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;
cst{ixOAR,6}.type        = 'square overdosing';
cst{ixOAR,6}.dose        = 30;
cst{ixOAR,6}.penalty     = 10;
cst{ixOAR,6}.EUD         = NaN;
cst{ixOAR,6}.volume      = NaN;
cst{ixOAR,6}.coverage    = NaN;
cst{ixOAR,6}.robustness  = 'none';

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;
cst{ixPTV,6}.type        = 'square deviation';
cst{ixPTV,6}.dose        = 60;
cst{ixPTV,6}.penalty     = 50;
cst{ixPTV,6}.EUD         = NaN;
cst{ixPTV,6}.volume      = NaN;
cst{ixPTV,6}.coverage    = NaN;
cst{ixPTV,6}.robustness  = 'none';
```

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 struct}
    {1×1 struct}
```

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;
```

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 = 901mm
matRad: Uniform primary photon fluence -> pre-compute kernel
 convolution for SSD = 901 mm ...
Progress: 100.00 %
Beam 2 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 %
```

Inverse Optimization for intensity-modulated photon therapy

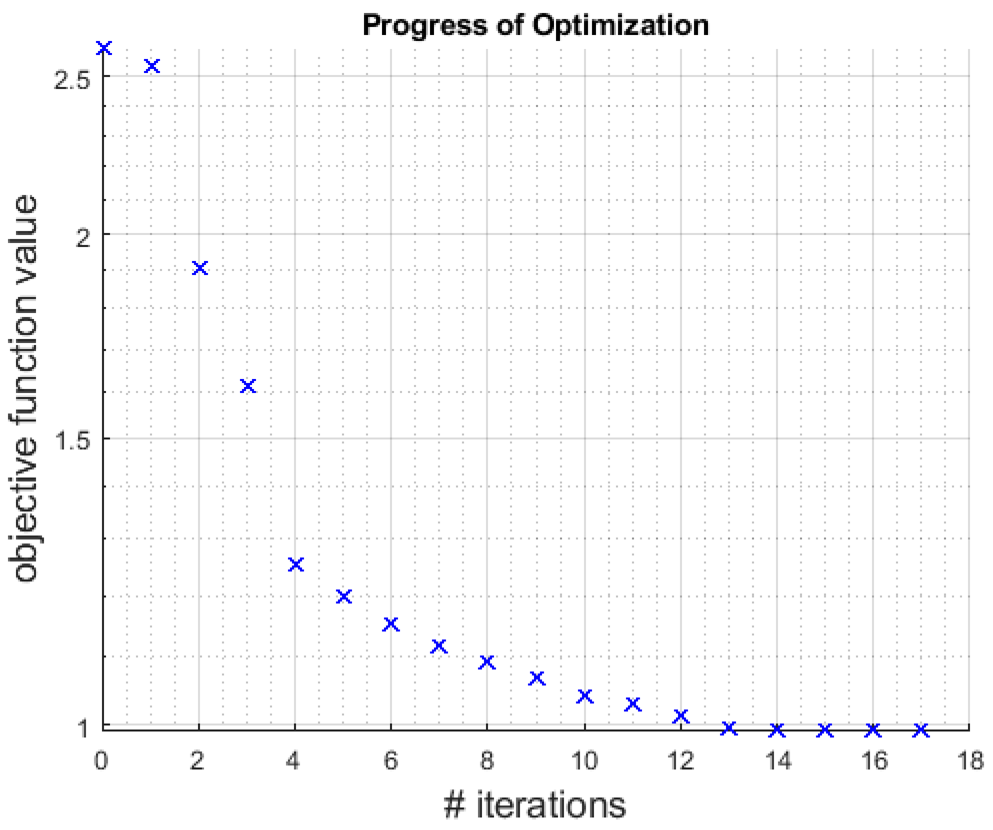
The goal of the fluence optimization is to find a set of bixel/spot weights which yield the best possible dose distribution according to the clinical objectives and constraints underlying the radiation treatment.

