

Table of Contents

- 1 FFTの応用
 - 1.1 周波数分解
 - 1.2 python code
 - 1.3 高周波フィルター
 - 1.4 python code
- 2 FFTの動作原理
- 3 関数内挿としてのFourier関数系
- 4 直交関係からの積分による係数決定
- 5 直接積分によるフーリエ係数
- 6 選点直交性による計算の簡素化
 - ○ ○ 6.0.0.1 直交関数系の選点直交性
 - 6.1 選点直交性を用いた結果
- 7 高速フーリエ変換アルゴリズムによる高速化
- 8 FFT関数を用いた結果
- 9 課題と解答例
 - 9.1 合成波のFFT

FFT(Fast Fourier Transformation)

file:/Users/bob/Github/TeamNishitani/jupyter_num_calc/fft
https://github.com/daddygongon/jupyter_num_calc/tree/master/notebooks_p
cc by Shigeto R. Nishitani 2017-19

FFTの応用

Fast Fourier Transformation(FFT)高速フーリエ変換(あるいはデジタル(離散)フーリエ変換(DFT))は、周波数分解やフィルターを始め、画像処理などの多くの分野で使われている。基本となる考え方とは、直交基底による関数の内挿法である。最初にその応用例を見た後、どのような理屈でFFTが動いているかを解説する。

いくつかFFTを解説する面白い動画があります。ただ、少し高度かも。 . .

1. 初音ミクを三角関数で描いてみた
2. A visual introduction.
3. From heat flow to circle drawings

