geometry_functions

March 30, 2024

1 geometry functions allow a *design* to be created more easily

the geometry functions here allow points to be created (part I), measured (part II) and moved/copied (part III)

geometry functions work best when x, y and z are all defined for all points

'tau' (equal to 2*pi) is used throughout this notebook - see tau section of the design tips notebook for further details

<this document is a jupyter notebook - if they're new to you, check out how they work: link, link, link>

run all cells in this notebook in order (keep pressing shift+enter)

```
[ ]: import fullcontrol as fc
     from math import tau, sin
```

1.1 I. create points

single points: - points defined by polar coordinates - midpoint of two points

lists of points: - arc - arc with variable radius - elliptical arc - rectangle - circle - polygon - spiral - helix (optionally with variable radius) - squarewave - segmented line

midpoint() and polar_to_point() (point defined by polar coordinates)

```
[ ]: pt1 = fc.Point(x=0,y=0,z=0)
     pt2 = fc.Point(x=0,y=10,z=0)
     pt3 = fc.polar_to_point(pt2, 10, tau/8)
     pt4 = fc.midpoint(pt1, pt2)
     steps = [pt1, pt2, pt3, pt4]
     steps.append(fc.PlotAnnotation(point=pt4, label="midpoint between point 1 and␣
       ↪point 2"))
     steps.append(fc.PlotAnnotation(point=pt1, label="point 1"))
     steps.append(fc.PlotAnnotation(point=pt2, label="point 2"))
     steps.append(fc.PlotAnnotation(point=pt3, label="point defined by polar␣
       ↪coordinates relative to point 2"))
     fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
       ↪style='line'))
```

rectangle

```
[ ]: start_point = fc.Point(x=10, y=10, z=0)
x_size = 10
y_size = 5
clockwise = True
steps = fc.rectangleXY(start_point, x_size, y_size, clockwise)
steps.append(fc.PlotAnnotation(point=steps[-1], label="start/end"))
steps.append(fc.PlotAnnotation(point=steps[1], label="first point after start"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
  ↪style='line'))
```

circle

```
[ ]: centre_point = fc.Point(x=10, y=10, z=0)
radius = 10
start_angle = 0
segments = 32
clockwise = True
steps = fc.circleXY(centre_point, radius, start_angle, segments, clockwise)
steps.append(fc.PlotAnnotation(point=steps[-1], label="start/end"))
steps.append(fc.PlotAnnotation(point=steps[1], label="first point after start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
  ↪style='line'))
```

ellipse

```
[ ]: centre_point = fc.Point(x=10, y=10, z=0)
a = 10
b = 5
start_angle = 0
segments = 32
clockwise = True
steps = fc.ellipseXY(centre_point, a, b, start_angle, segments, clockwise)
steps.append(fc.PlotAnnotation(point=steps[-1], label="start/end"))
steps.append(fc.PlotAnnotation(point=steps[1], label="first point after start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
  ↪style='line'))
```

polygon

```
[ ]: centre_point = fc.Point(x=10, y=10, z=0)
enclosing_radius = 10
start_angle = 0
sides = 6
clockwise = True
steps = fc.polygonXY(centre_point, enclosing_radius, start_angle, sides,␣
  ↪clockwise)
```

```
steps.append(fc.PlotAnnotation(point=steps[-1], label="start/end"))
steps.append(fc.PlotAnnotation(point=steps[1], label="first point after start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```

arc

[ ]:
```
centre_point = fc.Point(x=10, y=10, z=0)
radius = 10
start_angle = 0
arc_angle = 0.75*tau
segments = 64
steps = fc.arcXY(centre_point, radius, start_angle, arc_angle, segments)
steps.append(fc.PlotAnnotation(point=steps[-1], label="end"))
steps.append(fc.PlotAnnotation(point=steps[0], label="start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```

variable arc

[ ]:
```
centre_point = fc.Point(x=10, y=10, z=0)
radius = 10
start_angle = 0
arc_angle = 0.75*tau
segments = 64
radius_change = -6
z_change = 0
steps = fc.variable_arcXY(centre_point, radius, start_angle, arc_angle,␣
 ↪segments, radius_change, z_change)
steps.append(fc.PlotAnnotation(point=steps[-1], label="end"))
steps.append(fc.PlotAnnotation(point=steps[0], label="start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```

elliptical arc

[ ]:
```
centre_point = fc.Point(x=10, y=10, z=0)
a = 10
b = 5
start_angle = 0
arc_angle = 0.75*tau
segments = 64
steps = fc.elliptical_arcXY(centre_point, a, b, start_angle, arc_angle,␣
 ↪segments)
steps.append(fc.PlotAnnotation(point=steps[-1], label="end"))
steps.append(fc.PlotAnnotation(point=steps[0], label="start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
```

