PlantBox: RootSystem Tutorial

Daniel Leitner
www.simwerk.at

The following tutorial offers scripts to outline the usage of the CPlantBox Python binding *plantbox* for many different applications. CPlantBox was developed from CRootBox and is largely backward compatible by having the same underlying rootsystem model. For further documentation please refer to the Doxygen class documentation of the CPlantBox code.

Contents

1 Basic usage

The first example shows how to use CRootBox in the most simple situation: open a parameter file (L7), do the simulation (L13), and save the result (L16).

```
1  """small example"""
2  import sys
3  sys.path.append("../../..")
4  import plantbox as pb
5
6  rootsystem = pb.RootSystem()
7
8  # Open plant and root parameter from a file
9  path = "../../../modelparameter/rootsystem/"
10 name = "Anagallis_femina_Leitner_2010"
11 rootsystem.readParameters(path + name + ".xml")
12
13 # Initialize
14 rootsystem.initialize()
15
16 # Simulate
17 rootsystem.simulate(30, True)
18
19 # Export final result (as vtp)
20 rootsystem.write("results/example_1a.vtp")
```

Listing 1: Example 1a

Lets revise the above code in more detail:

1 Imports the CRootBox Python library (py_rootbox), and names it rb.

6 Constructs the root system object.

11 Opens an .xml containing parameters desribing the types of root (RootRandomParameters), and the type of pant (SeedRandomParameters). Alternatively, all parameter can be set or modified directly in Python (see Section 4.2).

14 Initializes the simulation: Creates the tap root the base roots (i.e. all basal roots, and shoot borne roots that might emerge), creates the tropisms and passes the domain geometry to it, and creates the elongation functions.

17 Performs the simulation. The value 30 is the simulation time in days. If no simulation time is passed the simulation time is taken from the .pparam file. Note that simulation results are independent from the time step, i.e. 30 simulate(1) calls should yield the same result as simulate(30).

20 Saves the resulting root system geometry in the VTK Polygonal Data format (VTP) as polylines, see Figure 1a.

The next example is an extension of the previous one, where the root system grows in one of two containers (a soil core or rectangular rhizotron). Such geometries are important if we want to mimic experimental settings. In CPlantBox the domain geometry is represented in a mesh free way using signed distance functions (SDF). A SDF returns the distance to the closest boundary, with negative sign if it lies inside of the domain, and a positive if it the point is outside.

```python
"""small example in a container"""
import sys
sys.path.append("../../..")
import plantbox as pb

rootsystem = pb.RootSystem()

# Open plant and root parameter from a file
path = "../../../modelparameter/rootsystem/"
name = "Anagallis_femina_Leitner_2010"
rootsystem.readParameters(path + name + ".xml")

# Create and set geometry

# 1.creates a cylindrical soil core with top radius 5 cm, bot
    radius 5 cm, height 50 cm, not square but circular
soilcore = pb.SDF_PlantContainer(5, 5, 40, False)

# 2. creates a square 27*27 cm containter with height 1.4 cm
rhizotron = pb.SDF_PlantBox(1.4, 27, 27)

# Pick 1, or 2
rootsystem.setGeometry(soilcore)  # soilcore, or rhizotron

# Initialize
rootsystem.initialize()

# Simulate
rootsystem.simulate(60)  # days

# Export final result (as vtp)
rootsystem.write("results/example_1b.vtp")

# Export container geometry as Paraview Python script
rootsystem.write("results/example_1b.py")
```

Listing 2: Example 1b

The geometry is first created by constructing some specialization of the class SignedDistanceFunction, and is passed to the root system by the method setGeometry:

16 Construct a soil core.

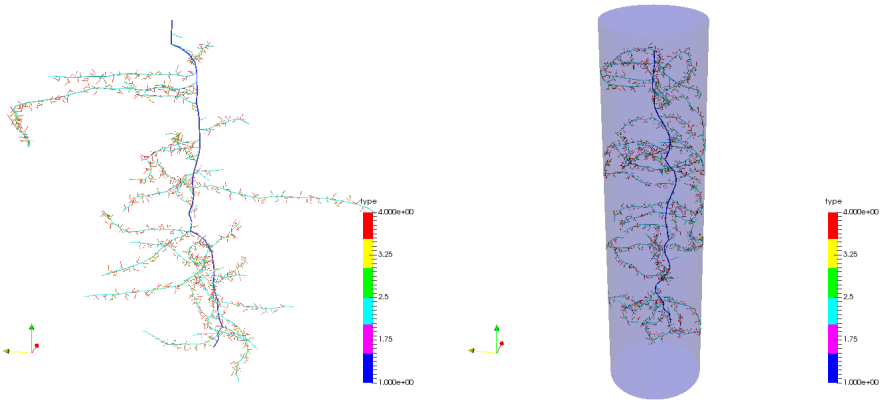
19 Construct a rhizotron.

22 Pick one of the two geometries. Note that it is important to call setGeometry before initialize.

34 Its possible to save the geometry as Paraview Python script for visualization (and debugging), see Figure 1b. Run this script in Paraview by Tools→Python Shell, Run Script.

2 More complex geometries

The section shows how to build more complex geometries with SDF. Furthermore, we show an example with multiple root systems that is computed in parallel.



(a) Unconfined growth (Example 1a) (b) Confined growth (Example 1b)

Figure 1: Resulting figures from Section 1

2.1 Using SDF with set operations

In the following example we create more complex geometries that we might encounter in experiments. First, we show how to rotate a rhizotron (e.g. to see more roots at the wall due to gravitropism). Second, we create a split box experiment, and furthermore, an example where rhizotubes act as obstacles.

The following examples show how to build a complex geometry using rotations, translations and set operations on the SDF.

```
"""more complex geometries"""
import sys
sys.path.append("../../..")
import plantbox as pb
import math

rs = pb.RootSystem()

# Open plant and root parameter from a file
path = "../../../modelparameter/rootsystem/"
name = "Zea_mays_4_Leitner_2014"
rs.readParameters(path + name + ".xml")

# 1. Creates a square rhizotron r*r, with height h, rotated around the x-axis
r, h, alpha = 20, 4, 45
rhizotron2 = pb.SDF_PlantContainer(r, r, h, True)
posA = pb.Vector3d(0, r, -h / 2)  # origin before rotation
A = pb.Matrix3d.rotX(alpha / 180.*math.pi)
posA = A.times(posA)  # origin after rotation
rotatedRhizotron = pb.SDF_RotateTranslate(rhizotron2, alpha, 0, posA.times(-1))

# 2. A split pot experiment
topBox = pb.SDF_PlantBox(22, 20, 5)
sideBox = pb.SDF_PlantBox(10, 20, 35)
left = pb.SDF_RotateTranslate(sideBox, pb.Vector3d(-6, 0, -5))
right = pb.SDF_RotateTranslate(sideBox, pb.Vector3d(6, 0, -5))
box_ = []
```

```
28 box_.append(topBox)
29 box_.append(left)
30 box_.append(right)
31 splitBox = pb.SDF_Union(box_)
32
33 # 3. Rhizotubes as obstacles
34 box = pb.SDF_PlantBox(96, 126, 130)  # box
35 rhizotube = pb.SDF_PlantContainer(6.4, 6.4, 96, False)  # a single
       rhizotube
36 rhizoX = pb.SDF_RotateTranslate(rhizotube, 90, pb.SDF_Axis.yaxis,
       pb.Vector3d(96 / 2, 0, 0))
37
38 rhizotubes_ = []
39 y_ = (-30, -18, -6, 6, 18, 30)
40 z_ = (-10, -20, -40, -60, -80, -120)
41 tube = []
42 for i in range(0, len(y_)):
43     v = pb.Vector3d(0, y_[i], z_[i])
44     tube.append(pb.SDF_RotateTranslate(rhizoX, v))
45     rhizotubes_.append(tube[i])
46
47 rhizotubes = pb.SDF_Union(rhizotubes_)
48 rhizoTube = pb.SDF_Difference(box, rhizotubes)
49
50 # Set geometry: rotatedRhizotron, splitBox, or rhizoTube
51 rs.setGeometry(rhizoTube)
52
53 # Simulate
54 rs.initialize()
55 rs.simulate(90)  # days
56
57 # Export results (as vtp)
58 rs.write("results/example_2a.vtp")
59
60 # Export container geometry as Paraview Python script
61 rs.write("results/example_2a.py")
```

