MATLAB Examples: Linear Block Codes

Henry D. Pfister
ECE Department
Texas A&M University

October 6th, 2006 (rev. 0)
November 6th, 2013 (rev. 1)

1 The Galois Field $\mathbb{F}_p$ for Prime $p$

Currently this document just gives snippets of example code which should help you get started. If you need help, keep the “help” and “lookfor” commands in mind.

1.1 A Few Commands

>> help eye

EYE Identity matrix.
EYE(N) is the N-by-N identity matrix.

EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1’s on the diagonal and zeros elsewhere.

EYE(SIZE(A)) is the same size as A.

EYE with no arguments is the scalar 1.

EYE(M,N,CLASSNAME) or EYE([M,N],CLASSNAME) is an M-by-N matrix with 1’s of class CLASSNAME on the diagonal and zeros elsewhere.

Example:

x = eye(2,3,’int8’);

See also SPEYE, ONES, ZEROS, RAND, RANDN.

>> help mod

MOD Modulus after division.
MOD(x,y) is x - n.*y where n = floor(x./y) if y ~= 0. If y is not an integer and the quotient x./y is within roundoff error of an integer, then n is that integer. The inputs x and y must be real arrays of the same size, or real scalars.

The statement "x and y are congruent mod m" means mod(x,m) == mod(y,m).

By convention:

MOD(x,0) is x.
MOD(x,x) is 0.

MOD(x,y), for x~y and y~0, has the same sign as y.

Note: REM(x,y), for x~y and y~0, has the same sign as x.
MOD(x,y) and REM(x,y) are equal if x and y have the same sign, but differ by y if x and y have different signs.
Overloaded functions or methods (ones with the same name in other directories)
help sym/mod.m

>> help de2bi
DE2BI Convert decimal numbers to binary numbers.
B = DE2BI(D) converts a nonnegative integer decimal vector D to a binary
matrix B. Each row of the binary matrix B corresponds to one element of D.
The default orientation of the of the binary output is Right-MSB; the first
element in B represents the lowest bit.

In addition to the vector input, three optional parameters can be given:
B = DE2BI(...,N) uses N to define how many digits (columns) are output.
B = DE2BI(...,N,P) uses P to define which base to convert the decimal
elements to.
B = DE2BI(...,FLAG) uses FLAG to determine the output orientation. FLAG
has two possible values, 'right-msb' and 'left-msb'. Giving a 'right-msb'
FLAG does not change the function's default behavior. Giving a 'left-msb'
FLAG flips the output orientation to display the MSB to the left.

Examples:
D = [12; 5];
B = de2bi(D) B = de2bi(D,5)
B =
0 0 1 1
1 0 1 0
T = de2bi(D,[],3) B = de2bi(D,5,'left-msb')
T =
0 1 1
2 1 0

See also BI2DE.

>> help dec2base
DEC2BASE Convert decimal integer to base B string.
DEC2BASE(D,B) returns the representation of D as a string in
base B. D must be a non-negative integer array smaller than 2^52
and B must be an integer between 2 and 36.

DEC2BASE(D,B,N) produces a representation with at least N digits.

Examples
dec2base(23,3) returns '212'
dec2base(23,3,5) returns '00212'

See also BASE2DEC, DEC2HEX, DEC2BIN.

>> help nchoosek
NCHOOSEK Binomial coefficient or all combinations.
NCHOOSEK(N,K) where N and K are non-negative integers returns N!/K!(N-K)!. 

See also REM.

Overloaded functions or methods (ones with the same name in other directories)
help sym/mod.m
This is the number of combinations of \( N \) things taken \( K \) at a time. When a coefficient is greater than \( 10^{15} \), a warning will be produced indicating possible inexact results. In such cases, the result is good to 15 digits.

\( \text{NCHOOSEK}(V,K) \) where \( V \) is a vector of length \( N \), produces a matrix with \( N!/K!(N-K)! \) rows and \( K \) columns. Each row of the result has \( K \) of the elements in the vector \( V \). This syntax is only practical for situations where \( N \) is less than about 15.

Class support for inputs \( N,K,V \):
- float: double, single

See also \text{PERMS}.

