ECE 4960

Spring 2017

Lecture 8

The Wilkinson Principle
(Applied to Debugging of Linear Algebra)

Edwin C. Kan
School of Electrical and Computer Engineering
Cornell University

Different Data Structures or Methods

- In the sparse matrix implementation, we often worry about the software bugs in indexing, memory storage, etc.
- We can use two different implementations for modular testing!
  - Two different data structures
  - Two different computational routes
  - Two different approximations
  - Two different …

- Intuitively, when the two implementations give exactly the same answer, the probability that both implementations are correct is very high.
- In this validation, we may NOT need to know the ground truth!

The Wilkinson Principle

- "The computed solution *x* is the EXACT solution of a nearby (perturbed) problem."
- Or paraphrased here: "The computed solution *x* with incorrect implementation is the EXACT solution of the wrongly specified problem".
- Two implementations by two different data structures, if there are bugs, will likely NOT give the same EXACT solution!

- Surely, if the common algorithm is wrong, then both implementations can give the SAME wrong answer, and therefore, this validation is useful, but **incomplete**.

Validation by the Wilkinson Principle

- The Wilkinson technique remains critical in software validation together with other validation method.
- The Wilkinson principle works well for accounting too!

- The Wilkinson principle is probably more famous (and applicable) in the overall precision testing: When two computation implementations (no software bug in either) give similar but different results, each answer can be treated as an exact solution of the respective perturbed problem.

Example of the Wilkinson Validation

- Implement in the full matrix and in the row-compressed formats.

(1) Row permute:

```
// Switch row[i] and row[j] for matrix A and vector x
int rowPermute(matrix* A, vec* x, int i, int j);
```

(2) Row scaling:

```
// Add a*row[i] to row[j] for matrix A and vector x
int rowScale(matrix* A, vec* x, int i, int j, double a);
```

(3) Vector product:

```
// Return the product of Ax = b
int productAx(matrix* A, vec* x, vec* b);
```

Example of the Wilkinson Validation

- The return integer is often reserved for the indicator of successful operations, e.x., 0 means no error or exception, 1 means unmatched rank of *A* and *x*, 2 means INF/NINF exception, 3 means NaN exception, etc.
- If you want to improve the code readability, use local `define` statement).
- You can implement the three methods in the full matrix and the sparse matrix formats and then make element-by-element of the final results after random order of the three operations!
- You do not have to know the ground truth.
- You can easily automate the check process!
- If your matrix manipulation has software bugs, it is likely that you can find out in the simple test.

Hacker Practice

Use the row-compressed storage and the full-matrix representations to implement the vector product:

```
int productAx(matrix* A, vec* x, vec* b);
// Compute the product of Ax = b
```

Write a test function to compare the resulting matrix and vector (all elements) for validation. For the full-matrix representation, feel free to use built-in utility functions.

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 3 \\ 4 & 5 & 6 & 0 & 0 \\ 0 & 7 & 8 & 0 & 9 \\ 0 & 0 & 0 & 10 & 0 \\ 11 & 0 & 0 & 0 & 12 \end{pmatrix} \qquad x = \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}$$

Other Types of Wilkinson Validation

- We may not have two distinctive data structures that are meaningful
- Two different computational procedures:
  - Perform `productAx( )` with $x = (1, 1, \ldots, 1)$ and a direct sum for all non-zero elements

- Two different algorithms or conditioning methods
- Some tests do not cover much at all: Checking `Norm(Ax - b)` in `productAx( )` will ONLY check the function implementation of `Norm( )`.

Three Basic Matrix Problems

$$Ax = b \qquad (1)$$

Minimization of $\left\| Ax = b \right\|_2$ (2)

$$Ar = \lambda r \qquad (3)$$

- λ is a scalar (eigenvalues), *x*, *b*, and *r* are vectors, and *A* is the matrix of interest.
- Problems (1) and (2) are equivalent.
- Problem (3) contains sufficient information for (1) and (2)
- Problem (3) of the eigenvalues and eigenfunctions characterizes *A* fully. If we know (3), for any given *b*, we can find *x* quickly.

Decomposition by Eigenvectors

- For the λ_i eigenvalues, if the corresponding eigenvector is r_i:

$$R = \begin{bmatrix} r_1 & r_2 & \dots & r_n \end{bmatrix}$$

We can easily show:

$$AR = R\Lambda$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}$$

For solution of $Ax = b$, we know x can be decomposed to r_i:

$$x = c_1 r_1 + c_2 r_2 + \dots + c_n r_n$$

We know $Ar_i = \lambda_i r_i$ in the eigenvalue problem

$$Ax = A \cdot \left(c_1 r_1 + c_2 r_2 + \dots + c_n r_n\right) = c_1 \lambda_1 r_1 + c_2 \lambda_2 r_2 + \dots + c_n \lambda_n r_n = b$$

Matrix Conditioning in $\boldsymbol{Ax = b}$ (1)

Example 1:

$$\begin{pmatrix} 100 & 99 \\ 99 & 98 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 199 \\ 197 \end{pmatrix}$$

x = y = 1

$$\begin{pmatrix} 100 & 99 \\ 99 & 98.009 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 199 \\ 197 \end{pmatrix}$$

x = –7.91; y = 10

Two lines are nearly degenerate: ill conditioning for intersection solution!

Matrix Conditioning in *Ax* = *b* (2)

Example 2:

$$A = \begin{pmatrix} 10 & 100 & 0 & 0 \\ 0 & 10 & 100 & 0 \\ 0 & 0 & 10 & 100 \\ 0 & 0 & 0 & 10 \end{pmatrix}$$

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 10$$

$$A = \begin{pmatrix} 10 & 100 & 0 & 0 \\ 0 & 10 & 100 & 0 \\ 0 & 0 & 10 & 100 \\ 10^{-6} & 0 & 0 & 10 \end{pmatrix}$$

$$\lambda_1 = 11,\ \lambda_2 = 10 + i,$$
$$\lambda_3 = 10 - i,\ \lambda_4 = 9$$

Hacker Practice

Use your matrix solver for:

$$\begin{pmatrix} 100 & 99 \\ 99 & 98.01 - e \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 199 \\ 197 \end{pmatrix}$$

for $e = 10^{-2}, 10^{-3}, \ldots, 10^{-9}$. Print out the value of (x, y) and the second norm of the residual vector:

$$\left\| \begin{pmatrix} 100 & 99 \\ 99 & 98.01 - e \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 199 \\ 197 \end{pmatrix} \right\|_2$$