FFT(Fast Fourier Transformation)

ー数値計算(10/12/3,10)ー

関西学院大学理工学部 西谷滋人

FFTの応用

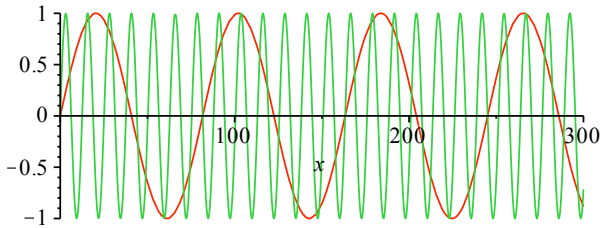
Fast Fourier Transformation(FFT)高速フーリエ変換(あるいはデジタル(離散)フーリエ変換(DFT))は，周波数分解やフィルターを初め，画像処理などの多くの分野で使われている．基本となる考え方は，直交基底による関数の内挿法である．最初にその応用例を見た後，どのような理屈でFFTが動いているかを解説する．

周波数分解

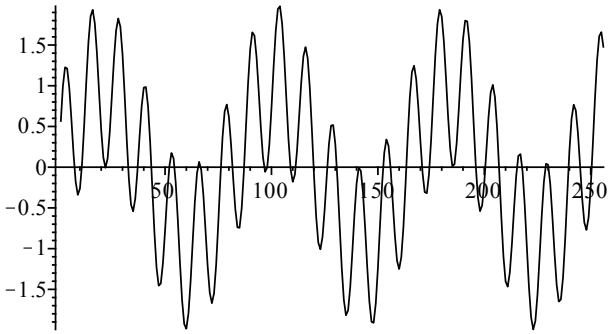
```
> restart:
> data1:=[];
  for i from 1 to 256 do
    data1:=[op(data1),evalf(sin(i/13))+evalf(sin(i/2))];
  end do:
```

$$data1 := [\,]$$

```
> plot([sin(x/13),sin(x/2)],x=0..300);
```



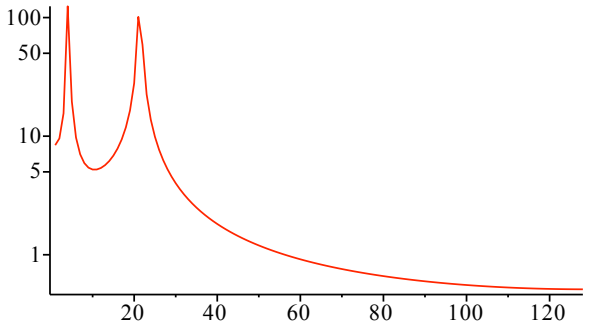

```
> with(plots):
  listplot(data1);
```




```
> X:=array(data1):
  Y:=array(1..256,sparse):
> FFT(8,X,Y);
```

256

```
> Data2:=[seq([i,sqrt(X[i]^2+Y[i]^2)],i=1..128)]:
> logplot(Data2);
```

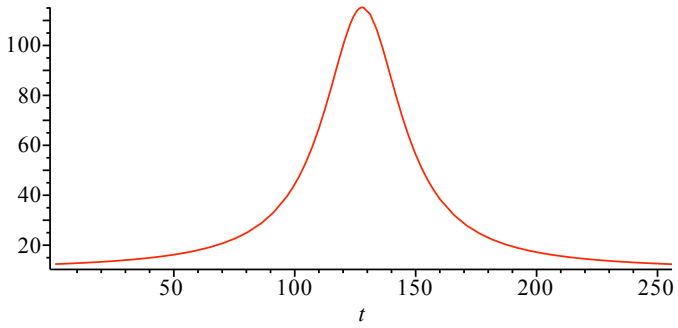


高周波フィルター

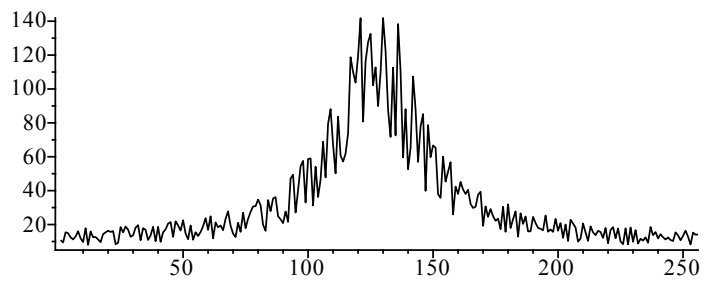
```
> f1:=t->subs(a=10,b=40000,c=380,d=128,a+b/(c+(t-d)^2));
```

$$f1 := t \rightarrow subs\left(a = 10, b = 40000, c = 380, d = 128, a + \frac{b}{c + (t - d)^2}\right)$$

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



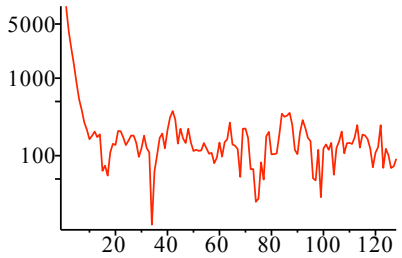

```
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
  #T:=[seq(evalf(rand()/10^12),i=1..256)]:
> with(plots):
> listplot(T);
```



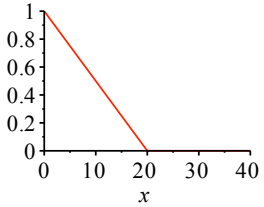

```
> Idata:=array([seq(0,i=1..256)]):
> Rdata:=convert(T,array):
> FFT(8,Rdata,Idata);
```

256

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



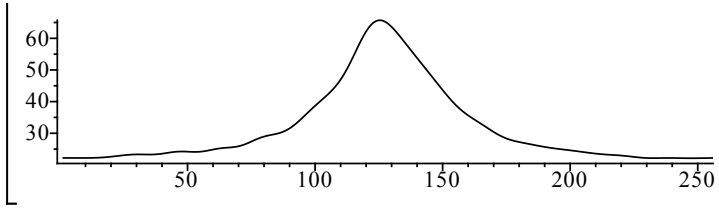

```
> filter:=x->piecewise(x>=0 and x<=20,(1-x/20)):
  #filter:=x->piecewise(x>=0 and x<=20,1);
> plot(filter(x),x=0..40);
```




```
> FRdata:=array([seq(Rdata[i]*filter(i),i=1..256)]):
> FIdata:=array([seq(Idata[i]*filter(i),i=1..256)]):
> iFFT(8,FRdata,FIdata);
```

256

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



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内容を含んでいる．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$\phi_n(x)$として

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

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$f_i, i=1\cdots M$と，近似の次数$(N)$とでつくる，

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

とした係数行列を求めて，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，

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

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．

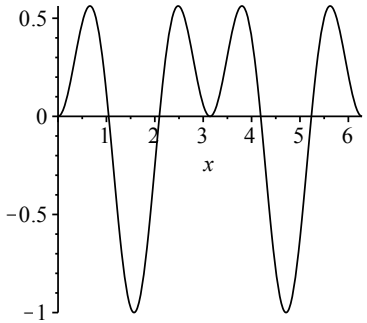
直交関係からの積分による係数決定

関数の直交関係は，

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

である．C_mは

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




```
> for i from 1 to 5 do
  for j from 1 to 5 do
  S:=int(sin(i*x)*sin(j*x),x=0..2*Pi);
  print(i,j,S);
  end do;
  end do;
```

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

を考える．先程の(1)式をいれると

$$\int_a^b F(x)\varphi_m(x)\,dx = \int_a^b \sum_{n=1}^{N} a_n\varphi_n(x)\varphi_m(x)\,dx = = \begin{bmatrix} {}_mC_m \text{ at } m = n \\ 0 \text{ at } m \neq n \end{bmatrix}$$

となる．こうして，係数a_nが

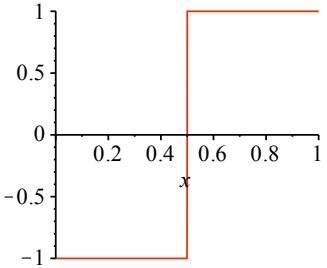
$$a_n = \frac{1}{C_n}\int_a^b F(x)\varphi_n(x)\,dx$$

で決定できる．

対象とする関数

```
> 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);
```

$$F := x \to piecewise\left(x < \frac{1}{2}, -1, \frac{1}{2} \le x, 1\right)$$




```
> KK:=2;
  N:=2^KK;L:=1-0;
```

$$KK := 2$$

$$N := 4$$

$$L := 1$$

直接積分によるフーリエ級数

```
> 2*Pi*1/L*x;
```

$$2\,\pi\,x \qquad (3.2.2.1)$$

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

$$0 \qquad (3.2.2.2)$$

```
> 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;
```

$$a_0 := 0$$

$$a_1 := 0$$

$$a_2 := 0$$

$$a_3 := 0$$

$$a_4 := 0$$

```
> 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;
```

$$b_0 := 0$$

$$b_1 := -\frac{4}{\pi}$$

$$b_2 := 0$$

$$b_3 := -\frac{4}{3\,\pi}$$

$$b_4 := 0$$

```
> 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;
```

Note that:
a[n]=c[n]+c[-n], b[n]=I (c[n]-c[-n])
c[-n]= 1/2 (a[n] + b[n]), c[n]=1/2 (a[n] - I b[n])

$$c_0 := 0$$

$$c_1 := \frac{2\,I}{\pi}$$

$$c_2 := 0$$

$$c_3 := \frac{\frac{2}{3}\,I}{\pi}$$

$$c_4 := 0$$

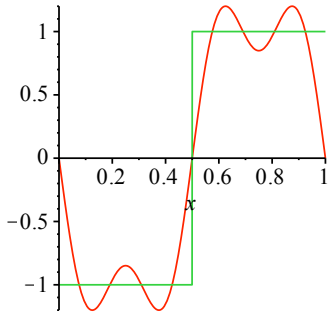
$$c_{-1} := -\frac{2\,I}{\pi}$$

$$c_{-2} := 0$$

$$c_{-3} := -\frac{\frac{2}{3}\,I}{\pi}$$

$$c_{-4} := 0$$

```
> 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);
```




```
>
```

選点直交性による計算の簡素化

ところが，実際に積分していては，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，

$$\varphi_n(x) = 0 \; at \; x = x_1, x_2, \cdots x_n$$

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，

$$\sum_{i=1}^{n} F(x_i)\phi_m(x_i) = \sum_{i=1}^{n}\sum_{l=1}^{N} a_l \varphi_l(x_i)\varphi_m(x_i)$$

$$= \sum_{l=1}^{N} a_l \sum_{i=1}^{n} \varphi_l(x_i)\varphi_m(x_i)$$

$$= \sum_{l=1}^{N} a_l \delta_{ml} C_m = a_m C_m$$

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

```
> KK:=3;
N:=2^KK;L:=1-0;
```

$$KK := 3$$

$$N := 8$$

$$L := 1$$ (3.3.1.1)

```
> 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;
```

$$c_0 := 0.$$

$$c_1 := -2. + 4.828427124\,I$$

$$c_2 := 0.$$

$$c_3 := -2. + 0.828427124\,I$$

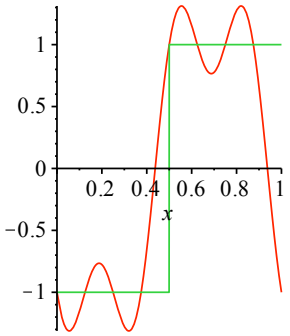
$$c_4 := 0.$$

$$c_5 := -2. - 0.828427124\,I$$

$$c_6 := 0.$$

$$c_7 := -2. - 4.828427124\,I$$

```
> 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，

$$\exp\left(\frac{2\pi}{N}\mathrm{I}\right) = \cos\left(\frac{2\pi}{N}\right) + \mathrm{I}\sin\left(\frac{2\pi}{N}\right)$$

と変換できる．これを使うと，

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

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

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

となる．この行列計算を素直に実行すると，8×8=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，

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

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

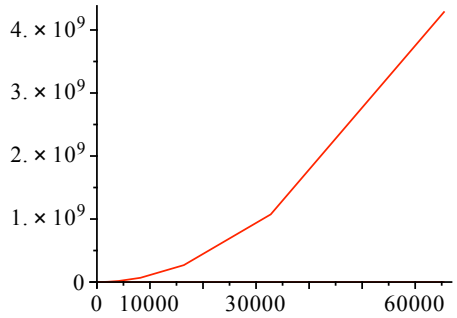
$$N\log_2 N$$

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

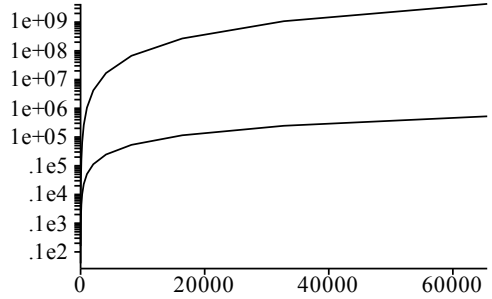
```
> dN2:=[];
  dFft:=[];
  for i from 2 to 16 do
  N:=2^i;
  n2:=N*N;
  Fft:=N/2*log[2](N);
  Fft/n2;
  printf("%10d %12d %12d %10.5f\n",N,n2,Fft,evalf(Fft/n2));
  dN2:=[op(dN2),[N,n2]];
  dFft:=[op(dFft),[N,Fft]];
  end do:
```

```
                        dN2 := [ ]

                        dFft := [ ]

         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
```

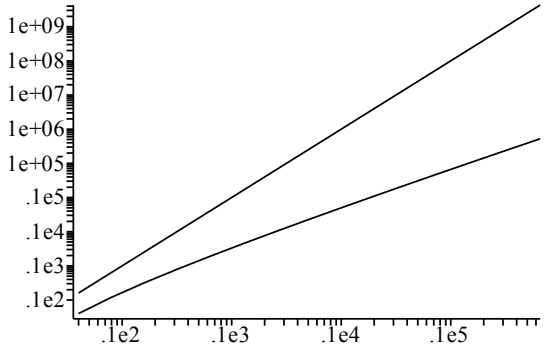
```
> with(plots):
Warning, the name changecoords has been redefined
> l1:=plot(dN2):
  l2:=plot(dFft):
> display(l1,l2);
```




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




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



FFT関数を用いた結果

```
> KK:=3;
  N:=2^KK;i:='i';
```

```
  x1:=array([evalf(seq(F(i/N),i=0..N-1))]);
  y1:=array([evalf(seq(0,i=0..N-1))]);
```

$$KK := 3$$

$$N := 8$$

$$i := i$$

$$x1 := [\,-1.\ -1.\ -1.\ -1.\ 1.\ 1.\ 1.\ 1.\,]$$

$$y1 := [\,0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\,]$$

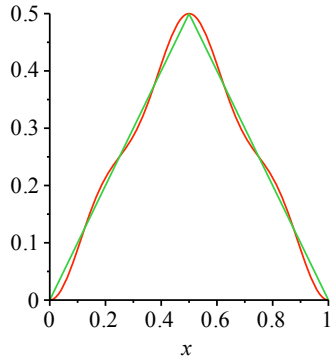
```
> FFT(KK,x1,y1);
```

8

```
> print(x1);print(y1);
```

```
[0., -2.000000001, 0., -1.999999999, 0., -1.999999999,
  0., -2.000000001]
[0., 4.828427122, 0., 0.828427124, 0., -0.828427124,
  0., -4.828427122]
```

```
> F2:=unapply(sum(evalf((x1[i]+I*y1[i])*exp(I*2*Pi*(i-1)
  /L*x)/N),i=1..N/2)+
  sum(evalf((x1[N-i+2]+I*y1[N-i+2])*exp(-I*2*Pi*(i-1)/L*
  x)/N),i=2..N/2),x):
> plot({Re(F2(x)),F(x)},x=0..1);
```




```
>
```