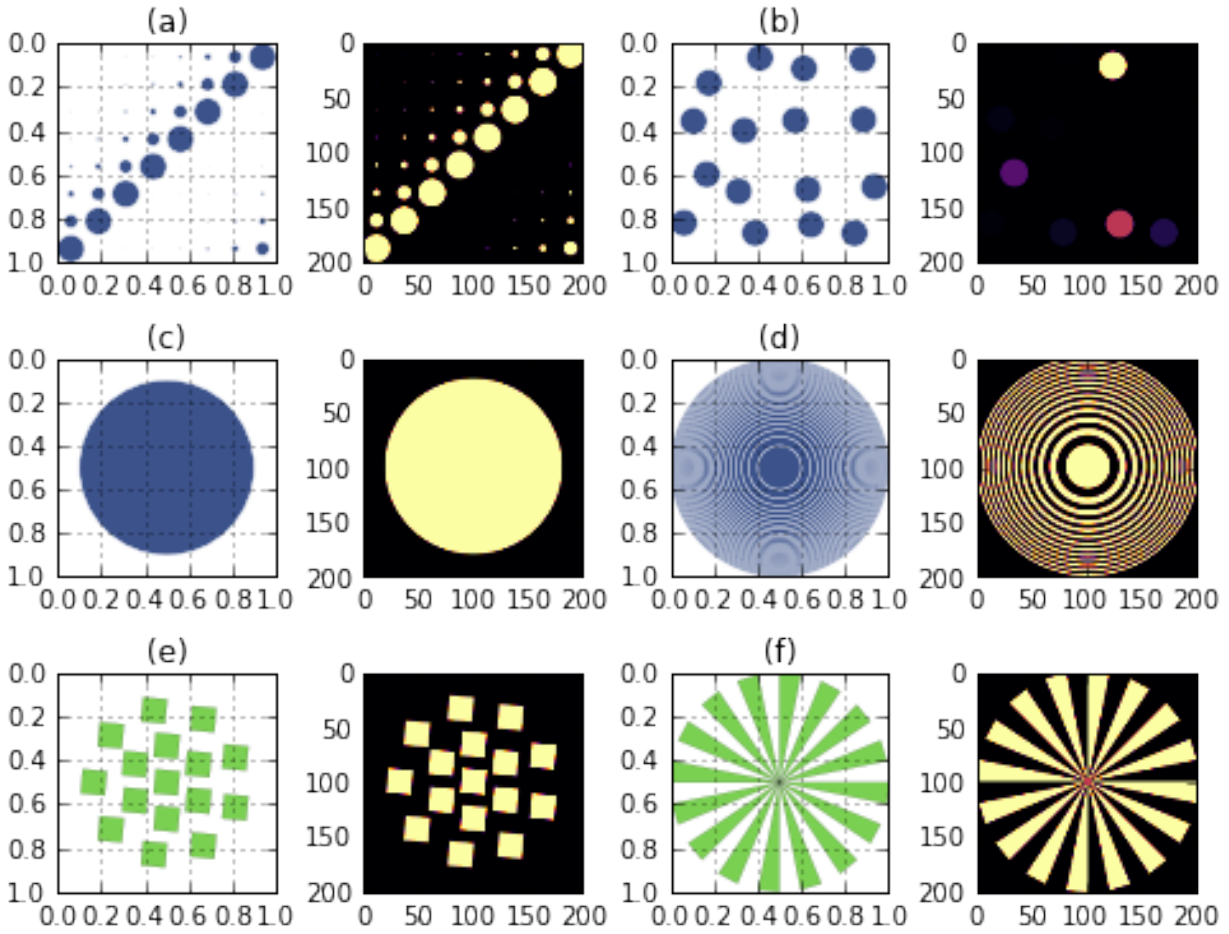

XDesign Documentation

Release 0.1

2016, UChicago Argonne, LLC

December 03, 2016

1	API	3
2	Examples	23
3	References	53
4	Features	55
5	Contribute	57
6	License	59
7	Indices and tables	61
	Bibliography	63
	Python Module Index	67



XDesign is an open-source Python package for generating configurable simulation phantoms for benchmarking tomographic image reconstruction.

XDesign Modules:

1.1 `xdesign.acquisition`

Functions:

<code>Beam(p1, p2[, size])</code>	Beam (thick line) in 2-D cartesian space.
<code>Probe(p1, p2[, size])</code>	
<code>sinogram(sx, sy, phantom[, noise])</code>	Generates sinogram given a phantom.
<code>angleogram(sx, sy, phantom[, noise])</code>	Generates angleogram given a phantom.
<code>raster_scan(sx, sy)</code>	Provides a beam list for raster-scanning.
<code>angle_scan(sx, sy)</code>	Provides a beam list for raster-scanning.

class `xdesign.acquisition.Beam` (*p1, p2, size=0*)

Bases: `xdesign.geometry.Line`

Beam (thick line) in 2-D cartesian space.

It is defined by two distinct points.

p1

Point

p2

Point

size

scalar, optional

Size of the beam.

half_space

Returns the half space polytope respresentation of the infinite beam.

class `xdesign.acquisition.Probe` (*p1, p2, size=0*)

Bases: `xdesign.acquisition.Beam`

measure (*phantom, noise=False*)

Return the probe measurement given phantom. When noise is > 0, poisson noise is added to the returned measurement.

record()

translate(dx)

Translates beam along its normal direction.

`xdesign.acquisition.sinogram(sx, sy, phantom, noise=False)`

Generates sinogram given a phantom.

Parameters

- **sx** (*int*) – Number of rotation angles.
- **sy** (*int*) – Number of detection pixels (or sample translations).
- **phantom** (*Phantom*)

Returns *ndarray* – Sinogram.

`xdesign.acquisition.angleogram(sx, sy, phantom, noise=False)`

Generates angleogram given a phantom.

Parameters

- **sx** (*int*) – Number of rotation angles.
- **sy** (*int*) – Number of detection pixels (or sample translations).
- **phantom** (*Phantom*)

Returns *ndarray* – Angleogram.

`xdesign.acquisition.raster_scan(sx, sy)`

Provides a beam list for raster-scanning.

Parameters

- **sx** (*int*) – Number of rotation angles.
- **sy** (*int*) – Number of detection pixels (or sample translations).

Yields *Probe*

`xdesign.acquisition.angle_scan(sx, sy)`

Provides a beam list for raster-scanning.

Parameters

- **sx** (*int*) – Number of rotation angles.
- **sy** (*int*) – Number of detection pixels (or sample translations).

Yields *Probe*

1.2 xdesign.algorithms

Functions:

<code>reconstruct</code> (probe, shape, data, rec)	Reconstruct single x-ray beam data.
<code>art</code> (probe, data, init[, niter])	Reconstruct data.
<code>sirt</code> (probe, data, init[, niter])	Reconstruct data.
<code>mlem</code> (probe, data, init[, niter])	Reconstruct data.
<code>stream</code> (probe, data, init)	Reconstruct data.

`xdesign.algorithms.reconstruct` (*probe, shape, data, rec*)
Reconstruct single x-ray beam data.

Parameters

- **probe** (*Probe*)
- **shape** (*list*) – shape of the reconstruction grid.
- **data** (*scalar*) – Probe measurement data.
- **rec** (*ndarray*) – Initialization matrix.

`xdesign.algorithms.art` (*probe, data, init, niter=10*)
Reconstruct data.

`xdesign.algorithms.sirt` (*probe, data, init, niter=10*)
Reconstruct data.

`xdesign.algorithms.mlem` (*probe, data, init, niter=10*)
Reconstruct data.

`xdesign.algorithms.stream` (*probe, data, init*)
Reconstruct data.

`xdesign.algorithms.update_progress` (*progress*)
Draws a process bar in the terminal.

Parameters *process* (*float*) – The percentage completed e.g. 0.10 for 10%

1.3 xdesign.feature

Functions:

Feature(*geometry*[, *mass_atten*]) A geometric region(s) and associated materials property(ies).

class `xdesign.feature.Feature` (*geometry, mass_atten=1*)
Bases: `object`

A geometric region(s) and associated materials property(ies). Properties and geometry can be manipulated from this level, but rendering must access the geometry directly.

geometry

Entity

Defines a region where the properties are valid.

mass_atten

scalar

The mass attenuation coefficient of the Feature.

add_property (*name, function*)

Adds a property by name to the Feature. NOTE: Properties added here are not cached because they are probably never called with the same parameters ever or the property is static so it doesn't need caching.

area

Returns the total surface area of the feature

center

Returns the centroid of the feature.

radius

Returns the radius of the smallest boundary circle

rotate (*theta*, *p*, *axis=None*)

Rotate feature geometry around a line. Rotating property functions is not supported.

translate (*x*, *y*)

Translate feature geometry. Translating property functions is not supported.

volume

Returns the volume of the feature

1.4 xdesign.geometry

Functions:

<i>Point</i> (x)	A point in ND cartesian space.
Superellipse(center, a, b, n)	Superellipse in 2-D cartesian space.
Ellipse(center, a, b)	Ellipse in 2-D cartesian space.
<i>Circle</i> (center, radius)	Circle in 2-D cartesian space.
<i>Line</i> (p1, p2)	Line in 2-D cartesian space.
<i>Triangle</i> (p1, p2, p3)	Triangle in 2-D cartesian space.
<i>Rectangle</i> (p1, p2, p3, p4)	Rectangle in 2-D cartesian space.
<i>Square</i> (center, side_length)	Square in 2-D cartesian space.

class xdesign.geometry.**Entity**

Bases: `object`

Base class for all geometric entities. All geometric entities should have these methods.

collision (*other*)

Returns True if this entity collides with another entity.

contains (*point*)

Returns True if the point(s) is within the bounds of the entity. point is either a Point or an N points listed as an NxN array.

dim

The dimensionality of the entity

distance (*other*)

Return the closest distance between entities.

midpoint (*other*)

Return the midpoint between entities.

rotate (*theta*, *point=None*, *axis=None*)

Rotate entity around an axis which passes through a point by theta radians.

scale (*vector*)

Scale entity in each dimension according to vector. Scaling is centered on the origin.

translate (*vector*)

Translate entity along vector.

```
class xdesign.geometry.Point(x)
    Bases: xdesign.geometry.Entity
    A point in ND cartesian space.

    _x
        1darray
        ND coordinates of the point.

    x
        scalar
        dimension 0 of the point.

    y
        scalar
        dimension 1 of the point.

    z
        scalar
        dimension 2 of the point.

    norm
        scalar
        The L2/euclidian norm of the point.

    collision (other)
        Returns True if this entity collides with another entity.

    contains (other)
        Returns True if the other is within the bounds of the entity.

    dim

    distance (other)
        Return the closest distance between entities.

    norm
        Reference: http://stackoverflow.com/a/23576322

    rotate (theta, point=None, axis=None)
        Rotate entity around an axis by theta radians.

    scale (vector)
        Scale entity in each dimension according to vector. Scaling is centered on the origin.

    translate (vector)
        Translate point along vector.

    x

    y

    z

class xdesign.geometry.Circle(center, radius)
    Bases: xdesign.geometry.Curve
    Circle in 2-D cartesian space.

    center
        Point
```

Defines the center point of the circle.

radius

scalar

Radius of the circle.

area

Return area.

circumference

Return circumference.

contains (*points*)

diameter

Return diameter.

list

Return list representation for saving to files.

patch

scale (*val*)

Scale.

class `xdesign.geometry.Line` (*p1*, *p2*)

Bases: `xdesign.geometry.LinearEntity`

Line in 2-D cartesian space.

It is defined by two distinct points.

p1

Point

p2

Point

distance (*other*)

Return the closest distance between entities. REF: http://geomalgorithms.com/a02-_lines.html

intercept (*n*)

Calculates the intercept for the nth dimension.

standard

Returns coefficients for the first N-1 standard equation coefficients. The Nth is returned separately.

xintercept

Return the x-intercept.

yintercept

Return the y-intercept.

class `xdesign.geometry.Polygon` (*vertices*)

Bases: `xdesign.geometry.Entity`

A convex polygon in 2-D cartesian space.

It is defined by a number of distinct points.

vertices

sequence of Points

area

Return the area of the entity.

bounds

Return a tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the geometric figure.

contains (*points*)**half_space****list**

Return list representation.

numpy

Return Numpy representation.

numverts**patch****perimeter**

Return the perimeter of the entity.

rotate (*theta*, *point=None*, *axis=None*)

Rotate entity around an axis which passes through a point by theta radians.

translate (*vector*)

Translate polygon.

class `xdesign.geometry.Triangle` (*p1*, *p2*, *p3*)

Bases: `xdesign.geometry.Polygon`

Triangle in 2-D cartesian space.

It is defined by three distinct points.

area**center**

class `xdesign.geometry.Rectangle` (*p1*, *p2*, *p3*, *p4*)

Bases: `xdesign.geometry.Polygon`

Rectangle in 2-D cartesian space.

It is defined by four distinct points.

area**center**

class `xdesign.geometry.Square` (*center*, *side_length*)

Bases: `xdesign.geometry.Rectangle`

Square in 2-D cartesian space.

It is defined by a center and a side length.

class `xdesign.geometry.Mesh` (*obj=None*)

Bases: `xdesign.geometry.Entity`

A mesh object. It is a collection of polygons

append (*t*)

Add a triangle to the mesh.

center**contains** (*points*)**half_space**

import_triangle (*obj*)
Loads mesh data from a Python Triangle dict.

patch

pop (*i=-1*)
Pop i-th triangle from the mesh.

rotate (*theta, point=None, axis=None*)
Rotate entity around an axis which passes through a point by theta radians.

scale (*vector*)
Scale entity.

translate (*vector*)
Translate entity.

1.5 xdesign.material

Functions:

<i>Material</i> (formula, density)	Placeholder for class which uses NIST data to automatically calculate material properties based on beam energy.
<i>HyperbolicConcentric</i> ([min_width, exponent])	Generates a series of cocentric alternating black and white circles whose radii increase exponentially.
<i>DynamicRange</i> ([steps, jitter, shape])	Generates a phantom of randomly placed circles for determining dynamic range.
<i>DogaCircles</i> ([n_sizes, size_ratio, n_shuffles])	Rows of increasingly smaller circles.
<i>SlantedSquares</i> ([count, angle, gap])	Generates a collection of slanted squares.
<i>UnitCircle</i> ([radius, mass_atten])	Generates a phantom with a single circle in its center.
<i>Soil</i> ([porosity])	Generates a phantom with structure similar to soil.
<i>WetCircles</i> ()	
<i>SiemensStar</i> ([n_sectors, center, radius])	Generates a Siemens star.
<i>Foam</i> ([size_range, gap, porosity])	Generates a phantom with structure similar to foam.
<i>Metal</i> ([shape])	
<i>SoftBiomaterial</i> ([shape])	
<i>Electronics</i> ([shape])	
<i>FiberComposite</i> ([shape])	

class xdesign.material.**Material** (*formula, density*)

Bases: `object`

Placeholder for class which uses NIST data to automatically calculate material properties based on beam energy.

atom_concentration
Number of atoms per unit volume [1/cm^3].

atomic_form_factor
Measure of the scattering amplitude of a wave by an isolated atom. Read from NIST database [Unitless].

compton_cross_section
Compton cross-section of the electron [cm^2].

compton_scattering
X-ray attenuation due to the Compton scattering [1/cm].

electron_density
Electron density [e/cm^3].

linear_attenuation

Total x-ray attenuation [1/cm].

mass_ratio()**number_of_elements()****photoelectric_absorption**

X-ray attenuation due to the photoelectric effect [1/cm].

photoelectric_cross_section**reduced_energy_ratio**

Energy ratio of the incident x-ray and the electron energy [Unitless].

refractive_index

class `xdesign.material.HyperbolicConcentric` (*min_width=0.1, exponent=0.5*)

Bases: `xdesign.phantom.Phantom`

Generates a series of cocentric alternating black and white circles whose radii are changing at a parabolic rate. These line spacings cover a range of scales and can be used to estimate the Modulation Transfer Function. The radii change according to this function: $r(n) = r0 \cdot (n+1)^k$.

radii

list

The list of radii of the circles

widths

list

The list of the widths of the bands

class `xdesign.material.DynamicRange` (*steps=10, jitter=True, shape=u'square'*)

Bases: `xdesign.phantom.Phantom`

Generates a phantom of randomly placed circles for determining dynamic range.

Parameters

- **steps** (*scalar, optional*) – The orders of magnitude (base 2) that the colors of the circles cover.
- **jitter** (*bool, optional*) – True : circles are placed in a jittered grid False : circles are randomly placed
- **shape** (*string, optional*)

class `xdesign.material.DogaCircles` (*n_sizes=5, size_ratio=0.5, n_shuffles=5*)

Bases: `xdesign.phantom.Phantom`

Rows of increasingly smaller circles. Initially arranged in an ordered Latin square, the initial arrangement can be randomly shuffled.

radii

ndarray

radii of circles

x

ndarray

x position of circles

y*ndarray*

y position of circles

class `xdesign.material.SlantedSquares` (*count=10, angle=0.08726646259972222, gap=0*)Bases: `xdesign.phantom.Phantom`

Generates a collection of slanted squares. Squares are arranged in concentric circles such that the space between squares is at least gap. The size of the squares is adaptive such that they all remain within the unit circle.

angle*scalar*

the angle of slant in radians

count*scalar*

the total number of squares

gap*scalar*

the minimum space between squares

side_length*scalar*

the size of the squares

squares_per_level*list*

the number of squares at each level

radius_per_level*list*

the radius at each level

n_levels*scalar*

the number of levels

class `xdesign.material.UnitCircle` (*radius=0.5, mass_atten=1*)Bases: `xdesign.phantom.Phantom`

Generates a phantom with a single circle in its center.

class `xdesign.material.Soil` (*porosity=0.412*)Bases: `xdesign.phantom.Phantom`

Generates a phantom with structure similar to soil.

