Module **LinearAlgebra**

Functions

```
def createFaces(vertIdx1, vertIdx2, closed=False, flipped=False)
```

```
def create_mesh_object(context, verts, edges, faces, name)
```

```
def draw_parametric_surface(eq, range_u_min, range_u_max, range_u_step, range_v_min, range_v_max, range_v_step,
                            name, wrap_u=False, wrap_v=False, close_v=False)
```

Classes

```
class Color (r, g, b, name)
```

Class that defines a color in RGB format

```
class Colors
```

Class that defines a list of colors by name

Class variables

```
var colorsbyname
```

Static methods

```
def color(name)
```

Function that returns a color from his name

Parameters

name: name of the color

```
def colors(names)
```

Return a list of colors fron their names

Parameters

names: list of names

```
class LinearAlgebra
```

Class used to define all the functions in this module to work with graphics in Blender

Initializes the values for scene, objects, meshes, collection, etc.

Methods

```
def add_ligth(self, location=[0, 0, 100], energy=3, direction=[0, 0, -1])
```

Adds a ligth to the scene

Parameters

location: location point of the light energy: energy of the ligth direction: direction of the light

```
def add_ligths(self, energy=1)
```

Adds diferent lights to the scene

Parameters

energy: energy of the lights

```
def add_material(self, obj, material_name, r, g, b, opacity=1.0)
```

Adds a material and color to an object

Parameters

obj: object material_name: material's name r, g, b: RGB color values opacity: the opacity

```
def animate_revolution_surface(self, fun=None, tmin=0.0, tmax=1.0, steps=256, curvethicknes=0.025,
                               thickness=0.025, frames=3, angle=3, radians=False, axis='Z',
                               symmetry=None, name='Revolution surface', color='AzureBlueDark',
                               point=None)
```

Draws and animates a revolution surface from a curve

Parameters

fun: parametric equacion of the curve steps: number of steps to graw the curve curvethicknes: thickness of the curve frames: number of frames at each step of revolution angle: step angle of the revolution radians: if True, angle must be in radians axis: axis of revolution. It must be 'X', 'Y' or 'Z' name: name of the surface color: color of the surface point: if not None draw three points and a cercle. Must be a float between tmax and tmin

```
def base_cilinder(self)
```

Draws a base cilinder with radius 1 and depth 1

```
def base_cone(self)
```

Draws a base cone with radius1=1.5, radius2=0, depth=2

```
def base_is_canonica(self)
```

Returns True if sel.base is the canonical basis

```
def comp_times_vector(self, u, v)
```

Computes the vectorial product u x v

Parameters

u, v: two Vectors

```
def cone(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, c2=1, half=False, principal=True,
         canonica=True, color='AzureBlueDark', name='Cone', xmax=None, cmax=15, pmax=15, thickness=0.02,
         opacity=1.0)
```

Draws a cone

Parameters

o: center of the cone u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2, c2: squares of semi-axes of the cone. The equation is x'^2/a^2 + y'^2/b^2 - z'^2/c^2 = 0 half: if True draws half cone principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the cone xmax: maximum value of the x coordinate cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the cone opacity: opacity of the cone

```
def curve(self, fun=None, tmin=0.0, tmax=1.0, steps=25, thickness=0.01, name='Curve', color='White',
axis=False, zaxis=True, o=Vector((0.0, 0.0, 0.0)), u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0,
1.0, 0.0)), symmetry=None, change=False)
```

Draws a curve in a reference R' determined by the origin o and basis {v1, v2, v3} constructed from u1 and u2 and the symmetric curve or curves from the parameter 'symmetry'

Parameters

fun: the parametric function tmin: minimum value of the parameter tmax: maximum value of the parameter steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis o: origin of the reference R' u1, u2: vectors to construct the basis {v1, v2, v3} symmetry: list of values in ('XY','XZ','YZ','X','Y','Z','O'). For every value S, draw the symmetric curve respect to S change: if True, set the reference self.orifin, self.base to {o; v1, v2, v3}

```
def delete_base_cilinder(self)
```

Removes the base cilinder

```
def delete_base_cone(self)
```

Removes the base cone

```
def draw_base_axis(self, scale=0.05, head_height=0.15, axis=0, name='Axis', positive=True, zaxis=True)
```

Draws a reference axis given by self.origin, self.rotation and the basis self.base

Parameters

scale: scale of the cylinder head_height: height of the head of the vector from self.base axis: length of the coordinate axis. If the length is 0, only the basis vectors are drawn name: name of the result object positive: if True, draw the positive part of the axis zaxis: if True, draw the z axis

```
def draw_circle(self, center=[0, 0, 0], u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)),
                axis=False, zaxis=False, radius=1, steps=25, thickness=0.01, name='Circle',
                color='White', change=False)
```

