Introduction to verified computation

August 2019

Toyo University

INIAD

Research on Numerical Analysis

Kouta Sekine

Contents

Chapter 1

Preliminary

1.1. Usage of g++

In this lecture, we will use the C++11 from the gcc compiler. If you have never used the C++ language, you try it now!

Exercise 1.1. *Write the Algorithm 1 and run it. Let's filename be "hello.cpp".*

Algorithm 1 hello.cpp

```
#incluide < iostream >
int main(void){
 std::cout << "Hello" << std::endl;
}
```

You can compile the following command for "hello.cpp" by using g++ on the Gnome terminal:

$$\$ \ \mathbf{g++ \ hello.cpp}$$

Finally, you execute the program using the following command:

$$\$ \ \mathbf{./a.out}$$

If display "Hello world!", then succeed.

1.2. Mathematical Notations

In this section, we prepare some mathematical notations.

- Let $\mathbb{R}$ be the set of real numbers.
- The closed interval denoted by $[a, b]$ which is the set of real numbers given by $[a, b] = \{x \in \mathbb{R} | a \leq x \leq b\}$.
- Let I denote the $n \times n$ identity matrix.
- Let O denote the $n \times n$ matrix of all zeros and $\mathbf{0}$ denote the n-vector of all zeros.
- Let e the n-vector of all ones i.e. $e := (1, \ldots, 1)^T \in \mathbb{R}^n$.

- Inequalities for matrices are understood componentwise, e.g. for real $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$ the notation $A \leq B$ means $a_{ij} \leq b_{ij}$ for all (i, j).
- The notation $|A|$ means $|A| = (|a_{ij}|) \in \mathbb{R}^{n \times n}$, the nonnegative matrix consisting of componentwise absolute values of A.
- Similar notation is applied to real vectors.
- Let $\| \cdot \|$ denote the norm of vector and matrix.
- For a n dimensional vector $x = (x_1, \cdots, x_n)^T$, the maximum norm is defined by $\|x\|_\infty := \max_{1 \leq i \leq n} |x_i|$.
- For a $m \times n$ matrix $A = (a_{ij})$, the maximum norm is defined by $\|A\|_\infty := \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$.
- Let $\mathbb{IR}$ be set of real intervals.
- Let $\mathbb{F}$ be set of floating-point numbers defined by IEEE 754 standard.
- Let $\mathbb{IF}$ be set of intervals for floating-point numbers.

Chapter 2

What's verified computation?

2.1. Numerical Computations and Its Error

Numerical computation solved to some problems. For example, system of linear equations

$$Ax = b,\ A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^n, \tag{1}$$

we can efficiently obtain an approximate solution $\tilde{x}$ of (1) using some software packages[1], even if n is 10000. Of course, I do not want to solve to this problem by hand calculation!! However, when floating-point arithmetic is used for numerical computations, then the solution includes rounding errors. In general, rounding errors are small. However, in some cases solutions are affected by rounding error. Let's feel rounding errors.

Exercise 2.1. *Write the Algorithm 2 and run it.*

Algorithm 2 sqrtpow.cpp

```
#incluide < iostream >
#incluide < cmath >

int main(void){
 int n;
 double x = 2.0;
 std::cout << "Please input a positive integer number" << std::endl;
 std::cin >> n;
 std::cout << "Exact: " << x << std::endl;

 for (int i = 0; i < n; i++){
 x = std::sqrt(x);
 }
 for (int i = 0; i < n; i++){
 x = std::pow(x,2);
 }
 std::cout << "Approximate: " << x << std::endl;
}
```

[1]In this lecture, we will use the Lapack(Linear Algebra PACKage) with the BLAS(Basic Linear Algebra Subprogram). See Section 6.1.

Example of execution:

```
$ g++ sqrtpow.cpp
$ ./a.out
Please input a positive integer number n =
0
Exact  :  2.0000000000000000
Approximate  :  2.0000000000000000
$ ./a.out
Please input a positive integer number n =
25
Exact  :  2.0000000000000000
Approximate  :  2.0000000066771721
$ ./a.out
Please input a positive integer number n =
50
Exact  :  2.0000000000000000
Approximate  :  1.6487212645509468
$ ./a.out
Please input a positive integer number n =
55
Exact  :  2.0000000000000000
Approximate  :  1.0000000000000000
```

Algorithm 2 first repeatedly run the square root of x by **for** statement. Next, the program repeatedly run the square of x by **for** statement. If real numbers, exactly the same as before. However, because we use floating-point numbers, results include rounding errors.

