Lecture 6
Calculus (IV)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

傅里叶分析
Fourier Analysis

- 在讲解傅里叶变换之前，我们先讲解泰勒展式。泰勒展式是研究函数特别是高阶可导函数的局部性质的重要工具。但我们这里是为了引入欧拉公式。

  Before explaining Fourier transform, we first explain Taylor series. Taylor series is an important tool for studying the local properties of functions, especially high-order differentiable functions. But here we are for introducing Euler's formula.

- 如果f是定义在区间$[x_0,\ x]$上的n次连续可微函数，则存在点$\xi \in [x_0,\ x]$，使得：

  If f is a function continuous and n times differentiable in an interval $[x_0,\ x]$, aka,$\xi f \in \mathcal{C}^{(n)}[x_0,\ x]$, then there exists $\xi \in [x_0,\ x]$, such that:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \frac{(x - x_0)^3}{3!}f^{(3)}(x_0) + \cdots + \frac{(x - x_0)^{n-1}}{(n-1)!}f^{(n-1)}(x_0) + \frac{(x - x_0)^n}{n!}f^{(n)}(\xi)$$

- 特别的，如果$\lim_{n\to\infty} \frac{(x-x_0)^n}{n!}f^{(n)}(x_0) = 0$，则有：

$$f(x) = \sum_{n=0}^{\infty} \frac{(x - x_0)^n}{n!}f^{(n)}(x_0)$$

傅里叶分析
Fourier Analysis

- 我们以$f(x)=x^n$为例来验证泰勒展式的正确性。

In the following we check the correctness of the Taylor expansion by considering $f(x)=x^n$.

$$f'(x)=nx^{n-1}, f''^{(x)}=n(n-1)x^{n-2}, \cdots, f^{(n)}(x)=n!\,x^{n-n}=n!$$

$$f(x)={x_0}^n+(x-x_0)n{x_0}^{n-1}+\frac{(x-x_0)^2}{2!}n(n-1)x^{n-2}+\cdots$$

$$+\frac{(x-x_0)^{n-1}}{(n-1)!}n(n-1)\cdots 2\cdot x_0+\frac{(x-x_0)^n}{n!}n!$$

$$={x_0}^n+\mathrm{C}_n^1(x-x_0){x_0}^{n-1}+\mathrm{C}_n^2(x-x_0)^2x^{n-2}+\cdots+\mathrm{C}_n^{n-1}(x-x_0)^{n-1}x_0+(x-x_0)^n$$

同时，由二项式公式，注意到$f(x)=x^n=\big((x-x_0)+x_0\big)^n$，则有：

Meanwhile, by the binomial formula, noting that $f(x)=x^n=\big((x-x_0)+x_0\big)^n$, we have:

$$\big((x-x_0)+x_0\big)^n$$

$$={x_0}^n+\mathrm{C}_n^1(x-x_0){x_0}^{n-1}+\mathrm{C}_n^2(x-x_0)^2x^{n-2}+\cdots+\mathrm{C}_n^{n-1}(x-x_0)^{n-1}x_0+(x-x_0)^n$$

傅里叶分析
Fourier Analysis

- 我们考查如下函数的泰勒展开： e^x、 $\sin x$， $\cos x$：

  We investigate the Taylor expansion of the function: e^x, $\sin x$, $\cos x$:

$$\frac{d^{(n)}e^x}{dx^n} = e^x, n = 1, 2, \cdots$$

$$e^x = e^0 + xe^0 + \frac{x^2}{2!}e^0 + \cdots + \frac{x^n}{n!}e^0 + \cdots = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$$

$$e^{xi} = 1 + xi - \frac{x^2}{2!} - \frac{x^3}{3!}i + \frac{x^4}{4!} + \frac{x^4}{5!}i + \cdots + \frac{x^n}{n!} + \cdots$$

$$= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} + \cdots\right) + i\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots\right)$$

$$\cos x = \cos 0 - x\sin 0 - \frac{x^2}{2!}\cos 0 + \frac{x^3}{3!}\sin 0 + \frac{x^4}{4!}\cos 0 + \cdots = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} + \cdots$$

$$\sin x = \sin 0 + x\cos 0 - \frac{x^2}{2!}\sin 0 - \frac{x^3}{3!}\cos 0 + \frac{x^4}{4!}\sin 0 + \cdots = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

$$\Longrightarrow e^{xi} = \cos x + i\sin x$$

傅里叶分析
Fourier Analysis

- 进一步地，我们有以下式子成立：

  Now we have the following equations hold:

$$\cos x = \frac{e^{ix} + e^{-ix}}{2}, \sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

- 下面，我们回忆在傅里叶级数中，关于傅里叶系数的计算：

  Below, we recall the calculation of Fourier coefficients in Fourier series:

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\cos\frac{2\pi}{T}nx\,dx\,, b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\sin\frac{2\pi}{T}nx\,dx$$

- 定义c_n如下，则我们有：

  If we define c_n as follows, then we have:

$$c_n = \begin{cases} \dfrac{a_n - ib_n}{2} & n > 0 \\ \dfrac{a_n + ib_n}{2} & n < 0 \end{cases}$$

傅里叶分析
Fourier Analysis

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(x)\cos\frac{2\pi}{T}nx\,dx - i\frac{1}{T}\int_{-T/2}^{T/2} f(x)\sin\frac{2\pi}{T}nx\,dx = \frac{1}{T}\int_{-T/2}^{T/2} f(x)e^{-i\frac{2\pi}{T}nx}dx$$

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n\cos\frac{2\pi}{T}nx + b_n\sin\frac{2\pi}{T}nx\right)$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty}\left((c_n + c_{-n})\cos\frac{2\pi}{T}nx + \frac{c_n - c_{-n}}{-i}\sin\frac{2\pi}{T}nx\right)$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(c_n\cos\frac{2\pi}{T}nx + ic_n\sin\frac{2\pi}{T}nx\right)$$

$$+ \sum_{n=1}^{\infty}\left(c_{-n}\cos\frac{2\pi}{T}(-n)x + ic_{-n}\sin\frac{2\pi}{T}(-n)x\right)$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} c_n e^{i\frac{2\pi}{T}nx} + \sum_{n=-1}^{-\infty} c_n e^{i\frac{2\pi}{T}nx}$$

- 注意到$c_0 = a_0/2$，则我们有：Note $c_0 = a_0/2$, then we have:

$$f(x) = c_0 + \sum_{n=1}^{\infty} c_n e^{i\frac{2\pi}{T}nx} + \sum_{n=-1}^{-\infty} c_n e^{i\frac{2\pi}{T}nx} = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi}{T}nx}$$

傅里叶分析
Fourier Analysis

- 实际上，当我们从时域变换到频域时，真正对我们有意义的是c_n的值；并且，我们可以借助c_n，表示原函数。当$f(x)$不是周期函数时，当$f(x)$的定义域有限，则我们可以做周期延拓，此时，我们往往将$\frac{n}{T}$记为μ，即：

  In fact, when we transform from the time domain to the frequency domain, the value of c_n is what really matters to us; and we can use c_n to represent the original function. When $f(x)$ is not a periodic function and its domain is finite, we can extend it periodically, in this situation, we often denote $\frac{n}{T}$ as μ:

$$F(\mu) = \frac{1}{T}\int_{-T/2}^{T/2} f(x)e^{-i2\pi\mu x}dx$$

- 有时，我们会进一步将2π与μ结合起来，记为ω，如下式所示。一般地，我们将$F(\mu)$或$F(\omega)$称为关于函数$f(x)$的傅里叶变换，无论$f(x)$是周期函数，还是非周期函数。

  Sometimes, we will further combine 2π and μ together and denoted as ω, and rewrite the above formula as below. Either $F(\mu)$ or $F(\omega)$, we usually call it the Fourier transform of the function $f(x)$, no matter $f(x)$ is a periodic function or non-periodic one.

$$F(\omega) = \frac{1}{T}\int_{-T/2}^{T/2} f(x)e^{-i\omega x}dx$$

傅里叶分析
Fourier Analysis

- 此时，我们对$f(x)$的傅里叶级数也可以考虑非周期的情况，如下式所示。在概念上不严格的场合，可能会叫傅里叶合成，特别是对下面将要讲到的离散的情况：

  Now we can consider the non-periodic case of the function $f(x)$, as shown below. In some scenario where mathematics rigor is not the first priority, it might be called as Fourier synthesis:

$$f(x) = \int_{-\infty}^{+\infty} F(u)e^{2\pi iux}du = \int_{-\infty}^{+\infty} F(\omega)e^{i\omega x}d\omega$$

- 当$f(x)$定义域无限时，则我们可以假设周期是无穷大，我们不再证明其傅里叶变换与傅里叶合成具有如下形式。

  when the domain is infinite, we can assume that the period is infinite. We just give its Fourier analysis and Fourier synthesis below without proof.

$$F(\omega) = \int_{-\infty}^{+\infty} f(x)e^{-i\omega x}dx$$

$$f(x) = \frac{1}{2\pi}\int_{+\infty}^{+\infty} F(\omega)e^{i\omega x}d\omega$$

离散傅里叶变换
Discrete Fourier Transform

- 我们知道在实际应用中，计算机只能处理离散的信号；对于一个连续的非周期信号，则时域和频域都是连续的非周期的，我们应该怎么处理呢？在讲这个问题之前，我们先引入一些关于信号的概念。从数学上讲，信号一般是关于一个或两个变元的函数，而时间与空间是常见的变量。

  We know that in practical applications, computers can only process discrete signals. For a continuous non-periodic signal, both the time domain and frequency domain are continuous and non-periodic. How should we handle this? Before discussing this issue, let‘s introduce some concepts about signals. A signal might be a function of one or of several variables, space and time are very common variables.