### 1.2 Now For Some Coding

```matlab
>> n = 6;
>> k = 3;
>> p = 2;
>> In = eye(n);
>> Ik = eye(k);
>> Ink = eye(n-k);

>> P = [1 1 0;0 1 1;1 0 1]

P =

1 1 0
0 1 1
1 0 1

>> G = [Ik P];
>> H = mod([-P' Ink],p)

H =

1 0 1 1 0 0
1 1 0 0 1 0
0 1 1 0 0 1

>> mod(G*H',p) % Test G and H construction

ans =

0 0 0
0 0 0
0 0 0
```

### 1.3 Encoding and Listing Codewords

We note that \( "u = \text{dec2base}(0:(p^k - 1),p,k)-'0'" \) can be used instead of \( "\text{de2bi}" \) for \( p > 2 \).

```matlab
>> u = de2bi(0:(2^k - 1),k) % List all binary input vectors

u =
```

3
\[
\begin{array}{cccccc}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\texttt{>> C = mod(u*G,p) \% List all codewords}

\texttt{C =}

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

1.4 Syndromes

\texttt{>> N2 = nchoosek(1:n,2)}

\texttt{ans =}

\[
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
1 \\
2 \\
2 \\
2 \\
2 \\
3 \\
3 \\
3 \\
4 \\
4 \\
5 \\
6 \\
6 \\
\end{array}
\]

\texttt{>> E2 = zeros(length(N2),n);}
\texttt{>> for i=1:length(N2); E2(i,N2(i,:)) = 1; end \% All weight 2 error patterns}
\texttt{>> E2}

\texttt{E2 =}

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

4
>> S2 = mod(E2*H',2)  % List syndromes of weight 2 patterns

S2 =
    1 0 1
    0 1 1
    0 1 0
    1 0 0
    1 1 1
    1 1 0
    1 1 1
    0 0 1
    0 1 0
    0 0 1
    1 0 0
    1 1 0
    1 0 1
    0 1 1

>> S2int = bi2de(S2);  % Assign each syndrome to an integer between 0 and 2^(n-k) - 1

1.5 Simulation

>> M = 5;  % Handle M transmissions at once
>> msg = floor(rand(M,1)*2^k)  % Generate uniform random message numbers

msg =
    4
    6
    7
    5
    1

>> u = de2bi(msg,k)  % Map message number to bit vector

u =
    0 0 1
    0 1 1
    1 1 1
    1 0 1
    1 0 0
>> c = mod(u*G,p); % Encode each message
>> noise = rand(M,n)<0.1 % Generate BSC noise with error prob. 0.1
noise =
    0 0 0 0 1 0
    0 1 0 0 0 0
    0 0 0 0 0 0
    0 0 0 0 1 0
    1 0 1 0 0 0

>> recv = mod(c+noise,p);

1.6 Matrix Tricks
The following tricks MATLAB into performing matrix inverses over prime fields. It uses the fact that det(A)A^{-1} is an integer matrix if A is an integer matrix and it uses the fact that a^{p-2} = a^{-1} for all a \in GF(p). Due to the finite precision of IEEE doubles, the first trick may fail if any element of det(A)A^{-1} is greater than 10^{16}. Likewise, the second may fail if (p - 1)^{p-2} > 10^{16}.

>> A = floor(2*rand(5,5)) % Generate random 5 by 5 binary matrix
A =
    1 0 1 0 1
    0 0 1 0 1
    0 1 0 0 1
    0 0 0 1 0
    0 0 0 1 1

>> det(A)
an =
    -1

>> invA = mod(round(inv(A)*det(A)),2) % Modulo 2 inverse trick (det(A) must be odd)
invA =
    1 1 0 0 0
    0 0 1 1 1
    0 1 0 1 1
    0 0 0 1 0
    0 0 0 1 1

>> mod(invA*A,2) % Verify that it works
ans =
    1 0 0 0 0
    0 1 0 0 0
    0 0 1 0 0
    0 0 0 1 0
    0 0 0 0 1

>> A = floor(7*rand(5,5)) % Generate random 5 by 5 matrix over GF(7)
A =

\[
\begin{bmatrix}
5 & 6 & 1 & 2 & 6 \\
1 & 6 & 0 & 1 & 4 \\
6 & 2 & 3 & 6 & 1 \\
0 & 3 & 5 & 4 & 4 \\
3 & 3 & 2 & 1 & 4
\end{bmatrix}
\]

