Example: Generate your own phantom geometry

Table of Contents

In this example we will show (i) how to create arbitrary ct data (resolution, ct numbers) (ii) how to create a cst structure containing the volume of interests of the phantom (iii) generate a treatment plan for this phantom

```
clc, clear, close all
```

Create a CT image series

```
xDim = 200;
yDim = 200;
zDim = 50;

ct.cubeDim      = [xDim yDim zDim];
ct.resolution.x = 2;
ct.resolution.y = 2;
ct.resolution.z = 3;
ct.numOfCtScen  = 1;

% create an ct image series with zeros - it will be filled later
ct.cubeHU{1} = ones(ct.cubeDim) * -1000; % assign HU of Air
```

Create the VOI data for the phantom

Now we define structures a contour for the phantom and a target

```
ixOAR = 1;
ixPTV = 2;

% define general VOI properties
cst{ixOAR,1} = 0;
cst{ixOAR,2} = 'contour';
cst{ixOAR,3} = 'OAR';

cst{ixPTV,1} = 1;
cst{ixPTV,2} = 'target';
cst{ixPTV,3} = 'TARGET';

% define optimization parameter for both VOIs
cst{ixOAR,5}.TissueClass = 1;
cst{ixOAR,5}.alphaX      = 0.1000;
cst{ixOAR,5}.betaX       = 0.0500;
cst{ixOAR,5}.Priority    = 2;
cst{ixOAR,5}.Visible     = 1;

% define objective as struct for compatibility with GNU Octave I/O
cst{ixOAR,6}{1} =
 struct(DoseObjectives.matRad_SquaredOverdosing(10,30));

cst{ixPTV,5}.TissueClass = 1;
cst{ixPTV,5}.alphaX      = 0.1000;
cst{ixPTV,5}.betaX       = 0.0500;
cst{ixPTV,5}.Priority    = 1;
cst{ixPTV,5}.Visible     = 1;

% define objective as struct for compatibility with GNU Octave I/O
cst{ixPTV,6}{1} =
 struct(DoseObjectives.matRad_SquaredOverdosing(10,30));
```

Lets create either a cubic or a spheric phantom

```
TYPE = 'spheric';   % either 'cubic' or 'spheric'

% first the OAR
cubeHelper = zeros(ct.cubeDim);

switch TYPE

   case {'cubic'}

      xLowOAR  = round(xDim/2 - xDim/4);
      xHighOAR = round(xDim/2 + xDim/4);
      yLowOAR  = round(yDim/2 - yDim/4);
      yHighOAR = round(yDim/2 + yDim/4);
```

```
        zLowOAR  = round(zDim/2 - zDim/4);
        zHighOAR = round(zDim/2 + zDim/4);

        for x = xLowOAR:1:xHighOAR
           for y = yLowOAR:1:yHighOAR
              for z = zLowOAR:1:zHighOAR
                 cubeHelper(x,y,z) = 1;
              end
           end
        end

     case {'spheric'}

        radiusOAR = xDim/4;

        for x = 1:xDim
           for y = 1:yDim
              for z = 1:zDim
                 currPost = [x y z] - round([ct.cubeDim./2]);
                 if  sqrt(sum(currPost.^2)) < radiusOAR
                    cubeHelper(x,y,z) = 1;
                 end
              end
           end
        end

  end

  % extract the voxel indices and save it in the cst
  cst{ixOAR,4}{1} = find(cubeHelper);


  % second the PTV
  cubeHelper = zeros(ct.cubeDim);

  switch TYPE

     case {'cubic'}

        xLowPTV  = round(xDim/2 - xDim/8);
        xHighPTV = round(xDim/2 + xDim/8);
        yLowPTV  = round(yDim/2 - yDim/8);
        yHighPTV = round(yDim/2 + yDim/8);
        zLowPTV  = round(zDim/2 - zDim/8);
        zHighPTV = round(zDim/2 + zDim/8);

        cubeHelper = zeros(ct.cubeDim);

        for x = xLowPTV:1:xHighPTV
           for y = yLowPTV:1:yHighPTV
              for z = zLowPTV:1:zHighPTV
                 cubeHelper(x,y,z) = 1;
              end
           end
```

```
        end

    case {'spheric'}

        radiusPTV = xDim/12;

        for x = 1:xDim
            for y = 1:yDim
                for z = 1:zDim
                    currPost = [x y z] - round([ct.cubeDim./2]);
                    if  sqrt(sum(currPost.^2)) < radiusPTV
                        cubeHelper(x,y,z) = 1;
                    end
                end
            end
        end

end



% extract the voxel indices and save it in the cst
cst{ixPTV,4}{1} = find(cubeHelper);


% now we have ct data and cst data for a new phantom
display(ct);
display(cst);


ct =

  struct with fields:

        cubeDim: [200 200 50]
     resolution: [1×1 struct]
    numOfCtScen: 1
         cubeHU: {[200×200×50 double]}


cst =

  2×6 cell array

  Columns 1 through 5

    {[0]}    {'contour'}    {'OAR'   }    {1×1 cell}    {1×1 struct}
    {[1]}    {'target' }    {'TARGET'}    {1×1 cell}    {1×1 struct}

  Column 6

    {1×1 cell}
    {1×1 cell}
```

Assign relative electron densities

```
vIxOAR = cst{ixOAR,4}{1};
vIxPTV = cst{ixPTV,4}{1};

ct.cubeHU{1}(vIxOAR) = 1;
ct.cubeHU{1}(vIxPTV) = 1;
```

Treatment Plan

The next step is to define your treatment plan labeled as 'pln'. This structure requires input from the treatment planner and defines the most important cornerstones of your treatment plan.

First of all, we need to define what kind of radiation modality we would like to use. Possible values are photons, protons or carbon. In this example we would like to use photons for treatment planning. Next, we need to define a treatment machine to correctly load the corresponding base data. matRad features generic base data in the file 'photons_Generic.mat'; consequently the machine has to be set to 'Generic'

```
pln.radiationMode = 'photons';
pln.machine       = 'Generic';
```

Define the flavor of biological optimization for treatment planning along with the quantity that should be used for optimization. Possible values are (none: physical dose based optimization; const_RBExD: constant RBE of 1.1; LEMIV_effect: effect-based optimization; LEMIV_RBExD: optimization of RBE-weighted dose. As we use photons, we select 'none' as we want to optimize the physical dose.

```
pln.propOpt.bioOptimization = 'none';
```

The remaining plan parameters are set like in the previous example files

```
pln.numOfFractions        = 30;
pln.propStf.gantryAngles  = [0 45];
pln.propStf.couchAngles   = [0 0];
pln.propStf.bixelWidth    = 5;
pln.propStf.numOfBeams    = numel(pln.propStf.gantryAngles);
pln.propStf.isoCenter     = ones(pln.propStf.numOfBeams,1) *
 matRad_getIsoCenter(cst,ct,0);
pln.propOpt.runDAO        = 0;
pln.propOpt.runSequencing = 0;

% dose calculation settings
pln.propDoseCalc.doseGrid.resolution.x = 3; % [mm]
pln.propDoseCalc.doseGrid.resolution.y = 3; % [mm]
pln.propDoseCalc.doseGrid.resolution.z = 3; % [mm]
```

Generate Beam Geometry STF

```
stf = matRad_generateStf(ct,cst,pln);
```

```
matRad: Generating stf struct... Warning: Could not find HLUT  in
 hlutLibrary folder. matRad default HLUT loaded
Progress: 100.00 %
```

Dose Calculation

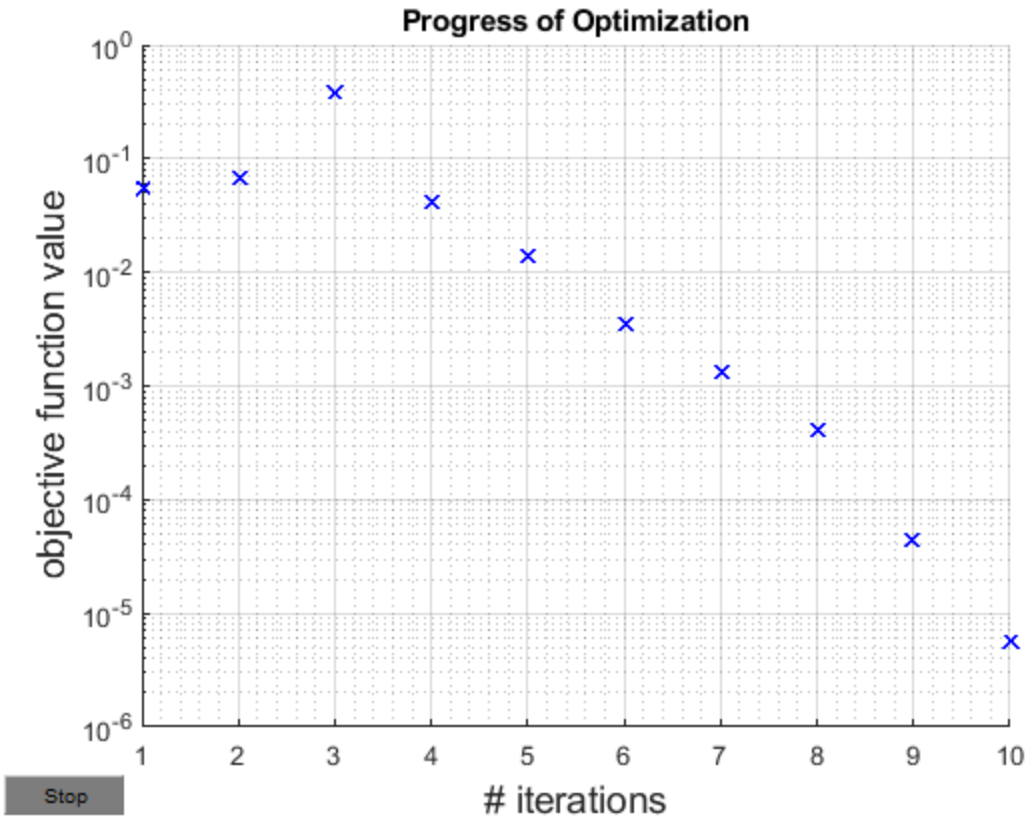
```
dij = matRad_calcPhotonDose(ct,stf,pln,cst);

Warning: Could not find HLUT  in hlutLibrary folder. matRad default
 HLUT loaded
matRad: Photon dose calculation...
Beam 1 of 2:
matRad: calculate radiological depth cube...done.
                   SSD = 900mm
matRad: Uniform primary photon fluence -> pre-compute kernel
 convolution for SSD = 900 mm ...
Progress: 100.00 %
Beam 2 of 2:
matRad: calculate radiological depth cube...done.
                   SSD = 901mm
matRad: Uniform primary photon fluence -> pre-compute kernel
 convolution for SSD = 901 mm ...
Progress: 100.00 %
```

Inverse Optimization for intensity-modulated photon therapy

The goal of the fluence optimization is to find a set of bixel/spot weights which yield the best possible dose distribution according to the clinical objectives and constraints underlying the radiation treatment.

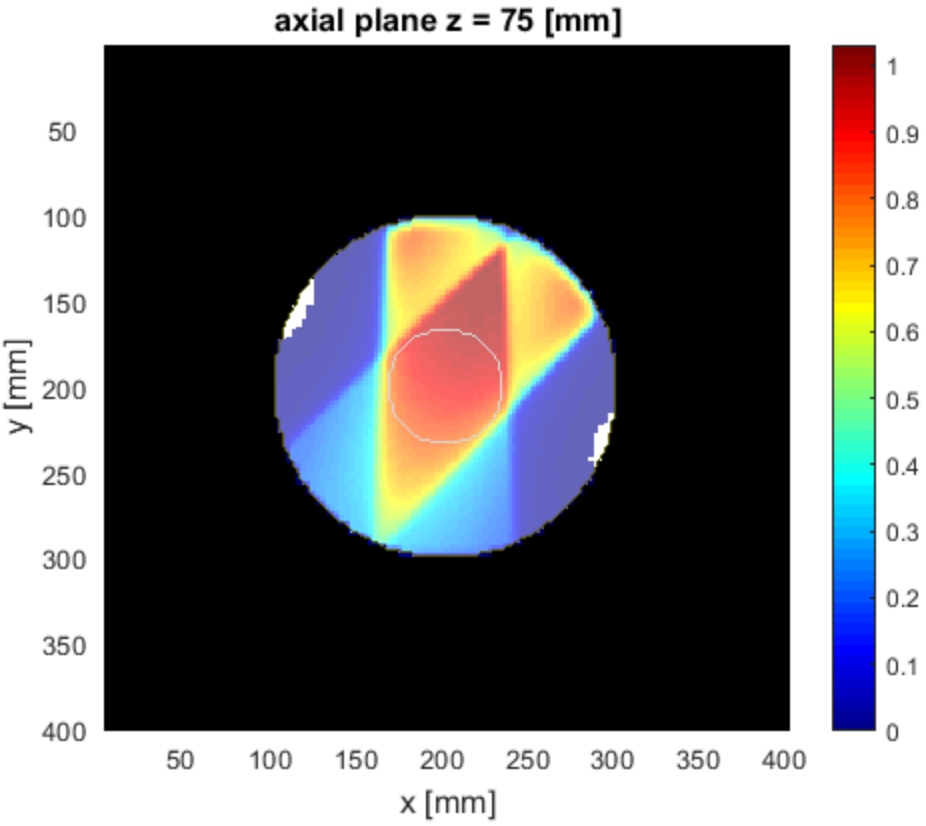
```
resultGUI = matRad_fluenceOptimization(dij,cst,pln);


Optimzation initiating...
Press q to terminate the optimization...
This is Ipopt version 3.11.8, running with linear solver ma57.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:      534
                     variables with only lower bounds:      534
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du
 alpha_pr  ls
   0 5.5566318e-002 0.00e+000 1.00e+000   0.0 0.00e+000    -  0.00e
+000 0.00e+000   0
```

```
   1 6.8752560e-002 0.00e+000 1.01e-002  -1.8 1.01e-002    -  1.00e
+000 1.00e+000f  1
   2 3.8835600e-001 0.00e+000 9.31e-003  -2.2 2.18e-001    -  1.00e
+000 1.00e+000f  1
   3 4.1226749e-002 0.00e+000 3.59e-003  -3.3 1.78e-001    -  1.00e
+000 1.00e+000f  1
   4 1.3859637e-002 0.00e+000 2.53e-003  -9.0 4.44e-002    -  1.00e
+000 1.00e+000f  1
   5 3.5600767e-003 0.00e+000 1.01e-003 -10.3 4.87e-002    -  1.00e
+000 1.00e+000f  1
   6 1.3435916e-003 0.00e+000 5.19e-004  -7.8 3.36e-002    -  1.00e
+000 1.00e+000f  1
   7 4.0393343e-004 0.00e+000 2.59e-004  -9.2 3.59e-002    -  1.00e
+000 1.00e+000f  1
   8 4.3910905e-005 0.00e+000 7.23e-005 -11.0 3.84e-002    -  1.00e
+000 1.00e+000f  1
   9 5.5991040e-006 0.00e+000 2.22e-005 -11.0 1.56e-002    -  1.00e
+000 1.00e+000f  1

Number of Iterations....: 9

                                   (scaled)                 (unscaled)
Objective...............:  5.5991040273938964e-006
 5.5991040273938964e-006
Dual infeasibility......:  2.2197775649255288e-005
 2.2197775649255288e-005
Constraint violation....:  0.0000000000000000e+000
 0.0000000000000000e+000
Complementarity.........:  5.6581367762155894e-011
 5.6581367762155894e-011
Overall NLP error.......:  2.2197775649255288e-005
 2.2197775649255288e-005


Number of objective function evaluations             = 10
Number of objective gradient evaluations             = 10
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      1.202
Total CPU secs in NLP function evaluations           =      1.153

EXIT: Solved To Acceptable Level.
```



Plot the resulting dose slice

```
plane      = 3;
slice      = round(pln.propStf.isoCenter(1,3)./ct.resolution.z);
doseWindow = [0 max([resultGUI.physicalDose(:)])];

figure,title('phantom plan')
matRad_plotSliceWrapper(gca,ct,cst,1,resultGUI.physicalDose,plane,slice,
[],[],colorcube,[],doseWindow,[]);
```



Published with MATLAB® R2018a