Table of Contents

Spectral inversion for multiple wavelengths

```
clear all
close all
format shortg

WaveLength = [680   , 710   , 750   , 850   , 920  , 950  ];
WaveLength =  [720 730 750 760 780 800 850 900];
NWavelength = length(WaveLength);
```

Input files

```
labelfilename = 'Phantom_ATROPOS_GMM.0001.nii.gz'; % initial tissue segmentation
maskfilename  = 'PhantomMask.nii.gz'; % ROI
pafilebase    = 'PhantomPadata'; % PA Data at each wavelength
setupfilename = 'PhantomLaserVesselSetup.nii.gz'; % laser location
```

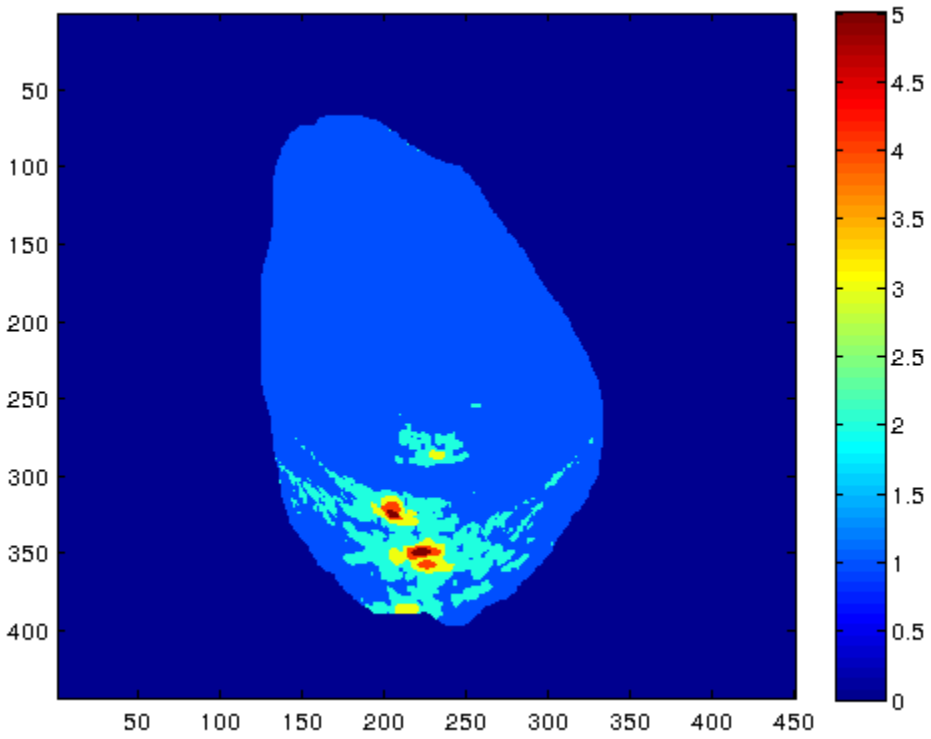
Loading tissue types

```
disp('loading GMM tissue types');
tumorlabel  = load_untouch_nii(labelfilename );

materialID = int32(tumorlabel.img);
if (size(materialID ,3) == 1) % store 2d image twice
   force3d = zeros (size(materialID ,1), size(materialID ,2), 2,'int32');
   force3d(:,:,1) = materialID;
   force3d(:,:,2) = materialID;
   materialID = force3d;
end
```

```
loading GMM tissue types
```

Initial Guess for volume fractions

```
ntissue = max(materialID(:));

[npixelx, npixely, npixelz] = size(materialID);
spacingX = tumorlabel.hdr.dime.pixdim(2)*1.e-3;
spacingY = tumorlabel.hdr.dime.pixdim(3)*1.e-3;
spacingZ = tumorlabel.hdr.dime.pixdim(4)*1.e-3;

idslice = 1;
handle1 = figure(2*NWavelength+1);
imagesc(materialID(:,:,idslice ),[0 5])
colorbar
```



Create distance map for each tissue type

```
labelbase = strsplit(labelfilename,'.');
for iii = 1:ntissue
    distancecmd = ['/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/
    disp(distancecmd );
end
```

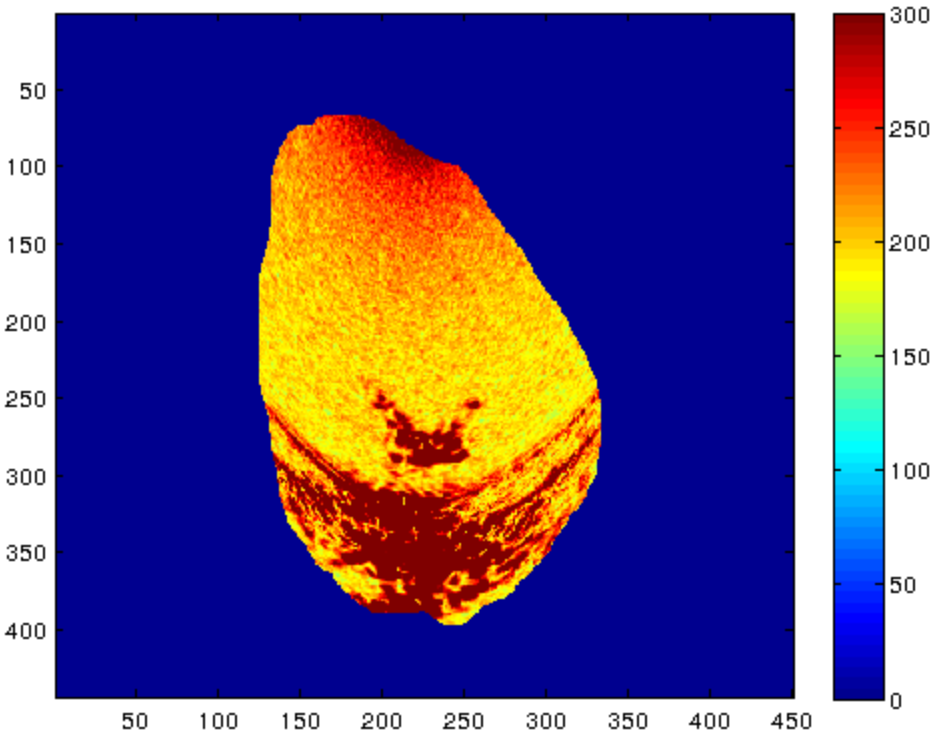
```
/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/ImageMath 3
/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/ImageMath 3
/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/ImageMath 3
```

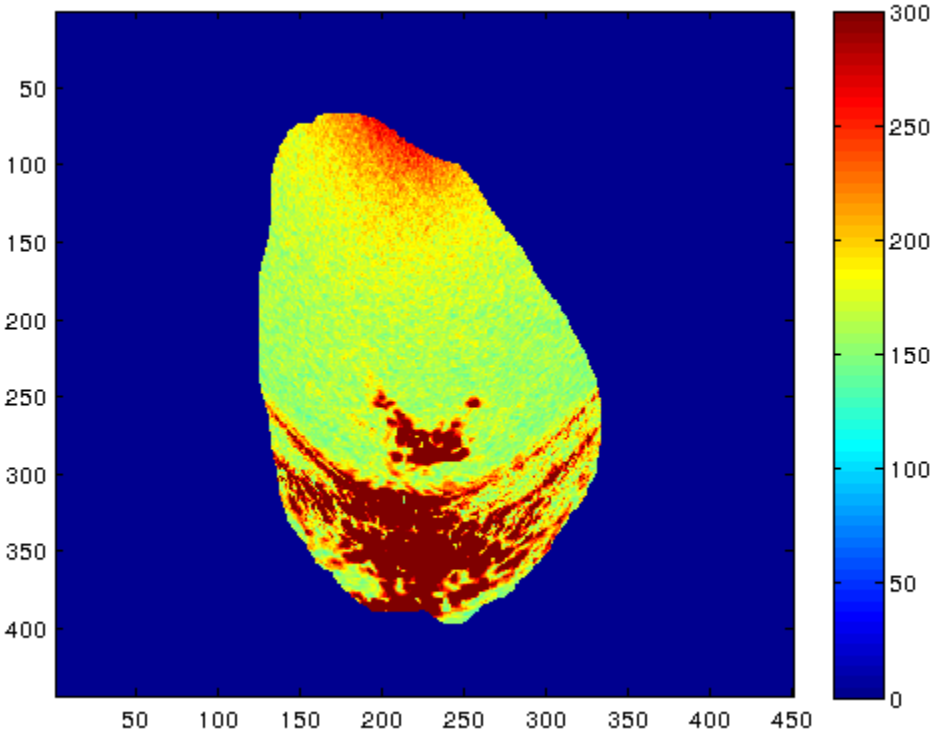
```
/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/ImageMath 3
/opt/apps/ANTsR/dev//ANTsR_src/ANTsR/src/ANTS/ANTS-build//bin/ImageMath 3
```

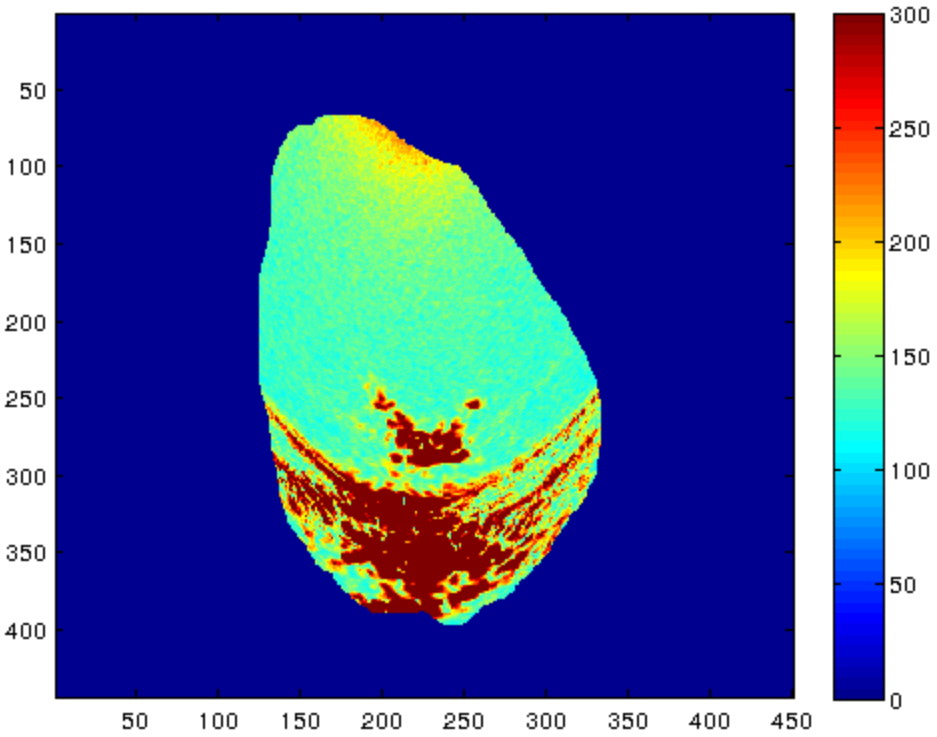
load tumor mask

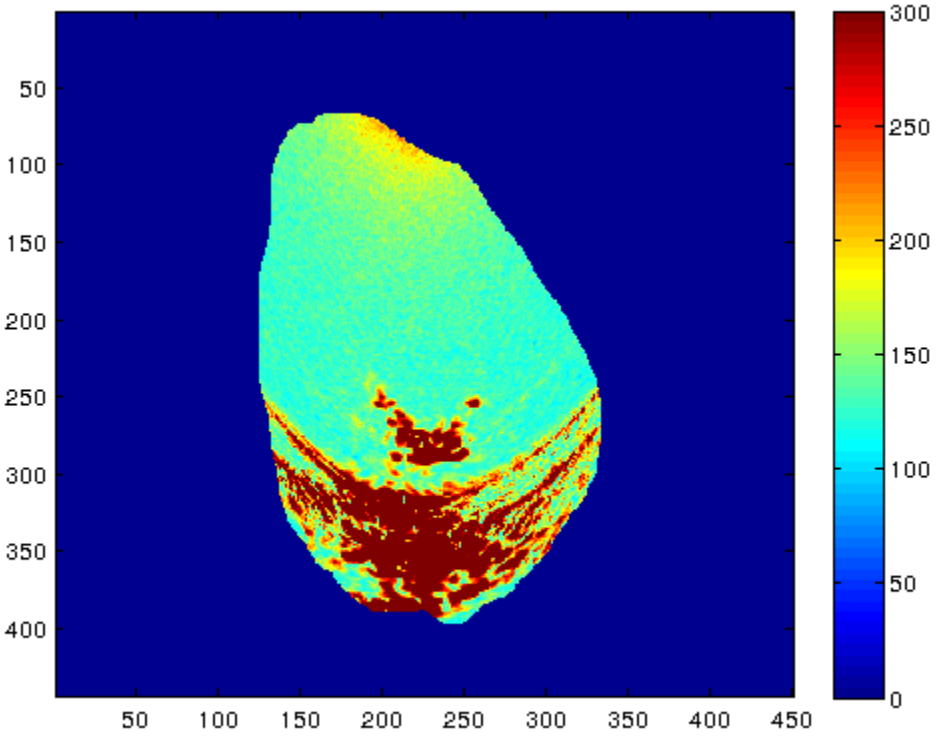
```
tumormask =load_untouch_nii(maskfilename);
maskimage = double(tumormask.img);
if (size(maskimage,3) == 1) % store 2d image twice
   force3d = zeros (size(maskimage ,1), size(maskimage ,2), 2);
   force3d(:,:,1) = maskimage;
   force3d(:,:,2) = maskimage;
   maskimage = force3d;
end

PAPlotRange = [0 300];
% load data
disp('loading PA data');
h_PAData = zeros(npixelx* npixely* npixelz,NWavelength);
for idwavelength = 1:NWavelength
  padatanii = load_untouch_nii([pafilebase '.' sprintf('%04d',idwavelength) '.nii.
  paimage   = double(padatanii.img);
  paimage(isnan(paimage))=0;
  if (size(paimage,3) == 1) % store 2d image twice
     force3d = zeros (size(paimage ,1), size(paimage ,2), 2);
     force3d(:,:,1) = paimage;
     force3d(:,:,2) = paimage;
     paimage = force3d;
  end

  % view data
  handle = figure(idwavelength);
  imagesc(maskimage(:,:,idslice).*paimage(:,:,idslice ),PAPlotRange )
  colorbar
  % store data array
  h_PAData(:,idwavelength) = maskimage(:).*paimage(:) ;
end
```

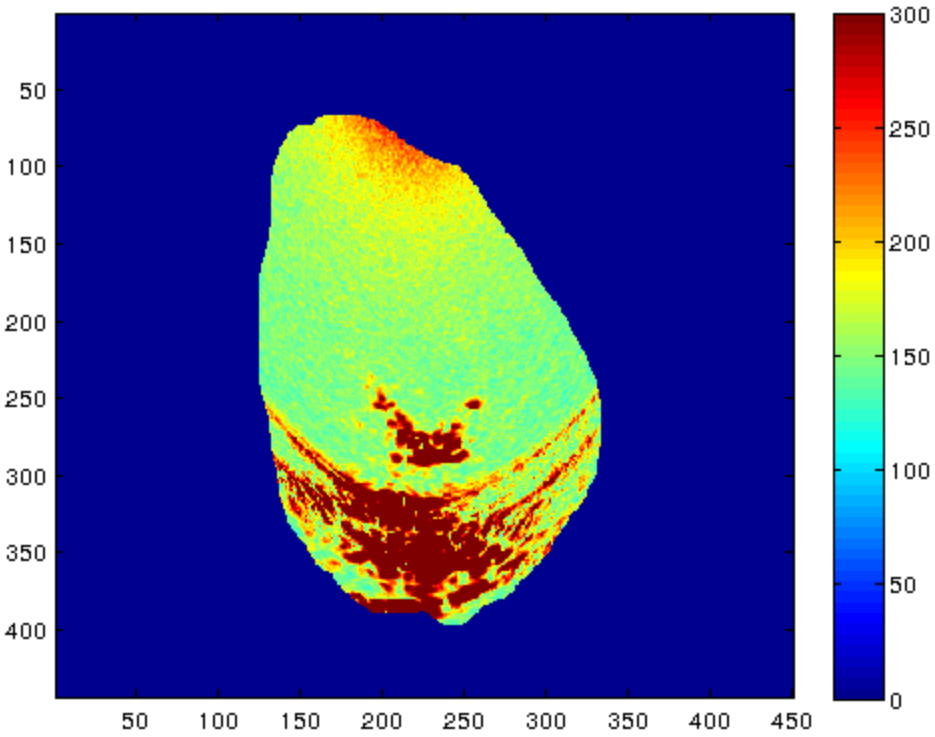
```
loading PA data
```

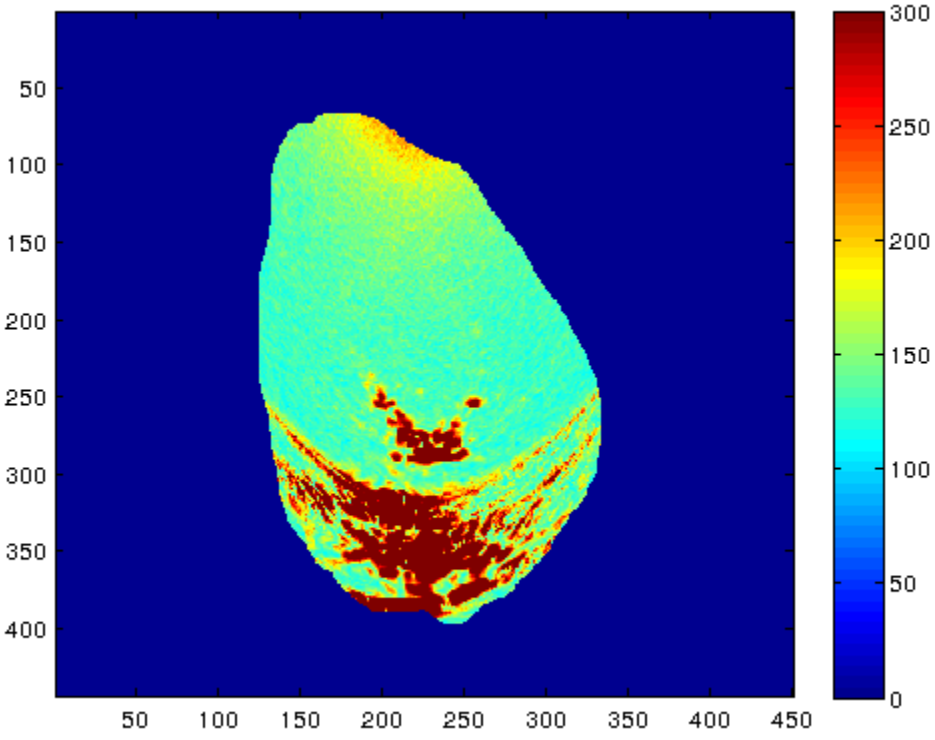

50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0

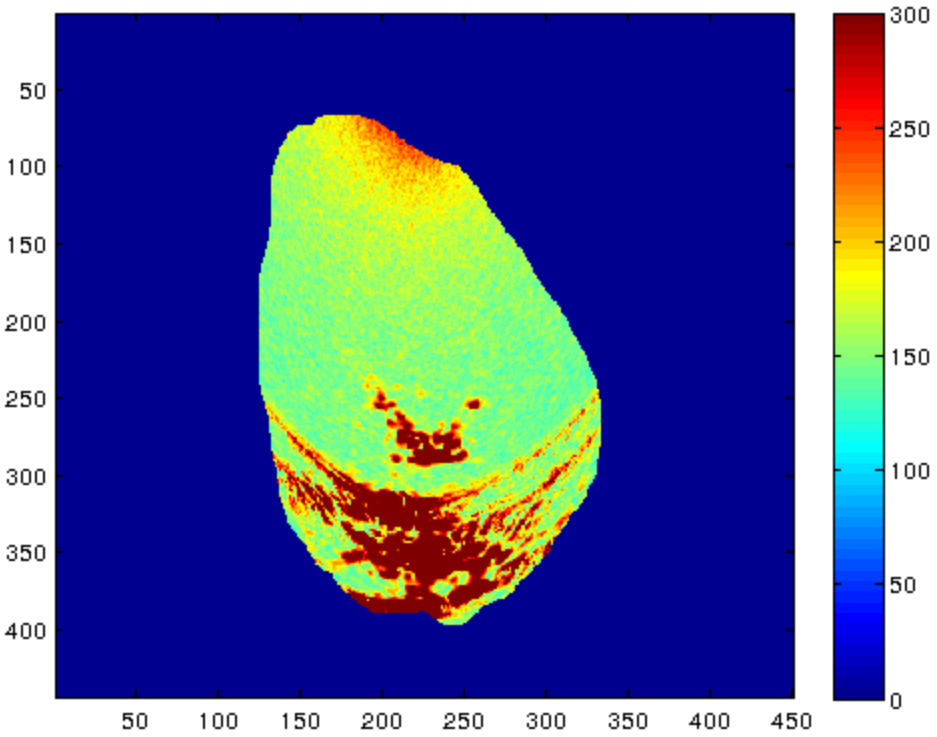

50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0

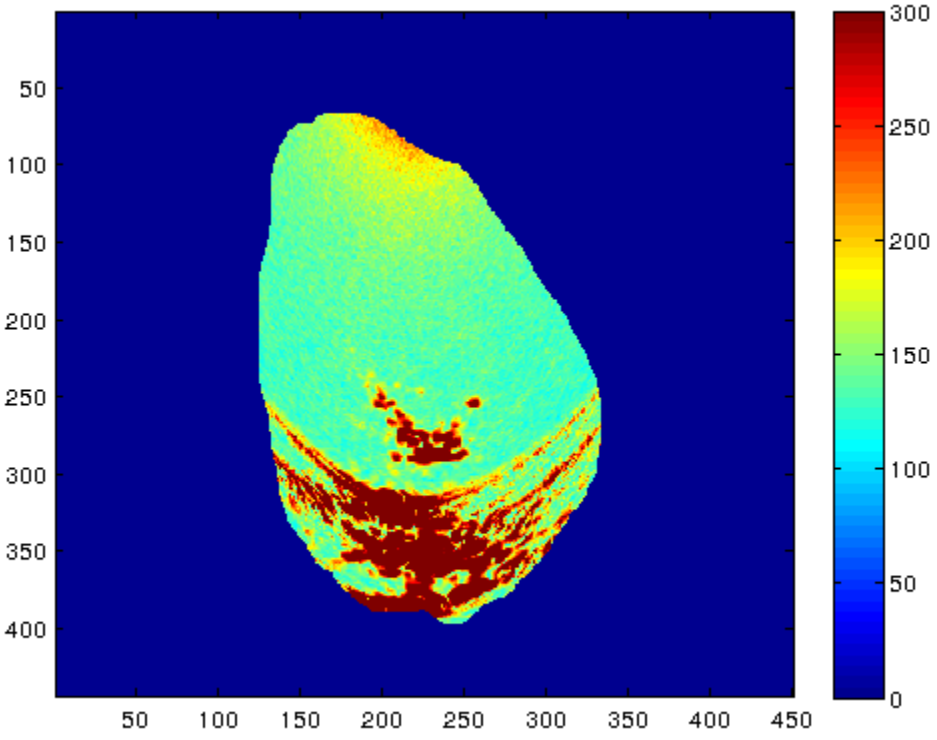

50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0


50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0


50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0


50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0


50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0


50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400 450
300
250
200
150
100
50
0

Get laser source locations

```
lasersource  = load_untouch_nii(setupfilename );
sourceimage  = double(lasersource.img);
if (size(sourceimage,3) == 1) % store 2d image twice
   force3d = zeros (size(sourceimage ,1), size(sourceimage ,2), 2);
   force3d(:,:,1) = sourceimage;
   force3d(:,:,2) = sourceimage;
   sourceimage = force3d;
end
[rows,cols,depth] = ind2sub(size(sourceimage),find(sourceimage));
nsource    = length(rows);
PowerLB    = .001;
PowerUB    =  .01;
PowerFnc   = @(x) PowerLB + x * (PowerUB-PowerLB);
```

Query the device

GPU must be reset on out of bounds errors reset(gpuDevice(1))

```
deviceInfo = gpuDevice(1);
numSMs = deviceInfo.MultiprocessorCount;
```

initialize data arrays

initialize on host and perform ONE transfer from host to device

```
tic;
h_pasource   = zeros(npixelx,npixely,npixelz);
d_pasource   = gpuArray( h_pasource  );
d_PAData     = gpuArray( h_PAData    );
d_materialID = gpuArray( materialID  );
d_maskimage  = gpuArray( maskimage   );
d_xloc       = gpuArray(1.e-3+ spacingX* rows );
d_yloc       = gpuArray(1.e-3+ spacingY* cols );
d_zloc       = gpuArray(1.e-3+ spacingZ* depth);
transfertime = toc;
disp(sprintf('transfer time to device %f',transfertime));
```

```
transfer time to device 0.014156
```

Compile and setup thread grid

grid stride loop design pattern, 1-d grid http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/

```
ssptx = parallel.gpu.CUDAKernel('sdaSpectralFluenceModel.ptx', 'sdaSpectralFluence
ssptx.GridSize =[numSMs*8 1];
threadsPerBlock= 768;
ssptx.ThreadBlockSize=[threadsPerBlock  1]
```

```
ssptx =

  CUDAKernel with properties:

      ThreadBlockSize: [768 1 1]
    MaxThreadsPerBlock: 768
              GridSize: [112 1 1]
    SharedMemorySize: 0
           EntryPoint: '_Z15sdaFluenceModelPKiPKdddidS2_S2_S2_Pddddiii'
    MaxNumLHSArguments: 1
       NumRHSArguments: 16
         ArgumentTypes: {1x16 cell}
```

create anonymous function

```
muaReference     = 3.e-4; % [1/m]
% TODO - change function signature to use struct
loss = @(x) FluenceModelObj([0,x(1:length(x)-2)],ssptx,d_pasource,x(length(x)),mua

% % tune kernel
% for blockpergrid = [numSMs*8,numSMs*16,numSMs*32,numSMs*48,numSMs*64];
% for threadsPerBlock = [128,256,512,768];
%   ssptx.GridSize=[blockpergrid 1];
%   ssptx.ThreadBlockSize=[threadsPerBlock  1];
%   tic;
%   f = loss(InitialGuess)
%   kernelruntime = toc;
%   disp(sprintf('blockpergrid=%d  threadsPerBlock=%d runtime=%f',blockpergrid,thr
% end
% end


% % Plot Runtimes
% NRuns = 100;
% runtimes = zeros(NRuns,1);
% tic;
% InitialGuess = [0.94868    0.96991      0.97169      0.98046       0.9873
% for iii = 1:NRuns
%   f = loss(InitialGuess);
%   runtimes(iii) = toc;
%   disp(sprintf('%d %12.5e',iii,runtimes(iii)));
% end
% handleRuntime = figure(2*NWavelength+2);
% plot([1:NRuns],runtimes)
% set(gca,'FontSize',16)
% xlabel('# of Function Evaluations')
% ylabel('Run time [s]')
% grid on
% legend('464x512x24 3D image' ,'Location','NorthWest')
% text(10,240,'GTX Titan - 2688 Cuda Cores - 4.5 Peak Teraflops')
% saveas(handleRuntime,'Runtimes','png')
```

run opt solver

```
disp('starting solver')
options = anneal();
options.MaxTries = Inf;
options.MaxConsRej = Inf;
options.StopTemp   = 1e-12;

% use least square direction for proposal distribution
% TODO - debug
% TODO - change function signature to use struct
options.Generator =  @(x) StochasticNewton([0,x(1:length(x)-1)],ssptx,d_pasource,m

% TODO - use Monte Carlo for now
options.Generator = @(x) rand(1,length(x));
options.Generator = @(x) (x+(randperm(length(x))==length(x))*randn/100);

% set plotting function
options.PlotLoss =  @(x) FluenceModelObj([0,x(1:length(x)-2)],ssptx,d_pasource,x(l

% uniformly search parameter space to find good initial guess
RandomInitialGuess = 10;
RandomInitialGuess = 1;
tic;
for iii = 1:RandomInitialGuess  % embarrasingly parallel on initial guess

  % initial guess
  % materialID[0] not used
  % assume 50/50 volume fraction initially
  % last entry is percent power
  InitialGuess = [0.80338    0.91354      0.95532      0.93679      0.88591
  InitialGuess = [.5*ones(1,ntissue),.6,.9];
  InitialGuess = [0.26718     0.2966       1.1601       2.2722       2.7978

  SolnVector   = InitialGuess;
  % uncomment to run optimizer
  %[SolnVector FunctionValue ] = anneal(loss,InitialGuess,options);
  % TODO store best solution
end
mcmcruntime = toc;
disp(sprintf('mcmc run time %f',mcmcruntime) );
```

```
starting solver
mcmc run time 0.000108
```

Plot Solution

```
f = options.PlotLoss(SolnVector)
```

```
Columns 1 through 6

         0      0.26718       0.2966       1.1601       2.2722       2.
```

```
  Columns 7 through 9

    0.030203       37.052        31.37

f =

        40.24
```

Published with MATLAB® R2013a