EGR 423 W'01
Scilab Quick Reference

- Constants:

```
%i, %pi, %e: j, π, e
%eps:        the smallest non-zero real number
%t, %f:      true and false (e.g., 2==2 gives %t)
%inf:        ∞
%nan:        NaN, not-a-number
```

- Entry of vectors:

```
v = [1, 2, 3]  // row vector
v = [1; 2; 3]  // column vector
v = [v, 1];    // Adding to an existing vector
```

- Indexing vectors:

```
w = v(1:3);          // Extract a sub-vector
v(1:3) = [0,0,0];    // Replace a sub-vector in-place
v(5:$)               // $ means the last index
v(5:length(v))       // same as v(5:$)
v = v(:);            // Force v to be a column vector
```

- Entry of matrices:

```
m = [1,2,3; 4,5,6; 7,8,9;] // row-at-a-time
m = [[1;2;3], [4;5;6], [7;8;9]] // column-at-a-time
m = [1,2,3; 4,5,6; 7,8,9;].' // rowwise, then transpose
```

- Indexing matrices:

```
M = A(1:4, 2:3);     // Extract a submatrix
M = A(:, 3:5);       // Extract whole rows
M = A(1, :);         // Extract whole columns
x = A(3,5);          // Indexing single elements
A(3,:) = [0,0,0];    // Replace a row in-place
A(:,4) = [1;1;1];    // Replace a column in-place
```

- Operations on complex numbers:

```
z = 2+3*%i;     // entry of complex constants
conj(z);        // complex conjugate
real(z);        // real part
imag(z);        // imaginary part
abs(z);         // magnitude |z|
atan(imag(z),real(z));   // phase ∠z
```

  HINT: `deff('p=phase(z)', 'p=atan(imag(z),real(z))')`
  (if you are offended by the lack of a built-in phase function)

```
[r,p] = polar(z);// compute r = |z|, p = ∠z
```

- Operations on matrices:

```
C = A*B;        // Matrix multiplication
C = A .* B;     // Elementwise multiplication (Kronecker product)
C = A^2;        // Same as A*A
C = A.^2;       // Elementwise power operation
C = A+B-C;      // Matrix addition, subtraction
x = A/b;        // Solution to x*b=A, NOT MATRIX "DIVISION"
C = A ./ B;     // Elementwise division[1]
C = 3*A/2;      // Scalar distributes over array
C = A.';        // Matrix transpose
C = A';         // Matrix Hermitian (conjugate transpose)
C = zeros(3,4);// Matrix of all 0's
C = ones(2,2); // Matrix of all 1's
C = eye(4,4);  // Identity matrix
d = diag(A);   // Extract the diagonal of matrix A
A = diag(d);   // Construct diagonal matrix from vector d
d = det(A);    // Determinant
Ai = inv(A);   // Matrix inverse
[row,col] = size(A);// Determine array dimensions
```

- Useful functions:

```
abs(x);         // Absolute value (or complex magnitude)
exp(x);         // e^x
log(x), log10(x)// Natural logarithm, base-10 logarithm
int(x);         // Truncate real to integer
min(v);         // Minimum of a vector or matrix
max(v);         // Maximum of a vector or matrix
median(v);      // Median of a vector or matrix
mean(v);        // Average value of a vector or matrix
find(v);        // Return INDICES where v is true[2]
sin, cos, tan, asin, acos, atan// Trigonometrics
rand(3,4);      // Create random array
sum(v);         // Sum up all elements in v
prod(v);        // Multiply together all elements in v
cumsum(v);      // Cumulative sum
cumprod(v);     // Cumulative product
linspace(1,10,4); // Create a linear range
logspace(0,4,4); // Create a logarithmic range
norm(M);        // Matrix norm
clear v         // Remove variables from workspace
```

1. A BIG gotcha: the expression "1./A" and "1 ./ A" are different, because of the way the dot '.' is either associated with the 1 or the / operator. The first expression "1./A" is really (1.0)/A and does NOT implement the element-wise reciprocal of each matrix entry. That's what the second expression does, "1 ./ A".
2. The idiom to find, for example, where in a vector a certain value exists is "find(v == 3)". Consider, for example, the idiom for finding a local maximum: "find((v(2:$-1) > v(1:$-2)) & (v(2:$-1) > v(3:$)))+1"

```
save('c:\backup.dat'); // Save workspace to file
load('c:\backup.dat'); // Load workspace from file
diary('c:\diary.dat'); // Start a diary
diary(0);              // Stop a diary
```

