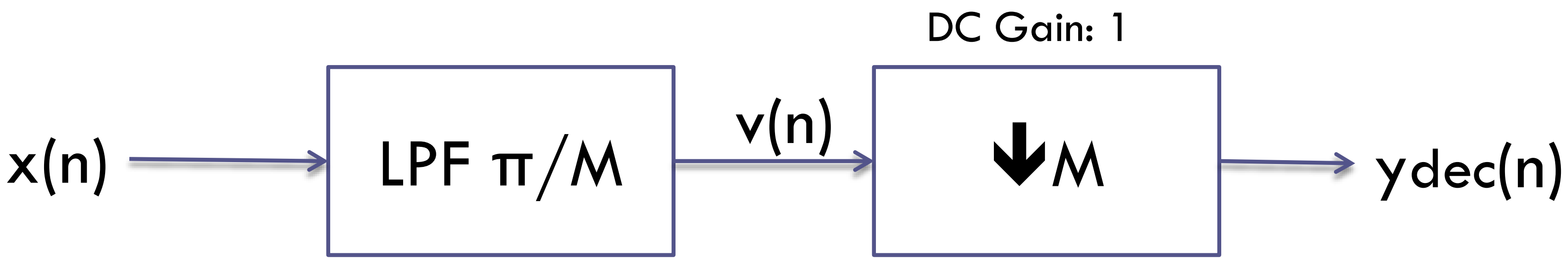
POLYPHASE FILTERS

Michele Buccoli

Polyphase decimation

- In order to decimate a signal, we lowpass it and then downsample it
  - We filter the whole signal, but we are going to use only a part of it (one every M samples)



- We design polyphase filters in order to save computations in the decimation phase

Polyphase decimation

- A filter is decomposed into its polyphase components
- The signal is decomposed into subsequences
- The subsequences are filtered with the correspondent subfilters
- The output is reconstructed as the sum of the subresults

Polyphase decimation

- Suppose we have x(n)=[x(0), x(1), …., x(N-1)]
- We want to decimate with M=3
- We have a 1/M lowpass filter
  - h(n)=[♣, ♦, ♥, ♠, ◊, ∇] with 6 weights
  - Both h(n) and x(n) start from n=0
- Let's compute the convolution and then the downsampling
- First step: folding h(n)→h(-n)
  - h(-n)=[∇ , ◊, ♠, ♥, ♦, ♣]

Polyphase decimation

x(n)= x(0), x(1), x(2), x(3), …., x(N)

h(-n)=∇ , ◊, ♠, ♥, ♦, ♣

y(0)= ♣ x(0)

- Next sample: y(1)
  - We know we are going to neglect y(1) and y(2) (M=3)
  - Let's compute directly y(3)

Polyphase decimation

x(n)= x(0), x(1), x(2), x(3), …., x(N)

h(-n)= ∇ , ◊, ♠, ♥, ♦, ♣

y(3)= ♠ x(0)+ ♥x(1)+♦x(2)+ ♣ x(3)

- Next sample: ~~y(4)~~ y(6)

Polyphase decimation

x(n)= x(0), x(1), x(2), x(3), x(4), x(5), x(6), …., x(N)

h(-n)= ∇ , ◊, ♠, ♥, ♦, ♣

y(6)= ∇ x(1)+ ◊ x(2)+ ♠x(3) +♥x(4)+♦x(5)+ ♣x(6)

- And so on…
- Let's summarize…

Polyphase decimation

$y(0) = \clubsuit x(0)$

$y(3) = \clubsuit x(3) + \blacklozenge x(2) + \heartsuit x(1) + \spadesuit x(0)$

$y(6) = \clubsuit x(6) + \blacklozenge x(5) + \heartsuit x(4) + \spadesuit x(3) + \Diamond x(2) + \nabla x(1)$

$y(9) = \clubsuit x(9) + \blacklozenge x(8) + \heartsuit x(7) + \spadesuit x(6) + \Diamond x(5) + \nabla x(4)$

Polyphase decimation

y(0)= ♣x(0)

y(3)= ♣x(3) +♦x(2)+♥x(1)+♠x(0)

y(6)= ♣x(6) +♦x(5)+♥x(4)+♠x(3) +◊x(2)+∇x(1)

y(9)= ♣x(9) +♦x(8)+♥x(7)+♠x(6) +◊x(5)+∇x(4)

