MATLAB Introduction

Overview Part 1

Andreas Spanias and Ted Painter

Spring 2000

Outline, Part I: MATLAB Overview

I. System Summary, Getting started

II. Data Types

III. M-Files: scripts, functions

IV. Programming

V. Visualization

VI. DSP Application Example

VII. Debugging, Profiling

System Summary

A. MATLAB vs. C (or FORTRAN or other)

B. Help system

C. Built-in functions

D. Toolboxes

E. M-files: functions and scripts

F. Editor

G. Debugger

H. Compiler, External Interfaces

Getting Started

`help`	Online help system, command window
`helpwin`	" " , separate window
`helpdesk`	" " , web browser
`doc`	
`lookfor`	Help keyword search

Help System Functions

`version`	MATLAB version number
`whatsnew`	Display MATLAB READMEs

About… Functions

Getting Started

`UPARROW`	Repeat last command(s) (stack)
`Quit`	Terminate MATLAB
`!, dos()`	Execute operating system command
`cd`	Change working directory
`dir`	Directory listing
`what`	Dir: M-files, MAT-files, MEX-files
`delete`	Delete files and graphics objects
`type`	List file
`path`	Control directory search path
`addpath`	Add directories to search path
`rmpath`	Remove directories from search path
`edit`	Edit an M-file

OS Functions

MATLAB Introduction

Overview Part 2

Andreas Spanias and Ted Painter

Spring 2000

Data Types

MATLAB data types:

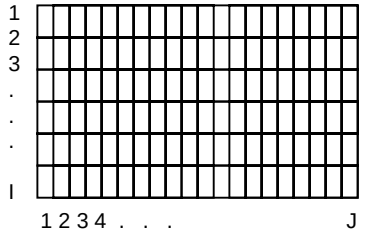
- Scalars

```
x = 7;    z = 7+2j;
```

- Vectors


```
v = 1:0.1:2;
w = [ a b 2 2 ];
```

- Matrices



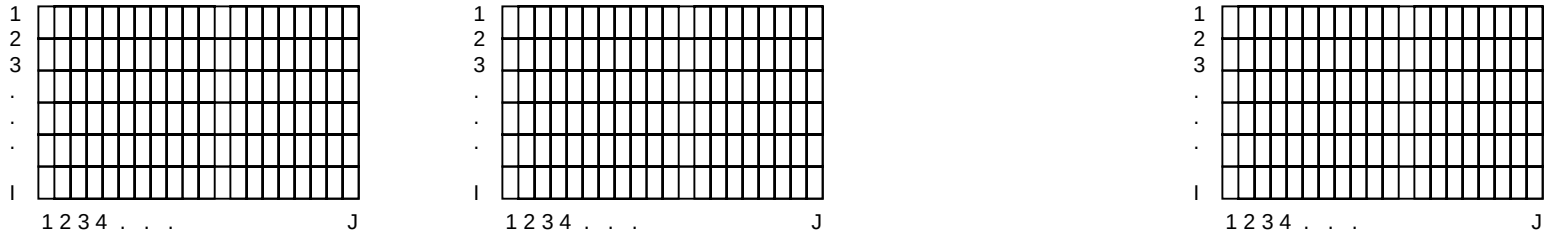

```
A = [ 1:3; 8:10; 4 6 8 ];
B = [ A, A' ];
```

Automatic allocation, no declarations.

NOTE: NAMES ARE CASE SENSITIVE!

Data Types

- Muti-D



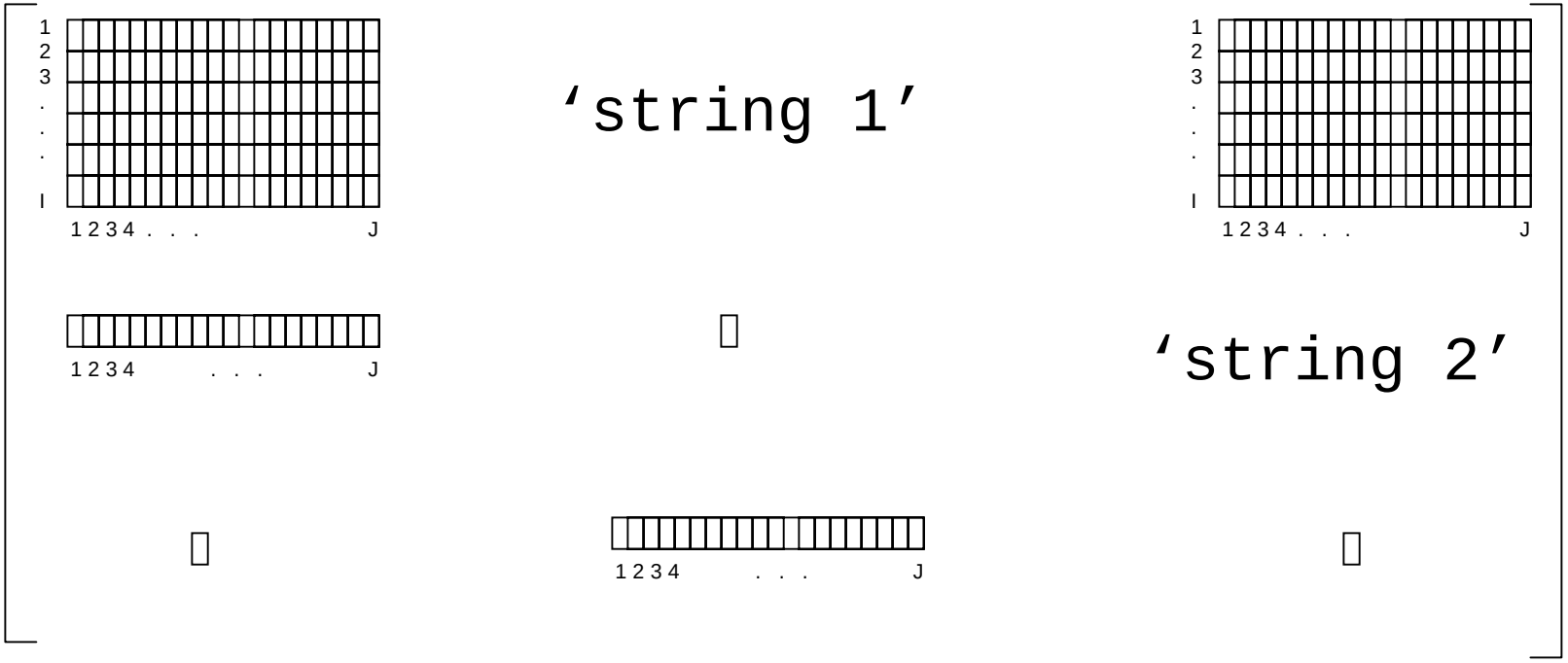
1 2 . . . K

```
M(:,:,1) = A;
M(:,:,2) = -A;
M(:,:,3) = A+B;
```

3-D example; can be extended to M-D.

Data Types

- Cell Arrays




```
c{1,1} = randn(5,5);
c{1,2} = 'string 1';
c{3,3} = 97+3j
```

3 x 3 example; can be arbitrarily extended.

Data Types

- Characters

```
x = ‘a’;
```

- Strings

```
msg1 = ‘CYSIP’;
msg2 = ‘MATLAB Intro’;
```

```
who          clear      clear global
whos         load       clear all
size         save
length       pack
```

Data structure and workspace functions

Function Calls

```
[ res1, res2, ... ] = funcname( arg1, arg2, ...);
```

```
x = [ 1:5; 11:15 ];

y = sum(x)

y =
     12     14     16     18     20
```

Example Function Call: sum()

M-files: Functions

```
function [ a, b, c ] = fname(arg1, arg2, . . .)

   % Help line 1
   % Help text
   % More help text
   {function body}
   {
   a = ;
   b = ;
   c =;
   }
   or
   {return}
```

Function Structure (in text file `fname.m`)

NOTES

1. `fname` **should match actual filename (.m)**
2. **All help text is optional**
3. **"{" and "}" denote optional contents**

M-files: Functions

```
function [s,n] = fgen(f,fs,type,N)
% Function generator
% fgen(f,fs,type,N) returns sin or square wave
% vector of length N, depeding on type.
% f specifies the desired frequency
% fs specifies the desired sample rate

w=2*pi*f/fs;
n=0:(N-1);
switch type
   case 'c', s=cos(w.*n);
   case 's', s=square(w.*n);
end
```

Example 1-1: Function Generator (fgen.m)

M-files: Sub-functions

```
function [mu,stdev] = stat(x)
     n = length(x);
     mu = avg(x,n);
     stdev = sqrt(sum((x-mu).^2/n));


function mean = avg(x,n)
     mean = sum(x)/n;
```

Main function

Sub function

Example 1-2: Vector Statistics (stat.m)

M-files: Scripts

```
fs = 44100;
f = 1000;
stype = 'c';
N = 20000;
```

```
w=2*pi*f/fs;
n=0:(N-1);
switch stype
   case 'c', s=cos(w.*n);
   case 's', s=square(w.*n);
end
```

Example 1-3: Function Generator (fgens.m)

MATLAB Introduction

Overview Part 3

Andreas Spanias and Ted Painter

Spring 2000

Workspace and Scope

- **Memory partitioned into “workspaces”**
- **Base workspace contains global memory**
- **Function scope = local workspace**
- **Sub-function scope = local workspace**
- **Script scope = current workspace**
- **Scope modifier:** `global`

