MATLAB Tips and Tricks

Gabriel Peyré
CEREMADE, Université Paris Dauphine
gabriel.peyre@ceremade.dauphine.fr

June 30, 2007

First keep in mind that this is **not** a MATLAB tutorial. This is just a list of tricks I have found useful while writing my toolboxes available on the Matlab Central repository

http://www.mathworks.com/matlabcentral/

You can e-mail me if you have corrections about these pieces of code, or if you would like to add your own tips to those described in this document.

Contents

1 General Programming Tips

- Suppress entries in a vector.

```
x( 3:5 ) = [];
```

- Reverse a vector.

```
x = x(end:-1:1);
```

- Use cell arrays to store stuff of various types.

```
x = {}; x{end+1} = 1; x{end+1} = [1 2]; % build incrementally
x = {x{end:-1:1}}; % reverse the same way as a vector
```

- Compute the running time of a function call.

```
tic; fft(rand(500)); disp( ['it takes ' num2str(toc) 's.']  );
```

- Make a array full of `NaN`

```
% guess which one is the fastest ?
tic; NaN*ones(2000,2000); toc;
tic; repmat(NaN,2000,2000); toc;
```

- Turn an nD array into a vector.

```
x = x(:);
```

- Compute the maximum value of an nD array.

```
m = max(x(:));
```

- Access a matrix from a list of entries. Here, we have `I = [I1; I2]` and `y(i) = M( I1(i), I2(i) )`

```
J = sub2ind(size(M), I(1,:),I(2,:)  );
y = M(J);
```

- Create a function that take optional arguments in a struct.

```
function y = f(x,options)
% parse the struct
if nargin<2
  options.null = 0; %  force creation of options
end
if isfield(options, 'a')
  options.a = 1; %  default value
end
a = options.a;
if isfield(options, 'b')
  options.b = 1; %  default value
end
b = options.b;
% Here the body of the function ...
```

- How to duplicate a character `n` times.

```
str = char( zeros(n,1)+'*' );
```

- Assign value v in a nD array at a position ind (lenth-n vector).

```
ind = num2cell(ind);
x( ind{:} ) = v; % the comma-separated trick
```

- Write a function fun(a,b,c) that can takes an arbitrary number of arguments.

```
% first method, simple but long
function fun(a,b,c)
if nargin<1
  a = 0.1246;
end
if nargin<2
  b = 1.2346;
end
if nargin<3
  c = 8.7643;
end
% second method, much more elegant
function fun(args)
default_values = {0.1246,1.2346,8.7643};
args_present =  cellfun(isempty,args);
default_values(args_presen) = args(args_present);
[a b c] = deal(default_values:);
```

- Remove the ticks from a drawing.

```
set(gca, 'XTick', []);
set(gca, 'YTick', []);
```

- Find the angle that makes a 2D vector x with the vector [1,0]

```
% just the angle
theta = atan2(x(2),x(1));
% if you want to compute the full polar decomposition
[theta,r] = cart2pol(x);
```

- Try to allocate memory before adding new data to an array

```
n = 10000; a = [];
tic;
for i=1:n
  a(i) = 1; % this will reallocate size
end
toc; tic;
for i=1:n
  a(i) = 1;
end
toc; % should be 15 times faster
```

2 Input/Output tips

- Create a graphical waitbar.

```
n = 100;
h = waitbar(0,'Waiting ...');
for i=1:n
  waitbar(i/n);
  % here perform some stuff
end
close(h);
```

- Output a string without carriage return.

```
fprintf('Some Text');
```

- Saving and loading an image.

```
% saving current display
saveas(gcf, 'my_image', 'png');
% saving a 2D or 3D matrix into an image
imwrite(M, 'my_image', 'png'); % M should have its values in [0,1]
% loading into a 2D (gray) or 3D (color) matrix
M = double( imread( 'my_image.png' ) );
```

- Saving and loading a matrix `M` in a binary file.

```
[n,p] = size(M); % saving
str = 'my_file'; % name of the file
fid = fopen(str,'wb');
if fid<0
  error(['error writing to file ', str]);
end
fwrite(fid,M','double'); % store it row-wise format
fclose(fid);
% loading
fid = fopen(str,'rb');
if fid<0
  error(['error reading file ',str]);
end
[M, cnt] = fread(fid,[n,p],'double'); M = M';
fclose(fid);
if cnt =n*p
error(['Error reading file ', str]);
end
```

