CST Part IA: Numerical Method, SV 1,2,3,4
Joe Yan
2017-5-4

2001P3Q10

β: the base.
p: the number of bit in precision.
e_{max}: the largest exponent.
e_{max}: the minimum exponent.
8 bits and 23 bits are respectively needed for the exponent and the significand.
23 bits other than 24 bits for the significand is due to the normal expression always begins with 1 in the significand except 0 and so this bit becomes the hidden bit.
The exponent: the unsigned integer representation of $e + 127$.
Zero: all exponent and significand bits are 0;
Denormal numbers: all exponent bits are 0 and significand bit are not all 0;
Normalised numbers: the exponent is in range -126 to $+127$ stored as 00000001 to 11111110.
Infinities: the exponent is 128 (11111111) and the significand is all 0. The sign bit defines the infinity is positive or negative.
NaN: the exponent is 128 (11111111) and the significand is not all 0.

(a) NaN

(b) NaN

(c) If x is zero its NaN. If x is positive it is $\pm\infty$. If x is negative it is $\mp\infty$

(d) If x is $-\infty$ its NaN. If x is $+\infty$ its $+\infty$.

(e) 1×2^{-126}

(f) $0.11111111111111111111111|_2 \times 2^{-126} = 2^{-126} - 2^{-149}$

(g) $0.00000000000000000000001|_2 \times 2^{-126} = 2^{-149}$

2003P3Q6

(a) Meaning answered in 2001P3Q10.
Sign, exponent and significand need respectively 1, 11 and 52 bits. Again for the reason of hidden bit significand needs 52 bits instead of 53.
Totally 64 bits are needed.

(b) The hidden bit is the most significant bit in the 53-bit "actual" significand and it is 1 for normalised numbers and 0 for denormal numbers.
Because all normalised numbers except 0 begin with 1 in the significand this bit as 1 is hidden to save the space and increase the precision as a hidden bit.
For denormalised numbers the exponent is 00000000000 representing -1022. The hidden bit is 0 to "discretely continue" (How to express this less confusing?) from the normalised number with smallest absolute value.

(c) Machine epsilon is the difference between 1 and the smallest representable number which is greater than 1.

Just considering the normalised representation here (otherwise the macheps can be arbitrarily large).
$\epsilon_m = 2^{-52}$ because the there are 52 bits for significand and 1 hidden bit.
(Wiki also shows it is the largest possible relative error due to rounding which will yield a slightly different answer $\epsilon_m = 2^{-53}$.)

(d) The idea is to make the total error as small as possible.
For truncation error:
$f(x+h) = f(x) + hf'(x) + h^2 f''(x)/2 + O(h^3)$
$\frac{f(x+h)-f(x)}{h} - f'(x) = \frac{h}{2} f''(x) + O(h^2) \approx h$
For rounding error:
$\frac{f(x+h)-f(x)}{h} - f'(x) \approx \frac{\epsilon}{h}$ because $f(x)$ and $f'(x)$ is $O(1)$ and the absolute error of subtraction.
Overall:
Total error $\approx \frac{\epsilon}{h} + h$
The minimum value of the total error is taken when $h \approx \sqrt{\epsilon}$.
In the double precision the total error $\approx 2\sqrt{\epsilon} = 2^{-20}$.
The absolute accuracy is about 2^{-20}.

(e) The same idea as (d).
For truncation (discretization) error:
$f(x+h) = f(x) + hf'(x) + h^2 f''(x)/2 + h^3 f'''(x)/6 + O(h^4)$
$2\frac{f(x+h)-f(x)-hf'(x)}{h^2} - f''(x) = hf'''(x)/3 + O(h^2) \approx h$
For rounding error:
$2\frac{f(x+h)-f(x)-hf'(x)}{h^2} - f''(x) \approx 2\frac{2\epsilon+2\sqrt{\epsilon}}{h^2} \approx \frac{4\epsilon^{1/2}}{h}$
Total error $\approx h + \frac{4\epsilon^{1/2}}{h}$
Total error takes minimum value $\approx 5\epsilon^{1/4} \approx 2^2\epsilon^{1/4}$ when $h = \epsilon^{1/4}$.
As required absolute accuracy is 2^{-10}. $2^2\epsilon^{1/4} < 2^{-11}$ giving $\epsilon < 52$ which suggests to use double precision floating number.