References

Schlüter, S., Sheppard, A., Brown, K., & Wildenschild, D. (2014). Image processing of multiphase images obtained via Xray microtomography: a review. *Water Resources Research*, 50(4), 3615-3639.

class `xdesign.material.WetCircles`Bases: `xdesign.phantom.Phantom`

class `xdesign.material.SiemensStar` (*n_sectors=4*, *center=<xdesign.geometry.Point object>*, *radius=0.5*)

Bases: `xdesign.phantom.Pantom`

Generates a Siemens star.

ratio

scalar

The spatial frequency times the proportional radius. e.g to get the frequency, *f*, divide this ratio by some fraction of the maximum radius: *f* = *ratio*/*radius_fraction*

class `xdesign.material.Foam` (*size_range=[0.05, 0.01]*, *gap=0*, *porosity=1*)

Bases: `xdesign.phantom.Pantom`

Generates a phantom with structure similar to foam.

class `xdesign.material.Metal` (*shape=u'square'*)

Bases: `xdesign.phantom.Pantom`

class `xdesign.material.SoftBiomaterial` (*shape=u'square'*)

Bases: `xdesign.phantom.Pantom`

class `xdesign.material.Electronics` (*shape=u'square'*)

Bases: `xdesign.phantom.Pantom`

class `xdesign.material.FiberComposite` (*shape=u'square'*)

Bases: `xdesign.phantom.Pantom`

1.6 xdesign.metrics

Functions:

<code>ImageQuality</code> (<i>original</i> , <i>reconstruction</i> [, <i>method</i>])	Stores information about image quality.
<code>probability_mask</code> (<i>phantom</i> , <i>size</i> [, <i>ratio</i> , <i>uniform</i>])	Returns the probability mask for each phase in the phantom.
<code>compute_quality</code> (<i>reference</i> , <i>reconstructions</i>)	Computes image quality metrics for each of the reconstructions.
<code>compute_PCC</code> (<i>A</i> , <i>B</i> [, <i>masks</i>])	Computes the Pearson product-moment correlation coefficients (PCC) for
<code>compute_likeness</code> (<i>A</i> , <i>B</i> , <i>masks</i>)	Predicts the likelihood that each pixel in <i>B</i> belongs to a phase based on the
<code>compute_background_ttest</code> (<i>image</i> , <i>masks</i>)	Determines whether the background has significantly different luminance t
<code>compute_mtf</code> (<i>phantom</i> , <i>image</i> [, <i>Ntheta</i>])	Uses method described in Friedman et al to calculate the MTF.
<code>compute_nps</code> (<i>phantom</i> , <i>A</i> [, <i>B</i> , <i>plot_type</i>])	Calculates the noise power spectrum from a unit circle image.
<code>compute_neq</code> (<i>phantom</i> , <i>A</i> , <i>B</i>)	Calculates the NEQ according to recommendations by JT Dobbins.
<code>compute_mtf_siemens</code> (<i>phantom</i> , <i>image</i>)	Calculates the MTF using the modulated Siemens Star method in Loebich

class `xdesign.metrics.ImageQuality` (*original*, *reconstruction*, *method=u''*)

Bases: `object`

Stores information about image quality.

orig

numpy.ndarray

recon

numpy.ndarray

qualities

list of scalars

maps

list of numpy.ndarray

scales

list of scalars

add_quality (*quality, scale, maps=None*)

Parameters

- **quality** (*scalar, list*) – The average quality for the image
- **map** (*array, list of arrays, optional*) – the local quality rating across the image
- **scale** (*scalar, list*) – the size scale at which the quality was calculated

sort ()

Sorts the qualities by scale

`xdesign.metrics.probability_mask` (*phantom, size, ratio=8, uniform=True*)

Returns the probability mask for each phase in the phantom.

Parameters

- **size** (*scalar*) – The side length in pixels of the resulting square image.
- **ratio** (*scalar, optional*) – The discretization works by drawing the shapes in a larger space then averaging and downsampling. This parameter controls how many pixels in the larger representation are averaged for the final representation. e.g. if ratio = 8, then the final pixel values are the average of 64 pixels.
- **uniform** (*boolean, optional*) – When set to False, changes the way pixels are averaged from a uniform weights to gaussian weights.

Returns **image** (*list of numpy.ndarray*) – A list of float masks for each phase in the phantom.

`xdesign.metrics.compute_quality` (*reference, reconstructions, method=u'MSSSIM', L=1*)

Computes image quality metrics for each of the reconstructions.

Parameters

- **reference** (*array*) – the discrete reference image. In a future release, we will determine the best way to compare a continuous domain to a discrete reconstruction.
- **reconstructions** (*list of arrays*) – A list of discrete reconstructions
- **method** (*string, optional*) – The quality metric desired for this comparison. Options include: SSIM, MSSSIM
- **L** (*scalar*) – The dynamic range of the data. This value is 1 for float representations and 2^{bitdepth} for integer representations.

Returns **metrics** (*list of ImageQuality*)

`xdesign.metrics.compute_PCC` (*A, B, masks=None*)

Computes the Pearson product-moment correlation coefficients (PCC) for the two images.

Parameters

- **A,B** (*ndarray*) – The two images to be compared
- **masks** (*list of ndarrays, optional*) – If supplied, the data under each mask is computed separately.

Returns **covariances** (*array, list of arrays*)

`xdesign.metrics.compute_likeness(A, B, masks)`

Predicts the likelihood that each pixel in B belongs to a phase based on the histogram of A.

Parameters

- **A** (*ndarray*)
- **B** (*ndarray*)
- **masks** (*list of ndarrays*)

Returns *likelihoods (list of ndarrays)*

`xdesign.metrics.compute_background_ttest(image, masks)`

Determines whether the background has significantly different luminance than the other phases.

Parameters

- **image** (*ndarray*)
- **masks** (*list of ndarrays*) – Masks for the background and any other phases. Does not auto-generate the non-background mask because maybe you want to compare only two phases.

Returns

- **tstat** (*scalar*)
- **pvalue** (*scalar*)

`xdesign.metrics.compute_mtf(phantom, image, Ntheta=4)`

Uses method described in Friedman et al to calculate the MTF.

Parameters

- **phantom** (*UnitCircle*) – Predefined phantom with single circle whose radius is less than 0.5.
- **image** (*ndarray*) – The reconstruction of the above phantom.
- **Ntheta** (*scalar*) – The number of directions at which to calculate the MTF.

Returns

- **wavenumber** (*ndarray*) – wavelenth in the scale of the original phantom
- **MTF** (*ndarray*) – MTF values
- **bin_centers** (*ndarray*) – the center of the bins if Ntheta >= 1

References

S. N. Friedman, G. S. K. Fung, J. H. Siewerdsen, and B. M. W. Tsui. “A simple approach to measure computed tomography (CT) modulation transfer function (MTF) and noise-power spectrum (NPS) using the American College of Radiology (ACR) accreditation phantom,” *Med. Phys.* 40, 051907-1 - 051907-9 (2013). <http://dx.doi.org/10.1118/1.4800795>

`xdesign.metrics.compute_nps(phantom, A, B=None, plot_type=u'frequency')`

Calculates the noise power spectrum from a unit circle image. The peak at low spatial frequency is probably due to aliasing. Investigation into supressing this peak is necessary.

References

S. N. Friedman, G. S. K. Fung, J. H. Siewerdsen, and B. M. W. Tsui. “A simple approach to measure computed tomography (CT) modulation transfer function (MTF) and noise-power spectrum (NPS) using the American College of Radiology (ACR) accreditation phantom,” *Med. Phys.* 40, 051907-1 - 051907-9 (2013). <http://dx.doi.org/10.1118/1.4800795>

Parameters

- **phantom** (*UnitCircle*) – The unit circle phantom.
- **A** (*ndarray*) – The reconstruction of the above phantom.
- **B** (*ndarray*) – The reconstruction of the above phantom with different noise. This second reconstruction enables allows use of trend subtraction instead of zero mean normalization.
- **plot_type** (*string*) – ‘histogram’ returns a plot binned by radial coordinate wavenumber ‘frequency’ returns a wavenumber vs wavenumber plot

Returns

- *bins* – Bins for the radially binned NPS
- *counts* – NPS values for the radially binned NPS
- *X, Y* – Frequencies for the 2D frequency plot NPS
- **NPS** (*2Darray*) – the NPS for the 2D frequency plot

`xdesign.metrics.compute_neq(phantom, A, B)`

Calculates the NEQ according to recommendations by JT Dobbins.

Parameters

- **phantom** (*UnitCircle*) – The unit circle class with radius less than 0.5
- **A** (*ndarray*) – The reconstruction of the above phantom.
- **B** (*ndarray*) – The reconstruction of the above phantom with different noise. This second reconstruction enables allows use of trend subtraction instead of zero mean normalization.

Returns

- *mu_b* – The spatial frequencies
- *NEQ* – the Noise Equivalent Quanta

`xdesign.metrics.compute_mtf_siemens(phantom, image)`

Calculates the MTF using the modulated Siemens Star method in Loebich et al. (2007).

Parameters

- **phantom** (*SiemensStar*)
- **image** (*ndarray*) – The reconstruciton of the SiemensStar

Returns

- **frequency** (*array*) – The spatial frequency in cycles per unit length
- **M** (*array*) – The MTF values for each frequency

1.7 xdesign.phantom

Functions:

Phantom([shape]) Phantom generation class.

class xdesign.phantom.**Phantom**(*shape=u'circle'*)
 Bases: *object*
 Phantom generation class.

shape
string
 Shape of the phantom. Available options: circle, square.

population
scalar
 Number of generated features in the phantom.

area
scalar
 Area of the features in the phantom.

feature
list
 List of features.

append(*feature*)
 Add a feature to the top of the phantom.

density
 Returns the area density of the phantom. (Does not account for mass_atten.)

insert(*i, feature*)
 Insert a feature at a given depth.

list

load(*filename*)
 Load phantom from file.

numpy()
 Returns the Numpy representation.

pop(*i=-1*)
 Pop i-th feature from the phantom.

reverse()
 Reverse the features of the phantom

rotate(*theta, origin=<xdesign.geometry.Point object>, axis=None*)
 Rotate phantom around a point.

save(*filename*)
 Save phantom to file.

sort(*param=u'mass_atten', reverse=False*)
 Sorts the features by mass_atten or size

sprinkle (*counts, radius, gap=0, region=None, mass_atten=1, max_density=1*)

Sprinkle a number of circles. Uses various termination criteria to determine when to stop trying to add circles.

Parameters

- **counts** (*int*) – The number of circles to be added.
- **radius** (*scalar or list*) – The radius of the circles to be added.
- **gap** (*float, optional*) – The minimum distance between circle boundaries. A negative value allows overlapping edges.
- **region** (*Entity, optional*) – The new circles are confined to this shape. None if the circles are allowed anywhere.
- **max_density** (*scalar, optional*) – Stops adding circles when the geometric density of the phantom reaches this ratio.
- **mass_atten** (*scalar, optional*) – A mass attenuation parameter passed to the circles.

Returns **counts** (*scalar*) – The number of circles successfully added.

translate (*dx, dy*)

Translate phantom.

1.8 xdesign.plot

Functions:

<code>plot_phantom</code> (phantom[, axis, labels, ...])	Plots a phantom to the given axis.
<code>plot_feature</code> (feature[, axis, alpha, c])	Plots a feature on the given axis.
<code>plot_mesh</code> (mesh[, axis, alpha, c])	Plots a mesh to the given axis.
<code>plot_polygon</code> (polygon[, axis, alpha, c])	Plots a polygon to the given axis.
<code>plot_curve</code> (curve[, axis, alpha, c])	Plots a curve to the given axis.
<code>discrete_phantom</code> (phantom, size[, ratio, ...])	Returns discrete representation of the property function, prop, in the phantom.
<code>sidebyside</code> (p[, size, labels, prop])	Displays the geometry and the discrete property function of the given phantom side by side.
<code>multiroll</code> (x, shift[, axis])	Roll an array along each axis.
<code>plot_metrics</code> (imqual)	Plots full reference metrics of ImageQuality data.
<code>plot_mtf</code> (faxis, MTF[, labels])	Plots the MTF.
<code>plot_nps</code> (X, Y, NPS)	Plots the 2D frequency plot for the NPS.
<code>plot_neq</code> (freq, NEQ)	Plots the NEQ.
<code>plot_histograms</code> (images[, masks, thresh])	Plots the normalized histograms for the pixel intensity under each mask.

`xdesign.plot.plot_phantom` (*phantom, axis=None, labels=None, c_props=[], c_map=None*)

Plots a phantom to the given axis.

Parameters

- **labels** (*bool, optional*) – Each feature is numbered according to its index in the phantom.
- **c_props** (*list of str, optional*) – List of feature properties to use for colormapping the geometries.
- **c_map** (*function, optional*) – A function which takes the a list of prop(s) for a Feature as input and returns a matplotlib color specifier.

`xdesign.plot.plot_feature` (*feature*, *axis=None*, *alpha=None*, *c=None*)

Plots a feature on the given axis.

Parameters

- **alpha** (*float*) – The plot opacity. 0 is transparent. 1 is opaque.
- **c** (*matplotlib color specifier*) – See http://matplotlib.org/api/colors_api.html

`xdesign.plot.plot_mesh` (*mesh*, *axis=None*, *alpha=None*, *c=None*)

Plots a mesh to the given axis.

Parameters

- **alpha** (*float*) – The plot opacity. 0 is transparent. 1 is opaque.
- **c** (*matplotlib color specifier*) – See http://matplotlib.org/api/colors_api.html

`xdesign.plot.plot_polygon` (*polygon*, *axis=None*, *alpha=None*, *c=None*)

Plots a polygon to the given axis.

Parameters

- **alpha** (*float*) – The plot opacity. 0 is transparent. 1 is opaque.
- **c** (*matplotlib color specifier*) – See http://matplotlib.org/api/colors_api.html

`xdesign.plot.plot_curve` (*curve*, *axis=None*, *alpha=None*, *c=None*)

Plots a curve to the given axis.

Parameters

- **alpha** (*float*) – The plot opacity. 0 is transparent. 1 is opaque.
- **c** (*matplotlib color specifier*) – See http://matplotlib.org/api/colors_api.html

`xdesign.plot.discrete_phantom` (*phantom*, *size*, *ratio=8*, *uniform=True*, *prop=u'mass_atten'*)

Returns discrete representation of the property function, *prop*, in the phantom. The values of overlapping features are additive.

Parameters

- **size** (*scalar*) – The side length in pixels of the resulting square image.
- **ratio** (*scalar, optional*) – The antialiasing works by supersampling. This parameter controls how many pixels in the larger representation are averaged for the final representation. e.g. if *ratio* = 8, then the final pixel values are the average of 64 pixels.
- **uniform** (*boolean, optional*) – When set to False, changes the way pixels are averaged from a uniform weights to gaussian weights.
- **prop** (*str, optional*) – The name of the property function to discretize

Returns **image** (*numpy.ndarray*) – The discrete representation of the phantom that is *size* x *size*.

`xdesign.plot.sidebyside` (*p*, *size=100*, *labels=None*, *prop=u'mass_atten'*)

Displays the geometry and the discrete property function of the given phantom side by side.

`xdesign.plot.mutiroll` (*x*, *shift*, *axis=None*)

Roll an array along each axis.

Parameters

- **x** (*array_like*) – Array to be rolled.
- **shift** (*sequence of int*) – Number of indices by which to shift each axis.

- **axis** (*sequence of int, optional*) – The axes to be rolled. If not given, all axes is assumed, and `len(shift)` must equal the number of dimensions of `x`.

Returns `y` (*numpy array, with the same type and size as `x`*) – The rolled array.

Notes

The length of `x` along each axis must be positive. The function does not handle arrays that have axes with length 0.

See also:

`numpy.roll()`

Example

Here's a two-dimensional array:

```
>>> x = np.arange(20).reshape(4,5)
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

Roll the first axis one step and the second axis three steps:

```
>>> multiroll(x, [1, 3])
array([[17, 18, 19, 15, 16],
       [ 2,  3,  4,  0,  1],
       [ 7,  8,  9,  5,  6],
       [12, 13, 14, 10, 11]])
```

That's equivalent to:

```
>>> np.roll(np.roll(x, 1, axis=0), 3, axis=1)
array([[17, 18, 19, 15, 16],
       [ 2,  3,  4,  0,  1],
       [ 7,  8,  9,  5,  6],
       [12, 13, 14, 10, 11]])
```

Not all the axes must be rolled. The following uses the *axis* argument to roll just the second axis:

```
>>> multiroll(x, [2], axis=[1])
array([[ 3,  4,  0,  1,  2],
       [ 8,  9,  5,  6,  7],
       [13, 14, 10, 11, 12],
       [18, 19, 15, 16, 17]])
```

which is equivalent to:

```
>>> np.roll(x, 2, axis=1)
array([[ 3,  4,  0,  1,  2],
       [ 8,  9,  5,  6,  7],
       [13, 14, 10, 11, 12],
       [18, 19, 15, 16, 17]])
```

References

Warren Weckesser <http://stackoverflow.com/questions/30639656/numpy-roll-in-several-dimensions>

`xdesign.plot.plot_metrics` (*imqual*)

Plots full reference metrics of ImageQuality data.

Parameters `imqual` (*ImageQuality*) – The data to plot.

References

Colors taken from this gist <<https://gist.github.com/thriveth/8560036>>

`xdesign.plot.plot_mtf` (*faxis*, *MTF*, *labels=None*)

Plots the MTF. Returns the figure reference.

`xdesign.plot.plot_nps` (*X*, *Y*, *NPS*)

Plots the 2D frequency plot for the NPS. Returns the figure reference.

`xdesign.plot.plot_neq` (*freq*, *NEQ*)

Plots the NEQ. Returns the figure reference.

`xdesign.plot.plot_histograms` (*images*, *masks=None*, *thresh=0.025*)

Plots the normalized histograms for the pixel intensity under each mask.

Parameters

- **images** (*list of ndarrays, ndarray*) – image(s) for comparing histograms.
- **masks** (*list of ndarrays, float, optional*) – If supplied, the data under each mask is plotted separately.
- **strict** (*boolean*) – If true, the mask takes values $\geq$ only. If false, the mask takes all values > 0 .

Examples

This section contains [Jupyter Notebooks](#) examples for various XDesign functions.

To run these examples in a [notebooks](#) install [Jupyter](#) or run the notebooks from scripts from [here](#)

2.1 Quality Metrics and Reconstruction Demo

Demonstrates the use of full reference metrics.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.exposure import adjust_gamma, rescale_intensity
from xdesign import *

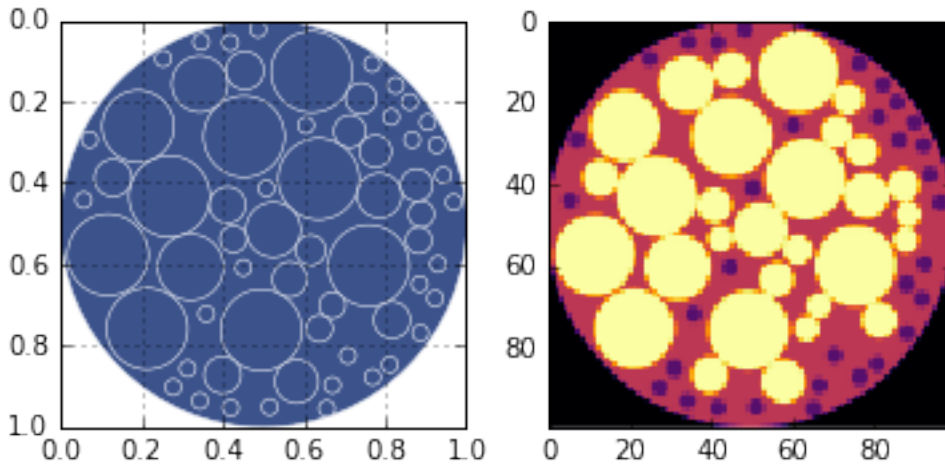
def rescale(reconstruction, hi):
    I = rescale_intensity(reconstruction, out_range=(0., 1.))
    return adjust_gamma(I, 1, hi)
```

Generate a predetermined phantom that resembles soil.

```
np.random.seed(0)
soil_like_phantom = Soil()
```

Generate a figure showing the phantom and save the discrete conversion for later.

```
discrete = sidebyside(soil_like_phantom, 100)
plt.savefig('Soil_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



Simulate data acquisition for parallel beam around 180 degrees.