周波数分解

はじめの例は、周波数分解。先ずは、非整合な波を二つ用意しておく。



これを重ねあわせた波を作る。



ゆっくり変化する波に、激しく変化する波が重なっていることが読み取れる。これにFFTを掛ける その強さを求めて、周波数で表示すると、



もとの2つの周波数に対応するところにピークができているのが確認できる。広がりは、誤差のせい。logplotでも良い。

python code

scipyにあるfft, ifftを使う。

```
In [1]: %matplotlib inline
from scipy import fft
import matplotlib.pyplot as plt
import numpy as np

def func(x):
    return np.sin(x/13)+np.sin(x/2)

x = np.linspace(0, 256, 256) #0から2πまでの範囲を100分割したnumpy配列
plt.plot(x, func(x), color = 'b')
plt.plot(x, np.sin(x/13), color = 'r', linewidth=0.8)
plt.plot(x, np.sin(x/2), color = 'r', linewidth=0.8)

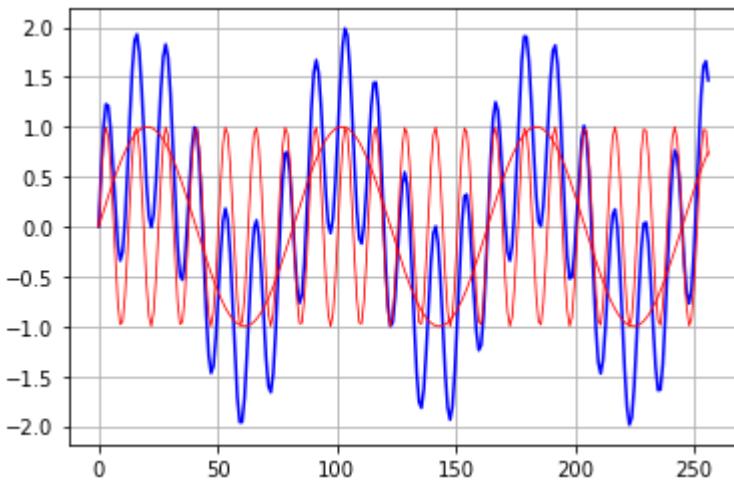
plt.grid()
plt.show()

yy = func(x)
out = fft(yy)

def spectrum_power(x):
    re, im = x.real, x.imag
    return np.sqrt(re**2+im**2)

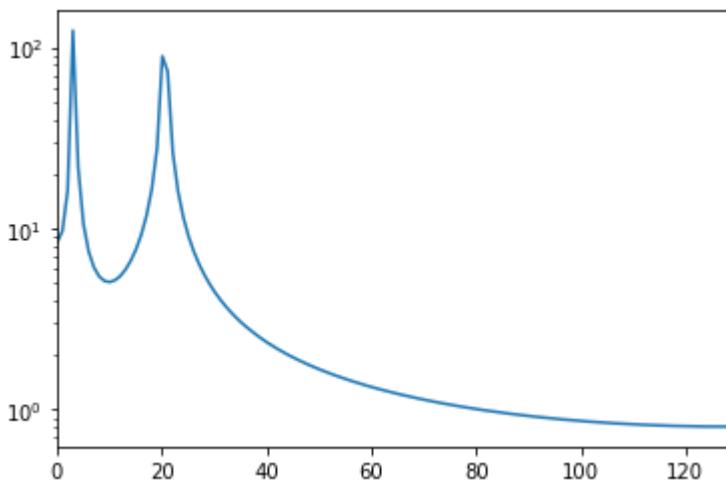
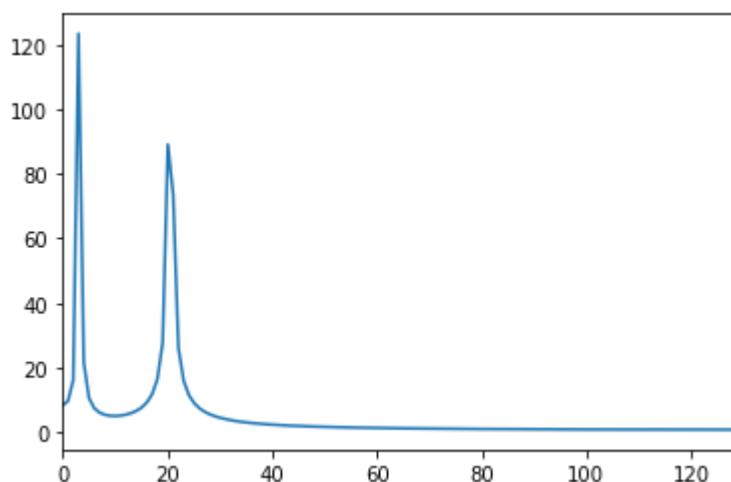
plt.plot(x,spectrum_power(out))
plt.xlim(0,128)
plt.show()

plt.plot(x,spectrum_power(out))
plt.xlim(0,128)
plt.yscale('log')
plt.show()
```



<ipython-input-1-c0c7b5db2ea9>:20: DeprecationWarning: Using scipy.fft as a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.

```
out = fft(yy)
```



高周波フィルター

次の例は、高周波フィルター。たとえば次のようなローレンツ関数を考える。

```
maple
> restart;
f1:=t->subs(a=10,b=40000,c=380,d=128,a+b/(c+(t-d)^2));
```

$$f1 := t \mapsto 10 + \frac{40000}{380 + (t - 128)^2}$$

```
maple
> plot(f1(t),t=1..256);
```



これにノイズがのると、次のようになる。

```
maple
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
#T:=[seq(evalf(rand()/10^12),i=1..256)]: #これはホワイトノイズ
#T:=[seq(f1(i),i=1..256)]: #これは元の関数そのまま
with(plots):
listplot(T);
```



これに高周波フィルターを掛けるとノイズが消えるが、その様子を示そう。先ずは、FFTをかける。

```
maple
> Idata:=array([seq(0,i=1..256)]):
Rdata:=convert(T,array):
FFT(8,Rdata,Idata);
```

256

これは次のような強度分布をもっている。

```
maple
> Adata:=[seq([i,sqrt(Idata[i]^2+Rdata[i]^2)],i=1..128)]:
> logplot(Adata);
```



低周波の部分に、ゆっくりとした変化を表す成分が固まっている。次のような三角フィルターを用意する。これは、低周波ほど影響を大きくするフィルター。

```
maple
> filter:=x->piecewise(x>=0 and x<=20,(1-x/20)): #三角フィルター
#filter:=x->piecewise(x>=0 and x<=20,1); #方形フィルター
plot(filter(x),x=0..40);
```