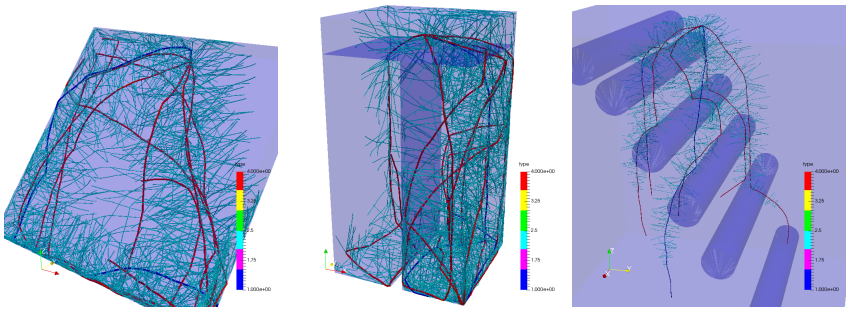
Listing 3: Example 2a

14-20 Definition of a rotated rhizotron, see Figure 2a: L16 creates the flat container with a small height, this container is then rotated and translated into the desired position. L17 is the position where the origin will lie, and L18 the rotational matrix around the x-axis. In L19 the origin position is rotated. Finally, in L20 the new rotated and translated geometry is created.

22-31 Definition of of a split box, see Figure 2b: The split box is composed of a left box, a right box, and a top box connecting left and right. In L31 the geometry is defined by the set operation union of the three compartments.

33-48 Definition of rhizotubes as obstacles, see Figure 2c: L34 is the surrounding box, L35 a single rhizotube, that is rotated around the y-axis in L36. L38-L45 create a list of rhizotubes at different locations that mimics the experimental setup. L48 and L48 compose the final geometry by to set operation, first a union of all tubes, and then cut them out the surrounding box by taking the difference.

51 Pick one of the three geometries for your simulation.



(a) Rotated rhizotron (b) Split box (c) Rhizotubes

Figure 2: Different geometries described by SDF (Example 2a)

61 Also more complex geometries can be visualized by the Paraview script, however set operations are not really performed, only the involved geometries are visualized.

2.2 Multiple root systems

Its possible to simulate multiple root systems. In the following we show a small plot scale simulation.

```
"""multiple root systems"""
import sys
sys.path.append("../../..")
import plantbox as pb

path = "../../../modelparameter/rootsystem/"
name = "Zea_mays_4_Leitner_2014"

simtime = 120
N = 3  # number of columns and rows
dist = 40  # distance between the root systems [cm]

# Initializes N*N root systems
allRS = []
for i in range(0, N):
    for j in range(0, N):
        rs = pb.RootSystem()
        rs.readParameters(path + name + ".xml")
        rs.getRootSystemParameter().seedPos = pb.Vector3d(dist * i, dist * j, -3.)  # cm
        rs.initialize(False)  # verbose = False
        allRS.append(rs)

# Simulate
for rs in allRS:
    rs.simulate(simtime, False)  # verbose = False

# Export results as single vtp files (as polylines)
ana = pb.SegmentAnalyser()  # see example 3b
for i, rs in enumerate(allRS):
      vtpname = "results/example_2b_" + str(i) + ".vtp"
      rs.write(vtpname)
      ana.addSegments(rs)  # collect all
```

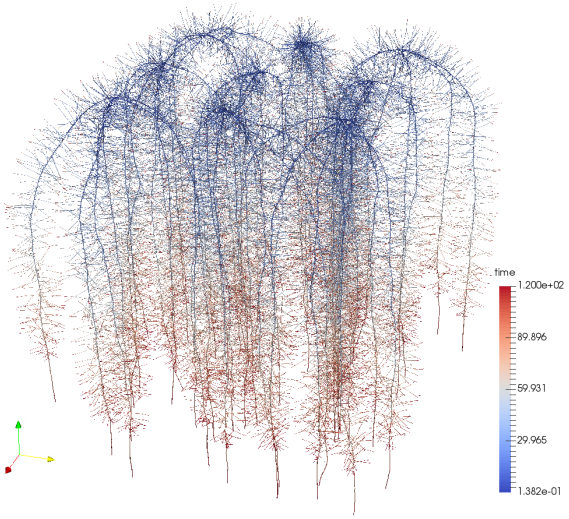


Figure 3: Multiple root systems (Example 2b), colors denote the creation time of the root segments

```
33
34 # Write all into single file (segments)
35 ana.write("results/example_2b_all.vtp")
```

Listing 4: Example 2b

10,11 Set the number of columns and rows of the plot, and the distance between the root systems.

14-21 Creates the root systems, and puts them into a list allRS. L19 sets the position of the seed.

24,25 Simulate all root systems

28-35 Saves each root systems, and additionally, saves all root systems into a single file. Therefore, we create an SegmentAnalyser object in L28 and merge all segments into it L32, and finally export the single file L35. The resulting geometry is shown in Figure 3.

Each root system has its own random number generator. By default the seed of the generator is initialized with the system clock. If this is not sufficient, e.g. if multiple root systems are initiated at the same time on multi-core systems, or the simulation shall be reproducable, the seed can be set manually using the method RootSystem::setSeed(int).

3 Analysis of simulation results

There are various post processing options, on a per root level in the class RootSystem, or a per segment level in the class SegmentAnalyser. We show some examples of the most frequently used methods.

3.1 Analysis per root

The class RootSystem offers several post-processing options on a per root level. The following script shows how to analyse length versus time. Additionally it demonstrates how to obtain root tip or root base positions.