- Writing/Reading to a text file a list of 3-uplets.

```
% A is a matrix with 3 rows.
fid = fopen(filename,'wt');
fprintf(fid, '%f %f %f\n', A);
fclose(fid); % Retrieving the values back from file to matrix B.
fid = fopen(filename,'r');
[B,cnt] = fscanf(fid,'%f %f %f');
```

- Building an AVI file.

```
mov = avifile('filename');
for i=1:nbrframes
% draw some stuff here
F = getframe(gca);
mov = addframe(mov,F);
end
mov = close(mov);
```

3 General Mathematical Tips

- Rescale the entries of a vector x so that it spans $[0, 1]$

```
m = min(x(:)); M = max(x(:));
x = (b-a) * (x-m)/(M-m) + a;
```

- Generate n points evenly sampled.

```
x = 0:1/(n-1):1; % faster than linspace
```

- Compute the L^2 squared norm of a vector or matrix x.

```
m = sum(x(:).^2);
```

- Subsample a vector x or an image M by a factor 2.

```
x = x(1:2:end); % useful for wavelet transform
M = M(1:2:end,1:2:end);
```

- Compute centered finite differences.

```
D1 = [x(2:end),x(end)];
D2 = [x(1),x(1:end-1)];
y = (D1-D2)/2;
```

- Compute the prime number just before n

```
n = 150;
P = primes(n); n = P(end);
```

- Compute J, the reverse of a permutation I, i.e. an array which contains the number 1:n in arbitrary order.

```
J(I) = 1:length(I);
```

- Shuffle an array x.

```
y = x( randperm(length(x)) );
```

4 Advanced Mathematical Tips

- Generate n points x sampled uniformly at random on a sphere.

```
% tensor product gaussian is isotropic
x = randn(3,n);
d = sqrt( x(1,:).^2+x(2,:).^2+x(2,:).^2 );
x(1,:)  = x(1,:)./d;
x(2,:)  = x(2,:)./d;
x(3,:)  = x(3,:)./d;
```

- Create a random vector x with s non-zero entry (s-sparse vector)

```
sel = randperm(n); sel = sel(1:s);
x = zeros(n,1); x(sel) = 1;
```

- Construct a polygon x whose ith sidelength is s(i). Here x(i) is the complex affix of the ith vertex.

```
theta = [0;cumsum(s)];
theta = theta/theta(end);
theta = theta(1:(end-1));
x = exp(2i*pi*theta);
L = abs(x(1)-x(2));
x = x*s(1)/L; % rescale the result
```

- Compute y, the inverse of an integer x modulo a prime p.

```
% use Bezout thm
[u,y,d] = gcd(x,p);
y = mod(y,p);
```

- Compute the curvilinear abscise s of a curve c. Here, c(:,i) is the ith point of the curve.

```
D = c(:,2:end)-c(:,1:(end-1));
s = zeros(size(c,2),1);
s(2:end) = sqrt( D(1,:).^2 + D(2,:).^2 );
s = cumsum(s);
```

- Compute the 3D rotation matrix M around an axis v

```
% taken from the OpenGL red book
v = v/norm(v,'fro');
S = [0 -v(3) v(2); v(3) 0 -v(1); -v(2) v(1) 0];
M = v*transp(v) + cos(alpha)*(eye(3) - v*transp(v)) + sin(alpha)*S;
```

- Compute a VanderMonde matrix M i.e. M(i,j)=x(i)^j for j=0:d.

```
n = length(x); % first method
[J,I] = meshgrid(0:d,1:n);
A = x(I).^J;
% second method, less elegant but faster
A = ones(n);
for j = 2:n
  A(:,j) = x.*A(:,j-1);
end
```

- Threshold (i.e. set to 0) the entries below `T`.

```
% first solution
x = (abs(x)>=T) .* x;
% second one :  nearly 2 times slower
I = find(abs(x)<T); x(I) = 0;
```

- Same, but with soft-thresholding

```
s = abs(x) - t;
s = (s + abs(s))/2;
y = sign(x).*s;
```