これとデータを各点で掛けあわせる事によって、フィルターを通したことになる。

```
maple
> FRdata:=array([seq(Rdata[i]*filter(i),i=1..256)]):
> FIdata:=array([seq(Idata[i]*filter(i),i=1..256)]):
```

先ほどと同様に表示すると

```
maple
> Bdata:=[seq([i,sqrt(FIdata[i]^2+FRdata[i]^2)],i=1..128)]:
> logplot(Bdata);
```



$i = 20$ 以上の領域がフィルターによってちぎり切られていることが確認できる。これを逆フーリエ変換する。

```
maple  
> iFFT(8, FRdata, FIdata);
```

256

これを表示すると、

```
maple  
> listplot(FRdata);
```



となる。ノイズが取り除かれているのが確認できる。元の関数に加えたホワイトノイズにFFTを掛ければ分かるが、全周波数域にわたって均質に広がった関数となる。これを三角フィルターなどで高周波成分をカットすることで、ノイズが取り除かれていくのが理解されよう。

python code

In [4]:

```
%matplotlib inline  
from scipy import fft, ifft  
import matplotlib.pyplot as plt  
import numpy as np  
  
def func(x):  
    a,b,c,d=10, 40000, 380, 128  
    return a+b/(c+(x-d)**2)  
  
xdata = np.linspace(0, 256, 256)  
y = func(xdata)  
y_noise = 10 * np.random.normal(size=xdata.size)  
ydata = y + y_noise  
plt.plot(xdata, ydata, 'b-', label='data')  
plt.grid()  
plt.show()  
  
out = fft(ydata)  
  
def spectrum_power(x):  
    re, im = x.real, x.imag  
    return np.sqrt(re**2+im**2)  
  
plt.plot(x,spectrum_power(out))  
plt.xlim(0,128)  
plt.show()  
  
def filter(x):  
    return (1-x/20)*np.piecewise(x, [x < 20, x >= 20], [1, 0])  
  
plt.plot(x,filter(x))  
plt.xlim(0,128)  
plt.show()  
  
filtered_out = []  
for i in range(0, 256):
```

