2072U Computational Science I
Winter 2022

Week	Topic
1	Introduction
1–2	Solving nonlinear equations in one variable
3–4	Solving systems of (non)linear equations
5–6	Computational complexity
6–8	Interpolation and least squares
8–10	Integration & differentiation
10–12	Additional Topics

1. Complexity of algorithms
2. Some standard sums
3. Standard algorithms

Key questions:

- What is *computational complexity*?
- What is the computational complexity of standard algorithms?
  - factorials and summation of sequences?
  - recursive algorithms?
  - matrix-vector multiplication?
  - matrix-matrix multiplication?

The concept of computational complexity will help us answer question 3:

1. When does my computation work?
2. How accurate is the result?
3. How fast does my computation work?

Floating-point operations (Flops):

- Computational complexity of an algorithm: amount of work required to execute/carry out algorithm from start to finish.
- Traditional unit of complexity for numerical algorithms: the flop.

Floating-point operations (Flops):

- Computational complexity of an algorithm: amount of work required to execute/carry out algorithm from start to finish.
- Traditional unit of complexity for numerical algorithms: the flop.
- One flop = one floating-point operation.

$\boxed{+}$	(addition)	$\boxed{\times}$	(multiplication)
$\boxed{-}$	(subtraction)	$\boxed{\div}$	(division)

Floating-point operations (Flops):

- Computational complexity of an algorithm: amount of work required to execute/carry out algorithm from start to finish.
- Traditional unit of complexity for numerical algorithms: the flop.
- One flop = one floating-point operation.

$\boxed{+}$ (addition) $\boxed{\times}$ (multiplication)
$\boxed{-}$ (subtraction) $\boxed{\div}$ (division)

How to count flops

1. Write pseudocode of algorithm clearly.
2. In each line, count number of flops ($\boxed{+}$, $\boxed{-}$, $\boxed{\times}$, $\boxed{\div}$).
3. Count number of times each line executes (e.g., in a **for** loop).
4. Multiply cost of each line by number of times it executes.

Summation identities and tricks:

- Use summation identities to count # times each line executes

$$\sum_{k=1}^{n} 1 = n, \qquad \sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$
$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{k=1}^{n} k^3 = \left[\frac{n(n+1)}{2}\right]^2 \tag{Σ}$$

Summation identities and tricks:

- Use summation identities to count # times each line executes

$$\sum_{k=1}^{n} 1 = n, \qquad \sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$
$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{k=1}^{n} k^3 = \left[\frac{n(n+1)}{2}\right]^2 \tag{Σ}$$

- Summation limits can be transformed if necessary:

$$\sum_{k=\alpha}^{\beta} a_k = \sum_{\ell=1}^{\beta-\alpha+1} a_{\ell+\alpha-1} \quad \text{(substitute } \ell = k - \alpha + 1\text{)}$$

Goal: change lower index of sum to 1 and apply identities (Σ)

Summation identities and tricks:

- Use summation identities to count # times each line executes

$$\sum_{k=1}^{n} 1 = n, \qquad \sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$
$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{k=1}^{n} k^3 = \left[\frac{n(n+1)}{2}\right]^2 \tag{Σ}$$

- Summation limits can be transformed if necessary:

$$\sum_{k=\alpha}^{\beta} a_k = \sum_{\ell=1}^{\beta-\alpha+1} a_{\ell+\alpha-1} \quad \text{(substitute } \ell = k - \alpha + 1\text{)}$$

$$\text{e.g., } \sum_{k=3}^{27} 5 = 5 \times \sum_{\ell=1}^{25} 1 = 5 \times 25 = 125$$

Goal: change lower index of sum to 1 and apply identities (Σ)

Computing a sum:

Input: vector $\mathbf{x} \in \mathbb{R}^n$
1: $S \leftarrow x_1$
2: **for** $k = 2 : n$
3: $\quad S \leftarrow S + x_k$
4: **end for**
Output: $S = \sum_{k=1}^{n} x_k$

$$S = \sum_{k=1}^{n} x_k \qquad \text{given } \mathbf{x} \in \mathbb{R}^n$$

Computing a sum:

Input: vector $\mathbf{x} \in \mathbb{R}^n$
1: $S \leftarrow x_1$
2: **for** $k = 2 : n$
3: $\quad S \leftarrow S + x_k$
4: **end for**
Output: $S = \sum_{k=1}^{n} x_k$

- One $\boxed{+}$ in Line 3: 1 flop

$$S = \sum_{k=1}^{n} x_k \qquad \text{given } \mathbf{x} \in \mathbb{R}^n$$

Computing a sum:

Input: vector $\mathbf{x} \in \mathbb{R}^n$
1: $S \leftarrow x_1$
2: **for** $k = 2:n$
3: $\quad S \leftarrow S + x_k$
4: **end for**
Output: $S = \sum_{k=1}^{n} x_k$

$$S = \sum_{k=1}^{n} x_k \qquad \text{given } \mathbf{x} \in \mathbb{R}^n$$

- One $\boxed{+}$ in Line 3: 1 flop
- Line 3 executes once for each $k = 2:n$

$$\text{Total cost of computing } \sum_{k=1}^{n} x_k \text{ is } \sum_{k=2}^{n} 1$$

Computing a sum:

Input: vector $\mathbf{x} \in \mathbb{R}^n$
1: $S \leftarrow x_1$
2: **for** $k = 2 : n$
3: $\quad S \leftarrow S + x_k$
4: **end for**
Output: $S = \sum_{k=1}^{n} x_k$

$$S = \sum_{k=1}^{n} x_k \qquad \text{given } \mathbf{x} \in \mathbb{R}^n$$

- One $\boxed{+}$ in Line 3: 1 flop
- Line 3 executes once for each $k = 2 : n$

Total cost of computing $\displaystyle\sum_{k=1}^{n} x_k$ is $\displaystyle\sum_{k=2}^{n} 1 = \boxed{n - 1 \text{ flops}}$

Computing a factorial:

Input: $n \in \mathbb{N}$ (assume $n > 2$)
1: $P \leftarrow 2$
2: **for** $k = 3 : n$
3: $\quad P \leftarrow P \times k$
4: **end for**
Output: $P = n!$

$$0! = 1,$$
$$1! = 1,$$
$$2! = 2,$$
$$n! = n(n-1)(n-2)\cdots(2)(1)$$
$$= \prod_{k=1}^{n} k \qquad \text{given } n \geq 2$$

Computing a factorial:

Input: $n \in \mathbb{N}$ (assume $n > 2$)
1: $P \leftarrow 2$
2: **for** $k = 3 : n$
3: $\quad P \leftarrow P \times k$
4: **end for**
Output: $P = n!$

$$
\begin{aligned}
0! &= 1, \\
1! &= 1, \\
2! &= 2, \\
n! &= n(n-1)(n-2)\cdots(2)(1) \\
&= \prod_{k=1}^{n} k \qquad \text{given } n \geq 2
\end{aligned}
$$

- One $\boxed{\times}$ in Line 3: 1 flop

Computing a factorial:

Input: $n \in \mathbb{N}$ (assume $n > 2$)
1: $P \leftarrow 2$
2: **for** $k = 3:n$
3: $\quad P \leftarrow P \times k$
4: **end for**
Output: $P = n!$

$$
\begin{aligned}
0! &= 1, \\
1! &= 1, \\
2! &= 2, \\
n! &= n(n-1)(n-2)\cdots(2)(1) \\
&= \prod_{k=1}^{n} k \qquad \text{given } n \geq 2
\end{aligned}
$$

- One $\boxed{\times}$ in Line 3: 1 flop
- Line 3 executes once for each $k = 3:n$

Total cost of computing $n! = \prod_{k=1}^{n} k$ is $\sum_{k=3}^{n}(1)$

Computing a factorial:

Input: $n \in \mathbb{N}$ (assume $n > 2$)
1: $P \leftarrow 2$
2: **for** $k = 3 : n$
3: $\quad P \leftarrow P \times k$
4: **end for**
Output: $P = n!$

$$
\begin{aligned}
0! &= 1, \\
1! &= 1, \\
2! &= 2, \\
n! &= n(n-1)(n-2)\cdots(2)(1) \\
&= \prod_{k=1}^{n} k \qquad \text{given } n \geq 2
\end{aligned}
$$

- One $\boxed{\times}$ in Line 3: 1 flop
- Line 3 executes once for each $k = 3 : n$

Total cost of computing $n! = \prod_{k=1}^{n} k$ is $\sum_{k=3}^{n}(1) = \boxed{n - 2 \text{ flops}}$

Computing an inner product:

Input: vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
1: $s \leftarrow x_1 \times y_1$
2: **for** $k = 2 : n$
3: $\quad s \leftarrow s + x_k \times y_k$
4: **end for**
Output: $s = \mathbf{x}^T\mathbf{y}$

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k \qquad \text{given } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

Computing an inner product:

Input: vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
1: $s \leftarrow x_1 \times y_1$
2: **for** $k = 2:n$
3: $\quad s \leftarrow s + x_k \times y_k$
4: **end for**
Output: $s = \mathbf{x}^T \mathbf{y}$

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = \sum_{k=1}^{n} x_k y_k \qquad \text{given } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

- One $\boxed{\times}$ in Line 1: 1 flop
- One $\boxed{+}$, one $\boxed{\times}$ in Line 3: 2 flops

Computing an inner product:

Input: vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
1: $s \leftarrow x_1 \times y_1$
2: **for** $k = 2:n$
3: $\quad s \leftarrow s + x_k \times y_k$
4: **end for**
Output: $s = \mathbf{x}^T\mathbf{y}$

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k \qquad \text{given } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

- One $\boxed{\times}$ in Line 1: 1 flop
- One $\boxed{+}$, one $\boxed{\times}$ in Line 3: 2 flops
- Line 1 executes exactly once
- Line 3 executes once for each $k = 2:n$

Total cost of computing $\mathbf{x}^T\mathbf{y}$ is $1 + \sum_{k=2}^{n} 2$

Computing an inner product:

Input: vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
1: $s \leftarrow x_1 \times y_1$
2: **for** $k = 2:n$
3: $\quad s \leftarrow s + x_k \times y_k$
4: **end for**
Output: $s = \mathbf{x}^T\mathbf{y}$

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k \qquad \text{given } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

- One $\boxed{\times}$ in Line 1: 1 flop
- One $\boxed{+}$, one $\boxed{\times}$ in Line 3: 2 flops
- Line 1 executes exactly once
- Line 3 executes once for each $k = 2:n$

Total cost of computing $\mathbf{x}^T\mathbf{y}$ is $1 + \sum_{k=2}^{n} 2 = \boxed{2n - 1 \text{ flops}}$

Computing a matrix-vector product (matvec):

Input: matrix $A \in \mathbb{R}^{n\times n}$, vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$
1: **for** $j = 1:n$
2: $\quad c_j \leftarrow A_{j1} \times x_1$
3: $\quad$ **for** $k = 2:n$
4: $\quad\quad c_j \leftarrow c_j + A_{jk} \times x_k$
5: $\quad$ **end for**
6: **end for**
Output: $\mathbf{c} = A\mathbf{x}$

$$\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n\times 1}$$

$$c_j = \sum_{k=1}^{n} A_{jk} b_k \quad (j = 1:n)$$

Computing a matrix-vector product (matvec):

Input: matrix $A \in \mathbb{R}^{n\times n}$, vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$
1: **for** $j = 1:n$
2: $\quad c_j \leftarrow A_{j1} \times x_1$
3: $\quad$ **for** $k = 2:n$
4: $\quad\quad c_j \leftarrow c_j + A_{jk} \times x_k$
5: $\quad$ **end for**
6: **end for**
Output: $\mathbf{c} = A\mathbf{x}$

$$\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n\times 1}$$

$$c_j = \sum_{k=1}^{n} A_{jk} b_k \quad (j = 1:n)$$

- Line 2: one $\boxed{\times}$; Line 4: one $\boxed{+}$, one $\boxed{\times}$

Computing a matrix-vector product (matvec):

Input: matrix $A \in \mathbb{R}^{n\times n}$, vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$
1: **for** $j = 1:n$
2: $\quad c_j \leftarrow A_{j1} \times x_1$
3: $\quad$ **for** $k = 2:n$
4: $\quad\quad c_j \leftarrow c_j + A_{jk} \times x_k$
5: $\quad$ **end for**
6: **end for**
Output: $\mathbf{c} = A\mathbf{x}$

$$\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n\times 1}$$

$$c_j = \sum_{k=1}^{n} A_{jk} b_k \quad (j = 1:n)$$

- Line 2: one $\boxed{\times}$; Line 4: one $\boxed{+}$, one $\boxed{\times}$
- Line 2 executes once for each $j = 1:n$
- Line 4 executes once for each $k = 2:n$ *and* $j = 1:n$

Total cost of computing $A\mathbf{x}$ is $\displaystyle\sum_{j=1}^{n}\left(1 + \sum_{k=2}^{n} 2\right)$

Computing a matrix-vector product (matvec):

Input: matrix $A \in \mathbb{R}^{n \times n}$, vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$

1: **for** $j = 1:n$
2: $\quad c_j \leftarrow A_{j1} \times x_1$
3: $\quad$ **for** $k = 2:n$
4: $\quad\quad c_j \leftarrow c_j + A_{jk} \times x_k$
5: $\quad$ **end for**
6: **end for**

Output: $\mathbf{c} = A\mathbf{x}$

$$\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n \times 1}$$

$$c_j = \sum_{k=1}^{n} A_{jk} b_k \quad (j = 1:n)$$

- Line 2: one $\boxed{\times}$; Line 4: one $\boxed{+}$, one $\boxed{\times}$
- Line 2 executes once for each $j = 1:n$
- Line 4 executes once for each $k = 2:n$ *and* $j = 1:n$

Total cost of computing $A\mathbf{x}$ is $\displaystyle\sum_{j=1}^{n} \left(1 + \sum_{k=2}^{n} 2 \right) = \boxed{2n^2 - n \text{ flops}}$