- Keep only the `n` biggest coefficients of a signal `x` (set the others to 0).

```
[tmp,I] = sort(abs(x(:))); x( I(1:end-n) ) = 0;
```

- Draw a 3D sphere.

```
p = 20; %  precision
t = 0:1/(p-1):1;
[th,ph] = meshgrid( t*pi,t*2*pi );
x = cos(th);
y = sin(th).*cos(ph);
z = sin(th).*sin(ph);
surf(x,y,z, z.*0);
% some pretty rendering options
shading interp; lighting gouraud;
camlight infinite; axis square; axis off;
```

- Project 3D points on a 2D plane (best fit plane). `P(:,k)` is the kth point.

```
for i=1:3 %  substract mean
  P(i,:)  = P(i,:)  - mean(P(i,:));
end
C = P*P'; %  covariance matrix
% project on the two most important eigenvectors
[V,D] = eigs(C);
Q = V(:,1:2)'*P;
```

- Compute the pairwise distance matrix `D` between a set of `p` points in R^d. Here, `X(:,i)` is the `i`th point.

```
X2 = sum(X.^2,1);   D = repmat(X2,p,1) + repmat(X2',1,p)-2*X'*X;
```

- Orthogonalize a matrix A by projection on the set of orthogonal matrix (not Gram-Schmidt)

```
[U,D,V] = svd(A); A = U*V';
```

5 Signal Processing Tips

- Compute circular convolution of x and y.

```
% use the Fourier convolution thm
z = real( ifft( fft(x).*fft(y) ) );
```

- Build a 1D gaussian filter of variance s.

```
x = -1/2:1/(n-1):1/2;
f = exp( -(x.^2)/(2*s^2) );
f = f / sum(sum(f));
```

- Perform a 1D convolution of signal f and filter h with symmetric boundary conditions. The center of the filter is 0 for odd length filter, and 1/2 otherwise

```
n = length(x); p = length(h);
if mod(p,2)==1
  d1 = (p-1)/2; d2 = (p-1)/2;
else
  d1 = p/2-1; d2 = p/2;
end
xx = [ x(d1:-1:1); x; x(end:-1:end-d2+1) ];
y = conv(xx,h);
y = y( (2*d1+1):(2*d1+n) );
```

- Generate a signal whose regularity is $\mathcal{C}^\alpha$ (Sobolev).

```
alpha = 2; n = 100;
y = randn(n,1); % gaussian noise
fy = fft(y);
fy = fftshift(fy);
% filter the noise with |omega|^-alpha
h = (-n/2+1):(n/2);
h = (abs(h)+1).^(-alpha-0.5);
fy = fy.*h';
fy = fftshift(fy);
y = real( ifft(fy) );
y = (y-min(y))/(max(y)-min(y));
```

- Generate a signal whose regularity is nearly $\mathcal{C}^{\alpha-1/2}$.

```
alpha = 3; n = 300;
x = rand(n,1); % uniform noise
for i=1:alpha % integrate the noise alpha times
  x = cumsum(x - mean(x));
end
```

- Compute the PSNR between to signals `x` and `y`.

```
d = mean( (x(:)-y(:)).^2 );
m = max( max(x(:)),max(y(:))  );
PSNR = 10*log10( m/d );
```

- Evaluate a cubic spline at value `t` (can be a vector).

```
x = abs(t) ;
I12 = (x>1)&(x<=2); I01 = (x<=1);
y = I01.*( 2/3-x.^2.*(1-x/2) ) + I12.*( 1/6*(2-x).^3 );
```

- Perform spectral interpolation of a signal `x` (aka Fourier zero-padding). The original size is `n` and the final size is `p`

```
n = length(x); n0 = (n-1)/2;
f = fft(x); % forward transform
f = p/n*[f(1:n0+1); zeros(p-n,1); f(n0+2:n)];
x = real( ifft(f) ); % backward transform
```

- Compute the approximation error `err`$= ||f - f_M||/||f||$ obtained when keeping the M best coefficients in an orthogonal basis.

```
% as an example we take the decomposition in the cosine basis
M = 500;
x = peaks(128); y = dct(x); % a sample function
[tmp,I] = sort(abs(y(:)));
y(I(1:end-M)) = 0;
err = norm(y,'fro')/norm(x,'fro'); % the relative error
xx = idct(y); imagesc(xx); % the reconstructed function
```

