2008년 2학기 HW#1

수치해석기초

HW#1 : Binary Representation of Real Numbers

원자핵공학과
2003-12491
이 원 재

1. 임의의 실수에 대한 이진화에서 가수와 지수를 찾는 알고리즘

① 먼저 양의 십진 실수를 읽는다.(음의 실수이면 sign 비트를 1로 바꾼 뒤 양의 실수로 변환)
② 양의 십진 실수를 정수부와 소수부로 나눈다.
③ 정수부와 소수부를 각각 이진법으로 표시하여 이진 비트열을 각각 얻는다.
④ 이제 정수부와 소수부 비트열로부터 지수(exponent)와 가수(mantissa)를 찾는다.
  (a) 이진법으로 표시된 정수부의 비트열의 길이를 읽어서 그 길이가 0이 아니면
    ->정수부의 길이를 지수로 취하고
    ->정수부와 소수부 비트들을 연결하여 허용되는 가수 비트 자리수 만큼 정수부 비트열로부터 저장하여 가수로 취한다.
  (b) 정수부 비트열의 길이가 0일 때
    ->소수부분 비트열에서 0이 아닌 것, 즉 1이 처음으로 몇 번째에 나오는지 카운트 하여 카운트의 음수 값을 지수(exponent)로 취한다.
    ->소수부분에서 처음 1이 나오는 곳부터 허용되는 가수 비트 자리수 만큼 저장하여 가수로 취한다.
⑤ 정수로 얻어진 (십진수)지수를 이진수로 변환한다. 음의 정수는 2의 보수를 사용하여 나타낸다. 즉, 음의 정수는 양의 정수로 취한다음 이진수로 바꾼뒤 1의 보수를 취하고 1을 더하여 2의 보수로 나타낸다. 이렇게 할 경우 8비트에서 가능한 정수는 -127~128이다.
⑥ 끝으로 (sign 비트) 지수비트열 가수비트열의 벡터로 반환한다.

2. 십진수를 이진수로 변환하는 프로그램(MATLAB)