>> det(A)/7  % Check determinant not divisible by 7
ans =
66.4286

>> invA = round(mod(mod(round(inv(A)*det(A)),7)*mod(det(A),7)^5,7))
invA =

\[
\begin{bmatrix}
6 & 0 & 4 & 3 & 1 \\
5 & 6 & 0 & 5 & 6 \\
5 & 5 & 3 & 1 & 5 \\
1 & 2 & 1 & 0 & 5 \\
3 & 3 & 4 & 4 & 0
\end{bmatrix}
\]

>> mod(invA*A,7)  % Test inverse
ans =

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

2 Extension Fields $\mathbb{F}_{2^m}$

2.1 A Few Commands

Matlab has built in routines that work for extension fields of characteristic 2. These commands can be listed by typing “help gfhelp”.

GF  Create a Galois field array.

\[
X_{GF} = GF(X,M)
\]

creates a Galois field array from X in the field $\mathbb{GF}(2^M)$, for $1 \leq M \leq 16$. The elements of X must be integers between 0 and $2^M-1$. $X_{GF}$ behaves like a MATLAB array, and you can use standard indexing and arithmetic operations (+, *, .*, .^, \, etc.) on it.

For a complete list of operations you can perform on $X_{GF}$, type “GFHELP”.

\[
X_{GF} = GF(X,M,PRIM_POLY)
\]

creates a Galois field array from X and uses the primitive polynomial PRIM_POLY to define the field. PRIM_POLY must be a primitive polynomial in decimal representation. For example, the polynomial $D^3+D^2+1$ is represented by the number 13, because 1 1 0 1 is the binary
form of 13.

\[ X_{GF} = GF(X) \] uses a default value of \( M = 1 \).

**Example:**

```
A = gf(randint(4,4,8,873),3); % 4x4 matrix in GF(2^3)
B = gf(1:4,3)'; % A 4x1 vector
C = A*B
```

\( C = GF(2^3) \) array. Primitive polynomial = 1+D+D^3 (11 decimal)

Array elements =

3
3
6
7

See also GFHELP, GFTABLE.

### 2.2 Simple Linear Block Code

```
>> n = 5;
>> k = 3;
>> m = 2;
>> In = gf(eye(n),m);
>> Ik = gf(eye(k),m);
>> Ink = gf(eye(n-k),m);

P = gf([1 1;1 2;1 3],m) % (5,3) Hamming code over GF(4)
```

\( P = GF(2^2) \) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

1 1
1 2
1 3

```
>> G = [Ik P];
>> H = [P' Ink];
```

\( H = GF(2^2) \) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

1 1 1 1 0
1 2 3 0 1

```
>> G*H' % Test G and H construction
```

\( ans = GF(2^2) \) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

0 0
2.3 Computing Minimal Polynomials

% Big field FQ, small field Fq
m = 8;
Q = 2^m;
a = gf(2,m);
gf1 = a^0;
q = 2^4;
b = round((Q-1)/(q-1));

% Compute polynomial product of all conjugate roots
m = [gf1 -c];
d = c^q;
while (d ~= c)
    m = conv(m,[gf1 -d]);
d = d^q
end

% Write coefficients in power representation for subfield m
log(m)/b
m = GF(2^8) array. Primitive polynomial = D^8+D^4+D^3+D^2+1 (285 decimal) Array elements =
ans =
0 2 3

2.4 Reed-Solomon Codes via FFTs

% Reed-Solomon code over GF(256)
m = 8;
q = 2^m;
n = q-1;
r = 6;
k = n-r;

% Encode message using FFT
u = gf([floor(rand(1,k)*q) zeros(1,n-k)],m);
x = fft(u);

% Construct random error pattern of weight "ne"
ne = 4;
e = gf(zeros(1,n),m);
loc = randperm(n);
mag = gf(floor(rand(1,ne)\*\(q-1)\)+1,m);
e(loc(1:ne)) = mag;

% Add errors and compute syndrome via IFFT
y = x+e;
syn = ifft(y);
syn = syn((k+1):n);