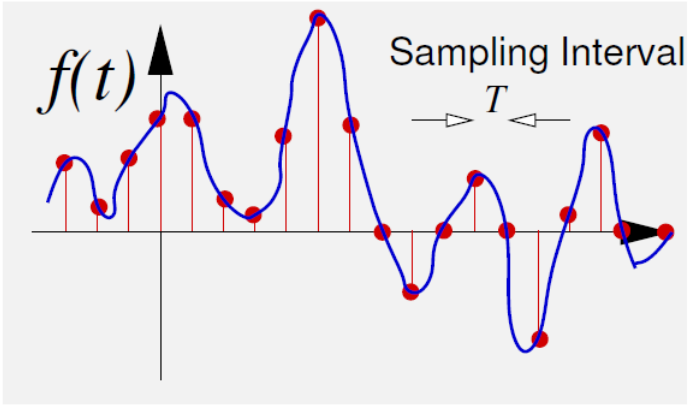
- 通常，信号倾向于使用单一时间变量t。若此时幅值（即函数的取值）是给定范围内的连续值，则称信号为模拟信号。

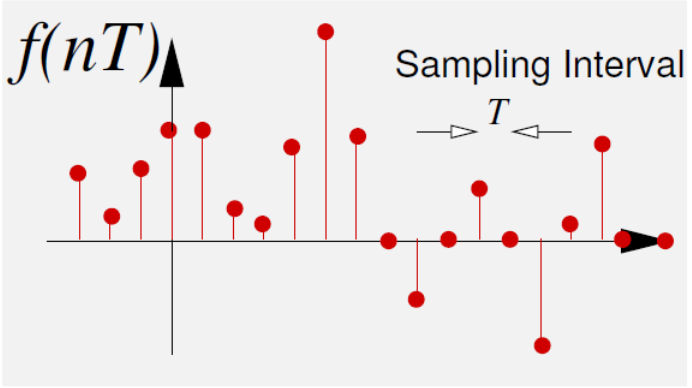
  Generally, a signal tend to stick to the time variable t. If at the same time the amplitude of the signal covers a continuous range, it is called an analogue signal.

离散傅里叶变换
Discrete Fourier Transform

- 若模拟信号在其参数t的连续范围内均有值，则连续时间模拟信号，记为$f(t)$。注意连续时间信号不一定是连续函数。若信号仅在某些时刻具有模拟值，则信号称为离散时间模拟信号，记为$f(nT)$，其中n是整数，T是采样间隔。显然函数有值的时刻是以固定的采样周期T发生的。

  A continuous-time analogue signal is one that has a value for a continuous sweep of values of its parameter t, denoted as $f(t)$. Notice that a continuous-time signal does not have to be a continuous function. A discrete-time analogue signal is one that has an analogue value at certain times only, usually denoted by $f(nT)$, where n is an integer, and T is the sampling interval. It is obvious that f only has values at regular intervals, which are arising from regular sampling.





离散傅里叶变换
Discrete Fourier Transform

- 当离散时间模拟信号的幅度只能取有限的离散值集合中的值时，称为数字信号。此时，信号的取值范围通常由某种二进制编码方案表示，称为量化。假设我们使用4位来表示，此时只能取0到15共16个值。即我们无法精确表示原先取值范围内的任一值。数字信号通常来自于对模拟信号在离散时间进行采样，再对幅值进行量化。

  When the amplitude of a continuous-time analogue signal is constrained to a discrete set of values represented by some binary coding scheme, namely, quantization, then it is called a digital signal. Suppose we used 4 bits, it could be capable of representing 0 to 15 these 16 numbers, which means it cannot represent any value in the original range. Digital signals usually arise from sampling at discrete times, followed by a proper quantization.