```

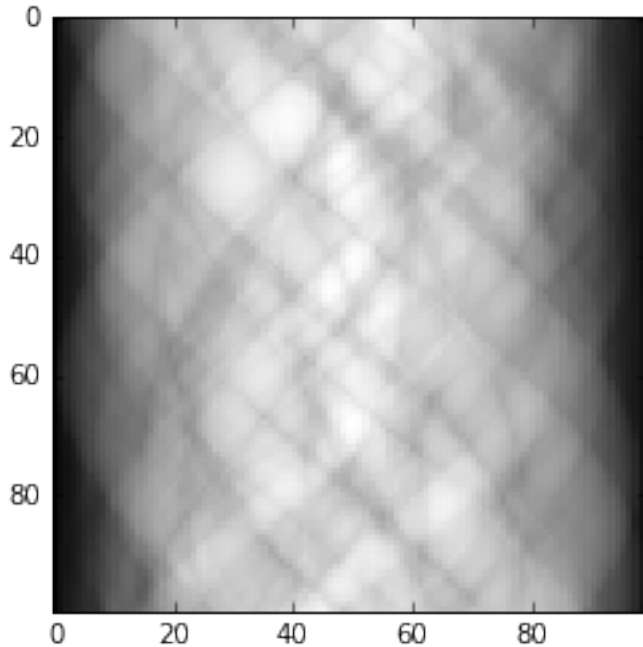
sx, sy = 100, 100
step = 1. / sy
prb = Probe(Point([step / 2., -10]), Point([step / 2., 10]), step)
theta = np.pi / sx
sino = np.zeros(sx * sy)
a = 0
for m in range(sx):
    for n in range(sy):
        update_progress((m*sy + n)/(sx*sy))
        sino[a] = prb.measure(soil_like_phantom)
        a += 1
        prb.translate(step)
    prb.translate(-1)
    prb.rotate(theta, Point([0.5, 0.5]))
update_progress(1)

```

```

plt.imshow(np.reshape(sino, (sx, sy)), cmap='gray', interpolation='nearest')
# plt.hist(sino)
plt.show(block=True)

```



Reconstruct the phantom using 3 different techniques: ART, SIRT, and MLEM

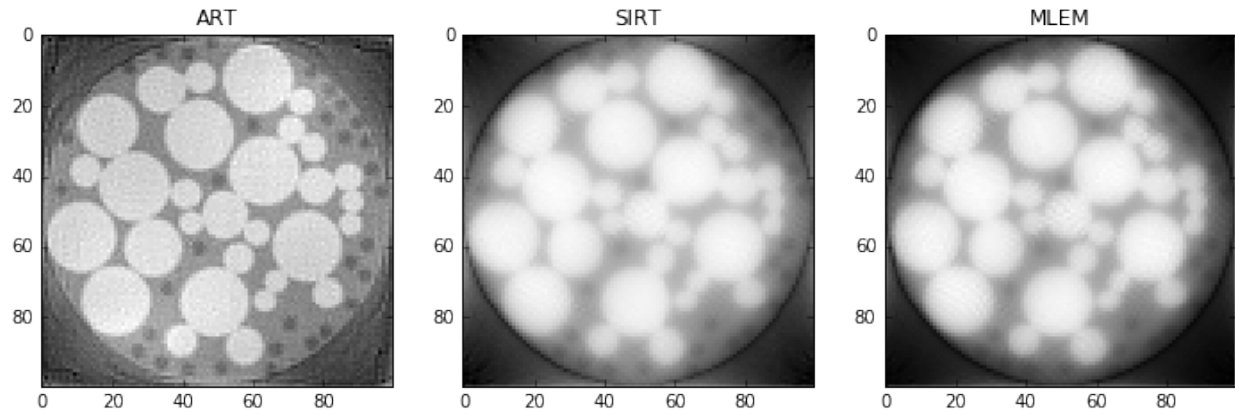
```
hi = 1 # highest expected value in reconstruction (for rescaling)
niter = 10 # number of iterations

init = 1e-12 * np.ones((sx, sy))
rec_art = art(prb, sino, init, niter)
rec_art = rescale(np.rot90(rec_art)[::-1], hi)

init = 1e-12 * np.ones((sx, sy))
rec_sirt = sirt(prb, sino, init, niter)
rec_sirt = rescale(np.rot90(rec_sirt)[::-1], hi)

init = 1e-12 * np.ones((sx, sy))
rec_mlem = mlem(prb, sino, init, niter)
rec_mlem = rescale(np.rot90(rec_mlem)[::-1], hi)
```

```
plt.figure(figsize=(12,4))
plt.subplot(131)
plt.imshow(rec_art, cmap='gray', interpolation='none')
plt.title('ART')
plt.subplot(132)
plt.imshow(rec_sirt, cmap='gray', interpolation='none')
plt.title('SIRT')
plt.subplot(133)
plt.imshow(rec_mlem, cmap='gray', interpolation='none')
plt.title('MLEM')
plt.show()
```



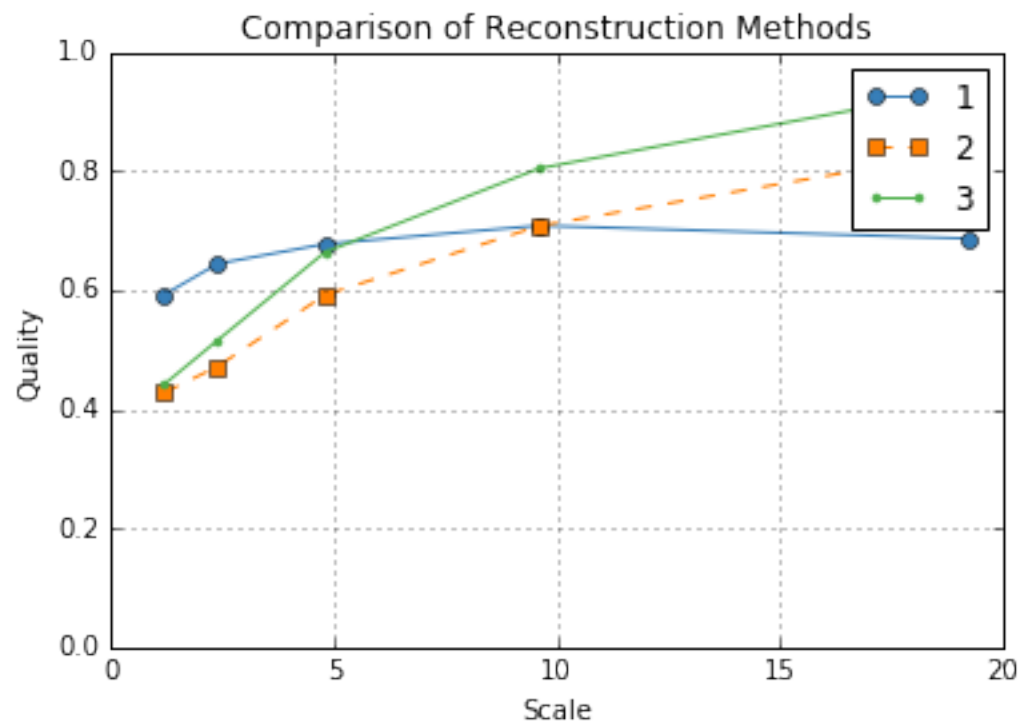
Compute quality metrics using the MSSSIM method.

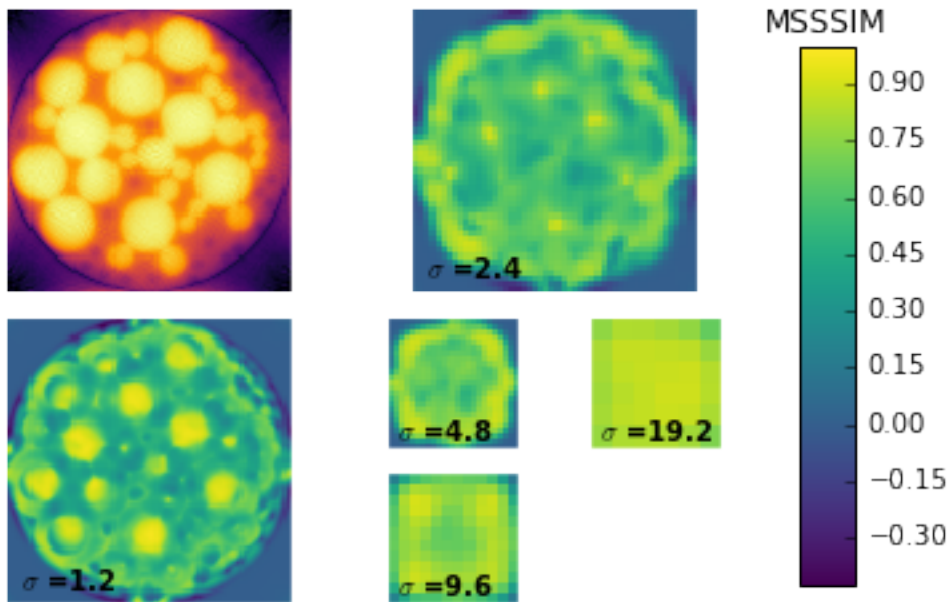
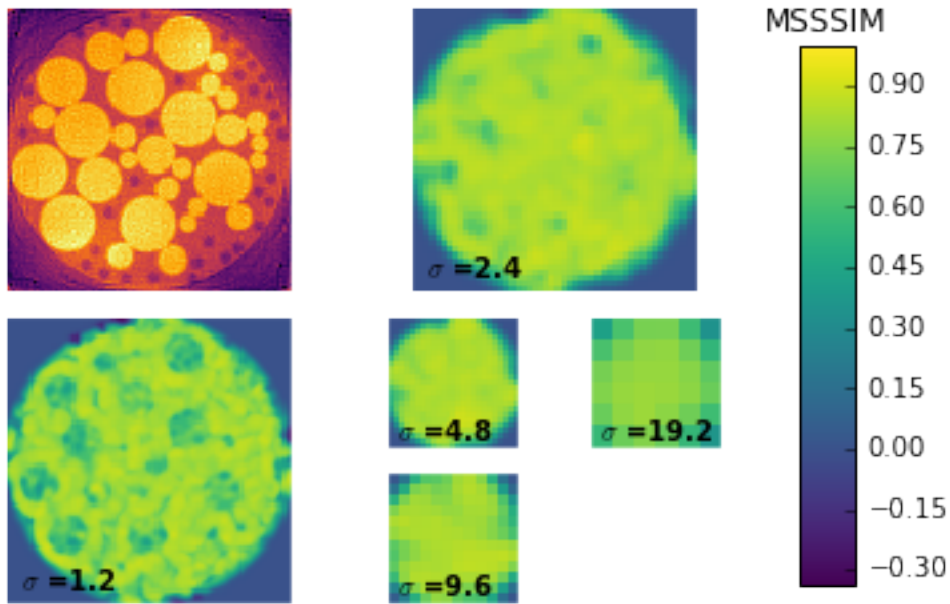
```
metrics = compute_quality(discrete, [rec_art, rec_sirt, rec_mlem], method="MSSSIM")
```

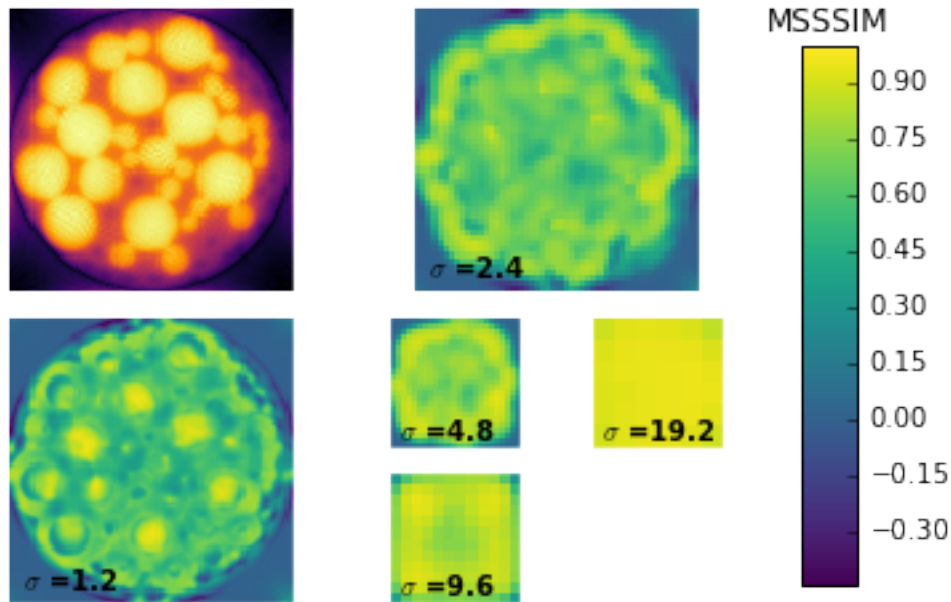
Plotting the results shows the average quality at each level for each reconstruction in a line plot. Then it produces the local quality maps for each reconstruction so we might see why certain reconstructions are ranked higher than others.

In this case, it's clear that ART is ranking higher than MLEM and SIRT at smaller scales because the small dark particles are visible; whereas for SIRT and MLEM they are unresolved. We can also see that the large yellow circles have a more accurately rendered luminance for SIRT and MLEM which is what causes these methods to be ranked higher at larger scales.

```
plot_metrics(metrics)
plt.show()
```







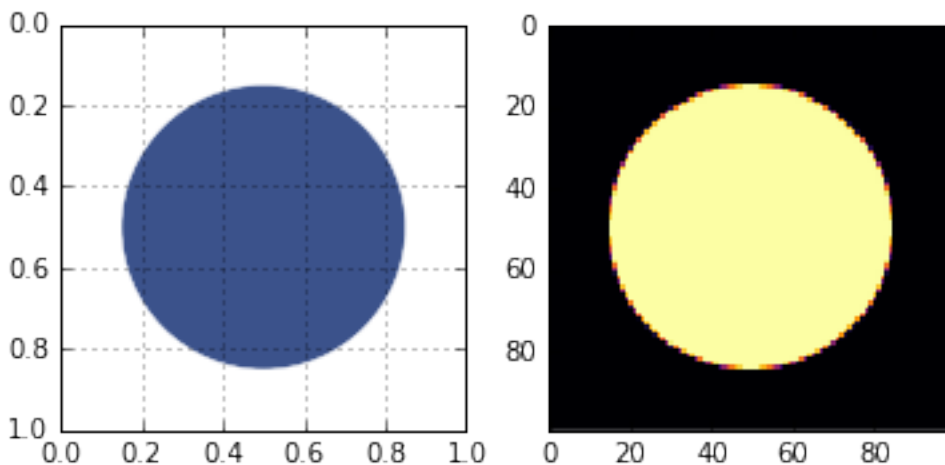
2.2 No Reference Metrics

Demonstrates the use of the no-reference metrics: NPS, MTF, and NEQ.

```
from xdesign import *
import tomopy
import numpy as np
import matplotlib.pyplot as plt
```

Generate a UnitCircle test phantom. For the MTF, the radius must be less than 0.5, otherwise the circle touches the edges of the field of view.

```
p = UnitCircle(mass_atten=4, radius=0.35)
sidebyside(p, 100)
plt.show()
```



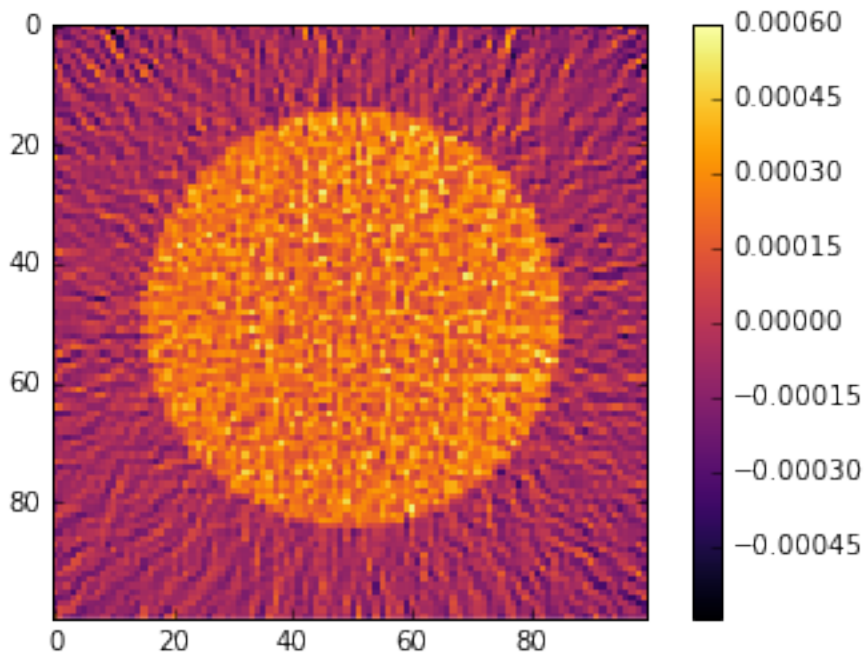
Generate two sinograms and reconstruct. Noise power spectrum and Noise Equivalent Quanta are meaningless without noise so add some poisson noise to the reconstruction process with the `noise` argument. Collecting two sinograms allows us to isolate the noise by subtracting out the circle.

