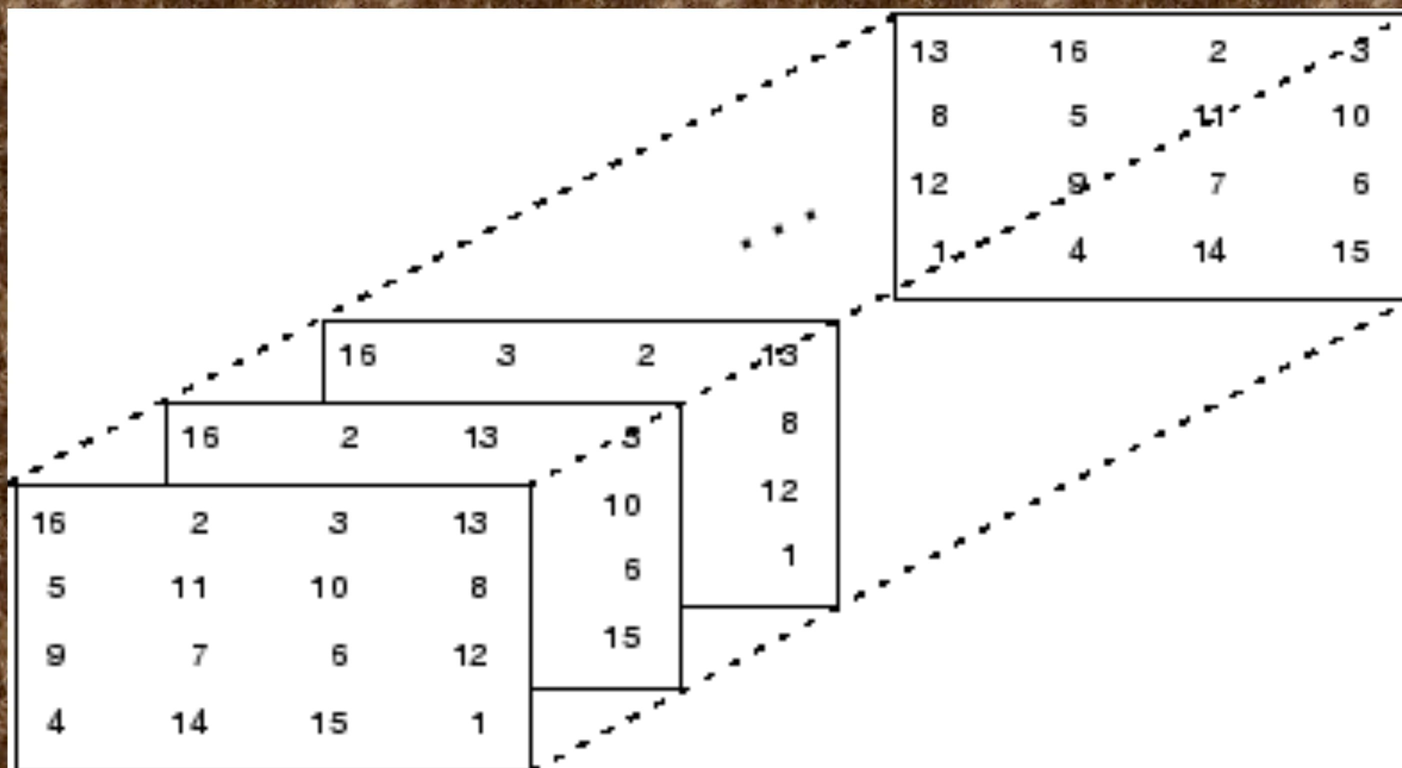
Multidimensional Arrays
Arrays with more than two subscripts

```
>>p = perms(1:4);
>>A = magic(4);
>>M = zeros(4,4,24);
>>for k = 1:24
M(:,:,k) = A(:,p(k,:));
```



Reshape command

Use to change the shape of matrices

```
>> x=[1 2 3 4 5 6 7 8]
x =
     1     2     3     4     5     6     7     8
>> x3d=reshape(x,2,2,2)
x3d(:,:,1) =
     1     3
     2     4
x3d(:,:,2) =
     5     7
     6     8
```

Reshape command

```
>> x=[1 2;3 4;5 6; 7 8]
x =
     1     2
     3     4
     5     6
     7     8
>> x3d=reshape(x,2,2,2)
x3d(:,:,1) =
     1     5
     3     7

x3d(:,:,2) =
     2     6
     4     8
>> x=reshape(x,2,4)
x =
     1     5     2     6
     3     7     4     8
>>
```

Building matrices by repeating parts

repmat command

```
>> x=[1 2;3 4]
x =
     1     2
     3     4
>> xr=repmat(x,2,1)
xr =
     1     2
     3     4
     1     2
     3     4
>> xr=repmat(x,1,2)
xr =
     1     2     1     2
     3     4     3     4
>>
```

Create constant matrix

```
>> val=pi
val =
    3.1416
>> siz=[2 2 2]
siz =
     2     2     2
>> x=repmat(val,siz)
x(:,:,1) =
    3.1416    3.1416
    3.1416    3.1416
x(:,:,2) =
    3.1416    3.1416
    3.1416    3.1416
>>
```

Another way (seems more roundabout)

```
>> xx(prod(siz))=val
xx =
         0         0         0         0         0         0
0    3.1416
>> xx(:)=xx(end)
xx =
    3.1416    3.1416    3.1416    3.1416    3.1416    3.1416
3.1416    3.1416
>> xx=reshape(xx,siz)
xx(:,:,1) =
    3.1416    3.1416
    3.1416    3.1416
xx(:,:,2) =
    3.1416    3.1416
    3.1416    3.1416
```

Another way (m, n and o have to be scalar variables)

```
>> m=2
m =
     2
>> n=2
n =
     2
>> o=2
o =
     2
>> x(m,n,o)=val
x(:,:,1) =
    3.1416    3.1416
    3.1416    3.1416
x(:,:,2) =
    3.1416    3.1416
    3.1416    3.1416
>>
```

Another way (val has to be a scalar variable, this syntax just populates the array with val)

```
>> x=val(ones(siz))
x(:,:,1) =
    3.1416    3.1416
    3.1416    3.1416
x(:,:,2) =
    3.1416    3.1416
    3.1416    3.1416
>>
```

Avoid using

```
X = val * ones(siz);
```

since it does unnecessary multiplications (versus just storing, above) and only works for classes for which the multiplication operator is defined.