- 为了推导离散傅里叶变换，我们先推导离散时间傅里叶变换。此时，我们要首先把采样过程数学化。令采样频率为f_s，则采样点时间间隔为$T_s = \frac{1}{f_s}$，我们定义冲击采样序列为：

  In order to derive the discrete Fourier transform, we first derive the discrete time Fourier transform. At this point, we need to mathematically formalize the sampling process. Let the sampling frequency be f_s, then the time interval between sampling $T_s = \frac{1}{f_s}$. We define the impulse sampling sequence as follows:

$$\delta_s(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_s)$$

离散傅里叶变换
Discrete Fourier Transform

- 则采样之后的信号可以表示为：

  The sampled signal can be represented as follows:

$$f_s(t) = \sum_{n=-\infty}^{+\infty} f(t)\delta(t - nT_s)$$

- 根据连续信号的傅里叶变换的定义，采样后的傅里叶变换可推导如下：

  According to the definition of the Fourier transform of a continuous signal, the Fourier transform after sampling can be derived as:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dt$$

$$F_s(\omega) = \int_{-\infty}^{+\infty} \left(\sum_{n=-\infty}^{+\infty} f(t)\delta(t - nT_s) \right) e^{-i\omega t}dt = \sum_{n=-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(t)\delta(t - nT_s)e^{-i\omega t}dt$$

$$= \sum_{n=-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}\delta(t - nT_s)dt = \sum_{n=-\infty}^{+\infty} f(nT_s)e^{-i\omega nT_s}$$

离散傅里叶变换
Discrete Fourier Transform

- 实际上，对数字系统而言，我们关注$f(t)$采样后的信号$\{f(n)\}$，该序列第n个采样发生在时间$t = nT_s$，因此离散时间傅里叶变换可写成下式所示：

  In fact, for digital systems, we are concerned with the sampled signal $\{f(n)\}$ of $f(t)$, where the n-th sample of the sequence occurs at time $t = nT_s$. Therefore, the discrete time Fourier transform can be written as follows:

$$F_s(\omega) = \sum_{n=-\infty}^{+\infty} f(n)e^{-i\omega nT_s}$$

- 在上面推导离散时间傅里叶变换的过程中，原始信号假设是无限长的，采样后，采样点也有无限多个，实际上，并不存在无限长的信号，实际中的信号总是有限的，处理这样的信号，我们需要引入离散傅里叶变换，做法就是将无限长的离散信号进行截短至N个采样点，然后将这个N个采样点进行周期延拓，变成周期信号。

  In deriving the discrete Fourier transform, it assumes the original signal lasts forever. As a consequence, there are also infinite number of sampled points. However, in reality the signals are always finite which incurs the finite number of sampled point. Hence, we introduce the discrete Fourier transform by cutting the infinite number of sampled points into totally N sampled points. Then way extend these sampled points in a periodic way to force it into periodic signals.

离散傅里叶变换
Discrete Fourier Transform

- 根据上面想法，我们将连续信号$x(t)$按照采样时间T_s进行采样N次，并将这N个数值进行周期延拓，可以得到周期的离散信号$x(n) = x(t)\delta(t - nT_s)$，其周期为$T = NT_s$，频率为$f = 2\pi/T$。在一个周期$T$内，其表达式为：

  Based on the given idea, we will sample the continuous signal $x(t)$ N times with a sampling time T_s, and extend these N values periodically. This will yield a periodic discrete signal $x(n) = x(t)\delta(t - nT_s)$, with a period of $T = NT_s$ and a frequency of $f = 2\pi/T$. Within one period T, its expression is as follows:

$$x_s(t) = \sum_{n=0}^{N-1} x(t)\delta(t - nT_s)$$

- 令$\omega = f = 2\pi/T$，则$x_s(t)$的傅里叶级数为：

  Set $\omega = f = 2\pi/T$, then the Fourier series of $x_s(t)$ is:

$$X[k\omega] = \frac{1}{T}\int_0^T \left(\sum_{n=0}^{N-1} x(t)\delta(t - nT_s)\right) e^{-i\frac{2\pi}{T}kt} dt$$

离散傅里叶变换
Discrete Fourier Transform

- 进一步地，由δ函数的性质，令$X[k\omega]\cdot T_s = X[k]$，有：

  Further, due to the property of δ function, we have:

$$X[k\omega] = \frac{1}{T}\sum_{n=0}^{N-1} x(nT_s)e^{-i\frac{2\pi}{NT_s}knT_s} = \frac{1}{NT_s}\sum_{n=0}^{N-1} x(nT_s)e^{-i\frac{2\pi}{N}kn}$$

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x(nT_s)e^{-i\frac{2\pi}{N}kn} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}kn}$$

- 上面即离散周期信号的傅里叶变换。理论上离散周期信号的频谱是有无穷多的，但是由于$e^{-i\frac{2\pi}{N}kn}$的周期性，一般我们只需取主值区间$0 \le k \le N-1$进行研究。其傅里叶逆变换或傅里叶合成如下式所示：