2008P3Q2

(a)
```
float minLargerFloat32(const float &f){
    int ansInt((int&)f);
    float* ans = &((float&)ansInt);
    if(ansInt < 0){
        if(ansInt == INT32_MIN){ //case 0
            ansInt =  8388608;
        }else if(ansInt >= - 8388608){ //case -inf or NaN
            ansInt = ansInt;
        }else if(ansInt > -2139095040){ //case normal
            ansInt = --ansInt;
        }else if(ansInt == -2139095040){ //case max negative normal
            ansInt = 0;
        }else{ //case denormal
            ansInt = -1;
        }
    }else{
        if(f==0){ // case 0
            ansInt =  8388608;
        }else if(ansInt < 8388608){ // case denormal
            ansInt = -1;
        }else if(ansInt < 2139095040){ //case normal
            ansInt = ++ansInt;
        }else{ //case inf or NaN
            ansInt = ansInt;
        }
    }
    return *ans;
}
```

This function also works for negative numbers and -2^{-126} goes to +0.

(b) When it is negative we need to minus one instead of plus one. Assuming we are doing this on a positive number (just in general case, ignoring the edge case). We get the largest number that smaller than the input number and flipping the sign will make the inequality flip either so if its a negative number then we get the smallest number that larger than the input number as required.

(c) Truncation error is the error made by approximating an infinite sum by finite sum.
The rounding error is the error made by the difference between the calculated floating value and the actual mathematical value.

(d) The idea is to make the total error as small as possible.
By assuming f and its derivative to be within an order of magnitude of 1.0.
For truncation error:
$f(x+h) = f(x) + hf'(x) + h^2f''(x)/2 + h^3f'''(x)/6 + O(h^4)$
$f(x-h) = f(x) - hf'(x) + h^2f''(x)/2 - h^3f'''(x)/6 + O(h^4)$
$\frac{f(x+h)-f(x-h)}{2h} - f'(x) = \frac{h^2}{3}f''(x) + O(h^3) \approx h^2$
For rounding error:
$\frac{f(x+h)-f(x-h)}{2h} - f'(x) \approx \frac{\epsilon}{h}$
Overall:
Total error $\approx \frac{\epsilon}{h} + h^2$
The minimum value of the total error is taken when $h \approx \sqrt[3]{\epsilon}$.

(e) $10^{-15} \approx 2^{-50}$ and for double precision floating point $p = 53$. 50 is slightly smaller than 53 so pick $\epsilon = 2^{-53}$ gives the suggesting that $h \approx 2^{-18}$.

2009P3Q6

(a) The 32-bit floating point format has 1-bit sign, 8-bit exponent and 23-bit significand.
$\beta = 2$
$e_{max} = 127$
$e_{min} = -126$
$p = 24$
Ignoring all denormalised numbers.
The range of normalised numbers is the numbers with absolute value in the range $[2^{-126}, (2-2^{-23})2^{127}]$ and less than 23 significant bits precision in the format of $1.significand\_bit \times 2^{exponent}$ while exponent bits are not all 0 or all 1.
If all exponent bits are 0 and all significand bits are also 0 the represented value is positive or negative zero based on the sign bit.
If all exponent bits are 0 and not all significand bits are 0 then it is a denormal representation which is ignored in this question.
If all exponent bits are 1 and all significand bits are 0 it represents either positive or negative infinity based on the sign bit.
If all exponent bits are 1 and significand bits are not all 0 it represents not a number.