2.2. Verified Computations and Interval Arithmetic on Real Numbers

Numerical solutions include rounding errors. Therefore, we do not know how accurate computed solutions are. Verified computations solve this problem. The essence of verified computation is interval. For example, floating-point numbers can't describe $1/3 = 0.3333\cdots$. In verified computation, we describe interval $[0.33, 0.34]$ which enclose $1/3$. Since intervals contain the exact solutions, we can understand how the accuracy of approximate solutions.

How do you calculate four arithmetic operators in intervals?

Definition 2.2 (Interval Arithmetic on real numbers). *Let $\mathbb{IR}$ be set of real intervals. For $x = [\underline{x}, \bar{x}] \in \mathbb{IR}$ and $y = [\underline{y}, \bar{y}] \in \mathbb{IR}$, we define the following interval arithmetic:*

$$
\begin{aligned}
x + y &= [\underline{x} + \underline{y}, \bar{x} + \bar{y}] \\
x - y &= [\underline{x} - \bar{y}, \bar{x} - \underline{y}] \\
x \cdot y &= [\min(\bar{x}\cdot\bar{y},\ \bar{x}\cdot\underline{y},\ \underline{x}\cdot\bar{y},\ \underline{x}\cdot\underline{y}),\ \max(\bar{x}\cdot\bar{y},\ \bar{x}\cdot\underline{y},\ \underline{x}\cdot\bar{y},\ \underline{x}\cdot\underline{y})] \\
x/y &= [\min(\bar{x}/\bar{y},\ \bar{x}/\underline{y},\ \underline{x}/\bar{y},\ \underline{x}/\underline{y}),\ \max(\bar{x}/\bar{y},\ \bar{x}/\underline{y},\ \underline{x}/\bar{y},\ \underline{x}/\underline{y})] \ \textit{for } 0 \notin y
\end{aligned}
$$

For $\circ = \{+, -, \cdot, /\}$, Definition 2.2 satisfy

$$\tilde{x} \circ \tilde{y} \in x \circ y,\ \forall \tilde{x} \in x,\ \forall \tilde{y} \in y.$$

EXAMPLE 2.3. *Let* $x = [-2, 4]$ *and* $y = [1, 2]$. *Calculate four arithmetic operators using the interval arithmetic.*

$$x + y = [-2, 4] + [1, 2] = [-2 + 1, 4 + 2] = [-1, 6]$$

$$x - y = [-2, 4] - [1, 2] = [-2 - 2, 4 - 1] = [-4, -3]$$

$$x \cdot y = [-2, 4] \cdot [1, 2] = [\min(-2, -4, 4, 8), \max(-2, -4, 4, 8)] = [-4, 8]$$

$$x/y = [-2, 4]/[1, 2] = [\min(-2, -1, 4, 2), \max(-2, -1, 4, 2)] = [-2, 4]$$

EXERCISE 2.4. *Let* $x = [2, 4]$ *and* $y = [-2, -1]$. *Calculate four arithmetic operators using the interval arithmetic.*

2.3. Disadvantages of interval arithmetic

Interval arithmetic leads to overestimation.

Example 1:

For set $x = [-1, 1]$, we can estimate

$$x^2 = [-1, 1]^2 = [-1, 1].$$

Of course, from $x^2 >= 0$, $x^2 = [-1, 1]$ is overestimate.

Example 2:

We consider a rotation matrix

$$A(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix},$$

and interval vector

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \text{[-1, 1]} \\ \text{[-1, 1]} \end{pmatrix}.$$

x means Figure 2.1.