In file dec2bin.m

```
function [bFull bSign arrayBinExponent arrayBinMantissa]=dec2bin(x,num_bit_mantissa)
%%%첫번째 argument는 입력받을 실수로 사용되고, 두번째 argument는 가수부를 출력할 이진 비트수의 길이를 말함
% 출력 스트림에서 첫 번째 인자는 뒤따르는 다른 인자들을 모두 합해 하나의 벡터로 나타낸 것.
% 두 번째 출력인자는 입력한 실수의 부호를 나타내는 비트. 음수일 경우 1, 양수일 경우 0
% 세 번째 출력인자는 2진수로 나타낸 지수.
% 네 번째 출력인자는 가수, 가수의 표현가능 비트수는 함수의 두 번째 매개변수로 넘겨받음
base=2;                                 %2진수 변환을 위해 base를 2로 줌
maxDigits=52;                           %최대 비트수 제한을 52로 줌

%%%confirm positive decimal number
if x<0                          %문제에서는 양의 실수만을 대상으로 했으나 실수 x가 양수인지 음수인지 판단해서
   bSign=1;                     %음수이면 sign 비트를 1로 하고
   x=-x;                        %일단 양수를 취함
else
   bSign=0;                     %양수이면 sign 비트를 0으로 함
end

%%%distinguish integer part and decimal part
xIntPart=fix(x);                %실수에서 정수부를 취해서 xIntPart에 저장
xDecPart=x-xIntPart;                    %소수부를 취해서 xDecPart에 저장

%%%Binary representation for integer part and decimal part
%%For integer part                      %정수부분에 대해 십진수를 2진수로 변환
i=0;                                    %iterator를 0으로 초기화
while (xIntPart>=1)
   i=i+1;
   tmpBinIntPart(i)=mod(xIntPart, base);          % 2로 나눠가면서 나머지를 임시 스트링(벡터)에 저장
   xIntPart=(xIntPart-tmpBinIntPart(i))/base;     % 다음에 나눌 수는 2로 나눈 몫
end
%Reverse digits                                          % 임시스트링의 숫자들의 순서를 뒤집는 코드
nIntPart=i;                                             %먼저 정수부를 2진수로 바꿨을때 소요한 비트수를 저장
if (nIntPart>0)
   arrayBinIntPart=zeros(1,nIntPart);           %필요한 배열(벡터)를 미리 마련하고 0으로 초기화
   for (i=1:nIntPart)
      arrayBinIntPart(i)=tmpBinIntPart(nIntPart-i+1); %뒤집어서 저장
   end
else
   arrayBinIntPart=zeros(1,1);                  %정수부가 0이면 그냥 0 저장
end
%%For decimal part
i=1;                                            %실수부분에 대해 십진수를 2진수로 변환
tmpBinDecPart=zeros(1,maxDigits);               %이진수들을 임시로 저장할 스트링의 마련
while (xDecPart<1 & i<=maxDigits)               %루프에서 2를 곱해가면서 정수부분을 저장
   xDecPart=xDecPart*base;
   if (xDecPart~=0)
      tmpBinDecPart(i)=fix(xDecPart);
      xDecPart=xDecPart-tmpBinDecPart(i);
   else                                                 %2를 곱해서 0이되면 끝난것
      break;
```

```
    end
    i=i+1;                                      %iterator 하나 증가시킴
end
nDecPart=i-1;                   %루프에서 iterator를 증가시키고 끝나게 되므로 이진화한 소수부의 개수는 한개 뺀것.
arrayBinDecPart=tmpBinDecPart(1:nDecPart);      %임시저장 스트링의 앞에서부터 이진화 소수부 개수만큼 취함

%%%get binary exponent(decimal number) and mantissa
%%
%이제 이진화한 정수부와 소수부로부터 지수와 가수부를 구함
arrayBinMantissa=zeros(1,num_bit_mantissa);     %최대 허용되는 가수부 비트수만큼 배열 마련
intBinExp=0;                                            %일단 십진수로 얻을 이진지수는 0으로 초기화
if (nIntPart ~=0)                               %정수부에서 이진수를 얻었을때
    intBinExp=nIntPart;                                 %이진지수는 정수부의 길이에 해당
    tmpBinMantissa=[arrayBinIntPart arrayBinDecPart];   %가수비트열은 정수이진비트열과 소수이진비트열을 더하고
    for (j=1:nIntPart+nDecPart)                 %위에서 마련한 배열에 허용되는 비트수 내에서 앞에서부터 저장
            if (j<=num_bit_mantissa)
                arrayBinMantissa(j)=tmpBinMantissa(j);
            else
                break;
            end
    end
else                                    %정수부에서 이진수를 얻지 못해서 정수부 이진비트열의 길이가 0일때
    for (i=1:nDecPart)                  %소수부에서 0이 아닌 비트를 얻을때까지 비트수를 세어서
        if (arrayBinDecPart(i)~=0)
            break;
        else
            i=i+1;
        end
    end
    intBinExp=-i+1;                             %이진지수의 값을 구함
    for (j=1:nDecPart-i+1)                      %마찬가지로 허용되는 비트수 범위 내에서 소수부 비트스트링에서
            if (j<=num_bit_mantissa)            %가수부를 취해옴
                arrayBinMantissa(j)=arrayBinDecPart(i+j-1);
            else
                break;
            end
    end
end

%지수를 이진수로 바꾸는 부프로그램 호출, 음의 정수는 2의 보수취한것의 2진 비트 스트림을 리턴 받음
arrayBinExponent=bexp(intBinExp);

bFull=[bSign,arrayBinExponent,arrayBinMantissa];

%이로써 전체 숫자의 sign 비트 1개(bSign),
%그 다음 이진지수의 이진표시로 얻은 부호포함 이진 비트열 11개(arrayBinExponent)
%그리고 가수부부분을 위한 두번째 입력 매개변수의 입력값만큼의 길이로 확보한 가수부를 리턴함
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
%%%converting binary exponent to binary representation form (optional). subroutine
function [arrayBinExp]=bexp(intBinExp)
n_bit_exp=11;                  % 부호 포함한 exponent 부분이 저장될 비트수 double 표기에서와 같이 11개로 함.
base=2;                        % 2진수 표기로 변환
if (intBinExp < 0)             % 지수가 음수일때
   posExp=-(intBinExp+1);      % %2의 보수를 취하기 위해 먼저 +1을 하고 양수를 취하고

   i=0;                                % 인덱스 초기화 후 2진수 변환해서 차례대로 임시저장용 배열에 저장
   while (posExp>=1)
      i=i+1;
      tmpBinExp(i)=mod(posExp, base);
      posExp=(posExp-tmpBinExp(i))/base;
   end
                                       %Reverse digits 임시저장용 배열을 뒤집어서 실제 리턴될 array에 저장
   nBinExp=i;
   arrayBinExp=zeros(1,n_bit_exp);
   for (i=1:nBinExp)
      arrayBinExp(n_bit_exp+1-i)=tmpBinExp(i);
   end
                                       %이제는 1의 보수를 취함 그런데 앞에서 이미 +1을 해주었으므로 2의 보수
   for (i=1:n_bit_exp)
      arrayBinExp(i)=~arrayBinExp(i);  %결과 비트스트림 리턴부분
   end

%원래는 이 부분에서 비트스트림에 +1연산을 해줘야 하지만 계산의 용이를 위해 앞에서 미리 해줌

else                                   %지수가 양수인 경우 양수 그대로 취해서 이진화시킴
   posExp=intBinExp;
   i=0;
   while (posExp>=1)
      i=i+1;
      tmpBinExp(i)=mod(posExp, base);
      posExp=(posExp-tmpBinExp(i))/base;
   end
%Reverse digits 임시저장용 배열을 뒤집어서 실제 리턴될 array에 저장
   nBinExp=i;
   arrayBinExp=zeros(1,n_bit_exp);
   for (i=1:nBinExp)
      arrayBinExp(n_bit_exp+1-i)=tmpBinExp(i);  %결과 비트스트림 리턴부분
   end
end
```