(b) This iteration will stop when the first time that $x == x + 1$ is true.
This happens when the 1 in 32-bit float get transfer to the same exponent as x and it underflows.
Because there are 23 significand bits so underflow happens when x is 2^{24}.
Now the iteration stops and the function returns x which is 2^{24}

(c) For x is infinity, NaN, adding one will not change the value of x.
The IEEE plus operation will first change the number with smaller exponent to be the same exponent as the number with larger exponent and shift the significand then adding up the significand and dealing with the carry to give the result.
There is underflow flow issues when shifting the significand.
If the absolute value of x is larger than 1, discussed in (b), underflow happens when the absolute value of x is larger or equal to2^{24}.
Also such adding will not accumulate rounding errors for x because 1 is in the exact form in 32-bit floating.
If the absolute value of x is smaller than 1 the significand bits of x get shifted.
If the exponent of x is smaller or equal to -24 the x will underflow totally during shifting. The function returns exactly 1.0f.
Such adding will accumulate the rounding errors for x because the significand bits of x shift to right and so the tail of the significand will be removed causing rounding error.
So the behaviour of the function including the result and its error is based on the input x.
Overall for x with the exponent smaller than -24 or larger than 24, the function will not change the value at all.
Generally closer x is to 1, more mathematically and precise the function will behave.

(d) If a small perturbation of input to the algorithm cause a large change of the output the algorithm is said to be ill-conditioned with this input.
For this algorithm the input is y and the output is x. If a small perturbation of x will change the output y greatly, then the algorithm is said to be ill-conditioned.

(e) (i) Let $\epsilon_n = x_n - \delta$ where δ is the exact value of $\sqrt{a}$.
Let $f(x) = x^2 - a$
$\epsilon_{n+1} = x_{n+1} - \delta = x_n - \frac{f(x_n)}{f'(x_n)} - \delta = \epsilon_n - \delta = \epsilon_n - \frac{f(\delta+\epsilon_n)}{f'(\delta+\epsilon_n)}$
$= \epsilon_n - \frac{f(\delta)+\epsilon_n f'(\epsilon_n)+\epsilon_n^2 f''(\epsilon_n)/2+O(\epsilon_n^3)}{f'(\delta)+\epsilon_n f''(\delta)+O(\epsilon_n^2)}$
Note $f(\epsilon) = 0$
$= \frac{\epsilon_n^2 f''(\epsilon_n)/2+O(\epsilon_n^3)}{f'(\delta)+\epsilon_n f''(\delta)+O(\epsilon_n^2)}$
By applying Taylor expansion to the fraction get $\frac{\epsilon_n^2 f''(\epsilon_n)}{2f'(\epsilon_n)} + O(\epsilon_n^3)$
Overall the ϵ is decaying quadratically.

(ii) By $\sqrt{a} \in [1, 2)$ the $x_0 = 1.5$ gets one significant bit right. And we know if the x_n is decaying quadratically then the number of accurate significant bits should be doubled each time.
Let numbers of iteration needed be n.
For 32-bit floating $p = 24$ and $2^{24} = 16 * 10^6 < 10^8$.
$2^n = 8$ gives $n = 3$
For 64-bit floating $p = 53$ and $2^{53} = 8 * 10^{15} < 10^{16}$
So just one more iteration is needed. $n = 4$

(iii) Here the algorithm is requiring a relative good x_0 to get start with.
Let s be the significand and e be the exponent.
Looking for $\sqrt{s \times 2^e} = \sqrt{s} \times 2^{e/2}$ or $\sqrt{2s} \times 2^{(e-1)/2}$
So $x_0 = 1.1|_2 \times 2^{e\ div\ 2}$ is a good starting point. Here div means integer division.
This may save several iteration comparing to some specific bad choices and also constructing such x_0 is very cheap.

1998P4Q9

(a) $\begin{bmatrix} 5 & 5 & 9 \\ 0 & -0.010 & 98 \\ 0 & 1.0 & 2.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.50 \\ 100 \\ 2.0 \end{bmatrix}$

(b) $-9800x_3 = -10000$
$x_3 = 1.0$

(c) $98x_3 = 100$
$x_3 = 1.0$

(d) For (b)
$x_2 = (100 - 98 \times 1.0)/0.01 = 200$
$x_1 = -200$
$(x_1, x_2, x_3) = (-200, 200, 1.0)$
For (c)
$x_2 = (2.0 - 2.0 \times 1.0)/1 = 0$
$x_1 = -1.7$
$(x_1, x_2, x_3) = (-1.7, 0.0, 1.0)$