scipy.fft] ← 正立
エラーの場合
ifft 戻り値

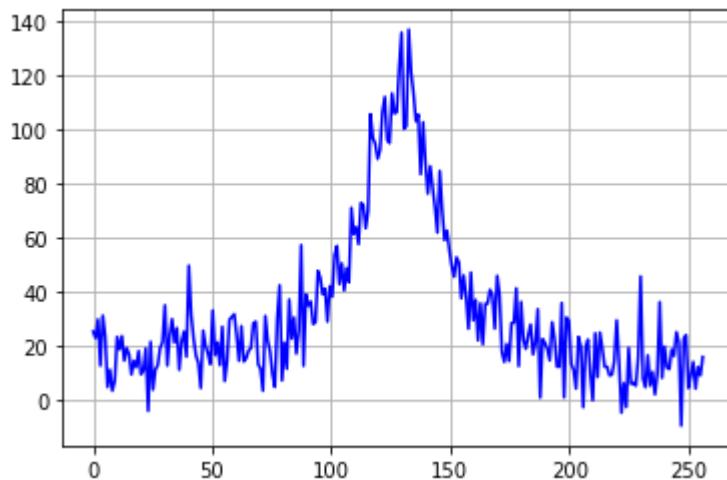
```

y = out[i]
re, im = y.real, y.imag
yy = complex(filter(i)*y.real,filter(i)*y.imag)
filtered_out.append(yy)

plt.plot(x,filtered_out)
plt.show()

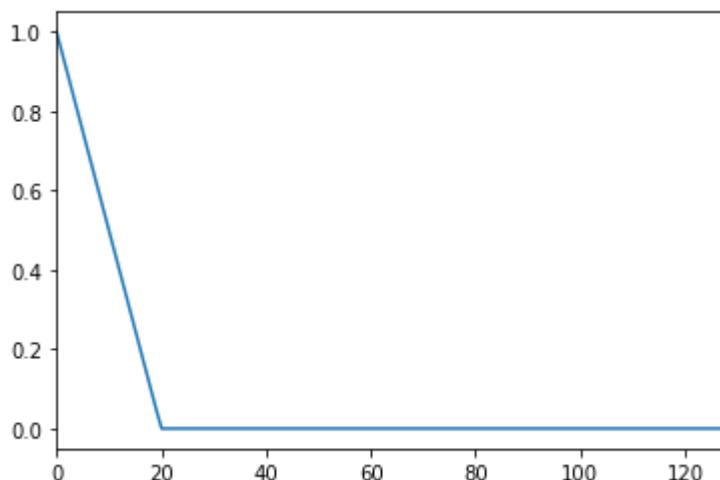
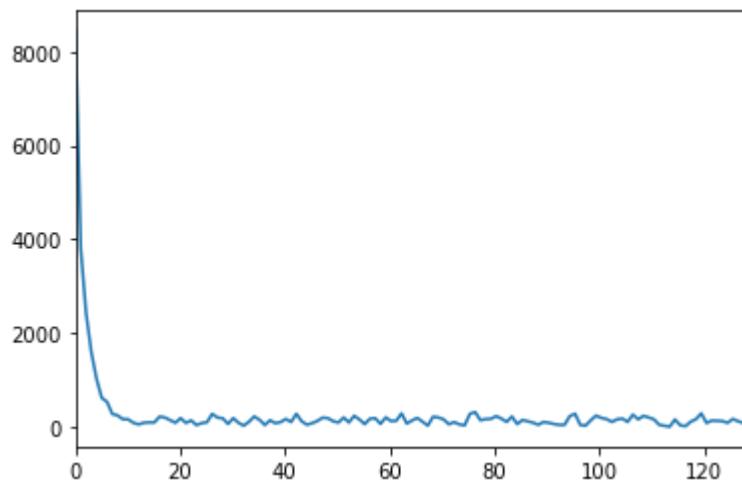
inved_data = ifft(filtered_out)
plt.plot(x,inved_data)
plt.show()

```



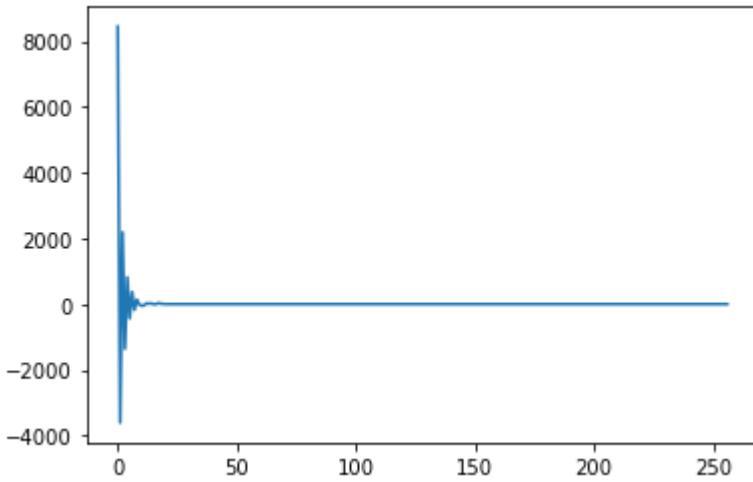
<ipython-input-4-4bc2838164ae>:18: DeprecationWarning: Using scipy.fft as a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.

```
out = fft(ydata)
```



/Users/bob/anaconda3/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part

```
return array(a, dtype, copy=False, order=order)
```