- h(n)=[♣, ♦, ♥, ♠, ◊, ∇]
- e_0 =[♣, ♠] → [x(0),x(3),…, x(kM+0)]

Polyphase decimation

y(0)= ♣x(0) + …?

y(3)= ♣x(3) +♦x(2)+♥x(1)+♠x(0)

y(6)= ♣x(6) +♦x(5)+♥x(4)+♠x(3) +◊x(2)+∇x(1)

y(9)= ♣x(9) +♦x(8)+♥x(7)+♠x(6) +◊x(5)+∇x(4)

- h(n)=[♣, ♦, ♥, ♠, ◊, ∇]
- e_0 =[♣, ♠] → [x(0),x(3),…, x(kM+0)]
- e_1 =[♦, ◊] → [**0** x(2),x(5),…, x(kM+2)]

Polyphase decimation

y(0)= ♣x(0) + …?

y(3)= ♣x(3) +♦x(2)+♥x(1)+♠x(0)

y(6)= ♣x(6) +♦x(5)+♥x(4)+♠x(3) +◊x(2)+∇x(1)

y(9)= ♣x(9) +♦x(8)+♥x(7)+♠x(6) +◊x(5)+∇x(4)

- h(n)=[♣, ♦, ♥, ♠, ◊, ∇]
- e_0 =[♣, ♠] → [x(0),x(3),…, x(kM+0)]
- e_1 =[♦, ◊] → [**0** x(2),x(5),…, x(kM+2)]
- e_2 =[♥, ∇] → [**0** x(1),x(4),…, x(kM-m)]

Polyphase decimation

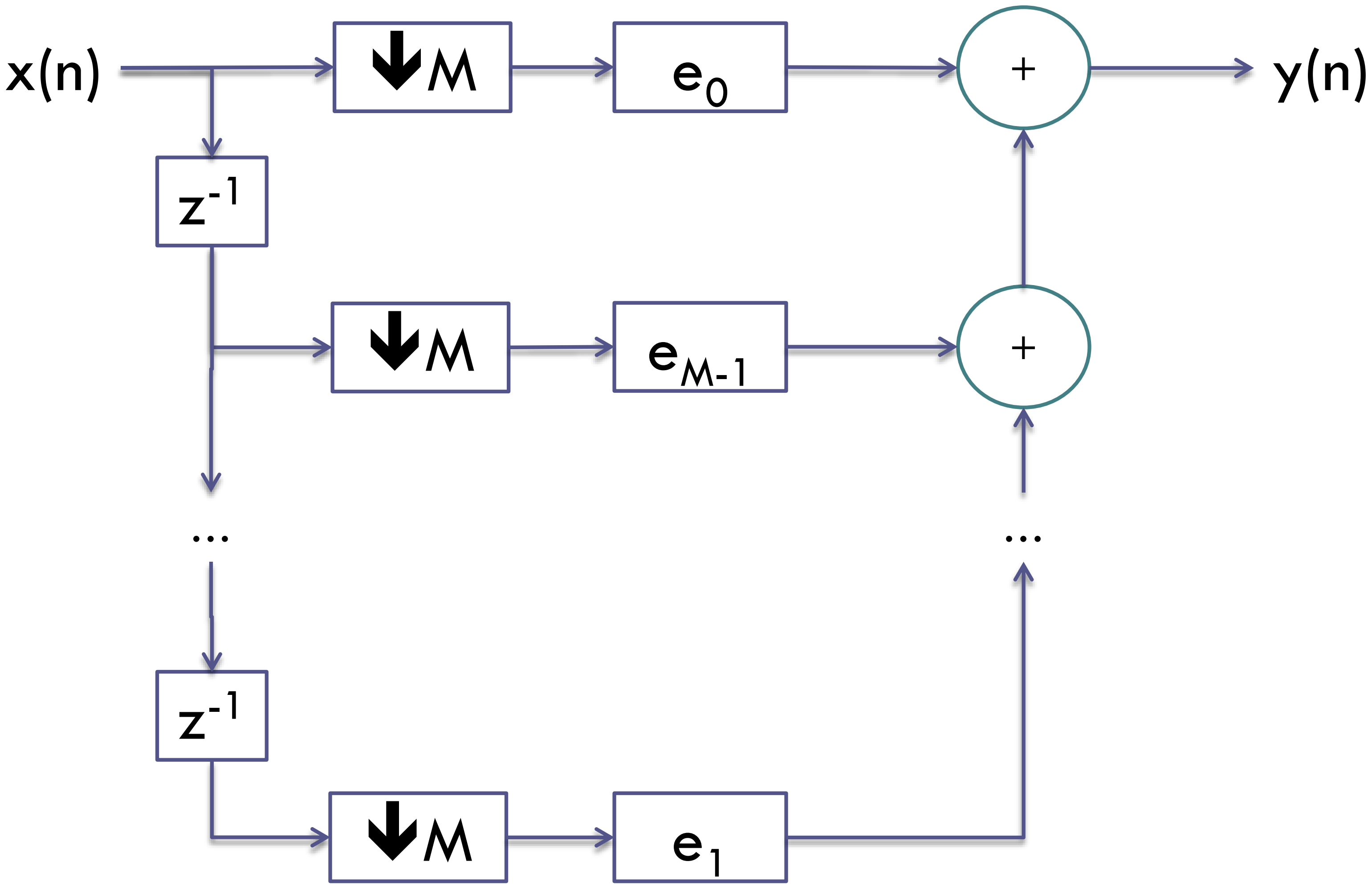
- Given a filter *h* with L weights
- Given a signal *x* with M down-sampling factor
- we have $K = \lceil L/M \rceil$ weights for each subfilter
- Each subfilter e_m is build as:
  - $e_m(k) = h(kM + m)$
    - k=[0,1…,K-1]
    - m=[0,1,…,M-1]
- Write a function `e=decompose_filter(h, M)`
  - `e` is a matrix with K rows (weights) and M columns (subfilters)