```
np.random.seed(0)
sinoA = sinogram(100, 100, p, noise=0.1)
sinoB = sinogram(100, 100, p, noise=0.1)
theta = np.arange(0, np.pi, np.pi / 100.)

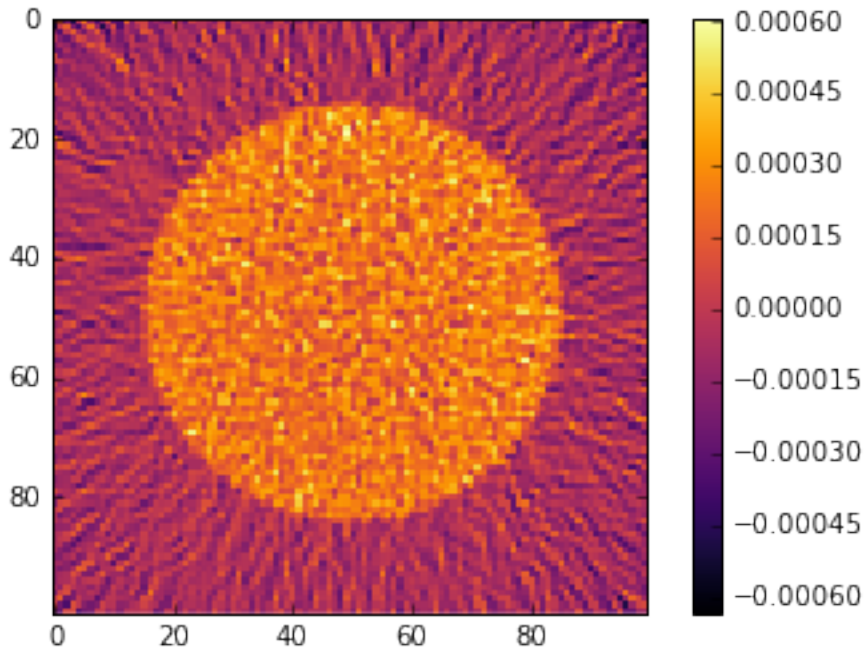
recA = tomopy.recon(np.expand_dims(sinoA, 1), theta, algorithm='gridrec', center=(sinoA.shape[1]-1)/2)
recB = tomopy.recon(np.expand_dims(sinoB, 1), theta, algorithm='gridrec', center=(sinoB.shape[1]-1)/2)
```

Take a look at the two noisy reconstructions.

```
plt.imshow(recA[0], cmap='inferno', interpolation="none")
plt.colorbar()
plt.savefig('UnitCircle_noise0.png', dpi=600,
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



```
plt.imshow(recB[0], cmap='inferno', interpolation="none")
plt.colorbar()
plt.show()
```



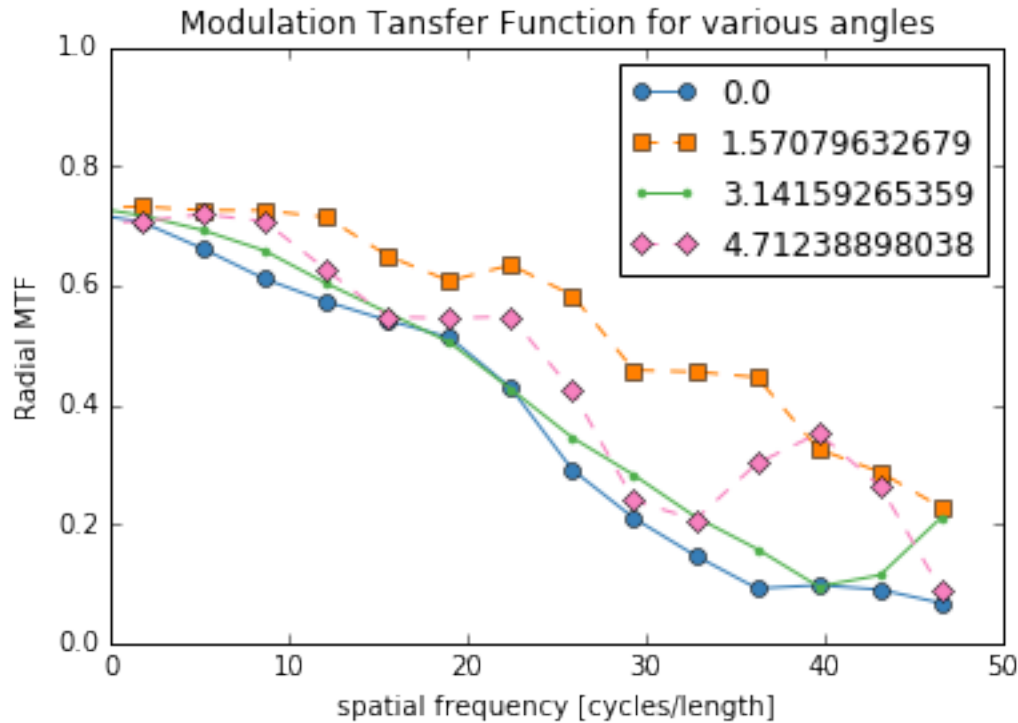
2.2.1 Calculate MTF

This metric is meaningful without noise. You can separate the MTF into multiple directions or average them all together using the `Ntheta` argument.

```
mtf_freq, mtf_value, labels = compute_mtf(p, recA[0], Ntheta=4)
```

The MTF is really a symmetric function around zero frequency, so usually people just show the positive portion. Sometimes, there is a peak at the higher spatial frequencies instead of the MTF approaching zero. This is probably because of aliasing noise content with frequencies higher than the Nyquist frequency.

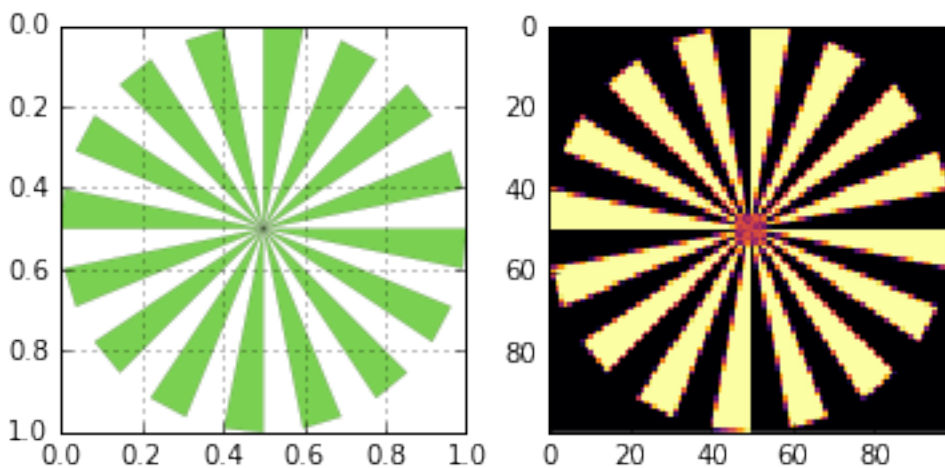
```
plot_mtf(mtf_freq, mtf_value, labels)
plt.gca().set_xlim([0,50]) # hide negative portion of MTF
plt.show()
```



You can also use a Siemens Star to calculate the MTF using a fitted sinusoidal method instead of the slanted edges that the above method uses.

```
s = SiemensStar(n_sectors=32, center=Point([0.5, 0.5]), radius=0.5)
d = sidebyside(s, 100)
```

```
plt.show()
```

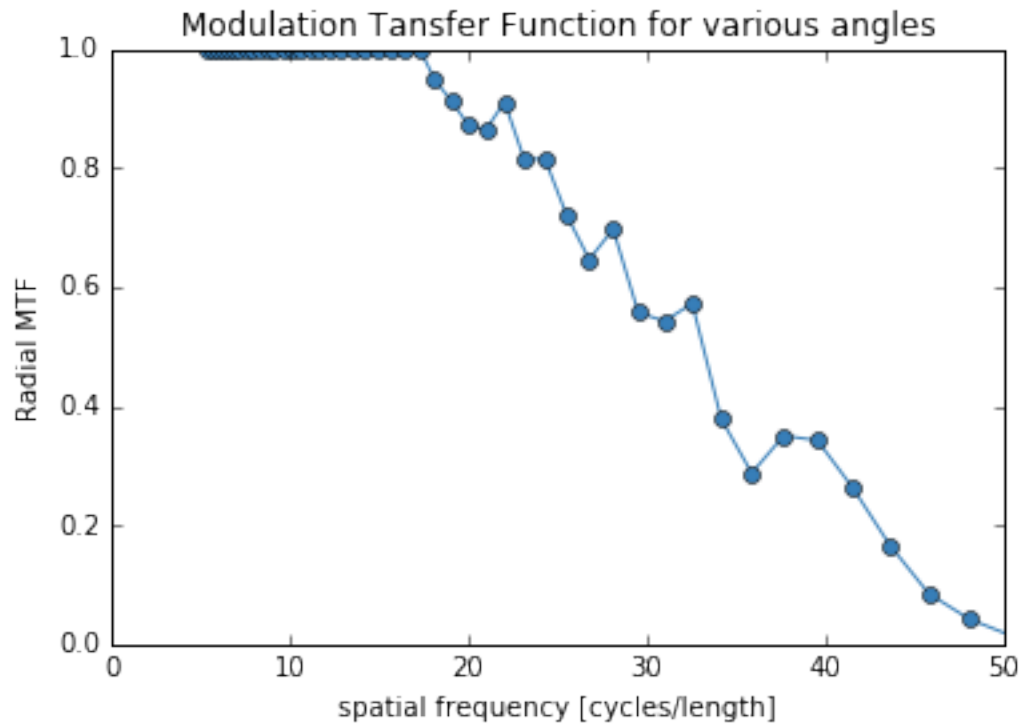


Here we are using the discrete version of the phantom (without noise), so we are only limited by the resolution of the image.

```
mtf_freq, mtf_value = compute_mtf_siemens(s, d)
```

```
plot_mtf(mtf_freq, mtf_value, labels=None)
plt.gca().set_xlim([0, 50]) # hide portion of MTF beyond Nyquist frequency
```

```
plt.show()
```

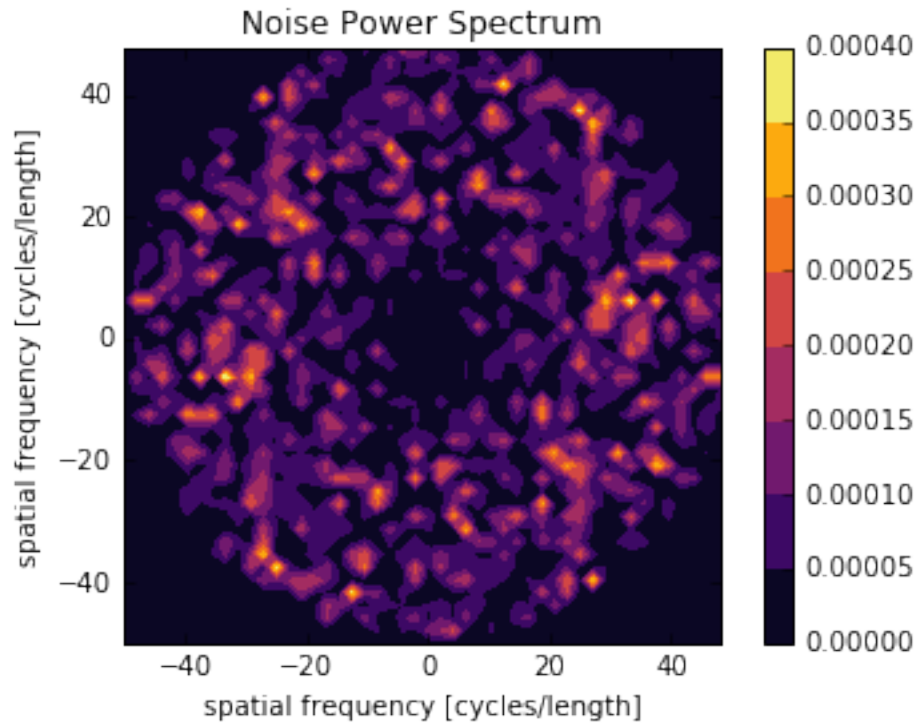


2.2.2 Calculate NPS

You can also calculate the radial or 2D frequency plot of the NPS.

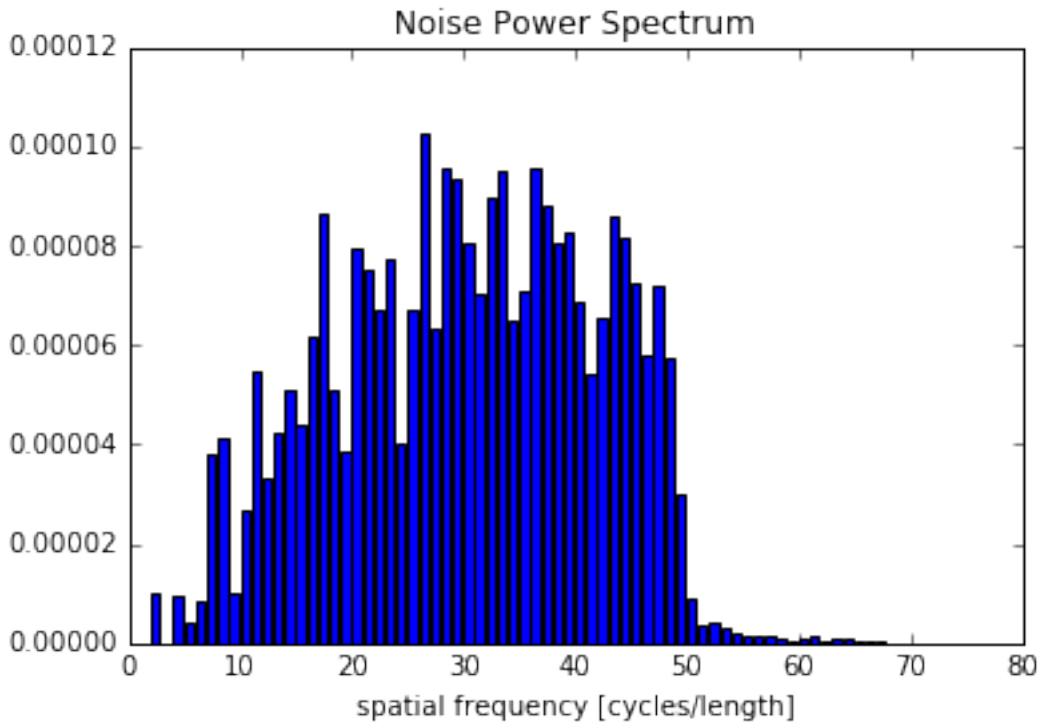
```
X, Y, NPS = compute_nps(p, recA[0], plot_type='frequency', B=recB[0])
```

```
plot_nps(X, Y, NPS)  
plt.show()
```



```
bins, counts = compute_nps(p, recA[0], plot_type='histogram', B=recB[0])
```

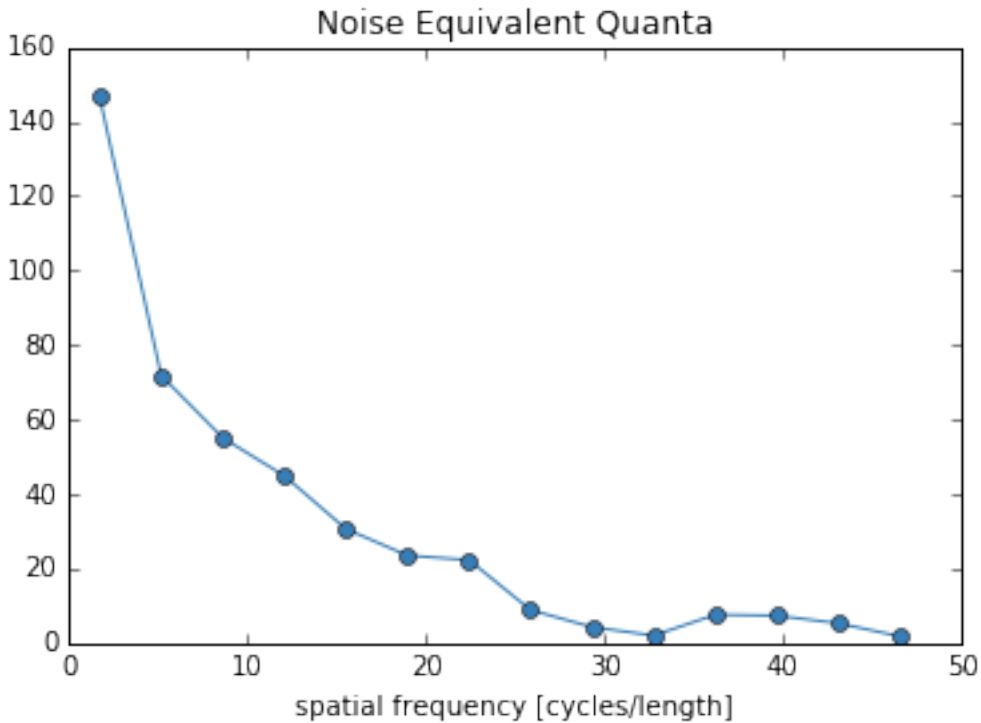
```
plt.figure()  
plt.bar(bins, counts)  
plt.xlabel('spatial frequency [cycles/length]')  
plt.title('Noise Power Spectrum')  
plt.show()
```



2.2.3 Calculate NEQ

```
freq, NEQ = compute_neq(p, recA[0], recB[0])
```

```
plt.figure()
plt.plot(freq.flatten(), NEQ.flatten())
plt.xlabel('spatial frequency [cycles/length]')
plt.title('Noise Equivalent Quanta')
plt.show()
```



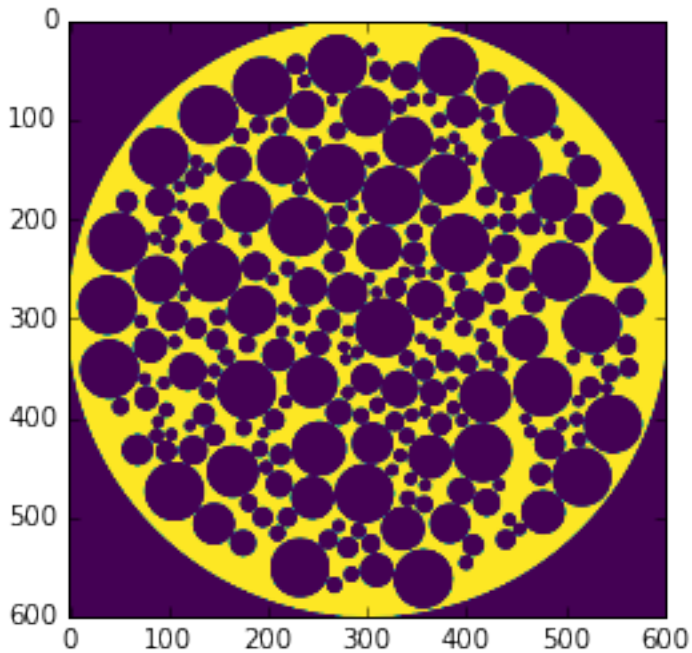
2.3 Parameterized Phantom Generation

Demonstrates how a parameterized function can generate 4 different phantoms from the same parameterized class.