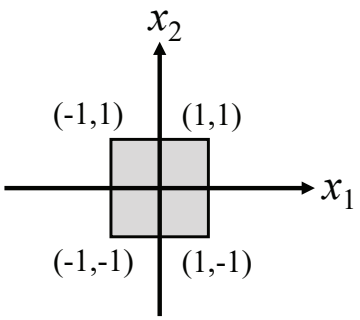

FIGURE 2.1. Wrapping effect1

$A(\pi/4)x$ means Figure 2.2. However, using interval arithmetic, we have

$$A(\pi/4)x = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \begin{pmatrix} [\text{-1, 1}] \\ [\text{-1, 1}] \end{pmatrix} = \begin{pmatrix} [-\sqrt{2}, \sqrt{2}] \\ [-\sqrt{2}, \sqrt{2}] \end{pmatrix},$$

and this interval means Figure 2.3. In this way, it is overestimated for the expression by intervals.

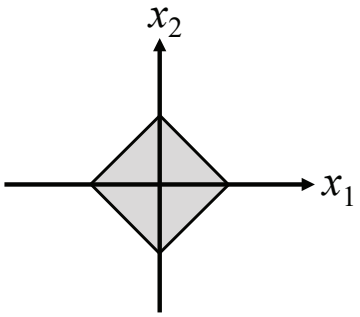

FIGURE 2.2. Wrapping effect2

Moreover, $A(\pi/4)\,(A(\pi/4)x)$ means Figure 2.2, however,

$$A\left(\frac{\pi}{4}\right)\left(A\left(\frac{\pi}{4}\right)x\right) = \begin{pmatrix} [-2, 2] \\ [-2, 2] \end{pmatrix}$$

likes Figure 2.4.

This overestimation is called a wrapping effect.

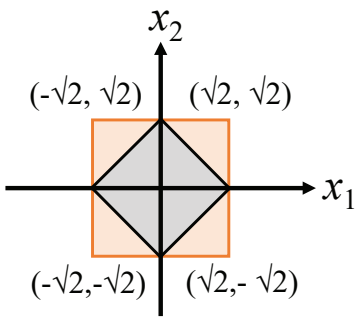

FIGURE 2.3. Wrapping effect3

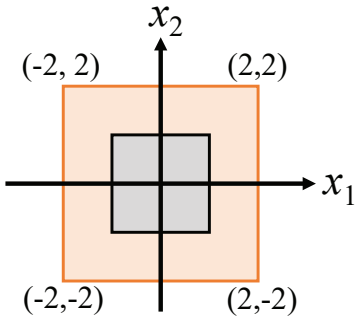

FIGURE 2.4. Wrapping effect4

Since interval arithmetic is overestimated, it is better to calculate it later. For example, changing the calculation order:

$$\left(A\left(\frac{\pi}{4}\right)A\left(\frac{\pi}{4}\right)\right)x = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} [\text{-}1,\ 1] \\ [\text{-}1,\ 1] \end{pmatrix} = \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix}.$$

There are other methodes to reduce overestimation by a means value form and affine arithmetic.

Chapter 3

Floating-Point Number and Its Interval Arithmetic

3.1. IEEE 754 : Standard for Floating-Point Arithmetic

The IEEE 754 : Standard for Floating-Point Arithmetic is a technical standard for floating-point number. Floating-point unit of many CPU use the IEEE 754 standard. For more detail, see

`http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935`

In this chapter, I will introduce to formats of floating-point number in IEEE 754. Next, I will describe rounding rules.

IEEE 754 standard have *single* and *double* formats which are binary floating-point basic formats. Representations of floating-point numbers in the binary formats are $(-1)^s \times 2^e \times m$, where

a) s is 1-bit sign (0 or 1)

b) e is any integer satisfying $emin \leq e \leq emax$.

c) m is a number represented by the form $d_0.d_1d_2 \cdots d_{p-1}$ where d_i is 0 or 1.

The value of $emax$ and p are given in Table 3.1 ($emin$ shall be $1 - emax$).

Table 3.1. Verification results.

Parameter	single	double
p	24	53
$emax$	127	1023

In Figure 3.1, we displays the representation of *Single* format. We note that start of the significand field is d_1 because we can put $d_0 = 1$ i.e. normalization.