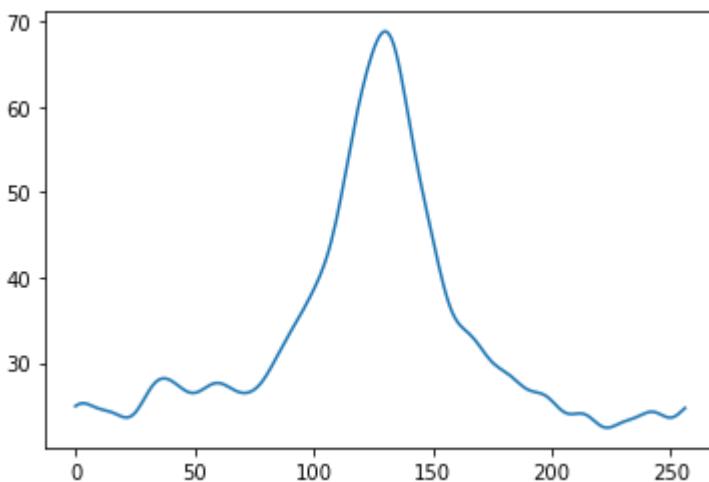


```
<ipython-input-4-4bc2838164ae>:45: DeprecationWarning: scipy.ifft is deprecated and will be removed in SciPy 2.0.0, use scipy.fft.ifft instead
```

```
inved_data = ifft(filtered_out)
```

```
/Users/bob/anaconda3/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
```

```
return array(a, dtype, copy=False, order=order)
```



FFTの動作原理

このように便利なFFTであるが、どのような理屈で導かれるのか？ Fourier変換法は、この課題だけでも何回ものコマ数が必要なほどの中を含んでいる。ここでは、その基本となる考え方（のひとつ）だけを提示する。

1. 関数の内挿で導入した基底関数を直交関数系でとる。ところが、展開係数を逆行列で求める手法では計算が破綻。
2. 直交関係からの積分による係数決定。
3. 選点直交性による計算の簡素化。
4. 高速フーリエ変換アルゴリズムによる高速化。

関数内挿としてのFourier関数系

一連の関数系による関数の内挿は、基底関数を $\varphi_n(x)$ として

$$F(x) = \sum_{n=1}^N a_n \varphi_n(x)$$

で得られることを見た。Fourier変換では基底関数として $\varphi_n(x) = \sin(2\pi nx)$, $\cos(2\pi nx)$ をとる。関数の内挿法で示したように、この x_i での値 $f_i, i = 1 \dots M$ と、近似の次数(N)とでつくる係数行列、

$$A = \begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_N(x_0) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_0(x_M) & \varphi_1(x_M) & \cdots & \varphi_N(x_M) \end{bmatrix}$$

を求めて、係数 a_i とデータ点 f_i をそれぞれベクトルと考えると、

$$\mathbf{A} \cdot \mathbf{a} = \mathbf{f}$$

から、通常の逆行列を求める手法で係数を決定することもできる。しかし、この強引な方法はデータ数、関数の次数が多い、フーリエ変換が対象としようとする問題では破綻する。もっといい方法が必要で、それが直交関数系では存在する。

直交関係からの積分による係数決定

関数の直交関係は、

$$\int_a^b \varphi_n(x) \varphi_m(x) dx = \delta_{mn} C_n = \begin{cases} C_m & \text{at } n = m \\ 0 & \text{at } n \neq m \end{cases}$$

である。定数 C_m は、 \sin, \cos の三角関数系では次の通り。

```
maple
> plot([sin(x), sin(3*x)], x=0..2*Pi);
```



```
maple
> plot([sin(x)*sin(3*x)], x=0..2*Pi, color=black);
```



```
maple
> int(sin(x)*sin(3*x), x=0..2*Pi);
```

0

```
maple
> for i from 1 to 3 do for j from 1 to 3 do
S:=int(sin(i*x)*sin(j*x), x=0..2*Pi);
> print(i,j,S); end do; end do;
```