Another way

Below does not work (NaN not scalar variable, same with Inf)

```
x = NaN(ones(siz));
```

But following does work

(NaN can be scalar variable or function)

```
>> X = repmat(NaN, siz)
X(:,:,1) =
   NaN   NaN
   NaN   NaN
X(:,:,2) =
   NaN   NaN
   NaN   NaN
```

For lots more of this – see Peter Acklam's tutorial (on class web site)

Flipping vectors, matrices

```
>> a=[1 2;3 4]
a =
     1     2
     3     4
>> fliplr(a)
ans =
     2     1
     4     3
>> flipud(a)
ans =
     3     4
     1     2
>> a=[1 2 3;4 5 6]
a =
     1     2     3
     4     5     6
```

```
>> rot90(a)
ans =
     3     6
     2     5
     1     4
>> a'
ans =
     1     4
     2     5
     3     6
>> flipdim(a,1)
ans =
     4     5     6
     1     2     3
```

How to represent "nothing"

Array = []
String = ''

Useful for defining a name to be used on LHS.

Size and length are zero.

Beyond simple array variables

Structures are variables that contain other variables. It is a way to organize data.

The different fields of a structure, can contain variables of different types, so if one gives the fields a meaningful name this becomes a great way to keep track of the data.

In MATLAB one can define a structure (as any other variable) as one goes.

Structures

Like nawk, Matlab allows you create structures so that you may refer to elements of an array using textual *field designators*

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+'
```

creates a scalar structure with three fields:

```
S =
name: 'Ed Plum'
score: 83
grade: 'B+'
```

Fields can be added one at a time

(a vector of the structure elements)

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

Or entire element added in single statement

```
S(3) = struct('name','Jerry Garcia',…
'score',70,'grade','C')
S =
1x3 struct array with fields:
Name
Score
Grade
>>scores = [S.score]
scores =
83 91 70
>>avg_score = sum(scores)/length(scores)
avg_score =
81.3333
```

Unfortunately structures don't behave as one might expect (hope?)

The following does not work.

```
>>avg_score = sum(S.score)/length(S.score)
```

You have to pull the vector you want to process out of the structure to use it (and make it a vector with the []).

```
>>scores = [S.score]
scores =
83  91  70
>>avg_score = sum(scores)/length(scores)
avg_score =
81.3333
```

Example of structure and its use.

```
image.data=[1 2 3; 4 5 6; 7 8 5];
image.date='13-Jan-2008';
image.blank=NaN;
image.ra=13.3212;
image.dec=43.3455;
```

Address element of structure using structure name, decimal point, and element name.

```
image.date
```

Operate on the fields as you would with any variable of that particular type. Ex., to invert the data matrix

```
inv(image.data).
```

Example for earthquake data

```
stn.name='mem';
stn.lat=34.5'
stn.lon=-89.5
stn.elev=70;
stn.inst='guralp cmg3'
stn.p=15.673
```

Pass structure by name of structure

```
some_fun(stn)
```

etc.

array of structures (and structure elements can be arrays).

Can be multidimensional.

```
stn(1).name='mem';
stn(1).lat=34.5'
stn(1).lon=-89.5
stn(1).elev=70;
stn(1).inst='guralp cmg3'
stn(1).arrival(1)=15.673
stn(1).arrival(2)=17.274
stn(2).name='ceri';
stn(2).lat=34.53'
stn(2).lon=-89.57
stn(2).elev=79;
stn(2).inst='guralp cmg3'
stn(2).arrival(1)=16.189
stn(2).arrival(2)=19.923
. . .
```

Create constant matrix Ex with non-numeric data.

```
>> s.x=1
s =
    x: 1
>> s.n='ceri'
s =
    x: 1
    n: 'ceri'
>> x=s(ones(siz))
x =
2x2x2 struct array with fields:
    x
    n
>> x
x =
2x2x2 struct array with fields:
    x
    n
```

```
>> x.x
ans =
     1
1
. . .  7 more times . . .
>> x.n
ans =
ceri
. . . 7 more times . . .
>> x(2,2,2)
ans =
    x: 1
    n: 'ceri'
```

Cell Arrays

multidimensional arrays whose elements are copies of other arrays.

cell arrays are created by enclosing a miscellaneous collection of things in curly braces, {}.

The curly braces are also used with subscripts to access the contents of various cells.

```
>>C = {A    sum(A)    prod(prod(A)) }
[4x4 double] [1x4 double] [20922789888000]
```

to retrieve a cell from a cell array

```
C{1}  -> A, the magic square
C{2} -> row vector of the sum of the columns of A
C{3} -> 16
```

Important distinction with respect to other programming languages –

cell arrays contain <u>copies</u> of other arrays, not <u>pointers</u> to those arrays.

Cell Arrays vs Multidimensional Arrays

You can use three-dimensional arrays to store a sequence of matrices of the *same size.*

Cell arrays can be used to store a sequence of matrices of *different sizes*.

Characters and Text

Matlab treats text like a character vector

Enter text into MATLAB using single quotes.

