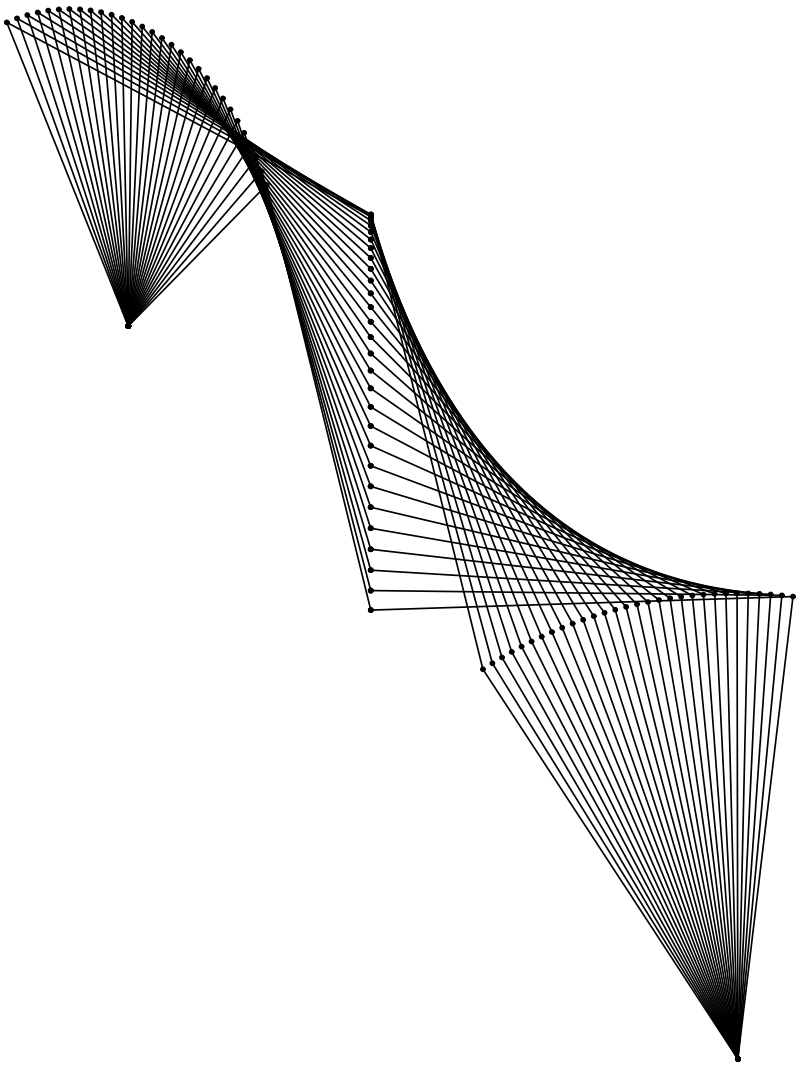
Modeling linkages with linkR

Simulating three-dimensional linkage mechanisms using the R package linkR



Aaron Olsen

June 17, 2015

linkR 1.0.1

Table of Contents

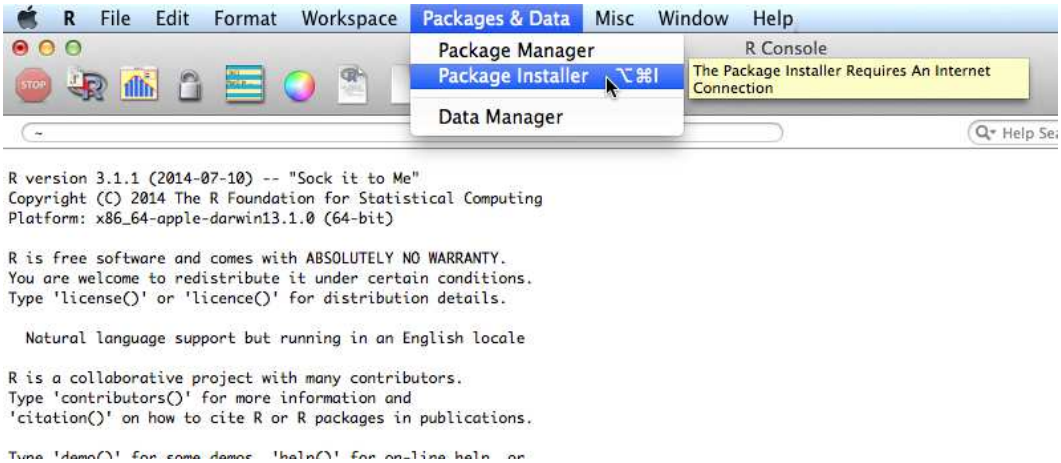
Getting started

This tutorial will show you how to create 2D and 3D linkage models using the R package linkR. The R project is a computing language and platform that allows users to freely upload and share software packages. The linkR package includes functions for simulating and analyzing the kinematics of 3D linkage mechanisms. The linkR package uses the R package svgViewR to create 3D interactive animations that can visualized in the web browser. The following steps will take you through process of installing the linkR package.