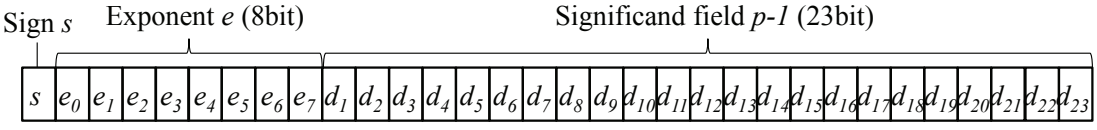

Figure 3.1. Representation of *Single* format.

EXAMPLE 3.1. *Convert from the following 10 decimal numbers to the floating point number. Here, $p = 4$ and e have 3 bit.*

$$\begin{aligned} 7.25 &= (111.01)_2 \\ &= (-1)^0 \times 2^2 \times (1.1101)_2 \end{aligned}$$

where $(\cdot)_2$ means 2 decimal numbers. Answer is

$$0\ 010\ 1101$$

Let $\mathbb{F}$ be set of floating-point numbers defined by IEEE 754 standard. You can notice that if $p = 3$ in Example 3.1 , then we can not represent the floating point number. Here, we have $\mathbb{F} \subset \mathbb{R}$. IEEE 754 standard defines five rounding rules. In this lecture, we introduce following three rounding rules:

a) Rounding to Nearest :

- Rounding to Nearest rounds to the nearest value.
- This operator is denoted by $\square : \mathbb{R} \to \mathbb{F}$

b) Rounding toward $+\infty$:

- Rounding toward $+\infty$ directed rounding towards positive infinity.
- This operator is denoted by $\triangle : \mathbb{R} \to \mathbb{F}$

c) Rounding toward $-\infty$:

- Rounding toward $-\infty$ directed rounding towards negative infinity.
- This operator is denoted by $\triangledown : \mathbb{R} \to \mathbb{F}$

From Figure 3.2 , the real number r is enclosed by interval $[\triangledown r, \triangle r]$.

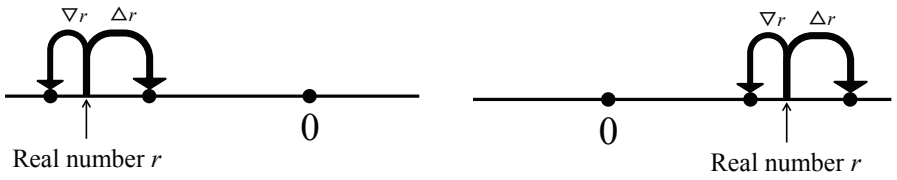

FIGURE 3.2. Rounding ($\cdot$ is a floating-point number).

3.2. Interval Arithmetic on Floating-Point Numbers

In Section 2.2, we define the interval arithmetic on real numbers. However, in some cases, real numbers are not represented floating-point numbers. We now extend Definition 2.2 to use a computer.

Definition 3.2 (Interval Arithmetic on floating-point numbers). *Let $\mathbb{IF}$ be set of intervals for floating-point numbers. For $x = [\underline{x}, \bar{x}] \in \mathbb{IF}$ and $y = [\underline{y}, \bar{y}] \in \mathbb{IF}$, we define the following interval arithmetic:*

$$
\begin{aligned}
& x + y = [\bigtriangledown(\underline{x} + \underline{y}), \bigtriangleup(\bar{x} + \bar{y})] \\
& x - y = [\bigtriangledown(\underline{x} - \bar{y}), \bigtriangleup(\bar{x} - \underline{y})] \\
& x \cdot y \\
= \ & [\min(\bigtriangledown(\bar{x} \cdot \bar{y}),\ \bigtriangledown(\bar{x} \cdot \underline{y}),\ \bigtriangledown(\underline{x} \cdot \bar{y}),\ \bigtriangledown(\underline{x} \cdot \underline{y})),\ \max(\bigtriangleup(\bar{x} \cdot \bar{y}),\ \bigtriangleup(\bar{x} \cdot \underline{y}),\ \bigtriangleup(\underline{x} \cdot \bar{y}),\ \bigtriangleup(\underline{x} \cdot \underline{y}))] \\
& x / y \\
= \ & [\min(\bigtriangledown(\bar{x}/\bar{y}),\ \bigtriangledown(\bar{x}/\underline{y}),\ \bigtriangledown(\underline{x}/\bar{y}),\ \bigtriangledown(\underline{x}/\underline{y})),\ \max(\bigtriangleup(\bar{x}/\bar{y}),\ \bigtriangleup(\bar{x}/\underline{y}),\ \bigtriangleup(\underline{x}/\bar{y}),\ \bigtriangleup(\underline{x}/\underline{y}))] \ \textit{for } 0 \notin y
\end{aligned}
$$

3.3. How to Change Rounding Rules

Using fenv.h in a C99-conformant C-language compiler allows you to use the fesetround function, which changes the rounding mode.