$1, 1, \pi$
 $1, 2, 0$
 $1, 3, 0$
 $2, 1, 0$
 $2, 2, \pi$
 $2, 3, 0$
 $3, 1, 0$
 $3, 2, 0$
 $3, 3, \pi$

$$\int_a^b F(x) \varphi_m(x) dx$$

を考える。先程の???式をいれると

$$\int_a^b F(x) \varphi_m(x) dx = \int_a^b \sum_{n=1}^N a_n \varphi_n(x) \varphi_m(x) dx = \begin{cases} a_m C_m & \text{at } n = m \\ 0 & \text{at } n \neq m \end{cases}$$

となる。こうして、係数 a_n が

$$a_n = \frac{1}{C_n} \int_a^b F(x) \varphi_n(x) dx$$

で決定できる。

直接積分によるフーリエ係数

対象とする関数をまず作る。

```

maple
> restart;
> #F:=x->piecewise(x=0,1/2,x>0,x);
#F:=x->piecewise(x<1/2,x,x>=1/2,1-x);
#F:=x->piecewise(x<1/2,-1,x>1/2,1);
F:=x->piecewise(x<1/2,-1,x>=1/2,1);
#F:=x->piecewise(x>0 and x<1/2,-1,x>1/2,1);
#F:=x->x-1/2;
plot(F(x),x=0..1);

```

$$F := x \mapsto \text{piecewise}(x < \frac{1}{2}, -1, \frac{1}{2} \leq x, 1)$$



piecewise関数は階段関数で、振る舞いはコメント(#)を適当に外して確認せよ。初期設定。

```

maple
> KK:=3; N:=2^KK; L:=1-0;
> 2*Pi*1/L*x;

```

$$2\pi x$$

```

maple
> int(F(x)*cos(2*Pi*1/L*x),x=0..L);

0

maple
> for n from 0 to N do
    a[n]:=2/L*int(F(x)*cos(2*Pi*n/L*x),x=0..L);
end do;

a0 := 0
a1 := 0
a2 := 0
a3 := 0
a4 := 0
a5 := 0
a6 := 0
a7 := 0
a8 := 0

```

```

maple
> for n from 0 to N do
    b[n]:=2/L*int(F(x)*sin(2*Pi*n/L*x),x=0..L);
end do;

```

$$\begin{aligned}
b_0 &:= 0 \\
b_1 &:= \frac{4}{\pi} \\
b_2 &:= 0 \\
b_3 &:= \frac{4}{3\pi} \\
b_4 &:= 0 \\
b_5 &:= \frac{4}{5\pi} \\
b_6 &:= 0 \\
b_7 &:= \frac{4}{7\pi} \\
b_8 &:= 0
\end{aligned}$$

ここで、オイラーの関係

$$\begin{aligned}
a[n] &= c[n] + c[-n], \quad b[n] = I(c[n] - c[-n]) \\
c[-n] &= \frac{1}{2}(a[n] + b[n]), \quad c[n] = \frac{1}{2}(a[n] - Ib[n])
\end{aligned}$$

を使って、三角関数系からexpへ変換する。

```

maple
> for n from 0 to N do c[n]:=1/L*int(F(x)*exp(-
I*2*Pi*n/L*x),x=0..L); end do;

```

```
> for n from 1 to N do c[-  
n]:=1/L*int(F(x)*exp(I*2*Pi*n/L*x),x=0..L); end do;
```

$$\begin{aligned}c_0 &:= 0 \\c_1 &:= \frac{2I}{\pi} \\c_2 &:= 0 \\c_3 &:= \frac{2I}{3\pi} \\c_4 &:= 0 \\c_5 &:= \frac{2I}{5\pi} \\c_6 &:= 0 \\c_7 &:= \frac{2I}{7\pi} \\c_8 &:= 0 \\c_{-1} &:= -\frac{2I}{\pi} \\c_{-2} &:= 0 \\c_{-3} &:= -\frac{2I}{3\pi} \\c_{-4} &:= 0 \\c_{-5} &:= -\frac{2I}{5\pi} \\c_{-6} &:= 0 \\c_{-7} &:= -\frac{2I}{7\pi} \\c_{-8} &:= 0\end{aligned}$$

```
maple  
> F1:=unapply(sum(evalf(c[i]*exp(I*2*Pi*i/L*x)),i=-(N-1)..(N-  
1)),x):  
> plot({Re(F1(x)),F(x)},x=0..1);
```



```
maple  
> evalf(2/3/Pi);
```

