MA2213 Numerical Analysis I

AY2021/22 Semester 1

Chapter 1 – Computer Arithmetic and Computational Error

- Single-precision floating-point numbers:

sign	exponent	digits
s	$e_1, \dots, e_8$	$b_1, \dots, b_{23}$
	not all 0's or 1's	$(-1)^s \cdot (1.b_1 b_2 \dots b_{23} \times 10^{e_1 e_2 \dots e_8 - 1111111})$
	00000000	$(-1)^s \cdot (0.b_1 b_2 \dots b_{23} \times 10^{1-1111111})$
0	11111111	00000000000000000000000 represents $+\infty$
1	11111111	00000000000000000000000 represents $-\infty$
0	11111111	not all 0's represents $+NaN$
1	11111111	not all 0's represents $-NaN$

  - Range: $(-1.1 \dots 11 \times 10^{11111110-1111})_2$ to $(-0.0 \dots 01 \times 10^{1-1111111})$ and $(+0.0 \dots 01 \times 10^{1-1111111})$ to $(+1.1 \dots 11 \times 10^{11111110-1111})_2$
  - Denormal numbers are less accurate than normal numbers.

- Computer arithmetic:
  - $x \oplus y = fl(fl(x) + fl(y))$
  - $x \ominus y = fl(fl(x) - fl(y))$
  - $x \otimes y = fl(fl(x) \times fl(y))$
  - $x \oplus y = fl(fl(x) \div fl(y))$

Chapter 2 – Matrix Multiplication

- Two ways to represent a matrix:

// row major order

…	a_{11}	…	a_{1n}	a_{21}	…	a_{2n}	…	a_{m1}	…	a_{mn}	…

// column major order

…	a_{11}	…	a_{m1}	a_{12}	…	a_{m2}	…	a_{1n}	…	a_{mn}	…

- General matrix multiplication:

```
for i = 1, …, m do
    for j = 1, …, p do
        c_ij ← a_i1 b_1j
        for k = 2, …, n do
            c_ij ← c_ij + a_ik b_kj
        end for
    end for
end for
return C = (c_ij)_mxp
```

No. of Multiplications: mnp

No. of Additions: $m(n-1)p$

  - Instead of b_{ij}, we can use b_{ji}^T to reduce hops in memory.

- Special matrix multiplication:

Table: Number of multiplications and additions in common matrix multiplication in $n \times n$

Operation	No. of Multiplications	No. of Additions
normal × normal	n^3	$n^2(n-1)$
normal × diagonal	n^2	0
normal × triangular	$\frac{n^2(n+1)}{2}$	$\frac{n^2(n-1)}{2}$
normal × Hessenberg	$\frac{n(n^2+3n-2)}{2}$	$\frac{n(n^2+n-2)}{2}$
normal × tridiagonal	$n(3n-2)$	$2n(n-1)$
diagonal × diagonal	n	0
diagonal × triangular	$\frac{n(n+1)}{2}$	0
diagonal × Hessenberg	$\frac{n^2+3n-2}{2}$	0
diagonal × tridiagonal	$3n-2$	0
triangular × triangular	$\frac{n(n+1)(n+2)}{6}$	$\frac{(n-1)n(n+1)}{6}$
triangular × opposite triangular	$\frac{n(n+1)(2n+1)}{6}$	$\frac{n(n-1)(2n-1)}{6}$
triangular × Hessenberg	$\frac{(n+2)(n^2+4n-3)}{6}$	$\frac{n(n+4)(n-1)}{6}$
triangular × opposite Hessenberg	$\frac{n(n^2+3n-1)}{3}$	$\frac{n(n+1)(n-1)}{3}$
triangular × tridiagonal	$\frac{(3n-2)(n+1)}{2}$	$n(n-1)$
Hessenberg × Hessenberg	$\frac{(n+4)(n+6)(n-1)}{6}$	$\frac{(n+6)(n+1)(n-1)}{6}$
Hessenberg × opposite Hessenberg	$\frac{(n+1)(2n^2+7n-6)}{6}$	$\frac{(n+2)(2n^2-n+3)}{2}$
Hessenberg × tridiagonal	$\frac{(n-1)(3n+10)}{2}$	$(n+2)(n-1)$
tridiagonal × tridiagonal	$9n-10$	$4n-4$

