ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

# FUNDAMENTALS OF OPTIMIZATION

## Unconstrained convex optimization

# CONTENT

- Unconstrained optimization problems
- Descent method
- Gradient descent method
- Newton method
- Subgradient method

# Unconstrained convex optimization

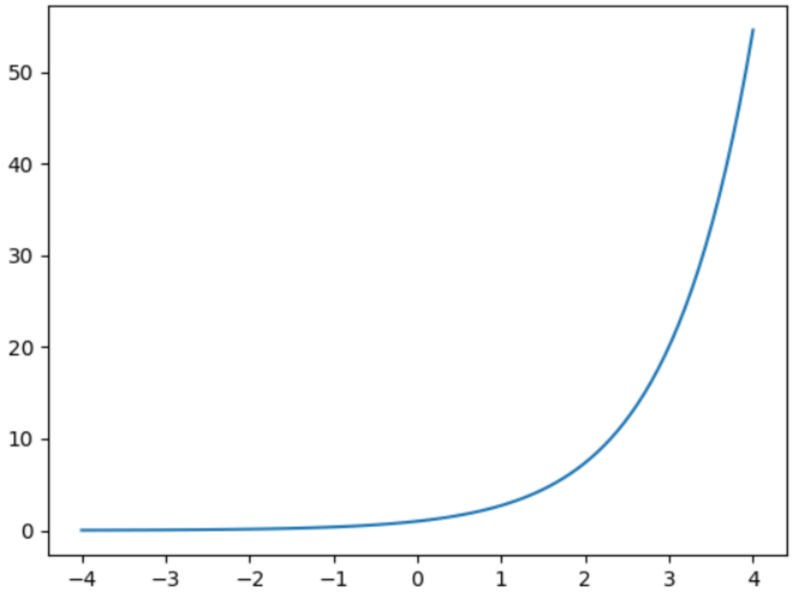
- Unconstrained, smooth convex optimization problem:

$$\min f(x)$$

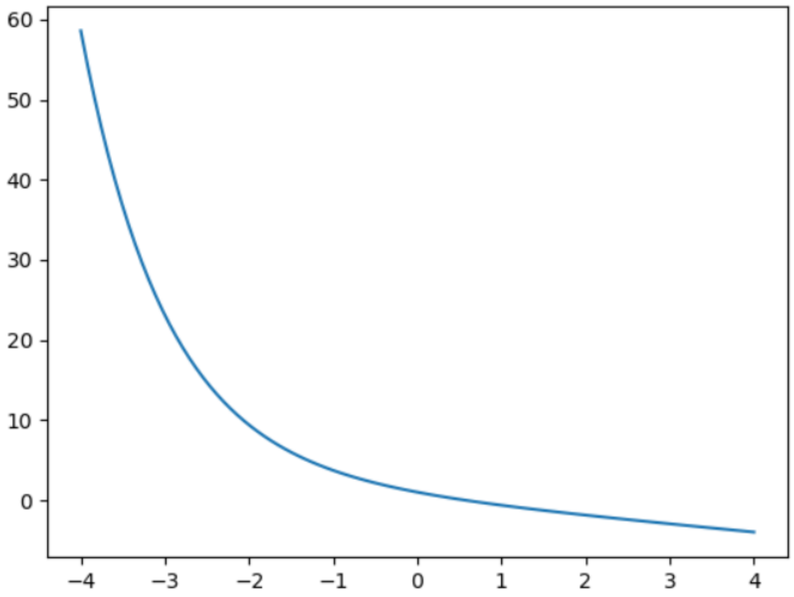
  - $f$: $R^n \rightarrow R$ is convex and twice differentiable
  - ***dom*** $f = R$: no constraint
  - Assumption: the problem is solvable with $f^* = \min_x f(x)$ and $x^* = \text{argMin}_x f(x)$
- To find $x$, solve equation $\nabla f(x^*) = 0$: not easy to solve analytically
- Iterative scheme is preferred: compute minimizing sequence $x^{(0)}, x^{(1)}, \ldots$ s.t. $f(x^{(k)}) \rightarrow f(x^*)$ as $k \rightarrow \infty$
- The algorithm stops at some point $x(k)$ when the error is within acceptable tolerance: $f(x^{(k)}) - f^* \leq \varepsilon$