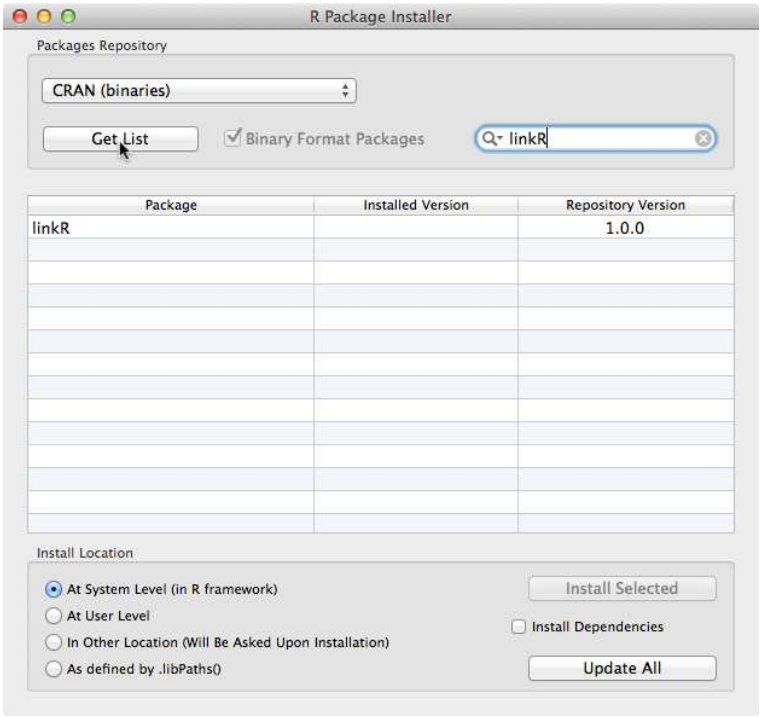
1. If you do not already have R installed on your computer, begin by installing R. R can be installed on Windows, Linux and Mac OS X.

2. Once installed, open R.

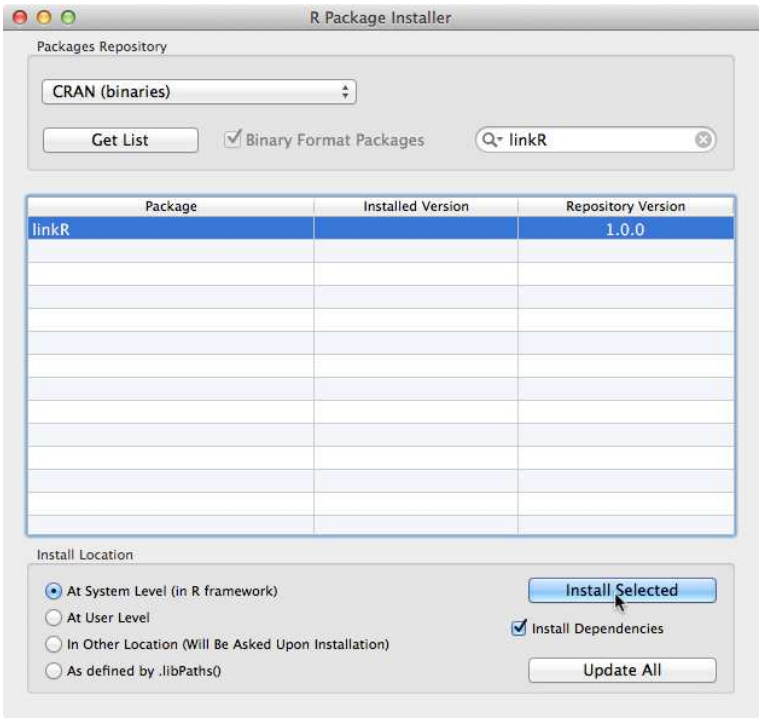
3. Go to Packages & Data > Package Installer.



4. Find the linkR package binary by typing "linkR" into the Package Search box and clicking Get List. (The repository version of linkR might be more recent than the version in the image below).



5. Check the box next to Install Dependencies. This ensures that all the packages that linkR requires to run (e.g. svgViewR) will be installed as well. Then click Install Selected to install linkR.