```
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```

spiral

[ ]:
```
centre_point = fc.Point(x=10, y=10, z=0)
start_radius = 10
end_radius = 8
start_angle = 0
n_turns = 5
segments = 320
z_change = 0
clockwise = True
steps = fc.spiralXY(centre_point, start_radius, end_radius, start_angle,␣
 ↪n_turns, segments, clockwise)
steps.append(fc.PlotAnnotation(point=steps[-1], label="end"))
steps.append(fc.PlotAnnotation(point=steps[0], label="start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))

# spirals are also possible by using fc.variable_arcXY with 'arc_angle' set to␣
 ↪the number of passes * tau and 'radius_change' set to the total change in␣
 ↪radius over the whole spiral
```

helix

[ ]:
```
centre_point = fc.Point(x=10, y=10, z=0)
start_radius = 10
end_radius = 10
start_angle = 0
n_turns = 5
pitch_z = 0.4
segments = 320
clockwise = True
steps = fc.helixZ(centre_point, start_radius, end_radius, start_angle, n_turns,␣
 ↪pitch_z, segments, clockwise)
steps.append(fc.PlotAnnotation(point=steps[-1], label="end"))
steps.append(fc.PlotAnnotation(point=steps[0], label="start"))
steps.append(fc.PlotAnnotation(point=centre_point, label="centre"))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))

# helices are also possible by using fc.variable_arcXY with 'arc_angle' set to␣
 ↪the number of passes * tau and 'z_change' set to the total helix length
```

waves

[ ]:
```
# wave 1
start_point = fc.Point(x=10, y=10, z=0)
direction = fc.Vector(x=1,y=0)
amplitude = 5
line_spacing = 1
periods = 10
extra_half_period = False
extra_end_line = False
steps = fc.squarewaveXY(start_point, direction, amplitude, line_spacing,␣
 ↪periods, extra_half_period, extra_end_line)
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 1"))

# wave 2
start_point = fc.Point(x=10, y=20, z=0)
extra_half_period = True
# steps.extend([fc.Extruder(on=False), start_point, fc.Extruder(on=True)])
steps.extend(fc.travel_to(start_point))
steps.extend(fc.squarewaveXY(start_point, direction, amplitude, line_spacing,␣
 ↪periods, extra_half_period, extra_end_line))
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 2"))
steps.append(fc.PlotAnnotation(label="extra half period"))

# wave 3
start_point = fc.Point(x=10, y=30, z=0)
extra_half_period = True
extra_end_line = True
# steps.extend([fc.Extruder(on=False), start_point, fc.Extruder(on=True)])
steps.extend(fc.travel_to(start_point))
steps.extend(fc.squarewaveXY(start_point, direction, amplitude, line_spacing,␣
 ↪periods, extra_half_period, extra_end_line))
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 3"))
steps.append(fc.PlotAnnotation(label="extra end-line"))

# wave 4
start_point = fc.Point(x=10, y=40, z=0)
direction_polar = tau/8
# steps.extend([fc.Extruder(on=False), start_point, fc.Extruder(on=True)])
steps.extend(fc.travel_to(start_point))
steps.extend(fc.squarewaveXYpolar(start_point, direction_polar, amplitude,␣
 ↪line_spacing, periods, extra_half_period, extra_end_line))
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 4␣
 ↪(squarewaveXYpolar)"))
steps.append(fc.PlotAnnotation(label="wave in polar direction tau/8"))

# wave 5
start_point = fc.Point(x=40, y=45, z=0)
direction_polar = 0.75*tau
```

```
tip_separation = 2
extra_half_period = False
# steps.extend([fc.Extruder(on=False), start_point, fc.Extruder(on=True)])
steps.extend(fc.travel_to(start_point))
steps.extend(fc.trianglewaveXYpolar(start_point, direction_polar, amplitude,␣
 ↪tip_separation, periods, extra_half_period))
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 5␣
 ↪(trianglewaveXYpolar)"))

# wave 5
start_point = fc.Point(x=50, y=35, z=0)
direction_polar = 0.75*tau
period_lenth = 2
segments_per_period = 16
extra_half_period = False
phase_shift = 0
# steps.extend([fc.Extruder(on=False), start_point, fc.Extruder(on=True)])
steps.extend(fc.travel_to(start_point))
steps.extend(fc.sinewaveXYpolar(start_point, direction_polar, amplitude,␣
 ↪period_lenth, periods, segments_per_period, extra_half_period, phase_shift))
steps.append(fc.PlotAnnotation(point=start_point, label="start of wave 6␣
 ↪(sinewaveXYpolar)"))

fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```

segmented line typically, a straight line is created with only start and end points. this means the shape of the line cannot be edited after creation. it is sometimes advantageous to be able to change the shape retrospectively. if the line is defined as a series of segments, all the points along the line can be edited after creation. the example below shows a straight line being modified based on each point

the function segmented_line() allows a segmented line to be created easily based on a start point and end point

[ ]:
```
start_point = fc.Point(x=0, y=0, z=0)
end_point = fc.Point(x=100, y=0, z=0.1)
steps = fc.segmented_line(start_point, end_point, segments=15)
for step in steps:
    step.y = step.y + 20*sin(tau*step.x/200)
for i in range(len(steps)):
    steps.insert(i*2+1,fc.PlotAnnotation(label='')) # add blank PlotAnnotations␣
 ↪at all points to highlight them in the plot
steps.append(fc.PlotAnnotation(point=fc.Point(x=50, y=40,z=0), label='points␣
 ↪along a segmented line can be modified after creation to form a curve'))
fc.transform(steps, 'plot', fc.PlotControls(style='line'))
```

1.2 II. measurements from points

functions: - distance - point_to_polar - angleXY_between_3_points