Chapter 3 – Numerical Methods for Solving Linear Systems

- Simple Gaussian Elimination to solve the linear system with augmented matrix $A = (A|b) = (a_{ij})_{n \times (n+1)}$:

```
// elimination
for i = 1, …, n − 1 do
    for j = i + 1, …, n do
        m_ji ← a_ji / a_ii
        for k = i + 1, …, n + 1 do
            a_jk ← a_jk - m_ji a_ik
        end for
    end for
end for

// backward substitution
x_n ← a_n,n+1 / a_nn
for i = n − 1, …, 1 do
    x_i ← a_i,n+1
    for j = i + 1, …, n do
        x_i ← x_i - a_ij x_j
    end for
    x_i ← x_i / a_ii
end for
return (x_1, …, x_n)^T
```

No. of Divisions: $\frac{n(n-1)}{2}$

No. of Subtractions: $\frac{n(n-1)(n+1)}{3}$

No. of Multiplications: $\frac{n(n-1)(n+1)}{3}$

No. of Divisions: n

No. of Subtractions: $\frac{n(n-1)}{2}$

No. of Multiplications: $\frac{n(n-1)}{2}$

- Gaussian Elimination with simple label swapping:

```
// swap function
j ← i
while j <= n and a_r(j),i = 0 do
    j ← j + 1
end while
if j = n + 1 then
    return "Error: Matrix is singular"
else if j != i then
    swap r(i) and r(j)
end if
// elimination
for i = 1, …, n do
    r(i) ← i
end for
for i = 1, …, n − 1 do
    swap
    for j = i + 1, …, n do
        m_ji ← a_r(j),i / a_r(i),i
        for k = i + 1, …, n + 1 do
            a_r(j),k ← a_r(j),k − m_ji a_r(i),k
        end for
    end for
end for
// backward substitution
for i = n, …, 1 do
    x_i ← a_r(i),n+1
    for j = i + 1, …, n do
        x_i ← x_i − a_r(i),j x_j
    end for
    x_i ← x_i / a_r(i),i
end for
return (x_1, …, x_n)
```

- Partial pivoting (use the largest entry in the column as pivot element):

```
// swap function with partial pivoting
j ← i
for k = i + 1, …, n do
    if |a_r(k),i| > |a_r(j),i| then
        j ← k
    end if
end for
if a_r(j),i = 0 then
    return "Error: Matrix is singular"
else if j != i then
```

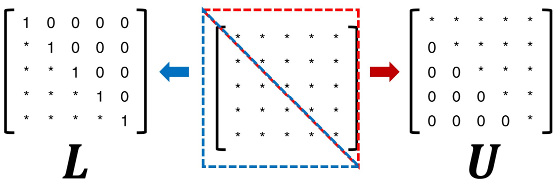
No. of Comparisons: $\frac{n(n-1)}{2}$

```
    swap r(i) and r(j)
end if
```

- Scaled partial pivoting (use the entry with the largest relative magnitude in its row in the column as pivot element):

```
// find largest entry in each row
for k = 1, …, n do
    s_k ← |a_k1|
    for j = 2, …, n do
        if |a_kj| > s_k then
            s_k ← |a_kj|
        end if
    end for
    if s_k = 0 then
        return "Error: Matrix is Singular"
    end if
end for
// swap function
j ← i
max ← |a_r(j),i| / s_r(j)
for k = i + 1, …, n do
    r ← |a_r(k),i| / s_r(k)
    if r > max then
        j ← k
        max ← r
    end if
end for
if a_r(j),i) = 0 then
    return "Error: Matrix is singular"
else if j != i then
    swap r(i) and r(j)
end if
```

- LU Factorisation to solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ (convert A to the product of a lower triangular matrix (L) and an upper triangular matrix (U)):




```
// preprocess matrix A
for i = 1, …, n − 1 do
    for j = i + 1, …, n do
        a_ji ← a_ji / a_ii
        for k = i + 1, …, n do
            a_jk ← a_jk - a_ji * a_ik
        end for
    end for
end for
return (a_ij)_nxn
// solve Lb* = b using forward substitution
for j = 2, …, n do
    for i = 1, …, j − 1 do
        b_j ← b_j − a_ji * b_i
    end for
end for
// solve Ux = b* using backward substitution
x_n ← b_n
for i = n − 1, …, 1 do
    x_i ← b_i
    for j = i + 1, …, n do
        x_i ← x_i − a_ij * x_j
    end for
    x_i ← x_i / a_ii
end for
return (x_1, …, x_n)^T
```

  - If the rows of matrix A needs to be rearranged, we can compute L and U such that $LU = PA$, then solve $LU\boldsymbol{x} = P\boldsymbol{b}$.