Data I/O

disp	load	auwrite
fprintf	save	auread
fscanf	fopen	wavread
fgets	fclose	wavwrite
fgetl	fread	feof
fseek	fwrite	ftell
sprintf	sscanf	

Partial List -- Should Look Familiar to ‘C’ Programmers

Special Values

`ans`	Most recent result
`eps`	Machine epsilon
`flops`	Total floating point ops during session
`i, j`	sqrt(-1)
`inf`	Infinity
`NaN`	Not-a-Number
`pi`	3.14...
`realmax`	Largest positive floating-point #
`realmin`	Smallest " " "

Partial List

Operators

1. **Arithmetic (+, -, *, /, etc.)**
2. **Relational (==, >, <, etc.)**
3. **Logical (AND, OR, NOT, etc.)**

Precedence

Arithmetic Operators

1. Transpose (.'), power (.^), hermitian (‘), matrix power (^)
2. Unary plus (+), Unary minus (-)
3. Mult (*), Div (./), Matrix mult. (*), Matrix div (/)
4. Addition (+), Subtraction (-)
5. Colon (:)

Relational Operators

1.	<	Less than
2.	<=	Less than or equal to
3.	>	Greater than
4.	>=	Greater than or equal to
5.	==	Equal to
6.	~=	Not equal to

Logical Operators and Functions

1.	&	AND
2.	|	OR
3.	~	NOT
4.	xor()	Exclusive OR
5.	all()	`u = [1 2 3], all(u < 3) = 0`
6.	any()	`u = [1 2 3], any(u < 3) = 1`
7.	find()	`u = [3 2 1], find(u < 3) = [2 3]`

MATLAB Introduction

Overview Part 4

Andreas Spanias and Ted Painter

Spring 2000

Indexing and Subscripting

```
A = magic(4) =  16   2   3  13
                 5  11  10   8
                 9   7   6  12
                 4  14  15   1

sum(A) = sum(A') = trace(A)
       = trace(flipud(A)) = 34

A(i,j) indexes row i, column j

A(1,3) = 3
A(3,1) = 9
A(:,3)' = 3 10 6 15
```

Indexing and Subscripting

```
A = magic(4) = 16   2   3  13
               5  11  10   8
               9   7   6  12
               4  14  15   1

A(2,2:4) = 11 10 8

A(2:3,3:4) = 10  8
              6 12

A(8) = 14

[1:3] + [4:6] = 5 7 9
```

Concatination

```
A=zeros(2,2);
B=ones(3,2);

C=[ [A-1;B+1], [B+3;A-4] ]

C =

    -1    -1     4     4
    -1    -1     4     4
     2     2     4     4
     2     2    -4    -4
     2     2    -4    -4
```

Conditional Branching

if, else, elseif

```
if x == y
     disp('x equals y')
elseif x > y
     disp('x bigger than y')
else
     disp('x smaller than y')
end
```

Conditional Branching

switch, case, otherwise

```
switch var
     case 1
          disp('1')
     case {2,3,4}
          disp('2 or 3 or 4')
     case 5
          disp('5')
     otherwise
          disp('Not in 1:5')
end
```

Looping

while

```
n=1
while prod(1:n) < 1e100
      n=n+1;
end

n =

     70
```

Finds first integer n for which n! contains 100-digits.

Looping

break

```
n=1
while 1
      if prod(1:n) >= 1e100
            break;
      else
            n=n+1
      end
end

n =

      70
```

Can be used to exit any loop.

Looping

for

```
M = 4;
N = 3;
for i = 1:M
     for j = 1:N
          A(i,j) = i+j-1;
     end
end

A =

     1     2     3
     2     3     4
     3     4     5
     4     5     6
```

Loop Vectorization

Vectorized code is more efficient

Inner loop eliminated

Row vector

```
M = 4;
N = 3;
j=1:N;
for i = 1:M
    A(i,:) = i+j-1;
end

A =

     1     2     3
     2     3     4
     3     4     5
     4     5     6
```

Loop Vectorization

Why Bother ? Consider equivalent code fragments:

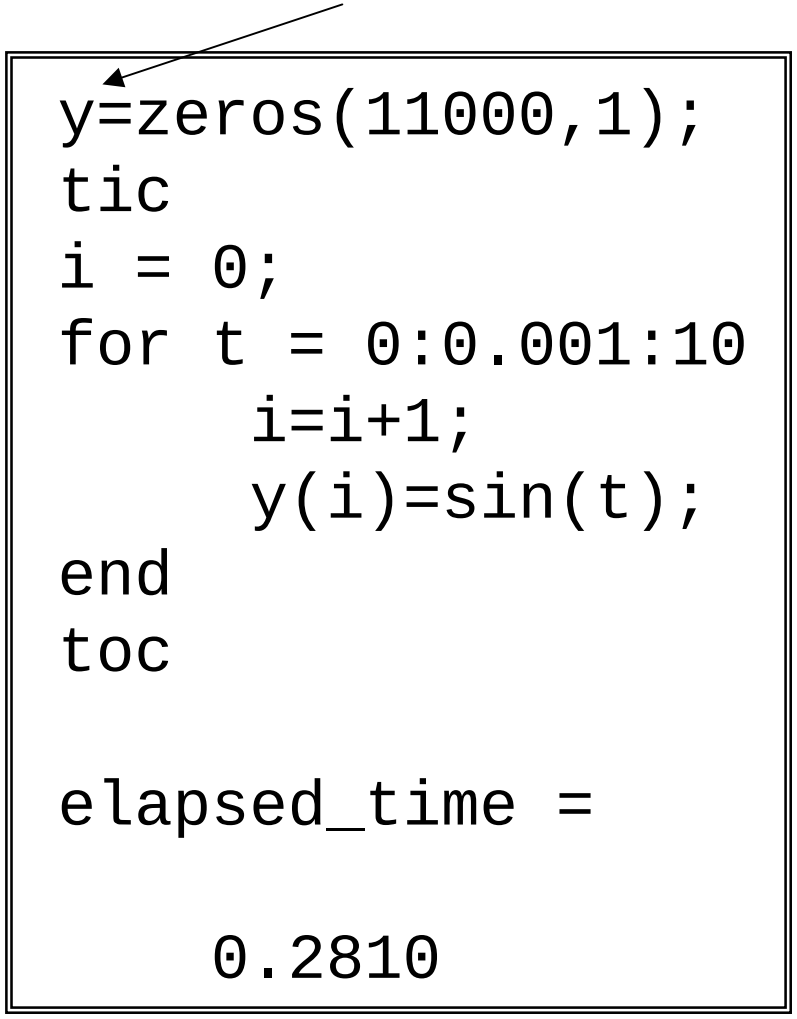
```
tic
i = 0;
for t = 0:0.001:10
     i=i+1;
     y(i)=sin(t);
end
toc

elapsed_time =

    3.8590
```

```
tic
t = 0:0.001:10;
y=sin(t);
toc

elapsed_time =

    0.0310
```

VECTORIZED:
125x FASTER !!!!

Preallocation

Allocate data structures before manipulating.



```
y=zeros(11000,1);
tic
i = 0;
for t = 0:0.001:10
     i=i+1;
     y(i)=sin(t);
end
toc

elapsed_time =

    0.2810
```

PREALLOCATED:

14x Speed Improvement

Visualization

Basic Visualization

`plot`	Plot vectors or matrices
`subplot`	Create tiled axes
`semilogx`	Semi-log scale plot, x-axis logarithmic
`semilogy`	" " ", y-axis
`loglog`	Log-log scale plot
`bar`	Vertical bar chart
`barh`	Horizontal bar chart
`hist`	Histograms
`hold`	Hold current graph (on/off)
`pie`	Pie chart
`print`	Print to clipboard, printer, or file

Partial List -- Demos for Each

Basic Visualization

`grid`	Grid lines on/off
`gtext`	Place text using mouse
`text`	Place text automatically
`plotyy`	Y-tick labels on left and right
`title`	Add title to plot
`xlabel`	Add x-axis label
`ylabel`	Add y-axis label

