Introduction to verified computation

September 2015

Waseda University

Graduate School of Fundamental Science and Engineering

Major in Pure and Applied Mathematics

Research on Numerical Analysis

Kouta Sekine

Contents

Chapter 1

Preliminary

1.1. Usage of gfortran

In this lecture, we will use the gfortran which is the fortran compiler. If you have never used the Fortran language, you try it now!

Exercise 1.1. *Write the Algorithm 1 and run it. Let's filename be "hello.f90".*

Algorithm 1 hello.f90

```
program hello
 write(*,*) 'Hello world!'
 stop
end program hello
```

You can compile the following command for "hello.f90" by using gfortran on the Gnome terminal:

$ **gfortran hello.f90**

Finally, you execute the program using the following command:

$ **./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.f90

```
program sqrtpow
 integer :: i, n
 real(8) :: x
 write(*,*) 'Please input a number n='
 read(*,*) n
 x = 2d0 ! Substitute 2 × 10^0 for x, and ! sign means comment out in fortran.
 write(*,*) 'Exact : ', x
 do i=1,n
 x = sqrt(x) ! sqrt(x) means the square root of x.
 end do
 do i=1,n
 x = x**2d0 ! x**2d0 means the square of x.
 end do
 write(*,*) 'Approximate : ', x
end program sqrtpow
```

[1]In this lecture, we will use the Lapack(Linear Algebra PACKage) with the BLAS(Basic Linear Algebra Subprogram). See Section 5.1.

Example of execution:

```
$ gfortran sqrtpow.f90
$ ./a.out
Please input a number n =
0
Exact : 2.0000000000000000
Approximate : 2.0000000000000000
$ ./a.out
Please input a number n =
25
Exact : 2.0000000000000000
Approximate : 2.0000000066771721
$ ./a.out
Please input a number n =
50
Exact : 2.0000000000000000
Approximate : 1.6487212645509468
$ ./a.out
Please input a number n =
55
Exact : 2.0000000000000000
Approximate : 1.0000000000000000
```

Algorithm 2 first repeatedly run the square root of x by **Do** statement. Next, the program repeatedly run the square of x by **Do** 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:*

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

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.*

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.

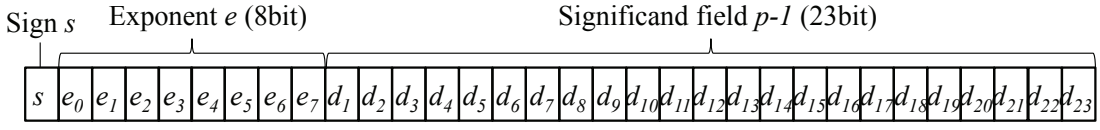

Figure 3.1. Representation of *Single* format.

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 $[\triangledown r, \triangle r]$.

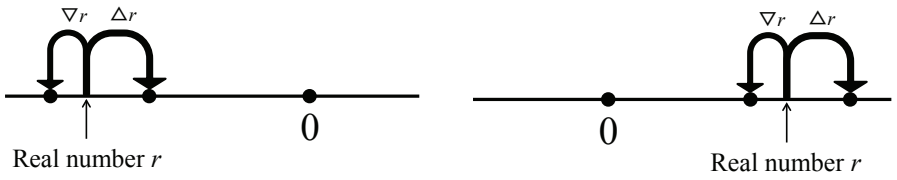

FIGURE 3.2. Rounding ($\cdot$ is a floating-point number).

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

If your fortran compiler support the official Fortran 2003 standard, then you can change rounding rules using fortran. For example, after Intel Fortran Compiler version 13 and gfortran version 5.1 are. If not support Fortran 2003 standard, then we change rounding rules using the C language. In this lecture, we will change rounding rules using the C language.

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:

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

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

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

There functions can use the fortran language.

Algorithm 3 cround.c

```
#include<stdio.h>
#include<fenv.h>

void setround(int a){
 if (a == -1) fesetround(FE_DOWNWARD);
 else if (a == 0) fesetround(FE_TONEAREST);
 else if (a == 1) fesetround(FE_UPWARD);
 else printf("ERROR : setround : Please input -1, 0, 1¥n");
}

void getround(int *a){
 volatile double e, x, y, z;
 e = 10e-30;
 x = 1.0 + e;
 y = 1.0 - e;
 if (x == y) *a = 0;
 else {
 z = (-1.0) + e;
 if (x == 1 ||z == -1) *a = -1;
 else if (y == 1) *a = 1;
 else *a = 2;
 }
}

void setround_(int *a){
 setround(*a);
}

void getround_(int *a){
 getround(a);
}
```

We next make static library which is cround.a file using following commands:

$ **gcc − c cround.c**

$ **ar cr cround.a cround.o**

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

$ **gfortran roundtest.f90 cround.a**

Algorithm 4 roundtest.f90

```
program roundtest
 real(8) :: a, b, c
 integer :: rnd
 call setround(0)
 a = -1d0
 b = 3d0

 call getround(rnd)
 write(*,*) 'Rounding rule is ', rnd

 call setround(1)
 c = a/b
 write(*,*) 'Rounding to +infinity'
 write(*,'(b64)') c

 call setround(-1)
 c = a/b
 write(*,*) 'Rounding to -infinity'
 write(*,'(b64)') c

 call setround(0)
 stop
end program roundtest
```

Example of execution:

$ **gfortran roundtest.f90 cround.a**

$ ./**a.out**

Rounding rule is 0

Rounding to + infinity

10111111110101010101010101010101010101010101010101010101010101

Rounding to − infinity

10111111110101010101010101010101010101010101010101010101010110

Chapter 4

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 4.1 (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 4.2. *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 4.1 as $D = I$ and $B = RA - I$. Since $D^{-1}B = I(RA - I)$, the condition (3) in Lemma 4.1 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 4.3. *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 4.2, (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 4.4. *From Theorem 4.6,*

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

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 4.5. *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 4.6. *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 5

How to Install Some Packages

5.1. 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.

