_name}")
else:
    # If not found in DataFrame, extract from path
    path_parts = file_path.split('/')
    solver_type = path_parts[1] if len(path_parts) > 1 else "Unknown"

    # Map directory names to proper solver names
    solver_map = {
        'pyamg': 'PyAMG',
        'gauss_seidel': 'Gauss-Seidel',
        'jacobi': 'Jacobi',
        'basic_cavity': 'Basic Cavity',
        'preconditioned_cg': 'Preconditioned CG'
    }

    solver_name = solver_map.get(solver_type, solver_type.replace('_',
' ').title())
    print(f"Extracted solver name from path: {solver_name}")

    # Get algorithm from filename
    filename = os.path.basename(file_path)
    algorithm = filename.split('_')[0] # e.g., 'SIMPLE'

    # Create label combining algorithm and solver
    label = f"{algorithm} - {solver_name}"
    print(f"Final label: {label}")

    # Remove NaN values and corresponding iterations
    mask = ~np.isnan(data)
    data = data[mask]
    iterations = iterations[mask]

    # Plot with a different color for each solver
    color = coolwarm(idx / (len(file_paths) - 1))
    plt.loglog(iterations, data, label=label, color=color, marker='o',
markersize=4, markevery=5)

```

DataFrame contents:

	algorithm_alpha_p	algorithm_alpha_u	simulation_algorithm \
0	0.3	0.7	SimpleSolver
1	0.3	0.7	SimpleSolver
2	0.3	0.7	SimpleSolver
3	0.3	0.7	SimpleSolver
4	0.3	0.7	SimpleSolver
5	0.3	0.7	SimpleSolver

6	0.3	0.7	SimpleSolver
---	-----	-----	--------------

	simulation_reynolds_number	simulation_timestamp	mesh_x	mesh_y	\
0	100	2025-04-09 16:50:23	15	15	
1	100	2025-04-09 16:49:17	15	15	
2	100	2025-04-09 16:51:12	15	15	
3	100	2025-04-09 16:51:34	15	15	
4	100	2025-04-09 17:08:33	15	15	
5	100	2025-04-09 17:43:39	15	15	
6	100	2025-04-09 16:51:53	15	15	

	momentum_solver_type	pressure_solver_max_iterations	\
0	StandardMomentumSolver	5000000	
1	StandardMomentumSolver	500000	
2	StandardMomentumSolver	100000	
3	StandardMomentumSolver	100000	
4	StandardMomentumSolver	1000000	
5	StandardMomentumSolver	100	
6	StandardMomentumSolver	100000	

	pressure_solver_tolerance	pressure_solver_type	\
0	1.000000e-07	GaussSeidelSolver	
1	1.000000e-07	JacobiSolver	
2	1.000000e-07	PyAMGSolver	
3	1.000000e-07	PreconditionedCGSolver	
4	1.000000e-05	MatrixFreeCGSolver	
5	1.000000e-07	MultiGridSolver	
6	1.000000e-07	ConjugateGradientSolver	

	performance_avg_time_per_iteration	performance_cpu_time	\
0	0.202733	21.346084	
1	0.257171	26.950761	
2	0.004262	0.439738	
3	0.006030	0.625720	
4	0.004763	0.495689	
5	0.183480	19.316477	
6	0.001980	0.204989	

	performance_iterations	performance_total_time	\
0	106	21.489706	
1	106	27.260111	
2	106	0.451810	
3	106	0.639218	
4	106	0.504928	
5	106	19.448919	
6	106	0.209876	

	file_path	\
--	-----------	---

```

0 /Users/philipnickel/Documents/GitHub/NaviFlow/...
1 /Users/philipnickel/Documents/GitHub/NaviFlow/...
2 /Users/philipnickel/Documents/GitHub/NaviFlow/...
3 /Users/philipnickel/Documents/GitHub/NaviFlow/...
4 /Users/philipnickel/Documents/GitHub/NaviFlow/...
5 /Users/philipnickel/Documents/GitHub/NaviFlow/...
6 /Users/philipnickel/Documents/GitHub/NaviFlow/...

pressure_solver_matrix_free
0 NaN
1 NaN
2 True
3 NaN
4 NaN
5 NaN
6 NaN

File paths:
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/04 gauss_seidel/
results/SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/03 jacobi/results/
SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/06 AMG/results/
SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/07 AMG_CG/results/
SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/02 Conjugate
Gradient/results/SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/05 geo_multigrid/
results/SIMPLE_Re100_mesh15x15_profile.h5
/Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/08 CG Matrix/
results/SIMPLE_Re100_mesh15x15_profile.h5

Processing file: /Users/philipnickel/Documents/GitHub/NaviFlow/main_scripts/04
gauss_seidel/results/SIMPLE_Re100_mesh15x15_profile.h5
Matching rows in DataFrame: 1

```

```

KeyError: 'solver_type'
[0;31m-----
[0m
[0;31mKeyError [0m                                Traceback (most recent call
last)
File [0;32m/opt/homebrew/Caskroom/miniconda/base/envs/cfdeeznutz/
lib/python3.11/site-packages/pandas/core/indexes/base.py:3791 [0m, in
[0;36mIndex.get_loc [0;34m(self, key) [0m
[1;32m 3790 [0m [38;5;28;01mtry [39;00m:
[0;32m-> 3791 [0m                                [38;5;28;01mreturn [39;00m

```