  The above is the Fourier transform of a discrete periodic signal. In theory, the frequency spectrum of a discrete periodic signal is infinite. However, due to the periodicity of $e^{-i\frac{2\pi}{N}kn}$, we usually only need to focus on the interval $0 \le k \le N-1$ where the principal value resides. The inverse Fourier transform or Fourier synthesis is as follows:

$$x[n] = \sum_{k=0}^{N-1} X[k]e^{-i\frac{2\pi}{N}nk}$$

离散傅里叶变换
Discrete Fourier Transform

- 下面代码展示了TensorFlow对离散傅里叶变换的支持：

  The following code demonstrates the support of TensorFlow for discrete Fourier transform:

```
import tensorflow as tf
x = tf.random.normal([16])
print(f"x => {x}")
X = tf.signal.fft(tf.cast(x, dtype=tf.complex64))
print(f"X => {X}")
# x => [-1.5582364 -0.8480835 -0.9617148 0.04200707 0.996157 0.91941357 0.00929826
0.43063083 1.0458945 0.52699715 -2.4119396 -1.5002035 1.6532117 -1.0652202 -0.9331501
0.07042459]
# X => [-3.5845137 +0.j -4.0175824 -3.9048243j -1.9002519 +3.959144j -3.2185025 -0.0616436j
 6.434533 -0.49024794j -2.7078633 +4.636216j -4.423169 -0.9404607j -0.47257495 -1.8351824j
-0.73644555 +0.j -0.4725747 +1.8351827j -4.4231696 +0.9404607j -2.7078633 -4.6362166j
 6.434533 +0.49024794j -3.2185025 +0.06164384j -1.9002521 -3.959144j -4.017582 +3.904824j ]
```

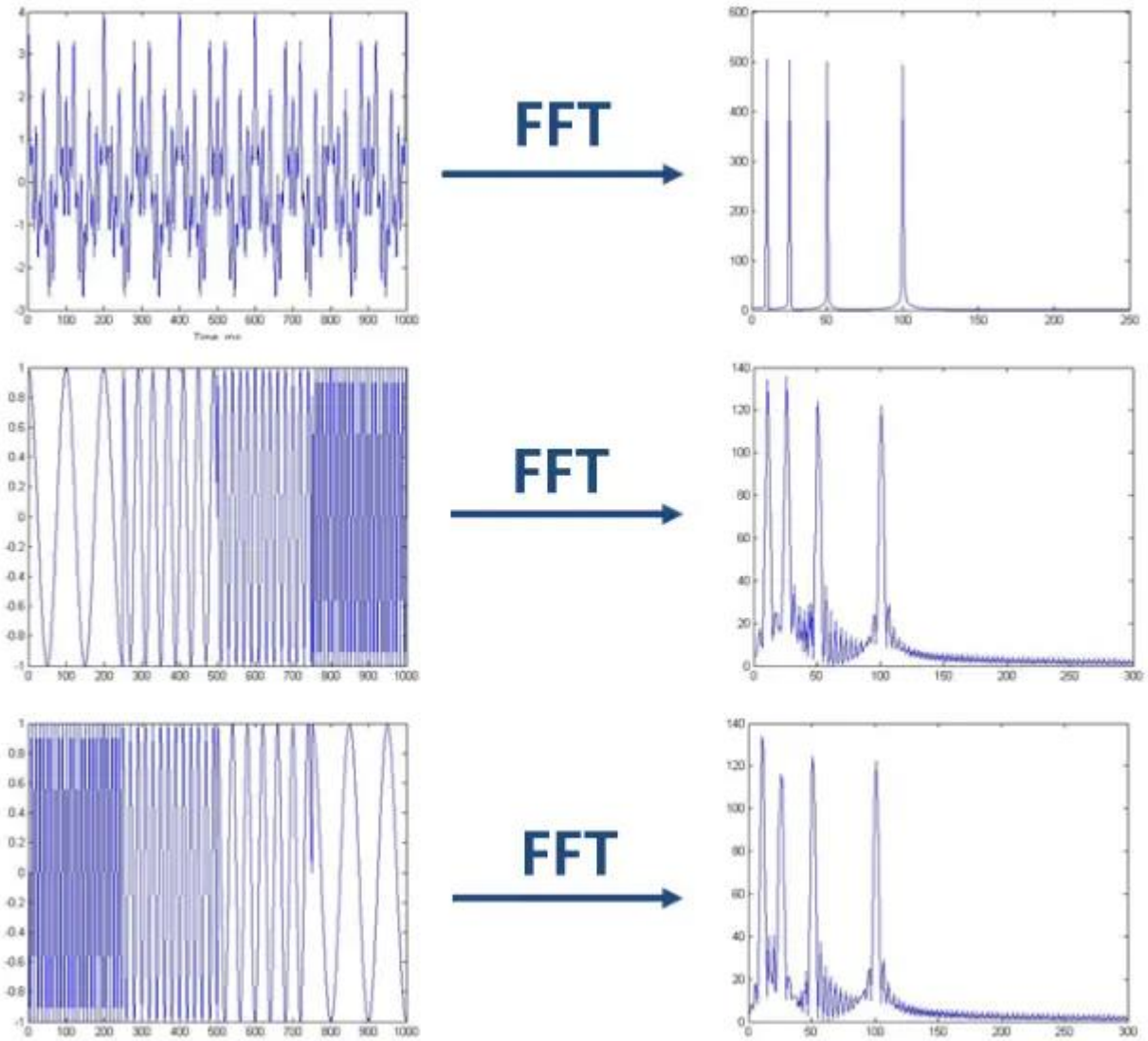
离散傅里叶变换
Discrete Fourier Transform

```
x_ = tf.signal.ifft(X)

print(f"x_ => {x_}")

x_ = tf.math.real(x_)

print(f"real x_ => {x_}")

tf.debugging.assert_near(x, x_, message="The original value x is too different from x_")

# x => [ 0.4364555 +2.9802322e-08j -1.1263499 +0.0000000e+00j  1.4907451 -2.9802322e-08j  0.30926138+0.0000000e+00j  0.25491363+2.5208543e-08j -0.23080462+5.9604645e-08j -0.7524136  +0.0000000e+00j -1.4539831  -4.4703484e-08j -0.19762635-2.9802322e-08j 0.67965126-5.9604645e-08j -0.21874231+2.9802322e-08j -0.8433911 -2.9802322e-08j

 -1.4368029  -2.5208543e-08j  0.8967164  +0.0000000e+00j  0.6258747  +0.0000000e+00j -0.21351022+7.4505806e-08j]

# real x_ => [ 0.4364555  -1.1263499  1.4907451  0.30926138  0.25491363 -0.23080462 -0.7524136  -1.4539831  -0.19762635  0.67965126  -0.21874231  -0.8433911  -1.4368029  0.8967164  0.6258747  -0.21351022]
```