6. Load the linkR package into the current R session using the library command.

```
> library(linkR)
```

7. Lastly, you'll need a compatible web browser to visualize the interactive linkage animations (basically any browser except Internet Explorer).

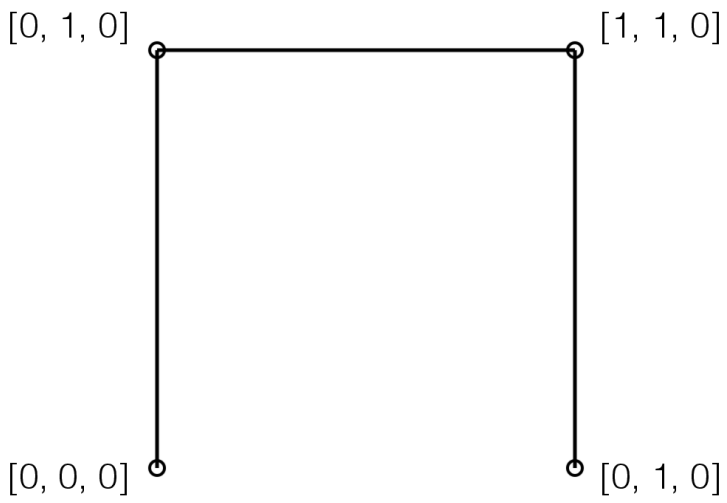
You are now ready to create linkage models!

Defining a linkage

Linkages are represented in linkR as a chain or network of links connected by joints, each of which allows a particular type of motion. This linkage chain or network is defined by four inputs:

1. The joint coordinates
2. The type of motion permitted at each joint
3. A vector describing the permitted motion, if applicable
4. The two links connected by each joint

Joint coordinates. The joint coordinates are specified as a two- or three-column matrix (for two or three dimensions, respectively) indicating the initial positions of the joints. For example, the joint coordinates of a simple, planar four-bar linkage (three mobile links and one ground link) with the joints initially arranged into a square



A simple four-bar linkage consisting of four joints and three mobile links. The lower two joints are connected to ground.

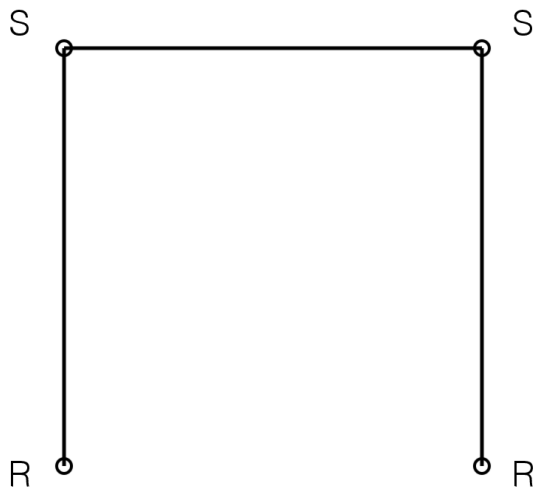
would be defined as follows:

```
> joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,1,0), c(1,0,0))
> joint.coor
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    1    0
[3,]    1    1    0
[4,]    1    0    0
```

Joint types. The joint type refers to the type of motion (translation and/or rotation) permitted at each joint. linkR currently supports four constraint types:

- **Linear** (**L**). Permits translation along a single line or vector
- **Planar** (**P**). Permits translation within a plane
- **Rotational** (**R**). Permits rotation about a single axis
- **Spherical** (**S**). Permits rotation about all three axes (including long-axis rotation)

The joint types are specified as a vector of the above abbreviations (L, P, R or S), having the same number of elements as the number of rows in the joint coordinate matrix and in the same order. For example, a simple four-bar linkage can be defined as having two R-joints connected to ground and two S-joints in between that permit rotation in any direction. These joint constraints give a four-bar mechanism, in two or three dimensions, a single degree of freedom.



Joint types for a simple four-bar linkage: rotational (R) and spherical (S).

