prob1

January 20, 2025

[10]:
```python
import concurrent.futures
import time

# run with cuda
import cupy as cp
import cupyx.scipy.sparse as sp
import cupyx.scipy.sparse.linalg as splinalg
cp.get_default_memory_pool().free_all_blocks()

# for running without cuda
# import numpy as cp
# import scipy.sparse as sp
# import scipy.sparse.linalg as splinalg
```

[11]:
```python
def timed_problem(problem, matrix_size):
    A, b, solver = problem(matrix_size)
    # we aren't counting the time it takes to construct the matrix
    start_time = time.time()
    solver(A, b)
    end_time = time.time()
    cp.get_default_memory_pool().free_all_blocks()
    return end_time - start_time


def run_problems(
    problems: list, matrix_sizes=[10**i for i in range(1, 5)], timeout=60 * 5
):
    run_times = {
        problem.__name__: {matrix_size: None for matrix_size in matrix_sizes}
        for problem in problems
    }

    for problem in problems:
        print(f"Running {problem.__name__} \n")
        # I don't want the problems to run in parallel
        with concurrent.futures.ThreadPoolExecutor(max_workers=1) as executor:
            futures = {
```

```
                matrix_size: executor.submit(timed_problem, problem,␣
↪matrix_size)
                for matrix_size in matrix_sizes
            }

            for future in concurrent.futures.as_completed(futures.values()):
                matrix_size = next(
                    key for key, value in futures.items() if value == future
                )
                matrix_size_text = f"{matrix_size:.0e}"
                try:
                    run_time = future.result(timeout=timeout)
                    print(f"Finished for matrix size {matrix_size_text} in␣
↪{run_time} seconds")
                    run_times[problem.__name__][matrix_size] = run_time
                except concurrent.futures.TimeoutError:
                    print(f"Timeout for matrix size {matrix_size_text}")
                    break
                except Exception as e:
                    print(f"Exception for matrix size {matrix_size_text}: {e}")
                    break

    return run_times

run_times = {}
```

[12]:
```
def prob1(matrix_size):
    """
    1. Diagonal matrix: for N  [10, 109] in factors of 10 until compute time
    seems unreasonable.
    • Lx and Ux are 0
    • D0 to DN and b0 to bN are 1

    This problem can be solved algebraically. Di = bi for all i.
    """
    A = sp.eye(matrix_size, format="csr")
    b = cp.ones(matrix_size)
    return A, b, splinalg.spsolve_triangular

run_times.update(run_problems([prob1], matrix_sizes=[10**i for i in range(1,␣
↪9)]))
```

```
Running prob1

Finished for matrix size 1e+01 in 0.0 seconds
Finished for matrix size 1e+02 in 0.007524013519287109 seconds
Finished for matrix size 1e+03 in 0.0010099411010742188 seconds
Finished for matrix size 1e+04 in 0.0009906291961669922 seconds
```

```
Finished for matrix size 1e+05 in 0.0014879703521728516 seconds
Finished for matrix size 1e+06 in 0.008454561233520508 seconds
Finished for matrix size 1e+07 in 0.22781896591186523 seconds
Finished for matrix size 1e+08 in 0.36722874641418457 seconds
```

[13]:
```python
def prob2(matrix_size):
    """
    2. Lower triangular matrix: for N  [10, 109] in factors of 10 until compute
    time seems unreasonable.
    • U1 to UN are 0
    • LA is -1/A for A  [1, N ]
    • D1 to DN is one minus the sum of LA in the row
    • b0 to bN are 1
    """
    # Create diagonal values
    diag_values = {i: -1 / i for i in range(1, matrix_size)}

    # Create diagonals
    diags = {i: cp.full((matrix_size - i), value) for i, value in diag_values.
 ↪items()}
    diags[0] = cp.cumsum(cp.array([1] + [-1 * i for i in diag_values.values()]))

    # Create the lower triangular matrix
    mat = sp.tril(
        sp.diags(
            list(diags.values()),
            -1 * cp.array(list(diags.keys())),
            format="csr",
            dtype=cp.float32,
        )
    )

    b = cp.ones(matrix_size, dtype=cp.float32)

    return mat, b, splinalg.spsolve_triangular

prob2_results = run_problems([prob2])
run_times.update(prob2_results)
```

```
Running prob2

Finished for matrix size 1e+01 in 0.010002613067626953 seconds
Finished for matrix size 1e+02 in 0.0284731388092041 seconds
Finished for matrix size 1e+03 in 0.030472993850708008 seconds
Finished for matrix size 1e+04 in 4.286558389663696 seconds
```