离散傅里叶变换
Discrete Fourier Transform

- 我们生活中常见的声音、图像等都是信号。尽管在时域上根据特定知识，可以对信号有一定的感性认识，但如果将信号变换到频域，则通常会更容易看出特征，如低频信号、中频信号、高频信号等。

  The sounds and images we commonly encounter in our daily lives are signals. Although we can have a certain intuitive understanding of signals in the time domain based on our knowledge, it is often easier to identify their characteristics, such as low-frequency signals, intermediate-frequency signals, high-frequency signals, etc., by transforming them into the frequency domain.



离散傅里叶变换
Discrete Fourier Transform

- 由于在频域上信号特征比较明显，因此一个直观的想法便是将信号从时域变换到频域进行处理。这通常会借助快速傅里叶变换进行。但由于技术的发展，计算能力的提高，因此，之前在频域进行处理的方式，现在可以直接在时域进行。但这需要如下定理，即频域的滤波等于时域的卷积：

  Since the characteristics of a signal are more evident in the frequency domain, an intuitive approach is to transform the signal from the time domain to the frequency domain for processing. This is commonly done using the Fast Fourier Transform (FFT). However, with the advancements in technology and increased computational power, it is now possible to perform processing directly in the time domain, which was previously done in the frequency domain. However, this requires the following theorem: filtering in the frequency domain is equivalent to convolution in the time domain.

- 在频域进行滤波通常是直观的，通常将代表滤波器的窗口函数直接与经过傅里叶变换的信号相乘，然后再进行傅里叶逆变换。

  Filtering in the frequency domain is typically intuitive. It involves directly multiplying the window function representing the filter with the signal that has undergone Fourier transform, and then performing the inverse Fourier transform.

离散傅里叶变换
Discrete Fourier Transform

- 时域的卷积从数学角度来定义更简洁：给定两个函数$f(t)$与$g(t)$，它们的卷积$(f*g)(t)$如下形式定义的积分：$(f*g)(t)=\int f(\tau)g(t-\tau)d\tau$，称为函数$f(t)$与$g(t)$的卷积。即卷积运算是将$f(\tau)$的每个值与对应的$g(t-\tau)$的值相乘，并对得到的乘积在所有可能的$\tau$值上进行积分。卷积运算被广泛应用于数学、工程和信号处理领域。

  The definition of convolution in time domain is more concise from the mathematical perspective: given two functions, $f(t)$与$g(t)$, their convolution, denoted as $(f*g)(t)$, is defined as the integral $(f*g)(t)=\int f(\tau)g(t-\tau)d\tau$. In this definition, the convolution operation involves multiplying each value of $f(\tau)$ with the corresponding value of $g(t-\tau)$ and integrating the resulting product over all possible values of τ. The convolution operation is widely used in mathematics, engineering, and signal processing.