Chapter 4 – Interpolation and Least Squares Approximation

- Horner's method to compute the value of a polynomial:

$$P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$$
$$= a_0 + x(a_1 + x(a_2 + x(\ldots + xa_m)\ldots))$$

- Weierstrass approximation Theorem: Let f be a continuous function on $[a,b]$, then for any $\epsilon > 0$ there exists a polynomial $P(x)$ such that $\forall x \in [a,b], |f(x) - P(x)| < \varepsilon$.

- Lagrange interpolation: Suppose we have n data points $(x_0, f(x_0))$, $(x_1, \ldots, f(x_1))$, …, $(x_n, f(x_n))$, then a polynomial P is called an interpolating polynomial if it satisfies:

$$P(x_0) = f(x_0)$$
$$P(x_1) = f(x_1)$$
$$\ldots$$
$$P(x_n) = f(x_n)$$

Therefore, we can easily compute the coefficients of a degree-n polynomial P by solving the following linear system in $O(n^3)$:

$$\begin{pmatrix} 1 & \ldots & x_0^n \\ \ldots & \ldots & \ldots \\ 1 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \ldots \\ a_n \end{pmatrix} = \begin{pmatrix} f(a_0) \\ \ldots \\ f(a_n) \end{pmatrix}$$

- Lagrange basis polynomials:

$$L_k(x) = \frac{(x - x_0)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_0)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)}$$

Note that:
(1) $L_k(x_k) = 1$; AND
(2) $\forall j \neq k, L_k(x_j) = 0$
Hence, we can easily compute an interpolating polynomial by:

$$P(x) = \sum_{k=0}^{n} f(x_k)L_k(x)$$

<u>Step I: Define $\omega(x) = (x - x_0)(x - x_1)\ldots(x - x_n)$, find the coefficients $(\beta_0, \beta_1, \ldots, \beta_{n+1})$.</u>
Base case: $\beta_0 = -x_0$, $\beta_1 = 1$, considering only $(x - x_0)$.
Recursion: Suppose we know the coefficients of $(x - x_0)(x - x_1)\ldots(x - x_k)$ as $(\beta_{k0}, \beta_{k1}, \ldots, \beta_{k,k+1})$, then we want to find the coefficients of $(x - x_0)(x - x_1)\ldots(x - x_{k+1}) = (\beta_{k0} + \beta_{k1}x + \cdots + \beta_{k,k+1}x^{k+1})(x - x_{k+1})$. By comparing coefficients, we have:

$$\beta_{k+1,0} = -x_{k+1}\beta_{k0}$$
$$\beta_{k+1,k+2} = \beta_{k,k+1}$$
$$\beta_{k+1,j} = \beta_{k,j-1} - x_{k+1}\beta_{kj}$$

<u>Step II: Define $\omega_k(x) = (x - x_0)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)$, find the coefficients $(\alpha_{k0}, \alpha_{k1}, \ldots, \alpha_{kn})$.</u>
Note that $\omega(x) = (x - x_k)\omega_k(x) = (x - x_k)(\alpha_{k0} + \alpha_{k1}x + \cdots + \alpha_{kn}x^n) = -x_k\alpha_{k0} + (\alpha_{k0} - x_k\alpha_{k1})x + \cdots + (\alpha_{k,n-1} - x_k\alpha_{kn})x^n + \alpha_{kn}x^{n+1}$. By comparing coefficients, we have:

$$\alpha_{kn} = \beta_{n+1}$$
$$\alpha_{k,n-1} = \beta_n + x_k\alpha_{kn}$$
$$\ldots$$
$$\alpha_{k0} = \beta_1 + x_k\alpha_{k1}$$