Polyphase decimation

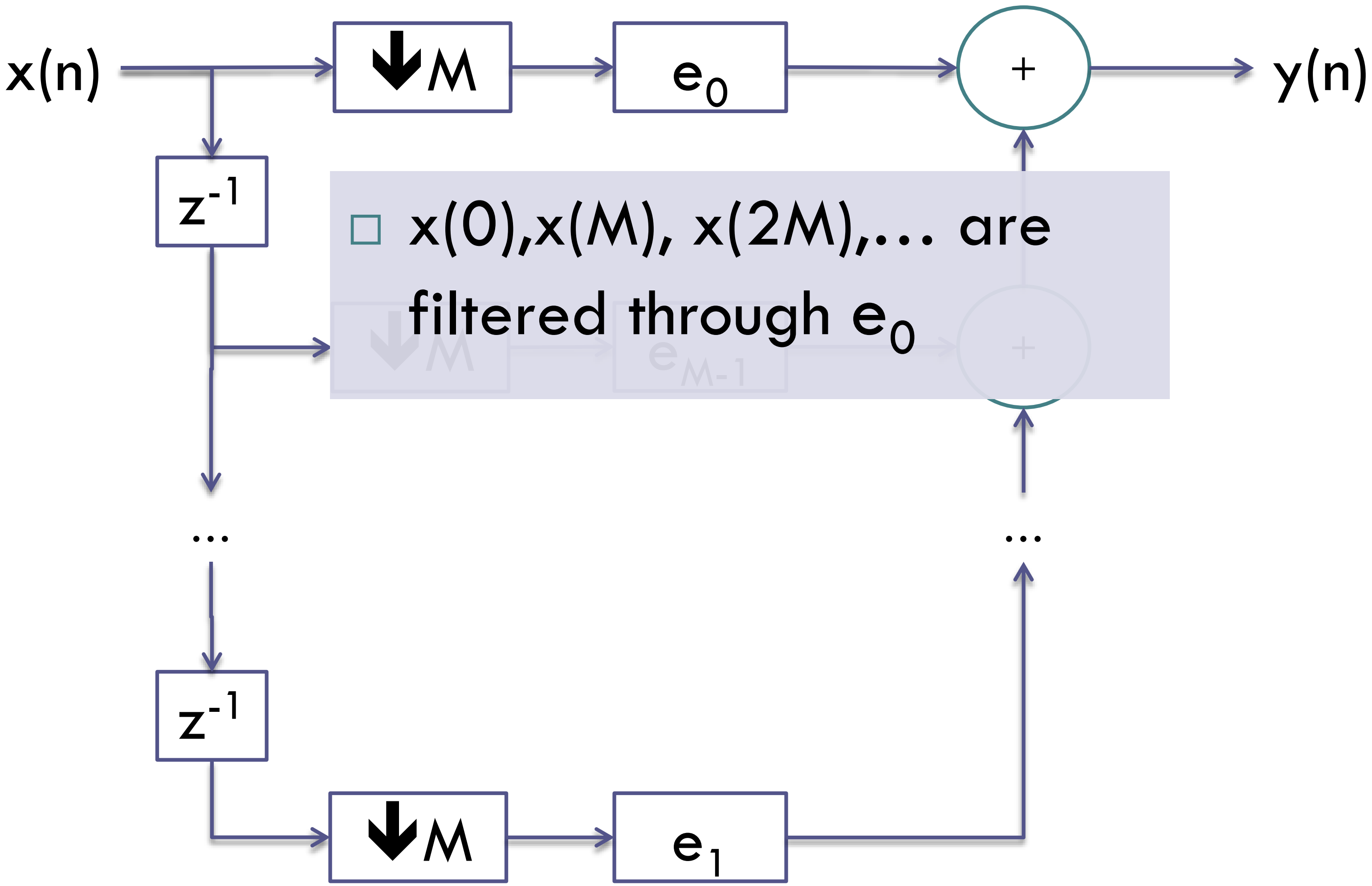

```
function [ e ] = decompose_filter(h, M )
  L=length(h); K=ceil(L/M);
  e=zeros(K,M);
  if K*M>L
     h(K*M)=0; % zero-pad up to K*M
  end
  for m=1:M
    sub=h(m:M:end);
    e(:,m)=sub(:);
  end
end
```