Draws a circle of center 'center' and radius 'radius' in the plane determined by vectors u1 and u2

Parameters

center: center of the circle u1, u2: vectors to construct the basis {v1, v2, v3} axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis radius: radius of the circle steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve change: if True, set the reference self.orifin, self.base to {o; v1, v2, v3}

```
def draw_components(self, vector=None, color='Cyan', name='Components', scale=0.005)
```

Draws the components of the the vector 'vector' in the reference given by self.origin, self.rotation and the basis self.base

Parameters

vector: the vector color: color of the lines of components name: name of the object scale: scale of the lines

```
def draw_cone(self, a=1.0, xmin=0.0, xmax=5.0, steps=50, scale=[1, 1, 1], half=False,
              color='AzureBlueDark', name='Cone', opacity=1.0, thickness=0.05)
```

Draws a cone from the line z = a*x in the XZ plane

Parameters

a: slope of the line xmin: minimum value of x xmax: maximum value of x steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions half: if True, draws half cone color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_cube(self, origin=None, scale=[1, 1, 1], scalelines=0.05, vectors=False, color='Blue',
              linecolor='Red', vectorcolor='Black', name='Parallelepiped', opacity=1.0, thickness=0.0)
```

Draws a rectangular parallelepiped

Parameters

origin: center of the parallelepiped scale: scale of the sides of the parallelepiped scalelines: scale of the edges of the parallelepiped vectors: if True, draws vectors from the origin to the vertices color: color of the parallelepiped linecolor: color of the edges vectorcolor: color of the vectors name: name of the parallelepiped opacity: opacity of the parallelepiped thickness: thickness of the parallelepiped

```
def draw_curve(self, fun=None, tmin=0.0, tmax=1.0, steps=25, thickness=0.01, name='Curve', color='White',
               axis=False, zaxis=True, o=Vector((0.0, 0.0, 0.0)), u1=Vector((1.0, 0.0, 0.0)),
               u2=Vector((0.0, 1.0, 0.0)))
```

Draws a curve in a reference R' determined by the origin o and basis {v1, v2, v3} constructed from u1 and u2

Parameters

fun: the parametric function tmin: minimum value of the parameter tmax: maximum value of the parameter steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis o: origin of the reference R' u1, u2: vectors to construct the basis {v1, v2, v3}

```
def draw_ellipse(self, center=[0, 0, 0], u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)), a=1,
                 b=1, axis=False, zaxis=False, steps=25, thickness=0.01, name='Ellipse', color='White',
                 change=False)
```

Draws an ellipse of center 'center' and semi-axes a and b in the plane determined by vectors u1 and u2

Parameters

center: center of the ellipse u1, u2: vectors to construct the basis {v1, v2, v3} a, b: semi-axes of the ellipse axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve change: if True, set the reference self.orifin, self.base to {o; v1, v2, v3}

```
def draw_ellipsoid(self, radius=5.0, scale=[1.2, 1.8, 0.8], color='AzureBlueDark', name='Ellipsoid',
                   opacity=1.0, thickness=0.05)
```

Draws en ellipsoid

Parameters

radius: radius of the sphere scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_elliptic_cylinder(self, a=8.0, b=5.0, amin=0.0, amax=6.283185307179586, length=20, steps=200,
                           scale=[1, 1, 1], color='AzureBlueDark', name='EllipticCylinder', opacity=1.0,
                           thickness=0.05)
```

Draws an eliptic cylinder from the ellipse x = a*cos(t) y = b*sin(t) in the XY plane

Parameters

a, b: coefficients of the ellipsw amin: minimum value of the angle t amax: maximum value of the angle t length: length in the Z direction steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_elliptic_paraboloid(self, a=0.5, xmin=0.0, xmax=3.0, steps=50, scale=[1, 1, 1],
                             color='AzureBlueDark', name='EllipticParaboloid', opacity=1.0,
                             thickness=0.05)
```

Draws an elliptic paraboloid from the parabola z=a*t^2

Parameters

a: coefficient of the parabola xmin: minimum value of x xmax: maximum value of x steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_frenet_curve(self, fun=None, var=None, tmin=0.0, tmax=1.0, radius=0.1, steps=25, thickness=0.01,
                        name='Curve', color='White', point=True, tangent=False, acceleration=False,
                        normal=False, osculator=False, frenet=False, units=False, sizex=8, sizey=8)
```