<u>Step III: Compute the coefficients of $P(x)$, $(a_0, a_1, \ldots, a_n)$.</u>
Note that $L_k(x) = \frac{\omega_k(x)}{\omega_k(x_k)}$ and $P(x) = \sum_{k=0}^{n} f(x_k)L_k(x)$, we have:

$$a_j = \sum_{k=0}^{n} f(x_k)\frac{\alpha_{kj}}{\omega_k(x_k)}$$

Total time complexity is $O(n^2)$.

```
// compute the coefficients of ω(x)
β_0 ← −x_0, β_1 ← 1
for k = 0, 1, ⋯ , n − 1 do
    β_{k+2} ← β_{k+1}
    for j = k + 1, k, ⋯ , 1 do
        β_j ← β_{j−1} − x_{k+1} β_j
    end for
    β_0 ← −x_{k+1} β_0
end for
```

```
// compute the coefficients of ωk(x)
for k = 0, 1, ··· , n do
    α_kn ← β_{n+1}
    for j = n − 1, ··· , 1, 0 do
        α_kj ← β_{j+1} + x_k α_{k,j+1}
    end for
end for
return α_kj, k, j = 0, 1, ··· , n
// compute the coefficients of P(x)
for k = 0, 1 ··· , n do
    c_k ← 1
    for j = 0, 1 ··· , n do
        if j ≠ k then
            c_k ← (x_k − x_j) c_k
        end if
    end for
    c_k ← f(x_k)/c_k
end for
for j = 0, 1 ··· , n do
    a_j ← c_0 α_0j
    for k = 1, ··· , n do
        a_j ← a_j + c_k α_kj
    end for
end for
return (a_0, a_1, ··· , a_n)^T
```

- Newton's divided difference: Define $Q_n(x) = P_n(x) - P_{n-1}(x)$, then $Q_n(x) = f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1)\dots(x - x_{n-1})$, where:

$$f[x_0, x_1, \dots, x_n] = \sum_{k=0}^{n} f(x_k) \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{1}{x_k - x_j}$$

  Then $f[x_0, x_1, \dots, x_n]$ is called the n-th divided difference of f. Set $f[x_0] = f(x_0)$.

  Intuitively, we have:

$$P_0(x) = f(x_0)$$
$$P_1(x) = P_0(x) + f[x_0, x_1](x - x_0)$$
$$\dots$$
$$P_n(x) = P_{n-1}(x) + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1)\dots(x - x_{n-1})$$

  By adding everything together, we have:

$$P_n(x) = \sum_{k=0}^{n} f[x_0, \dots, x_i] \prod_{j=0}^{k-1} (x - x_j)$$

  By theorem we also have:

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

- Error of Lagrange interpolation: Suppose we interpolation $f(x)$ on $[a, b]$ as $P_n(x)$, then for any $x \in [a, b]$, there exists $\xi \in (\min\{x, x_0, \dots, x_n\}, \max\{x, x_0, \dots, x_n\})$ such that:

$$f(x) - P_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!}(x - x_0)\dots(x - x_n)$$

  This can be proved by defining the function:

$$h(t) = f(t) - P_n(t) - [f(x) - P_n(x)]\frac{(t - x_0)\dots(t - x_n)}{(x - x_0)\dots(x - x_n)}$$

  and apply Rolle's Theorem to $h^{n+1}(t)$.

- Chebyshev nodes: If we want to choose $n + 1$ nodes from $[-1, 1]$, we can choose Chebyshev nodes:

$$x_k = \cos\left(\frac{\left(k + \frac{1}{2}\right)\pi}{n + 1}\right)$$

Then the following function is a degree-n polynomial woth leading coefficient 2^{n-1}:

$$T_n(x) = \cos(n \arccos x)$$

This polynomial is called Chebyshev polynomial, which satisfies:

$$\prod_{k=0}^{n} (x - x_k) = \frac{1}{2^n} T_{n+1}(x)$$

- Least squares approximation: The value of $\sqrt{(y_0 - P(x_0))^2 + (y_1 - P(x_1))^2 + \dots + (y_n - P(x_n))^2}$ is minimised if and only if $X^T X \boldsymbol{a} = X^T \boldsymbol{y}$ where $X = \begin{pmatrix} 1 & \dots & x_0^m \\ \dots & \dots & \dots \\ 1 & \dots & x_n^m \end{pmatrix}$.