# Local minimizer

- $x^*$ is a local minimizer for $f$: $R^n \rightarrow R$ if $f(x^*) \leq f(x)$ for $||x^*-x|| \leq \varepsilon$ ($\varepsilon > 0$ is a constant)
- $x^*$ is a global minimizer for $f$: $R^n \rightarrow R$ if $f(x^*) \leq f(x)$ for all $x \in R^n$



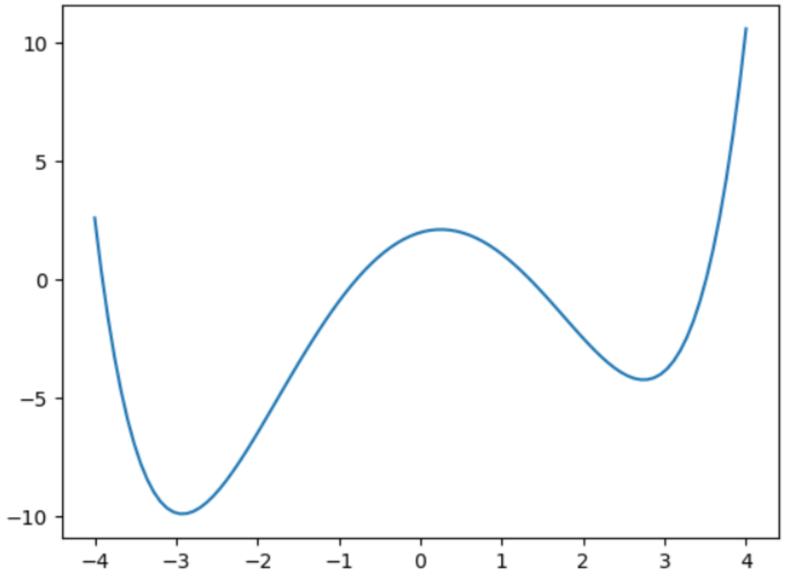

$f(x) = e^x$ has no minimizer



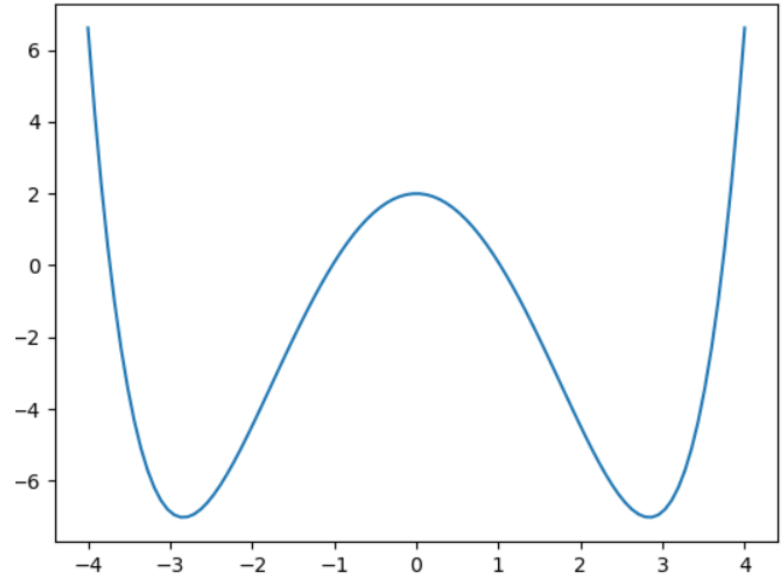

$f(x) = -x + e^{-x}$ has no minimizer

# Local minimizer

- $x^*$ is a local minimizer for $f$: $R^n \rightarrow R$ if $f(x^*) \leq f(x)$ for $||x^*-x|| \leq \varepsilon$ ($\varepsilon > 0$ is a constant)
- $x^*$ is a global minimizer for $f$: $R^n \rightarrow R$ if $f(x^*) \leq f(x)$ for all $x \in R^n$