Draws a curve and diferents elements related to the curve

Parameters

fun: the parametric function var = parameter variable of the function fun tmin: minimum value of the parameter tmax: maximum value of the parameter radius: radius of the point steps: number of steps frames: increment of the frame set thickness: thickness of the curve name: name of the curve color: color of the curve point: if True draw a point along the curve tangent: if True draw the tangent vector along the curve acceleration: if True draw the acceleration vector along the curve normal: if True draw the normal vector along the curve osculator: if True draw the osculating plane along the curve frenet: if True draw the Frenet trihedron along the curve units: if True normalize the tangent and normal vectors sizex, sizey: sizes of the osculating plane

```
def draw_function(self, f=None, xmin=-3, xmax=3, xsteps=64, ymin=-3, ymax=3, ysteps=64, thickness=0.02,
                  opacity=1.0, pmax=10, name='Function', color='AzureBlueDark', axis=False,
                  o=Vector((0.0, 0.0, 0.0)), u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)))
```

Draws a function of two variables f(x,y) i the reference R' = {o, v1, v2, v3}

Parameters

f: the function of two variables f(x,y) xmin: minimum value of x xmax: maximum value of x xsteps: steps in the x direction ymin: minimum value of y ymax: maximum value of y ysteps: steps in the x direction thickness: thickness of the surface opacity: opacity of the surface pmax: the axis are drawn between -pmax and pmax name: name of the surface color: color of the surface axis: if True the axis of the reference R' are drawn o: origin of the reference R' u1, u2: vectors to construct the basis {v1, v2, v3}

```
def draw_hyperbole(self, center=[0, 0, 0], u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)), a=1,
                   b=1, ymax=3.0, axis=False, zaxis=False, steps=25, thickness=0.01, name='Hyperbole',
                   color='White', change=False)
```

Draws an hyperbole of center 'center' and semi-axes a and b in the plane determined by vectors u1 and u2

Parameters

center: center of the hyperbole u1, u2: vectors to construct the basis {v1, v2, v3} a, b: semi-axes of the hyperbole ymax: maximum value of the y' axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve change: if True, set the reference self.orifin, self.base to {o; v1, v2, v3}

```
def draw_hyperbolic_cylinder(self, a=1.0, b=4.0, xmin=2.0, xmax=6.0, length=20, steps=50, scale=[1, 1,
                             1], color='AzureBlueDark', name='HyperbolicCylinder', opacity=1.0,
                             thickness=0.05)
```

Draws an hyperbolic cylinder from the hyperbole y = a * sqrt(x**2 - b) in the XY plane

Parameters

a, b: coefficients of the hyperbole xmin: minimum value of x xmax: maximum value of x length: length in the Z direction steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_hyperbolic_paraboloid(self, a=0.2, b=0.4, xmax=10.0, ymax=10.0, steps=64, scale=[1, 1, 1],
                               color='AzureBlueDark', name='HyperbolicParaboloid', opacity=1.0,
                               thickness=0.05)
```

Draws an hyperbolic paraboloid with equation z = a*x^2 - b*y^2

Parameters

a, b: coefficients of the parabolic hyperboloid xmax: maximum value of x ymax: maxim value y steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_line(self, start=[1, 1, 1], end=[10, 10, 10], scale=0.05, name='Line', color='Black')
```

Draws a line from the point start to the point end. The reference given by self.origin, self.rotation and the basis self.base is used

Parameters

start: starting point of the line end: ending point of the line scale: scale of the cylinder name: name of the object color: color of the vector

```
def draw_mesh(self, mesh=None, name='Mesh', color='Blue', opacity=1)
```

Draws a mesh. This function is used by other functions

Parameters

mesh: the mesh to be drawn name: name of the mesh color: color of the mesh opacity: opacity of the mesh

```
def draw_one_sheet_hyperboloid(self, a=2.0, b=2.0, xmin=1.4142135623730951, xmax=5.0, steps=256, scale=
                               [1, 1, 1], color='AzureBlueDark', name='HyperboloidOneSheet', opacity=1.0,
                               thickness=0.05)
```

Draws a one sheet hyperboloid from the hyperbole z = \pm a*sqrt(x^2-b) in the XZ plane

Parameters

a, b: coefficients of the hyperbole xmin: minimum value of x xmax: maximum value of x steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_parabola(self, vertex=[0, 0, 0], u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)), a=1,
                  xmax=3.0, axis=False, zaxis=False, steps=25, thickness=0.01, name='Parabola',
                  color='White', change=False)
```

Draws a parabola of vertex 'vertex' of equation y'=ax'^2 in the reference {vertex; v1, v2, v3} determined by vectors u1 and u2

Parameters

vertex: vertex of the parabola u1, u2: vectors to construct the basis {v1, v2, v3} a: coefficient of the parabola xmax: maximum value of x' axis: if True draws the axis of the reference R' zaxis: if True draws the z' axis steps: number of steps thickness: thickness of the curve name: name of the curve color: color of the curve change: if True, set the reference self.orifin, self.base to {o; v1, v2, v3}

```
def draw_parabolic_cylinder(self, p=0.25, xmin=0.0, xmax=6.0, length=20, steps=50, scale=[1, 1, 1],
                            color='AzureBlueDark', name='ParabolicCylinder', opacity=1.0, thickness=0.05)
```

Draws a parabolic cylinder from the parabola z=p*x^2 in the XZ plane

Parameters

p: coefficients of the parabola xmin: minimum value of x xmax: maximum value of x length: length in the Y direction steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_parallelepiped(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], u3=[0, 0, 1],
                        scalelines=0.025, color='AzureBlueDark', linecolor='OrangeObscureDull',
                        name='Parallelepiped', opacity=1.0, thickness=0.0)
```

Draws a parallelepiped

Parameters

origin: base vertex of the parallelepiped u1, u2, u3: vectors that gives the edges scalelines: scale of the edges of the parallelepiped color: color of the parallelepiped linecolor: color of the edges name: name of the parallelepiped opacity: opacity of the parallelepiped thickness: thickness of the parallelepiped

```
def draw_parallelogram(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], scalelines=0.025,
                      color='AzureBlueDark', linecolor='OrangeObscureDull', name='Parallelogram',
                      opacity=1.0, thickness=0.0)
```

Draws a parallelogram

Parameters

origin: base vertex of the parallelogram u1, u2: vectors that gives the edges scalelines: scale of the edges of the parallelogram color: color of the parallelogram linecolor: color of the edges name: name of the parallelogram opacity: opacity of the parallelogram thickness: thickness of the parallelogram

```
def draw_plane(self, normal=None, base=None, sizex=10, sizey=10, color='AzureBlueDark', name='Plane',
               opacity=1.0, thickness=0.01)
```

Draws a plane with normal vector or base vectors. It passes through the point self.origin. Only normal or base can be not None

Parameters

normal: normal vector to the plane base: list of two independent vectors sizex: x-size of the plane sizey: y-size of the plane color: color of the plane name: name of the plane opacity: opacity of the plane thickness: thickness of the plane

```
def draw_plane_surface(self, origin=None, normal=None, base=None, sizex=10, sizey=10, vectors=False,
                       scalelines=0.05, scalevector=0.01, color='AzureBlueDark',
                       linecolor='BlueDarkDull', vectorcolor='Black', name='Plane', opacity=1.0,
                       thickness=0.0)
```

Draws a plane with normal vector or base vectors. It passes through the point origin. Only normal or base can be not None

Parameters

origin: a point in the plane normal: normal vector to the plane base: list of two independent vectors sizex: x-size of the plane sizey: y-size of the plane vectors: if True, draw the generators of the plane scalelines: scale of the lines limiting the plane scalevector: scale of the generators color: color of the plane linecolor: color of the lines limiting the plane vectorcolor: color of the generators name: name of the plane opacity: opacity of the plane thickness: thickness of the plane

```
def draw_point(self, radius=0.2, location=(0, 0, 0), name='Point', color='Black', opacity=1.0)
```

Draws a point (in the reference self.origin, self.base)

Parameters

radius: radius of the point location: location of the point name: name of the point color: color of the point opacity: opacity of the point

```
def draw_points(self, points=[], name='Points', color='Blue', opacity=1)
```

Draws a list of points

Parameters

origin: points: list of points name: name of the list of points color: color of the points opacity: opacity of the points

```
def draw_polygon(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], points=[[0, 0], [1, 0], [0, 1]],
                 scalelines=0.075, color='AzureBlueDark', linecolor='OrangeObscureDull', name='Polygon',
                 opacity=1.0, thickness=0.0, vectors=None, scalevectors=0.01)
```

Draws a polygon

Parameters

origin: base vertex of the polygon u1, u2: base vectors for the polygon points: list of coordinates of points. The coordinates are taken in the reference {origin; u1, u2} scalelines: scale of the edges of the polygon color: color of the polygon linecolor: color of the edges name: name of the polygon opacity: opacity of the polygon thickness: thickness of the polygon

```
def draw_pyramid(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], u3=[0.5, 0.5, 1], scalelines=0.025,
                color='AzureBlueDark', linecolor='OrangeObscureDull', name='Pyramid', opacity=1.0,
                thickness=0.0)
```

Draws a pyramid

Parameters

origin: base vertex of the pyramid u1, u2, u3: vectors that gives the edges scalelines: scale of the edges of the pyramid color: color of the pyramid linecolor: color of the edges name: name of the pyramid opacity: opacity of the pyramid thickness: thickness of the pyramid

```
def draw_regular_polygon(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], vertexs=5, radius=1,
                         scalelines=0.075, color='AzureBlueDark', linecolor='OrangeObscureDull',
                         name='RegularPolygon', opacity=1.0, thickness=0.0, vectors=None)
```

Draws a regular polygon

Parameters

origin: base vertex of the polygon u1, u2: base vectors for the polygon vertexs: number of vertices of the polygon radius: radius of the polygon scalelines: scale of the edges of the polygon color: color of the polygon linecolor: color of the edges name: name of the polygon opacity: opacity of the polygon thickness: thickness of the polygon

```
def draw_simple_curve(self, fun=None, tmin=0.0, tmax=1.0, steps=25, thickness=0.02, color='White',
                      name='Curve')
```

Draws a parametric curve

Parameters

fun: the parametric function tmin: minimum value of the parameter tmax: maximum value of the parameter steps: number of steps thickness: thickness of the curve color: color of the curve name: name of the curve

```
def draw_surface(self, eq=None, umin=-1, umax=1, usteps=64, vmin=-1, vmax=1, vsteps=64, thickness=0.02,
                 opacity=1.0, pmax=10, name='Surface', color='AzureBlueDark', axis=False, o=Vector((0.0,
                 0.0, 0.0)), u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)), wrap_u=False,
                 wrap_v=False, close_v=False)
```

Draws a parametric surface in the reference R'

Parameters

eq: parametric equacion f(u,v) umin: minimum value of u umax: maximum value of u usteps: steps in the u direction vmin: minimum value of v vmax: maximum value of v vsteps: steps in the v direction thickness: thickness of the surface opacity: opacity of the surface color: color of the surface pmax: the principal axis are drawn between -cmax and cmax name: name of the surface color: color of the surface axis: if True draw the axis of the reference {o, v1, v2, v3} o: origin of the reference R' u1, u2: vectors to construct the basis {v1, v2, v3} scale: scale coefficients wrap_u: wrap the u coordinate wrap_v: wrap the u coordinate close_v: close the v coordinate

```
def draw_tetrahedron(self, origin=[0, 0, 0], u1=[2, 0, 0], u2=[1.0000000000000002, 1.7320508075688772,
                     0], u3=[1.0, 0.5773502691896257, 2], scalelines=0.025, color='AzureBlueDark',
                     linecolor='OrangeObscureDull', name='Tetrahedron', opacity=1.0, thickness=0.0)
```

Draws a tetrahedron

Parameters

origin: base vertex of the tetrahedron u1, u2, u3: vectors that gives the edges scalelines: scale of the edges of the tetrahedron color: color of the tetrahedron linecolor: color of the edges name: name of the tetrahedron opacity: opacity of the tetrahedron thickness: thickness of the tetrahedron

```
def draw_triangle(self, origin=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], points=[[0, 0], [1, 0], [0, 1]],
                  scalelines=0.075, color='AzureBlueDark', linecolor='OrangeObscureDull',
                  name='Triangle', opacity=1.0, thickness=0.0)
```

Draws a triangle. It's a polygon with three vertices

Parameters

origin: base vertex of the triangle u1, u2: base vectors for the triangle points: list of coordinates of points. The coordinates are taken in the reference {origin; u1, u2} scalelines: scale of the edges of the triangle color: color of the triangle linecolor: color of the edges name: name of the triangle opacity: opacity of the triangle thickness: thickness of the triangle

```
def draw_two_sheets_hyperboloid(self, a=2.0, b=1.0, xmin=0.0, xmax=5.0, steps=50, scale=[1, 1, 1],
                                color='AzureBlueDark', name='HyperboloidTwoSheets', opacity=1.0,
                                thickness=0.05)
```

Draws a two sheet hyperboloid from the hyperbole z = \pm a * math.sqrt(x**2+b) in the XZ plane

Parameters

a, b: coefficients of the hyperbole xmin: minimum value of x xmax: maximum value of x steps: numbers of steps to draw the parabola scale: scaling factors in the X, Y and Z directions color: color of the surface name: name of the surface opacity: opacity of the surface thickness: thickness of the surface

```
def draw_vector(self, vector=None, canonica=False, color='Black', scale=0.05, arrow=True,
                head_height=0.15, axis=0, name='Vector', positive=True)
```

Draw the vector with components 'vector'

Parameters

vector: components of the vector canonica: if True, the components are in the canonical basis, else they are in the basis self.base. Finally, self.rotation is applied color: color of the vector scale: scale of the cylinder arrow: if True draws the vector itself head_height: height of the head of the vector axis: if not zero, draw also the line generated by the vector positive: if axis is not zero and positive is True, draw only the positive part of the line generated by the vector

```
def draw_vector_field(self, f=None, xmin=-3, xmax=3, xsteps=8, ymin=-3, ymax=3, ysteps=8, zmin=-3,
                      zmax=3, zsteps=8, name='Vector Field', color='Red', scale=0.02, head_height=0.05)
```

Draws a vector field

Parameters

f: the vector field xmin: minimum value of x xmax: maximum value of x xsteps: steps in the x direction ymin: minimum value of y ymax: maximum value of y ysteps: steps in the y direction zmin: minimum value of z zmax: maximum value of z zsteps: steps in the z direction name: name of the vector field color: color of the vector field scale: scale of the vectors head_height: head height of the vectors

```
def draw_vectors(self, vectors=[], canonica=False, color='Black', scale=0.05, head_height=0.2,
                 name='Vectors', axis=0)
```

Draws a list of vectors.

Parameters

vectors: list of vectors canonica: if True, the the vectors are expressed in the canonical basis. color: color of the vectors scale: scale of the cylinder head_height: height of the head of the vector axis: if not zero, draw also the line generated by every vector

```
def ellipsoid(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, c2=1, principal=True,
              canonica=True, color='AzureBlueDark', name='Ellipsoid', cmax=15, pmax=15, thickness=0.02,
              opacity=1.0)
```

Draws an ellipsoid

Parameters

o: center of the ellipsoid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors in the following way v1 = u1 v2 = u2 - u2.project(v1) v1.normalize() v2.normalize() v3 = v1.cross(v2) a2, b2, c2: squares of semi-axes of the ellipsoid. The equation is x'^2/a^2 + y'^2/b^2 + z'^2/c^2 = 1 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the ellipsoid cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -pmax and pmax thickness: thickness of the ellipsoid opacity: opaccity of the ellipsoid

```
def elliptic_cylinder(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, principal=True,
                      canonica=True, color='AzureBlueDark', name='EllipticCylinder', zmax=20, cmax=15,
                      pmax=15, thickness=0.02, opacity=1.0)
```

Draws an elliptic cylinder

Parameters

o: center of the elliptic cylinder u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2: squares of semi-axes of the elliptic cylinder. The equation is x'^2/a^2 + y'^2/b^2 = 1 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the elliptic cylinder zmax: the elliptic cylinder is drawn between -zmax and zmax cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the elliptic cylinder opacity: opacity of the elliptic cylinder

```
def elliptic_paraboloid(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, principal=True,
                        canonica=True, color='AzureBlueDark', name='EllipticParaboloid', xmax=None,
                        cmax=15, pmax=15, thickness=0.02, opacity=1.0)
```

Draws an elliptic paraboloid

Parameters

o: vertex of the elliptic paraboloid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2: squares of semi-axes of the elliptic paraboloid. The equation is z = x'^2/a^2 + y'^2/b^2 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the elliptic paraboloid xmax: maximum value of the coordinate x cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the elliptic paraboloid opacity: opacity of the elliptic paraboloid

```
def hyperbolic_cylinder(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, principal=True,
                        canonica=True, color='AzureBlueDark', name='HyperbolicCylinder', xmax=None,
                        zmax=20, cmax=15, pmax=15, thickness=0.02, opacity=1.0)
```

Draws an hyperbolic cylinder

Parameters

o: center of the hyperbolic cylinder u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2: squares of semi-axes of the hyperbolic cylinder. The equation is x'^2/a^2 - y'^2/b^2 = 1 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the hyperbolic cylinder xmax: maximum value of the x coordinate zmax: the hyperbolic cylinder is drawn between -zmax and zmax cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the hyperbolic cylinder opacity: opacity of the hyperbolic cylinder

```
def hyperbolic_paraboloid(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, principal=True,
                          canonica=True, color='AzureBlueDark', name='HyperbolicParaboloid', xmax=None,
                          ymax=None, cmax=15, pmax=15, thickness=0.02, opacity=1.0)
```

Draws an hyperbolic paraboloid

Parameters

o: vertex of the hyperbolic paraboloid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2: squares of semi-axes of the hyperbolic paraboloid. The equation is z = x'^2/a^2 - y'^2/b^2 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the elliptic paraboloid xmax: maximum value of the coordinate x ymax: maximum value of the coordinate y cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the hyperbolic paraboloid opacity: opacity of the hyperbolic paraboloid

```
def join(self, list)
```

Joins a list of objects

Parameters

list: list of objects

```
def new_components(self, vector=None)
```

Returns the components of the vector 'vector' in the basis determined by self.rotation ans the basis self.base

Parameters

vector: components of the vector in the canonical basis

```
def new_coordinates(self, point=None)
```

Returns the coordinates of the point 'point' in the reference determined by self.origin, self.rotation and the basis self.base

Parameters

point: coordinates of the point in the canonical reference

```
def one_sheet_hyperboloid(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, c2=1,
                          principal=True, canonica=True, color='AzureBlueDark',
                          name='OneSheetHyperboloid', xmax=None, cmax=15, pmax=15, thickness=0.02,
                          opacity=1.0)
```

Draws an one sheet hyperboloid

Parameters

o: center of the hyperboloid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2, c2: squares of semi-axes of the hyperboloid. The equation is x'^2/a^2 + y'^2/b^2 - z'^2/c^2 = 1 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the hyperboloid xmax: maximum value of the x coordinate cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the hyperboloid opacity: opacity of the hyperboloid

```
def parabolic_cylinder(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a=1, principal=True, canonica=True,
                       color='AzureBlueDark', name='ParabolicCylinder', xmax=None, ymax=30, cmax=20,
                       pmax=20, thickness=0.02, opacity=1.0)
```

Draws an hyperbolic paraboloid

Parameters

o: vertex of the hyperbolic paraboloid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2: squares of semi-axes of the hyperbolic paraboloid. The equation is z = x'^2/a^2 - y'^2/b^2 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the elliptic paraboloid xmax: maximum value of the coordinate x ymax: maximum value of the coordinate y cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the hyperbolic paraboloid opacity: opacity of the hyperbolic paraboloid

```
def reset(self)
```

Resets origin, base, rotation, frames and colors

```
def reset_base(self)
```

Sets self.base to the canonical basis

```
def reset_colors(self)
```

Set self.colors to default colors

```
def reset_frames(self)
```

Set self.frame to 0

Parameters

name: name of a color

```
def reset_origin(self)
```

Sets the origin to the point (0,0,0)

```
def reset_rotation(self)
```

Sets the rotation to identity, i.e., rotation of 0 degrees around the vector (1,0,0)

```
def revolution_surface(self, fun=None, tmin=0.0, tmax=1.0, o=Vector((0.0, 0.0, 0.0)), u1=Vector((1.0,
                       0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)), pmax=0, steps=256, thickness=0.025,
                       axis='Z', name='Revolution surface', color='AzureBlueDark')
```

Draws a revolution surface from a curve in the reference R'

Parameters

fun: parametric equacion of the curve steps: number of steps axis: axis of revolution. It must be 'X', 'Y' or 'Z' o: origin of the reference R' u1, u2: vectors to construct the basis {v1, v2, v3} pmax: the principal axis are drawn between -pmax and pmax color: color of the surface

```
def rotate_euler(self, obj=None, psi=0.0, theta=0.0, phi=0.0, frames=3, axis='ZXZ', amax=15,
                 scaleaxis=0.075, reverse=False, local=False, radians=False, positive=True)
```

Rotates an object by the Euler angles psi, theta and phi

Parameters

object: the object psi, theta, phi: the Euler angles expressed in degrees axis: it must be 'XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX', 'XYX', 'XZX', 'YXY', 'YZY', 'ZXZ' or 'ZYZ' amax: axis valur for draw_base_axis scaleaxis: scale value for draw_base_axis local: if True the center of rotation is the location of the object radians: if True, psi, theta and phi must be in radians positive: if False and psi, theta or phi are greather than 180 degrees, they rae converted to negative angles

```
def rotate_object(self, obj=None, axis='Z', frames=1, origin=Vector((0.0, 0.0, 0.0)), localaxis=None,
                  localangle=None, helicoidal=0.0, rounds=1)
```

Rotates an object around the axis

Parameters

obj: the object axis: it must be 'X', 'Y', 'Z' or a Vector local: if True the center of rotation is the location of the object

```
def rotate_object_by_axis_angle(self, obj=None, axis=Vector((1.0, 0.0, 0.0)), angle=90, amax=15,
                                frames=1, scaleaxis=0.075, local=False)
```

Rotates an object around an angle 'angle' around the axis

Parameters

obj: the object axis: any non nul Vectors angle: the angle of rotation in degrees frames: increment of the frame set scaleaxis: scale value for draw_base_axis local: if True the center of rotation is the location of the object

```
def rotate_vector(self, vector=None, axis='Z')
```

Rotates a vector around the axis

Parameters

vector: the vector axis: it must be 'X', 'Y', 'Z' or a vector

```
def set_base(self, base, orthonormal=False)
```

Sets the self.base, i.e., the basis of the reference coordinates used to display objects

Parameters

base: list of three vectors orthonormal: if True, the Gram-Schmidt method is applied and the vectors are normalized.

`def set_colors(self, names)`

Set self.colors to the list of colors with names 'names'

Parameters

names: list of name colors

`def set_cursor(self, origin=[0, 0, 0], direction=[1, 0, 0], axis='x')`

Sets the cursor position and direction

Parameters

origin: position of the cursor direction: vector that indicates the direction of the axis 'axis' axis: 'x', 'y' or 'z'

`def set_cursor_rotation(self, origin=[0, 0, 0], rotation=Matrix(((1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0))))`

Sets the rotation of the cursor

Parameters

origin: position of the cursor rotation: matrix of a rotation

`def set_default_color(self, name)`

Set self.defaultcolor to the color with name 'name'

Parameters

name: name of a color

`def set_origin(self, vector=[0, 0, 0])`

Sets the origin of the reference coordinates used to display objects.

Parameters

vector: origin's position

`def set_rotation(self, angle=None, vector=None, quaternion=None)`

Sets self.rotation to the rotation defined by an angle and an axis or by a quaternion.

Parameters

angle: angle of rotation in degrees vector: axis of rotation quaternion: quaternion that defines a rotation The angle and vector takes precedence over the quaternion

`def simple_curve(self, f=None, tmin=0.0, tmax=1.0, steps=25, name='Simple curve', symmetry=None, draw=False)`

Return a curve defined by the parametrization f

Parameters

f: Parametrization of the curve tmin: minimum value of the parameter tmax: maximum value of the parameter steps: number of steps name: name of the curve symmetry: None or a value in the list ('XY','XZ','YZ','X','Y','Z','O'). Symmetry of the curve draw: if True, the curve is drawn

`def sphere(self, o=[0, 0, 0], r2=1, principal=True, canonica=True, color='AzureBlueDark', name='Sphere', cmax=15, pmax=15, thickness=0.02, opacity=1.0)`

Draws a sphere of center 'o' and radius squared equal to 'r2'

Parameters

o: center of the sphere r2: radius squared principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the sphere cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the sphere opacity: opacity of the sphere

```
def two_sheets_hyperboloid(self, o=[0, 0, 0], u1=[1, 0, 0], u2=[0, 1, 0], a2=1, b2=1, c2=1,
                          principal=True, canonica=True, color='AzureBlueDark',
                          name='TwoSheetParaboloid', xmax=None, cmax=15, pmax=15, thickness=0.02,
                          opacity=1.0)
```

Draws a two sheets hyperboloid

Parameters

o: center of the hyperboloid u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors a2, b2, c2: squares of semi-axes of the hyperboloid. The equation is x'^2/a^2 + y'^2/b^2 - z'^2/c^2 = -1 principal: if True, the principal axis are drawn canonica: if True, the canonical axis are drawn color: color of the surface name: name of the hyperboloid xmax: maximum value of the x coordinate cmax: the canonical axis are drawn between -cmax and cmax pmax: the principal axis are drawn between -cmax and cmax thickness: thickness of the hyperboloid opacity: opacity of the hyperboloid

```
def vectors_to_quaternion(self, u1=Vector((1.0, 0.0, 0.0)), u2=Vector((0.0, 1.0, 0.0)))
```

Returns the quaternion correspondint to the base {v1,v2,v3} u1, u2: the principal basis {v1, v2, v3} is constructed from this vectors in the following way v1 = u1 v2 = u2 - u2.project(v1) v1.normalize() v2.normalize() v3 = v1.cross(v2)

```
class Rotation (angle=None, vector=None, axis=None, quaternion=None, radians=False)
```

Class used for work with rotations. The stored value in the class is a quaternion

Initializes the value for a rotation

Parameters

angle: angle of rotation vector: axis of rotation quaternion: The quaternion itself radians: must be True if the angle is entered in radians and False if the is entered in degrees.

Static methods

```
def from_euler_angles(psi, theta, phi, axis='ZXZ', radians=False)
```

Initializes a rotation from its Euler angles in the order ZXZ

Parameters

phi, theta, psi: Euler angles axis: it must be 'XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX', 'XYX', 'XZX', 'YXY', 'YZY', 'ZXZ' or 'ZYZ' radians: if radians, psi, theta and must be in radians

Methods

```
def apply(self, v)
```

Applies the rotation to an object v Parameters: v: any object that can be transformed by a rotation

```
def to_axis_angle(self, radians=False)
```

Returns the axis and angle of the rotation

Parameters

radians: if True, the angle returned is in radians, if not, is returned in degrees

```
def to_euler_angles(self, axis='ZXZ', randomize=False, radians=False)
```

Returns the Euler angles according to axis 'axis'

Parameters

axis: it must be 'XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX', 'XYX', 'XZX', 'YXY', 'YZY', 'ZXZ' or 'ZYZ' radians: if True, the angle returned is in radians, if not, is returned in degrees

Index

Functions

Classes

draw_regular_polygon
draw_simple_curve
draw_surface
draw_tetrahedron
draw_triangle
draw_two_sheets_hyperboloid
draw_vector
draw_vector_field
draw_vectors
ellipsoid
elliptic_cylinder
elliptic_paraboloid
hyperbolic_cylinder
hyperbolic_paraboloid
join
new_components
new_coordinates
one_sheet_hyperboloid
parabolic_cylinder
reset
reset_base
reset_colors
reset_frames
reset_origin
reset_rotation
revolution_surface
rotate_euler
rotate_object
rotate_object_by_axis_angle
rotate_vector
set_base
set_colors
set_cursor
set_cursor_rotation
set_default_color
set_origin
set_rotation
simple_curve
sphere
two_sheets_hyperboloid
vectors_to_quaternion

Rotation

apply
from_euler_angles
to_axis_angle
to_euler_angles

Generated by *pdoc* 0.10.0.