Partial List -- Demos for Each

Basic Visualization

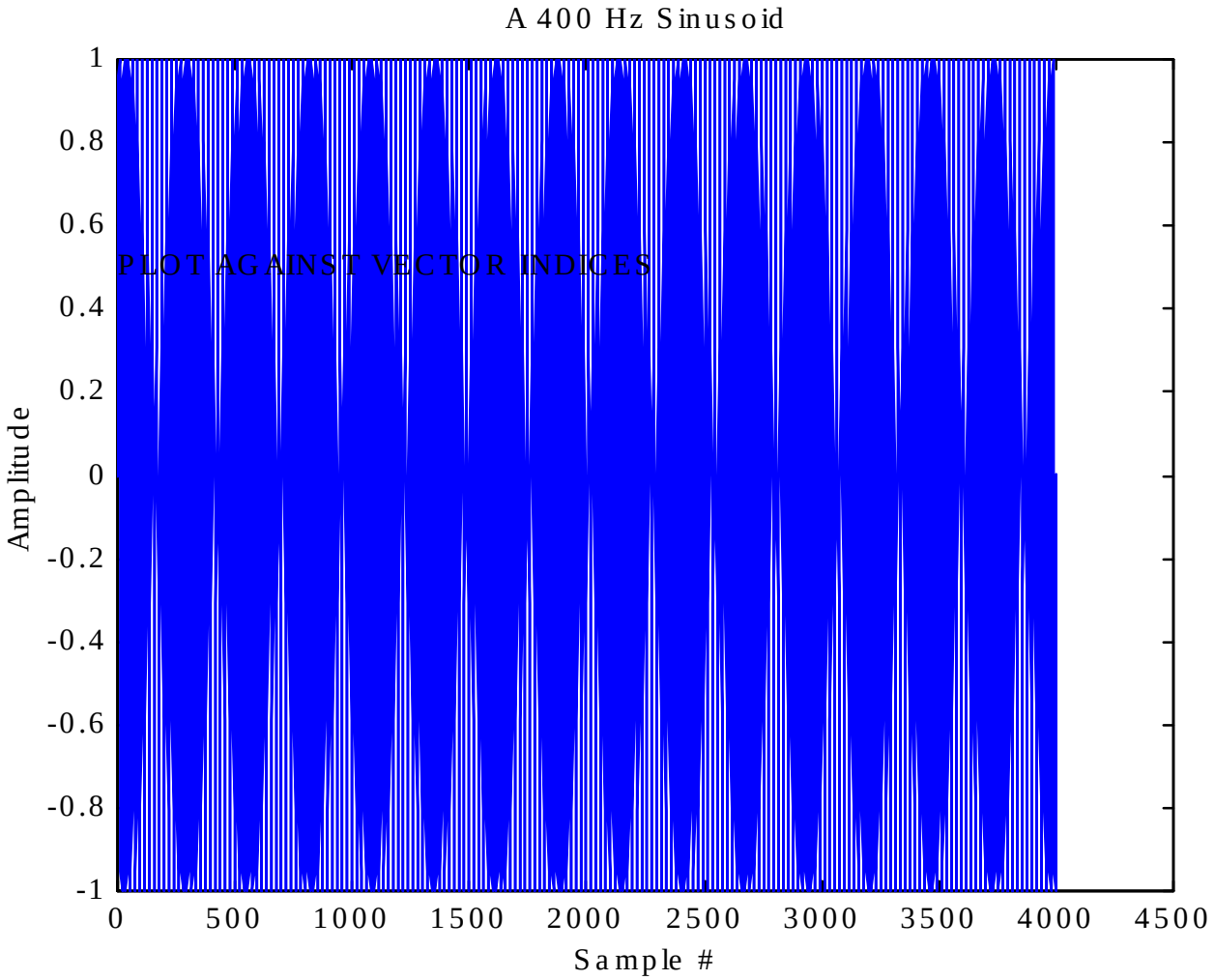
figure
plot
xlabel
ylabel
title

```
% Generate a signal, noise
n=(0:4000)';
N=length(n);
f=400;
fs=8000;
t=n/fs;
sig=sin(2*pi*f/fs.*n);
noise=randn(N,1);
noise=noise./(2*max(abs(noise)));
peak=max(sig);
i=find(sig==peak);
tp=n(i)/fs;
```

Example 1-4 (vis1.m)

Basic Visualization

```
figure
plot
xlabel
ylabel
title
```



Example 1-4 (vis1.m)

Basic Visualization

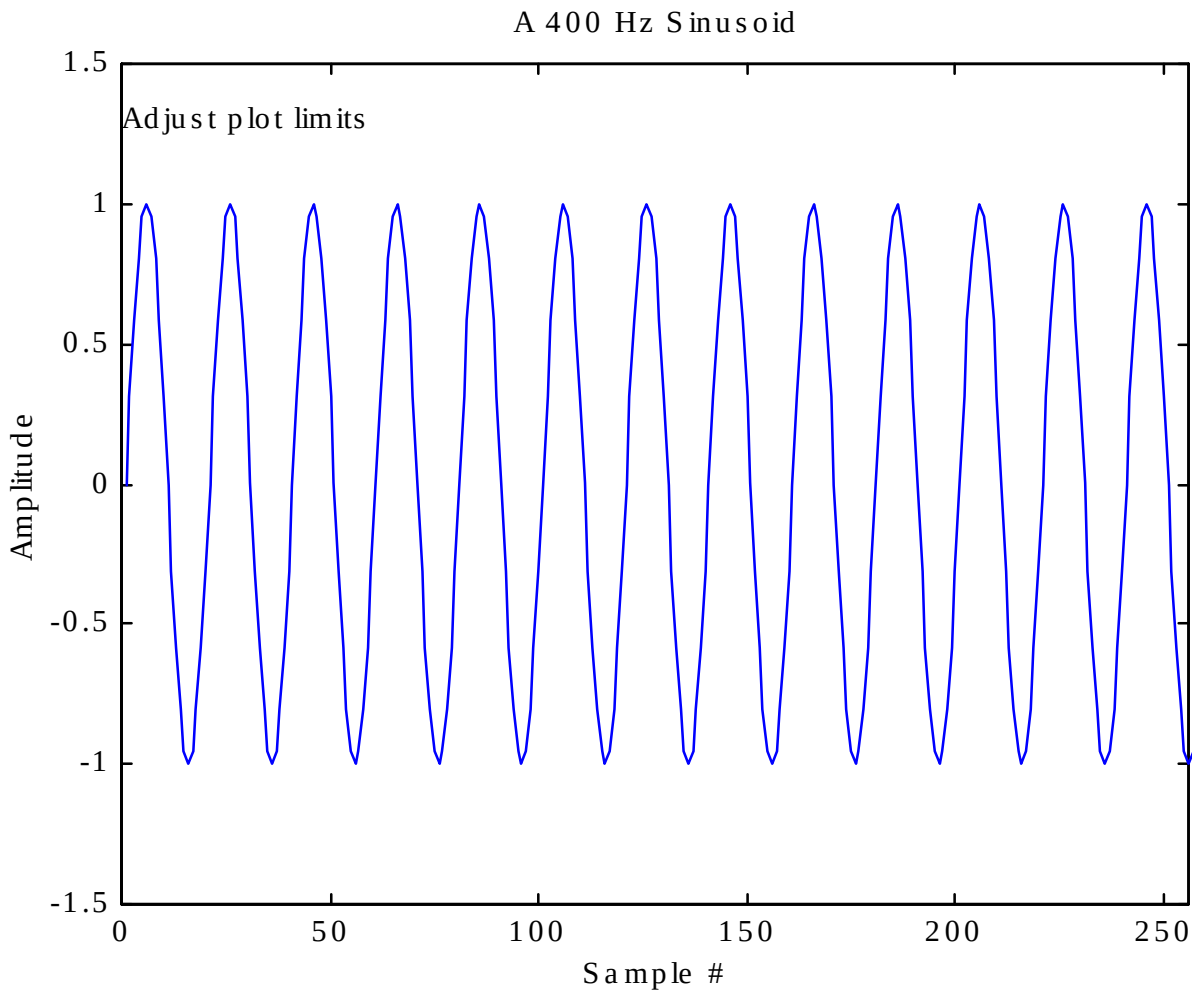
set 'xlim'
set 'ylim'
grid
subplot

```
% Adjust limits
set(gca,'xlim',[0 256]);
set(gca,'ylim',[-1.5 1.5]);

% Plot against time index
plot(t,sig);

% Two plots, special plot symbol and color
plot(t,sig,tp,sig(i),'rx');

% Add a grid
grid

% Create 2 subplots for signal, noise
subplot(211)
plot(t,sig,tp,sig(i),'rx');
subplot(212)
plot(t,2*noise,'g:');
```

Example 1-4 (vis1.m)

