Lecture 9 - Computational Linear Algebra Part 1

DSE 512

Drew Schmidt
2022-02-22

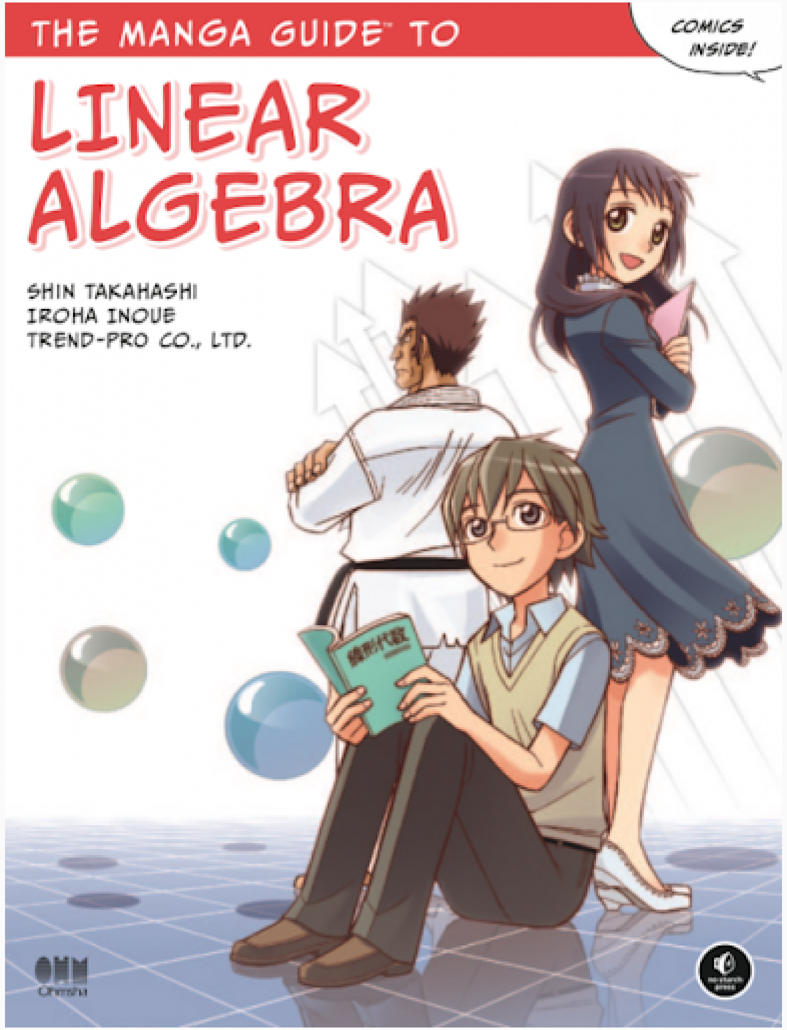
From Last Time

- Homework is out --- due Saturday
- Haven't made the slack channel yet
- Questions?

Linear Algebra

Linear Algebra

- LA dominates scientific and data computing
- Our focus will be on LA for data
  - All matrices are real-valued
  - Usually non-square



Linear Algebra On Your Computer

What happens when you multiply two matrices?

Python

```
np.dot(A, B)
```

R

```
A %*% B
```

Recall: Terminology

- **gemm** - matrix-matrix multiply
- **BLAS** - Basic Linear Algebra Subprograms; matrix library
- **FLOPS** - Floating Point Operations Per Second (adds and multiplies)
- **LINPACK** - Solve $Ax = b$
- **TOP500** - list of computers ranked by LINPACK benchmark

gemm

```
NAME
     dgemm - perform one of the matrix-matrix operations    C  :=
     alpha*op( A )*op( B ) + beta*C

SYNOPSIS
     SUBROUTINE DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
           BETA, C, LDC)

     CHARACTER * 1 TRANSA, TRANSB
     INTEGER M, N, K, LDA, LDB, LDC
     DOUBLE PRECISION ALPHA, BETA
     DOUBLE PRECISION A(LDA,*), B(LDB,*), C(LDC,*)

     SUBROUTINE DGEMM_64(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
           BETA, C, LDC)

     CHARACTER * 1 TRANSA, TRANSB
     INTEGER*8 M, N, K, LDA, LDB, LDC
     DOUBLE PRECISION ALPHA, BETA
     DOUBLE PRECISION A(LDA,*), B(LDB,*), C(LDC,*)
```