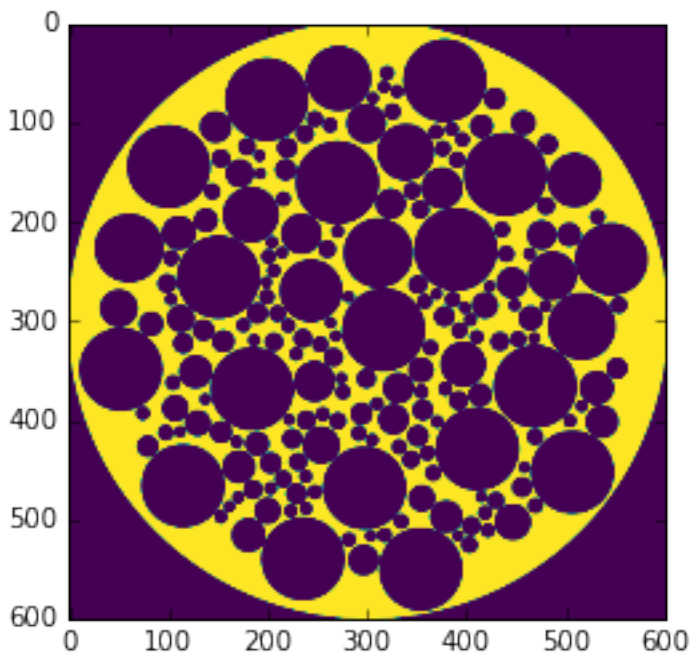
```
from xdesign import *
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec

SIZE = 600
```

```
np.random.seed(0) # random seed for repeatability
p1 = Foam(size_range=[0.05, 0.01], gap=0, porosity=1)
d1 = discrete_phantom(p1, SIZE)
plt.imshow(d1, cmap='viridis')
plt.show()
```

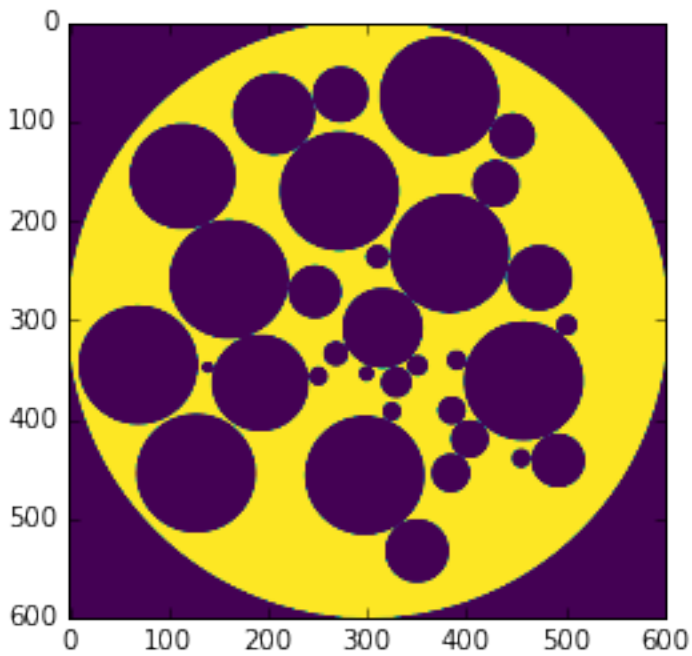


```
np.random.seed(0) # random seed for repeatability
p2 = Foam(size_range=[0.07, 0.01], gap=0, porosity=0.75)
d2 = discrete_phantom(p2, SIZE)
plt.imshow(d2, cmap='viridis')
plt.show()
```

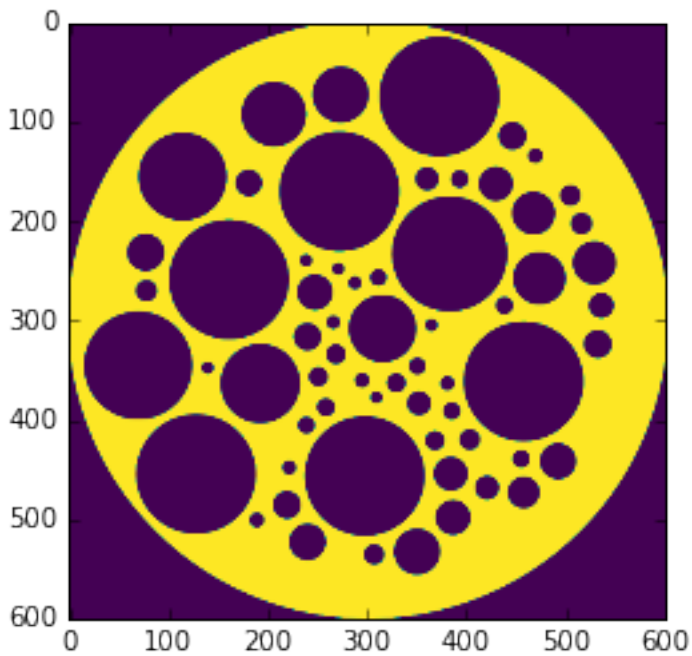


```
np.random.seed(0) # random seed for repeatability
p3 = Foam(size_range=[0.1, 0.01], gap=0, porosity=0.5)
d3 = discrete_phantom(p3, SIZE)
plt.imshow(d3, cmap='viridis')
```

```
plt.show()
```



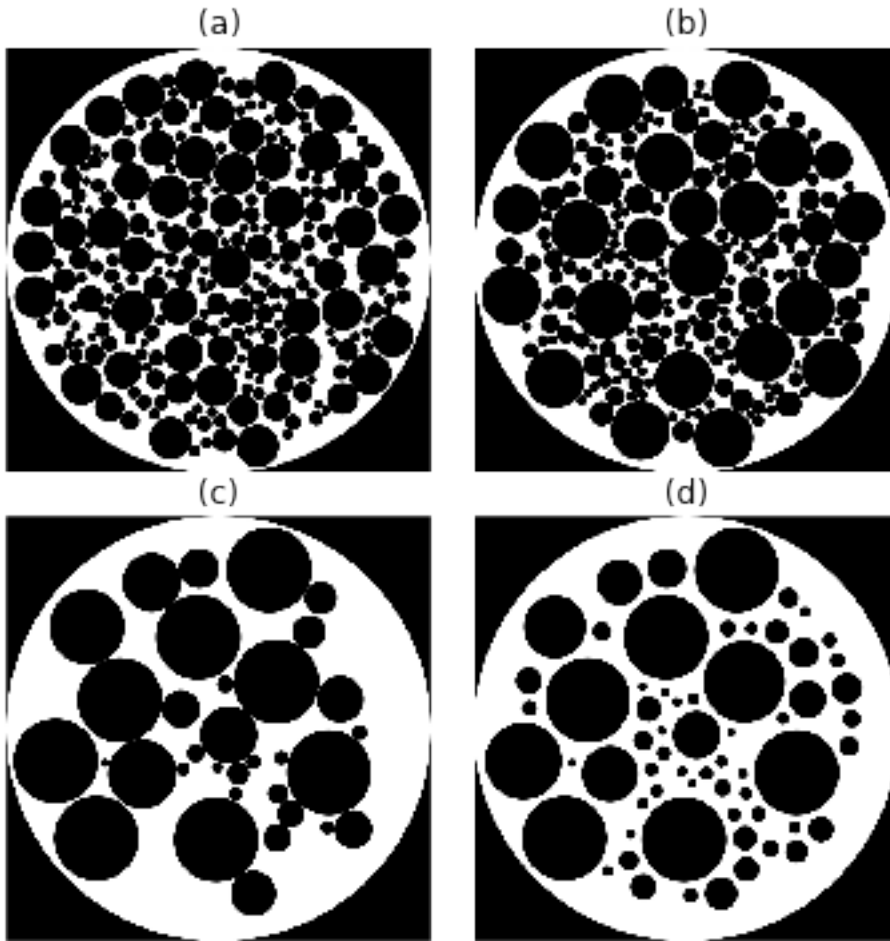
```
np.random.seed(0) # random seed for repeatability
p4 = Foam(size_range=[0.1, 0.01], gap=0.015, porosity=0.5)
d4 = discrete_phantom(p4, SIZE)
plt.imshow(d4, cmap='viridis')
plt.show()
```



Create a composite figure of all four discrete phantoms.

```
fig = plt.figure(dpi=600)
gs1 = gridspec.GridSpec(2, 2)
gs1.update(wspace=0.1, hspace=0.1) # set the spacing between axes.

plt.subplot(gs1[0])
plt.title('(a)')
plt.axis('off')
plt.imshow(d1, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[1])
plt.title('(b)')
plt.axis('off')
plt.imshow(d2, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[2])
plt.title('(c)')
plt.axis('off')
plt.imshow(d3, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[3])
plt.title('(d)')
plt.axis('off')
plt.imshow(d4, interpolation='none', cmap=plt.cm.gray)
fig.set_size_inches(6, 6)
plt.savefig('Foam_parameterized.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



2.4 Simple Phantom Construction Demo

Demonstrates simple basic custom phantom and sinogram generation.

```
import matplotlib.pyplot as plt
import numpy as np
from xdesign import *
```

Create various Features with geometries and assign attenuation values to each of the Features.

```
head = Feature(Circle(Point([0.5, 0.5]), radius=0.5))
head.mass_atten = 1
eyeL = Feature(Circle(Point([0.3, 0.5]), radius=0.1))
eyeL.mass_atten = 1
eyeR = Feature(Circle(Point([0.7, 0.5]), radius=0.1))
eyeR.mass_atten = 1
mouth = Feature(Triangle(Point([0.2, 0.7]), Point([0.5, 0.8]), Point([0.8, 0.7])))
mouth.mass_atten = -1
```

Add 'Features' to 'Phantom'.

```

Shepp = Phantom()
Shepp.append(head)
Shepp.append(eyeL)
Shepp.append(eyeR)
Shepp.append(mouth)

```

Plot the Phantom geometry and properties with a colorbar.

```

fig = plt.figure(figsize=(7, 3), dpi=600)

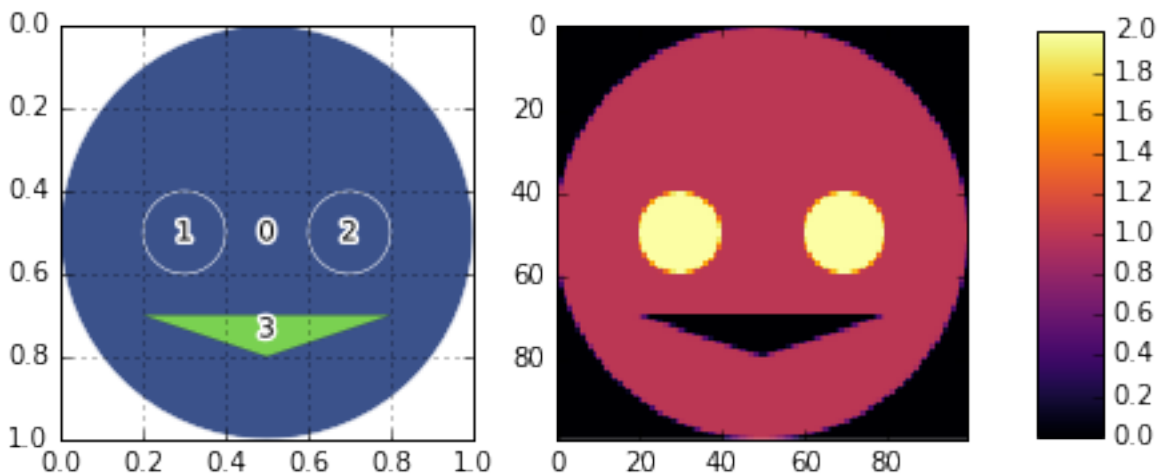
# plot geometry
axis = fig.add_subplot(121, aspect='equal')
plt.grid('on')
plt.gca().invert_yaxis()
plot_phantom(Shepp, axis=axis, labels=False)

# plot property
plt.subplot(1, 2, 2)
im = plt.imshow(discrete_phantom(Shepp, 100, prop='mass_atten'), interpolation='none', cmap=plt.cm.inferno)

# plot colorbar
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.16, 0.05, 0.7])
fig.colorbar(im, cax=cbar_ax)

# save the figure
plt.savefig('Shepp_sidebyside.png', dpi=600,
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()

```



Simulate data acquisition for parallel beam around 180 degrees.

```

sx, sy = 100, 100
step = 1. / sy
prb = Probe(Point([step / 2., -10]), Point([step / 2., 10]), step)
theta = np.pi / sx
sino = np.zeros(sx * sy)

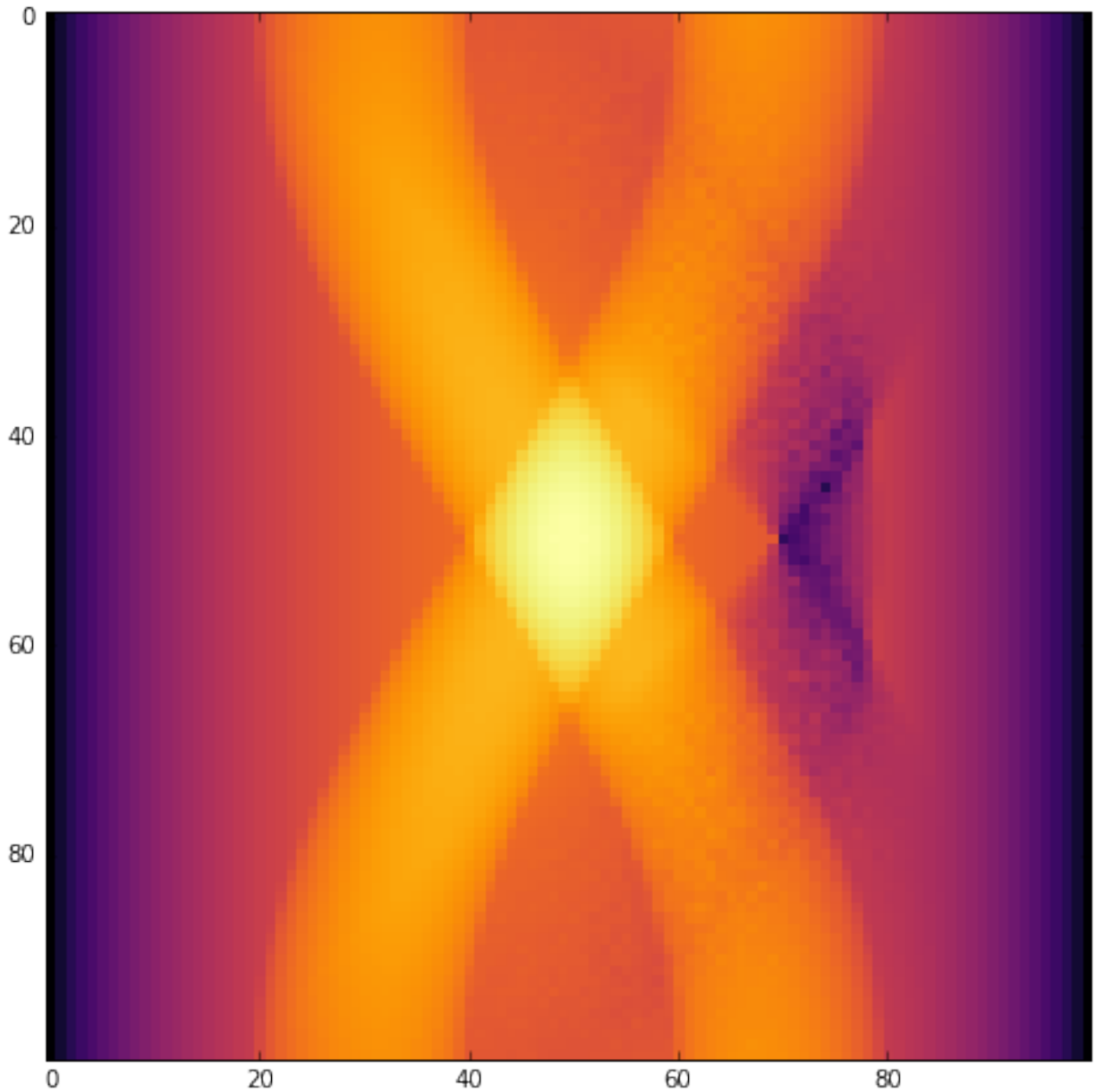
a = 0

```

```
for m in range(sx):
    for n in range(sy):
        update_progress((m*sy + n)/(sx*sy))
        sino[a] = prb.measure(Shepp)
        a += 1
        prb.translate(step)
    prb.translate(-1)
    prb.rotate(theta, Point([0.5, 0.5]))
update_progress(1)
```

Plot the sinogram.

```
plt.figure(figsize=(8, 8))
plt.imshow(np.reshape(sino, (sx, sy)), cmap='inferno', interpolation='nearest')
plt.savefig('Shepp_sinogram.png', dpi=600,
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



2.5 Standard Test Patterns

Generates sidebyside plots of all the standard test patterns in xdesign.