```
>>s = 'Hello'
```

essentially, s is now a 1 x 5 array with each element equal to a character: H,e,l,l,o

Characters are stored as numbers using ASCII coding with the type *char*

```
a = double(s)
a =
72   101   108    108     111
```

Because characters are stored as numbers, you can convert numeric vectors to their ASCII characters, if the character exists

```
s=char(a)
```

Printable ASCII characters go from 32 to 127

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

To manipulate a body of text with lines of different lengths, you have two choices

- a padded character array
- a cell array of strings.

When creating a character array, each row of the array must be the same length. The char function pads with spaces to create equal rows

```
S = char('A','rolling','stone','gathers','momentum.')
```

produces a 5-by-9 character array:

```
S =
A
rolling
stone
gathers
momentum.
```

You don't have to worry about this with a cell array

```
C = {'A';'rolling';'stone';'gathers';'momentum.'}
```

You can convert a padded character array to a cell array of strings with

```
C = cellstr(S)
```

and reverse the process with

```
S = char(C)
```

To create a character array from one of the text fields in a structure (name, for example), call the char function on the comma-separated list produced by S.name:

```
>>names = char(S.name)
names =
Ed Plum
Toni Miller
Jerry Garcia
```

Checking for special elements (NaN, Inf)

isnan(a) Returns 1 for every NaN in array a.

isinf(a) Returns 1 for every Inf in array a.

isfinite(a) Returns 1 for every finite number (not a (Nan or Inf)) in array a.

isreal(a) Returns 1 for every non-complex number array a.

Using special elements to your advantage.

Since NaNs propagate through calculations (answer is NaN if there is a NaN somewhere in the calculation), it is sometimes useful to throw NaNs out of operations like taking the mean.

(A handy trick to ignore stuff you don't want while you continue calculating.)

So the function that identifies NaNs can be very useful:

```
ix=find(~isnan(a));
m=mean(a(ix));
```

this finds all values that are not NaNs and averages them.

help

Built into matlab

help “command”

To get help on the command “command”

Problem when you don't know the name of the command

Just type "help"

```
>> help
HELP topics:

Documents/MATLAB   - (No table of contents file)
matlab/general     - General purpose commands.
matlab/ops         - Operators and special characters.
matlab/lang        - Programming language constructs.
matlab/elmat       - Elementary matrices and matrix
                                           manipulation.
matlab/randfun     - Random matrices and random streams.
```

Lists topics of help available

Then to get contents of topics type help "topic"

```
>> help elmat
  Elementary matrices and matrix manipulation.

  Elementary matrices.
    zeros       - Zeros array.
    ones        - Ones array.
    eye         - Identity matrix.
    repmat      - Replicate and tile array.
    linspace    - Linearly spaced vector.
    logspace    - Logarithmically spaced vector.
    freqspace   - Frequency spacing for frequency response.
    meshgrid    - X and Y arrays for 3-D plots.
    accumarray  - Construct an array with accumulation.
    :           - Regularly spaced vector and index into
matrix.

  Basic array information.
    size        - Size of array.
```

Help on individual command

```
>> help zeros
 ZEROS   Zeros array.
    ZEROS(N) is an N-by-N matrix of zeros.
    ZEROS(M,N) or ZEROS([M,N]) is an M-by-N matrix of zeros.
    ZEROS(M,N,P,...) or ZEROS([M N P ...]) is an M-by-N-by-P-
by-... array of
    zeros.
    ZEROS(SIZE(A)) is the same size as A and all zeros.
    ZEROS with no arguments is the scalar 0.
    ZEROS(M,N,...,CLASSNAME) or ZEROS([M,N,...],CLASSNAME) is
an
    M-by-N-by-... array of zeros of class CLASSNAME.
    Note: The size inputs M, N, and P... should be nonnegative
integers.
    Negative integers are treated as 0.
    Example:
       x = zeros(2,3,'int8');
    See also eye, ones.
    Reference page in Help browser
       doc zeros
```

Some unix commands (pwd, ls, ???) "work" in matlab (they are actually matlab commands)

```
a=pwd;
b=ls;
```

Some matlab commands have the same names as unix commands, but are not the same

"cat" is a matlab command that concatenates matrices (not files)

Matlab does not pass things it does not understand to the OS to see if they are OS commands.

MATLAB vectorized high level language

Requires change in programming style
(if one already knows a non-vectorized programming language such as Fortran, C, Pascal, Basic, etc.)

Vectorized languages allow operations over arrays using simple syntax, essentially the same syntax one would use to operate over scalars.
(looks like math again.)

What is vectorization?
(with respect to matlab)

Vectorization is the process of writing code for MATLAB that uses matrix operations or other fast builtin functions instead of using for loops.
The benefits of doing this are usually sizeable.
The reason for this is that MATLAB is an interpreted language. Function calls have very high overhead, and indexing operations (inherent in a loop operation) are not particularly fast.

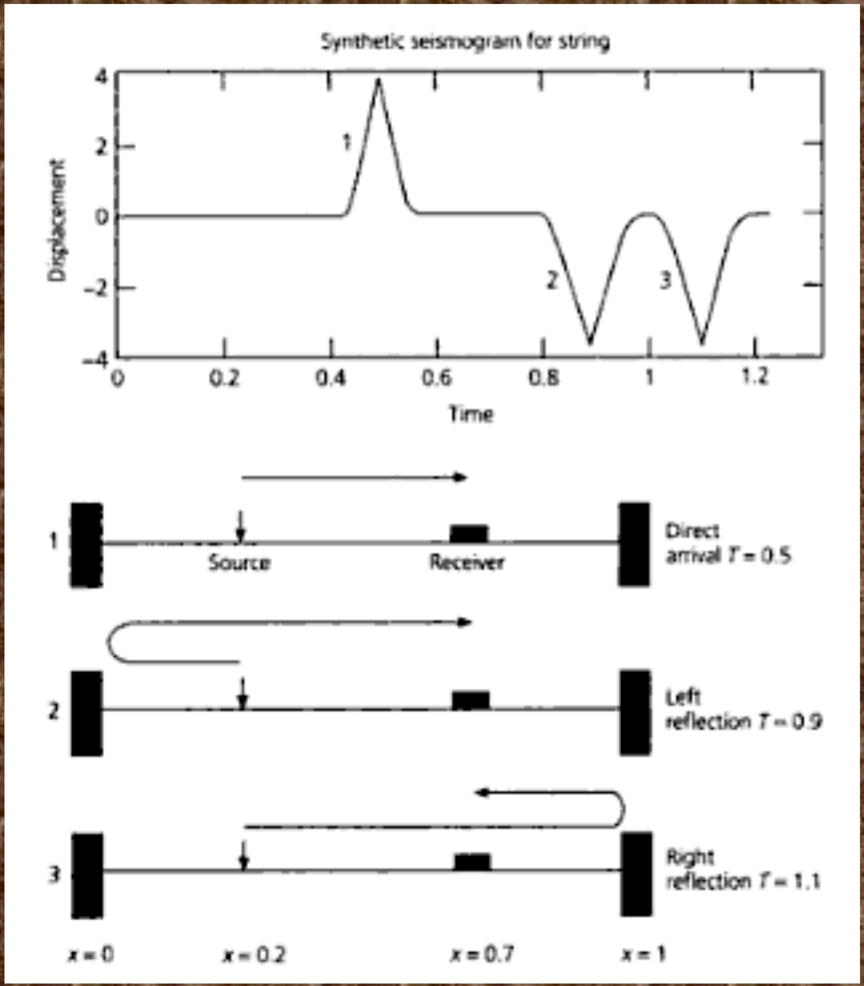
Loop versus vectorized version of same code. New commands "tic" and "toc" - time the execution of the code between them.