BLAS and LAPACK

Basic Linear Algebra Subprograms

- Fortran 77 library
- Basic arithmetic operations
- Level 1: vector/vector operations
- Level 2: matrix/vector operations
- Level 3: matrix/matrix operations

Linear Algebra PACKage

- Fortran 77 library
- Matrix factorizations

BLAS Library: Python

```python
import numpy as np
np.show_config()
```

```
## blas_mkl_info:
##   NOT AVAILABLE
## blis_info:
##   NOT AVAILABLE
## openblas_info:
##   NOT AVAILABLE
## atlas_3_10_blas_threads_info:
##   NOT AVAILABLE
## atlas_3_10_blas_info:
##   NOT AVAILABLE
## atlas_blas_threads_info:
##   NOT AVAILABLE
## atlas_blas_info:
##   NOT AVAILABLE
## accelerate_info:
##   NOT AVAILABLE
## blas_info:
```

BLAS Library: Python

```
np.__config__.blas_info
```

```
## {'libraries': ['blas', 'blas'], 'library_dirs': ['/usr/lib/x86_64-linux-gnu'], 'include_dirs': ['
```

BLAS Library: R

```r
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices datasets  utils     methods   base
```

BLAS Library: R

```r
sessionInfo()$BLAS
```

```
## [1] "/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3"
```

```r
sessionInfo()$LAPACK
```

```
## [1] "/usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3"
```

BLAS Interfaces

- HLL
  - Matlab
  - NumPy
  - base R
  - Basically every language
  - Many extensions ...
- C
  - lapacke
  - GSL
- C++
  - Armadillo
  - Eigen
  - Boost
  - fml