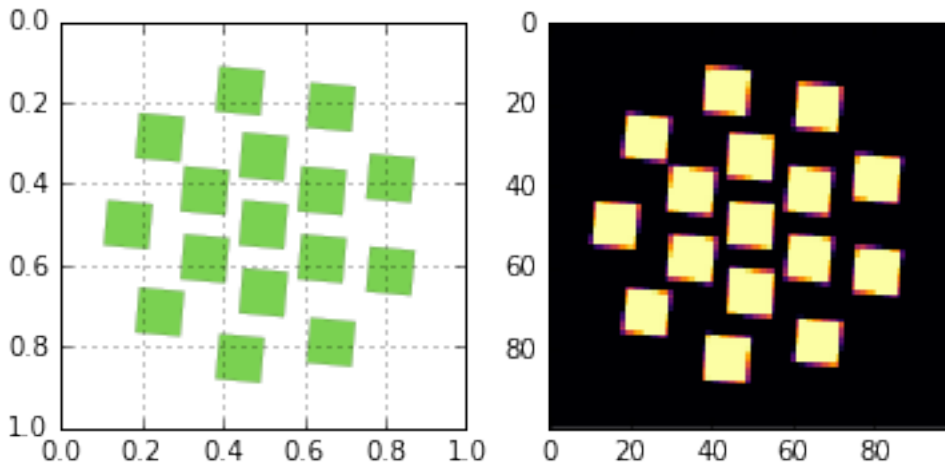
```
from xdesign import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
```

```
p = SlantedSquares(count=16, angle=5/360*2*np.pi, gap=0.01)
sidebyside(p)
plt.savefig('SlantedSquares_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
```

```

    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()

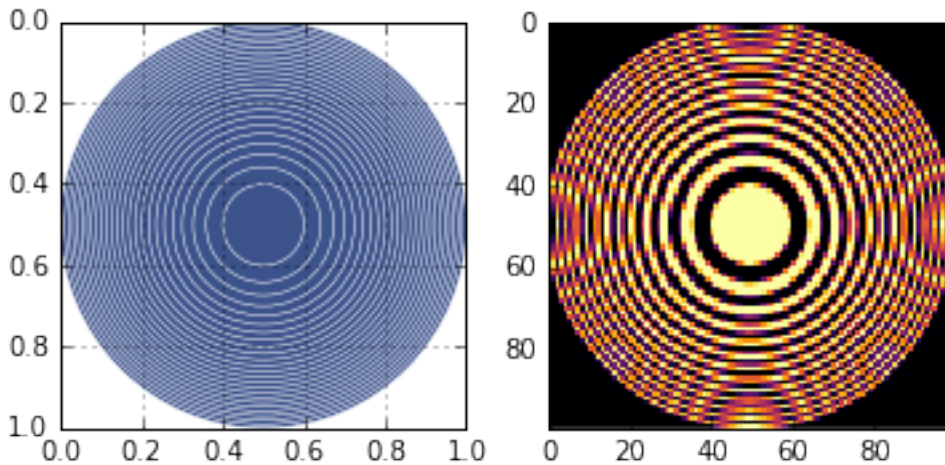
```



```

h = HyperbolicConcentric()
sidebyside(h)
plt.savefig('HyperbolicConcentric_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()

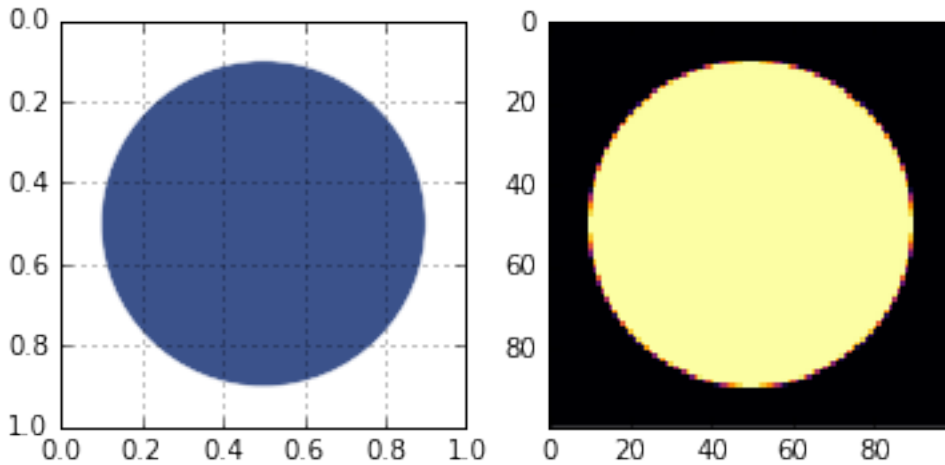
```



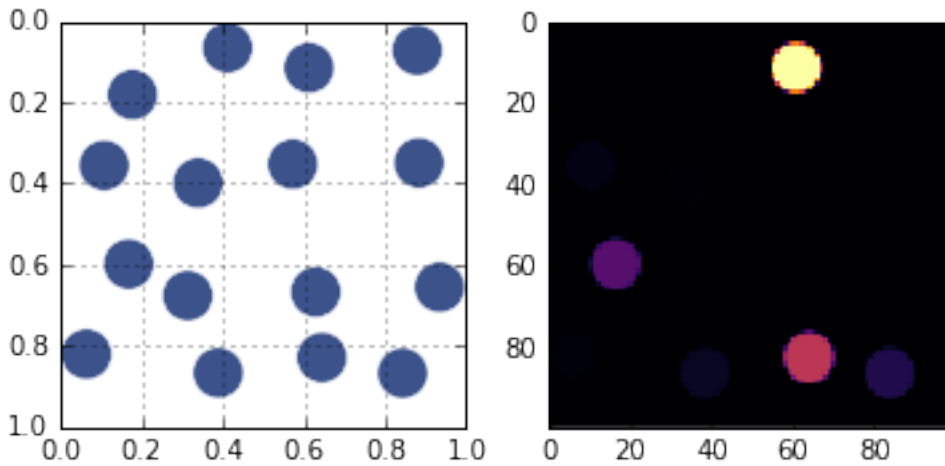
```

u = UnitCircle(radius=0.4, mass_atten=1)
sidebyside(u)
plt.savefig('UnitCircle_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()

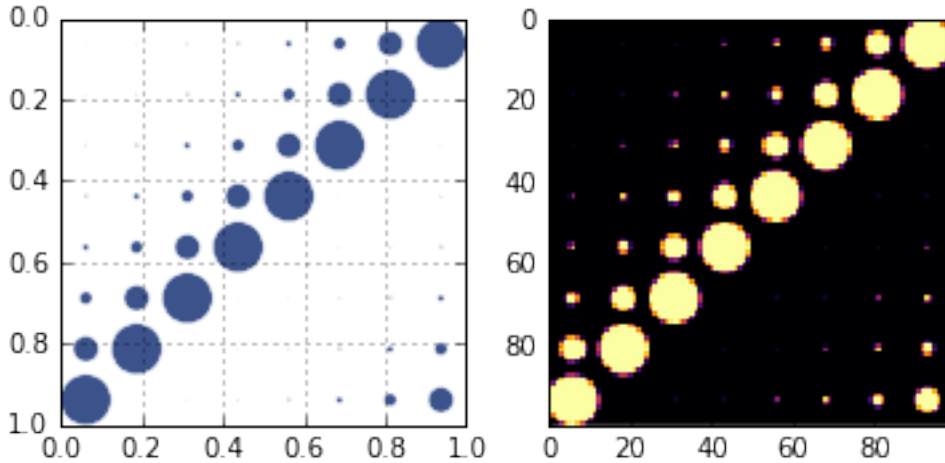
```



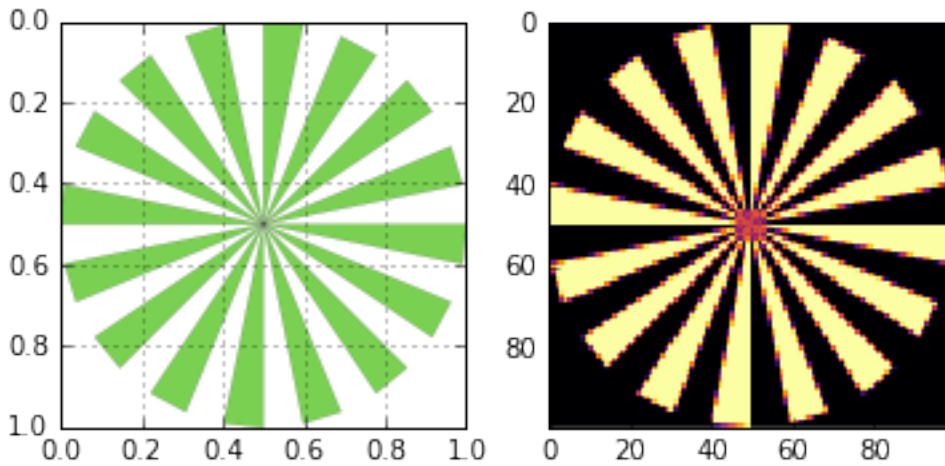
```
d = DynamicRange(steps=16, jitter=True, shape='square')
sidebyside(d)
plt.savefig('DynamicRange_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



```
l = DogaCircles(n_sizes=8, size_ratio=0.5, n_shuffles=0)
l.rotate(np.pi/2, Point([0.5, 0.5]))
sidebyside(l)
plt.savefig('DogaCircles_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



```
s = SiemensStar(32)
sidebyside(s)
plt.savefig('SiemensStar_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
```



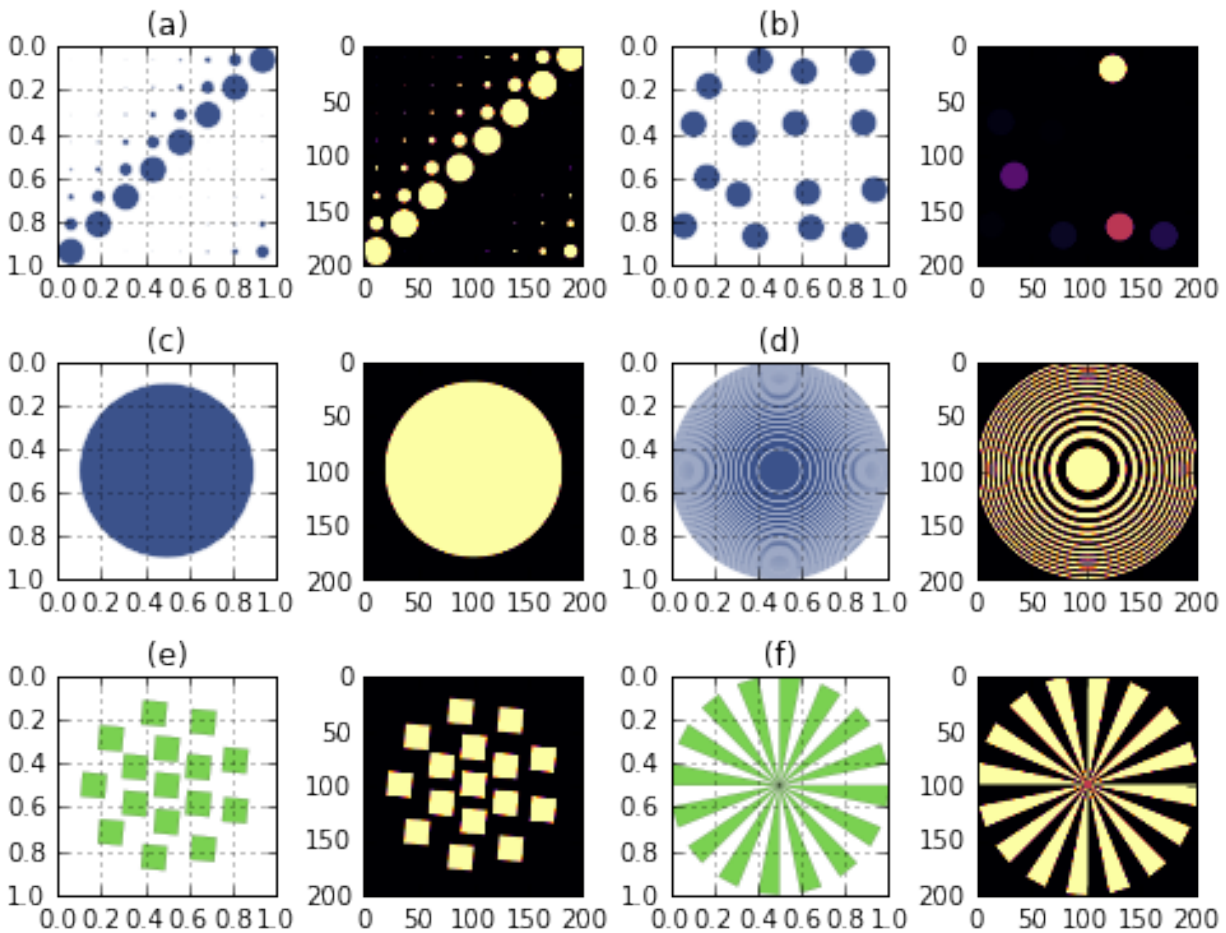
```
fig = plt.figure(figsize=(8, 6), dpi=600)
gs1 = gridspec.GridSpec(3, 4)
gs1.update(wspace=0.4, hspace=0.4) # set the spacing between axes.
phantoms = [l, d, u, h, p, s]
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
for i in range(0, len(phantoms)):
    axis = plt.subplot(gs1[2*i], aspect=1)
    plt.grid('on')
    plt.gca().invert_yaxis()
    plot_phantom(phantoms[i], axis=axis)
    plt.title('(' + letters[i] + ')')
    plt.subplot(gs1[2*i+1], aspect=1)
    plt.imshow(discrete_phantom(phantoms[i], 200), cmap='inferno')

plt.savefig('standard_patterns.png', dpi='figure',
```

```

orientation='landscape', papertype=None, format=None,
transparent=True, bbox_inches='tight', pad_inches=0.0,
frameon=False)
plt.show()

```



2.6 Wet Circles

```

import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt
from xdesign import *
from skimage.exposure import adjust_gamma, rescale_intensity

def rescale(reconstruction, hi):
    I = rescale_intensity(reconstruction, out_range=(0., 1.))
    return adjust_gamma(I, 1, hi)

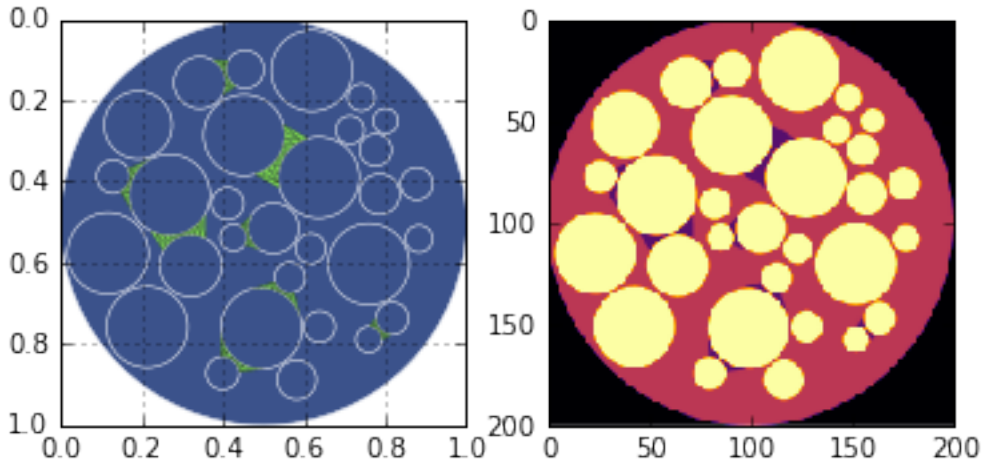
```

```

wet = WetCircles()
sidebyside(wet, size=200)
plt.savefig('Wet_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,

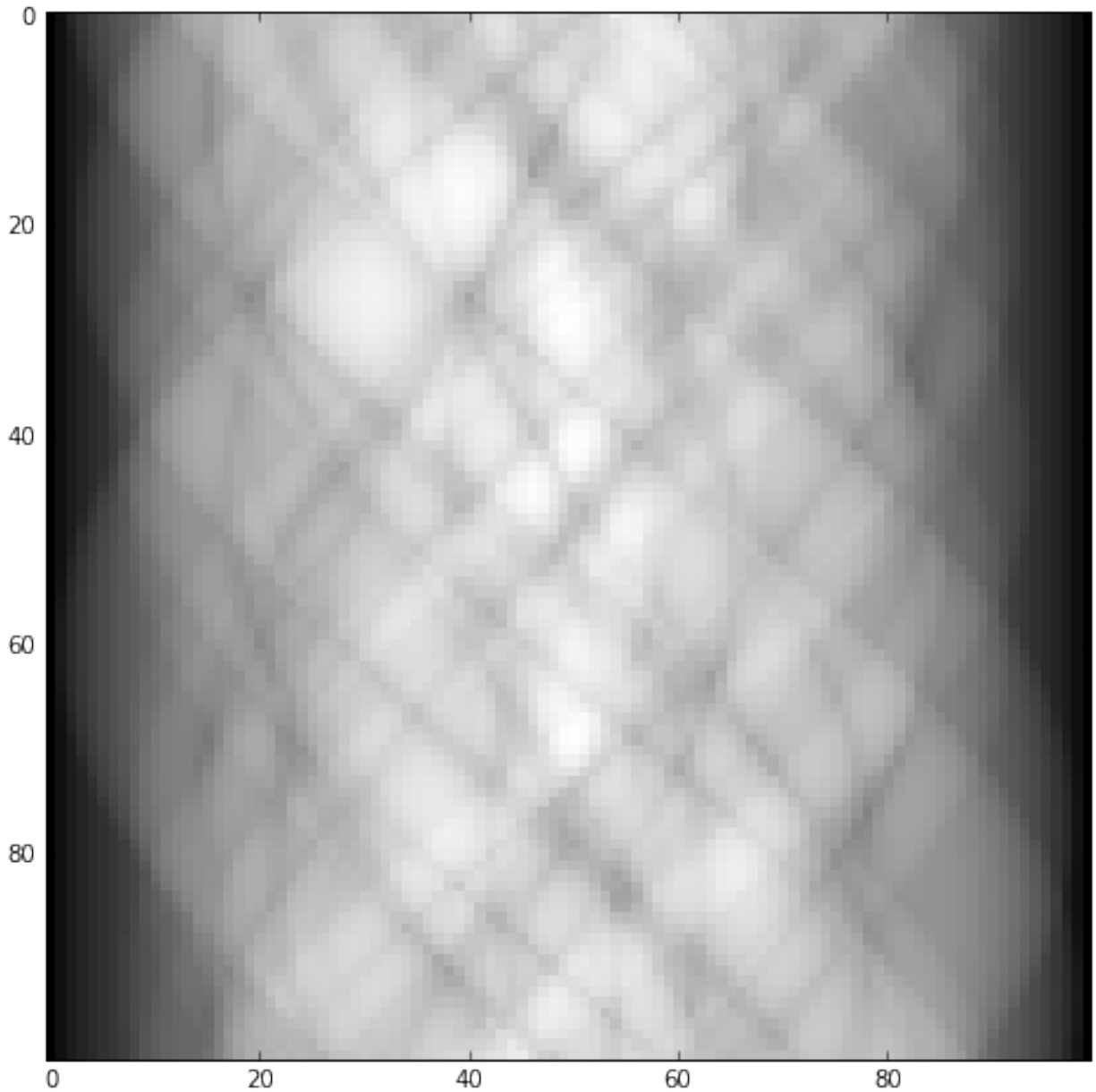
```

```
frameon=False)
plt.show(block=True)
```



```
sx, sy = 100, 100
step = 1. / sy
prb = Probe(Point([step / 2., -10]), Point([step / 2., 10]), step)
theta = np.pi / sx
sino = np.zeros(sx * sy)
a = 0
for m in range(sx):
    for n in range(sy):
        update_progress((m*sy + n)/(sx*sy))
        sino[a] = prb.measure(wet)
        a += 1
        prb.translate(step)
    prb.translate(-1)
    prb.rotate(theta, Point([0.5, 0.5]))

plt.figure(figsize=(8, 8))
plt.imshow(np.reshape(sino, (sx, sy)), cmap='gray', interpolation='nearest')
plt.show(block=True)
```