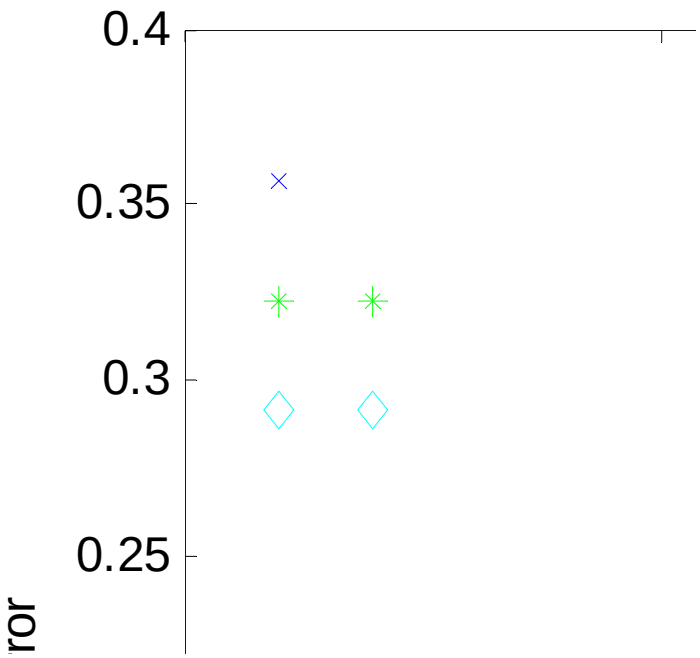
3. 어떤 다양한 수에 대해서 가수를 나타내는 비트수를 변화시킬 때 이진수 표기로 인해 발생하는 오차에 대한 분석

먼저 난수발생함수로 발생시킨 5개의 0~1사이의 수에 대해서 각각 가수부 저장 비트수를 1부터 52까지 변화시켜가면서 오차를 얻은 후에 각 숫자들에 대한 오차들을 가수부 저장 비트수에 대해서 plot 해보기로 한다.

In matlab command window
>>format long
>>a=rand(1,5)

a = 0.706046088019609 0.031832846377421 0.276922984960890 0.046171390631154 0.097131781235848

>> for(j=1:5) for(i=1:52) E(j,i)=errorOf(a(j),i); end; end
>> x=1:15;
>> plot(x,E(1,x),'cd',x,E(2,x),'m+ ',x,E(3,x),'ro',x,E(4,x),'g*',x,E(5,x),'bx')



위에서 사용한 errorOf 함수는 사용자 정의 함수로서 첫 번째 argument로 임의의 실수를 받고, 두 번째 argument로 가수부 저장 비트수를 받아서 실수를 2진수로 고쳤다가 다시 십진수로 변환한 값과 원래의 실수와의 상대오차를 반환한다.

그래프를 보면 가수부 비트수가 10정도 되면 오차가 거의 무시할 정도로 작아짐을 볼 수 있다. 가수부 비트수가 점점 줄어들면 상대오차가 커지기 시작하는데 비트수 5개에서는 어느정도 모여있던 오차들이 4개가 되면서 어떤 것은 커지고 어떤 것은 오차가 그대로 유지됨을 확인할 수 있다.