Basic Visualization

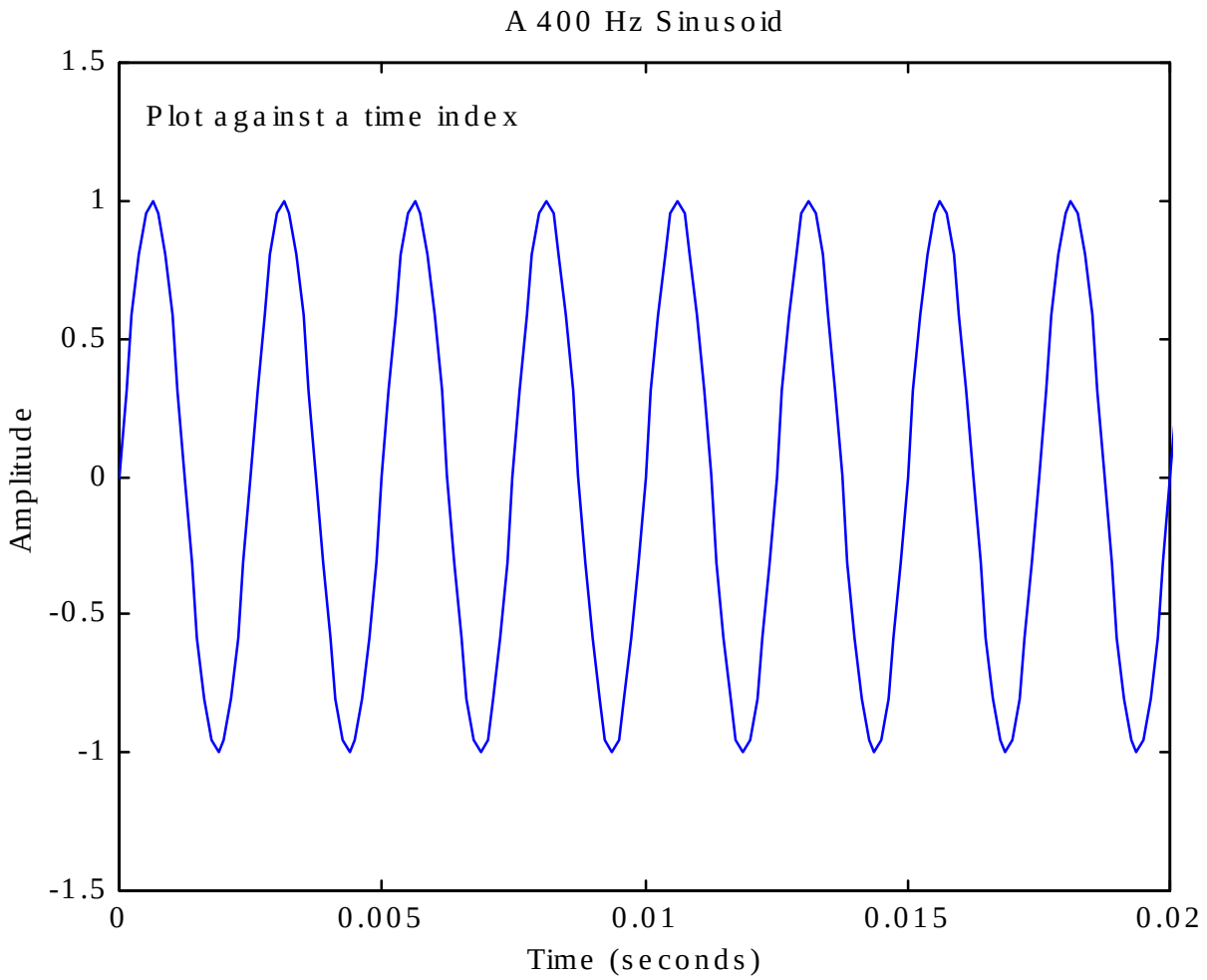
```
set ‘xlim’
set ‘ylim’
```



Example 1-4 (vis1.m)

Basic Visualization

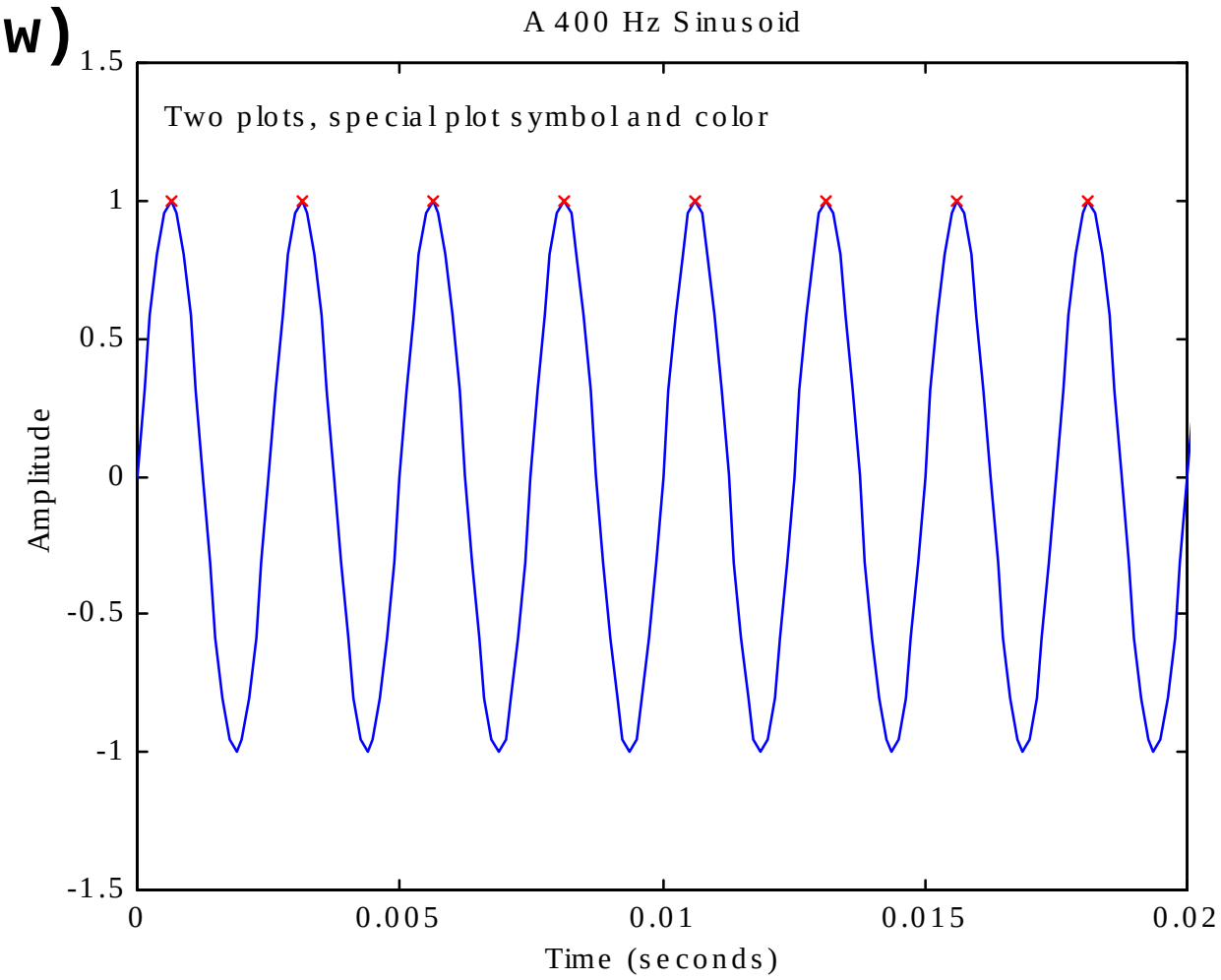
`plot(x,y)`



Example 1-4 (vis1.m)