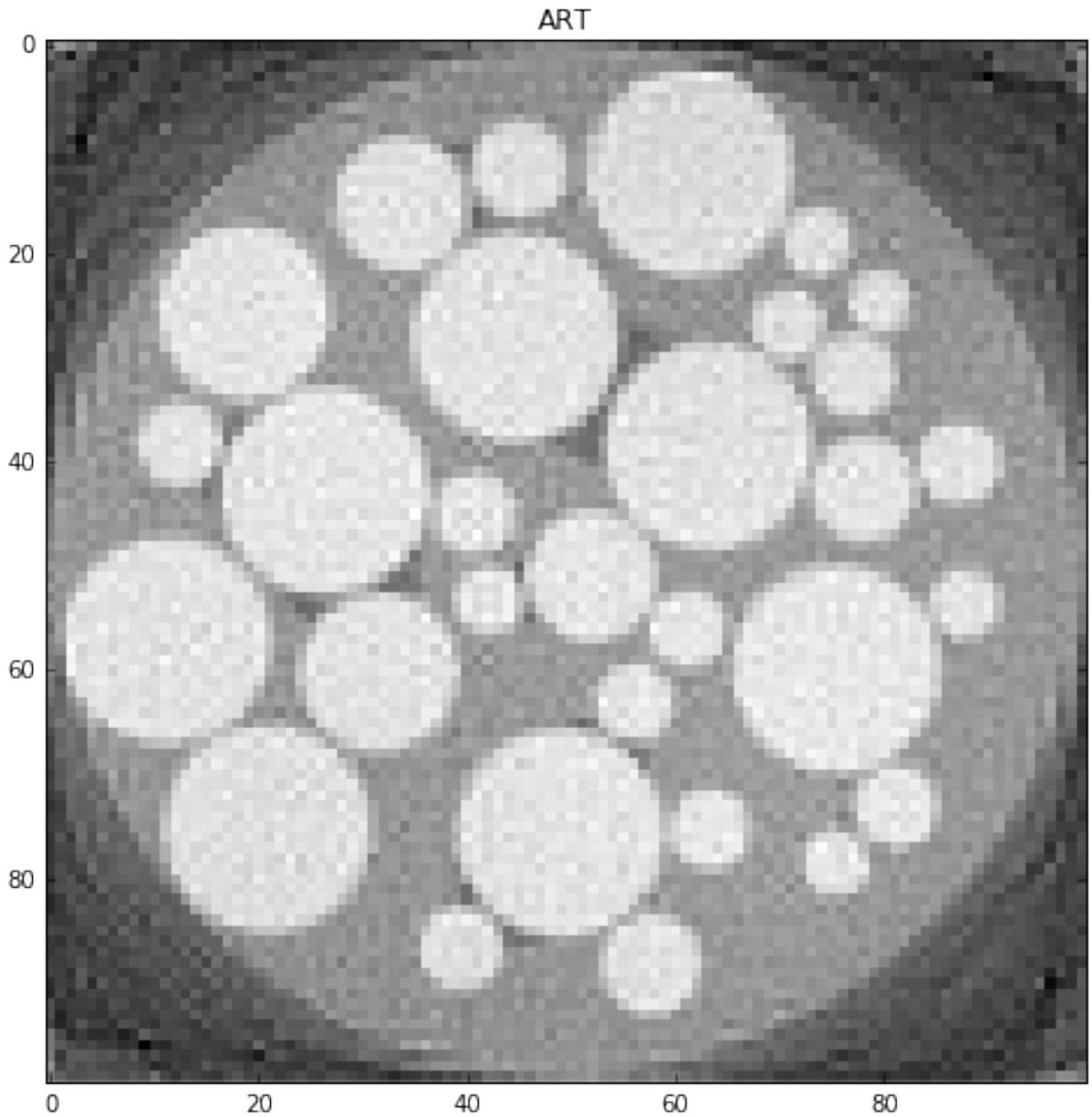


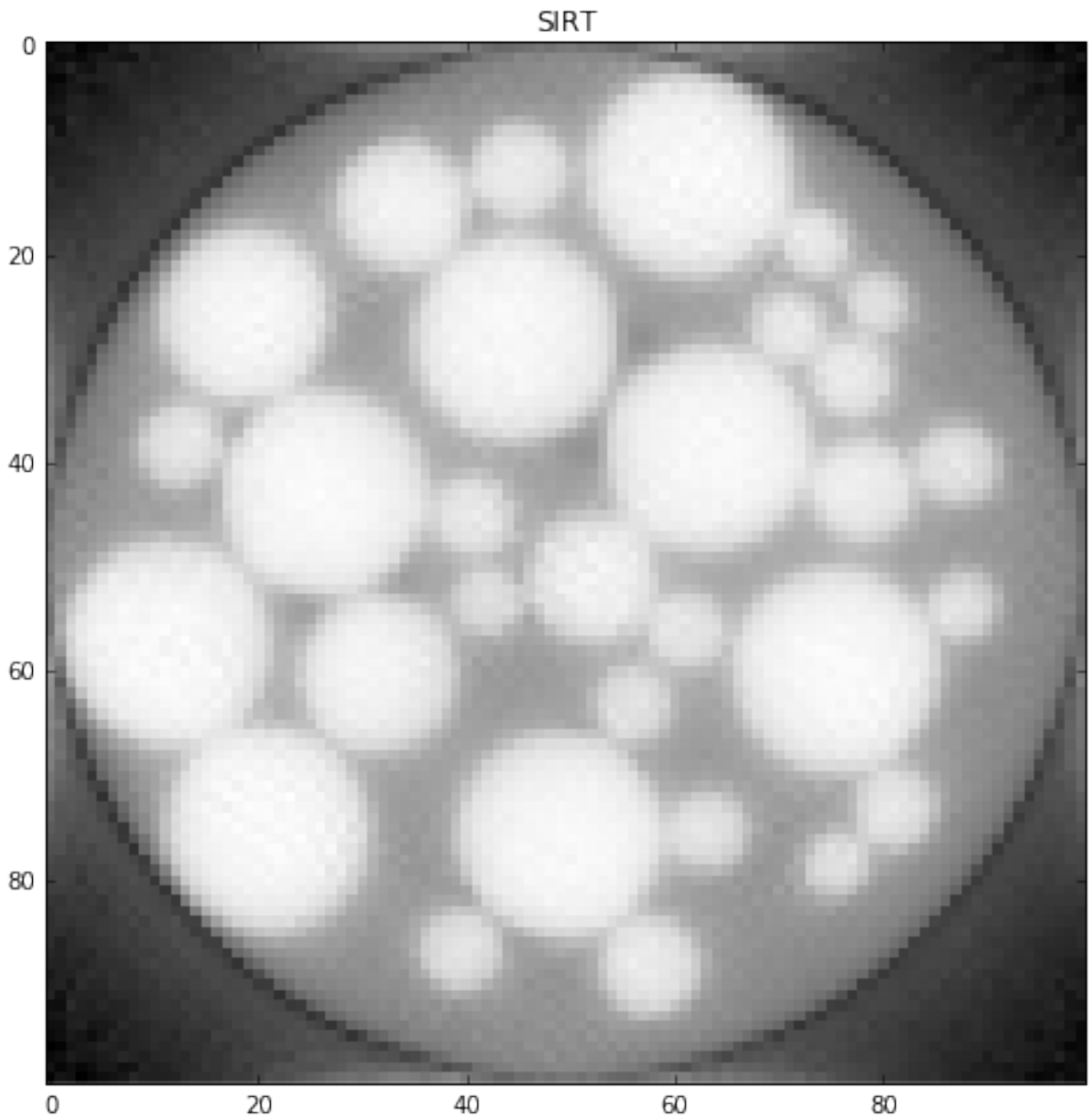
```
hi = 1
niter = 20
# Reconstruct object.
init = 1e-12 * np.ones((sx, sy))
rec_art = art(prb, sino, init, niter)
rec_art = rescale(np.rot90(rec_art)[::-1], hi)
plt.figure(figsize=(8, 8))
plt.imshow(rec_art, cmap='gray', interpolation='none')
plt.title('ART')

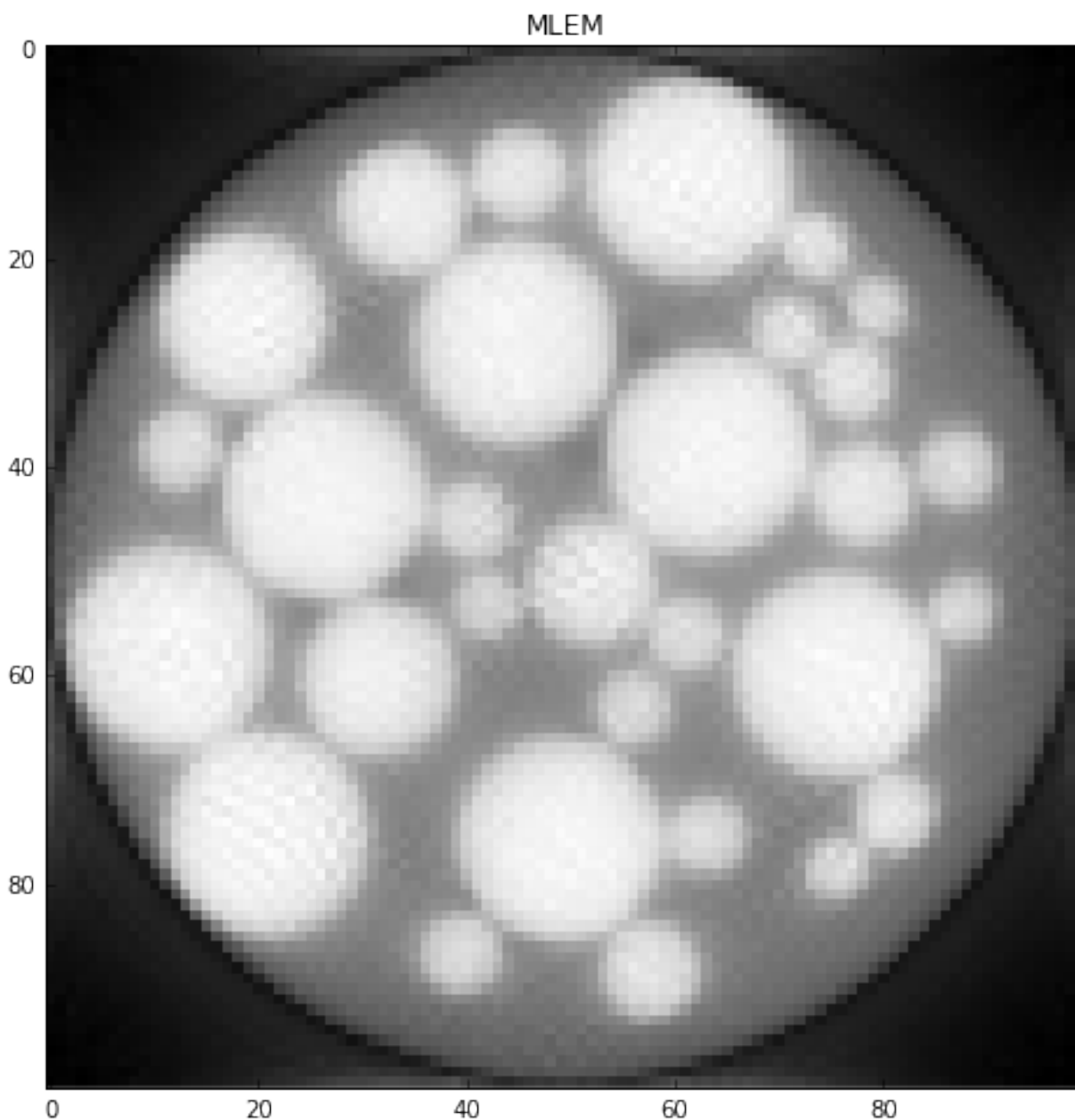
init = 1e-12 * np.ones((sx, sy))
rec_sirt = sirt(prb, sino, init, niter)
rec_sirt = rescale(np.rot90(rec_sirt)[::-1], hi)
plt.figure(figsize=(8, 8))
plt.imshow(rec_sirt, cmap='gray', interpolation='none')
```

```
plt.title('SIRT')

init = 1e-12 * np.ones((sx, sy))
rec_mlem = mlem(prb, sino, init, niter)
rec_mlem = rescale(np.rot90(rec_mlem)[::-1], hi)
plt.figure(figsize=(8, 8))
plt.imshow(rec_mlem, cmap='gray', interpolation='none')
plt.title('MLEM')
plt.show()
```







References

Features

- Configurable analytic 2D phantoms.
- Various visualization tools for statistics.
- Analytic projection operators.

Contribute

- Issue Tracker: <https://github.com/tomography/xdesign/issues>
- Documentation: <https://github.com/tomography/xdesign/tree/master/doc>
- Source Code: <https://github.com/tomography/xdesign/tree/master/xdesign>
- Tests: <https://github.com/tomography/xdesign/tree/master/test>

License

The project is licensed under the [BSD-3](#) license.

Indices and tables

- `genindex`
- `modindex`
- `search`

-
- [1] Stefano Agostinelli and Gabriella Paoli. An object oriented fully 3d tomography visual toolkit. *Computer Methods and Programs in Biomedicine*, 65(1):61 – 69, 2001. URL: <http://www.sciencedirect.com/science/article/pii/S0169260700001012>, doi:[http://dx.doi.org/10.1016/S0169-2607\(00\)00101-2](http://dx.doi.org/10.1016/S0169-2607(00)00101-2).
- [2] R Bayford, Y Hanquan, K Boone, and D S Holder. Experimental validation of a novel reconstruction algorithm for electrical impedance tomography based on backprojection of lagrange multipliers. *Physiological Measurement*, 16(3A):A237, 1995. URL: <http://stacks.iop.org/0967-3334/16/i=3A/a=022>.
- [3] Antoine Bergamaschi, Kadda Medjoubi, Cédric Messaoudi, Sergio Marco, and Andrea Somogyi. \it MMX-I: data-processing software for multimodal X-ray imaging and tomography. *Journal of Synchrotron Radiation*, 23(3):783–794, May 2016. URL: <http://dx.doi.org/10.1107/S1600577516003052>, doi:10.1107/S1600577516003052.
- [4] Xia C, Zhu K, Cao Y, Sun H, Kou B, and Wang Y. X-ray tomography study of the random packing structure of ellipsoids. *Soft Matter*, 10(7):990–996, 2014.
- [5] Lotfi Chaâri, Jean-Christophe Pesquet, Amel Benazza-Benyahia, and Philippe Ciuciu. A wavelet-based regularized reconstruction algorithm for \SENSE\ parallel \MRI\ with applications to neuroimaging. *Medical Image Analysis*, 15(2):185 – 201, 2011. URL: <http://www.sciencedirect.com/science/article/pii/S1361841510001052>, doi:<http://dx.doi.org/10.1016/j.media.2010.08.001>.
- [6] X. L. Dean-Ben, A. Buehler, V. Ntziachristos, and D. Razansky. Accurate model-based reconstruction algorithm for three-dimensional optoacoustic tomography. *IEEE Transactions on Medical Imaging*, 31(10):1922–1928, Oct 2012. doi:10.1109/TMI.2012.2208471.
- [7] James T. Dobbins. Effects of undersampling on the proper interpretation of modulation transfer function, noise power spectra, and noise equivalent quanta of digital imaging systems. *Medical Physics*, 22(2):171–181, 1995. URL: <http://scitation.aip.org/content/aapm/journal/medphys/22/2/10.1118/1.597600>, doi:<http://dx.doi.org/10.1118/1.597600>.
- [8] Saul N. Friedman, George S. K. Fung, Jeffrey H. Siewerdsen, and Benjamin M. W. Tsui. A simple approach to measure computed tomography (ct) modulation transfer function (mtf) and noise-power spectrum (nps) using the american college of radiology (acr) accreditation phantom. *Medical Physics*, 40(5):, 2013. URL: <http://scitation.aip.org/content/aapm/journal/medphys/40/5/10.1118/1.4800795>, doi:<http://dx.doi.org/10.1118/1.4800795>.
- [9] B. Kelly Han, Katharine L.R. Grant, Ross Garberich, Martin Sedlmair, Jana Lindberg, and John R. Lesser. Assessment of an iterative reconstruction algorithm (safire) on image quality in pediatric cardiac \CT\ datasets. *Journal of Cardiovascular Computed Tomography*,
-

- 6(3):200 – 204, 2012. URL: <http://www.sciencedirect.com/science/article/pii/S1934592512001402>, doi:<http://dx.doi.org/10.1016/j.jcct.2012.04.008>.
- [10] Beverley F Holman, Vesna Cuplov, Brian F Hutton, Ashley M Groves, and Kris Thielemans. The effect of respiratory induced density variations on non-tof pet quantitation in the lung. *Physics in Medicine and Biology*, 61(8):3148, 2016. URL: <http://stacks.iop.org/0031-9155/61/i=8/a=3148>.
- [11] J. Hsieh, E. Chao, J. Thibault, B. Grekowicz, A. Horst, S. McOlash, and T. J. Myers. A novel reconstruction algorithm to extend the ct scan field-of-view. *Medical Physics*, 31(9):2385–2391, 2004. URL: <http://scitation.aip.org/content/aapm/journal/medphys/31/9/10.1118/1.1776673>, doi:<http://dx.doi.org/10.1118/1.1776673>.
- [12] Jiang Hsieh, John Londt, Melissa Vass, Jay Li, Xiangyang Tang, and Darin Okerlund. Step-and-shoot data acquisition and reconstruction for cardiac x-ray computed tomography. *Medical Physics*, 33(11):4236–4248, 2006. URL: <http://scitation.aip.org/content/aapm/journal/medphys/33/11/10.1118/1.2361078>, doi:<http://dx.doi.org/10.1118/1.2361078>.
- [13] J. D. Hunter. Matplotlib: a 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.
- [14] S Jan, G Santin, D Strul, S Staelens, K Assié, D Autret, S Avner, R Barbier, M Bardiès, P M Bloomfield, D Brasse, V Breton, P Bruyndonckx, I Buvat, A F Chatziioannou, Y Choi, Y H Chung, C Comtat, D Donnarieix, L Ferrer, S J Glick, C J Groiselle, D Guez, P-F Honore, S Kerhoas-Cavata, A S Kirov, V Kohli, M Koole, M Krieguer, D J van der Laan, F Lamare, G Largeron, C Lartizien, D Lazaro, M C Maas, L Maigne, F Mayet, F Melot, C Merheb, E Pennacchio, J Perez, U Pietrzyk, F R Rannou, M Rey, D R Schaart, C R Schmidlein, L Simon, T Y Song, J-M Vieira, D Visvikis, R Van de Walle, E Wieërs, and C Morel. Gate: a simulation toolkit for pet and spect. *Physics in Medicine and Biology*, 49(19):4543, 2004. URL: <http://stacks.iop.org/0031-9155/49/i=19/a=007>.
- [15] K. P. Kostli, D. Frauchiger, J. J. Niederhauser, G. Paltauf, H. P. Weber, and M. Frenz. Optoacoustic imaging using a three-dimensional reconstruction algorithm. *IEEE Journal of Selected Topics in Quantum Electronics*, 7(6):918–923, Nov 2001. doi:10.1109/2944.983294.
- [16] Arjun S. Kumar, Pratiti Mandal, Yongjie Zhang, and Shawn Litster. Image segmentation of nanoscale zernike phase contrast x-ray computed tomography images. *Journal of Applied Physics*, 117(18):, 2015. URL: <http://scitation.aip.org/content/aip/journal/jap/117/18/10.1063/1.4919835>, doi:<http://dx.doi.org/10.1063/1.4919835>.
- [17] Thomas Köhler, Bernhard Brendel, and Ewald Roessl. Iterative reconstruction for differential phase contrast imaging using spherically symmetric basis functions. *Medical Physics*, 38(8):4542–4545, 2011. URL: <http://scitation.aip.org/content/aapm/journal/medphys/38/8/10.1118/1.3608906>, doi:<http://dx.doi.org/10.1118/1.3608906>.
- [18] Yijin Liu, Florian Meirer, Phillip A. Williams, Junyue Wang, Joy C. Andrews, and Piero Pianetta. vit TXM-Wizard: a program for advanced data collection~and evaluation in full-field transmission X-ray microscopy. *Journal of Synchrotron Radiation*, 19(2):281–287, Mar 2012. URL: <http://dx.doi.org/10.1107/S0909049511049144>, doi:10.1107/S0909049511049144.
- [19] Stefano Marchesini, Hari Krishnan, David A. Shapiro, Talita Perciano, James A. Sethian, Benedikt J. Daurer, and Filipe R. N. C. Maia. SHARP: a distributed, GPU-based ptychographic solver. *Report LBNL-1003977*, 2016. doi:10.1107/S1600576716008074.
- [20] F. Marone and M. Stampanoni. Regridding reconstruction algorithm for real-time tomographic imaging. *Journal of Synchrotron Radiation*, 19(6):1029–1037, Nov 2012. URL: <http://dx.doi.org/10.1107/S0909049512032864>, doi:10.1107/S0909049512032864.
- [21] Willem Jan Palenstijn, K Joost Batenburg, and Jan Sijbers. The astra tomography toolbox. In *13th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE*, volume 2013. 2013.

- [22] Al-Raoush R and Willson CS. Extraction of physically realistic pore network properties from three-dimensional synchrotron x-ray microtomography images of unconsolidated porous media systems. *Journal of Hydrology*, 300(1):44–64, 2005.
- [23] Al-Raoush R, Thompson K, and Willson CS. Comparison of network generation techniques for unconsolidated porous media. *Soil Science Society of America Journal*, 67(6):1687–1700, 2003.
- [24] Ranadhir Roy, Alan B Thompson, Anuradha Godavarty, and Eva M Sevik-Muraca. Tomographic fluorescence imaging in tissue phantoms: a novel reconstruction algorithm and imaging geometry. *IEEE transactions on medical imaging*, 24(2):137–154, 2005.
- [25] H. R. Sheikh and A. C. Bovik. Image information and visual quality. *IEEE Transactions on Image Processing*, 15(2):430–444, Feb 2006. doi:10.1109/TIP.2005.859378.
- [26] A Sufian, AR Russell, AJ Whittle, and M Saadatfar. Pore shapes, volume distribution and orientations in monodisperse granular assemblies. *Granular Matter*, 17(6):727–742, 2015.
- [27] Kris Thielemans, Charalampos Tsoumpas, Sanida Mustafovic, Tobias Beisel, Pablo Aguiar, Nikolaos Dikaos, and Matthew W Jacobson. Stir: software for tomographic image reconstruction release 2. *Physics in Medicine and Biology*, 57(4):867, 2012. URL: <http://stacks.iop.org/0031-9155/57/i=4/a=867>.
- [28] Bradley E. Treeby and B. T. Cox. K-wave: matlab toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*, 15(2):021314–021314–12, 2010. URL: <http://dx.doi.org/10.1117/1.3360308>, arXiv:, doi:10.1117/1.3360308.
- [29] Matthias Vogelgesang, Tomas Farago, Thilo F. Morgeneyer, Lukas Helfen, Tomy dos Santos Rolo, Anton Myagotin, and Tilo Baumbach. Real-time image-content-based beamline control for smart 4D X-ray imaging. *Journal of Synchrotron Radiation*, 23(5):, Sep 2016. URL: <http://dx.doi.org/10.1107/S1600577516010195>, doi:10.1107/S1600577516010195.
- [30] Zhou Wang and Alan C Bovik. Modern image quality assessment. *Synthesis Lectures on Image, Video, and Multimedia Processing*, 2(1):1–156, 2006.
- [31] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, 1398–1402. Ieee, 2003.
- [32] Jie Wu and Jiabi Chen. A phase retrieval algorithm for x-ray phase contrast imaging. *Optik - International Journal for Light and Electron Optics*, 124(9):864 – 866, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S0030402612001635>, doi:10.1016/j.ijleo.2012.02.030.
- [33] Zhaozheng Yin, Takeo Kanade, and Mei Chen. Understanding the phase contrast optics to restore artifact-free microscopy images for segmentation. *Medical Image Analysis*, 16(5):1047 – 1062, 2012. URL: <http://www.sciencedirect.com/science/article/pii/S1361841512000035>, doi:10.1016/j.media.2011.12.006.
- [34] Daniele Zanaga, Folkert Bleichrodt, Thomas Altantzis, Naomi Winckelmans, Willem Jan Palenstijn, Jan Sijbers, Bart de Nijs, Marijn A van Huis, Ana Sánchez-Iglesias, Luis M Liz-Marzán, and others. Quantitative 3d analysis of huge nanoparticle assemblies. *Nanoscale*, 8(1):292–299, 2016.

X

- `xdesign`, [51](#)
- `xdesign.acquisition`, [3](#)
- `xdesign.algorithms`, [4](#)
- `xdesign.feature`, [5](#)
- `xdesign.geometry`, [6](#)
- `xdesign.material`, [10](#)
- `xdesign.metrics`, [13](#)
- `xdesign.phantom`, [17](#)
- `xdesign.plot`, [18](#)

Symbols

`_x` (xdesign.geometry.Point attribute), 7

A

`add_property()` (xdesign.feature.Feature method), 5
`add_quality()` (xdesign.metrics.ImageQuality method), 14
`angle` (xdesign.material.SlantedSquares attribute), 12
`angle_scan()` (in module xdesign.acquisition), 4
`angleogram()` (in module xdesign.acquisition), 4
`append()` (xdesign.geometry.Mesh method), 9
`append()` (xdesign.phantom.Phantom method), 17
`area` (xdesign.feature.Feature attribute), 5
`area` (xdesign.geometry.Circle attribute), 8
`area` (xdesign.geometry.Polygon attribute), 8
`area` (xdesign.geometry.Rectangle attribute), 9
`area` (xdesign.geometry.Triangle attribute), 9
`area` (xdesign.phantom.Phantom attribute), 17
`art()` (in module xdesign.algorithms), 5
`atom_concentration` (xdesign.material.Material attribute), 10
`atomic_form_factor` (xdesign.material.Material attribute), 10

B

`Beam` (class in xdesign.acquisition), 3
`bounds` (xdesign.geometry.Polygon attribute), 8

C

`center` (xdesign.feature.Feature attribute), 5
`center` (xdesign.geometry.Circle attribute), 7
`center` (xdesign.geometry.Mesh attribute), 9
`center` (xdesign.geometry.Rectangle attribute), 9
`center` (xdesign.geometry.Triangle attribute), 9
`Circle` (class in xdesign.geometry), 7
`circumference` (xdesign.geometry.Circle attribute), 8
`collision()` (xdesign.geometry.Entity method), 6
`collision()` (xdesign.geometry.Point method), 7
`compton_cross_section` (xdesign.material.Material attribute), 10

`compton_scattering` (xdesign.material.Material attribute), 10
`compute_background_ttest()` (in module xdesign.metrics), 15
`compute_likeness()` (in module xdesign.metrics), 14
`compute_mtf()` (in module xdesign.metrics), 15
`compute_mtf_siemens()` (in module xdesign.metrics), 16
`compute_neq()` (in module xdesign.metrics), 16
`compute_nps()` (in module xdesign.metrics), 15
`compute_PCC()` (in module xdesign.metrics), 14
`compute_quality()` (in module xdesign.metrics), 14
`contains()` (xdesign.geometry.Circle method), 8
`contains()` (xdesign.geometry.Entity method), 6
`contains()` (xdesign.geometry.Mesh method), 9
`contains()` (xdesign.geometry.Point method), 7
`contains()` (xdesign.geometry.Polygon method), 9
`count` (xdesign.material.SlantedSquares attribute), 12

D

`density` (xdesign.phantom.Phantom attribute), 17
`diameter` (xdesign.geometry.Circle attribute), 8
`dim` (xdesign.geometry.Entity attribute), 6
`dim` (xdesign.geometry.Point attribute), 7
`discrete_phantom()` (in module xdesign.plot), 19
`distance()` (xdesign.geometry.Entity method), 6
`distance()` (xdesign.geometry.Line method), 8
`distance()` (xdesign.geometry.Point method), 7
`DogaCircles` (class in xdesign.material), 11
`DynamicRange` (class in xdesign.material), 11

E

`electron_density` (xdesign.material.Material attribute), 10
`Electronics` (class in xdesign.material), 13
`Entity` (class in xdesign.geometry), 6

F

`Feature` (class in xdesign.feature), 5
`feature` (xdesign.phantom.Phantom attribute), 17
`FiberComposite` (class in xdesign.material), 13
`Foam` (class in xdesign.material), 13

G

gap (xdesign.material.SlantedSquares attribute), 12
 geometry (xdesign.feature.Feature attribute), 5

H

half_space (xdesign.acquisition.Beam attribute), 3
 half_space (xdesign.geometry.Mesh attribute), 9
 half_space (xdesign.geometry.Polygon attribute), 9
 HyperbolicConcentric (class in xdesign.material), 11

I

ImageQuality (class in xdesign.metrics), 13
 import_triangle() (xdesign.geometry.Mesh method), 9
 insert() (xdesign.phantom.Phantom method), 17
 intercept() (xdesign.geometry.Line method), 8

L

Line (class in xdesign.geometry), 8
 linear_attenuation (xdesign.material.Material attribute), 10
 list (xdesign.geometry.Circle attribute), 8
 list (xdesign.geometry.Polygon attribute), 9
 list (xdesign.phantom.Phantom attribute), 17
 load() (xdesign.phantom.Phantom method), 17

M

maps (xdesign.metrics.ImageQuality attribute), 13
 mass_atten (xdesign.feature.Feature attribute), 5
 mass_ratio() (xdesign.material.Material method), 11
 Material (class in xdesign.material), 10
 measure() (xdesign.acquisition.Probe method), 3
 Mesh (class in xdesign.geometry), 9
 Metal (class in xdesign.material), 13
 midpoint() (xdesign.geometry.Entity method), 6
 mlem() (in module xdesign.algorithms), 5
 multiroll() (in module xdesign.plot), 19

N

n_levels (xdesign.material.SlantedSquares attribute), 12
 norm (xdesign.geometry.Point attribute), 7
 number_of_elements() (xdesign.material.Material method), 11
 numpy (xdesign.geometry.Polygon attribute), 9
 numpy() (xdesign.phantom.Phantom method), 17
 numverts (xdesign.geometry.Polygon attribute), 9

O

orig (xdesign.metrics.ImageQuality attribute), 13

P

p1 (xdesign.acquisition.Beam attribute), 3
 p1 (xdesign.geometry.Line attribute), 8
 p2 (xdesign.acquisition.Beam attribute), 3

p2 (xdesign.geometry.Line attribute), 8
 patch (xdesign.geometry.Circle attribute), 8
 patch (xdesign.geometry.Mesh attribute), 10
 patch (xdesign.geometry.Polygon attribute), 9
 perimeter (xdesign.geometry.Polygon attribute), 9
 Phantom (class in xdesign.phantom), 17
 photoelectric_absorption (xdesign.material.Material attribute), 11
 photoelectric_cross_section (xdesign.material.Material attribute), 11
 plot_curve() (in module xdesign.plot), 19
 plot_feature() (in module xdesign.plot), 18
 plot_histograms() (in module xdesign.plot), 21
 plot_mesh() (in module xdesign.plot), 19
 plot_metrics() (in module xdesign.plot), 21
 plot_mtf() (in module xdesign.plot), 21
 plot_neq() (in module xdesign.plot), 21
 plot_nps() (in module xdesign.plot), 21
 plot_phantom() (in module xdesign.plot), 18
 plot_polygon() (in module xdesign.plot), 19
 Point (class in xdesign.geometry), 6
 Polygon (class in xdesign.geometry), 8
 pop() (xdesign.geometry.Mesh method), 10
 pop() (xdesign.phantom.Phantom method), 17
 population (xdesign.phantom.Phantom attribute), 17
 probability_mask() (in module xdesign.metrics), 14
 Probe (class in xdesign.acquisition), 3

Q

qualities (xdesign.metrics.ImageQuality attribute), 13

R

radii (xdesign.material.DogaCircles attribute), 11
 radii (xdesign.material.HyperbolicConcentric attribute), 11
 radius (xdesign.feature.Feature attribute), 6
 radius (xdesign.geometry.Circle attribute), 8
 radius_per_level (xdesign.material.SlantedSquares attribute), 12
 raster_scan() (in module xdesign.acquisition), 4
 ratio (xdesign.material.SiemensStar attribute), 13
 recon (xdesign.metrics.ImageQuality attribute), 13
 reconstruct() (in module xdesign.algorithms), 5
 record() (xdesign.acquisition.Probe method), 3
 Rectangle (class in xdesign.geometry), 9
 reduced_energy_ratio (xdesign.material.Material attribute), 11
 refractive_index (xdesign.material.Material attribute), 11
 reverse() (xdesign.phantom.Phantom method), 17
 rotate() (xdesign.feature.Feature method), 6
 rotate() (xdesign.geometry.Entity method), 6
 rotate() (xdesign.geometry.Mesh method), 10
 rotate() (xdesign.geometry.Point method), 7
 rotate() (xdesign.geometry.Polygon method), 9

rotate() (xdesign.phantom.Pantom method), 17

S

save() (xdesign.phantom.Pantom method), 17

scale() (xdesign.geometry.Circle method), 8

scale() (xdesign.geometry.Entity method), 6

scale() (xdesign.geometry.Mesh method), 10

scale() (xdesign.geometry.Point method), 7

scales (xdesign.metrics.ImageQuality attribute), 14

shape (xdesign.phantom.Pantom attribute), 17

side_length (xdesign.material.SlantedSquares attribute), 12

sidebyside() (in module xdesign.plot), 19

SiemensStar (class in xdesign.material), 12

sinogram() (in module xdesign.acquisition), 4

sirt() (in module xdesign.algorithms), 5

size (xdesign.acquisition.Beam attribute), 3

SlantedSquares (class in xdesign.material), 12

SoftBiomaterial (class in xdesign.material), 13

Soil (class in xdesign.material), 12

sort() (xdesign.metrics.ImageQuality method), 14

sort() (xdesign.phantom.Pantom method), 17

sprinkle() (xdesign.phantom.Pantom method), 17

Square (class in xdesign.geometry), 9

squares_per_level (xdesign.material.SlantedSquares attribute), 12

standard (xdesign.geometry.Line attribute), 8

stream() (in module xdesign.algorithms), 5

T

translate() (xdesign.acquisition.Probe method), 4

translate() (xdesign.feature.Feature method), 6

translate() (xdesign.geometry.Entity method), 6

translate() (xdesign.geometry.Mesh method), 10

translate() (xdesign.geometry.Point method), 7

translate() (xdesign.geometry.Polygon method), 9

translate() (xdesign.phantom.Pantom method), 18

Triangle (class in xdesign.geometry), 9

U

UnitCircle (class in xdesign.material), 12

update_progress() (in module xdesign.algorithms), 5

V

vertices (xdesign.geometry.Polygon attribute), 8

volume (xdesign.feature.Feature attribute), 6

W

WetCircles (class in xdesign.material), 12

widths (xdesign.material.HyperbolicConcentric attribute), 11

X

x (xdesign.geometry.Point attribute), 7

x (xdesign.material.DogaCircles attribute), 11

xdesign (module), 21, 51

xdesign.acquisition (module), 3

xdesign.algorithms (module), 4

xdesign.feature (module), 5

xdesign.geometry (module), 6

xdesign.material (module), 10

xdesign.metrics (module), 13

xdesign.phantom (module), 17

xdesign.plot (module), 18

xintercept (xdesign.geometry.Line attribute), 8

Y

y (xdesign.geometry.Point attribute), 7

y (xdesign.material.DogaCircles attribute), 11

yintercept (xdesign.geometry.Line attribute), 8

Z

z (xdesign.geometry.Point attribute), 7