```
[38;5;28;43mself [39;49m [38;5;241;43m. [39;49m [43m_engine [49m [38;5;241;43m.
[39;49m [43mget_loc [49m [43m( [49m [43mcasted_key [49m [43m) [49m
[1;32m      3792 [0m [38;5;28;01mexcept [39;00m [38;5;167;01mKeyError [39;00m
[38;5;28;01mas [39;00m err:
```

```
File [0;32mindex.pyx:152 [0m, in
[0;36mpandas._libs.index.IndexEngine.get_loc [0;34m() [0m
```

```
File [0;32mindex.pyx:181 [0m, in
[0;36mpandas._libs.index.IndexEngine.get_loc [0;34m() [0m
```

```
File [0;32mpandas/_libs/hashtable_class_helper.pxi:7080 [0m, in
[0;36mpandas._libs.hashtable.PyObjectHashTable.get_item [0;34m() [0m
```

```
File [0;32mpandas/_libs/hashtable_class_helper.pxi:7088 [0m, in
[0;36mpandas._libs.hashtable.PyObjectHashTable.get_item [0;34m() [0m
```

```
[0;31mKeyError [0m: 'solver_type'
```

The above exception was the direct cause of the following exception:

```
[0;31mKeyError [0m Traceback (most recent call
last)
```

```
Cell [0;32mIn[154], line 24 [0m
```

```
[1;32m 21 [0m
```

```
[38;5;28mprint [39m( [38;5;124mf [39m [38;5;124m" [39m [38;5;124mMatching rows
in DataFrame: [39m [38;5;132;01m{ [39;00m [38;5;28mlen [39m(matching_rows)
[38;5;132;01m} [39;00m [38;5;124m" [39m)
```

```
[1;32m 23 [0m [38;5;28;01mif [39;00m [38;5;28mlen [39m(matching_rows)
```

```
[38;5;241m> [39m [38;5;241m0 [39m:
```

```
[0;32m--> 24 [0m solver_name [38;5;241m= [39m [43mmatching_rows [49m [43m[ [49m [38;5;124;43m' [3
```

```
[49m [38;5;241m. [39miloc[ [38;5;241m0 [39m]
```

```
[1;32m 25 [0m
```

```
[38;5;28mprint [39m( [38;5;124mf [39m [38;5;124m" [39m [38;5;124mFound
solver name: [39m [38;5;132;01m{ [39;00msolver_name [38;5;132;01m}
[39;00m [38;5;124m" [39m)
```

```
[1;32m 26 [0m [38;5;28;01melse [39;00m:
```

```
[1;32m 27 [0m [38;5;66;03m# If not found in DataFrame, extract from
path [39;00m
```

```
File [0;32m/opt/homebrew/Caskroom/miniconda/base/envs/cfdeeznutz/
lib/python3.11/site-packages/pandas/core/frame.py:3893 [0m, in
[0;36mDataFrame.__getitem__ [0;34m(self, key) [0m
```

```
[1;32m 3891 [0m [38;5;28;01mif [39;00m [38;5;28mself [39m [38;5;241m.
```

```
[39mcolumns [38;5;241m. [39mnlevels [38;5;241m> [39m [38;5;241m1 [39m:
```

```
[1;32m 3892 [0m [38;5;28;01mreturn [39;00m [38;5;28mself [39m [38;5;241m.
```

```
[39m_getitem_multilevel(key)
```

```
[0;32m-> 3893 [0m indexer [38;5;241m= [39m
```

```

[38;5;28;43mself [39;49m [38;5;241;43m. [39;49m [43mcolumns [49m [38;5;241;43m.
[39;49m [43mget_loc [49m [43m( [49m [43mkey [49m [43m) [49m
[1;32m 3894 [0m [38;5;28;01mif [39;00m is_integer(indexer):
[1;32m 3895 [0m indexer [38;5;241m= [39m [indexer]

File [0;32m/opt/homebrew/Caskroom/miniconda/base/envs/cfdeeznutz/
lib/python3.11/site-packages/pandas/core/indexes/base.py:3798 [0m, in
[0;36mIndex.get_loc [0;34m(self, key) [0m
[1;32m 3793 [0m [38;5;28;01mif [39;00m [38;5;28misinstance [39m(casted_key,
[38;5;28mslice [39m) [38;5;129;01mor [39;00m (
[1;32m 3794 [0m [38;5;28misinstance [39m(casted_key, abc [38;5;241m.
[39mIterable)
[1;32m 3795 [0m [38;5;129;01mand [39;00m
[38;5;28many [39m( [38;5;28misinstance [39m(x, [38;5;28mslice [39m)
[38;5;28;01mfor [39;00m x [38;5;129;01min [39;00m casted_key)
[1;32m 3796 [0m ):
[1;32m 3797 [0m [38;5;28;01mraise [39;00m InvalidIndexError(key)
[0;32m-> 3798 [0m [38;5;28;01mraise [39;00m
[38;5;167;01mKeyError [39;00m(key) [38;5;28;01mfrom [39;00m [38;5;250m
[39m [38;5;21;01merr [39;00m
[1;32m 3799 [0m [38;5;28;01mexcept [39;00m [38;5;167;01mTypeError [39;00m:
[1;32m 3800 [0m [38;5;66;03m# If we have a listlike key,
_check_indexing_error will raise [39;00m
[1;32m 3801 [0m [38;5;66;03m# InvalidIndexError. Otherwise we fall through
and re-raise [39;00m
[1;32m 3802 [0m [38;5;66;03m# the TypeError. [39;00m
[1;32m 3803 [0m [38;5;28mself [39m [38;5;241m.
[39m_check_indexing_error(key)

[0;31mKeyError [0m: 'solver_type'

```