[14]:
```python
def prob3(matrix_size):
    """
```

```
    3. Upper-Triangular matrix: for N  [10, 109] in factors of 10 until compute
    time seems unreasonable.
    • L1 to LN are 0
    • UA is -1/A for A  [1, N ]
    • D1 to DN is one minus the sum of UA in the row
    • b0 to bN are 1
    """
    # Create diagonal values
    diag_values = {i: -1 / i for i in range(1, matrix_size)}

    # Create diagonals
    diags = {i: cp.full((matrix_size - i), value) for i, value in diag_values.
 ↪items()}
    diags[0] = cp.cumsum(cp.array([1] + [-1 * i for i in diag_values.
 ↪values()]))[::-1]

    # Create the lower triangular matrix
    mat = sp.diags(
        list(diags.values()),
        1 * cp.array(list(diags.keys())),
        format="csr",
        dtype=cp.float32,
    )

    b = cp.ones(matrix_size, dtype=cp.float32)

    return mat, b, splinalg.spsolve_triangular


prob3_results = run_problems([prob3])
run_times.update(prob3_results)
```

```
Running prob3

Finished for matrix size 1e+01 in 0.0005018711090087891 seconds
Finished for matrix size 1e+02 in 0.0 seconds
Finished for matrix size 1e+03 in 0.0010006427764892578 seconds
Finished for matrix size 1e+04 in 0.004502773284912109 seconds
```

[15]:
```
def prob4(matrix_size):
    """
    4. Tridiagonal matrix: for N  [10, 109] in factors of 10 until compute time
    seems unreasonable.
    • L1 to LN are -1
    • U1 to UN are -1
    • D1 to DN are 3
    • b0 to bN are 1
```

```
    """
    value_map = {-1 : -1, 0 : 3, 1 : -1}
    mat = sp.diags(
        list(value_map.values()),
        list(value_map.keys()),
        shape=(matrix_size, matrix_size),
        format="csr",
    )
    b = cp.ones(matrix_size)

    return mat, b, splinalg.spsolve

prob4_results = run_problems([prob4], matrix_sizes=[10**i for i in range(1, 7)])
run_times.update(prob4_results)
```

```
Running prob4

Finished for matrix size 1e+01 in 0.005156993865966797 seconds
Finished for matrix size 1e+02 in 0.0016646385192871094 seconds
Finished for matrix size 1e+03 in 0.01132059097290039 seconds
Finished for matrix size 1e+04 in 0.10631823539733887 seconds
Finished for matrix size 1e+05 in 1.092623233795166 seconds
Finished for matrix size 1e+06 in 11.00407862663269 seconds
```

[16]:
```
def prob5(matrix_size):
    """
    5. Banded matrix: for N  [10, 109] in factors of 10 until compute time␣
 ↪seems
    unreasonable.
    • U1, U5, L1, and L5 are -1
    • D1 to DN is 5
    • b0 to bN are 1
    • The rest of U and L are zero.
    """
    value_map = {-1: -1, -5 : -1, 1 : -1, 5 : -1, 0 : 5}
    mat = sp.diags(
        list(value_map.values()),
        list(value_map.keys()),
        shape=(matrix_size, matrix_size),
        format="csr",
    )
    b = cp.ones(matrix_size)

    return mat, b, splinalg.spsolve


prob5_results = run_problems([prob5], matrix_sizes=[10**i for i in range(1, 7)])
```

```
run_times.update(prob5_results)
```

```
Running prob5

Finished for matrix size 1e+01 in 0.001978635787963867 seconds
Finished for matrix size 1e+02 in 0.002065420150756836 seconds
Finished for matrix size 1e+03 in 0.01174616813659668 seconds
Finished for matrix size 1e+04 in 0.11741328239440918 seconds
Finished for matrix size 1e+05 in 1.1982793807983398 seconds
Finished for matrix size 1e+06 in 11.911452054977417 seconds
```

[18]:
```
def prob6(matrix_size):
    """
    7. Upwind matrix: for N  [10, 109] in factors of 10 until compute time␣
 ↪seems
    unreasonable.
    • U1 to UN and L2 to LN are 0
    • L1 = -0.9
    • D1 to DN is 1
    • b1 to bN are 0, but b0 = 1
    """
    value_map = {-1: -0.9, 0 : 1}
    A = sp.diags(
        list(value_map.values()),
        list(value_map.keys()),
        shape=(matrix_size, matrix_size),
        format="csr",
    )
    b = cp.zeros(matrix_size)
    b[0] = 1

    return A, b, splinalg.spsolve

prob6_results = run_problems([prob6], matrix_sizes=[10**i for i in range(1, 8)])
run_times.update(prob6_results)
```

```
Running prob6

Finished for matrix size 1e+01 in 0.010062694549560547 seconds
Finished for matrix size 1e+02 in 0.017606019973754883 seconds
Finished for matrix size 1e+03 in 0.10295820236206055 seconds
Finished for matrix size 1e+04 in 0.277296781539917 seconds
Finished for matrix size 1e+05 in 1.1173458099365234 seconds
Finished for matrix size 1e+06 in 10.917600870132446 seconds
Finished for matrix size 1e+07 in 179.83606696128845 seconds
```