특히 data2 : 0.031832846377421에 대한 오차는 주목할 만한데, 비트수를 1로 했을 때와 6으로 했을 때의 오차가 차이가 없는 것을 볼 수 있다. 이는 0.031832846377421를 2진수로 바꿔서 가수부의 비트들을 확인해 보면 그 이유를 알 수 있는데

```
>> [f s e m]=dec2bin(a(2),15)
f =
Columns 1 through 18
0     1     1     1     1     1     1     1     1     1     0     0     1     0     0
0     0     0
Columns 19 through 27
1     0     0     1     1     0     0     0     1
s =
0
e =
1     1     1     1     1     1     1     1     1     0     0
m =
1     0     0     0     0     0     1     0     0     1     1     0     0     0     1
>>
```

여기서 f는 실수부호, 지수부, 가수부가 모두 하나의 비트스트림으로 표현된 벡터이고, s는 실수의 부호(0이면 +,1이면 -), e는 지수의 2진수표현(2의보수로 나타냄), m은 가수부이다.

가수부의 비트들을 확인하면 맨 처음 비트가 1이고 잇달아서 0이 5개가 따르는 것을 볼 수 있는데, 이 점이 적은 가수부 비트수1일 때의 오차와 6일 때의 오차가 같았던 이유라고 할 수 있다. 따라서 결국 가수부가 111111..식으로 나온다면 오차는 ~2^(-x)식으로 감소하는 것을 그래프에서 알 수 있다. data4는 가수부가 101111....로 될 것으로 예상되는데 이를 확인해보면..

```
>> [f s e m]=dec2bin(a(4),15)
f=.....(생략)
m =
1     0     1     1     1     1     0     1     0     0     0     1     1     1     1
>>
```

로 나와서 예상이 정확하게 맞음을 확인할 수 있다.

따라서 최대로 나올 수 있는 오차는 2^(-x)로 나옴을 알 수 있다. 여기서 x는 가수저장 비트수이다.

4. single precision floa에서 가수부분은 23개의 비트로 나타내어지고 double precision은 52개의 비트로 나타내어진다.

$$.100...00100...001 = \frac{1}{2} + \frac{1}{2^{24}} + \epsilon \approx .100...01 = \frac{1}{2} + \frac{1}{2^{23}}$$

$$\frac{|\frac{1}{2^{24}} - \frac{1}{2^{23}}|}{\frac{1}{2} + \frac{1}{2^{24}} + \epsilon} \simeq \frac{\frac{1}{2^{24}}}{\frac{1}{2}} = \frac{1}{2^{23}} = 1.2 \times 10^{-7} < 5 \times 10^{-7}$$

즉 앞에서 본 바와 같이 상대오차는 2^(-x)로 나타나는데 위와 같이 single precision에서 유효숫자가 7개로 나타남을 알 수 있으며 마찬가지로 하면 double precision에서는 $\frac{1}{2^{52}} = 2.2 \times 10^{-16} < 5 \times 10^{-16}$ 으로 유효숫자가 16임을 알 수 있다.

이를 위해서 다음과 같은 코드를 생각해 보면

```
>> b=rand(1,15);
>> for(j=1:15)  S(1,j)=sigdigits(b(j),23);S(2,j)=sigdigits(b(j),52); end
>> S
S =
7     7     7     7     8     7     7     7     7     7     7     8     8    10     8
16   323   323    16   323   323   323   323   323    16    16   323    16    16   323

>>
```

위에서 강조한 **sigdigits** 는 사용자 정의 함수로서 함수정의는 다음과 같으며 주어진 실수와 가수부 비트수에 대해서 유효자리수를 반환한다.

```
function [n]=sigdigits(x,bits)
e=errorOf(x,bits);
base=10;
abase=1/base;
n=0;
while (e<5*abase)
    abase=abase/base;
    n=n+ 1;
end
```

난수 발생 함수로 생성시킨 임의의 수들에 대해서 S 행렬의 첫 번째 행에서는 single precision float에 대한 유효자리수를 나타내고, 두 번째 행에서는 double precision float의 유효자리수들을 나타낸다. 각각에 대해 최소값은 7과 16이며 이보다 큰 값들이 나타나는 이유는 난수 발생함수로 발생한 수들 중에 가수부가 52개 비트 내에서 곱해갈 때 2로 떨어지는 수가 있기 때문인 것으로 생각된다. 마치 64가 2^4으로 이 수를 나타내는데 많은 가수부 비트가 필요하지 않은 것과 비슷하다. 요약하면 single precision float과 double precision float에 대한 유효자리수는 각각 7자리와 16자리이다.

