Signal Processing Using Scilab

Manas Das
Indian Institute of Technology, Bombay

February 21, 2012

Introduction

- Signal Basics

Introduction

- Signal Basics
  - What is signal?

Introduction

- Signal Basics
  - What is signal?
  - Different types of signal

Introduction

- Signal Basics
  - What is signal?
  - Different types of signal
    - Impulse function

Introduction

- Signal Basics
  - What is signal?
  - Different types of signal
    - Impulse function
    - Step function

Introduction

- Signal Basics
  - What is signal?
  - Different types of signal
    - Impulse function
    - Step function
    - Ramp function

Introduction

- Signal Basics
  - What is signal?
  - Different types of signal
    - Impulse function
    - Step function
    - Ramp function

Linear Time-Invariant Systems

Convolution

- Continuous time convolution

Linear Time-Invariant Systems

Convolution

- Continuous time convolution
- Discrete time convolution

Linear Time-Invariant Systems

Convolution

- Continuous time convolution
- Discrete time convolution
- Circular convolution

Linear Time-Invariant Systems

Convolution

- Continuous time convolution
- Discrete time convolution
- Circular convolution
- Correlation

Linear Time-Invariant Systems

Convolution

- Continuous time convolution
- Discrete time convolution
- Circular convolution
- Correlation

Different types of Transform

- Fourier Transform

Different types of Transform

- Fourier Transform
- Laplace Transform

Different types of Transform

- Fourier Transform
- Laplace Transform
- Z-Transform

Different types of Transform

- Fourier Transform
- Laplace Transform
- Z-Transform

Fourier Transform

A mathematical operation that converts signal from time domain to its frequency domain

Fourier Transform

A mathematical operation that converts signal from time domain to its frequency domain

- Fourier Transform of continuous time signal

Fourier Transform

A mathematical operation that converts signal from time domain to its frequency domain

- Fourier Transform of continuous time signal
- Fourier Transform of Discrete time signal

Fourier Transform

A mathematical operation that converts signal from time domain to its frequency domain

- Fourier Transform of continuous time signal
- Fourier Transform of Discrete time signal
  - Discrete Fourier Transform (DFT)
  - Fast Fourier Transform(FFT)

DFT

Discrete Fourier transform

DFT

Discrete Fourier transform

Calling Sequence

[xf]=dft(x,flag);

- x :input vector

DFT

Discrete Fourier transform

Calling Sequence

```
[xf]=dft(x,flag);
```

- x :input vector
- flag: indicates dft (flag=-1) or idft (flag=1)

DFT

Discrete Fourier transform

Calling Sequence

```
[xf]=dft(x,flag);
```

- x :input vector
- flag: indicates dft (flag=-1) or idft (flag=1)
- xf: output vector

DFT

Discrete Fourier transform

Calling Sequence

```
[xf]=dft(x,flag);
```

- x :input vector
- flag: indicates dft (flag=-1) or idft (flag=1)
- xf: output vector

FFT

Fast Fourier transform

FFT

Fast Fourier transform

Calling Sequence

```
[x]=fft(a);
```

- x :real or complex vector

FFT

Fast Fourier transform

Calling Sequence

```
[x]=fft(a);
```

- x :real or complex vector
- a: real or complex vector, matrix or multidimensional array.

FFT

Fast Fourier transform

Calling Sequence

```
[x]=fft(a);
```

- x :real or complex vector
- a: real or complex vector, matrix or multidimensional array.