Basic Visualization

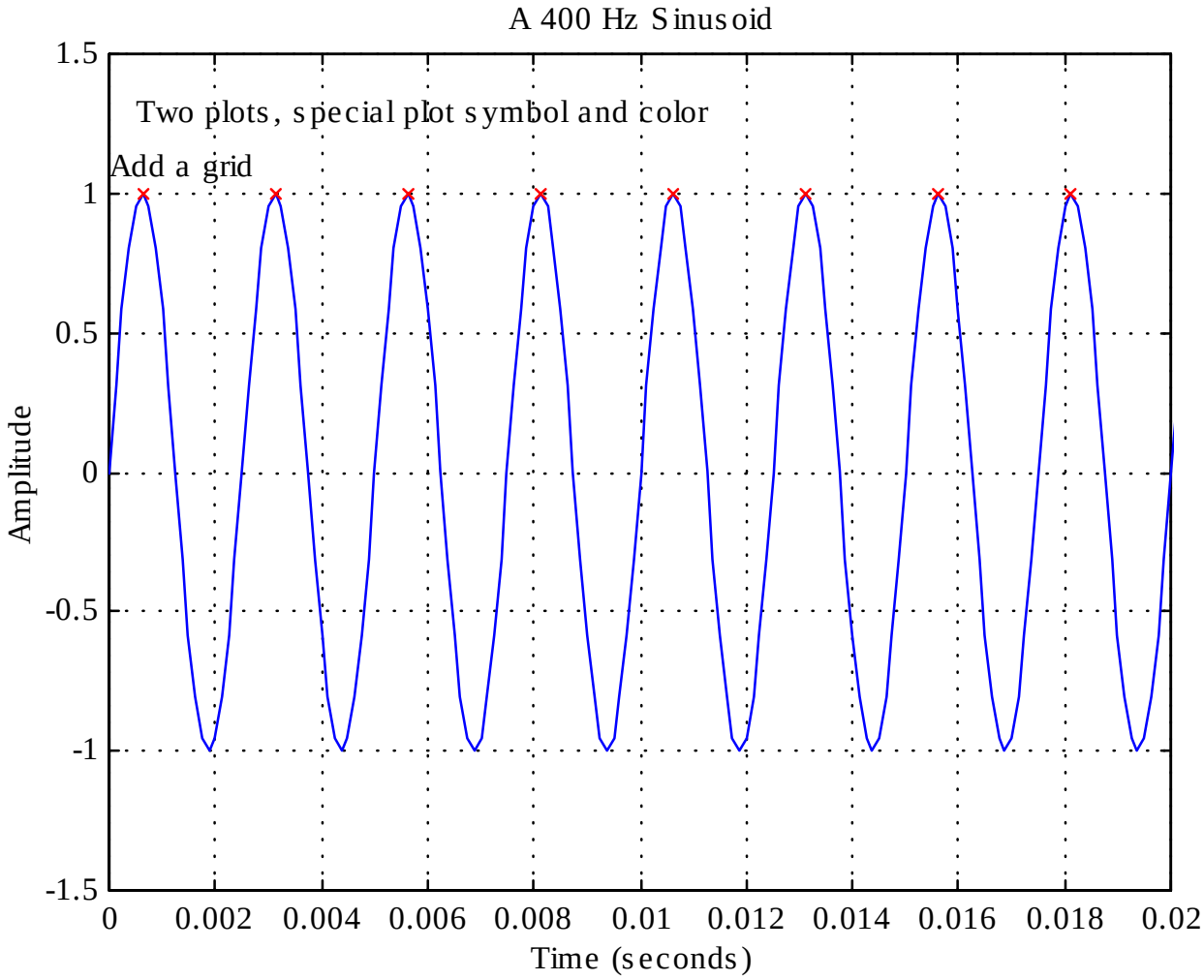
`plot(x,y,v,w)`



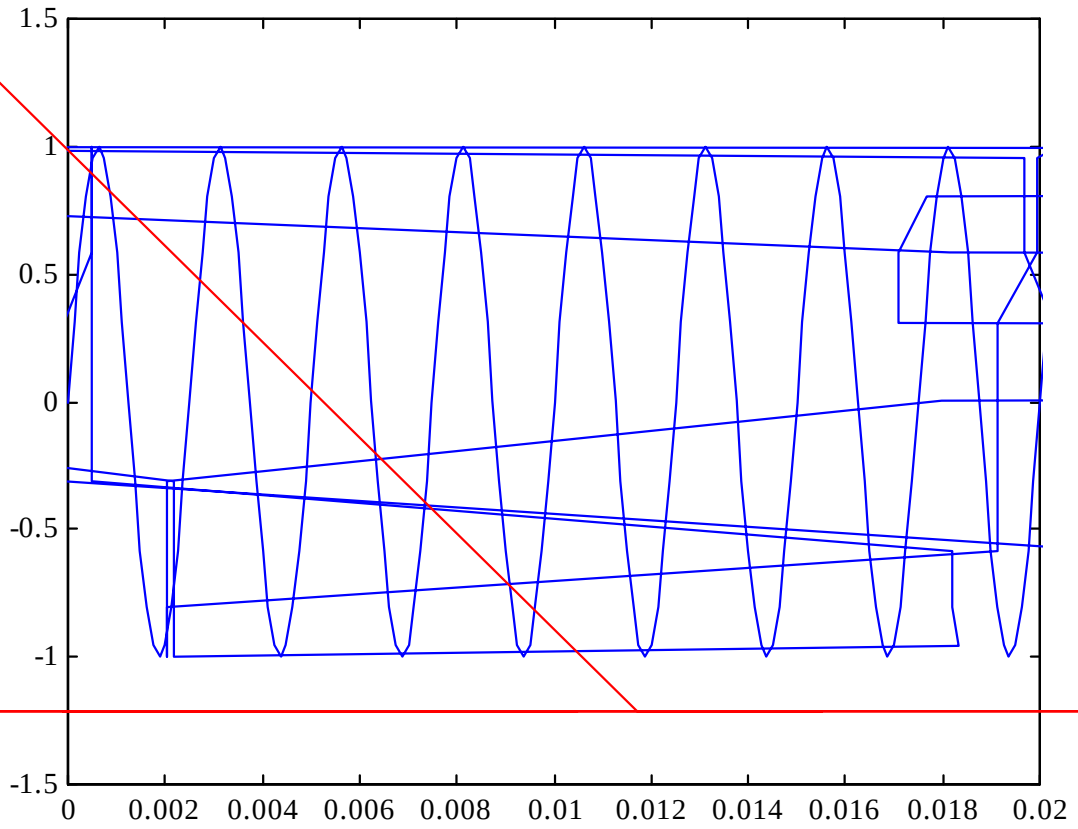
Example 1-4 (vis1.m)

Basic Visualization

`grid`

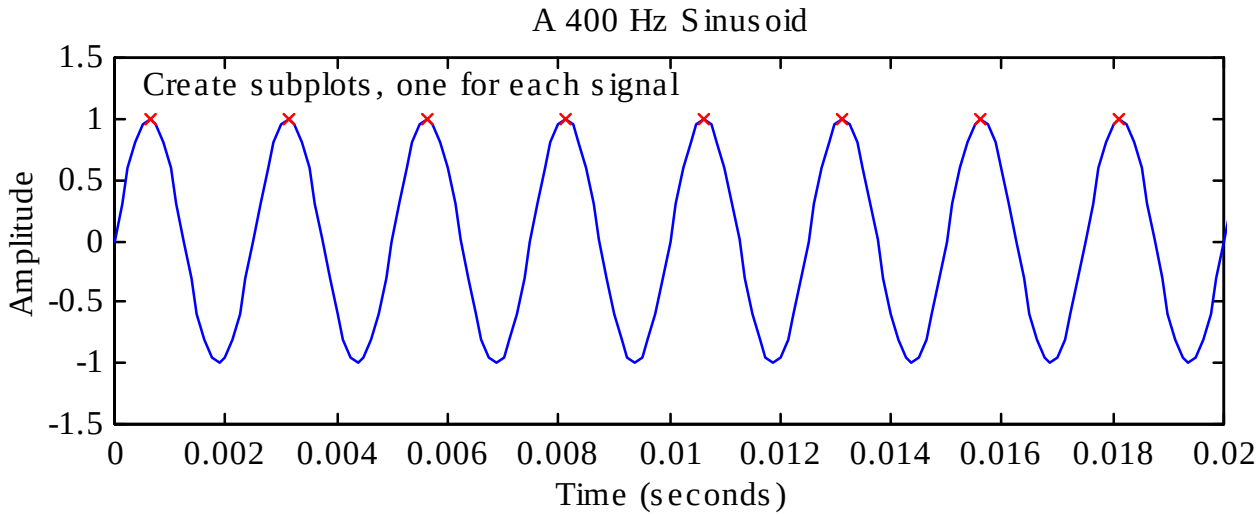


Example 1-4 (vis1.m)

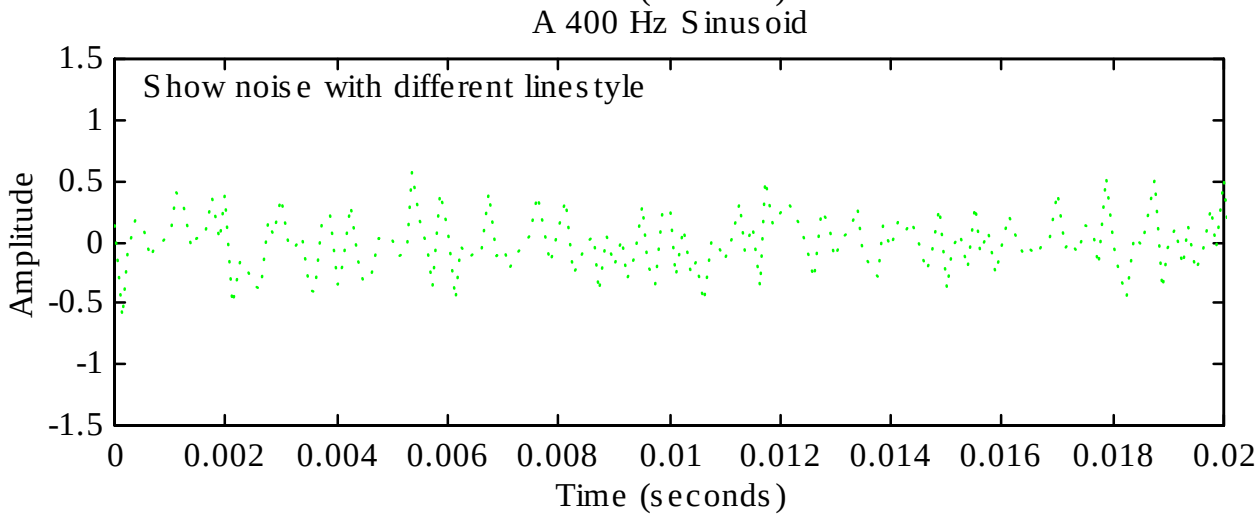


Basic Visualization

subplot(211)



subplot(212)



Example 1-4 (vis1.m)

Basic Visualization

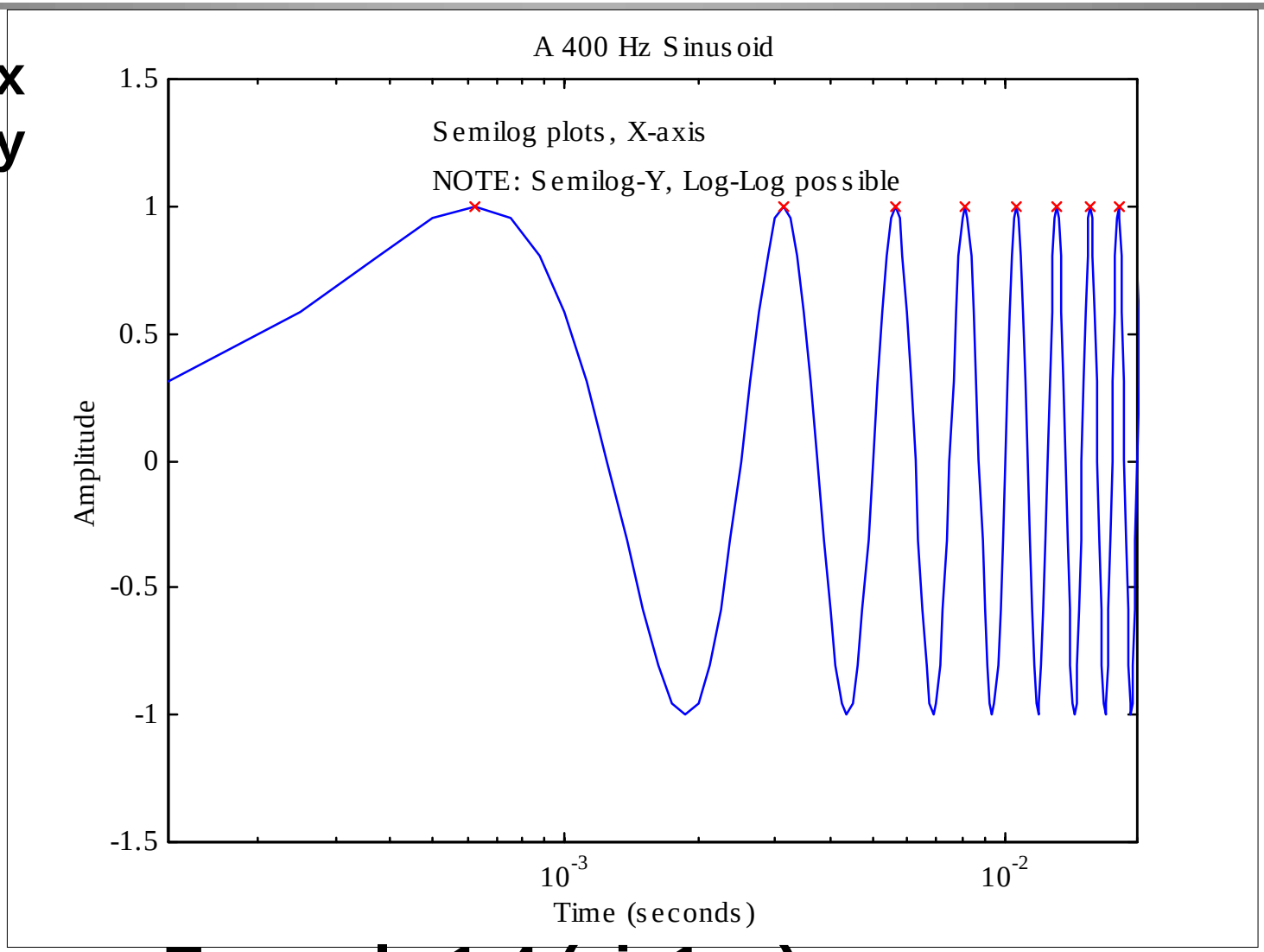
gtext
semilogx
semilogy
loglog

```
% Place text using the mouse
gtext('Like this');

% Semilog-x, Semilog-y, Log-Log
semilogx(t,sig,tp,sig(i),'rx');
```