- Selected operators:

```
&, |, ~   // logical AND, OR, NOT (not bitwise and,or,not)
**, ^     // exponentiation (synonyms)
=         // assignment (e.g., "v=3")
==        // logical equality (e.g., "tf = (v==3)")
~=        // logical non-equality (e.g., "if v ~= 3")
<, >, <=, >= // logical inequality tests
```

- Polynomials:

```
z = poly(0,'z');    // Create the polynomial P(z)=z ...
p = z^2 + 3*z;      // ... then use P(z) to form z^2 + 3z
p = poly([0,3,1], 'z', 'coeff');// Alternate way
p = poly([0,-3], 'z', 'roots'); // Polynomial from roots
roots(p);           // Find polynomial roots
polfact(p);         // Factor polynomial
horner(p,3);        // Evaluate polynomial at a number
horner(p,z^2+1);    // Polynomial substitution
p*q, p/q, p+q       // Polynomial operations
numer(p);           // Numerator of polynomial fraction
denom(p);           // Denominator of polynomial fraction
```

- Control Flow:

```
if i > 2,
  stuff();
end

if i ~= 3,
  stuff();
else
  other();
end

while i < 10,
  disp(i);
end

for i=1:10,
  disp(i);
end
```

```
for str = [‘abc’, ‘def’],
  printf(“String: %s”, str);
end

select value,
  case 0,
    printf(“Ooops...value is 0!”);
  case 1,
    printf(“It’s 1”);
  else
    break;
end
```

- Functions:

```
function noargs()        // No return value, no parameters
function onearg(x)       // No return value, one parameter
function twoargs(x,y)    // No return value, two parameters
function x = stuff(x,y)  // One return value
function [x,y]=stuff(a,b)// Two return values

getf(‘c:\myfunc.sci’);   // Load function into workspace
deff(‘tryit’, ‘exec(‘‘c:\tryit.sce’’)’);
                         // On-the-fly function definition

    function [a,b,c]=checkargs(x,y,z)
    // Shows how to check for the number of parameters
    // actually passed and how many return values were
    // asked for.
    [lhs,rhs] = argn(0);
    printf(“You specified %d parameters”, rhs);
    printf(“You asked for %d return values”, lhs);

    if rhs < 3,
      z = 0;    // default value
    end
    if rhs < 2,
      y = 10;   // default value
    end
```

- Plotting:

```
plot(v);             // Simplest way to plot a vector
plot(xaxis,y);       // Plot y against axis in a vector
plot(x,y,’t’,’x(t)’,’Title’);// Specify captions and title
plot2d(...);         // Control over plot styles, axes, legend
plot2d1(...);        // Also allows logarithmic axes
plot2d2(...);        // Piecewise constant (stairstep) curves
```

```
plot2d3(...);       // Isolated vertical bars
plot2d4(...);       // Arrows
xbasc();            // Clear plot window
xdel();             // Close plot window
errbar(...);        // Add error bars
locate(...);        // Pick values off a graph with the mouse
xclick();           // Wait for mouse button press in window
x_dialog('Message', 'Press OK');// Create dialog window
```

- Linear Systems:

```
H = syslin('d', z^2+1, 1-z);// Create linear system with
                            // H(z) = N(z)/D(z)
                            // and z=poly(0,'z')
H('num') or numer(H)   // Extract numerator
H('den') or denom(H)   // Extract denominator
H*G                 // Linear systems in series
H+G                 // Linear systems in parallel
H /. G              // Negative feedback configuration
[f,r]=repfreq(H);   // Frequency response
pfss(H);            // Partial fraction decomposition
y=rtitr(numer(H), denom(H), u);// Time response
```

- Signal Processing:

```
convol(h,x);        // Convolution
eqfir(...);         // Parks-McLellan FIR filter design
eqiir(...);         // IIR filter design (bilinear transform)
fft(x,-1);          // Forward DFT (NOTE: -1)
fft(x,+1);          // Inverse DFT (NOTE: +1)
frmag(...);         // Filter frequency response
wfir(...);          // Window method of FIR filter design
window('hm',128);   // Create window functions
```