BE 537 - Grand Challenge 1

James Wang, Michael Lautman, Shreel Vijayvargiya

March 22, 2015

Introduction

For this project we explored methods for performing groupwise registration between a set of brain images. We performed registration on a set of 3D brain images solving for transformations from the set of images into a common reference space. The quality of a registration was established by measuring how closely a set of labeled features in the images corresponded when projected into the common frame via the transformations we built.

3.1 The Basic Component

To configure the environment in matlab we have included a setup script.

```
% Setup the environment
addpath(genpath('./train/'))
addpath(genpath('./test/'))
addpath(genpath('./NIFTI_20110921/'))
addpath(genpath('./starter_code/'))
```

3.1.1 Extend myView to Display Registration Results

Our project uses a visualizer that takes in a fixed image, a moving image, the voxel spacing in the image, a rotation matrix and a translation vector.

```
function myViewAffineReg(fixed, moving, spacing, A, b)
```

```
function myViewAffineReg(fixed, moving, spacing, A, b)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% myView extended to display affine transformation results  %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% inputs - fixed,moving,spacing,A,b                                     %%
%% fixed - fixed image                                                   %%
%% moving - moving image                                                 %%
%% spacing - voxel spacing                                               %%
%% A - 3 x 3 rotation, scaling and shearing matrix                       %%
%% b - 3 x 1 translation matrix                                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fixed = double(fixed);
moving = double(moving);
```

```
dimen = size(fixed);

% crosshair at the center of the image
crosshair = round(dimen/2);

% k-means segmentation for XY slice
fixed_xy = transpose(squeeze(fixed(:,:,crosshair(3))));
[~, ctr_xy] = kmeans(fixed_xy(fixed_xy > 0), 3);
cont_xy = conv(sort(ctr_xy), [0.5 0.5], 'valid');

% k-means segmentation for XZ slice
fixed_xz = transpose(squeeze(fixed(:,crosshair(2),:)));
[~, ctr_xz] = kmeans(fixed_xz(fixed_xz > 0), 3);
cont_xz = conv(sort(ctr_xz), [0.5 0.5], 'valid');

% k-means segmentation for YZ slice
fixed_yz = transpose(squeeze(fixed(crosshair(1),:,:)));
[~, ctr_yz] = kmeans(fixed_yz(fixed_yz > 0), 3);
cont_yz = conv(sort(ctr_yz), [0.5 0.5], 'valid');

resampled = myTransformImage(fixed, moving, A, b);

% extract 2D images from  3D array
image_xy = transpose(squeeze(resampled(:,:,crosshair(3))));
image_xz = transpose(squeeze(resampled(:,crosshair(2),:)));
image_yz = transpose(squeeze(resampled(crosshair(1),:,:)));

% intensity range
imin = min(min(min(resampled)));
imax = max(max(max(resampled)));
crange = [imin imax];

cmap = 'gray';

figure;

% show image for XY plane
subplot(2,2,1)
imagesc(image_xy);
set(gca,'XDir','reverse');
set(gca,'YDir','normal');

% set aspect ratio
daspect([spacing(2);spacing(1);1]);

% set color axis limits
caxis(crange);
title(['z = ',num2str(crosshair(3))]);

% draw crosshairs
line([0 dimen(1)],[crosshair(2) crosshair(2)],'color','b');
line([crosshair(1) crosshair(1)],[0 dimen(2)],'color','b');

% draw contours
hold on
contour(fixed_xy, cont_xy, 'g');
hold off

% image for YZ plane
subplot(2,2,2)
imagesc(image_yz);
set(gca,'XDir','reverse');
set(gca,'YDir','normal');
daspect([spacing(3);spacing(2);1]);
caxis(crange);
```

```
title(['x = ',num2str(crosshair(1))]);
line([0 dimen(2)],[crosshair(3) crosshair(3)],'color','b');
line([crosshair(2) crosshair(2)],[0 dimen(3)],'color','b');
hold on
contour(fixed_yz, cont_yz, 'g');
hold off

% image for XZ plane
subplot(2,2,3)
imagesc(image_xz);
set(gca,'XDir','reverse');
set(gca,'YDir','normal');
daspect([spacing(3);spacing(1);1]);
caxis(crange);
title(['y = ',num2str(crosshair(2))]);
line([0 dimen(1)],[crosshair(3) crosshair(3)],'color','b');
line([crosshair(1) crosshair(1)],[0 dimen(3)],'color','b');
hold on
contour(fixed_xz, cont_xz, 'g');
hold off

% show colorbar
subplot(2,2,4)
caxis(crange);
axis off;
colorbar('south');

% display crosshair position and dimensions of image
text(0.2,0.8,sprintf('xhair = [%d %d %d]\ndimen = [%d %d %d]', crosshair, dimen));

% set colormap
colormap(cmap)
hold off

end
```