We now show Algorithm 3 which include setround function and getround function. The *setround* function is setting rounding rules and this argument is -1, 0 and 1.

$$
\begin{aligned}
& \textbf{setround}(\mathbf{0}) \ : \ \textbf{Rounding to Nearest} \\
& \textbf{setround}(\mathbf{1}) \ : \ \textbf{Rounding toward } +\infty \\
& \textbf{setround}(\mathbf{-1}) \ : \ \textbf{Rounding toward } -\infty
\end{aligned}
$$

The *getround* function returns current rounding rules (but not setting). Return value is:

$$\mathbf{0} \ : \ \textbf{Rounding to Nearest}$$

$$\mathbf{1} \ : \ \textbf{Rounding toward } +\infty$$

$$\mathbf{-1} \ : \ \textbf{Rounding toward } -\infty$$

Algorithm 3 change_round.hpp

```
#ifndef CHANGE_ROUND_HPP
#define CHANGE_ROUND_HPP

#include <iostream>
#include <fenv.h>

void setround(volatile int a){
 if (a == -1) fesetround(FE_DOWNWARD);
 else if (a == 0) fesetround(FE_TONEAREST);
 else if (a == 1) fesetround(FE_UPWARD);
 else std::cout << "ERROR: setround" << std::endl;
}

volatile int getround(){
 volatile int r = fegetround();
 switch (r){
 case FE_DOWNWARD:
 return -1;
 break;
 case FE_TONEAREST:
 return 0;
 break;
 case FE_UPWARD:
 return 1;
 break;
 default:
 std::cout << "ERROR: getround" << std::endl;
 return -1000;
 }
}
#endif
```

We next show how to use the setround function and the getround function

EXERCISE 3.3. *Write the Algorithm 4 and run it. Here, you need to use the following command:*

$$\$\ \mathbf{g++\ -I.\ -std==c11\ roundtest.cpp}$$

Algorithm 4 roundtest.cpp

```
#include<iostream>
#include<change_round.hpp>

int main(void){
 std::cout << std::hexfloat;
 volatile double x, y, z;
 x = 1.0;
 y = 3.0;

 std::cout << getround() << std::endl;
 z = x/y;
 std::cout << z << std::endl;

 setround(1);
 std::cout << getround() << std::endl;
 z = x/y;
 std::cout << z << std::endl;

 setround(-1);
 std::cout << getround() << std::endl;
 z = x/y;
 std::cout << z << std::endl;
}
```

Example of execution:

0

0x1.5555555555555p − 2

1

0x1.5555555555556p − 2

−1

0x1.5555555555555p − 2

Chapter 4

Interval Arithmetic for Matrix on floating-point number

In Section 3.2, we define the interval arithmetic on floating-point number. We extend Definition 3.2 to use a matrix.

DEFINITION 4.1 (Add and Subtract : Interval Arithmetic for matrix). *Let* $\mathbb{IF}$ *be set of intervals for floating-point numbers. For* $X = [\underline{X}, \bar{X}] \in \mathbb{IF}^{n\times m}$ *and* $Y = [\underline{Y}, \bar{Y}] \in \mathbb{IF}^{n\times m}$*, we define the following interval arithmetic:*

$$X + Y = [\bigtriangledown(\underline{X} + \underline{Y}), \bigtriangleup(\bar{X} + \bar{Y})]$$
$$X - Y = [\bigtriangledown(\underline{X} - \bar{Y}), \bigtriangleup(\bar{X} - \underline{Y})]$$

Next, we want to show the matrix multiplication. However, matrix multiplication is calculated by many add and multiply. Therefor, we define midrad form of intervals.