$f(x) = e^x + e^{-x} - 3x^2 + x$ has two local minimizers and one global minimizer




$f(x) = e^x + e^{-x} - 3x^2$ has two global minimizers

# Local minimizer

- **Theorem** (Necessary condition for local minimum) If $x^*$ is a local minimizer for $f$: $R^n \rightarrow R$, then $\nabla f(x^*) = 0$ (x* is also called *stationary point* for *f*)
- **Proof** Suppose that $x^*$ is a local minimizer but $\nabla f(x^*) \neq 0$. We can find a vector $z$ such that $<\nabla f(x^*), z> < 0$ (for instance $z = -\nabla f(x^*)$, $<\nabla f(x^*), z> = -||\nabla f(x^*)||^2 < 0$)
- For a constant $t \geq 0$, consider vector $x(t) = x^* + tz$, and $\varphi(t) = f(x(t))$
- $\frac{d\varphi(t)}{dt}\Big|_{t=0} = \frac{df(x(t))}{dt}\Big|_{t=0} = <\nabla f(x^*), z> < 0 \rightarrow$ with constant $t$ small enough, $\varphi(t) < \varphi(0)$ or $f(x(t)) < f(x^*)$ (conflict with the assumption that $x^*$ is a local minimizer) □

# Local minimizer

**Example**

- $f(x,y) = x^2 + y^2 - 2xy + x$
- $\nabla f(x,y) = \begin{pmatrix} 2x - 2y + 1 \\ 2y - 2x \end{pmatrix} = 0$ has no solution

$\rightarrow$ there is no minimizer of $f(x,y)$

# Local minimizer

- **Theorem** (Sufficient condition for a local minimum) Assume $x^*$ is a stationary point and that $\nabla^2 f(x^*)$ is positive definite, then $x^*$ is a local minimizer

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial xn} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial xn} \\ \cdots & & & \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial xn \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial xn \partial xn} \end{pmatrix}$$

# Local minimizer

- Matrix $A_{nxn}$ is called positive definite if

$$A^i = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,i} \\ & \cdots\cdots & & \\ a_{i,1} \;.\;. & a_{i,2} & \cdots & a_{i,i} \end{pmatrix}, \quad \det(A^i) > 0,\ i = 1,\ldots,n$$

# Local minimizer

- **Example** $f(x,y) = e^{x^2+y^2}$

$$\nabla f(x) = \begin{pmatrix} 2xe^{x^2+y^2} \\ 2ye^{x^2+y^2} \end{pmatrix} = 0 \text{ has unique solution } x^* = (0,0)$$

$$\nabla^2 f(x^*) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} > 0 \rightarrow (0,0) \text{ is a minimizer of f}$$

# Local minimizer

- **Example** $f(x,y) = x^2 + y^2 - 2xy - x$

$$\nabla f(x) = \begin{pmatrix} -2x + 2y + 1 \\ -2x - 2y \end{pmatrix} = 0$$

has unique solution $x^* = (-1/4, 1/4)$

$$\nabla^2 f(x^*) = \begin{pmatrix} -2 & 2 \\ -2 & -2 \end{pmatrix}$$ is not positive definite

→ cannot conclude $x^*$

# Descent method

Determine starting point $x^{(0)} \in R^n$;

k $\leftarrow$ 0;

while( stop condition not reach){

Determine a search direction $p_k \in R^n$;

Determine a step size $\alpha_k > 0$ s.t. $f(x^{(k)} + \alpha_k p_k) < f(x^{(k)})$;

$x^{(k+1)} \leftarrow x^{(k)} + \alpha_k p_k$;

k $\leftarrow$ k+1;

}

Stop condition may be

- $||\nabla f(x^k)|| \leq \varepsilon$
- $||x^{k+1} - x^k|| \leq \varepsilon$
- k > K (maximum number of iterations)