- QR Factorisation to solve $X^T X \boldsymbol{a} = x^T \boldsymbol{y}$:

$$\boldsymbol{p_0} = \begin{pmatrix} 1 \\ \dots \\ 1 \end{pmatrix}, \boldsymbol{p_1} = \begin{pmatrix} x_0 \\ \dots \\ x^n \end{pmatrix}, \dots, \boldsymbol{p_m} = \begin{pmatrix} x_0^m \\ \dots \\ x_n^m \end{pmatrix}$$

  Step I: Gram-Schmitz Orthonormalisation

$$\widetilde{\boldsymbol{p_0}} = \boldsymbol{p_0} \qquad \widetilde{\boldsymbol{p_0}}^* = \widetilde{\boldsymbol{p_0}}/\|\widetilde{\boldsymbol{p_0}}\|$$
$$\widetilde{\boldsymbol{p_1}} = \boldsymbol{p_1} - (\boldsymbol{p_1} \cdot \widetilde{\boldsymbol{p_0}}^*)\, \widetilde{\boldsymbol{p_0}}^* \qquad \widetilde{\boldsymbol{p_1}}^* = \widetilde{\boldsymbol{p_1}}/\|\widetilde{\boldsymbol{p_1}}\|$$
$$\dots \qquad \dots$$

  Step II: QR Factorisation

$$Q = (\widetilde{\boldsymbol{p_0}}^* \quad \dots \quad \widetilde{\boldsymbol{p_m}}^*)$$
$$R = \begin{pmatrix} \|\widetilde{\boldsymbol{p_0}}\| & \boldsymbol{p_1} \cdot \boldsymbol{p_0}^* & \cdots & \boldsymbol{p_m} \cdot \boldsymbol{p_0}^* \\ 0 & \|\widetilde{\boldsymbol{p_1}}\| & \cdots & \boldsymbol{p_m} \cdot \boldsymbol{p_1}^* \\ 0 & 0 & \cdots & \dots \\ 0 & 0 & \cdots & \|\widetilde{\boldsymbol{p_m}}\| \end{pmatrix}$$

  Step III: Solve $X^T X \boldsymbol{a} = x^T \boldsymbol{y}$

$$R^T Q^T Q R \boldsymbol{a} = R^T Q^T \boldsymbol{y}$$
$$R \boldsymbol{a} = Q^T \boldsymbol{y}$$

Chapter 5 – Numerical Integration

- Trapezoidal rule: Suppose $n = 1$, we have:

$$\int_a^b f(x)dx \approx \frac{b - a}{2}[f(a) + f(b)]$$

  For $n \geq 1$, we have:

$$\int_a^b f(x)dx \approx \frac{b - a}{2n}[f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(n)]$$

  There exists $\xi \in (a, b)$ such that:

$$\int_a^b f(x)dx - \frac{b - a}{2}[f(a) + f(b)] = -\frac{b - a}{12} h^2 f''(\xi)$$

- Simpson's rule: Suppose $n = 2$, we have:

$$\int_a^b f(x)dx \approx \frac{b - a}{6}\left[f(a) + 4f\left(\frac{a + b}{2}\right) + f(b)\right]$$

  For $n \geq 2$, we have:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \dots + f(x_n)]$$

  There exists $\xi \in (a, b)$ such that:

$$\int_a^b f(x)dx - \frac{b - a}{6}\left[f(a) + 4f\left(\frac{a + b}{2}\right) + f(b)\right] = \frac{b - a}{180} h^4 f^4(\xi)$$

- Newton-Cotes formula:

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} w_k f(x_k)$$
$$w_k = \frac{b - a}{n} \int_0^n \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{x - j}{k - j} dx$$

  For closed Newton-Cotes formula with $n + 1$ nodes, when n is odd and $f(x)$ is $(n + 1)$-order differentiable, there exists $\xi \in (a, b)$ such that:

$$\int_a^b f(x)dx - \sum_{k=0}^{n} w_k f(x_k) = \frac{h^{n+2} f^{n+1}(\xi)}{(n+1)!} \int_0^n s(s-1)\dots(s-n)ds$$

  When n is even and $f(x)$ is $(n + 2)$-order differentiable, there exists $\xi \in (a, b)$ such that:

$$\int_a^b f(x)dx - \sum_{k=0}^{n} w_k f(x_k) = \frac{h^{n+3} f^{n+2}(\xi)}{(n+2)!} \int_0^n s^2(s-1)\dots(s-n)ds$$