Example 1-4 (vis1.m)

Basic Visualization

semilogx
semilogy
loglog



Example 1-4 (vis1.m)

Basic Visualization

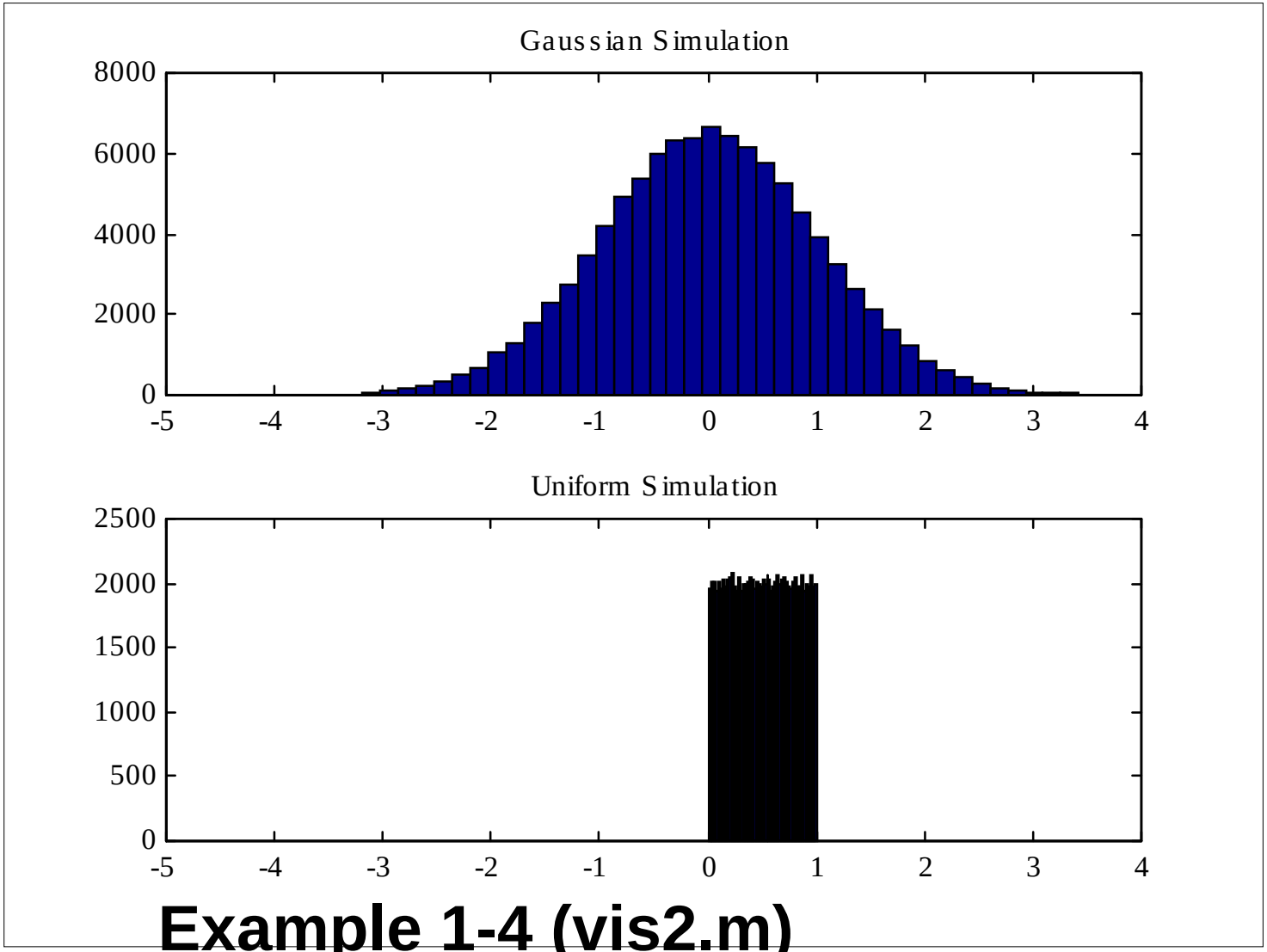
hist

```
function vis2(n,m)

% n - number of samples in the random vectors
% m - number of histogram bins

Xn=randn(n,1);
Xu=rand(n,1);
subplot(211)
hist(Xn,m);
title('Gaussian Simulation');
xlim=get(gca,'xlim');
subplot(212)
hist(Xu,m);
set(gca,'xlim',xlim);
title('Uniform Simulation');
```

Example 1-4 (vis2.m)

Basic Visualization

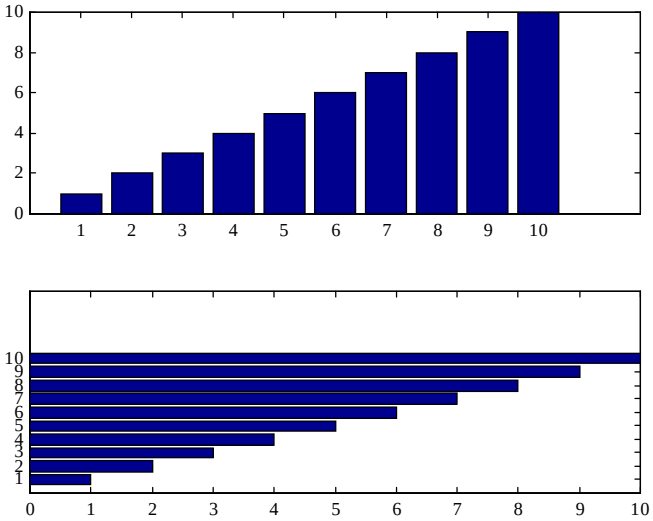
`hist`



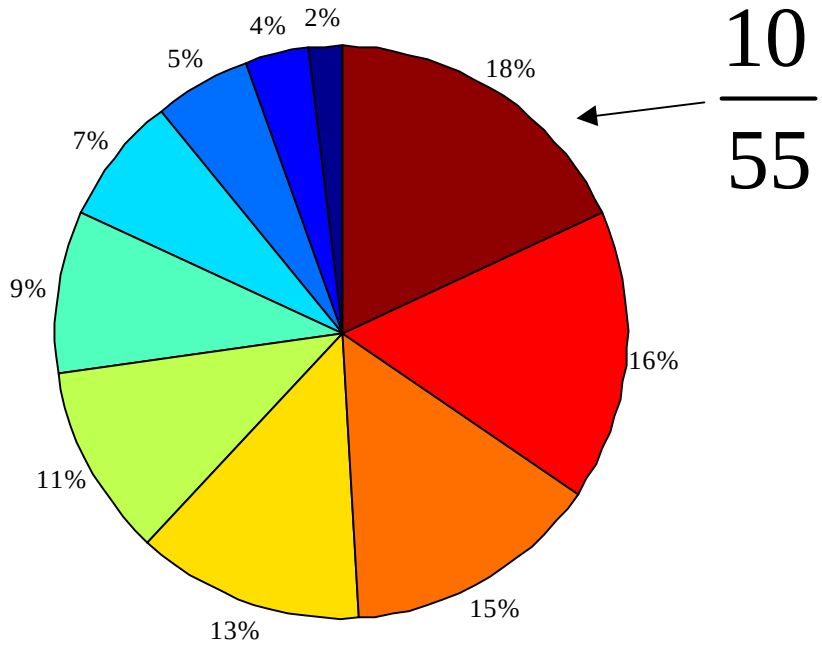
Example 1-4 (vis2.m)