```
[ ]: pt1, pt2, pt3 = fc.Point(x=0, y=0, z=0), fc.Point(x=0, y=10, z=0), fc.
      ↪Point(x=10, y=0, z=0)

     distance = fc.distance(pt1, pt2)
     print('distance between pt1 and pt2: ' + str(distance))

     polar_data = fc.point_to_polar(pt2, fc.Point(x=0, y=0, z=0))
     print("\n'polar radius' of pt2 relative to x=0,y=0,z=0: " + str(polar_data.
      ↪radius))
     print("'polar angle' of pt2 relative to x=0,y=0,z=0: " + str(polar_data.angle)␣
      ↪+ ' (radians: 0 to tau)')
     print("'polar angle' of pt2 relative to x=0,y=0,z=0: " + str((polar_data.angle/
      ↪tau)*360) + ' (degrees: 0 to 360)')
     # see the creation of a point from polar coordinates elsewhere in this notebook␣
      ↪- fc.polar_to_point()

     angle = fc.angleXY_between_3_points(pt1, pt2, pt3)
     print('\nangle between pt1-pt2-pt3: ' + str(angle) + ' (radians: -tau to tau)')
     print('angle between pt1-pt2-pt3: ' + str((angle/tau)*360) + ' (degrees: -360␣
      ↪to 360)')
```

1.3 III. move and copy points

the move() function in FullControl allows moving and coping of a point, list of points, or a combined list of points and other ***state***-changing object

the amount of movement is defined by FullControl's Vector object

move

```
[ ]: vector = fc.Vector(x=0, y=0, z=0.5)

     start_point = fc.Point(x=0,y=0,z=0)
     print('start_point: ' + str(start_point))
     moved_start_point = fc.move(start_point, vector)
     print('moved_start_point: ' + str(moved_start_point))

     steps = fc.rectangleXY(start_point, 50, 20)
     print('\noriginal points for a rectangle: ' + str(steps))
     steps = fc.move(steps, vector)
     print('moved rectangle: ' + str(steps))

     steps=[start_point,fc.Fan(speed_percent=90),moved_start_point]
     print('\nlist with non-point object: ' + str(steps))
     print('moved list with non-point object ' + str(fc.move(steps,vector)))
```

copy