# Gradient descent method

- Gradient descent schema

$$x^{(k)} = x^{(k-1)} - \alpha_k \nabla f(x^{(k-1)})$$

init $x^{(0)}$;
$k = 1$;
while stop condition not reach{
    specify constant $\alpha_k$;
    $x^{(k)} = x^{(k-1)} - \alpha_k \nabla f(x^{(k-1)})$;
    $k = k + 1$;
}

- $\alpha_k$ might be specified in such a way that $f(x^{(k-1)} - \alpha_k \nabla f(x^{(k-1)}))$ is minimized: $\frac{\partial f}{\partial \alpha_k} = 0$

# Minimize $f(x) = x^4 - 2x^3 - 64x^2 + 2x + 63$

```python
# compute the function value
def f(x):
    return x**4 - 2*x**3 - 64*x**2 + 2*x + 63

# compute the derivative value
def grad(x):
    return 4*x**3 - 6*x**2 - 128 * x + 2

# Gradient descent algorithm with given alpha and initial point
def myGD(alpha, x0):
    x = [x0]
    # loop to evaluate a series of candidate
    for it in range(100000):
        # x[k+1] = x[k] - alpha * f'(x[k])
        x_new = x[-1] - alpha*grad(x[-1])

        # check stop condition (f'(x[k+1]) <= epsilon)
        if abs(grad(x_new)) < 1e-3:
            break

        # append x[k+1] into list
        x.append(x_new)

    # return a list of evaluated candidates and the iteration at which the algorithm stops
    return (x, it)
```

# Minimize $f(x) = x^4 + 3x^2 - 10x + 4$

```
def grad(x):
    return 4*x**3+ 6*x - 10

def f(x):
    return x**4 + 3* x**2  - 10 * x + 4

def myGD(alpha, x0):
    x = [x0]
    for it in range(1000):
        x_new = x[-1] - alpha*grad(x[-1])
#           if abs(grad(x_new)) < 1e-3:
#               break

        if(abs(x[-1] - x_new) < 1e-3):
            break

        x.append(x_new)
    return (x, it)
```

# Minimize $f(x) = x^2 + 5sin(x)$

```
def grad(x):
    return 2*x+ 5*np.cos(x)

def f(x):
    return x**2 + 5*np.sin(x)

def myGD(delta, x0):
    x = [x0]
    for it in range(100):
        x_new = x[-1] - delta*grad(x[-1])
        if abs(grad(x_new)) < 1e-3:
            break
        x.append(x_new)
    return (x, it)
```

# Minimize $f(x, y) = x^2 + y^2 + xy - x - y$

```
def grad(x, y):
    return (2*x + y - 1, 2*y + x - 1)

def f(x, y):
    return x**2 + y**2 + x*y - x - y

def myGD(delta, x0, y0):
    X = [(x0, y0)]
    for it in range(1000):
        x_new = X[-1][0] - delta*grad(X[-1][0], X[-1][1])[0]
        y_new = X[-1][1] - delta*grad(X[-1][0], X[-1][1])[1]
        if abs(grad(x_new, y_new)[0]) < 1e-6 and abs(grad(x_new, y_new)[1]) < 1e-6:
            break
        X.append((x_new, y_new))
    return (X, it)
```

# Minimize $f(x) = x_1^2 + x_2^2 + x_3^2 - x_1x_2 - x_2x_3 + x_1 + x_3$

- $\alpha_k$ might be specified in such a way that $f(x^{(k-1)} - \alpha_k \nabla f(x^{(k-1)}))$ is minimized: $\frac{\partial f}{\partial \alpha_k} = 0$

```
def grad(x1, x2, x3):
    return [2*x1 + 1 - x2, -x1 + 2*x2 - x3, -x2 + 2*x3 + 1]

def f(x1, x2, x3):
    return x1**2 + x2**2 + x3**2 - x1*x2 - x2*x3 + x1 + x3

def myGD(v1, v2, v3):
    x1 = v1
    x2 = v2
    x3 = v3
    for it in range(1000):
        print(f(x1, x2, x3))
        [D1,D2,D3] = grad(x1,x2,x3)
        A = 2*x1*D1 + 2*x2*D2 + 2*x3*D3 - x1*D2 - x2*D1 - x2*D3 -x3*D2 + D1 + D3
        B = 2*D1*D1 + 2*D2*D2 + 2*D3*D3 -2*D1*D2 - 2*D2*D3
        if B == 0:
            break
        alpha = A/B

        x1 = x1 - alpha*D1
        x2 = x2 - alpha*D2
        x3 = x3 - alpha*D3

        val = grad(x1, x2, x3)
        if (val[0]**2 + val[1]**2 + val[2]**2) < 1e-6:
            break

        X.append([x1, x2, x3])
    return (X, it)
```