```
>> a=rand(1000);
>> tic;b=a*a;toc
Elapsed time is 0.229464 seconds.
>> tic;for k=1:1000,for l=1:1000,c(k,l)=0;for m=1:1000,
c(k,l)=c(k,l)+a(k,m)*a(m,l);end, end, end, toc
Elapsed time is 22.369451 seconds.
>> whos
  Name         Size                Bytes  Class     Attributes
  a         1000x1000            8000000  double
  b         1000x1000            8000000  double
  c         1000x1000            8000000  double
  k            1x1                     8  double
  l            1x1                     8  double
  m            1x1                     8  double

>> max(max(b-c))
ans =
   9.6634e-13
```

Factor 100 difference in time for multiplication of 10^6x10^6 matrix!

Vectorization of synthetic seismogram example from Stein and Wysession, Intro to Seismology and Earth Structure.



$$u(x,t)=\sum_{n=1}^{\infty}\sin(n\pi x/L)\sin(n\pi x_s/L)\cos(\omega_n t)\exp[-(\omega_n \tau/4)]$$

This is just the Fourier transform for a standing wave

$$u(x,t)=\sum_{n=1}^{\infty}\sin\left(n\pi x_s/L\right)\sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)\exp\left[-\left(\omega_n\tau/4\right)\right]$$

$$\left(Note:\ \omega_n = n*\omega_0\right)$$

$$u(x,t)=\sum_{n=1}^{\infty}\left(\sin\left(n\pi x_s/L\right)\exp\left[-\left(\omega_n\tau/4\right)\right]\right)\sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)$$

Wt - no dependence on x or t

$$u(x,t)=\sum_{n=1}^{\infty}a_n\sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)$$

Standing wave from 2 opposite direction traveling waves

$$u(x,t)=\sum_{n=1}^{\infty}a'_n\left[\cos\left(n\pi x/L+\omega_n t\right)+\cos\left(n\pi x/L-\omega_n t\right)\right]$$

Look at the basic element of Fourier series, weighted sum of sin and cos functions

(look at cos only to see how works).

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m)$$

$$u(t_m) = \frac{a_0}{2} + (a_1\ a_2\ a_3\ \cdots\ a_n) \bullet \left(\cos(\omega_1 t_m)\ \cos(\omega_2 t_m)\ \cos(\omega_3 t_m)\ \cdots\ \cos(\omega_n t_m)\right)$$

$$u(t_m) = \frac{a_0}{2} + (a_1\ a_2\ a_3\ \cdots\ a_n) \begin{pmatrix} \cos(\omega_1 t_m) \\ \cos(\omega_2 t_m) \\ \cos(\omega_3 t_m) \\ \cdots \\ \cos(\omega_n t_m) \end{pmatrix} = \frac{a_0}{2} + \left(\cos(\omega_1 t_m)\ \cos(\omega_2 t_m)\ \cos(\omega_3 t_m)\ \cdots\ \cos(\omega_n t_m)\right) \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\,\vec{a}$$