Let x be an interval of *real number. Real numbers* x_m and x_r are defined by

$$\begin{aligned} x_m &= \frac{\bar{x} - \underline{\mathrm{x}}}{2} + \underline{\mathrm{x}}, \\ x_r &= x_m - \underline{\mathrm{x}}. \end{aligned}$$

Then, we rewrite the interval x as

$$\langle x_m, x_r \rangle = [\underline{\mathrm{x}}, \bar{x}].$$

Here, x_m and x_r means midpoint and radius of interval x, respectively.

Next, let x be an interval of *floating-point number. floating-point numbers* x_m and x_r are defined by

$$\begin{aligned} x_m &= \bigtriangleup\left(\frac{\bar{x} - \underline{\mathrm{x}}}{2} + \underline{\mathrm{x}}\right), \\ x_r &= \bigtriangleup\left(x_m - \underline{\mathrm{x}}\right), \end{aligned}$$

where $\bigtriangleup(\cdot)$ means that all calculation in curly bracket is upward. Here, we have

$$[\underline{\mathrm{x}}, \bar{x}] \subset \langle x_m, x_r \rangle \subset [\bigtriangledown(x_m - x_r), \bigtriangleup(x_m + x_r)].$$

The midrad form is applied to matrix multiplication for interval.

DEFINITION 4.2 (Add and Subtract : Interval Arithmetic for matrix). *Let* $\mathbb{IF}$ *be set of intervals for floating-point numbers. For* $X = \langle X_m, X_r \rangle \in \mathbb{IF}^{n \times m}$ *and* $Y = \langle Y_m, Y_r \rangle \in \mathbb{IF}^{m \times n}$*, we define the following interval arithmetic:*

$$X \cdot Y = [\bigtriangledown(X_m \cdot Y_m - C), \bigtriangleup(X_m \cdot Y_m + C)]$$

where

$$C = \bigtriangleup((|X_m| + X_r) \cdot Y_r + X_r \cdot |Y_m|).$$

Chapter 5

Verification Theory for System of Linear Equations

In this chapter, we will introduce verification theories for system of linear equations

$$Ax = b,\ A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^n. \tag{2}$$

We first present the Banach perturbation lemma.

LEMMA 5.1 (Banach's perturbation lemma). *Let X and Y be Banach spaces. A bounded linear operator $D : X \to Y$ has the bounded inverse operator $D^{-1} : Y \to X$. If a bounded linear operator $B : X \to Y$ holds*

$$\|D^{-1}B\| < 1, \tag{3}$$

then the bounded linear operator $D + B : X \to Y$ has the bounded inverse operator $(D + B)^{-1} : Y \to X$ and we have

$$\|(D + B)^{-1}\| \le \frac{\|D^{-1}\|}{1 - \|D^{-1}B\|} \tag{4}$$

We next present the following theorem for regularity of a matrix $A \in \mathbb{R}^{n\times n}$.

LEMMA 5.2. *Let I denote the $n \times n$ identity matrix. Let A be an $n \times n$ real matrix and R be some approximate inverse of A. If*

$$\|RA - I\| < 1 \tag{5}$$

is satisfied, then $(RA)^{-1}$ exists and we have

$$\|(RA)^{-1}\| \le \frac{1}{1 - \|RA - I\|} \tag{6}$$

Proof

We use Lemma 5.1 as $D = I$ and $B = RA - I$. Since $D^{-1}B = I(RA - I)$, the condition (3) in Lemma 5.1 is the same as the condition (5). Therefore, the matrix $RA(= D + B)$ has the inverse matrix $(RA)^{-1}$ and we have (6).

□

We next present the verification theorem for system of linear equations.

THEOREM 5.3. *Let I denote the $n \times n$ identity matrix. Let A be an $n \times n$ real matrix and b an n real vector. Let x^* be a exact solution of $Ax = b$ and $\tilde{x}$ an its approximate solution. Let R be some approximate inverse of A. If*

$$\|RA - I\| < 1$$

is satisfied, then the exact solution of $Ax = b$ exists and we have

$$\|x^* - \tilde{x}\| \leq \frac{\|R(b - A\tilde{x})\|}{1 - \|RA - I\|}. \tag{7}$$