0.2122065907

選点直交性による計算の簡素化

ところが、実際に積分していくには、時間がかかりすぎる。直交関数系の選点直交性を使うとより簡単になる。

直交関数系の選点直交性

直交多項式は,

$$\varphi_n(x) = 0 \text{ at } x_1, x_2, \dots, x_n$$

である. $n - 1$ 以下の次数 m, l では,

$$\sum_{i=1}^n \phi_l(x_i) \varphi_m(x_i) = \delta_{ml} C_l$$

が成り立つ. これは, 直交関係と違い積分でないことに注意. 証明は略. これを使えば, この先程の直交関数展開

$$F(x) = \sum_{l=1}^N a_l \varphi_l(x)$$

の両辺に $\varphi_m(x_i)$ を掛けて i について和をとれば,

$$\begin{aligned} & \sum_{i=1}^n F(x_i) \varphi_m(x_i) = \\ & \sum_{i=1}^n \sum_{l=1}^N a_l \varphi_l(x_i) \varphi_m(x_i) \\ & = \sum_{l=1}^N a_l \sum_{i=1}^n \varphi_l(x_i) \varphi_m(x_i) \\ & = \sum_{l=1}^N a_l \delta_{ml} C_m = a_m C_m \end{aligned}$$

となる. つまり,

$$a_m = \frac{1}{C_m} \sum_{i=1}^n F(x_i) \varphi_m(x_i)$$

となり, 単純な関数の代入とかけ算で係数が決定される.

選点直交性を用いた結果

```
maple
> KK:=4; N:=2^KK; L:=1..0;
> for k from 0 to N-1 do
    c[k]:=evalf(sum(F(i*L/N)*exp(-I*2*Pi*k*i/N), i=0..N-1));
end do;

maple
c_0:=0.
c_1:=-2.000000000 + 10.05467898 I
c_2:=0.
c_3:=-2.000000000 + 2.993211524 I
c_4:=0.
c_5:=-2.000000001 + 1.336357276 I
c_6:=0.
```

```

c_7:=-2.000000001 + 0.3978247331 I
c_8:=0.
c_9:=-2.000000001 - 0.3978247331 I
c_10:=0.
c_11:=-2.000000001 - 1.336357276 I
c_12:=0.
c_13:=-2.000000000 - 2.993211524 I
c_14:=0.
c_15:=-2.000000000 - 10.05467898 I

```

```

maple
> F1:=unapply(sum(evalf(c[i]*exp(I*2*Pi*i/L*x)/N), i=0..(N/2-1))+
> sum(evalf(c[N-i]*exp(-I*2*Pi*i/L*x)/N), i=1..(N/2-1)),x):
> plot({Re(F1(x)),F(x)},x=0..1);

```



高速フーリエ変換アルゴリズムによる高速化

\sin , \cos と \exp 関数を結びつけるオイラーの関係を使うと,

と変換できる。これを使うと,

$$c_k = \frac{1}{C_m} \sum_{i=0}^{N-1} F(x_i) \exp\left(\frac{-2\pi I}{N}\right)$$

となる。 $N = 8$ の場合を実際に計算すると、 $z = \exp(-\frac{2\pi}{8}I)$ として、 $z^8 = 1, z^9 = z, \dots$ を使うと,

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 & z^4 & z^5 & z^6 & z^7 \\ 1 & z^2 & z^4 & z^6 & 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z & z^4 & z^7 & z^2 & z^5 \\ 1 & z^4 & 1 & z^4 & 1 & z^4 & 1 & z^4 \\ 1 & z^5 & z^2 & z^7 & z^4 & z^1 & z^6 & z^3 \\ 1 & z^6 & z^4 & z^2 & 1 & z^6 & z^4 & z^2 \\ 1 & z^7 & z^6 & z^5 & z^4 & z^3 & z^2 & z \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{bmatrix}$$