Look at the basic Fourier series

constant time, weighted sum of cosines at different frequencies at that time

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\,\vec{a}$$

constant frequency cosine as function of time (basis functions)

Is multiplication of a matrix (with cosines as functions of frequency - across - and time - down) times a vector with the Fourier series weights.

We have just vectorized the equations!

Even though this is a major improvement over doing this with for loops, and is clear conceptually, it is still not computable as it takes $O(N^2)$ operations (and therefore time) to do it. This is OK for small N, but quickly gets out of hand.

Fourier analysis is typically done using the Fast Fourier transform algorithm – which is $O(N \log N)$.

Fourier decomposition. “Basis” functions are the sine and cosine functions.

Notice that first sine term is all zeros (so don’t really need it) and last sine term (not shown) is same as last cosine term, just shifted one – so will only need one of these).

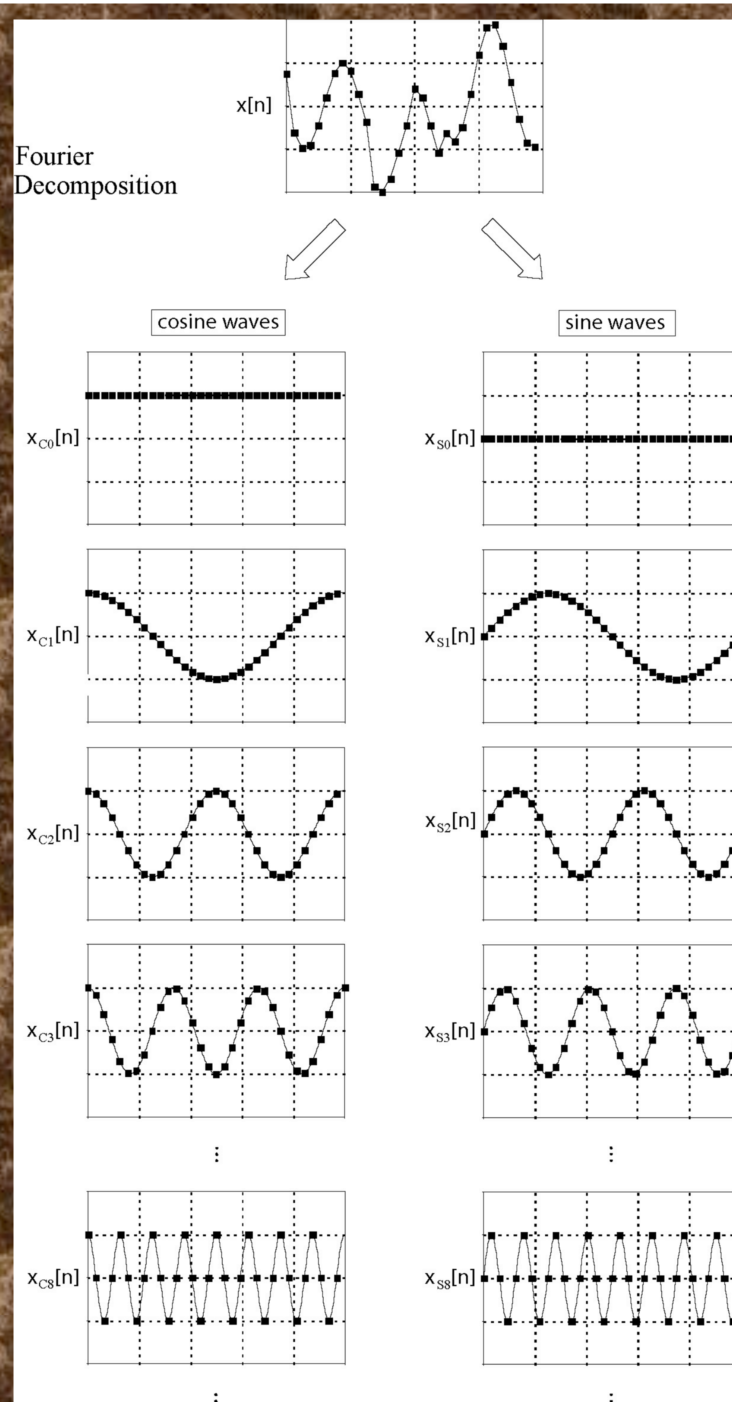


Figure from Smith

Fourier transform

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m) + \sum_{n=1}^{N} b_n \sin(\omega_n t_m)$$

The Fast Fourier Transform (FFT) depends on noticing that there is a lot of repetition in the calculations – each higher frequency basis function can be made by selecting points from the w_0 function. The weight value is multiplied by the same basis function value an increasing number of times as w increases.

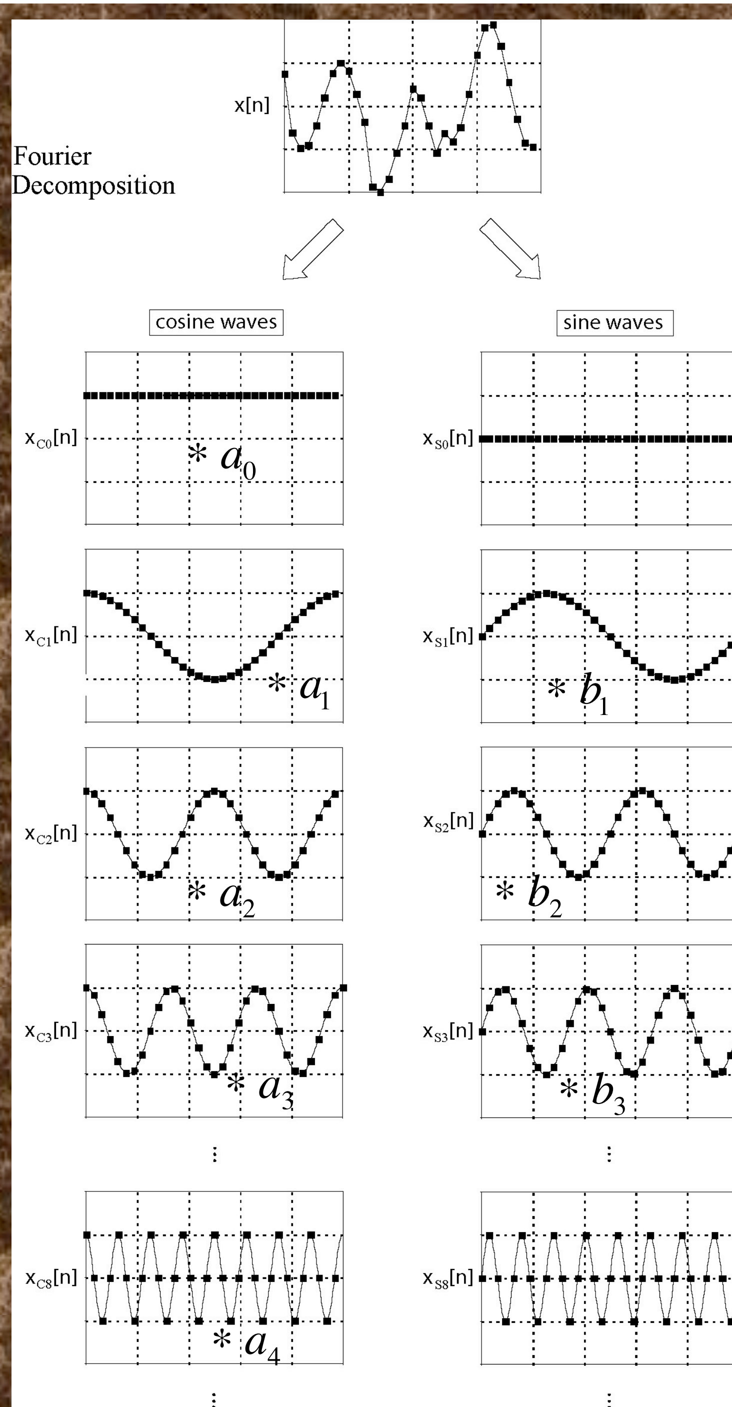


Figure from Smith

FFT

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m) + \sum_{n=1}^{N} b_n \sin(\omega_n t_m)$$

The FFT basically does each unique multiplication only once, stores it, and then does the bookeeping to add them all up correctly.

The points in the trace at the top are made from vertical sums of the weighted points at the same time in the cos and sin traces in the bottom.

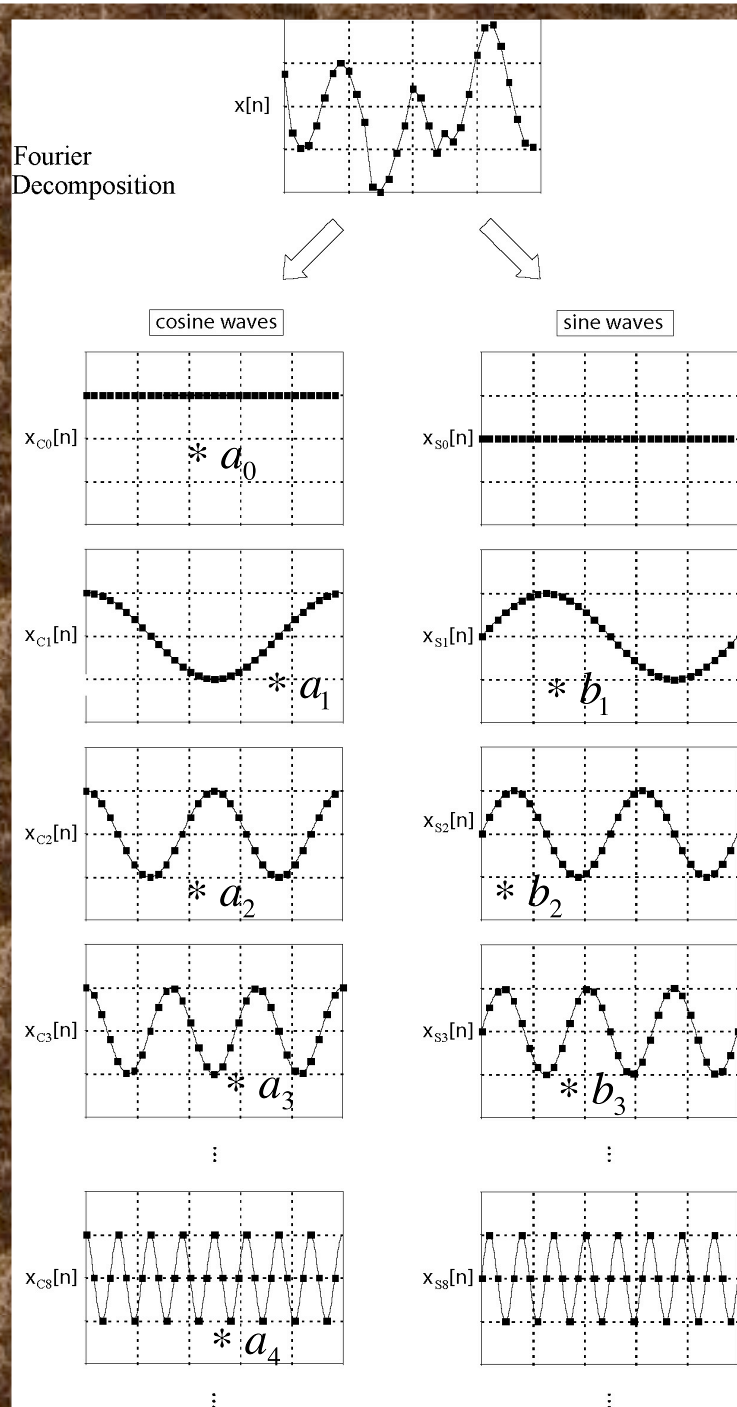


Figure from Smith