- Try to implement it using the `reshape` function

Polyphase decimation

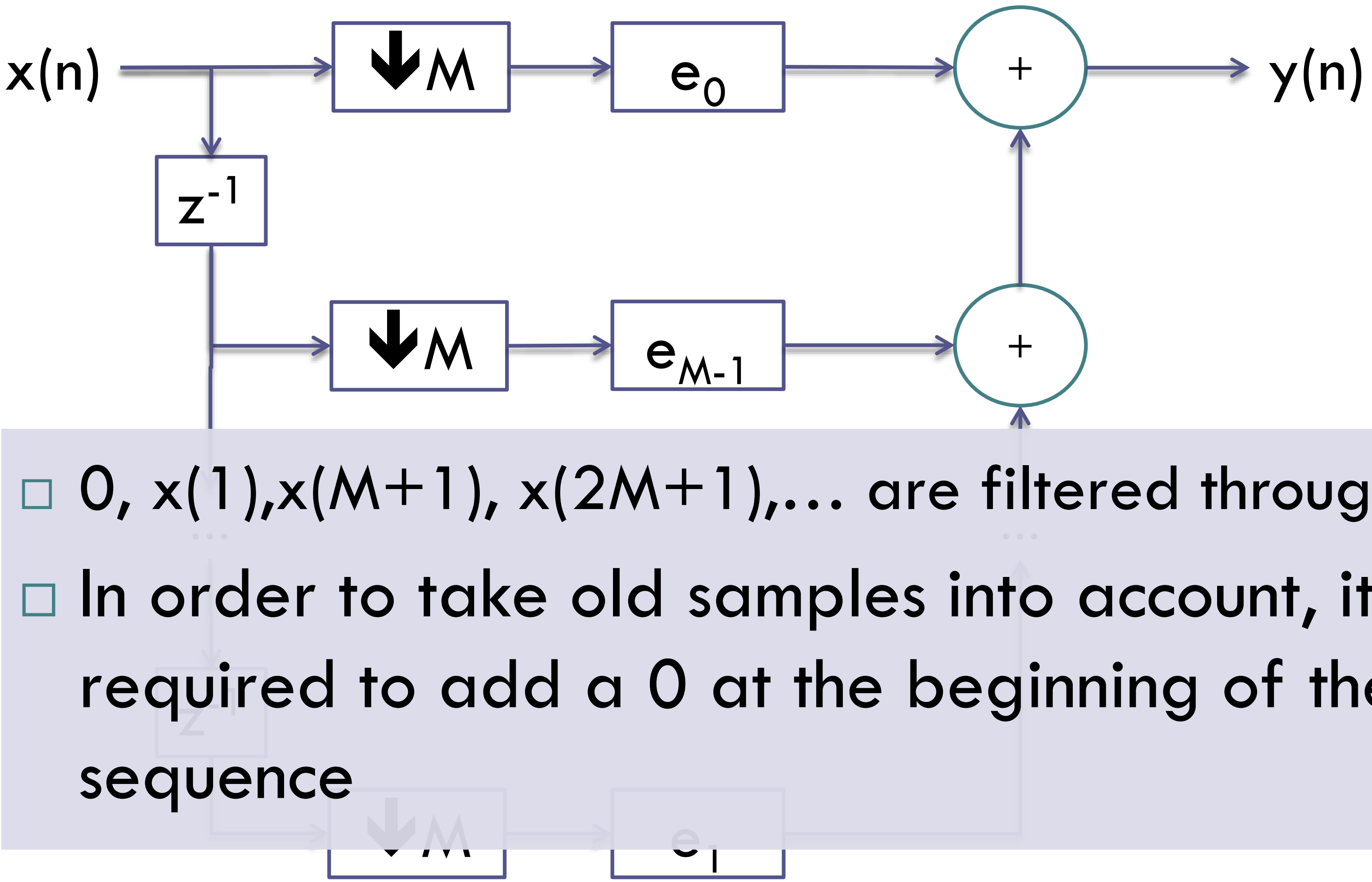


Polyphase decimation



- x(0),x(M), x(2M),... are filtered through e_0

Polyphase decimation



- 0, x(1),x(M+1), x(2M+1),… are filtered through e_{M-1}
- In order to take old samples into account, it is required to add a 0 at the beginning of the sequence

Polyphase decimation

- Given a sequence *x* with L weights
- Given a signal *x* with M down-sampling factor
- we have $K = \lceil L/M \rceil$ weights for each subfilter e_m
- We build K subsequences x_m
  - $x_m(k) =$
    - 0 if k==0 and m!=0 (take old samples into account)
    - $x(Mk - m)$ otherwise
    - k=[0,1…,K-1]
    - m=[0,1,…,M-1]
- Write a function `x_m=decompose_signal(x, M)`
  - `x_m` is a matrix with Nm rows (samples) and M columns (subfilters)

Polyphase decimation


```
function [ x_m ] = decompose_signal( x,M )
    Nm=floor(length(x)/M);
    x_m=zeros(Nm,M);
    x_0=x(1:M:end);
    x_m(1:length(x_0),1)=x_0;
    for m=1:M-1
        x_=x(M-(m-1):M:end);
        x_m(2:length(x_)+1,m+1)=x_;
    end
end
```

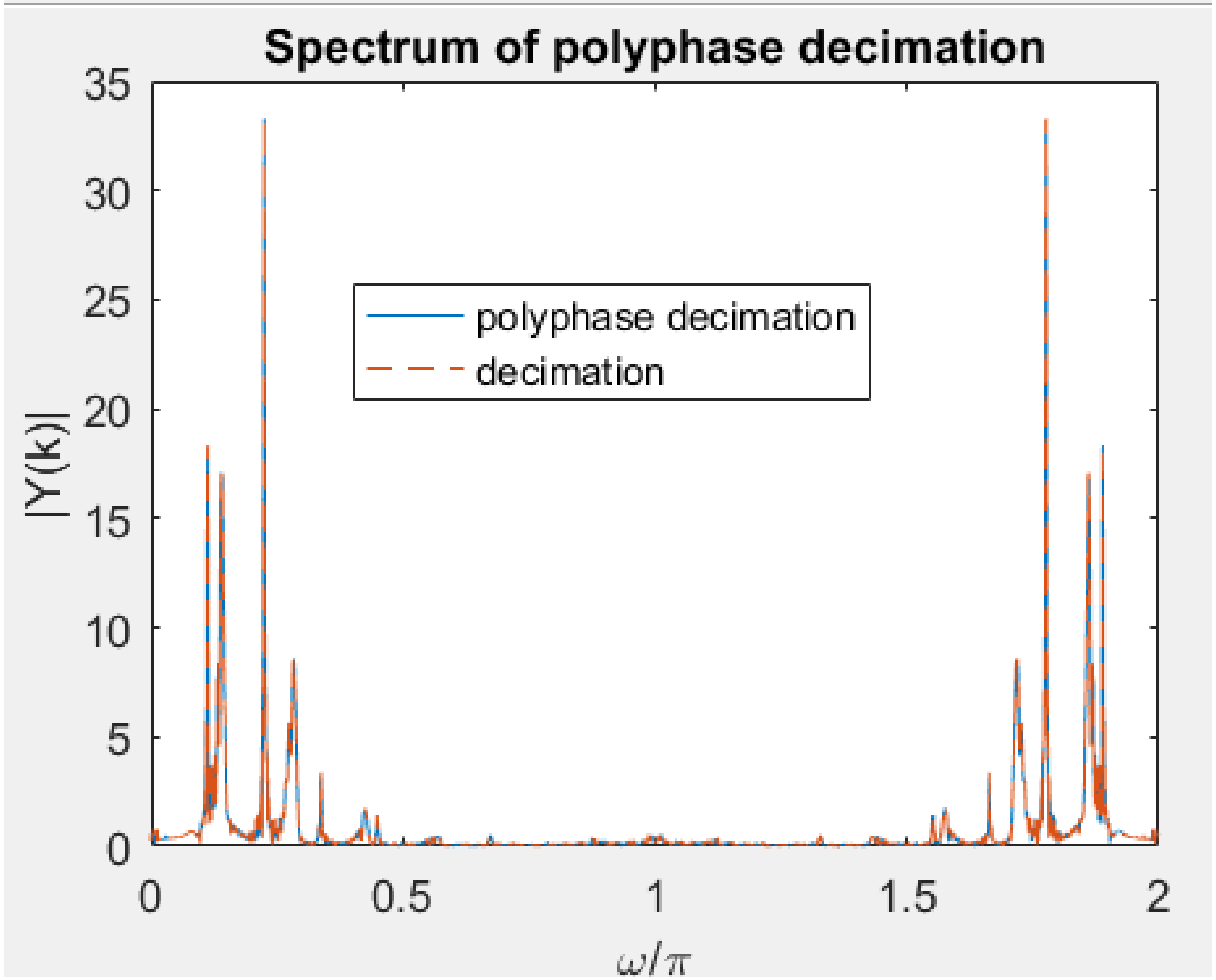
Polyphase decimation


```
[x,Fs]=audioread('Toms_diner_16.wav');
n_x=[0:length(x)-1]/Fs;
M=4; h=fir1(32,1/M);

e=decompose_filter(h,M);
x_m=decompose_signal(x,M);

y=zeros(size(x_m(:,1)));
for m=1:M
   y_m=filter(e(:,m),1,x_m(:,m));
   y=y+y_m;
end
n_y=[0:length(y)-1]/(Fs/M);
```

