04_regression_matrix_operations

May 7, 2025

1 Parallelisation of Matrix Operations

```
[ ]: import os
     user = os.getenv('USER')
     os.chdir(f'/scratch/cd82/{user}/notebooks')
```

1.0.1 The reason why matrix operations are important

In regression, we often use the following computation to solve for β:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

For *large* datasets X, or *many* smaller ones, these operation can take a long time.

1.1 BLAS and LAPACK libraries

The underlying libraries that are used by the Python NumPy library (and many other data analysis packages) are based on the BLAS and LAPACK software library interfaces. These interfaces are designed to provide a generic interface to linear algebra functions and alow hardware vendors to provide optimised versions of the libraries that perform well on their hardware.

BLAS libraries provide various fundamental vector-vector, vector-matrix and matrix-matrix operations and LAPACK provides various matrix decompostion, factorisation and solver routines for various matrix types. The library specifies its vector and matrix operations using BLAS functions.

```
[1]: import numpy as np
     np.show_config()
```

```
Build Dependencies:
  blas:
    detection method: pkgconfig
    found: true
    include directory: /opt/_internal/cpython-3.10.15/lib/python3.10/site-
packages/scipy_openblas64/include
    lib directory: /opt/_internal/cpython-3.10.15/lib/python3.10/site-
packages/scipy_openblas64/lib
    name: scipy-openblas
    openblas configuration: OpenBLAS 0.3.28  USE64BITINT DYNAMIC_ARCH
```

```
NO_AFFINITY
      Haswell MAX_THREADS=64
    pc file directory: /project/.openblas
    version: 0.3.28
  lapack:
    detection method: pkgconfig
    found: true
    include directory: /opt/_internal/cpython-3.10.15/lib/python3.10/site-
packages/scipy_openblas64/include
    lib directory: /opt/_internal/cpython-3.10.15/lib/python3.10/site-
packages/scipy_openblas64/lib
    name: scipy-openblas
    openblas configuration: OpenBLAS 0.3.28  USE64BITINT DYNAMIC_ARCH
NO_AFFINITY
      Haswell MAX_THREADS=64
    pc file directory: /project/.openblas
    version: 0.3.28
Compilers:
  c:
    commands: cc
    linker: ld.bfd
    name: gcc
    version: 10.2.1
  c++:
    commands: c++
    linker: ld.bfd
    name: gcc
    version: 10.2.1
  cython:
    commands: cython
    linker: cython
    name: cython
    version: 3.0.12
Machine Information:
  build:
    cpu: x86_64
    endian: little
    family: x86_64
    system: linux
  host:
    cpu: x86_64
    endian: little
    family: x86_64
    system: linux
Python Information:
  path: /tmp/build-env-k6_20yf7/bin/python
  version: '3.10'
SIMD Extensions:
```

```
baseline:
- SSE
- SSE2
- SSE3
found:
- SSSE3
- SSE41
- POPCNT
- SSE42
- AVX
- F16C
- FMA3
- AVX2
- AVX512F
- AVX512CD
- AVX512_SKX
- AVX512_CLX
- AVX512_CNL
- AVX512_ICL
not found:
- AVX512_KNL
- AVX512_KNM
```

1.1.1 Controlling the number of threads

The BLAS libraries are controlled using environment variables that are sent in the shell being used to run Python.

e.g.
- **OMP_NUM_THREADS** The 'generic' variable used to set the number of threads. - **OPENBLAS_NUM_THREADS** - **MKL_NUM_THREADS** Used for setting Intel's MKL library

OMP_NUM_THREADS should typically be available. It is the generic variable used by the OpenMP library, which is the library that is typically used for parallelisation.

Please note, the variables controlling the number of threads to use need to be set in the environment before the code is run.

[ ]:
```python
import os
user = os.getenv('USER')
os.chdir(f'/scratch/cd82/{user}')
```

[2]:
```python
# The following script will be saved to the filesystem and run in a seperate␣
 ↪process,
# alowing us to change the environment to change the number of threads used.
script = """
import numpy as np
import time
import os

ob_nthreads = os.getenv('OPENBLAS_NUM_THREADS')
mkl_nthreads = os.getenv('MKL_NUM_THREADS')
omp_nthreads = os.getenv('OMP_NUM_THREADS')

print(f"Threads: OpenBLAS: {ob_nthreads} MKL: {mkl_nthreads}  OMP:␣
 ↪{omp_nthreads} ")

threads = {ob_nthreads, mkl_nthreads, omp_nthreads}
# select a value that is not 'None'
nthreads = [num for num in threads if num is not None]

nthreads=max(nthreads)

# np.show_config()
N=4000

# Create two numpy arrays
array1 = np.random.rand(N, N)
array2 = np.random.rand(N, N)

# Start the timer
start_time = time.time()

# Multiply the arrays
result = np.dot(array1, array2)

# Stop the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time

print(f"Time taken to multiply the arrays on {nthreads} cores: {elapsed_time}␣
 ↪seconds")
"""
with open("matrixmult.py", "w") as file:
    # Write the string to the file
    file.write(script)
```

1.1.2 Timing of larger problems

Sending larger computations to the supercomputer queue The environemnt variable NCPUS is available on NCI machines that are running the PBS batch system.

```
[3]: !echo "Number of cores allocated: $NCPUS"
```

```
Number of cores allocated:
```

```
[5]: import subprocess
     import os
     os.environ['OPENBLAS_NUM_THREADS'] = '1'  # Set the number of threads to 1
     subprocess.run(['python', 'matrixmult.py'])
```

```
Threads: OpenBLAS: 1 MKL: None  OMP: None
Time taken to multiply the arrays on 1 cores: 12.476385354995728 seconds
```

```
[5]: CompletedProcess(args=['python', 'matrixmult.py'], returncode=0)
```

```
[6]: os.environ['OPENBLAS_NUM_THREADS'] = '2'  # Set the number of threads to 2
     subprocess.run(['python', 'matrixmult.py'])
```

```
Threads: OpenBLAS: 2 MKL: None  OMP: None
Time taken to multiply the arrays on 2 cores: 6.897268295288086 seconds
```

```
[6]: CompletedProcess(args=['python', 'matrixmult.py'], returncode=0)
```

```
[7]: os.environ['OPENBLAS_NUM_THREADS'] = '4'  # Set the number of threads to 4
     subprocess.run(['python', 'matrixmult.py'])
```

```
Threads: OpenBLAS: 4 MKL: None  OMP: None
Time taken to multiply the arrays on 4 cores: 4.938734769821167 seconds
```

```
[7]: CompletedProcess(args=['python', 'matrixmult.py'], returncode=0)
```

```
[ ]:
```