```
C SYNTHETIC SEISMOGRAM FOR HOMOGENEOUS STRING
C DISPLACEMENT U AS FUNCTION OF TIME T
C CALCULATED BY NORMAL MODE SUMMATION
      DIMENSION U(200)
      PI = 3.1415927
C
C PARAMETERS (NORMALLY WOULD COME FROM INPUT)
C STRING LENGTH (M)
      ALNGTH = 1.0
C VELOCITY (M/S)
      C = 1.0
C NUMBER OF MODES
      NMODE = 200
C SOURCE POSITION (M)
      XSRC = 0.2
C RECEIVER POSITION (M)
      XRCVR = 0.7
C SEISMOGRAM TIME DURATION (S)
      TDURAT = 1.25
C NUMBER TIME STEPS
      NTSTEP = 50
C TIME STEP (S)
      DT = TDURAT/NTSTEP
C SOURCE SHAPE TERM
      TAU = .02
C
C LIST PARAMETERS
      WRITE (6,3000)
3000  FORMAT('SYNTHETIC SEISMOGRAM FOR STRING')
      WRITE (6,3001) NMODE
3001  FORMAT('NUMBER OF MODES', I6)
      WRITE (6,3002) ALNGTH, C
3002  FORMAT ('LENGTH (M)' F7.3, 'VELOCITY,
     X (M/S)', F7.3)
      WRITE (6,3003) XSRC, XRCVR
3003  FORMAT ('POSITION (M): SOURCE', F7.3,
     X 'RECEIVER', F7.3)
      WRITE (6,3004) TDURAT, NTSTEP
3004  FORMAT ('SEISMOGRAM DURATION (S)', F7.3,
     X I6, 'TIME STEPS')
      WRITE (6,3005) TAU
3005  FORMAT ('SOURCE SHAPE TERM', F7.3)
C
C INITIALIZE DISPLACEMENT
        DO 5 I = 1, NTSTEP
      U(I) = 0.0
5     CONTINUE
C
C OUTER LOOP OVER MODES
      DO 10 N = 1, NMODE
        ANPIAL = N*PI/ALNGTH
C SPACE TERMS: SOURCE AND RECEIVER
        SXS = SIN(ANPIAL*XSRC)
        SXR = SIN(ANPIAL*XRCVR)
C MODE FREQUENCY
        WN = N*PI*C/ALNGTH
C TIME INDEPENDENT TERMS
        DMP = (TAU*WN)**2
        SCALE = EXP(-DMP/4.)
        SPACE = SXS*SXR*SCALE
C
C INNER LOOP OVER TIME STEPS
        DO 15 J = 1, NTSTEP
          T = DT*(J - 1)
          CWT = COS(WN*T)
C COMPUTE DISPLACEMENT
          U(J) = U(J) + CWT*SPACE
15      CONTINUE
10    CONTINUE
C
C OUTPUT SEISMOGRAM FOR LATER PLOTTING
      WRITE (6, 3101)(U(J), J = 1, NTSTEP)
3101  FORMAT (7F10.4)
      STOP
      END
```

Traditional programming with nested loops.

Related to the details of the math (as if you were doing it by hand).