Computing a matrix-matrix product:

Input: matrices $A, B \in \mathbb{R}^{n \times n}$

```
for j = 1 : n
    for ℓ = 1 : n
        C_{jℓ} ← A_{j1} × B_{1ℓ}
        for k = 2 : n
            C_{j,ℓ} ← C_{jℓ} + A_{jk} × B_{kℓ}
        end for
    end for
end for
```

Output: $C = AB \in \mathbb{R}^{n \times n}$

Computing a matrix-matrix product:

Input: matrices $A, B \in \mathbb{R}^{n\times n}$

for $j = 1:n$
  for $\ell = 1:n$
    $C_{j\ell} \leftarrow A_{j1} \times B_{1\ell}$ $\Leftarrow$ 1 flop $(j, \ell = 1:n)$
    for $k = 2:n$
      $C_{j,\ell} \leftarrow C_{j\ell} + A_{jk} \times B_{k\ell}$ $\Leftarrow$ 2 flops $(j, \ell = 1:n;\ k = 2:n)$
    end for
  end for
end for

Output: $C = AB \in \mathbb{R}^{n\times n}$

Computing a matrix-matrix product:

Input: matrices $A, B \in \mathbb{R}^{n\times n}$

for $j = 1:n$

  for $\ell = 1:n$

    $C_{j\ell} \leftarrow A_{j1} \times B_{1\ell}$ $\Leftarrow$ 1 flop $(j, \ell = 1:n)$

    for $k = 2:n$

      $C_{j,\ell} \leftarrow C_{j\ell} + A_{jk} \times B_{k\ell}$ $\Leftarrow$ 2 flops $(j, \ell = 1:n;\ k = 2:n)$

    end for

  end for

end for

Output: $C = AB \in \mathbb{R}^{n\times n}$

Total cost of computing AB is $\displaystyle\sum_{j=1}^{n}\sum_{\ell=1}^{n}\left(1 + \sum_{k=2}^{n} 2\right) = \boxed{2n^3 - n^2 \text{ flops}}$

Summarised:

- Sum $S = \sum_{k=1}^{n} x_k$: cost is $n-1$ $\boxed{+}$ or $\boxed{n-1 \text{ flops}}$

Summarised:

- Sum $S = \sum_{k=1}^{n} x_k$: cost is $n-1$ $\boxed{+}$ or $\boxed{n-1 \text{ flops}}$
- Inner product $\mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k$: n $\boxed{\times}$ & $n-1$ $\boxed{+}$ or $\boxed{2n-1 \text{ flops}}$

Summarised:

- Sum $S = \sum_{k=1}^{n} x_k$: cost is $n-1$ $\boxed{+}$ or $\boxed{n-1 \text{ flops}}$
- Inner product $\mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k$: n $\boxed{\times}$ & $n-1$ $\boxed{+}$ or $\boxed{2n-1 \text{ flops}}$
- Matvec (matrix-vector product) $\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n\times 1}$: for $j = 1:n$,

$$c_j = \sum_{k=1}^{n} A_{jk} b_k = A_{j:}\,\mathbf{b} \quad \Rightarrow \quad n \text{ inner products: } \boxed{2n^2 - n \text{ flops}}$$

Summarised:

- Sum $S = \sum_{k=1}^{n} x_k$: cost is $n-1$ $\boxed{+}$ or $\boxed{n-1 \text{ flops}}$

- Inner product $\mathbf{x}^T\mathbf{y} = \sum_{k=1}^{n} x_k y_k$: n $\boxed{\times}$ & $n-1$ $\boxed{+}$ or $\boxed{2n-1 \text{ flops}}$

- Matvec (matrix-vector product) $\mathbf{c} = A\mathbf{x} \in \mathbb{R}^{n\times 1}$: for $j = 1:n$,

$$c_j = \sum_{k=1}^{n} A_{jk} b_k = A_{j:}\,\mathbf{b} \quad \Rightarrow \quad n \text{ inner products: } \boxed{2n^2 - n \text{ flops}}$$

- Matrix-matrix product $C = AB$: for $j = 1:n$ and $\ell = 1:n$,

$$C_{j\ell} = \sum_{k=1}^{n} A_{jk} B_{k\ell} = A_{j:}\,B_{:\ell} \quad \Rightarrow n^2 \text{ inner products: } \boxed{2n^3 - n^2 \text{ flops}}$$

Remarks

- Assume all floating-point operations have equal cost
- Ignore memory access or overwriting in computing cost
- Precise definitions of flops vary in distinct texts/papers
- Count special function evaluations (e.g., `sqrt`, etc.) as needed
- Branching statements (`if` or `case`) can require extra care
- Complexity analysis possible for memory/storage, etc.
- Complexity analysis of recursively defined functions yields recurrence relations to solve