Polyphase decimation



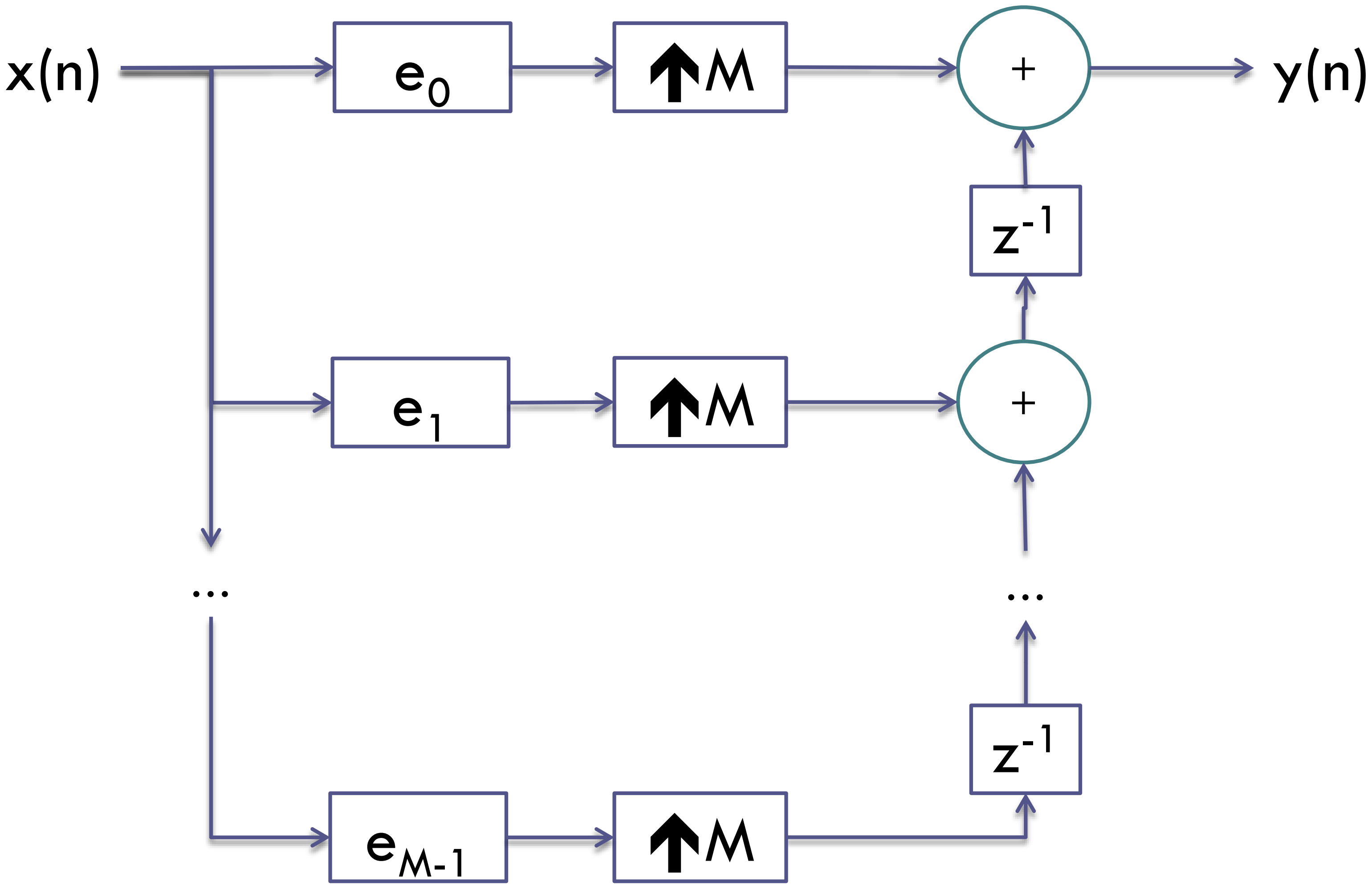
Polyphase interpolation

- Polyphase interpolation is the dual of polyphase decimation
  - In classic intepolation, the sequence is upsampled before the low-pass filtering
  - a signal L times longer then the original is filtered
- Polyphase interpolation is used to achieve computational efficiency

Polyphase interpolation

- Given a filter h with L weights
- Given a signal with M up-sampling factor
- we have $K = \lceil L/M \rceil$ weights for each subfilter
- Each subfilter e_m is build as:
  - $e_m(k) = h(kM + m)$
    - k=[0,1…,K-1]
    - m=[0,1,…,M-1]

Polyphase interpolation



Polyphase interpolation


```
% see above for y and e
y_int=zeros(length(y)*M+M,1);
for m=1:M
  y_m_f=filter(M*e(:,m),1,y); % scaling
  y_int(m:M:end-M)=y_int(m:M:end-M)+y_m_f;
End
```

Polyphase interpolation