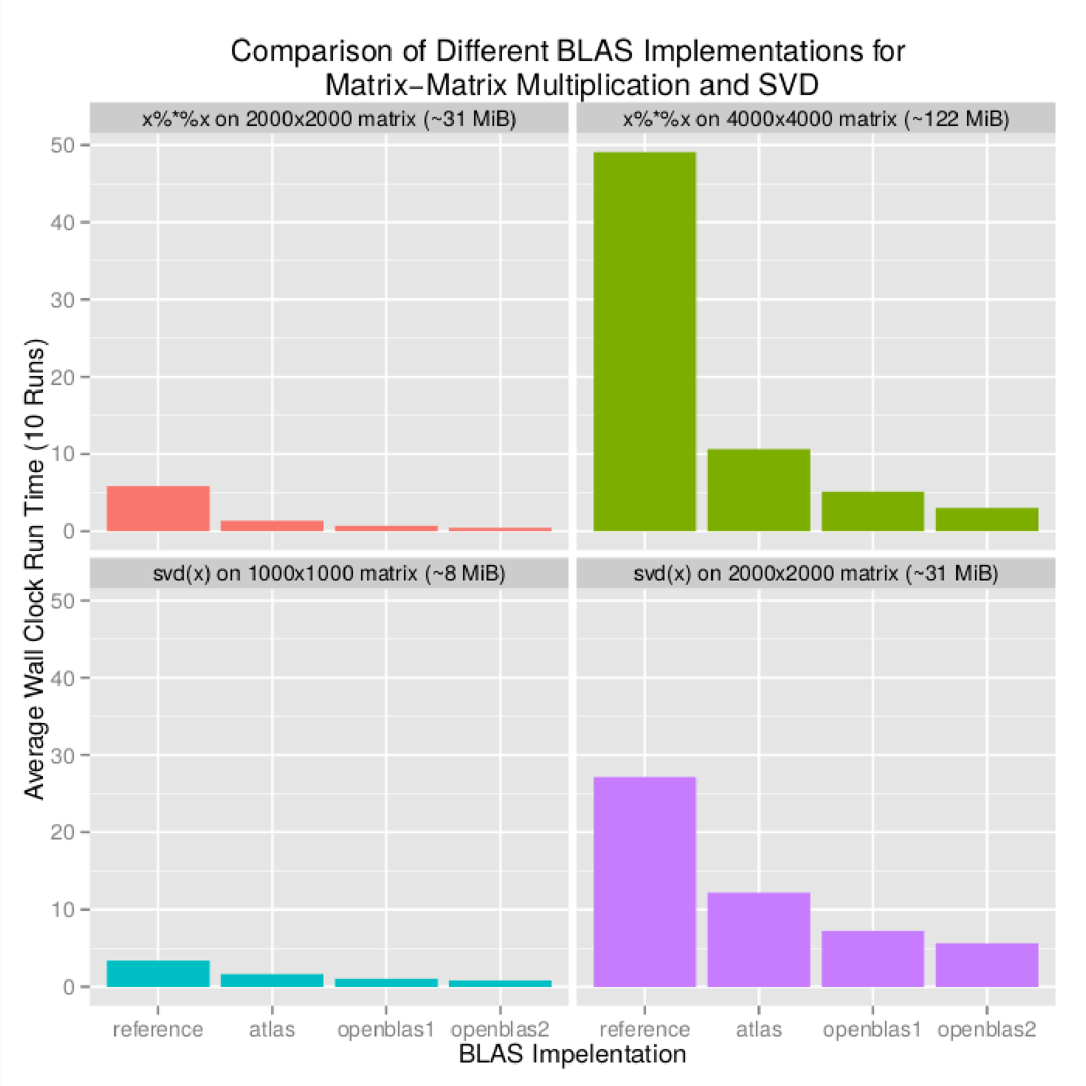
BLAS Implementations

- Open source BLAS/LAPACK implementations
  - netlib (reference)
  - Atlas
  - OpenBLAS
- Proprietary BLAS/LAPACK implementations
  - Intel MKL
  - Apple Accelerate
  - AMD Optimizing CPU Libraries (AOCL)
- GPU semi-implementations
  - NVIDIA cuBLAS and NVBLAS
  - AMD roc/hip

Question

What makes a BLAS implementation "high performance"?

Some Benchmarks



The LINPACK Benchmark

- Solve the system $Ax = b$
  - A- $n \times n$ matrix (you choose n)
  - Double precision
  - Must use LU with partial pivoting
    - $A = LU$
    - $b = Ax = LUx$
    - Double triangular system solver
  - Solution must satisfy some accuracy conditions.
- $\frac{2}{3}n^3 + 2n^2$ operations
- Most FLOPS wins!

Basic LINPACK Benchmarking

- Pick an n
- Create random $A_{n \times n}$ and x
- Define $b = Ax$
- Forget x momentarily
- Get wallclock time t for solving $A\hat{x} = b$ (following the rules)
- $GFLOPS = \dfrac{\frac{2}{3}n^3 + 2n^2}{t \cdot 10^9}$
- Try to find the best such n

Running LINPACK

- HPL http://www.netlib.org/benchmark/hpl/
- Python
  - `numpy.linalg.solve(A, b)`
- R
  - `solve(A, b)`
  - okcpuid package

    ```
    okcpuid::linpack()
    ```

- Your iPhone?! https://apps.apple.com/us/app/linpack/id380883195

My Machine

```
okcpuid::cpu_clock()
```

```
## $ncores
## [1] 16
##
## $clock.os
## 1993 MHz
##
## $clock.tested
## 3393 MHz
##
## $peak
## 217.152 GFLOPS
```

My LINPACK Results

- OpenBLAS with 8 threads
- N.max: 50,000
- R.max: 186.655 GFLOPS
- R.peak: 217.152 GFLOPS
- max-to-peak: 85.956%

Numerics in Regression

Linear Regression

$$
\begin{aligned}
y = X\beta \iff & X^T y = X^T X \beta \\
\iff & \left(X^T X\right)^{-1} X^T y = \beta
\end{aligned}
$$

...right?