```
%synthetic seismogram for homogeneous string, u(t)
%calculated by normal mode summation
%string length
alngth=1;
%velocity m/sec
c=1.0;
%number modes
nmode=200;
%source position
xsrc=0.2;
%receiver position
xrcvr=0.7;
%seismogram time duration
tdurat=1.25;
%number time steps
nstep=50;
%time step
dt=tdurat/nstep;
%source shape term
tau=0.02;
fprintf('%s\n','synthetic seismogram for string')
fprintf('%s %0.5g\n','number modes', nmode)
fprintf('%s %0.5g %0.5g\n','length and velocity', alngth, c)
fprintf('%s %0.5g %0.5g\n','posn src and rcvr',xsrc,xrcvr)
fprintf('%s %0.5g %0.5g %0.5g\n','durn, time steps, del
t',tdurat,nstep,dt)
fprintf('%s %0.5g\n','source shape', tau)
%initialize displacement
for cnt=1:nstep
    u(cnt)=0;
end
for k=1:nstep
    t(k)=dt*(k-1);
end
%outer loop over modes
for n=1:nmode
    anpial=n*pi/alngth;
%space terms - src & rcvr
    sxs=sin(anpial*xsrc);
    sxr=sin(anpial*xrcvr);
%mode freq
    wn=n*pi*c/alngth;
%time indep terms
    dmp=(tau*wn)^2;
    scale=exp(-dmp/4);
    space=sxs*sxr*scale;
%inner loop oner time  steps
    for k=1:nstep
%        t=dt*(k-1);
%        cwt=cos(wn*t);
        cwt=cos(wn*t(k));
%compute disp
        u(k)=u(k)+cwt*space;
    end
end
plot(t,u)
```

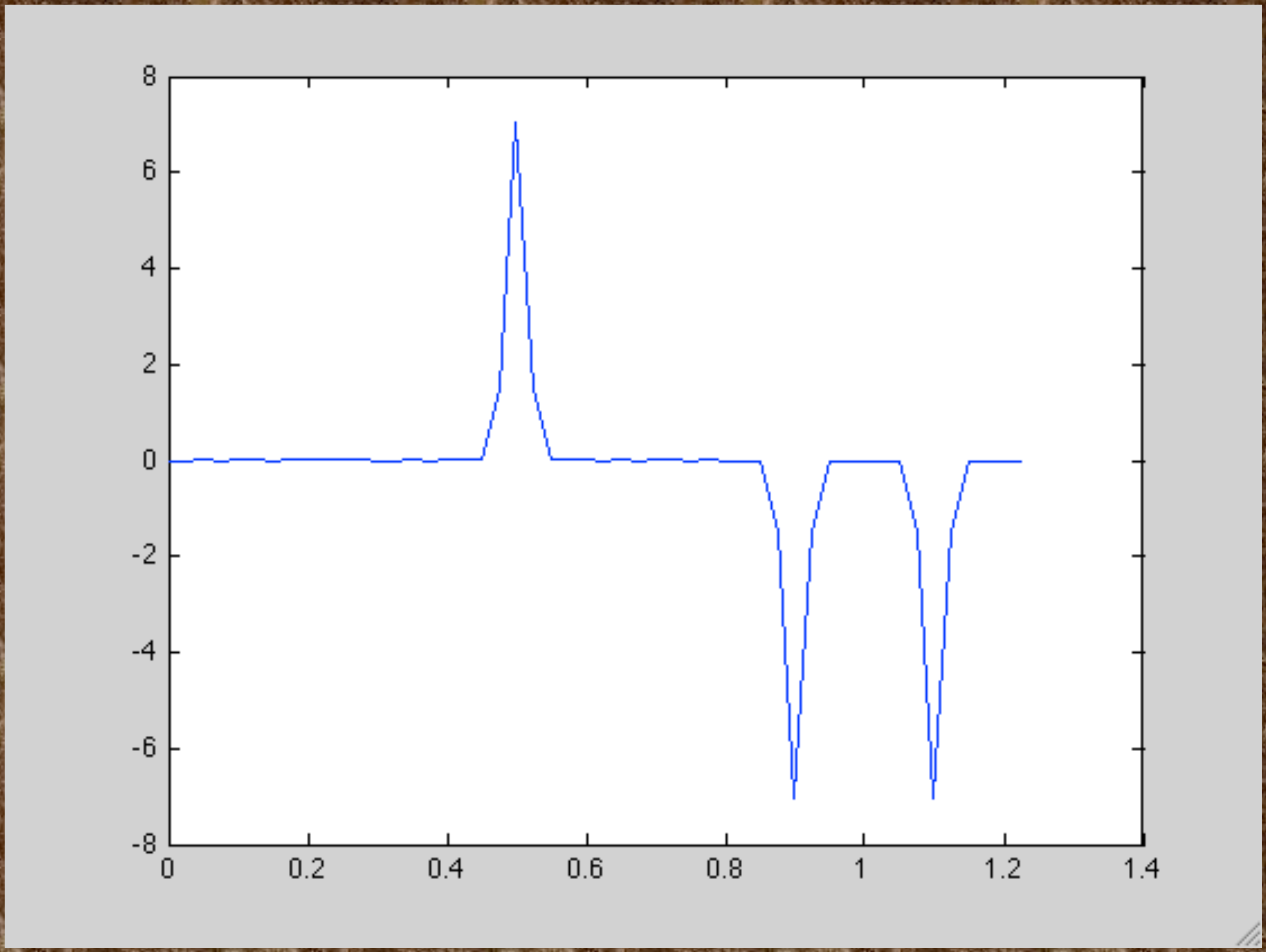
```
%synthetic seismogram for homogeneous
string, u(t)
%calculated by normal mode summation
%string length
alngth=1;
%velocity m/sec
c=1.0;
%number modes
nmode=200;
%source position
xsrc=0.2;
%receiver position
xrcvr=0.7;
%seismogram time duration
tdurat=1.25;
%number time steps
nstep=50;
%time step
dt=tdurat/nstep;
%source shape term
tau=0.02;
fprintf('%s\n','synthetic seismogram for
string')
fprintf('%s %0.5g\n','number modes',
nmode)
fprintf('%s %0.5g %0.5g\n','length and
velocity', alngth, c)
fprintf('%s %0.5g %0.5g\n','posn src and
rcvr',xsrc,xrcvr)
fprintf('%s %0.5g %0.5g %0.5g\n','durn,
time steps, del t',tdurat,nstep,dt)
fprintf('%s %0.5g\n','source shape',
tau)
%initialize displacement
for cnt=1:nstep
    u(cnt)=0;
end
for k=1:nstep
    t(k)=dt*(k-1);
end
%outer loop over modes
for n=1:nmode
    anpial=n*pi/alngth;
%space terms - src & rcvr
    sxs=sin(anpial*xsrc);
    sxr=sin(anpial*xrcvr);
%mode freq
    wn=n*pi*c/alngth;
%time indep terms
    dmp=(tau*wn)^2;
    scale=exp(-dmp/4);
    space=sxs*sxr*scale;
%inner loop oner time  steps
    for k=1:nstep
%        t=dt*(k-1);
%        cwt=cos(wn*t);
        cwt=cos(wn*t(k));
%compute disp
        u(k)=u(k)+cwt*space;
    end
end
plot(t,u)
```

Slightly cleaned up version of Fortran program in Stein and Wysession “translated” to Matlab.