# CONTENT

- Unconstrained optimization problems
- Descent method
- Gradient descent method
- Newton method
- Subgradient method

# Newton method

- Second-order Taylor approximation $g$ of $f$ at $x$ is

$$f(x+h) \approx g(x+h) = f(x) + h\nabla f(x) + \frac{1}{2}h^2 \nabla^2 f(x)$$

- Which is a convex quadratic function of $h$
- g($x$+$h$) is minimized when $\frac{\partial g}{\partial h} = 0 \rightarrow h = -\nabla^2 f(x)^{-1} \nabla f(x)$

# Newton method

Generate $x^{(0)}$; // starting point

$k = 0$;

while stop condition not reach{

$x^{(k+1)} \leftarrow x^{(k)} - \nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)})$;

$k = k + 1$;

}

$$f(x) = x_1^2 + x_2^2 + x_3^2 - x_1 x_2 - x_2 x_3 + x_1 + x_3$$

# Newton method

```
import numpy as np


def newton(f,df,Hf,x0):
    x = x0
    for i in range(10):
        iH = np.linalg.inv(Hf(x))
        D = np.array(df(x)).T #transpose matrix: convert from list to
                                    #column vector
        print('df = ',D)
        y = iH.dot(D) #multiply two matrices
        if np.linalg.norm(y) == 0:
            break
        x = x - y
        print('Step ',i,': ',x,' f  = ',f(x))
```

# Newton method

```
def main():
    print('main start....')
    f = lambda x: x[0] ** 2 + x[1] ** 2 + x[2] ** 2 - x[0] * x[1] - x[1] *
                x[2] + x[0] + x[2]  # function f to be minimized
    df = lambda x: [2 * x[0] + 1 - x[1], -x[0] + 2 * x[1] - x[2], -x[1] + 2
                * x[2] + 1]  # gradient
    Hf = lambda x: [[2,-1,0],[-1,2,-1],[0,-1,2]]# Hessian
    x0 = np.array([0,0,0]).T
    newton(f,df,Hf,x0)


if __name__ == '__main__':
    main()
```

# Newton method

| Step | x | y | f |
|---|---|---|---|
| Initialization | [0,0,0] | [1, 1, 1] | 0 |
| Step 1 | [-1., -1., -1.] | [-2.46519033e-32  1.11022302e-16  2.22044605e-16] | -1.0000000000000004 |
| Step 2 | [-1., -1., -1.] | [0.,  0.,  0.] | -1 |

# CONTENT

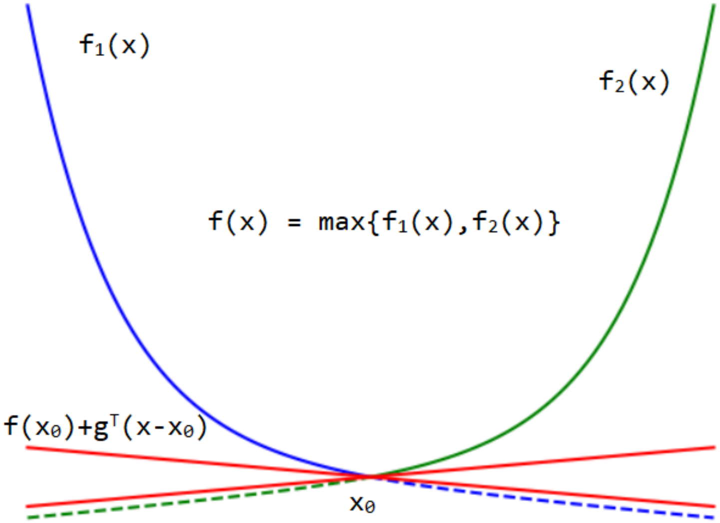
- Unconstrained optimization problems
- Descent method
- Gradient descent method
- Newton method
- Subgradient method