Linear Regression

```
x = matrix(1:30, nrow=10)
y = runif(10)

solve(t(x) %*% x) %*% t(x) %*% y
```

```
## Error in solve.default(t(x) %*% x): Lapack routine dgesv: system is exactly singular: U[3,3] = 0
```

```
lm.fit(x, y)$coefficients
```

```
##          x1          x2          x3
## -0.07202751  0.06781070          NA
```

Linear Regression

```r
set.seed(1234)
m = 10000
n = 250
x = matrix(rnorm(m*n), nrow=m, ncol=n)
y = runif(m)

system.time(solve(t(x) %*% x) %*% t(x) %*% y)
```

```
##    user  system elapsed
##   0.900   1.907   0.323
```

```r
system.time(lm.fit(x, y))
```

```
##    user  system elapsed
##   0.348   0.003   0.352
```

Condition Numbers

> [T]he condition number […] measures how much the output value […] can change for a small change in the input argument
>
> In linear regression the condition number of the moment matrix can be used as a diagnostic for multicollinearity.

Source: https://en.wikipedia.org/wiki/Condition_number

What's Going On?

```
set.seed(1234)
x = matrix(rnorm(30), 10)

kappa(x)
```

```
## [1] 1.649968
```

```
kappa(x)^2
```

```
## [1] 2.722393
```

```
kappa(t(x) %*% x)
```

```
## [1] 2.891373
```

Conclusion

There are *good ways* and *bad ways* to compute the same thing.

Quick Review of Matrix Factorizations

Quick Background

- Matrix products are associative but not (in general) commutative
- rank --- number of linearly independent columns
- $(AB)^T = B^T A^T$
- $(AB)^{-1} = B^{-1} A^{-1}$
- $\det(AB) = \det(A)\det(B)$
- $A_{n\times n}$ is orthogonal if $A^T A = AA^T = I_{n\times n}$

LU

$$A_{n\times n} = L_{n\times n}U_{n\times n}$$

- solving a square system
- (general) matrix inversion
- calculating determinants
- $A = PLU$

Cholesky

$$A_{n\times n} = L_{n\times n}L_{n\times n}^T = U_{n\times n}^T U_{n\times n}$$

- specialized LU for "positive definite" matrices
- inversion of correlation/covariance matrices

QR

$$A_{m \times n} = Q_{m \times n} R_{n \times n}$$

- Q orthogonal ($Q^T Q = I$), R upper triangular
- solving over/under-determined systems
- linear regression
- useful in distributed contexts for PCA
- $A = LQ$

Eigendecomposition

$$A_{n\times n} = V_{n\times n}\Delta_{n\times n}V_{n\times n}$$

- V is orthogonal, Δ is diagonal
- Very useful to engineers; not that useful for data science
- Can be used for PCA
- Important connections to ***the other*** factorization…

Singular Value Decomposition

Singular Value Decomposition (SVD)

- ***THE*** most important matrix factorization for data
- SVD is to data what Eigendecomposition is to engineers
- SVD can do *almost anything*
  - Not always the fastest
  - Often the most accurate

"Compact" SVD

$$A_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^T_{n\times k}$$

- k is usually minimum of m and n
- may be taken to be the rank of A which is no greater than the minimum of m and n
- Σ is diagonal
- U and V are orthogonal
  - $U^TU = I_{n\times n}$
  - $V^TV = I_{k\times k}$

Orthogonality of U and V

```
X = matrix(1:30, 10)
s = svd(X, nu=2, nv=2)
U = s$u
V = s$v
crossprod(U) |> round(digits=8)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
crossprod(V) |> round(digits=8)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
tcrossprod(V) |> round(digits=8)
```

```
##            [,1]      [,2]       [,3]
## [1,]  0.8333333 0.3333333 -0.1666667
## [2,]  0.3333333 0.3333333  0.3333333
## [3,] -0.1666667 0.3333333  0.8333333
```