Laplace Transform

A linear operator of a function f(t) with a real argument t (t=0) that transforms it to a function F(s) with a complex argument s

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`
- `phi:  phase increment`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`
- `phi:  phase increment`
- `a:initial magnitude`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`
- `phi:  phase increment`
- `a:initial magnitude`
- `theta:initial phase`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`
- `phi:  phase increment`
- `a:initial magnitude`
- `theta:initial phase`
- `czx:chirp z-transform output`

z-Transform

Calling Sequence

`[czx]=czt(x,m,w,phi,a,theta)`

Arguements

- `x :input data sequence`
- `m:  czt is evaluated at m points in z-plane`
- `w:magnitude multiplier`
- `phi:  phase increment`
- `a:initial magnitude`
- `theta:initial phase`
- `czx:chirp z-transform output`

Sampling

- Nyquist Criteria

Sampling

- Nyquist Criteria
  An analog signal that has been sampled can be perfectly reconstructed from an infinite sequence of samples if the sampling rate exceeds 2B samples per second, where B is the highest frequency of the original signal
- Aliasing

Sampling

- Nyquist Criteria
  An analog signal that has been sampled can be perfectly reconstructed from an infinite sequence of samples if the sampling rate exceeds 2B samples per second, where B is the highest frequency of the original signal
- Aliasing
  - Ambiguity from reconstruction

Sampling

- Nyquist Criteria
  An analog signal that has been sampled can be perfectly reconstructed from an infinite sequence of samples if the sampling rate exceeds 2B samples per second, where B is the highest frequency of the original signal
- Aliasing
  - Ambiguity from reconstruction
  - Shannon-Nyquist Sampling theorem

Sampling

- Nyquist Criteria
  An analog signal that has been sampled can be perfectly reconstructed from an infinite sequence of samples if the sampling rate exceeds 2B samples per second, where B is the highest frequency of the original signal
- Aliasing
  - Ambiguity from reconstruction
  - Shannon-Nyquist Sampling theorem
  - Under-sampling

FIR Filter Equation

FIR Filters are represented as:-

$$H(z) = \sum_{k=0}^{N} h_k Z^{-k} \tag{1}$$

Window Functions for FIR Filter Design

- `Hamming Window`

Window Functions for FIR Filter Design

- Hamming Window
  `win_hamming=window('hm',n)`
- Kaiser Window

Window Functions for FIR Filter Design

- Hamming Window
  `win_hamming=window('hm',n)`
- Kaiser Window
  `win_kaiser=window('kr',n,alpha)`
- Chebyshev Window

Window Functions for FIR Filter Design

- Hamming Window
  `win_hamming=window('hm',n)`
- Kaiser Window
  `win_kaiser=window('kr',n,alpha)`
- Chebyshev Window
  `win_chebyshev=window('ch',n,par)`

Window based Linear Phase FIR filter

Calling Sequence

`[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)`

Arguments

- `ftype:  'lp','hp','bp','sb' (filter type)`

Window based Linear Phase FIR filter

Calling Sequence

`[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)`

Arguments

- `ftype:  'lp','hp','bp','sb' (filter type)`
- `wtype:  Window type ('re','tr','hm','hn','kr','ch')`

Window based Linear Phase FIR filter

Calling Sequence

`[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)`

Arguments

- `ftype:  'lp','hp','bp','sb' (filter type)`
- `wtype:  Window type ('re','tr','hm','hn','kr','ch')`
- `wft:  time domain filter coefficients`

Window based Linear Phase FIR filter

Calling Sequence

`[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)`

Arguments

- `ftype: 'lp','hp','bp','sb' (filter type)`
- `wtype: Window type ('re','tr','hm','hn','kr','ch')`
- `wft: time domain filter coefficients`
- `wfm: frequency domain filter response on the grid fr`

Window based Linear Phase FIR filter

Calling Sequence

`[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)`

Arguments

- `ftype: 'lp','hp','bp','sb' (filter type)`
- `wtype: Window type ('re','tr','hm','hn','kr','ch')`
- `wft: time domain filter coefficients`
- `wfm: frequency domain filter response on the grid fr`
- `fr: Frequency grid`

Equiripple FIR Filter Design

Calling Sequence

`[hn]=eqfir(nf,bedge,des,wate)`

Arguments

- `nf: number of output filter points desired`

Equiripple FIR Filter Design

Calling Sequence

`[hn]=eqfir(nf,bedge,des,wate)`

Arguments

- `nf: number of output filter points desired`
- `bedge: Mx2 matrix giving a pair of edges for each band`

Equiripple FIR Filter Design

Calling Sequence

`[hn]=eqfir(nf,bedge,des,wate)`

Arguments

- `nf: number of output filter points desired`
- `bedge: Mx2 matrix giving a pair of edges for each band`
- `des: M-vector giving desired magnitude for each band`

Equiripple FIR Filter Design

Calling Sequence

`[hn]=eqfir(nf,bedge,des,wate)`

Arguments

- `nf:` number of output filter points desired
- `bedge:` Mx2 matrix giving a pair of edges for each band
- `des:` M-vector giving desired magnitude for each band
- `wate:` M-vector giving relative weight of error in each band

Equiripple FIR Filter Design

Calling Sequence

`[hn]=eqfir(nf,bedge,des,wate)`

Arguments

- `nf:` number of output filter points desired
- `bedge:` Mx2 matrix giving a pair of edges for each band
- `des:` M-vector giving desired magnitude for each band
- `wate:` M-vector giving relative weight of error in each band
- `hn:` output of linear-phase FIR filter coefficients

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- `n:the filter order`

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- `n:the filter order`
- `ftype:filter type,lp' for low-pass,'hp' for high pass,'bp' for band pass and 'sb' for stop band`

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- n:the filter order
- ftype:filter type,lp' for low-pass,'hp' for high pass,'bp' for band pass and 'sb' for stop band
- fdesign:the analog filter design, the possible values are: 'butt', 'cheb1', 'cheb2' and 'ellip'

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- n:the filter order
- ftype:filter type,lp' for low-pass,'hp' for high pass,'bp' for band pass and 'sb' for stop band
- fdesign:the analog filter design, the possible values are: 'butt', 'cheb1', 'cheb2' and 'ellip'
- frq:2-vector of discrete cut-off frequencies (i.e., 0<frq<.5). For 'lp' and 'hp' filters only frq(1) is used. For 'bp' and 'sb' filters frq(1) is the lower cut-off frequency and frq(2) is the upper cut-off frequency

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- n:the filter order
- ftype:filter type,lp' for low-pass,'hp' for high pass,'bp' for band pass and 'sb' for stop band
- fdesign:the analog filter design, the possible values are: 'butt', 'cheb1', 'cheb2' and 'ellip'
- frq:2-vector of discrete cut-off frequencies (i.e., 0<frq<.5). For 'lp' and 'hp' filters only frq(1) is used. For 'bp' and 'sb' filters frq(1) is the lower cut-off frequency and frq(2) is the upper cut-off frequency
- delta: 2-vector of error values

IIR Digitalfilter

Calling Sequence

`[hz]=iir(n,ftype,fdesign,frq,delta)`

Arguments

- n:the filter order
- ftype:filter type,lp' for low-pass,'hp' for high pass,'bp' for band pass and 'sb' for stop band
- fdesign:the analog filter design, the possible values are: 'butt', 'cheb1', 'cheb2' and 'ellip'
- frq:2-vector of discrete cut-off frequencies (i.e., 0<frq<.5). For 'lp' and 'hp' filters only frq(1) is used. For 'bp' and 'sb' filters frq(1) is the lower cut-off frequency and frq(2) is the upper cut-off frequency
- delta: 2-vector of error values

Filter

To design filter of any magnitude
Function- fremezb
Calling Sequence
an=remezb(nc,fg,ds,wt)

- nc: half-filter length
- fg: dense grid of frequency
- s: derived magnitude values on this grid
- wt: error weighting vectors
- an: filter coeffficients

Filter

Filtering of discrete signals by **flts** function
Function- **flts**
Calling Sequence
y,[x]=**flts(u,s1[,x0])**

- **u: hte data to be filtered**
- **x0: initial state vector/matrix giving necessaty i/p-o/p.it allows for filtering of length signals**
- **x: optimal variable which givesh the state sequence.**

Textbook Companion

- You already know Textbook Companion Project
- There are books on Signal Processing using Scilab under this project

Textbook Companion

- You already know Textbook Companion Project
- There are books on Signal Processing using Scilab under this project
- Refer to the link: http://www.scilab.in/Completed_Books