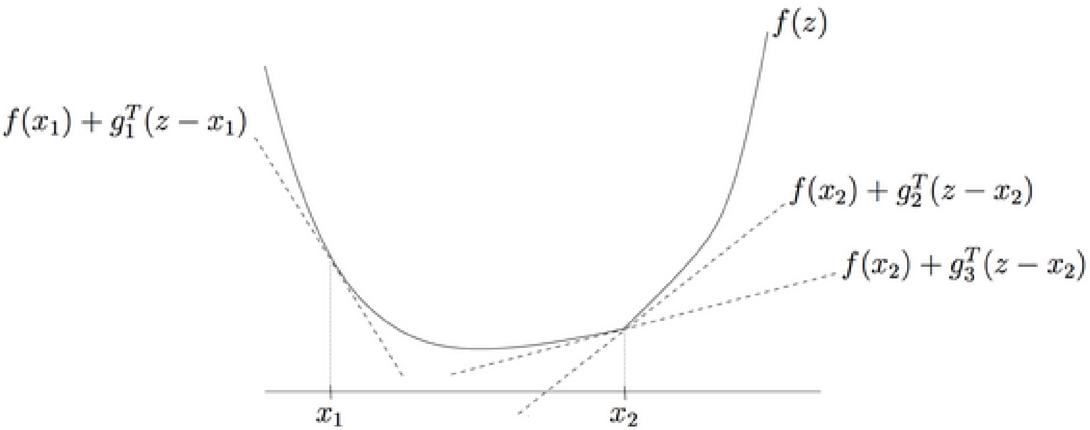
# Subgradient method

- For minimize nondifferentiable convex function
- Subgradient method is not a descent method: the function value can increase

# Subgradient method

- Subgradient of *f* at *x*
  - Any vector *g* such that $f(x') \geq f(x) + g^T(x'-x)$
  - If f is differentiable, only possible choice is $g^{(k)} = \nabla f(x^{(k)})$, → the subgradient method reduces to the gradient method



# Basic subgradient method

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

- $x^{(k)}$: is at the $k^{\text{th}}$ iteration
- $g^{(k)}$: any subgradient of $f$ at $x^{(k)}$
- $\alpha_k > 0$ is the $k^{\text{th}}$ step size
- Note: subgradient is not a descent method, thus $f_{best}^{(k)} = \min\{f(x^{(1)}), f(x^{(2)}), \ldots, f(x^{(k)})\}$

# Example

$$\text{minimize } f(x) = \max_{i=1,\ldots,m}(a_i^T x + b_i)$$

- Finding subgradient: given $x$, the index $j$ for which

$$a_j^T x + b_j = \max_{i=1,\ldots,m}(a_i^T x + b_i)$$

→ subgradient at $x$ is $g = a_j$

# Example

```
import numpy as np
def solve(A,b):
    f = lambda x: np.max(A.dot(x) + b)
    sg = lambda x: A[np.argmax(A.dot(x) + b)]
    x0 = [0,0,0,0]
    x = np.array(x0).T
    f_best = f(x)
    for i in range(100000):
        alpha = 2
        x = x - alpha*sg(x)
        if f_best > f(x):
            f_best = f(x)
    return f_best
```

# Example

```
def main():
    A = np.array([[1,-2,3,-5],[2,-2,1,1],[-3,2,-2,7]],dtype='double’)
    b = np.array([3,4,5]).T
    rs = solve(A,b)
    print('rs = ',rs)

if __name__ == '__main__':
    main()
```

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

# Thank you for your attentions!