- 将时域的卷积与频域滤波联系起来的是卷积定理：如果$F(f)$表示函数$f(t)$的傅里叶变换，而$F^{-1}(F)$表示函数$F(\omega)$的傅里叶逆变换，则如下式子成立：

  What relates convolution in the time domain to filtering in the frequency domain is the Convolution Theorem. Let $F(f)$ represents the Fourier transform of function $f(t)$, and $F^{-1}(F)$ represents the inverse Fourier transform of function $F(\omega)$, then the following equation holds:

$$f*g=F^{-1}\big(F(f)\times F(g)\big)$$

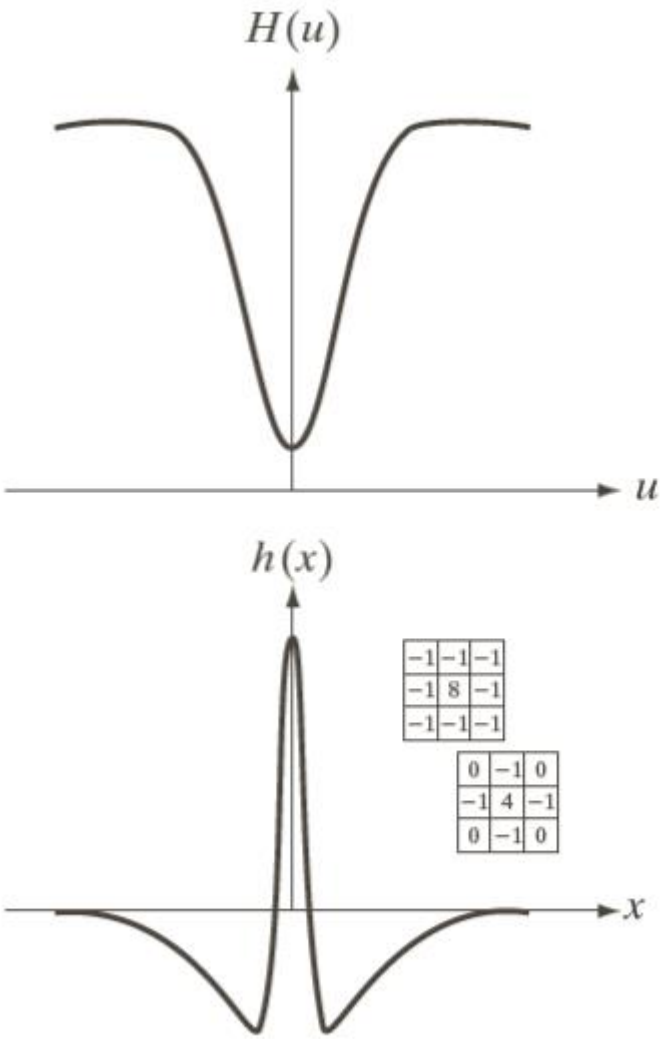
离散傅里叶变换
Discrete Fourier Transform

- 实际上，滤波是在传统信号处理中是如此之有效与重要，因此，自然是工程领域依仗的手段之一。如果算力的发展能在时域上进行，则不必再费时费力转换到频域再转回来。但如何针对特定问题选择合适的滤波器往往是一个难题。

  In fact, filtering is incredibly effective and important in traditional signal processing, making it one of the first-rate tools in engineering. If computational power allows for processing in the time domain, there is no need to go through the Fourier and inverse Fourier transform. However, choosing the appropriate filter for specific problems is often a challenging task.

- 通常情况下，如果对所解决的问题需要用的滤波器比较熟悉的话，可以首先在频域上进行设计，接着利用傅里叶逆变换变换到时域，然后进行采样，得到卷积核：

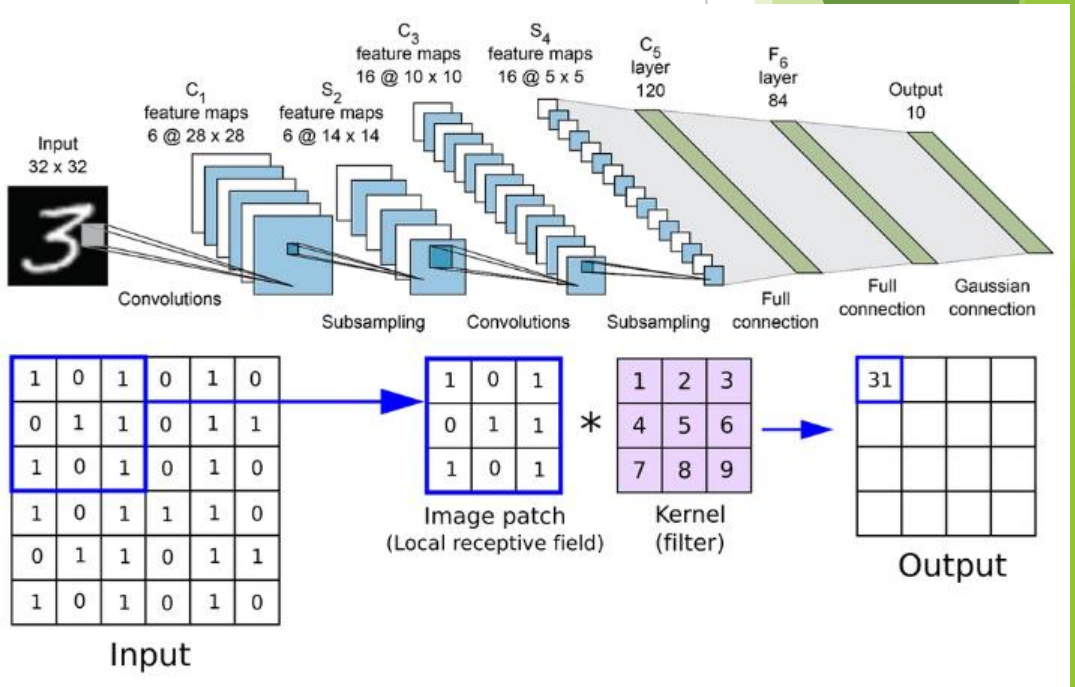
  Usually, if you are familiar with the filter required for the targeting problem, you can first design it in the frequency domain, followed by transforming it to the time domain. Then, you can sample it to obtain the intended convolution kernel.



离散傅里叶变换
Discrete Fourier Transform

- 在深度学习背景下，卷积核的确定，是依靠学习方法，即反向传播算法得到的。因为当我们所解决的问题类似图像分类、目标侦测等非线性问题时，我们并不知道什么样的卷积核最适用于该问题，我们只能借助于构造神经网络，进行端到端的训练，来得到问题的答案，即最优卷积核的问题，尽管这里的卷积实际上是互相关。

- In the context of deep learning, the determination of convolutional kernels relies on the learning methods, specifically, the backpropagation algorithm. This is because for non-linear problems such as image classification, object detection, and similar tasks, we do not know beforehand which convolutional kernels are most suitable. Instead, we construct neural networks and perform end-to-end training to obtain the solution to the problem, which involves finding the optimal convolutional kernels. It is important to note that in this context, the convolution operation is actually cross-correlation.



离散傅里叶变换
Discrete Fourier Transform

- 下面代码展示了利用Numpy库对卷积定理进行验证：
  The following code demonstrates the proof of convolution theorem by using Numpy:

```
import numpy as np
import tensorflow as tf

f = tf.random.normal([16])
g = tf.random.normal([16])
F = tf.signal.fft(tf.cast(f, dtype=tf.complex64))
G = tf.signal.fft(tf.cast(g, dtype=tf.complex64))
FG = F*G
fg_inv = tf.signal.ifft(FG)
fg_real = tf.math.real(fg_inv)

print("fg_real -> {}".format(fg_real))
```

```
fg_real -> [-2.02225666 -8.38421999 -2.80856457
4.18250688  1.25864585  3.35757428
 -2.15035387  6.00664167  7.44348496  3.55013951
-3.67200832  3.60799784
  3.66479749 -3.23781187 -8.57165387  4.45650584]
```

离散傅里叶变换
Discrete Fourier Transform

```
# List of shifts for each row
shifts = tf.range(g.shape[0]) + 1
g = tf.reverse(g, axis=[0])
t = tf.repeat(tf.expand_dims(g, axis=0), repeats=[g.get_shape().as_list()[0]], axis=0)
# Function to roll each row by its index
@tf.function
def roll_rows(tensor, shifts):
    result = tf.TensorArray(dtype=tensor.dtype, size=tensor.shape[0])
    for i in tf.range(tensor.shape[0]):
        rolled_row = tf.roll(tensor[i], shift=shifts[i], axis=0)
        result = result.write(i, rolled_row)
    return result.stack()
```

```
# Apply the function to roll each row
fg_conv = tf.linalg.matmul(roll_rows(t, shifts), tf.expand_dims(f, axis=-1))
fg_conv = tf.squeeze(fg_conv, axis=-1)


# Print the result
print("fg_conv -> {}".format(fg_conv))
```

```
fg_conv -> [-2.02225666 -8.38421999 -2.80856457
4.18250688  1.25864585  3.35757428
 -2.15035387  6.00664167  7.44348496  3.55013951
-3.67200832  3.60799784
  3.66479749 -3.23781187 -8.57165387  4.45650584]
```