- Evaluate the number of bits needed to code a vector `v` with arithmetic coding.

```
h = hist(v,100); % use 100 bins for histogram
% use Shannon's upper bound
nbr_bits = - length(v(:))  * sum( h.*log2(h) );
```

- Perform the histogram equalization of a vector `x` so that it matches the histogram of another vector `y`.

```
[vx,Ix] = sort(x); [vy,Iy] = sort(y);
nx = length(x); ny = length(y);
ax = linspace(1,ny,nx); ay = 1:ny;
vx = interp1(ay,vy,ax);
x(Ix) = vx;
```

6 Image Processing Tips

- Display the result of an FFT with the 0 frequency in the middle.

```
x = peaks(256);
imagesc( real( fftshift( fft2(x) ) ) );
```

- Resize an image `M` (new size is `(p1,q1)`).

```
[p,q] = size(M); % the original image
[X,Y] = meshgrid( (0:p-1)/(p-1), (0:q-1)/(q-1) );
%  new sampling location
[XI,YI] = meshgrid( (0:p1-1)/(p1-1) , (0:q1-1)/(q1-1) );
M1 = interp2( X,Y, M, XI,YI ,'cubic'); %  the new image
```

- Build a 2D gaussian filter of variance `s`.

```
x = -1/2:1/(n-1):1/2;
[Y,X] = meshgrid(x,x);
f = exp( -(X.^2+Y.^2)/(2*s^2) );
f = f / sum(f(:));
```

- Image convolution with centered filter

```
n = length(x); p = length(h);
if mod(p,2)==1
  d1 = (p-1)/2; d2 = (p-1)/2;
else
  d1 = p/2-1; d2 = p/2;
end
xx = [ x(d1:-1:1,:); x; x(end:-1:end-d2+1,:)  ];
xx = [ xx(:,d1:-1:1), xx, xx(:,end:-1:end-d2+1) ];
y = conv2(xx,h);
y = y( (2*d1+1):(2*d1+n), (2*d1+1):(2*d1+n) );
```

- Extract all 0th level curves from an image `M` an put these curves into a cell array `c_list`.

```
c = contourc(M,[0,0]);
k = 0; p = 1;
while p < size(c, 2) % parse the result
  lc = c(2,p); %  length of the curve
  cc = c(:,(p+1):(p+lc));
  p = p+lc+1;
  k = k+1;
  c_list{k} = cc;
end
```

- Quick computation of the integral `y` of an image `M` along a 2D curve `c` (the curve is assumed in $[0,1]^2$)

```
cs = c*(n-1) + 1; %  scale to [1,n]
I = round(cs);
J = sub2ind(size(M), I(1,:),I(2,:)  );
y = sum(M(J));
```

- Draw the image of a disk and a square.

```
n = 100; x = -1:2/(n-1):1;
[Y,X] = meshgrid(x,x);
c = [0,0]; r = 0.4; %  center and radius of the disk
D = (X-c(1)).^2 + (Y-c(2)).^2 < r^2;
imagesc(D); % a disk
C = max(abs(X-c(1)),abs(Y-c(2)))<r;
imagesc(C); % a square
```

- Draw a 2D function whose value `z` is known only at scattered 2D points `(x,y)`.

```
n = 400;
x = rand(n,1); y = rand(n,1);
% this is an example of surface
z = cos(pi*x) .* cos(pi*y);
tri = delaunay(x,y); % build a Delaunay triangulation
trisurf(tri,x,y,z);
```

- Generate a noisy cloud-like image M whose Fourier spectrum amplitude is $\widehat{M}(\omega) = \omega^{-\alpha}$.

```
x = -n/2:n/2-1;
[Y,X] = meshgrid(x,x);
d = sqrt(X.^2 + Y.^2) + 0.1;
f = rand(n)*2*pi;
M = (d.^(-alpha)) .* exp(f*1i);
M = real( ifft2( ifftshift(M) ) );
```