(e) For (b) $\begin{bmatrix} 5 & 5 & 9 \\ 1 & 0.99 & 100 \\ 1 & 2 & 3.8 \end{bmatrix} \begin{bmatrix} -200 \\ 200 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 9.0 \\ 98 \\ 203,8 \end{bmatrix}$
For (c) $\begin{bmatrix} 5 & 5 & 9 \\ 1 & 0.99 & 100 \\ 1 & 2 & 3.8 \end{bmatrix} \begin{bmatrix} -1.7 \\ 0.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 98.3 \\ 2.1 \end{bmatrix}$
Here we calculate the distance between the 2 s.f. answer substituted result and the result it should be.
For (b) $\sqrt{(9.0 - 0.5)^2 + (98 - 100)^2 + (203.8 - 2.1)} \approx 201.89$
For (c) $\sqrt{(0.5 - 0.5)^2 + (98 - 100)^2 + (2.1 - 2.1)} = 2.00$
So the result in (c) is closer, (c) gives a better result than (b).
The reason for this is -0.010 is a very small value and using it as a pivot will cause large error when we apply the back substitution to get a full answer.

2001P4Q9

(a) A matrix A is symmetric positive definite if and only if $x^T Ax > 0$ for all $x \neq 0$ and $A^T = A$.

(b) Let $x = (a, b)$.
$x^T Ax = 2a^2 + 2b^2 + 2ab = (a + b)^2 + a^2 + b^2$
This is always larger than 0 except $(a, b) = (0, 0)$ $\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.
So A is positive definite.

(c) We are looking for x which satisfying $Ax = b$.
Substituting $A = LDL^T$ gives $LDL^T x = b$
From question we have $(DL^T)x = y = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$ by applying forward substitution for L is a lower triangular matrix.
Notice DL^T is a upper triangle matrix.
$\begin{bmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{bmatrix} x = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$

(d) By applying forward substitution $x = \begin{bmatrix} 1/3 \\ 1/3 \end{bmatrix}$

(e) Let $x_n \to x$ as $n \to \inf$ and $\epsilon_n = x_n - x$.
$\lim_{x\to\inf} |\frac{\epsilon_{n+1}}{f(|\epsilon_n)|}| = C$ where C is a constant.
The order of convergence is defined as $|\epsilon_{n+1}| = O(f(|\epsilon_n|))$.

(f) Let x_0 be a number in the domain of f, assuming it is in the convergence region.
Let $x_{n+1} = x_n - \frac{f(x)}{f'(x)}$.
If $x_n \to x$ as $n \to \infty$ then $f(x) = 0$.
The order of the convergence is $O(x^2)$, quadratic as proved previously.

(f) The quadratic order of convergence doubles the significant bits for each iteration.
Assume $O(|\epsilon_n|) = O(b^{-k})$.
Since $|\epsilon_{n+1}| = O(|\epsilon_n^2|) = O((b^{-k})^2) = O(b^{-2k})$ where b is the base and $k \in \mathbb{Z}^+$.
So the number of significant digits for x_5 is roughly 18.

1999P8Q14

(a) $T_0(x) = 1$
$T_1(x) = x$
$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$
$T_2(x) = 2x(x) - 1 = 2x^2 - 1$
$T_3(x) = 2x(2x^2 - 1) - (x) = 4x^3 - 3x$
$T_4(x) = 2x(4x^3 - 3x) - (2x^2 - 1) = 8x^4 - 8x^2 + 1$
$T_5(x) = 2x(8x^4 - 8x^2 + 1) - (4x^3 - 3x) = 16x^5 - 20x^3 + 5x$