Basic Visualization

bar
barh
pie




```
x=1:10;
subplot(211)
bar(x)
subplot(212)
barh(x)
```

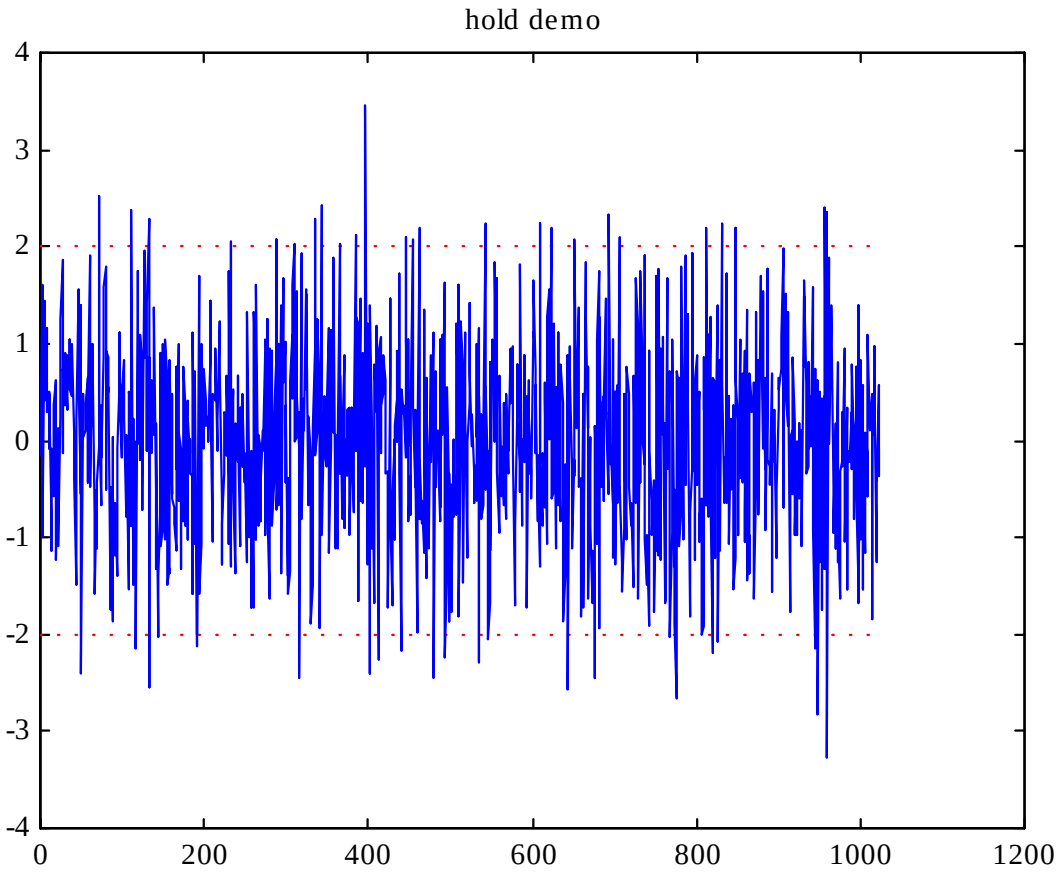



```
x=1:10;
pie(x)
```

Basic Visualization

hold

```
N=1024;
n=1:N;
o=ones(N,1);
x=randn(N,1);
sigma=std(x);
plot(n,2*sigma*o,'r:',
     n,-2*sigma*o,
     'r:');
hold on
plot(n,x)
```



Example 1-4 (vis3.m)

Basic Visualization

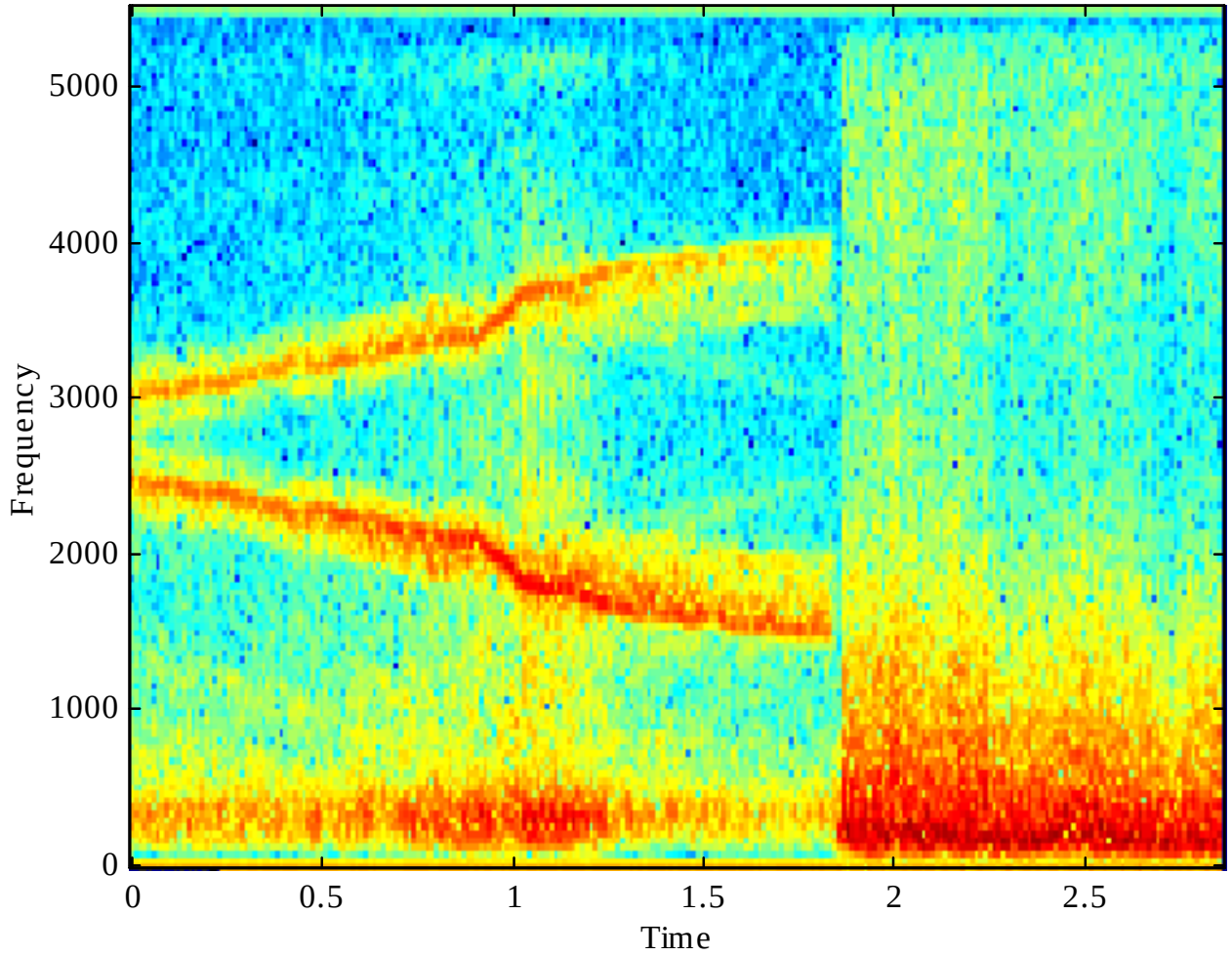
specgram

```
function sgdemo1

[s,fs,bits]=wavread('bomb');
specgram(s,[],fs);
pause
sound(s,fs);
pause

[s,fs,bits]=wavread('thx');
specgram(s(:,1),[],fs);
pause
sound(s,fs);
```

Example 1-5 (sgdemo1.m)

Basic Visualization

`specgram`



Example 1-5 (sgdemo1.m)

Basic Visualization

specgram

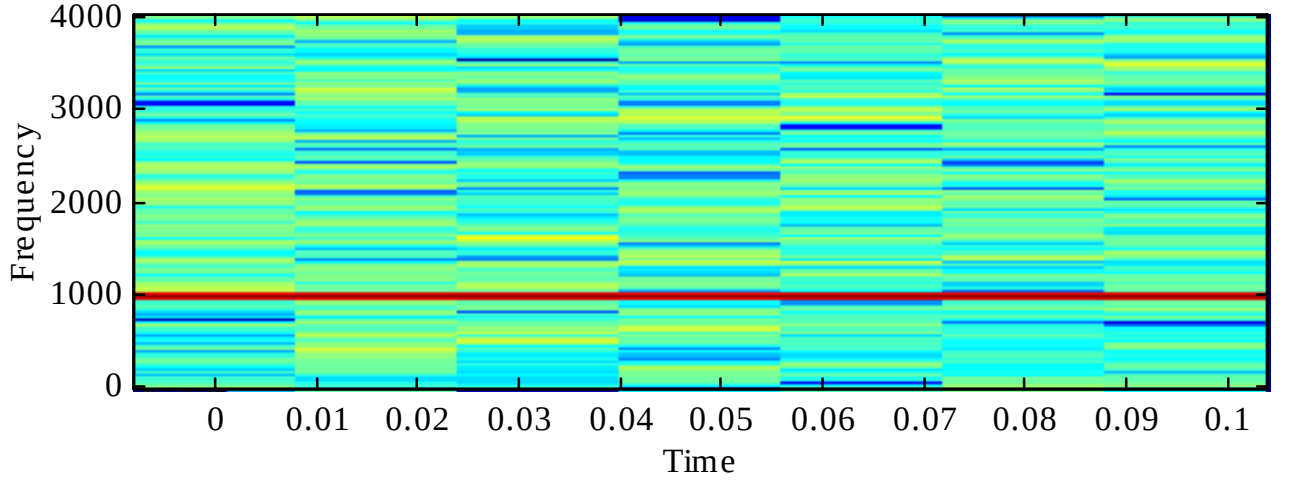
```
function sgdemo2(f,fs,N)

% Defaults
switch nargin
case 0
   f=1000;
   fs=8000;
   N=1024;
case 1
   fs=8000;
   N=1024;
case 2
   N=1024
end
% Force column vector of frequencies
f=f(:);
% Demo 1:  Spectogram for sinusoidal ensemble
K=length(f);
s=zeros(N,1);
n=(0:(N-1))';
for k=1:K
   s=s+sin(2*pi*f(k)/fs*n);
end
% Normalize
s=s./max(abs(s));
```