[부록] 문제를 풀면서 작성한 함수들의 matlab 소스코드

In file b2d.m

```
function [resultX]=b2d(bitStream)
%dec2bit이 리턴한 2진수 비트열을 매개변수로 받아서 10진수로 변환하여 리턴한다.
n_sign_bit=1; %부호비트는 1개
n_exp_bit=11; %지수표현 비트는 11개
n_full=length(bitStream); %전체 비트스트림의 길이 저장

argSign=bitStream(1:n_sign_bit);
argExp=bitStream(n_sign_bit+ 1:n_sign_bit+ n_exp_bit);
argMan=bitStream(n_sign_bit+ n_exp_bit+ 1:n_full);

resultX=bin2dec(argSign,argExp,argMan); %아래에 있는 함수 호출


%%%%sub program


function [x]=bin2dec(bSign,arrayBinExponent,arrayBinMantissa)
base=2;
maxDigits=52;

%%%먼저 지수부 비트스트림을 십진수로 고침

intBinExp=bit2int(arrayBinExponent);                    %십진수로 고치는 서브 함수 콜

%%%지수의 부호와 크기에 따라 가수부 비트스트림을 정수부와 소수부로 분리
%지수가 0을 포함한 양수이면 가수부 비트스트림에서 그 크기만큼 카운트하여
%정수부로 보내고 나머지를 가수부로 취함
%지수가 음수이면 정수부는 일단 0이고 가수부에서 지수의 절대값만큼 0을 앞에 추가함

len=length(arrayBinMantissa);                           %일단 가수부 비트스트림의 길이저장

if (intBinExp==0)                                       %지수가 0인경우
    bi=0;
    bd=arrayBinMantissa;
elseif (intBinExp>0)                                    % 지수가 1 이상인 경우
    bi=arrayBinMantissa(1:intBinExp);
    bd=arrayBinMantissa(intBinExp+ 1:len);
else                                                            % 지수가 음수인 경우
    bi=0;
    tmp=zeros(1,-intBinExp);
    bd=[tmp,arrayBinMantissa];
                                % 여기서 maxDigits만큼 앞에서 잘라내고 뒤는 버려야 하지만
                                % 어차피 소수부 이진비트스트림 만들때 최대 허용비트수 내에서 만들었으므로 안해도 됨
end

%%%이제는 만들어진 정수부 2진 비트열과 소수부 2진 비트열로부터 실수를 구함

sum=0;
abase=1;
```

```
len=length(bi);
iup=len;

for (i=1:len)
    sum=sum+ bi(iup)*abase;
    abase=abase*base;
    iup=iup-1;
end
len=length(bd);
abase=1/base;
for (i=1:len)
    sum=sum+ bd(i)*abase;
    abase=abase/base;
end
%%%sign bit가 0이면 양의 실수, 0이 아니면 음의 실수 리턴
if (bSign==0)
    x=sum;
else
    x=-sum;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
%%%converting binary stream to integer for binary exponent term
function [intBinExp]=bit2int(arrayBinExp)
base=2;                              % 2진수 표기로 변환
sum=0;
abase=1;
% 비트스트림의 첫번째 비트가 1이면 음수 0이면 양수이므로
if (arrayBinExp(1)==0)               % 첫 비트가 0이면 양수
    len=length(arrayBinExp);    % 전체비트수를 저장
    iup=len;                         % array indexer를 어레이 크기로 초기화
    for (i=1:len)
        sum=sum+ arrayBinExp(iup)*abase;
        abase=abase*base;            % 그 다음 차수를 위해 2의 계승을 해감
        iup=iup-1;
    end
else                            %첫 비트가 1이면 음수이므로 2의 보수취하는 연산을 반대로 해줌
    len=length(arrayBinExp);
    iup=len;
    for (i=1:len)               % 1의 보수취해줌 비트연산
        arrayBinExp(i)=~arrayBinExp(i);
    end
    for (i=1:len)
        sum=sum+ arrayBinExp(iup)*abase;
        abase=abase*base;            % 그 다음 차수를 위해 2의 계승을 해감
        iup=iup-1;
    end
    sum=sum+ 1;                      % 2의 보수취하기 위해 +1해줌
    sum=-sum;                        % 마침내 음의 정수를 취함
end
intBinExp=sum;
```

In file errorOf.m

```
function [relE,approxX]=errorOf(x,bits)
approxX=b2d(dec2bin(x,bits));
relE=abs((approxX-x)/x);
```

In file sigdigits.m

```
function [n]=sigdigits(x,bits)
e=errorOf(x,bits);
base=10;
abase=1/base;

n=0;
while (e<5*abase)
    abase=abase/base;
    n=n+ 1;
end
```