(b) Notice $T_n(x) = 2^{n-1}x^n + O(x^{n-2})$ hints a nice approximation.
$x^n \approx x^n - 2^{1-n}T_n(x)$
The degree of this polynomial is $n - 2$ and it only consists of $n - 2k \leq 0, k \in \mathbb{Z}$ ordered terms. This is a nice approximation for large n because the error $|2^{1-n}T_n(x)| < 2^{1-n}$ is exponentially converging to 0 with n for fixed $x \in [-1, 1]$ due to $T_i(x) \in [0, 1]$
Using this property that $|2^{1-n}T_n(x)| < 2^{1-n}$ is very small with large n.
$f(x) = \sum_{i=0}^{N} a_i x^i \approx \sum_{i=0}^{N} a_i x^i - a_i \frac{1}{2^{N-1}} T_N(x)$
Now the degree of the polynomial f is reduced by 1 from N to N-1.
The maximum possible error is 2^{1-N} for one economisation step for the term with degree N.

(c) As the Chebyshev approximation of x^n gives the minimax error. So using the Chebyshev polynomial $p(x)$ to approximate the $f(x)$ in order n will give the minimax error for $f(x)$. Then the solution of $p(x) = f(x)$ should be x_i for the Lagrange interpolation.

(d) This is a Taylor expansion of $sin(x)$.
$P_3(x) = x - \frac{x^3}{3!}$
$P_5(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!}$
Let ϵ be the error.
$\epsilon_3(x) = |sinx - x + \frac{x^3}{3!}|$
Differentiate it get $cosx - 1 + \frac{x^2}{2!}$ which is always larger than 0 except $x = 0$.
$\epsilon_3(x)_{max} = \epsilon_3(1) \approx 8.14 \times 10^{-3}$
$\epsilon_5(x) = |sinx - x + \frac{x^3}{3!} - \frac{x^5}{5!}|$
Differentiate it get $cosx - 1 + \frac{x^2}{2!} - \frac{x^4}{4!}$ which is always smaller than 0 except $x = 0$.
$\epsilon_5(x)_{max} = \epsilon_5(1) \approx 1.96 \times 10^{-4}$
After one economisation step $P_5'(x) = P_5(x) - \frac{1}{5! \times 2^4} T_5(x) = \frac{383}{384}x - \frac{15}{96}x^3$

$\epsilon_5'(x) = |sinx - \frac{383}{384}x + \frac{15}{96}x^3|$
Differentiate it get $cosx - \frac{385}{384} + \frac{17}{32}x^2$ which is always larger than 0 except $x = 0$.
$\epsilon_5'(x)_{max} = \epsilon_5'(1) \approx 3.25 \times 10^{-4}$ which is more accurate than $P_3(x)$

2003P4Q7

(a) For an upper triangular $n \times n$ matrix A.
Let $A_{ij}x_j = t_i$
$t_i = A_{ij}x_j = \sum_{j=i}^{n} A_{ij}x_j = A_{ii}x_i + \sum_{j=i+1}^{n} A_{ij}x_j$
$x_i = \frac{ti - \sum_{j=i+1}^{n} A_{ij}x_j}{A_{ii}}$
When $i = n$, $x_n = \frac{t_n}{A_{nn}}$ then we can find all x_i by the formula inductively. This method is called backward substitution.
This is important because (I guess) it is $O(n^2)$ complexity and it is a neat method to solve the linear system after the process of Gaussian elimination. Also for symmetric matrix, by considering the triangular decomposition, the complexity to solve the linear system can be reduced to $O(x^2)$ instead of $O(n^3)$ by Gaussian elimination.
Forward substitution works for lower triangular matrix, the idea is similar to the back substitution.

(b) A matrix A is symmetric positive definite if and only if $x^T Ax > 0$ for all $x \neq 0$ and $A^T = A$.

(c) Let $L = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$ and $Ax = LL^T x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$
By forward substitution $L^T x = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$
By back substitution $x = \begin{bmatrix} 7 \\ -2 \end{bmatrix}$

(d) $\begin{bmatrix} 3 & 4 & 1 \\ 0 & 0 & -18 \\ 0 & 2 & -4 \end{bmatrix} x = \begin{bmatrix} 16 \\ 18 \\ 8 \end{bmatrix}$
By upper substitution after row exchanging $x = \begin{bmatrix} 3 \\ 2 \\ -1 \end{bmatrix}$