3.1.2 3D Affine Registration Objective Function

We compute the objective function for 3D registration using the equation below.

$$E(A,b) = \int_{\Omega} [I(x) - J(Ax+b)]^2 dx$$

```
function [E,g] = myAffineObjective3D(p,I,J,varargin)
```

```
function [E,g] = myAffineObjective3D(p,I,J,varargin)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Objective function for 3D Affine Transform  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% inputs - p,I,J                                                       %%
%% optional - dJ/dy,dJ/dx,dJ/dz                                         %%
%% p - 12 x 1 parameter vector                                          %%
%% I - fixed image                                                      %%
%% J - moving image                                                     %%
%% dJ/dy - gradient of moving image in y direction                      %%
%% dJ/dx - gradient of moving image in x direction                      %%
%% dJ/dz - gradient of moving image in z direction                      %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% outputs - E,g                                                          %%
%% E - value of the objective function                                    %%
%% g - gradient of the objective function                                 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check if number of arguments are between 3 and 6
minarg = 3;
maxarg = 6;
narginchk(minarg, maxarg);

A = reshape(p(1:9),[3,3]);
b = p(10:12);

% coordinates of voxels in fixed image
[x, y, z] = ndgrid(1:1:size(I,1),1:1:size(I,2),1:1:size(I,3));

X = transpose([x(:) y(:) z(:)]);

b_rep = repmat(b, 1, numel(x));

% transformation parameters
phi = A*X + b_rep;

phi_x = phi(1,:);
phi_y = phi(2,:);
phi_z = phi(3,:);

phi_x = reshape(phi_x, size(I));
phi_y = reshape(phi_y, size(I));
phi_z = reshape(phi_z, size(I));

% resample moving image
data = my_interp3_precompute(size(I), phi_x, phi_y, phi_z);
J_t = my_interp3(J,data);

% J_t = interpn(J,phi_x,phi_y,phi_z,'linear',0);

% compute the difference image
diff_image = I - J_t;

% compute the value of the objective function
E = sum(sum(sum(diff_image.^2)));

% compute gradient of resampled image
if (nargin == 6 && ~isempty(varargin{1}) && ~isempty(varargin{2}) && ~isempty(varargin{3}))
    dJdy = varargin{1};
    dJdx = varargin{2};
    dJdz = varargin{3};
else
    [dJdy, dJdx, dJdz] = gradient(J);
end

dJdx_phi = my_interp3(dJdx,data);
dJdy_phi = my_interp3(dJdy,data);
dJdz_phi = my_interp3(dJdz,data);

% dJdx_phi = interpn(dJdx,phi_x,phi_y,phi_z,'linear',0);
% dJdy_phi = interpn(dJdy,phi_x,phi_y,phi_z,'linear',0);
% dJdz_phi = interpn(dJdz,phi_x,phi_y,phi_z,'linear',0);

% compute partial derivative of E w.r.t. p
g = [-2*sum(sum(sum(diff_image.*dJdx_phi.*x)));
    -2*sum(sum(sum(diff_image.*dJdy_phi.*x)));
    -2*sum(sum(sum(diff_image.*dJdz_phi.*x)));
    -2*sum(sum(sum(diff_image.*dJdx_phi.*y)));
```

```
    -2*sum(sum(sum(diff_image.*dJdy_phi.*y)));
    -2*sum(sum(sum(diff_image.*dJdz_phi.*y)));
    -2*sum(sum(sum(diff_image.*dJdx_phi.*z)));
    -2*sum(sum(sum(diff_image.*dJdy_phi.*z)));
    -2*sum(sum(sum(diff_image.*dJdz_phi.*z)));
    -2*sum(sum(sum(diff_image.*dJdx_phi)));
    -2*sum(sum(sum(diff_image.*dJdy_phi)));
    -2*sum(sum(sum(diff_image.*dJdz_phi)))];

end
```

3.1.3 Testing the Correctness of Gradient Computation

We verify our analytic gradient computation by computing a numerical gradient aproximation that utilizes the central finite difference approximation.

$$\frac{\partial E}{\partial p_j}|_p \simeq \frac{E(p+\epsilon e_j) - E(p-\epsilon e_j)}{2\epsilon}$$

```
% 3.1.3 Testing the Correctness of Gradient Computation
clear;

[image1,spacing1] = myReadNifti('sub001_mri.nii');
[image2,spacing2] = myReadNifti('sub002_mri.nii');
p = [1,0,0,0,1,0,0,0,1,0,0,0]';

% Gaussian LPF
sigma = 1;
smoothedimage1 = myGaussianLPF(image1,sigma);
smoothedimage2 = myGaussianLPF(image2,sigma);

% analytical gradient
[E,g] = myAffineObjective3D(p,smoothedimage1,smoothedimage2);

% numerical gradient
epsilon = 1e-4;
gnumer = ones(12,1);

for j = 1:12
    % Create ej vector
    ej = zeros(12,1);
    ej(j) = 1;

    % Add/subtract ej*epsilon vector to p
    pup = p + ones(12,1).*ej*epsilon;
    pdown = p - ones(12,1).*ej*epsilon;

    % Compute E terms for numerical gradient approximation
    [Eup,~] = myAffineObjective3D(pup,smoothedimage1,smoothedimage2);
    [Edown,~] = myAffineObjective3D(pdown,smoothedimage1,smoothedimage2);

    % Compute dE/dpj
    gnumer(j) = (Eup - Edown)/(2*epsilon);
end

% Compute relative error
diffvector = g - gnumer;
relerr = 100.*(diffvector./g);
```

The maximum relative error we found using an epsilon of $1e-4$ was 0.01%.

3.1.4 Testing our objective function by registering two images

```
% problem 3.1.4
% Minimizing the objective function and displaying resulting registration

clear;

% load images
[image1,spacing] = myReadNifti('sub001_mri.nii');
[image2,spacing2] = myReadNifti('sub002_mri.nii');

% Gaussian LPF
sigma = 1;
smoothedimage1 = myGaussianLPF(image1,2*sigma);
smoothedimage2 = myGaussianLPF(image2,2*sigma);

% subsample images in each dimension by 2
subsamp1 = smoothedimage1(1:2:end,1:2:end,1:2:end);
subsamp2 = smoothedimage2(1:2:end,1:2:end,1:2:end);

% Create a struct object for storing gradient of moving image
g_struct = struct('dy',{},'dx',{},'dz',{});

% compute gradient of moving image
[g_struct(1).dy, g_struct(1).dx, g_struct(1).dz] = gradient(subsamp2);

% Set options for optimization
options = optimset('GradObj','on','Hessian','on','Display','iter','MaxIter',50);
% options = optimset('GradObj','on','Display','iter','MaxIter',50);

% Run optimization
pstart = [1,0,0,0,1,0,0,0,1,0,0,0]';
[p,fval] = fminunc(@(x)(myAffineObjective3DwithHessian(x, subsamp1, subsamp2, g_struct(1).dy, g_struct(1).dx, g_s
% [p,fval] = fminunc(@(x)(myAffineObjective3D(x, subsamp1, subsamp2, g_struct(1).dy, g_struct(1).dx, g_struct(1).
% [p,fval] = fminunc(@(x)(myAffineObjective3D(x, subsamp1, subsamp2)), pstart, options);

% output original results
Aorig = reshape(pstart(1:9),[3,3]);
borig = pstart(10:12);
myViewAffineReg(subsamp1,subsamp2,spacing,Aorig,borig);

% output optimal results
A = reshape(p(1:9),[3,3]);
b = p(10:12);
myViewAffineReg(subsamp1,subsamp2,spacing,A,b);
```

Running our solution on an example