- Perform a JPEG-like transform of an image `x` (replace `dct` by `idct` to compute the inverse transform).

```
bs = 8; % size of the blocks
n = size(x,1); y = zeros(n,n);
nb = n/bs; % n must be a multiple of bs
for i=1:nb
for j=1:nb
  xsel = ((i-1)*bs+1):(i*bs);
  ysel = ((j-1)*bs+1):(j*bs);
  y(xsel,ysel) = dct(x(xsel,ysel));
end
end
```

- Extract interactively a part `MM` of an image `M`.

```
[n,p] = size(M);
imagesc(M);
axis image; axis off;
sp = getrect;
sp(1) = max(floor(sp(1)),1); % xmin
sp(2) = max(floor(sp(2)),1); % ymin
sp(3) = min(ceil(sp(1)+sp(3)),p); % xmax
sp(4) = min(ceil(sp(2)+sp(4)),n); % ymax
MM = M(sp(2):sp(4), sp(1):sp(3));
```

- Compute the boundary points of a shape represented as a binary image `M`.

```
M1 = conv2(M,ones(3)/9, 'same');
% A indicates with 1 the location of the boundary A = Mh>0 & Mh<9 & M==1;
I = find(A); % index of the boundary
[x,y] = ind2sub(size(A),I);
plot(x,y); % display the boundary
```

- Solve the Poisson equation $\Delta G = \text{div}(f)$ where $\text{div}(f)$ is stored in image `d` and G in image `G`.

```
% Compute the Laplacian filter in Fourier [Y,X] = meshgrid(0:n-1,0:n-1);
mu = sin(X*pi/n).^2; mu = -4*( mu+mu' );
mu(1) = 1; % avoid division by 0
% Inverse the Laplacian convolution
G = fft2(d) ./ mu; G(1) = 0;
G = real( ifft2( G ) );
```

- Extract all the $w \times w$ patches with a spacing `q` from an image `M`

```
[Y,X] = meshgrid(1:q:n-w, 1:q:n-w); p = size(X,1);
[dY,dX] = meshgrid(0:w-1,0:w-1);
Xp = repmat( reshape(X,[1,1,p]) ,[w w 1]) + repmat(dX,[1 1 p]);
Yp = repmat( reshape(Y,[1,1,p]) ,[w w 1]) + repmat(dY,[1 1 p]);
I = sub2ind([n n], Xp,Yp);
H = M(I);
```

7 Graph Theory Tips

- Compute the shortest distance between all pair of nodes (`D` is the weighted adjacency matrix).

```
% non connected vectices must have Inf value
N = length(D);
for k=1:N
  D = min(D,repmat(D(:,k),[1 N])+repmat(D(k,:),[N 1]));
end
D1 = D;
```

- Turn a triangulation into an adjacency matrix.

```
nvert = max(max(face));
nface = length(face);
A = zeros(nvert);
for i=1:nface
  A(face(i,1),face(i,2)) = 1; A(face(i,2),face(i,3)) = 1;
  A(face(i,3),face(i,1)) = 1;
  %  make sure that all edges are symmetric
  A(face(i,2),face(i,1)) = 1; A(face(i,3),face(i,2)) = 1;
  A(face(i,1),face(i,3)) = 1;
end
```

8 Wavelets and Multiresolution Tips

- Compute a 2D Haar transform of an image x

```
if dir==1
  for j=Jmax:-1:Jmin
    sel = 1:2^(j+1);
    x(sel,sel) = fwd_step(x(sel,sel));
    x(sel,sel) = fwd_step(x(sel,sel)')';
  end
  y = x;
else
  for j=Jmin:Jmax
    sel = 1:2^(j+1);
    x(sel,sel) = bwd_step(x(sel,sel)')';
    x(sel,sel) = bwd_step(x(sel,sel));
  end
  y = x;
end

function M1 = fwd_step(M)
C = M(1:2:end,:); D = M(2:2:end,:);
M1 = [(C+D)/sqrt(2); (C-D)/sqrt(2)];

function M1 = bwd_step(M)
C = M(1:end/2,:); D = M(end/2+1:end,:);
M1 = M; M1(1:2:end,:)  = (C+D)/sqrt(2);
M1(2:2:end,:)  = (C-D)/sqrt(2);
```