となる。この行列計算を素直に実行すると、 $8 \times 8 = 64$ 回の演算が必要となる。これを減らせないかと考えたのが、高速フーリエ変換の始まりである。この行列をよく見ると同じ計算を重複しておこなっていることが分かる。そこで、行列の左側と右側で同じ計算をしている部分をまとめると、

$$\begin{aligned}
c_4 &= F_0 + z^4 F_1 + F_2 + z^4 F_3 + F_4 + z^4 F_5 + F_6 + z^4 F_7 \\
&= (F_0 + F_2 + F_4 + F_6) + z^4 (F_1 + F_3 + F_5 + F_7)
\end{aligned}$$

$$\begin{array}{c}
 \text{Left: } \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z & z^2 & z^3 \\ 1 & z^2 & z^4 & z^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^3 & z^6 & z \\ 1 & z^4 & 1 & z^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^5 & z^2 & z^7 \\ 1 & z^6 & z^4 & z^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^7 & z^6 & z^5 \end{bmatrix} \\
 \text{Right: } \begin{bmatrix} F_0 + F_4 \\ F_1 + F_5 \\ F_2 + F_6 \\ F_3 + F_7 \\ F_0 - F_4 \\ F_1 - F_5 \\ F_2 - F_6 \\ F_3 - F_7 \end{bmatrix}
 \end{array}$$

とすることができる。ここで、 $z^4 = -1$ などを使っている。右側のベクトルの計算でロスするが、行列の中の計算の回数を半分に減らすことができる。再度できあがった行列を見れば、同じ計算をさらにまとめることができそうである。こうして、次々と計算回数を減らしていくことが可能で、最終的に行列部分の計算が一切なくなる。残るのは、右側のベクトルの足し算引き算だけになる。

このベクトルの組み合わせは、一見相当複雑そうで、その条件分岐で時間がかかりそうに思われる。しかし、よく調べてみれば、単純なビット演算で処理することが可能であることが判明した。こうして、2の整数乗のデータの組に対しては、極めて高速にフーリエ変換を実行することが可能となった。FFTでの演算回数は、データ数をNとすると

$$N \log_2 N$$

となる。単純な場合の N^2 と比較すると、以下のようにになり、どれだけ高速化されているかが理解されよう。

```

maple
> dN2:=[];
dFft:=[];
for i from 2 to 16 do N:=2^i;
n2:=N*N;
Fft:=N/2*log[2](N);
> Fft/n2;
printf("%10d %12d %12d
%10.5f\n",N,n2,Fft,evalf(Fft/n2));
> dN2:=[op(dN2),[N,n2]];
dFft:=[op(dFft),[N,Fft]];
end do;

```

maple	N	N^2	$N \log_2 N$	高速化比
	4	16	4	0.25000
	8	64	12	0.18750
	16	256	32	0.12500
	32	1024	80	0.07812
	64	4096	192	0.04688
	128	16384	448	0.02734
	256	65536	1024	0.01562
	512	262144	2304	0.00879
	1024	1048576	5120	0.00488
	2048	4194304	11264	0.00269
	4096	16777216	24576	0.00146
	8192	67108864	53248	0.00079
	16384	268435456	114688	0.00043
	32768	1073741824	245760	0.00023
	65536	4294967296	524288	0.00012

maple
> with(plots);
> l1:=plot(dN2); l2:=plot(dFft);
> display(l1,l2);

1千1回
FLOPS

(G)
1D



```
maple
> l1:=logplot(dN2): l2:=logplot(dFft):
> display(l1,l2);
```



```
maple  
> l1:=loglogplot(dN2): l2:=loglogplot(dFft):  
> display(l1,l2);
```



FFT関数を用いた結果



課題と解答例

合成波のFFT

下の例に従って、 $\sin(i/13)$ と $\sin(i/2)$ の合成波を作成し、FFTをかけた後、周波数での強度を表示せよ。
合成波($2\sin(i/2)+\sin(i/13)$)との違いをのべよ。



In []: