Examples of discrete Fourier transforms (DFT)

Let's compute DFT of following discrete functions

a) constant $f_j = 1, \quad j = 1, \dots, N$

b) "basic frequency"

$$f_j = 2 \cdot \cos\left(2\pi \frac{j-1}{N-1}\right), \quad j = 1, \dots, N$$

c) 1,5 multiple of "basic frequency"

$$f_j = 2 \cdot \sin\left(1{,}5 \cdot 2\pi \frac{j-1}{N-1}\right), \quad j = 1, \dots, N$$

d) "highest" possible frequency

$$f_j = 1 + (-1)^j, \quad j = 1, \dots, N$$

We use Maple to compute DFT of the above originals. Note that used FFT (fast Fourier transform) algorithm gives

182

the same results as DFT, but it is "faster". FFT needs $N = 2^n$ samples.

Let's choose $n=6 \rightarrow N = 64$.

183 a)

```
> restart:
> readlib(FFT):
> n:=6;
```

$$n := 6$$

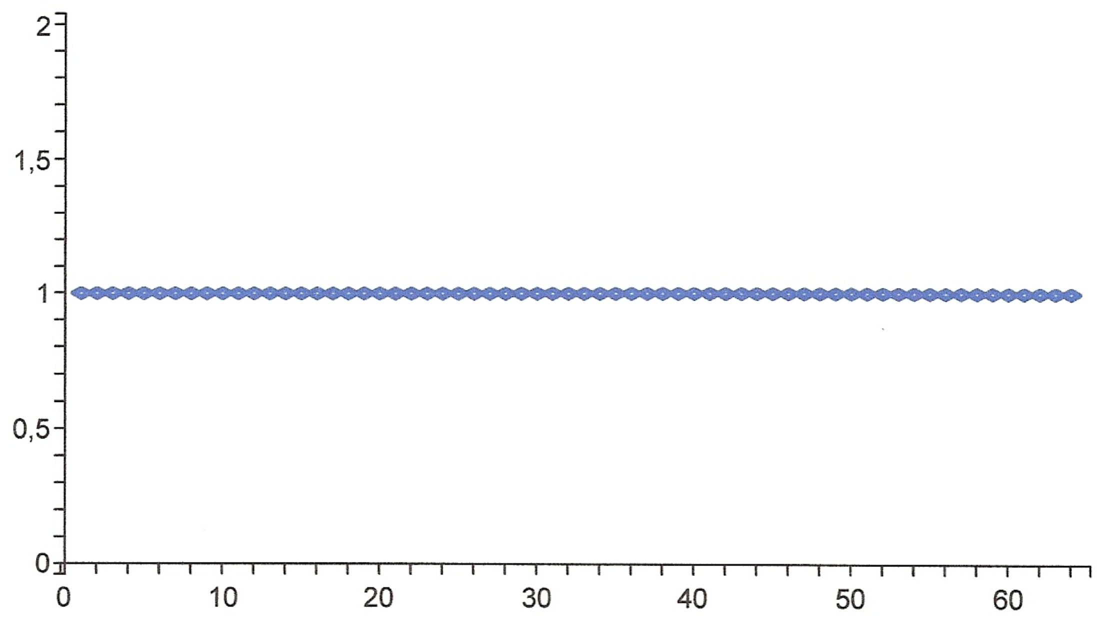
```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=1:
```

← real part of f_j

```
> Y[r]:=0:
```

← imaginary part of f_j

```
> od:
> with(plots):
Warning, the name changecoords has been redefined
> listplot(X,style=point,symbolsize=10,thickness=2,color=blue);
```




```
> FFT(n,X,Y);
```

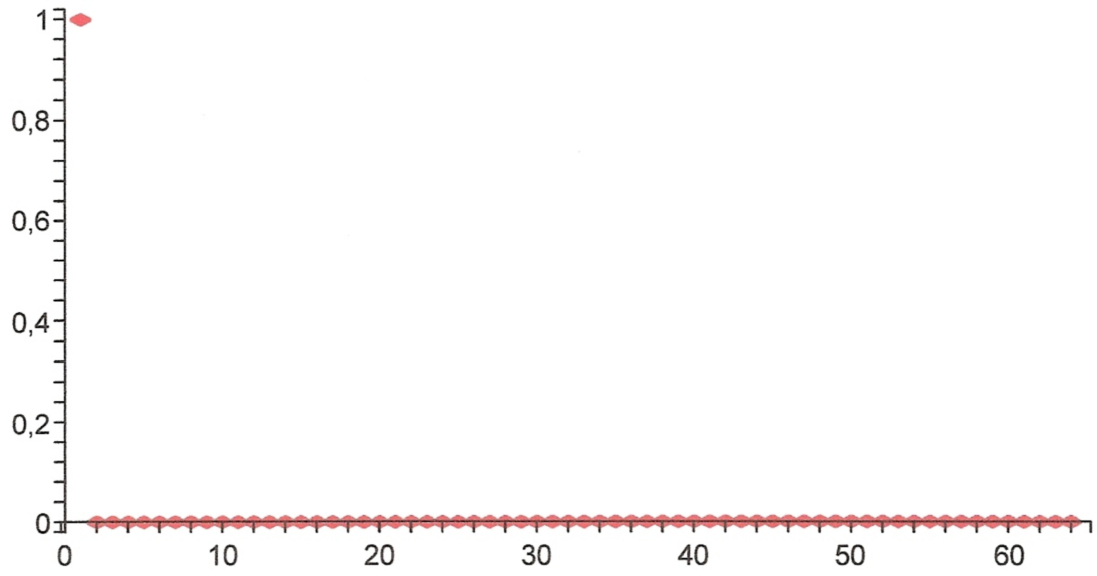
64

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
```

↑ imaginary part of DFT coefficient

↑ real part of DFT coefficient

184




```
> evalf(A[1]);evalf(A[2]);evalf(A[3]);evalf(A[4]);
```

1.

0.

0.

0.

this corresponds to amplitude of 0th frequency, i.e. the mean value of f_j

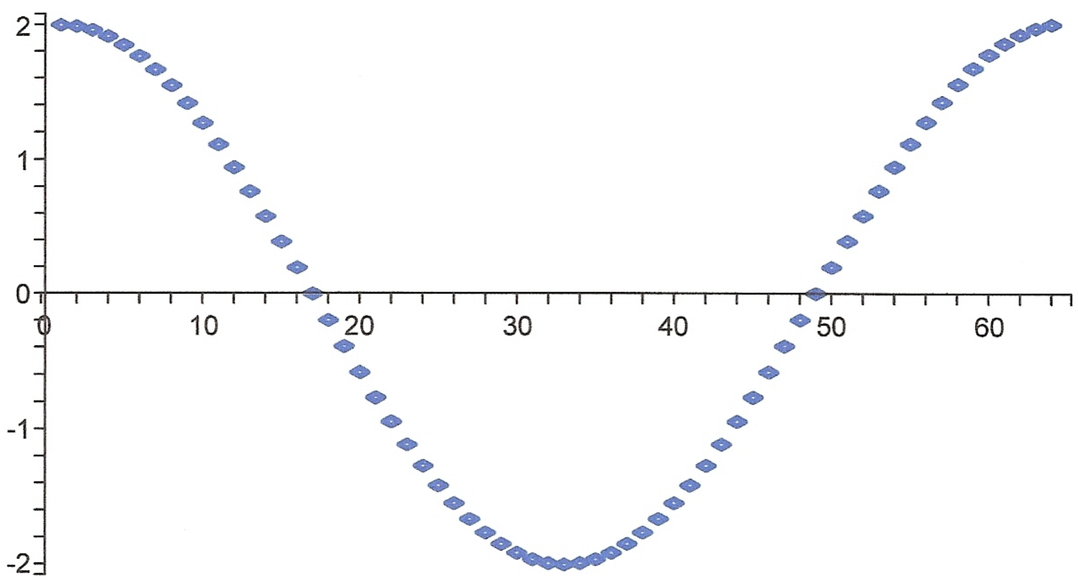
```
>
```

185 b)

```
> restart:
> readlib(FFT):
> n:=6;
```

$$n := 6$$

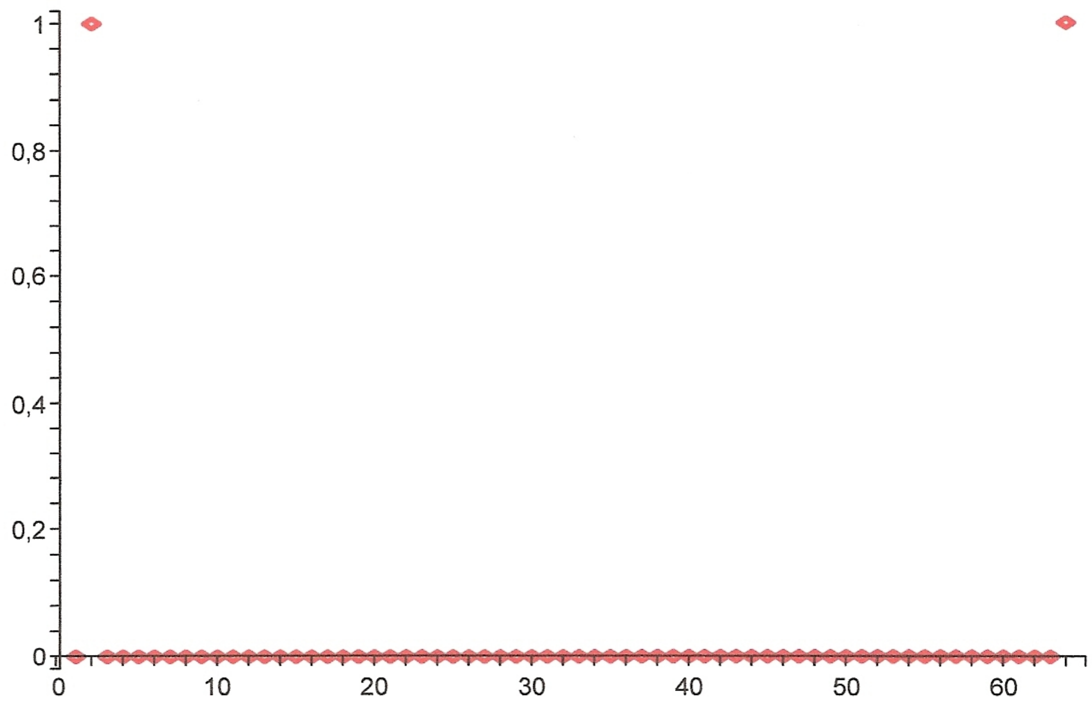
```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=2*cos(2*Pi*(r-1)/(2^n)):
> Y[r]:=0:
> od:
> with(plots):
Warning, the name changecoords has been redefined
> listplot(X,style=point,symbolsize=10,thickness=2,color=blue);
```




```
> FFT(n,X,Y);
```

$$64$$

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
```




```
> evalf(A[1]);evalf(A[2]);evalf(A[3]);evalf(A[4]);evalf(A[63]);evalf(A[6
);
```

$$0.$$
$$1.000000001$$
$$0.$$
$$5.460621516\ 10^{-10}$$
$$0.$$
$$1.000000000$$

$\frac{1}{2}$ of amplitude for basic frequency

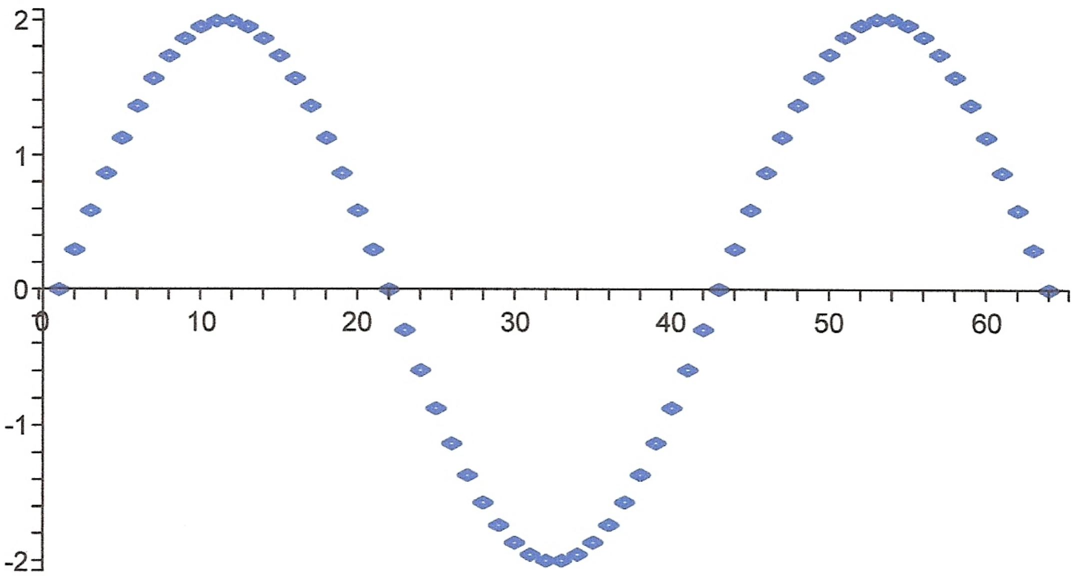
```
>
```

187 c)

```
> restart:
> readlib(FFT):
> n:=6;
```

$$n := 6$$

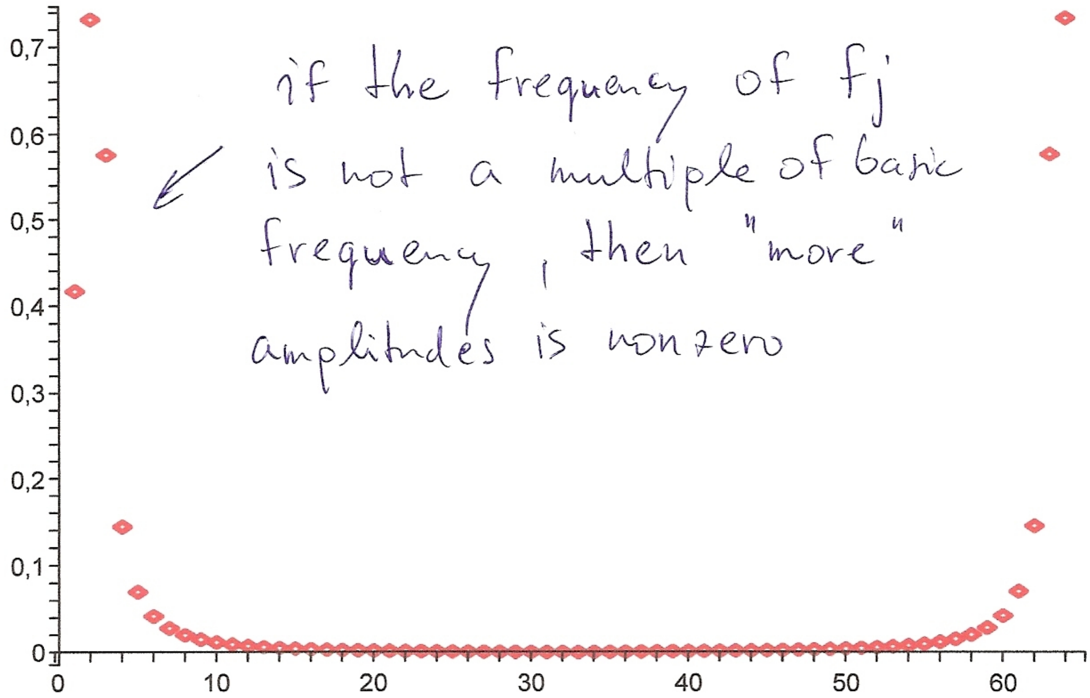
```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=2*sin(3*Pi*(r-1)/(2^n-1)):
> Y[r]:=0:
> od:
> with(plots):
Warning, the name changecoords has been redefined
> listplot(X,style=point,symbolsize=10,thickness=2,color=blue);
```




```
> FFT(n,X,Y);
```

$$64$$

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
```



if the frequency of f_j is not a multiple of basic frequency, then "more" amplitudes is nonzero

```
> evalf(A[1]);evalf(A[2]);evalf(A[3]);evalf(A[4]);evalf(A[63]);evalf(A[6
);
```

0.4170022699

0.7321425025

0.5761129758

0.1444681338

0.5761129752

0.7321425025

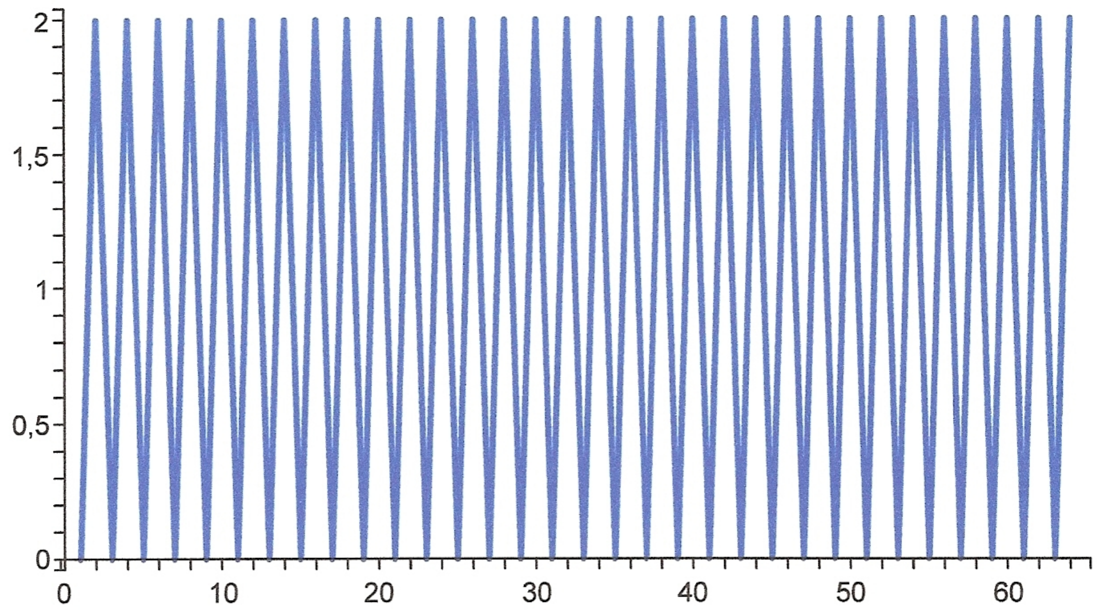
```
>
```

189 d)

```
> restart:
> readlib(FFT):
> n:=6;
```

$$n := 6$$

```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=(-1)^r+1:
> Y[r]:=0:
> od:
> with(plots):
Warning, the name changecoords has been redefined
> listplot(X,style=line,thickness=2,color=blue);
```

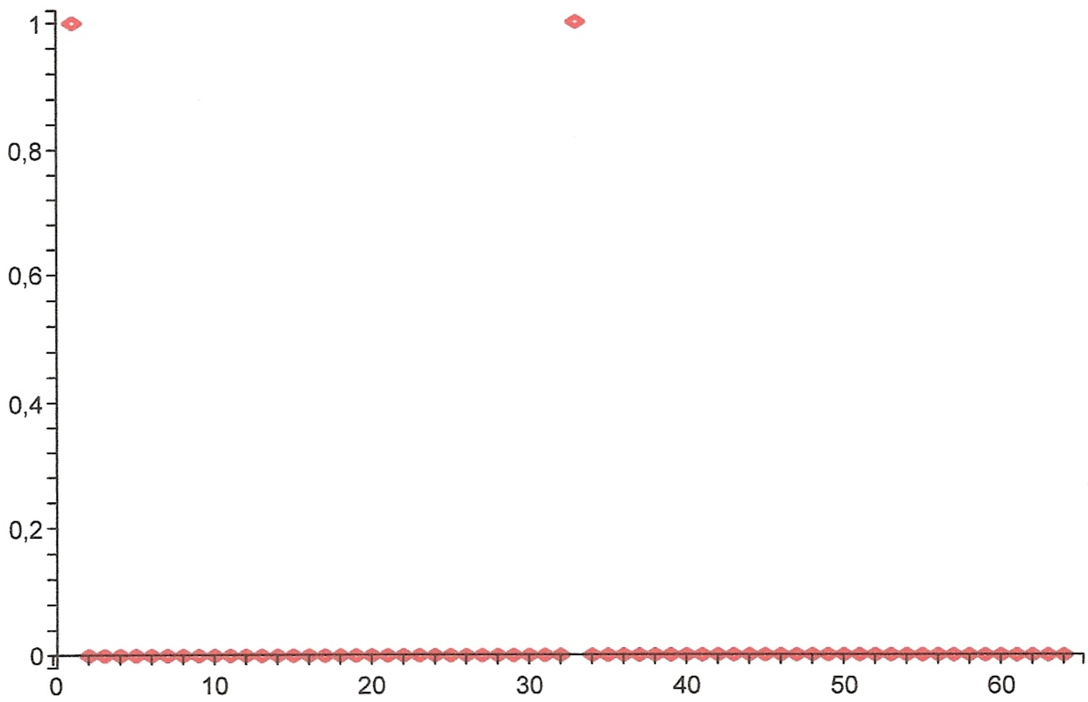



```
> FFT(n,X,Y);
```

$$64$$

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
```

(190)




```
> evalf(A[1]);evalf(A[32]);evalf(A[33]);evalf(A[34]);
```

1. ← mean value (A[1])

0.

1. ← amplitude (A[33])

0.

> Note that mean value is A[1] and real amplitude of f_j is $2 \cdot A[r]$ for $r = 1, ..., 32$ and $1 \cdot A[33]$ for $r = 33$.

Discrete convolution

Consider we have two vectors (real)

$X = (x_0, x_1, x_2, \dots, x_{N-1})$ and

$Y = (y_0, y_1, y_2, \dots, y_{N-1})$

Discrete convolution $X * Y$ is vector with n-th component defined as

$$(X * Y)_n = \sum_{j=0}^{N-1} (x_{n-j} \cdot y_j), \quad n = 0, 1, \dots, N-1$$

Note if we run out of $0, 1, \dots, N-1$ range with coefficient $(n-j)$, then according to periodicity $x_{n-j} = x_{n-j+p \cdot N}$

(or $x_j = x_{j+p \cdot N}$), e.g. $x_{-2} = x_{N-2}$.

Remark: Discrete convolution is commutative

i.e. $(X * Y)_n = (Y * X)_n = \sum_{j=0}^{N-1} (y_{n-j} \cdot x_j)$

Theorem (DFT of convolution):

$$\mathcal{F}_D\{X * Y\} = N \cdot \mathcal{F}_D\{X\} \otimes \mathcal{F}_D\{Y\} \quad (1)$$

where
$$\mathcal{F}_D\{X\} = C = (c_0, c_1, \dots, c_{N-1})$$
$$\mathcal{F}_D\{Y\} = D = (d_0, d_1, \dots, d_{N-1})$$

and
$$C \otimes D = (c_0 \cdot d_0, c_1 \cdot d_1, \dots, c_{N-1} \cdot d_{N-1})$$

Proof: Formula (1) can be written as

$$\mathcal{F}_D^{-1}\{N \cdot C \otimes D\} = X * Y$$

we know that $\mathcal{F}_D\{X\} = C$, i.e.

$$c_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i}{N} kn}$$

193

$$\left(\mathcal{F}_D^{-1}\{N\, C \otimes D\}\right)_n = N \sum_{k=0}^{N-1} c_k\, d_k\, e^{\frac{2\pi i}{N}kn} =$$

$$= \cancel{N} \sum_{k=0}^{N-1} \left(\frac{1}{\cancel{N}} \sum_{l=0}^{N-1} x_l\, e^{-\frac{2\pi i}{N}kl}\right)\cdot\left(\frac{1}{N}\sum_{m=0}^{N-1} y_m\, e^{-\frac{2\pi i}{N}km}\right)\cdot$$

$$\cdot\, e^{\frac{2\pi i}{N}kn} = \sum_{l=0}^{N-1} x_l \left[\sum_{m=0}^{N-1} y_m \underbrace{\left(\frac{1}{N} \sum_{k=0}^{N-1} e^{\frac{2\pi i}{N}k(n-l-m)}\right)}_{\text{function of } (n-l-m)}\right] =$$

let's denote $n-l-m=s$

$$= \text{II}$$

note that $\displaystyle \frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi i}{N}k\cdot s} =$

$$= \frac{1}{N}\sum_{k=0}^{N-1}\left(e^{\frac{2\pi i s}{N}}\right)^k = \frac{1}{N}\,\frac{\left(e^{\frac{2\pi i s}{N}}\right)^N - 1}{e^{\frac{2\pi i s}{N}} - 1} =$$

$= 0$ for $s \neq p\cdot N$ since

$\left(e^{\frac{2\pi i s}{N}}\right)^N = 1$ for $s \neq p\cdot N$ and $e^{\frac{2\pi i s}{N}} \neq 1$

for $s \neq p\cdot N$

(194)

if $s = p \cdot N$, then

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi i}{N} ks} = \frac{1}{N}\sum_{k=0}^{N-1} \underbrace{e^{2\pi i k p}}_{=1} = 1$$

consider $s = n-l-m = 0 \Rightarrow m = n-l$

$$\Rightarrow \text{II} = \sum_{l=0}^{N-1} x_l \sum_{m=0}^{N-1} y_m \cdot \delta_{m,n-l} =$$

$$= \sum_{l=0}^{N-1} x_l y_{n-l} = (X * Y)_n \qquad \square$$

where $\delta_{m,n-l} = \begin{cases} 1 & \text{for } m = n-l \\ 0 & \text{for } m \neq n-l \end{cases}$

<u>Remark:</u> We can finish the proof also like $s = n-l-m = 0 \Rightarrow l = n-m$

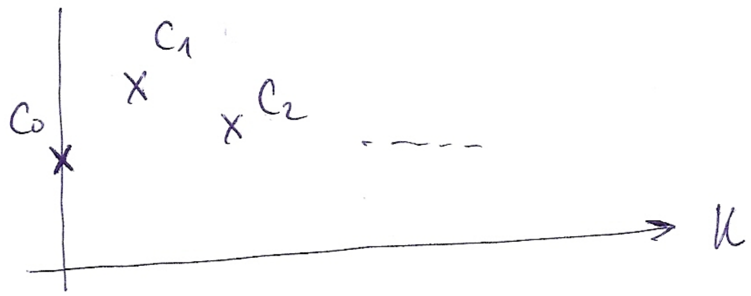
$$\text{II} = \dots = \sum_{m=0}^{N-1} x_{n-m} y_m .$$

Application of discrete convolution

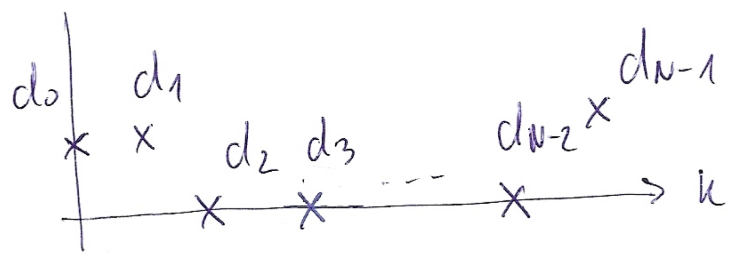
→ filtering of discrete signal

Consider we have the discrete signal $X = (x_0, x_1, \dots, x_{N-1})$, then

$$C = \mathcal{F}_D\{X\} = (c_0, c_1, \dots, c_{N-1})$$



we apply filter D, e.g.



DFT of filtered signal is $C \otimes D$

$$\Rightarrow \mathcal{F}_D^{-1}\{C \otimes D\} = \frac{1}{N} X * Y,$$

where $Y = \mathcal{F}_D^{-1}\{D\}$

196

Example:

We have signal $\sin\left(10\pi \frac{r-1}{2^n-1}\right)$, $r = 1, \dots, 2^8$ with some random noise of "higher" frequency. We perform DFT (FFT) and take only first 8 harmonics (i.e. we set the amplitudes for higher frequencies to zero). Such filtered signal is then transformed back, see the Maple file:

197

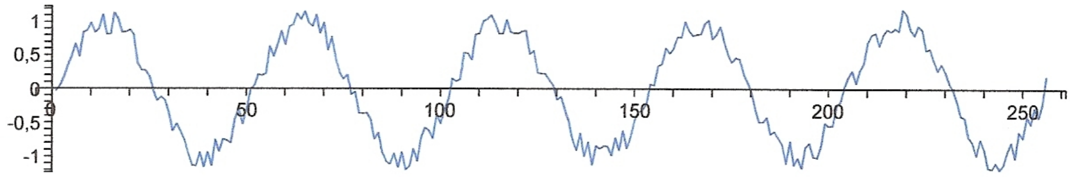
```
> restart:
> readlib(FFT):
> n:=8;
```

$$n := 8$$

```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=sin(10*Pi*(r-1)/(2^n-1))+0.4*(rand()/10^12-0.5):
> Y[r]:=0:
> od:
```

random noise

```
> with(plots):
Warning, the name changecoords has been redefined
> p1:=listplot(X,style=line,symbolsize=10,thickness=1,color=blue):displa
  p1);
```

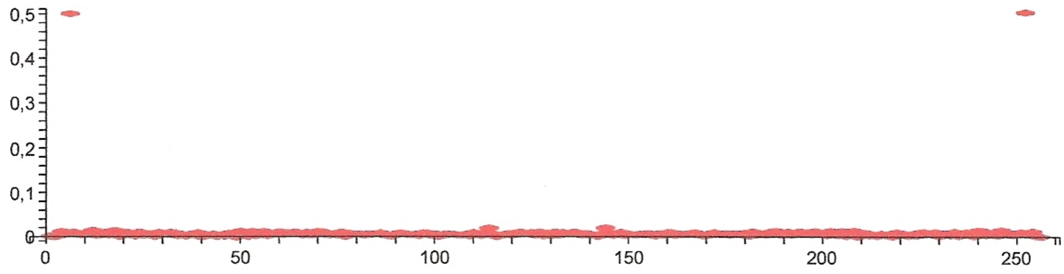



```
> FFT(n,X,Y);
```

$$256$$

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
> listplot(A,style=point,symbolsize=10,thickness=2,color=red);
```

198




```
> for r from 8 to 2^n-8 do
> X[r]:=0:
> Y[r]:=0:
> od:
```

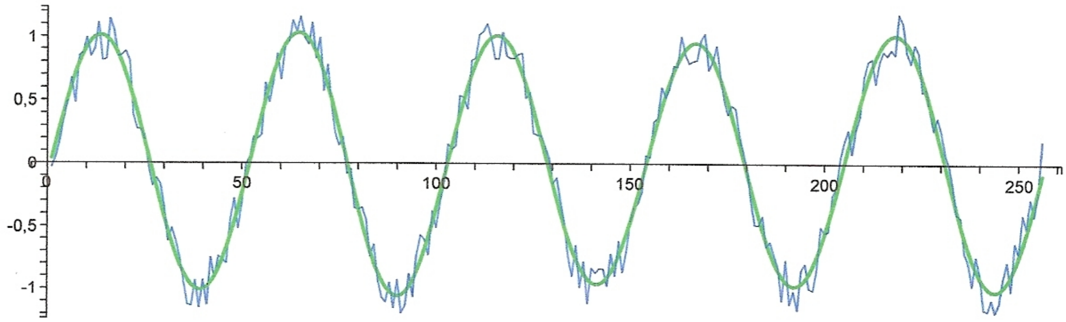
filtering

```
> iFFT(n,X,Y);
```

inverse FFT

256

```
> p2:=listplot(X,style=line,symbolsize=10,thickness=2,color=green):
> display(p1,p2);
```



comparison of original and filtered signal

```
>
```

<u>Remark:</u> Previous example showed filtering in the frequency domain. We can perform filtering also in time (original) domain using discrete convolution.

<u>Example:</u> We have already seen, that the second derivative models the dissipation (note the heat equation) and has tendency to quickly damp "high" frequencies (noise), i.e. the continuous noisy signal $f(t)$ can be filtered like

$$f_{filtered}(t) = f(t) + \varepsilon \cdot f''(t),$$

where ε is some appropriate positive coefficient.

In discrete case

$$f_j^{filtered} = f_j + \varepsilon \, \frac{f_{j-1} - 2 f_j + f_{j+1}}{\Delta^2} =$$

$$= \alpha f_{j-1} + (1-2\alpha) f_j + \alpha f_{j+1}, \quad \alpha > 0$$

200

The later expression can be written using discrete convolution

$$X = (f_0, f_1, f_2, \dots, f_{N-1})$$

$$Y = (1-2\alpha, \alpha, 0, 0, \dots, 0, 0, \alpha)$$

$$f_n^{\text{filtered}} = (X * Y)_n = \sum_{j=0}^{N-1} X_j \, Y_{n-j}$$

see the Maple file with example of $\mathcal{F}_D\{Y\}$ for $\alpha = 0{,}2$

(201)

```
> restart:
> readlib(FFT):
> n:=8;
```

$$n := 8$$

```
> X:=array(1..2^n):
> Y:=array(1..2^n):
> for r from 1 to 2^n do
> X[r]:=0:
> Y[r]:=0:
> od:
> X[1]:=0.6;X[2]:=0.2;X[2^n]:=0.2;
```

$$X_1 := 0.6$$

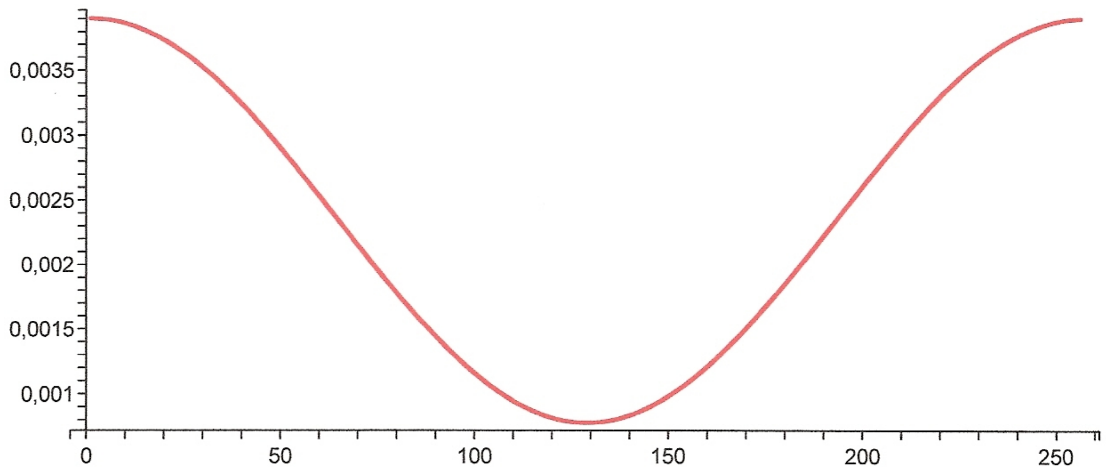
$$X_2 := 0.2$$

$$X_{256} := 0.2$$

```
> FFT(n,X,Y);
```

$$256$$

```
> A:=array(1..2^n):
> for r from 1 to 2^n do
> A[r]:=sqrt(X[r]^2+Y[r]^2)/2^n:
> od:
> with(plots):listplot(A,style=line,symbolsize=10,thickness=2,color=red)
Warning, the name changecoords has been redefined
```



202

Next Maple file shows already used example of signal $\sin(10\pi \frac{r}{2^n})$, $r = 1, \dots, 2^n$ with random noise, filtered using formula

$$f_j^{filtered} = \lambda f_{j-1} + (1-2\lambda) f_j + \lambda f_{j+1}$$

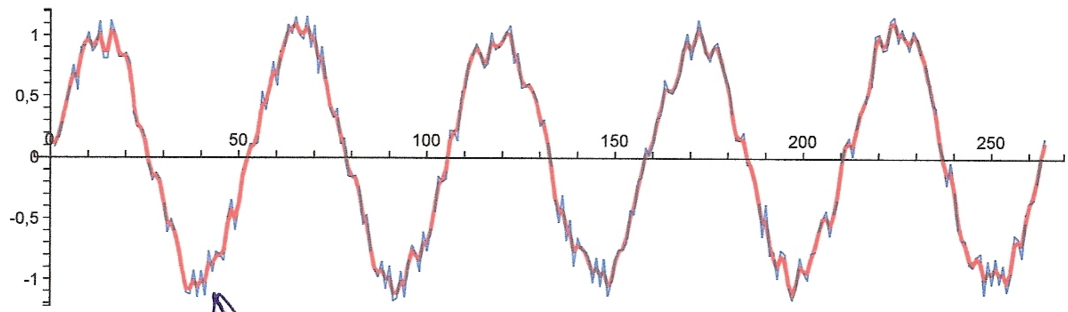
with $\lambda = 0{,}2$

203

```
> restart:
> n:=264;
```

$n := 264$

```
> X:=array(1..n):Y:=array(1..n):
> for r from 1 to n do
> X[r]:=sin(10*Pi*r/n)+0.4*(rand()/10^12-0.5):
> od:
> alpha:=0.2:
  for r from 2 to n-1 do
> Y[r]:=alpha*X[r-1]+(1-2*alpha)*X[r]+alpha*X[r+1]:
> od:
> Y[1]:=alpha*X[n]+(1-2*alpha)*X[1]+alpha*X[2]:
> Y[n]:=alpha*X[n-1]+(1-2*alpha)*X[n]+alpha*X[1]:
> with(plots):
Warning, the name changecoords has been redefined
> p1:=listplot(X,style=line,thickness=1,color=blue):
> p2:=listplot(Y,style=line,thickness=2,color=red):
> display(p1,p2);
```



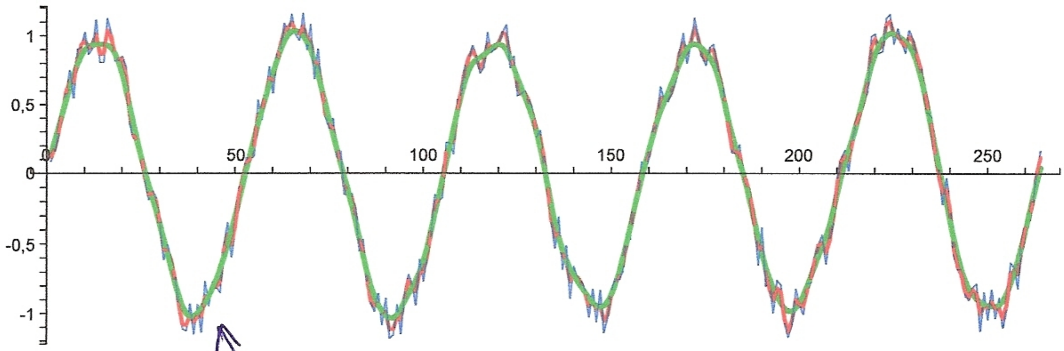
red line denotes filtering (smoothing) after one step, it is not enough

```
> for j from 1 to 10 do
    for r from 1 to n do
>     X[r]:=Y[r]:
>   od:
```

← 10 additional cycles of filtering

204

```
>   od:
>   Y[1]:=alpha*X[n]+(1-2*alpha)*X[1]+alpha*X[2]:
>   Y[n]:=alpha*X[n-1]+(1-2*alpha)*X[n]+alpha*X[1]:
> od:
>
> p3:=listplot(Y,style=line,thickness=3,color=green):
> display(p1,p2,p3);
```



green line denotes signal after 11 cycles of filtering (smoothing)

```
>
```