```
>> whos
  Name         Size                Bytes  Class     Attributes

  alngth       1x1                     8  double
  anpial       1x1                     8  double
  c            1x1                     8  double
  cnt          1x1                     8  double
  cwt          1x1                     8  double
  dmp          1x1                     8  double
  dt           1x1                     8  double
  k            1x1                     8  double
  n            1x1                     8  double
  nmode        1x1                     8  double
  nstep        1x1                     8  double
  scale        1x1                     8  double
  space        1x1                     8  double
  sxr          1x1                     8  double
  sxs          1x1                     8  double
  t            1x1                     8  double
  tau          1x1                     8  double
  tdurat       1x1                     8  double
  u            1x50                  400  double
  wn           1x1                     8  double
  xrcvr        1x1                     8  double
  xsrc         1x1                     8  double
```

Synthetic seismogram produced by Matlab code on previous slide.



```
% number of time samples M
% points
% source position xs (meters)
% speed c (meters/sec)
% length L (meters)
% number of modes N
% source pulse duration Tau
% (sec)
% length of seismogram T (sec)

M=50;
xs=0.25;
c=1;
L=1;
N=200;
Tau=0.02;
T=1.25;

%time vector, 1 row by M
% columns
%start, step, stop
dt=T/M;
t=0:dt:T-dt;

% receiver posn
```

Same problem in Matlab after vectorization (is mostly comments!)

```
xr = 0.7;

%stein actually starts at mode
% 1
%freq vector from 0 to n*pi*c/L
%, 1 row by N columns
wn=linspace(1,N,N);
wn=wn*pi*c/L;

%time independent terms - modes
%- 1xN vector (row vector)
timeindep=sin(wn*xr).*sin(wn*xs
).*exp(-(wn*Tau).^2/4);

%time dependent terms -
%time*freqs = MxN matrix
timedep=cos(t'*wn);

%use matrix * vector multiply
%to do "loop"
%(MxN)times(Nx1)=(Mx1) (column
%vector)
seism=timedep*timeindep';

plot(t,seism)
```

Get same figure as before.