In this lecture, 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)

 # Fortran compiler. Default is g77

 # FC = gfortran

⇓

(After)

 # Fortran compiler. Default is g77

FC = gfortran

(Before)

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

 # NO_CBLAS = 1

⇓

(After)

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

NO_CBLAS = 1

(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

(Before)

 # gfortran option for LAPACK

 # enable this flag only on 64bit Linux and if you need a thread safe lapack library

 # FCOMMON_OPT = -frecursive

⇓

(After)

 # gfortran option for LAPACK

 # enable this flag only on 64bit Linux and if you need a thread safe lapack library

FCOMMON_OPT = -frecursive

4) make command :

· You type “make” in Gnome terminal.

5) Move the libopenblas_???.so file:

· In home directory, you make a directory that name “lib”.

· libopenblas_???.so file in OpenBLAS directory is. moved to lib directory.

· libopenblas_???.so file change to name “oblas.so”.

Next, we install Lapack. Lapack provides rotines for solving system of linear equations, eigenvalue problems, and singular value problems.

1) Download :

· You can download tgz file from the following URL.

`http://www.netlib.org/lapack/`

2) Unzip :

· You need to unzip to Home directory.

3) Edit make.inc.example file :

· You may edit the make.inc.example file.

(Before)

BLASLIB = ../../librefblas.a

⇓

(After)

BLASLIB = ~/lib/oblas.so

4) Change the filename of make.inc.example:

· make.inc.example file change to name "make.inc".

5) make command :

· You type "make" in Gnome terminal.

6) Move the liblapack.a file:

· liblapack.a file in Lapack directory is. moved to lib directory.

· liblapack.a file change to name "lapack.a".

5.2. Install Slib

Slib is not yet open my interval arithmetic library.

1) Download :

- You can download zip file from the following URL (Japanese site...).

`http://271.jp/slib/index.php`

- Here, you need the following account and password.

`Account:guest`

`Password:niigata`

2) Unzip :

- You need to unzip to Home directory.

3) make command :

- You type “make” in Gnome terminal.

4) Move the libslib.a file:

- libslib.a file in Slib directory is. moved to lib directory.
- libslib.a file change to name “slib.a”.

Chapter 6

Programming of Verified Theory

6.1. 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 6.1 (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 $\triangle(\cdot)$ means that all calculation in curly bracket is upward. Here, we have

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

The midrad form is applied to matrix multiplication for interval.

DEFINITION 6.2 (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:*

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

where

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

EXERCISE 6.3. *Write the Algorithm 5 and run it. Here, you need to use the following command:*

$ **gfortran matint.f90** $\sim$ **/lib/slib.a** $\sim$ **/lib/lapack.a** $\sim$ **/lib/oblas.a**

You need to make the matint.f90 in slib directory.

Algorithm 5 matint.f90

```
program matint
 use slib
 real(8), dimension(:,:), allocatable :: Aup, Adown, Bup, Bdown, Cup, Cdown
 real(8), dimension(:,:), allocatable :: Amid, Arad, Bmid, Brad, C
 integer :: n=5
 call setround(0)
 call rand(Aup,n)
 call rand(Bup,n)
 Adown = Aup
 Bdown = Bup

 !Add
 call setround(1)
 Cup = Aup + Bup
 call setround(-1)
 Cdown = Adown + Bdown
 call setround(0)

 !Multiplication
 call setround(1)
 Amid = (Aup - Adown)/2d0 + Adown
 Arad = Amid - Adowm
 Bmid = (Bup - Bdown)/2d0 + Bdown
 Brad = Bmid - Bdowm
 C = ((abs(Amid) + Arad) .gemm. Brad) + (Arad .gemm. abs(Bmid))
 Cup = (Amid .gemm . Bmid) + C
 call setround(-1)
 Cdown = (Amid .gemm . Bmid) - C
 call setround(0)
end program matint
```

6.2. Programming of Verified Theory for System of Linear Equations

Algorithm 6 Vlinear1.f90

```
program vlinear1
 use slib
 implicit none
 real(8), allocatable, dimension(:,:) :: A, b, x, R, I, Gd, Gu
 real(8), allocatable, dimension(:,:) :: rd, ru, Gnorm, Rnorm, resnorm, D, Ainv, error
 integer :: n
 call setround(0)
 n = 1000
 call rand(A,n)
 call rand(b,n,1)

 call inv(A,R)
 x = R .gemm. b
 call eye(I,n)

 call setround(-1)
 Gd = abs((R.gemm.A) - I)
 rd = abs((A .gemm. x) - b)

 call setround(1)
 Gu = abs((R.gemm.A) - I)
 ru = abs((A .gemm. x) - b)
 call output(ru)
 Gu = max(Gu,Gd)
 ru = max(ru,rd)
 call norm(Gu,0,Gnorm)
 call norm(R,0,Rnorm)
 call norm(ru,0,resnorm)

 call setround(-1)
 D = 1d0 - Gnorm

 if (D(1,1) > 0d0) then
 call setround(1)
 Ainv = Rnorm/D
 error = Ainv*resnorm
 write(*,*) error
 else
 write(*,*) 'Verification is failed...'
 end if
 call setround(0)
end program vlinear1
```