First, lets analyse the root length versus time, and consider the total root length, and root length per type.

```
"""root system length over time"""
import sys
sys.path.append("../../..")
import plantbox as pb

import numpy as np
import matplotlib.pyplot as plt

path = "../../../modelparameter/rootsystem/"
name = "Brassica_napus_a_Leitner_2010"

rs = pb.RootSystem()
rs.readParameters(path + name + ".xml")
rs.initialize()

simtime = 60.  # days
dt = 1.
N = round(simtime / dt)  # steps

# Plot some scalar value over time
stype = "length"
v_, v1_, v2_, v3_ = np.zeros(N), np.zeros(N), np.zeros(N), np.zeros
    (N)
for i in range(0, N):
    rs.simulate(dt)
    t = np.array(rs.getParameter("type"))
    v = np.array(rs.getParameter(stype))
    v_[i] = np.sum(v)
    v1_[i] = np.sum(v[t == 1])
    v2_[i] = np.sum(v[t == 2])
    v3_[i] = np.sum(v[t == 3])

t_ = np.linspace(dt, N * dt, N)
plt.plot(t_, v_, t_, v1_, t_, v2_, t_, v3_)
plt.xlabel("time (days)")
plt.ylabel(stype + " (cm)")
plt.legend(["total", "tap root", "lateral", "2. order lateral"])
plt.savefig("results/example_3a.png")
plt.show()
```

Listing 5: Example 3a

6 NumPy is Python's scientific computing package.

7 Matplotlib is Python's easy way to create figures like in Matlab.

9-14 Sets up the simulation.

16-18 Defines the simulation time, time step, and the resulting number of simulate(dt) calls.

21 First we state which scalar type we want to analyse (others are type, radius, order, time, surface, one, parenttype)

22 Pre-definition of the NumPy arrays storing the lengths over time.

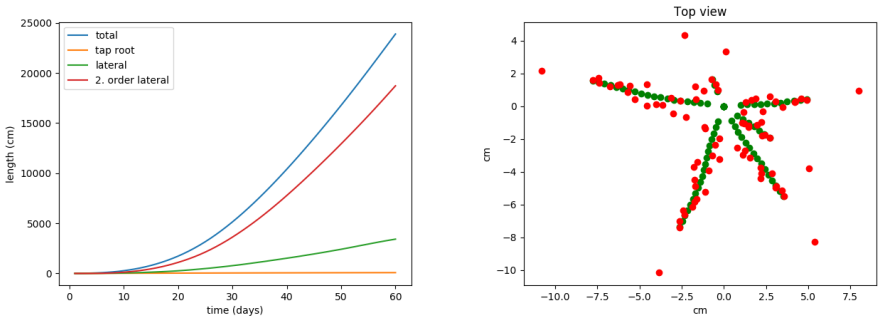
23-30 The simulation loops executes the simulation for a single time step L24. L25 calculates the type of each root, L26 the length (or other scalar type) of the root. L27-L30 calculates the total root length in the time step for all roots, and for specific root types.

32-38 Creates Figure 4a.

Next we show two options how to retrieve root tip postions and root base positions from a simulation:

```
"""find root tips and bases (two approaches)"""
import sys
sys.path.append("../../..")
import plantbox as pb

import numpy as np
import matplotlib.pyplot as plt

path = "../../../modelparameter/rootsystem/"
name = "Anagallis_femina_Leitner_2010"  # "Brassica_napus_a_Leitner_2010"

rs = pb.RootSystem()
rs.readParameters(path + name + ".xml")
rs.initialize()
rs.simulate(7)  # 7 days young....

print(rs.getNumberOfNodes(), "nodes")
print(rs.getNumberOfSegments(), "segments")

# Use polyline representation of the roots
polylines = rs.getPolylines()
bases = np.zeros((len(polylines), 3))
tips = np.zeros((len(polylines), 3))
for i, r in enumerate(polylines):
    bases[i, :] = [r[0].x, r[0].y, r[0].z]
    tips[i, :] = [r[-1].x, r[-1].y, r[-1].z]

# Or, use node indices to find tip or base nodes
nodes = np.array((list(map(np.array, rs.getNodes()))))
tipI = rs.getRootTips()
baseI = rs.getRootBases()

# Plot results (1st approach)
plt.title("Top view")
plt.xlabel("cm")
plt.ylabel("cm")
plt.scatter(nodes[baseI, 0], nodes[baseI, 1], c = "g", label = "root bases")
plt.scatter(nodes[tipI, 0], nodes[tipI, 1], c = "r", label = "root tips")
plt.legend()
plt.savefig("results/example_3b.png")
plt.show()

 # check if the two approaches yield the same result
uneq = np.sum(nodes[baseI, :] != bases) + np.sum(nodes[tipI, :] != tips)
print("Unequal tips and basals:", uneq)
```

Listing 6: Example 3b



(a) Total root length versus time (b) Top view of the root tip and root bases

Figure 4: Root system analysis: Example 3a (a), Example 3b (b)

14,15 Reset the simulation and simulate for only 7 days (otherwise there are so many root tips).

17-18 Outputs the number of nodes and segments to get an idea how big the resulting root system is. Note that number of segments equals the number of nodes minus the number of base roots that will emerge. Base roots are tap roots, basal roots and shootborne roots.

20-36 The first approach retrieves all roots as polylines L50. Root tips are the last nodes of the polylines L54, root bases the first nodes L53. Roots that have not started to grow have only 1 node, and are not retrieved by getPolylines().

28-31 Second approach: L29 rs.getNodes() returns all nodes of the root system as a list of Vector3d objects. Each Vector3d object can be converted into a numpy array automatically, but is necessary to do that for each element of the list. The methods L30, L31 return the indices of the tips and bases.

33-41 Creates Figure 4b using the second approach.

44,45 Verifies that both approaches yield the same result.

3.2 Analysis per segment

The class SegmentAnalyser offers post-processing methods per root segment. The advantage is that we can do distributions or densities, and that we can crop the segments with any geometry.

3.3 Analysis per segment, rootsystem length density

We start with a small example plotting the root surface densities of a root system versus root depth.

```
"""root system surface density"""
import sys
sys.path.append("../../..")
import plantbox as pb
```

```
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  path = "../../../modelparameter/rootsystem/"
9  name = "Crypsis_aculeata_Clausnitzer_1994"
10
11 rs = pb.RootSystem()
12 rs.readParameters(path + name + ".xml")
13
14 depth = 130
15 layers = 50
16 runs = 10
17
18 rl_ = []
19 for i in range(0, runs):
20     rs.initialize(False)
21     rs.simulate(120, False)
22     ana = pb.SegmentAnalyser(rs)
23     rl_.append(ana.distribution("length", 0., -depth, layers, True)
       )
24
25 soilvolume = (depth / layers) * 10 * 10
26 rl_ = np.array(rl_) / soilvolume  # convert to density
27 rl_mean = np.mean(rl_, axis = 0)
28 rl_err = np.std(rl_, axis = 0) / np.sqrt(runs)
29
30 z_ = np.linspace(0, -depth, layers)  # z - axis
31 plt.plot(rl_mean, z_, "b")
32 plt.plot(rl_mean + rl_err, z_, "b:")
33 plt.plot(rl_mean - rl_err, z_, "b:")
34
35 plt.xlabel("root surface (cm^2 / cm^3)")
36 plt.ylabel("z-coordinate (cm)")
37 plt.legend(["mean value (" + str(runs) + " runs)", "error"])
38 plt.savefig("results/example_3c.png")
39 plt.show()
```

Listing 7: Example 3c

8-12 Pick a root system.

14-16 Depth describes the y-axis of the graph, layers the number of vertical soil layers, where the root surface is accumulated, and runs is the number of simulation runs.

18-23 Perfoms the simulations. L23 creates a distribution of a parameter (name) over a vertical range (bot, top). The data are accumulated layers, segments are either cut (exact = True) or accumaled by their mid point (exact = False).

25 In order to calculate a root surface density from the summed up surface, we need to define a soil volume. The vertical height is the layer length, lenght and widht (here 10 cm), can be determined by planting widht, or by geometry, if root growth is confined.

26-28 Calculates the densitis, mean densities, and standard error.

30-39 Prepares teh plot (see Figure 5).

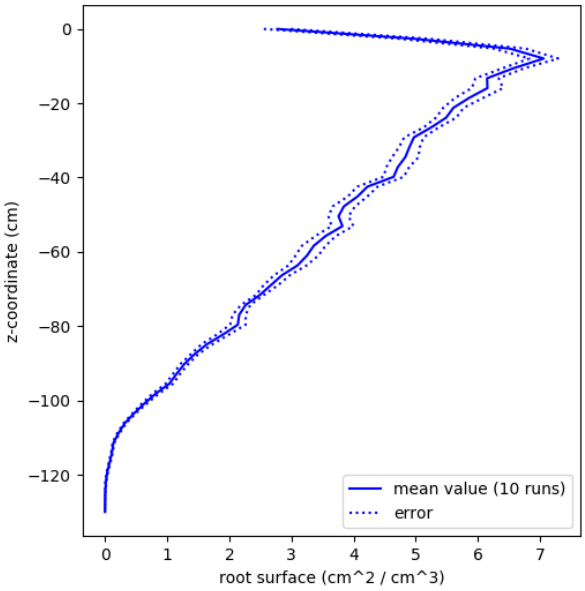


Figure 5: Root surface densitiy versus depth mean (Example 3c), and standard error (N=10)

3.4 Analysis per segment using SDF

The following script demonstrates some of the post processing possibilities by setting up a virtual soil core experiment, where we analyse the content of two soil cores located at other positions.

```
"""analysis of results using signed distance functions"""
import sys
sys.path.append("../../..")
import plantbox as pb

import numpy as np
import matplotlib.pyplot as plt

path = "../../../modelparameter/rootsystem/"
name = "Zea_mays_1_Leitner_2010"

rs = pb.RootSystem()
rs.readParameters(path + name + ".xml")
rs.initialize()
rs.simulate(120)

# Soil core analysis
r, depth, layers = 10, 100., 100
soilcolumn = pb.SDF_PlantContainer(r, r, depth, False)  # in the
    center of the root
soilcolumn2 = pb.SDF_RotateTranslate(soilcolumn, 0, 0, pb.Vector3d
    (10, 0, 0))  # shift 10 cm

# pick one geometry for further analysis
```

```
23 geom = soilcolumn2
24
25 z_ = np.linspace(0, -1 * depth, layers)
26 fig, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (16, 8))
27 for a in axes:
28     a.set_xlabel('RLD (cm/cm)')  # layer size is 1 cm
29     a.set_ylabel('Depth (cm)')
30
31 # Make a root length distribution
32 ana = pb.SegmentAnalyser(rs)
33 rl_ = ana.distribution("length", 0., -depth, layers, True)
34 axes[0].set_title('All roots (120 days)')
35 axes[0].plot(rl_, z_)
36
37 # Make a root length distribution along the soil core
38 ana = pb.SegmentAnalyser(rs)
39 ana.crop(geom)
40 ana.pack()
41 rl_ = ana.distribution("length", 0., -depth, layers, True)
42 axes[1].set_title('Soil core (120 days)')
43 axes[1].plot(rl_, z_)
44
45 # How it looked after 30 days?
46 ana = pb.SegmentAnalyser(rs)
47 ana.filter("creationTime", 0, 30)
48 ana.crop(geom)
49 ana.pack()
50 rl_ = ana.distribution("length", 0., -depth, layers, True)
51 axes[2].set_title('Soil core (30 days)')
52 axes[2].plot(rl_, z_)
53
54 # Only laterals
55 ana = pb.SegmentAnalyser(rs)
56 ana.filter("subType", 2)  # assuming laterals are of type 2
57 ana.crop(geom)
58 ana.pack()
59 rl_ = ana.distribution("length", 0., -depth, layers, True)
60 axes[3].set_title('Soil core, lateral roots (120 days)')
61 axes[3].plot(rl_, z_)
62
63 fig.subplots_adjust()
64 plt.savefig("results/example_3d.png")
65 plt.show()
```

Listing 8: Example 3d

9-15 Performs the simulation.

17-20 We define two soil cores, one in the center of the root and one 10 cm translated. In L16 we pick which one we use for the further analysis. Figure 6 shows the resulting geometry.

23 We pick which geometry we will use for further analysis.

25-29 Prepares the plot. We use four sub-figures.

31-35 Creates a root length distribution versus depth. L32 creates the SegmentAnalyser object, and L41 creates the distribution.

37-43 Performs the same distribution in the soil core. In L39 we crop the segments to the geometry. L49 is used to delete unused nodes.

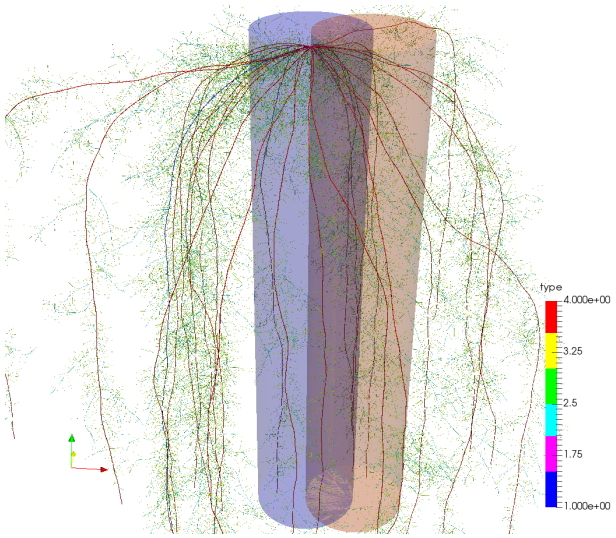


Figure 6: Virtual soil cores experiment (Example 3d): Central core (blue), shifted core (red)

45-52 Same as before, but we are only interested in segments that are younger than 30 days. In L47 we use the filer method (name, min, max) to keep only the segments we want in the analysis.

54-61 The filter method can be used for many different applications. In the following we use it to analyse lateral roots only. Possible rb.ScalarType are: 'type', 'radius', 'order', segment creation time 'time', 'length', 'surface', 'volume', 'one', 'userdata1', 'userdata2', 'userdata3', and 'parenttype'.

63-65 Show and save resulting Figure 7, and 8.

In this example the central core captures only a little amount of laterals (Figure 7) because the root system is wide spread. The shifted root core represents the overall root system slightly better, see Figure 8. The basic idea is that such simulations help to increase the understanding of experimental observations.

3.5 SegmentAnalyser without RootSystem

It is possible to make use of the SegmentAnalyser class without any ohter CPlantBox classes (e.g. to plot densities, or for writing vtp). The following example shows how to construct the class with arbitrary nodes and segments (e.g. from measurements).

```
""" analysis of nodes and segments from measurements """
import sys
sys.path.append("../../..")
import plantbox as pb

nodes = [ [0, 1, 0], [0, 1, -1], [0, 1, -2], [0, 1, -3], ]
segs = [ [0, 1], [1, 2], [2, 3] ]
cts = [0., 0., 0.]
radii = [ 0.1, 0.1, 0.1]
```

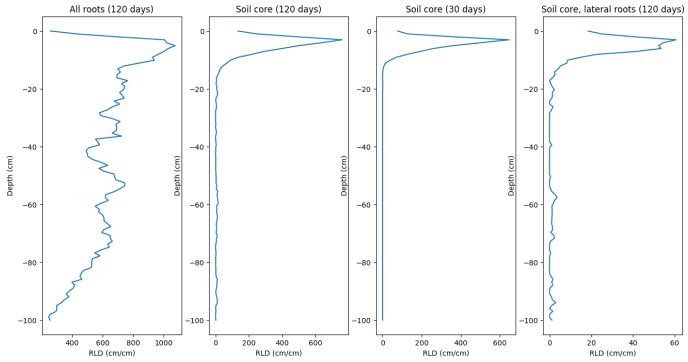


Figure 7: Central core (Example 3d)

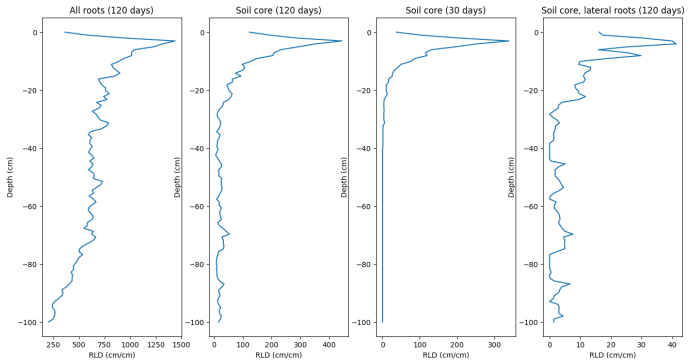


Figure 8: Shifted core (Example 3d)

```
10
11 # convert from python to c++ types
12 nodes = [pb.Vector3d(n[0], n[1], n[2]) for n in nodes]
13 segs = [pb.Vector2i(s[0], s[1]) for s in segs]
14
15 # use C++ without RootSystem
16 ana = pb.SegmentAnalyser(nodes, segs, cts, radii)
17
18 print("length", ana.getSummed("length"))
19
20 ana.write("example_3e.vtp", ["radius"])  # working
21 # ana.write("test.vtp") # not working, but with a meaningful
       exception
```

Listing 9: Example 3e

4 Changing model parameters

In the following we show how model parameters can be modified, and how a sensitivity analysis can be performed. Additionally, we comment on how to best make an animation from a CPlantBox simulation.

4.1 Set up a simulation from scratch

In the previous examples we always opened the root system parameters from a file. In the following example we show to do everything in a Python script without the need of any parameter files. This is especially important if we want to modify any of the parameters in our scripts, like it is needed for sensitivity analysis in the next section.

```
1 """everything from scratch (without parameter files)"""
2 import sys
3 sys.path.append("../../..")
4 import plantbox as pb
5
6 import math
7
8 rs = pb.RootSystem()
9
10 # Root random parameter
11 p0 = pb.RootRandomParameter(rs)  # with default values,
12 p1 = pb.RootRandomParameter(rs)  # all standard deviations are 0
13
14 p0.name = "taproot"
15 p0.subType = 1
16 p0.lb = 1
17 p0.la = 10
18 p0.nob = 20
19 p0.ln = 89. / 19.
20 p0.theta = 30. / 180.*math.pi
21 p0.r = 1
22 p0.dx = 0.5
23 p0.successor = [2]  # add successors
24 p0.successorP = [1]
25 p0.tropismT = pb.TropismType.gravi
26 p0.tropismN = 1.
27 p0.tropismS = 0.2
28
```

```
29 p1.name = "lateral"
30 p1.subType = 2
31 p1.la = 25
32 p1.las = 10  # add standard deviation
33 p1.ln = 0
34 p1.r = 2
35 p1.dx = 0.1
36 p1.tropismS = 0.3
37
38 rs.setOrganRandomParameter(p0)
39 rs.setOrganRandomParameter(p1)
40
41 # Root system parameter (neglecting shoot borne)
42
43 rsp = pb.SeedRandomParameter(rs)
44 rsp.seedPos = pb.Vector3d(0., 0., -3.)
45 rsp.maxB = 100
46 rsp.firstB = 10.
47 rsp.delayB = 3.
48 rs.setRootSystemParameter(rsp)
49
50 rs.initialize()
51 rs.simulate(40, False)
52
53 rs.write("../results/example_4a.vtp")
```

Listing 10: Example 4a

8,9 Create the root type parameters of type 1 and type 2.

11-33 We set up a simple root system by hand. First we define the tap root L11-L24, then the laterals L26-L33. In L20 and L21 we have to convert the Python types to the exposed C++ types.

35,36 Set the root type parameters.

38-43 Create the root system parameter stating when basal and shoot borne roots emerge.

45 Sets the root system parameters.

47-50 Initialize, simulate, export.

Note that all parameters can be set and modified within the script. Especially, standard deviations can be set to zero in order to be able to precisely predict the result. For example we can calculate the total root system length analytically, and check if the numerical simulation yield the (exact) same result. This is performed in the tests withing test_root.py, and test_rootsystem, which is used to test and validate CPlantBox and formerly CRootBox.

4.2 Sensitivity analysis

In the next part we vary given parameters in order to make a sensitivity analysis. This takes a lot of simulation runs and we demonstrate the of use parallel computing to speed up execution. We vary the insertion angle of the tap root and basal root, and look at the change in mean root tip depth and radial distance.

```
"""sensitivity analysis: insertion anlge on root tip distribution
    """
import sys
sys.path.append("../../..")
import plantbox as pb

import math
from multiprocessing import Pool
import numpy as np
import matplotlib.pyplot as plt


# sets all standard deviation to value*s
def set_all_sd(rs, s):
    for p in rs.getRootRandomParameter():
        p.lbs = p.lb * s
        p.las = p.la * s
        p.lns = p.ln * s
        p.nobs = p.nob * s
        p.rs = p.r * s
        p.a_s = p.a * s


# Parameters
path = "../../../modelparameter/rootsystem/"
name = "Triticum_aestivum_a_Bingham_2011"
simtime = 20
N = 50  # resolution of paramter
runs = 10  # iterations
theta0_ = np.linspace(0, math.pi / 2, N)


# One simulation
def simulate(i):
    rs = pb.RootSystem()
    rs.readParameters(path + name + ".xml")
    set_all_sd(rs, 0.)  # set all sd to zero
    rs.initialize()  # copy to tap to basal root parameters

    # vary parameter
    p1 = rs.getRootRandomParameter(1)  # tap root
    p4 = rs.getRootRandomParameter(4)  # basal roots
    p1.theta = theta0_[i]
    p4.theta = theta0_[i]

    # simulation
    rs.initialize()  # build again with theta0
    rs.simulate(simtime, True)

    # target
    roots = rs.getPolylines()
    depth = 0.
    rad_dist = 0.
    for r in roots:
        depth += r[-1].z
        rad_dist += math.hypot(r[-1].x, r[-1].y)
    depth /= len(roots)
    rad_dist /= len(roots)

    return depth, rad_dist


```

```
62 depth_ = np.zeros(N)
63 rad_dist_ = np.zeros(N)
64
65 for r in range(0, runs):
66
67     # Parallel execution
68     param = []  # param is a list of tuples
69     for i in range(0, N):
70         param.append((i,))
71     pool = Pool()
72     output = pool.starmap(simulate, param)
73     pool.close()
74
75     # Copy results
76     for i, o in enumerate(output):
77         depth_[i] += (o[0] / runs)
78         rad_dist_[i] += (o[1] / runs)
79
80 # Figure
81 fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 8))
82 axes[0].set_xlabel('Insertion angle theta (-)')
83 axes[1].set_xlabel('Insertion angle theta (-)')
84 axes[0].set_ylabel('Mean tip depth (cm)')
85 axes[1].set_ylabel('Mean tip radial distance (cm)')
86 axes[0].plot(theta0_, depth_)
87 axes[1].plot(theta0_, rad_dist_)
88 fig.subplots_adjust()
89 plt.savefig("results/example_4b.png")
90 plt.show()
```

Listing 11: Example 4b

8-16 Defines a function to set all standard deviations proportional to the parameter values. We use this function in the following to set the standard deviation to zero everywhere.

19-23 Parameters of the analysis. N denotes the resolution of the parameter we vary, and $runs$ the number of iterations, i.e. $N \cdot runs$ simulations are performed. In L24 we define the insertion angle to be varied linearly between 0 and $\pi/2$.

26-51 Definition of a function that performs the simulation and returns mean root tip depth and radial distance. First we create a root system and set the standard deviation to zero L27-L29. L32-L35 sets the insertion angle, tap root is always of type 1, and in the parameter file basal roots are of type 4. L38-39 performs the simulation. L42-L49 calculates the mean root tip depth and radial distance.

53-69 This section performs the computation. L53-54 preallocates the resulting arrays. L58-L64 performs the parallel computations, index i is the index of the insertion angle. L67-L69 calculates the mean values.

72-82 Creates the resulting Figure 9

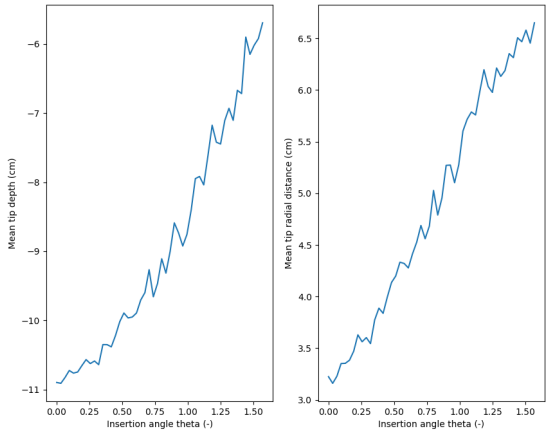


Figure 9: Sensitivity of mean root tip depth (left) and radial distance (right) to the insertion angle theta (Example 3d)

4.3 How to make an animation

In order to create an animation in Paraview we have to consider some details. The main idea is to export the result file as segments using the class SegmentAnalyser. A specific frame is then obtained by thresholding within Paraview using the segments creation times.

We modify example1b.py to demonstrate how to create an animation.

```
"""increase axial resolution (e.g. for animation)"""
import py_rootbox as rb

rs = rb.RootSystem()

# Open plant and root parameter from a file
name = "Anagallis_femina_Leitner_2010"
rs.readParameters("modelparameter/" + name + ".xml")

# Set Geometry
soilcore = rb.SDF_PlantContainer(5, 5, 40, False)
rs.setGeometry(soilcore)

# Modify axial resolution
for p in rs.getRootTypeParameter():
    p.dx = 0.1  # adjust resolution

# Simulate
rs.initialize()
rs.simulate(60, True)  # days

# Export results as segments
rb.SegmentAnalyser(rs).write("../results/example_3e.vtp")

# Export container geometry as Paraview Python script
rs.write("../results/example_3e.py")

```

```
28 print("done.")
```

Listing 12: Example 4c (modified from Example 1b)

14-16 Its important to use a small resolution in order to obtain a smooth animation. L16 set the axial resolution to 0.1 cm.

23 Instead of saving the root system as polylines, we use the SegmentAnalyser to save the root system as segments.

26 We save the geometry as Python script for the visualization in ParaView.

After running the script we perform the following operations to create an animation:

1. Open the .vtp file in ParaView (File→Open...).
2. Optionally, create a tube plot with the help of the script scripts/tubePlot.py (Tools→Python Shell, press 'Run script').
3. Optionally, visualize the domain boundaries by running the script results/example_3e.py (Tools→Python Shell, press 'Run script').
4. Run the script scripts/animate.py (Tools→Python Shell, press 'Run script'). The script creates the threshold filter and the animation.
5. Use File→Save Animation... to render and save the animation.

5 Tropisms

The change in root growth direction is described by tropisms.

5.1 Hydro- and chemotropism

Root growth direction is influenced by soil conditions such as water content, soil strength, or nutrient concentration. In the following we show how this is achieved in CRootBox.

```
"""hydrotropism in a thin layer"""
import sys
sys.path.append("../../..")
import plantbox as pb

rs = pb.RootSystem()
path = "../../../modelparameter/rootsystem/"
name = "Anagallis_femina_Leitner_2010"
rs.readParameters(path + name + ".xml")

# Manually set tropism to hydrotropism for the first ten root types
sigma = [0.4, 1., 1., 1., 1. ] * 2
for p in rs.getRootRandomParameter():
        p.dx = 0.25  # adjust resolution
        p.tropismT = pb.TropismType.hydro
        p.tropismN = 2  # strength of tropism
        p.tropismS = sigma[p.subType - 1]

# Static soil property in a thin layer
```

```
20 maxS = 0.7  # maximal
21 minS = 0.1  # minimal
22 slope = 5  # linear gradient between min and max (cm)
23 box = pb.SDF_PlantBox(30, 30, 2)  # cm
24 layer = pb.SDF_RotateTranslate(box, pb.Vector3d(0, 0, -16))
25 soil_prop = pb.SoilLookUpSDF(layer, maxS, minS, slope)
26
27 # Set the soil properties before calling initialize
28 rs.setSoil(soil_prop)
29
30 # Initialize
31 rs.initialize()
32
33 # Simulate
34 simtime = 100  # e.g. 30 or 60 days
35 dt = 1
36 N = round(simtime / dt)
37 for _ in range(0, N):
38     # in a dynamic soil setting you would need to update soil_prop
39     rs.simulate(dt)
40
41 # Export results (as vtp)
42 rs.write("../results/example_5a.vtp")
43
44 # Export geometry of static soil
45 rs.setGeometry(layer)  # just for vizualisation
46 rs.write("../results/example_5a.py")
```

Listing 13: Example 5a

3-5 Creates the root system and opens the parameter file

7-14 Change the tropism for the first ten root types: L10 retrieves the parameter, where the CRootBox parameters run from 1 to 11. L11 modifies the axial resolution, and L12-14 sets the three tropism parameters.

16-22 Definition of a static soil property using SDF. We first define the geometry (L20-L21), and then create a static soil (L22) that obtains the maximal value $maxS$ inside the geometry, $minS$ outside the geometry, and linear slope with length *slope*. At the boundary the soil has the value $(maxS + minS)/2$.

25 Sets the soil. Must be called before RootSystem::initialize()

28 Initializes the root system, and among others sets up the hydrotropism.

30-36 Simulation loop

39 Exports the root system geometry

42-43 We actually do not wish to set this geometry, but we abuse the writer of the class RootSytem to export a Python script showing the layer geometry. The resulting ParaView visualization is presented in Figure 10.

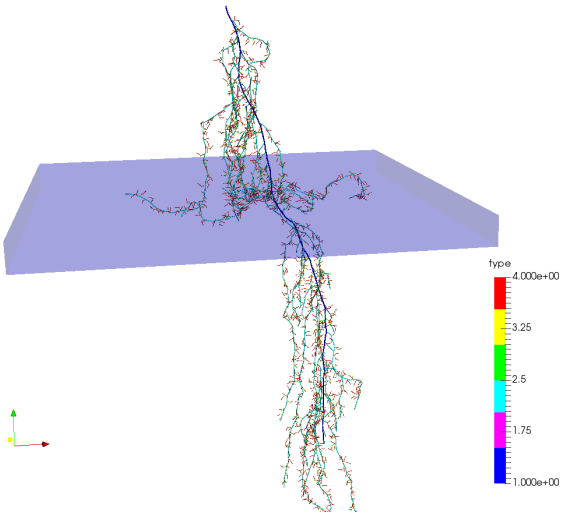


Figure 10: Chemotropism in a nutrient rich layer (Example 4a)

5.2 User defined tropisms

Normally, the simulation is created from a set of parameters. For tropisms these are the type of tropism tt, number of trials N , and tortuosity σ. There are two ways to add user defined tropisms:

1. The first is C++ only: It is to extend the CRootBox class and overwrite the method RootSystem::createTropism. This is the function that is called, when the tropisms are created from the parameters (tt, N, σ) in RootSystem::initialize(). This is necessary when user defined tropism are created from a parameter file, where tt is the number of the new tropism type.

2. The second is to manually set the tropisms using the method RootSystem::setTropism. Make sure to call Rootsystem::setTropism(...) after Rootsystem::initialize().

In both approaches the user has to extend the new tropism class from the class Tropism. If just the objective function of the tropism is changed, it is enough to overwrite Tropism::tropismObjective. This can done in Python or in C++,the classes Hydrotropism, Gravitropism, and Plagiotropism are examples for this procedure.

If the whole concept of the random optimization is altered, Tropism: :getUCHeading must be overwritten. If the geometry model is also changed Tropism: :getHeading must be overwritten.

The following example shows how to implement a new tropism in Python. Two new tropism are introduced: The first does nothing put to output the incoming arguments of the method Tropism::tropismObjective to the command line (e.g. for debugging). The second one, is a Plagiotropism that changes with time to Gravitropism depending on the root age.

```
1 """user defined tropism in python"""
```

```
import sys
sys.path.append("../../..")
import plantbox as pb


# User tropism 1: print input arguments to command line
class My_Info_Tropism(pb.Tropism):

    def tropismObjective(self, pos, old, a, b, dx, root):
        print("Postion \t", pos)
        print("Heading \t", old.column(0))
        print("Test for angle alpha = \t", a)
        print("Test for angle beta = \t", b)
        print("Eesolution of next segment \t", dx)
        print("Root id", root.getId())
        print()
        return 0.


# User tropism 2: depending on root age use plagio- or gravitropism
class My_Age_Tropism(pb.Tropism):

    def __init__(self, rs, n, sigma, age):
        super(My_Age_Tropism, self).__init__(rs)
        self.plagio = pb.Plagiotropism(rs, 0., 0.)
        self.gravi = pb.Gravitropism(rs, 0., 0.)
        self.setTropismParameter(n, sigma)
        self.age = age

    def tropismObjective(self, pos, old, a, b, dx, root):
        age = root.getAge()
        if age < self.age:
            d = self.plagio.tropismObjective(pos, old, a, b, dx,
    root)
            return d
        else:
            return self.gravi.tropismObjective(pos, old, a, b, dx,
    root)


# set up the root system
rs = pb.RootSystem()
path = "../../../modelparameter/rootsystem/"
name = "Anagallis_femina_Leitner_2010"
rs.readParameters(path + name + ".xml")
rs.initialize()

# Set useer defined after initialize
mytropism1 = My_Info_Tropism(rs)
mytropism1.setTropismParameter(2., 0.2)
mytropism2 = My_Age_Tropism(rs, 2., 0.5, 5.)  # after 5 days switch
     from plagio- to gravitropism
rs.setTropism(mytropism2, 2)  # 2 for laterals, -1 for all root
    types

# Simulate
simtime = 100  # e.g. 30 or 60 days
dt = 1
N = round(simtime / dt)
for _ in range(0, N):
    rs.simulate(dt)

```

```
60 # Export results (as vtp)
61 rs.write("../results/example_5b.vtp")
```

Listing 14: Example 5b

3-14 Creates a new tropism that just writes incoming arguments of Tropism: :tropismObjective to the command line. This can be used for debugging. The new class is extended from rb.Tropism, and the method Tropism::tropismObjective is overwritten with the right number of arguments.

16-32 Again, we extend the new class from rb.Tropism. In L19-24 we define our own constructor. Doing this two things are important: (1) the constructor of the super class must be called (L20), and (2) the tropism parameters n, and σ must be set (L23). Furthermore, the constructor defines two tropisms: plagio- and gravitropism, that are used in Tropism::tropismObjective at a later point, and a root age, when to switch from plagi- to gravitropism.
In L26-L32 the method Tropism::tropismObjective is defined. We choose the predefined objective function depending on the root age.

34-38 Sets up the simulation.

40-44 L40,L41 creates the first user defined tropsim. Since we did not define a constructor Tropism::setTropismParameter must be called. L43 creates the second user defined tropism. In L44 the tropism is chosen, using the method Tropism::setTropism. The second argument states for which root type it applies. Number 2 is the root type number of the laterals, -1 states that the tropism applies for all root types (default = -1).

46-51 The simulation loop.

54 Exports the result producing Figure (11). Comparing to Figure (1a) we can see the effect of the new user defined tropsim.

6 Root functional modelling

Root growth is strongly influenced by pedo-climate conditions, and plant internal state. CRootBox offers build in ways to develop such models. In this section we assume static soil conditions, and describe predefined ways how the soil can affect root growth. Dynamic soil conditions are described in the following section 'Coupling with a soil model'.

Implemented root responses are (1) the change in direction of the growing root tip, described in Sections 5.1 and 5.2 (2) the scaling of the elongation rate (3) the change of insertion angle (4) the change of lateral emergence probability, (2)-(4) are described in Section 6.1.

6.1 Scaling elongation rate, insertion angle, and lateral emergence probability

The following script is an example to this and the following two sections, where (a) the elongation rate, (b) the insertion angle, and (c) the probability of lateral emergence are scaled.

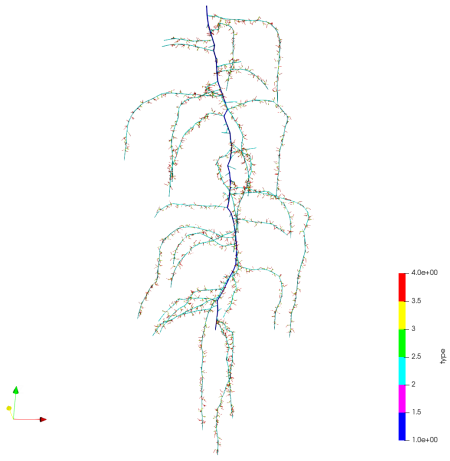


Figure 11: Depending on root age the laterals follow plagio- or gravitropism (Example 4b)

Without giving a specific model these mechanisms are considered important. For example the elongation rate as well as the probability of lateral emergence is dependent on soil properties like water saturation, soil strength, and temperature (among others). The insertion angle is reported to be dependent on nutrient supply in some species. Furthermore, theses mechanisms are influenced by plant systemic responses.

The scaling itself can be performed in the following way:

```
"""Three types of interaction, setting f_se, f_sa, f_sbp"""
import sys
sys.path.append("../../..")
import plantbox as pb
import math

rs = pb.RootSystem()
path = "../../../modelparameter/rootsystem/"
name = "Anagallis_femina_Leitner_2010"
rs.readParameters(path + name + ".xml")

# box with a left and a right compartment for analysis
sideBox = pb.SDF_PlantBox(10, 20, 50)
left = pb.SDF_RotateTranslate(sideBox, pb.Vector3d(-4.99, 0, 0))
right = pb.SDF_RotateTranslate(sideBox, pb.Vector3d(4.99, 0, 0))
leftright = pb.SDF_Union(left, right)
rs.setGeometry(leftright)

# left compartment has a minimum of 0.01, 1 elsewhere
maxS = 1.  # maximal
minS = 0.01  # minimal
slope = 1.  # [cm] linear gradient between min and max
leftC = pb.SDF_Complement(left)
soilprop = pb.SoilLookUpSDF(leftC, maxS, minS, slope)  # for root
    elongation
soilprop2 = pb.SoilLookUpSDF(left, 1., 0.002, slope)  # for
    branching

```

```
27 # Manually set scaling function and tropism parameters
28 sigma = [0.4, 1., 1., 1., 1. ] * 2
29 for p in rs.getRootRandomParameter():
30     p.dx = 0.25  # adjust resolution
31     p.tropismS = sigma[p.subType - 1]
32
33     # 1. Scale elongation
34 #    p.f_se = soilprop
35
36     # 2. Scale insertion angle
37 #    p.f_sa = soilprop
38
39 # 3. Scale branching probability
40 p = rs.getRootRandomParameter(2)
41 p.ln = p.ln / 5
42 p.nob = p.nob * 5
43 p = rs.getRootRandomParameter(3)
44 p.f_sbp = soilprop2
45
46 # simulation
47 rs.initialize()
48 simtime = 120.
49 dt = 1.
50 N = 120 / dt
```

Listing 15: Example 6a (1)

4-6 Creates the root system and opens the parameter file

8-13 We create a confining box with two overlapping boxes left and right. This geometries are used for later analysis.

15-21 We define two static soil properties using SDF (L20, L21) as explained in Section 5.1. The left compartment has the value $minS$, the right $maxS$, between them is a linear gradient of length *slope*.

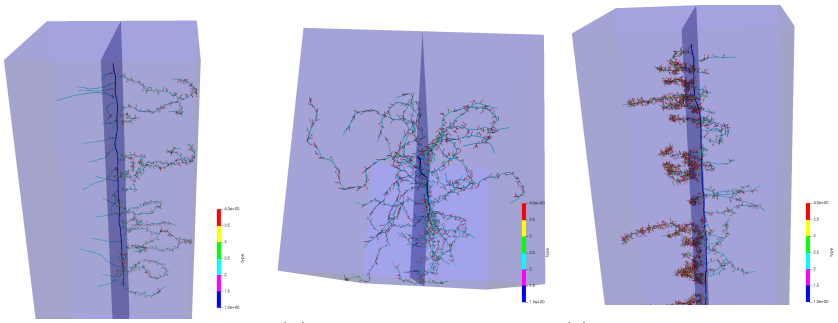
23-34 Sets the scaling functions. L24-L28 adjusts axial resolution and tortuosity *sigma*. L31 sets the scale elongation function *se* to the soil property. L34 sets the scale insertion angle function *sa*. Comment and uncomment the relevant code parts to achieve the desired scaling, to achieve the resulting Figures 12a and 12b.

36-41 Sets the lateral emergence probability. First, we increase the number of laterals of the first laterals (root type 2) by a factor of five and decrease the inter-nodal distances for the same factor L37-L39. This is the maximal lateral density that can be achieved. Then, we set the scale branching probability function *sbp* to the soil property defined in L21, see Figure 12c.

44-49 Initialization and simulation loop.

When playing with model parameters, it is not always clear if the suggested effect is realized from the figures alone. The following script helps to quantify the effects of above simulation:

Listing 16: Example 6bÂ´ (2)



(a) Root elongation rate (b) Insertion angle of laterals (c) Branching density (probabilistic model)

Figure 12: Predefined root responses (Example 4c)

6.2 Soil

6.3 User defined soil

7 Model coupling

In the previous section root responses were described in a static soil. In this section we will extend this to a dynamic soil setting, where we update the soil in the simulation loop, and then update the root system iteratively for small time steps.

General properties of the soil, are passed to the root model via a look up method SoilLookUp::getValue in the class SoilLookUp. In the following subsection we will first describe this metod and some implemented usefull extensions of the SoilLookUp class (Section 6.2), and show how we can create an interface to a generic soil in Pyhton (Section 6.3).

Next, we show how we can use the soil representation to implement fully coupled models. First we discuss how to obtain a graph representation of the root system, and solve water flow within the root system. Then we discuss the example published in .

Finally, we present features that can be used to analyse the dynamic behaviour of the root system development.

7.1 Graph representation

In this section we show how to build an adjacency matrix, and how to calculate fluxes within the root system.

7.2 Coupling to 1D water content

Explain and link to paper example (to do)

7.3 Dynamic root system grid

In this section it is described how information about the last time step can be retrieved, and how we can incrementally obtain the root system from only new

nodes and segments. These methods are especially important if we couple to other numerical software like DuMux

References