"Full" SVD

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V_{n\times n}^T$$

- Rarely done in software
- $U^TU = UU^T = I_{m\times m}$
- $V^TV = VV^T = I_{n\times n}$

Connection to Eigendecomposition

$$
\begin{aligned}
A^T A &= \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) \\
&= V\Sigma U^T U\Sigma V^T \\
&= V\Sigma^2 V^T
\end{aligned}
$$

- Likewise $AA^T = U\Sigma^2 U^T$
- We will use this extensively in the parallelism module!

Question

So what can SVD do?

SVD: Matrix Inversion

$$
\begin{aligned}
A^{-1} &= \left(U\Sigma V^T\right)^{-1} \\
&= V\Sigma^{-1}U^T
\end{aligned}
$$

SVD: System Solving

$$
\begin{aligned}
Ax = b &\iff U\Sigma V^T x = b \\
&\iff x = V\Sigma^{-1}U^T b
\end{aligned}
$$

SVD: Determinants

$$
\begin{aligned}
|\det(A)| &= |\det(U \Sigma V^T)| \\
&= \det(\Sigma) \\
&= \prod_{i=1}^{n} \sigma_{ii}
\end{aligned}
$$

```r
x = matrix(rnorm(25), nrow=5)
det(x)
```

```
## [1] -0.9838918
```

```r
prod(svd(x, nu=0, nv=0)$d)
```

```
## [1] 0.9838918
```

SVD: Regression (over/under-determined systems)

$$\begin{aligned} y = X\beta &\iff y = U\Sigma V^T \beta \\ &\iff V\Sigma^{-1}U^T y = \beta \end{aligned}$$

SVD: Column Rank

$$\begin{aligned} rank(A) &= rank\left(U\Sigma V^T\right) \\ &= \left|\{\sigma \mid \sigma > 0\}\right| \end{aligned}$$

SVD: Condition Number

$$cond(A) = \frac{\sigma_1}{\sigma_n}$$

Ungraded Homework

- (Orthogonal matrices)
  1. Prove that a product of 2 orthogonal matrices is again orthogonal.
  2. Prove that if Q is a square orthogonal matrix, its determinant must be 1 or -1 (hint: start with $Q^TQ = I$).
  3. If a square matrix has determinant 1, is it necessarily orthogonal?
- Let A be a square, symmetric matrix of order n, and let $A = U\Sigma V^T$ be its SVD. If each singular value $\sigma_i \geq 0$, find the matrix square root of A. That is, find the matrix B so that $BB = A$.

Ungraded Homework

- Let A be a square matrix of order n. The polar decomposition is a matrix factorization where $A = UP$, where U is orthogonal and P is positive semi-definite, each of order n. Derive U and P via the SVD (hint: U_{polar} is not U_{svd}).
- Let A be a square symmetric matrix of order n.
  1. Use its eigendecomposition to calculate $det(A)$.
  2. Using only its eigendecomposition, show that the trace of A is equal to the sum of the eigenvalues (hint: first show that by combining the definitions of trace and matrix product, $tr(AB) = tr(BA)$; or use your fancy inner product math if you know that).

Wrapup

- Next time:
  - The linear algebra of PCA and regression
  - More than you ever wanted to know!
- Resources
  - Golub, G.H. and Van Loan, C.F., 2013. Matrix computations. JHU press.
  - Rencher, A.C. and Schimek, M.G., 1997. Methods of multivariate analysis. Computational Statistics, 12(4), pp.422-422.
  - Matrix Factorizations for Data Analysis https://fml-fam.github.io/blog/2020/07/03/matrix-factorizations-for-data-analysis/
  - SVD Via Lanczos Iteration https://fml-fam.github.io/blog/2020/06/15/svd-via-lanczos-iteration/

Questions?