```
[ ]: vector = fc.Vector(x=0, y=0, z=0.5)
     start_point = fc.Point(x=0,y=0,z=0)
     steps = fc.rectangleXY(start_point, 50, 20)
     steps = fc.move(steps, vector,copy=True, copy_quantity=5)
     fc.transform(steps, 'plot', fc.PlotControls(style='line'))
```

move/copy (polar coordinates)

```
[ ]: array_centre = fc.Point(x=50,y=50,z=0)
     first_helix_centre = fc.Point(x=20, y=50, z=0)
     steps = fc.helixZ(first_helix_centre,10,10,0,5,0.5,100)
     steps = fc.move_polar(steps, array_centre, 0, tau/6, copy=True, copy_quantity=6)
     fc.transform(steps, 'plot', fc.PlotControls(style='line'))
```

reflect points can be reflected about a line that is defined by either two points (fc.reflectXY), or by one point and a polar angle (fc.reflectXYpolar) - polar angle convention: - 0 = positive x direction - 0.25*tau = positive y direction - 0.5*tau = negative x direction - 0.75*tau = negative y direction.

```
[ ]: steps = []

     pt1 = fc.Point(x=50, y=50, z=0)
     print('point before reflecting: \n' + str(pt1))
     pt1_reflected = fc.reflectXY(pt1, fc.Point(x=0, y=0), fc.Point(x=1, y=0))
     print("point after reflecting about x-axis using 'reflectXY()': \n" +␣
      ↪str(pt1_reflected))
     pt1_reflected = fc.reflectXYpolar(pt1, fc.Point(x=0, y=0), tau/4)
     print("point after reflecting about y-axis using 'reflect_polar()': \n" +␣
      ↪str(pt1_reflected))
```

reflecting a list FullControl's reflect functions can only be used on individual points. reflecting lists of points is not simple because a reflected list of points must typically be printed in reverse order. otherwise, the nozzle would jump from the last point to the first point of the list before printing its reflection. if the list contained instructions halfway through to change ***state*** beyond a point (e.g. turn extrusion on/off), these instructions would affect different sections of the print path for the reflected and non-reflected lists since their sequences are reversed. therefore, FullControl allows the designer to reflect points only - it is up to the designer to iterate through a list of points, as demonstrated below. if ***state***-changing objects are included in the list, it is up to the designer to decide the appropriate location for them in the reflected list and to not attempt a reflectXY() function on them since they will not have xyz attributes

```
[ ]: steps = []
     steps.extend(fc.arcXY(fc.Point(x=50, y=50, z=0), 10, (5/8)*tau, tau/8, 16))
     steps.extend(fc.arcXY(fc.Point(x=80, y=55, z=0), 15, 0.75*tau, 0.5*tau, 16))
     steps.extend(fc.arcXY(fc.Point(x=80, y=65, z=0), 5, 0.25*tau, 0.5*tau, 16))
     steps.extend(fc.arcXY(fc.Point(x=80, y=55, z=0), 5, 0.25*tau, -0.5*tau, 16))
     steps.extend(fc.arcXY(fc.Point(x=60, y=60, z=0), 10, 0.75*tau, -tau/8, 16))
```

```
steps_and_annotation = steps + [fc.PlotAnnotation(label='this geometry is␣
 ↪reflected in next plot')]
fc.transform(steps_and_annotation, 'plot', fc.
 ↪PlotControls(color_type='print_sequence', style='line'))

steps_reflected = []
step_count = len(steps)
for i in range(step_count):
    # reflect about a line connecting the first point (steps[0]) and last point␣
 ↪(steps[-1])
    steps_reflected.append(fc.reflectXY(steps[(step_count-1)-i], steps[0],␣
 ↪steps[-1]))
steps.extend(steps_reflected)
steps.extend([fc.PlotAnnotation(point = pt, label='') for pt in fc.
 ↪segmented_line(fc.Point(x=43, y=43, z=0), fc.Point(x=52.5, y=52.5, z=0),␣
 ↪20)])
# add some points to the plot to indicate the reflection line
steps.append(fc.PlotAnnotation(point=steps[step_count], label='all points from␣
 ↪the previous plot were reflected and added to the path (in reverse order)'))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style='line'))
```