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

```

Optimzation initiating...
Press q to terminate the optimization...

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear
 optimization.
 Ipopt is released as open source code under the Eclipse Public
 License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

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 2.5998464e+000 0.00e+000 1.02e+000   0.0 0.00e+000    -  0.00e
+000 0.00e+000   0
   1 2.5357843e+000 0.00e+000 2.83e-002  -1.4 3.24e-002    -
 9.99e-001 1.00e+000f  1
   2 1.9092313e+000 0.00e+000 4.52e-002  -2.7 2.07e-001    -
 9.99e-001 1.00e+000f  1
   3 1.6142827e+000 0.00e+000 2.81e-002  -3.5 1.08e-001    -
 9.94e-001 1.00e+000f  1
   4 1.2545654e+000 0.00e+000 1.96e-002  -4.3 3.69e-001    -  1.00e
+000 1.00e+000f  1
   5 1.1976709e+000 0.00e+000 2.20e-002  -4.6 1.50e-001    -
 9.99e-001 1.00e+000f  1
   6 1.1517353e+000 0.00e+000 8.72e-003  -5.7 4.48e-002    -  1.00e
+000 1.00e+000f  1
   7 1.1162275e+000 0.00e+000 6.55e-003  -6.8 1.01e-001    -  1.00e
+000 1.00e+000f  1
   8 1.0932738e+000 0.00e+000 8.62e-003  -8.1 1.05e-001    -  1.00e
+000 1.00e+000f  1
   9 1.0687354e+000 0.00e+000 2.02e-002  -8.9 2.81e-001    -  1.00e
+000 1.00e+000f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du
 alpha_pr  ls
  10 1.0407284e+000 0.00e+000 1.04e-002  -9.8 1.50e-001    -  1.00e
+000 1.00e+000f  1
  11 1.0295842e+000 0.00e+000 7.99e-003  -9.7 3.43e-002    -  1.00e
+000 1.00e+000f  1
  12 1.0108424e+000 0.00e+000 7.89e-003 -10.5 1.00e-001    -  1.00e
+000 1.00e+000f  1
  13 9.9503865e-001 0.00e+000 1.50e-002 -11.0 2.23e-001    -  1.00e
+000 1.00e+000f  1
  14 9.9074286e-001 0.00e+000 1.38e-002 -11.0 1.85e-001    -  1.00e
+000 1.89e-001f  1
  15 9.9069925e-001 0.00e+000 1.38e-002 -11.0 3.40e-001    -  1.00e
+000 1.02e-003f  1
  16 9.9069723e-001 0.00e+000 3.16e-002 -11.0 2.53e-001    -  1.00e
+000 4.42e-005f  1
  17 9.9017677e-001 0.00e+000 2.48e-002 -11.0 3.73e-001    -  1.00e
+000 8.15e-003f  1

Number of Iterations....: 17

                                   (scaled)                 (unscaled)
Objective...............:  9.9017677048753372e-001
 9.9017677048753372e-001
Dual infeasibility......:  2.4847495822923300e-002
 2.4847495822923300e-002
Constraint violation....:  0.0000000000000000e+000
 0.0000000000000000e+000
```

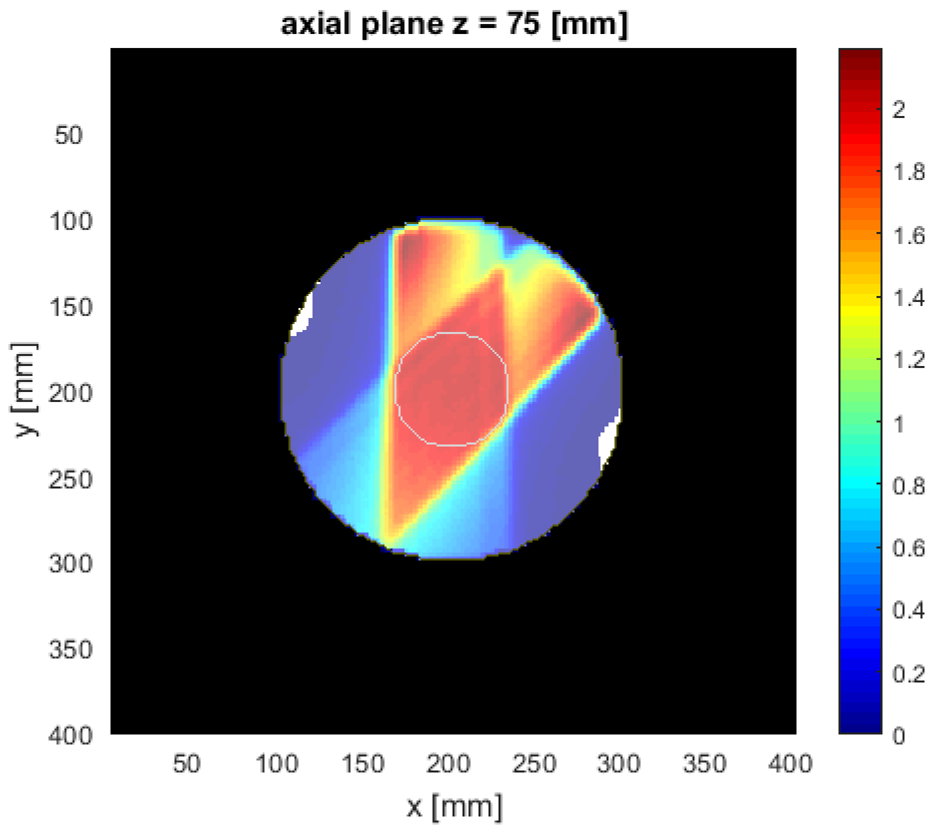
```
Complementarity.........:  2.0831495922667587e-007
 2.0831495922667587e-007
Overall NLP error.......:  2.4847495822923300e-002
 2.4847495822923300e-002


Number of objective function evaluations             = 18
Number of objective gradient evaluations             = 18
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)   =      1.273
Total CPU secs in NLP function evaluations           =      2.456

EXIT: Solved To Acceptable Level.
Calculating final cubes...
```



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