Proof

We rewrite (2) as

$$RAx = Rb. \tag{8}$$

Since RA is nonsingular from Lemma 5.2, (8) has an exact solution

$$x^* = (RA)^{-1}Rb.$$

We have

$$\begin{aligned} x^* - \tilde{x} &= (RA)^{-1}Rb - \tilde{x} = (RA)^{-1}(Rb - RA\tilde{x}) \\ &= (RA)^{-1}R(b - A\tilde{x}). \end{aligned}$$

Finally, from (6), we have

$$\begin{aligned} \|x^* - \tilde{x}\| &= \|(RA)^{-1}R(b - A\tilde{x})\| \\ &\leq \|(RA)^{-1}\|\|R(b - A\tilde{x})\| \\ &\leq \frac{\|R(b - A\tilde{x})\|}{1 - \|RA - I\|}. \end{aligned}$$

□

REMARK 5.4. *From Theorem 5.6,*

$$\|x^* - \tilde{x}\| \leq \frac{\|R(b - A\tilde{x})\|}{1 - \|RA - I\|} \leq \frac{\|R\|\|b - A\tilde{x}\|}{1 - \|RA - I\|}. \tag{9}$$

The notation $|A|$ means $|A| = (|a_{ij}|) \in \mathbb{R}^{n\times n}$, the nonnegative matrix consisting of componentwise absolute values of A. Similar notation is applied to real vectors. Finally, we present an componentwise error bounds $|x^* - \tilde{x}|$ for $Ax = b$. Before that we show the following lemma.

LEMMA 5.5. *Let e the n-vector of all ones i.e. $e := (1, \ldots, 1)^T \in \mathbb{R}^n$. For any $x \in \mathbb{R}^n$,*

$$|x| \leq \|x\|_\infty e \tag{10}$$

Proof

Form the definition of infinity norm, we have

$$|x_j| \leq \|x\|_\infty,\ 1 \leq j \leq n.$$

Therefore,

$$|x| \leq \|x\|_\infty e.$$

□

THEOREM 5.6. *Let I denote the $n \times n$ identity matrix. Let A be an $n \times n$ real matrix and b an n real vector. Let x^* be a exact solution of $Ax = b$ and $\tilde{x}$ an its approximate solution. Let R be some approximate inverse of A. If*

$$\|RA - I\| < 1$$

is satisfied, then the exact solution of $Ax = b$ exists and we have

$$|x^* - \tilde{x}| \leq |R(b - A\tilde{x})| + \frac{\|R(b - A\tilde{x})\|_\infty}{1 - \|RA - I\|_\infty}|RA - I|e. \tag{11}$$

Proof

We have

$$\begin{aligned} x^* - \tilde{x} &= RA\tilde{x} - RA\tilde{x} + Rb - Rb + x^* - \tilde{x} \\ &= R(b - A\tilde{x}) + RA\tilde{x} - Rb + x^* - \tilde{x} \\ &= R(b - A\tilde{x}) + RA\tilde{x} - RAx^* + x^* - \tilde{x} \\ &= R(b - A\tilde{x}) - RA(x^* - \tilde{x}) + x^* - \tilde{x} \\ &= R(b - A\tilde{x}) + (I - RA)(x^* - \tilde{x}) \end{aligned}$$

From (10) and (11),

$$\begin{aligned} |x^* - \tilde{x}| &= |R(b - A\tilde{x}) + (I - RA)(x^* - \tilde{x})| \\ &\leq |R(b - A\tilde{x})| + |(I - RA)(x^* - \tilde{x})| \\ &\leq |R(b - A\tilde{x})| + |I - RA||x^* - \tilde{x}| \\ &\leq |R(b - A\tilde{x})| + |RA - I|\|x^* - \tilde{x}\|_\infty e \\ &\leq |R(b - A\tilde{x})| + \frac{\|R(b - A\tilde{x})\|_\infty}{1 - \|RA - I\|_\infty}|RA - I|e \end{aligned}$$

□

Chapter 6

How to Install Some Packages

In this chapter, some software for numerical computation with guaranteed accuracy is installed.