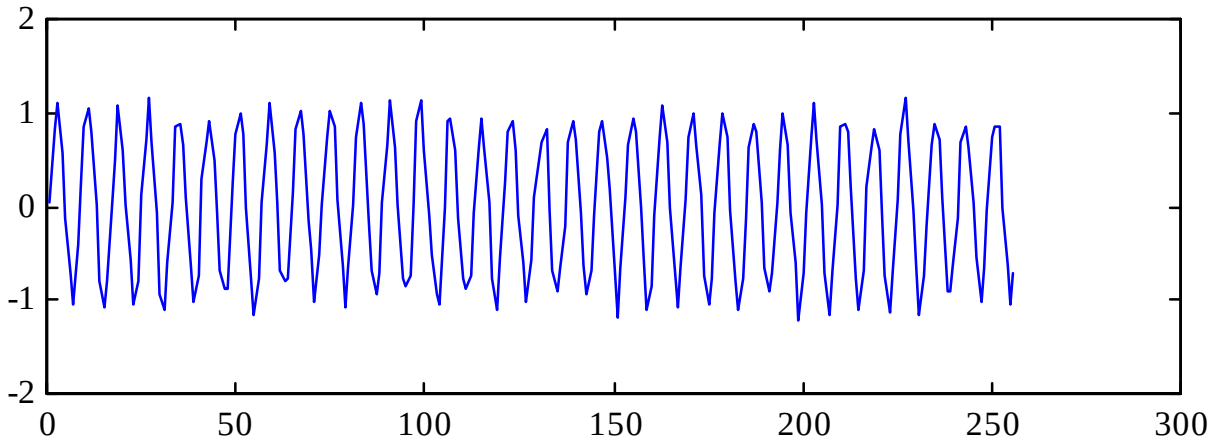
```
clc
fprintf('Spectogram contains …
         %d pure sinusoids.\n\n',K);
% Display spectogram
specgram(s,[],fs);
pause
fprintf('Now add white noise...\n');
% Add incrementally more powerful  noise
G=0.0001;
close
for k=1:20
   sn=s+G*randn(N,1);
   noise=s-sn;
   SNR=10*log10((s'*s)/(noise'*noise));
   fprintf('SNR = %2.1f\n',SNR);
   subplot(211)
   specgram(sn,[],fs);
   subplot(212)
   plot(sn(1:256));
   pause
   G=G*2;
end
close
```

Example 1-5 (sgdemo2.m)

Basic Visualization

specgram





Example 1-5 (sgdemo2.m)

Basic Visualization

`zoom on`
`zoom off`
`zoom(factor)`
	Left mouse button - zoom in
	Right mouse button - zoom out
	Click/drag - zoom in to box
	Double click - return to original

`print`	Generate hardcopy
`print -deps`	encapsulated ps
`print -dpsc`	color ps
`print -dljet4`	laserjet 4
`print file -dps`	to a file, ps
`print -dmeta`	to windows clipboard

Example DSP Application

Debugging,

Profiling

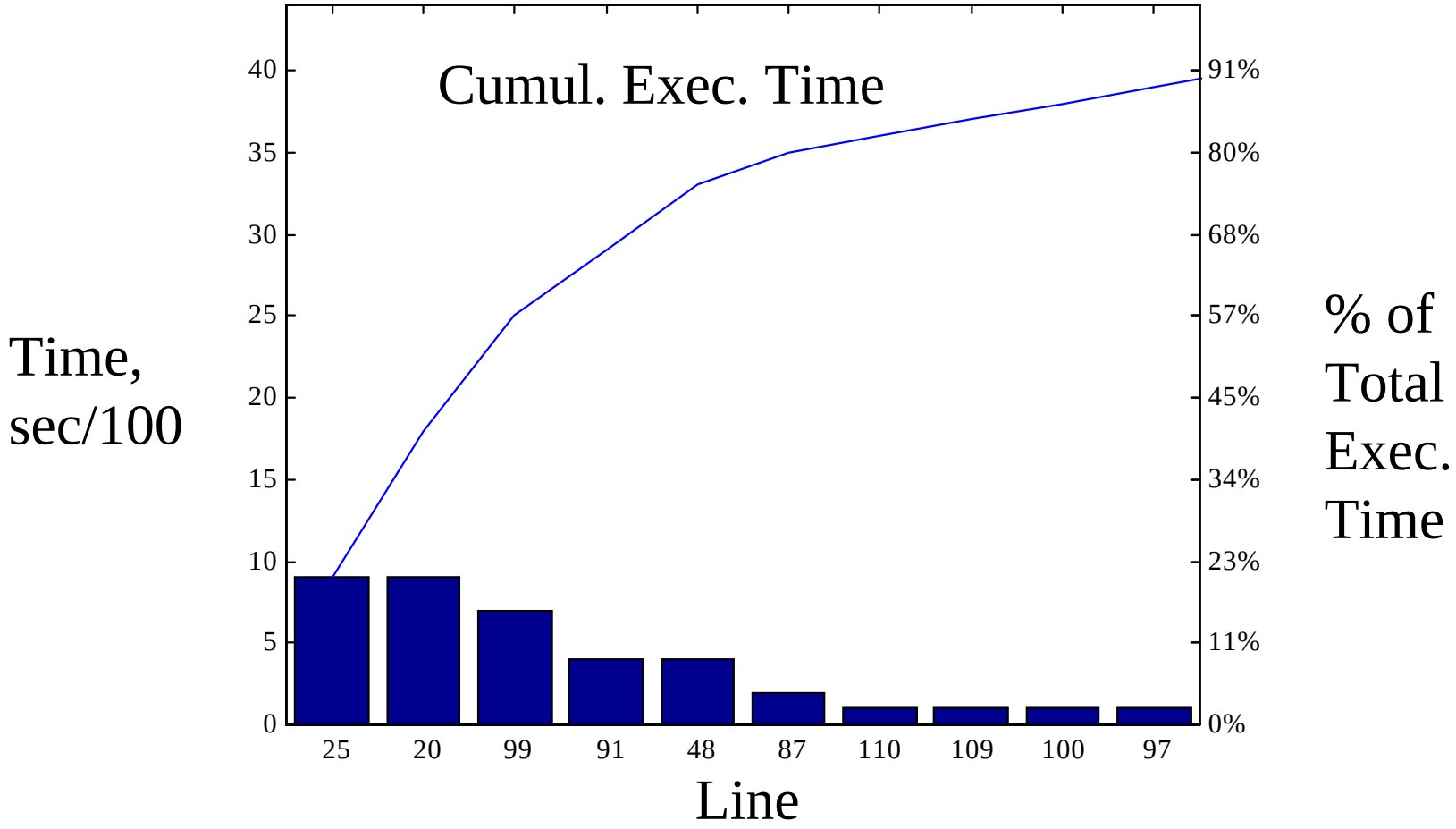
Debugging

`keyboard`	Suspend m-file execution, K>>
`return`	Return to m-file execution
`dbstop`	Set breakpoint
`dbclear`	Clear breakpoint
`dbcont`	Resume execution
`dbstatus`	List all breakpoints
`dbstep`	Execute 1 or more lines
`dbtype`	List m-file with line numbers
`dbquit`	Quit debug mode

Also with MATLAB 5.x, Integrated Debugger

Profiling

Allows detailed analysis of m-file CPU allocation.



Example: erfcore.m, z=erfcore(0:0.01:100);, PII-300

Profiling

The example profile for erfcore.m was generated as follows:

```
>> profile erfcore

>> z = erf(0:0.01:100)

>> profile report

>> t = profile

>> pareto(t.count)
```

This technique can be applied to any m-file.

On-Line

Resources

MATLAB Resources Online

1. ***MATLAB Primer.*** **(K. Sigmon, Univ. of FL)**

   http://www.yale.edu/secf/software/matlab.html

   http://www.stanford.edu/class/ee392g/#ClassMaterial

2. **MATLAB Tutorial Sites:**

   http://www.math.arizona.edu/~erker/matlab/matlab.html

   http://anxiety-closet.mit.edu:8001/afs/athena.mit.edu/software/matlab/www/home.html

   http://www.mathworks.com/

MATLAB Resources Online

3. MATLAB-to-C Translator: MATCOM

http://www.mathtools.com/

4. Toolbox Links Galore

http://www.mathtools.com/mtoolbox.html

Signal Processing - T/F Analysis, Speech and Audio, etc.
Communications
Controls
Numerical methods, Optimization
Neural Networks
Fuzzy Logic
Genetic and Evolutionary Algorithms
User interface and Graphics
MORE!