First, create a folder to install:

$ **mkdir niigata2019**

$ **cd niigata2019**

6.1. Install BLAS and Lapack

BLAS (Basic Linear Algebra Subprograms) are routines that provide basic vector and matrix operations (e.g. dot product and matrix multiplication). Because BLAS are efficient and portable, linear algebra software use this (e.g. Lapack). Now, we can choose some BLAS:

a) Reference BLAS :

· It is a freely-available software package but slow.

b) Intel MKL :

· It is very fast.

· You need to charge.

· MATLAB use this.

c) ATLAS :

· It is free.

· ATLAS is faster than Reference BLAS.

· However, I feel slow.

d) Open BLAS :

· It is free.

· Open BLAS is as fast as Intel MKL.

6.1.1. Install Open BLAS. In this subsection, we install the Open BLAS.

1) Download :

- You can download zip file or ter.gz file from the following URL.

`http://www.openblas.net/`

2) Unzip :

- You need to unzip to Home directory.

3) Edit Make.rule file :

- You need to edit the Make.rule file.

(Before)

 # If you don't need LAPACK, please comment it in.

 # If you set NO_LAPACK=1, the library automatically sets NO_LAPACKE=1.

 # NO_LAPACK = 1

⇓

(After)

 # If you don't need LAPACK, please comment it in.

 # If you set NO_LAPACK=1, the library automatically sets NO_LAPACKE=1.

NO_LAPACK = 1

(Before)

 # If you need to synchronize FP CSR between threads (for x86/x86_64 only).

 # CONSISTENT_FPCSR = 1

⇓

(After)

 # If you need to synchronize FP CSR between threads (for x86/x86_64 only).

CONSISTENT_FPCSR = 1

4) make command :

- You type "make" in Gnome terminal.

We show the following commands to install the OpenBLAS. (Newest version: 0.3.6 If you want to use the latest version, replace 0.2.20 with 0.3.6.)

$ **wget http : //github.com/xianyi/OpenBLAS/archive/v0.2.20.zip**

$ **unzip v0.2.20.zip**

$ **cd OpenBLAS − 0.2.20/**

$ **vim Makefile.rule**

$ **make**

$ **cp libopenblas_haswell − r0.2.20.a** ../

$ **cd** ..

6.1.2. (Optional) Install MKL. MKL is free for academic users.

URL: `https://software.intel.com/en-us/performance-libraries`

a) Download MKL :

- Register and download mkl for linux in Windows

b) Transfer to Linux environment with scp command :

c) Expand MKL's compressed folder.

d) Open the unzipped folder silent.cfg with a text editor.

e) Edit $ACCEPT\_EULA = decline \implies ACCEPT\_EULA = accept$.

f) Close the editor.

g) sudo ./install.sh -s silent.cfg

h) source /opt/intel/bin/compilervars.sh intel64

i) source /opt/intel/mkl/bin/mklvars.sh intel64 lp64

6.2. Install kv library and VCP Library

kv library is created by Masahide Kashiwagi. The kv library contains scalar interval operations, highly accurate calculations, verified numerical integration, verified ODE solver, and so on.

kv library

```
$ wget http://verifiedby.me/kv/download/kv-0.4.46.tar.gz
$ tar -zxvf kv-0.4.46.tar.gz
$ cp -r kv-0.4.46/kv/ ./
```

The vcp library contains matrix calculations, simultaneous linear equations solver, eigenvalues solver, PDE solver, and so on.

VCP library

```
$ wget https://verified.computation.jp/VCP_Lib/vcp_a0.0.6.zip
$ unzip vcp_a0.0.6.zip
$ cp -r vcp_a0.0.6/vcp/ ./
$ cp -r vcp_a0.0.6/test_matrix/ ./
$ cp -r vcp_a0.0.6/test_PDE/ ./
```

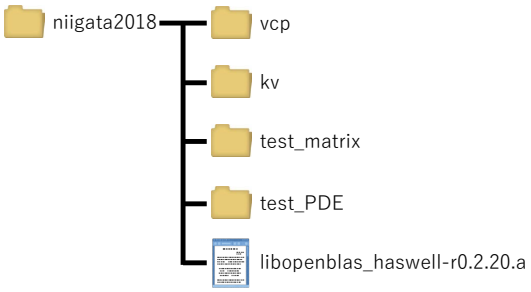

FIGURE 6.1. Configuring niigata2018 folders