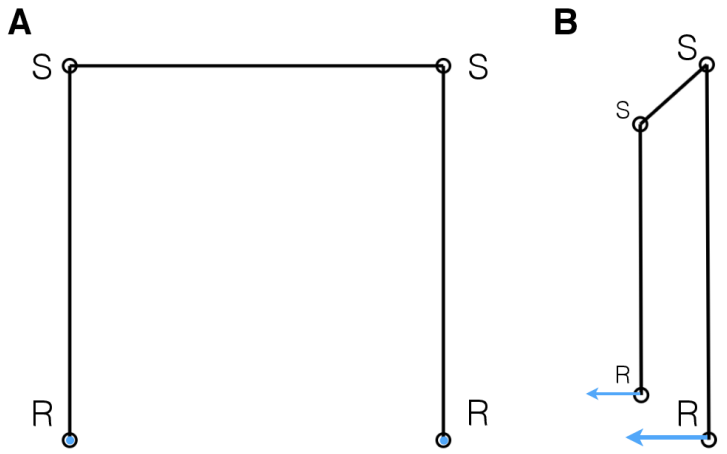
The corresponding vector to define these joint constraints would then be:

```
> joint.types <- c('R', 'S', 'S', 'R')
```

Joint constraint vectors. The joint constraint vector refers to a vector that describes the orientation or direction of motion permitted by each joint. For a linear joint, this vector is the line along which the joint moves. For a planar joint, the vector is perpendicular (orthogonal) to the plane in which the joint moves. For a rotational joint, the vector is the axis of rotation. Since a spherical joint permits rotation in all directions, no vector is needed for this joint type. It should instead be NA.

The joint types are specified as a list having the same number of elements as the number of rows in the joint coordinate matrix and in the same order. For example, for a simple two-dimensional (planar) four-bar linkage, the axes of rotation at the two R-joints are oriented perpendicular to the plane of the linkage. For the previously given coordinates these vectors are oriented parallel to the z-axis.

```
> joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1))
```



The joint constraint vectors (blue) for two R-joints in a planar four-bar linkage seen in full-on view (A) and an oblique side view (B). The vectors are orthogonal to the linkage plane.

Joint connections. Linkages are defined in linkR as links interconnected by joints, with each joint connecting two (and only two) links. The two links connected by each joint are specified in a two-column matrix in which each column corresponds to the two links and each row corresponds to each joint. Thus, the matrix should have the same number of rows, and be in the same order, as the joint coordinate matrix.

Each linkage will have a single ground link, which is link 0. The numbering of the remaining links is arbitrary but will be used to associate linkage input motions with the appropriate link. For the simple four-bar linkage, the joint connects form a simple chain: the first joint connects the ground link to link 1, the second joint connects link 1 to link 2, etc. The matrix defining the joint connections would be as follows:

```
> joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0))
> joint.conn
     [,1] [,2]
[1,]    0    1
[2,]    1    2
[3,]    2    3
[4,]    3    0
```

While each joint is allowed to connect only two links, links may be connected by any number of joints. For linkages that have parallel, interconnected sets of links this matrix (see Examples), the joint.conn matrix specifies these branched interconnections among the links.

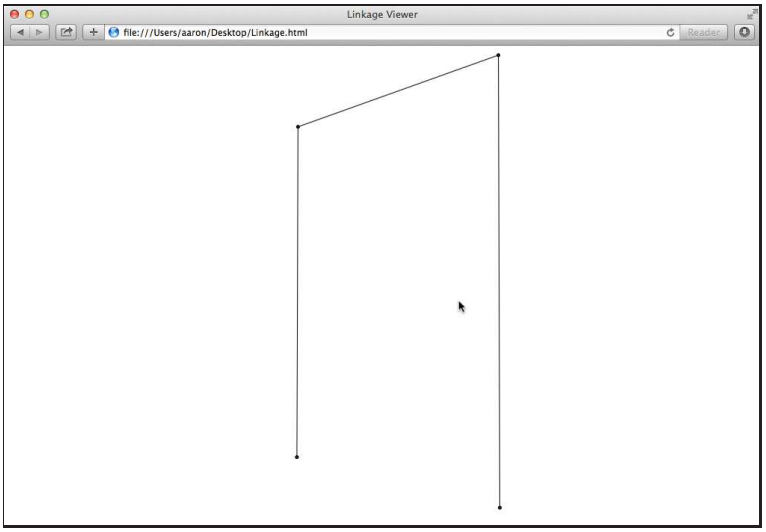
These four inputs are passed to the function "defineLinkage" in order to define the linkage for kinematic simulation:

```
> linkage <- defineLinkage(joint.coor=joint.coor,
   joint.types=joint.types, joint.cons=joint.cons,
   joint.conn=joint.conn)
```

The returned object (here, "linkage") can be passed to other linkR functions for visualization and kinematic simulation. The linkR package works with a new R package, svgViewR, to create interactive, 3D animated visualizations of linkages that can be viewed in any major web browser. To visualize the linkage in its initial state, input the linkage object into the function "drawLinkage", specifying the name of the file to be written to the current working directory.

```
> drawLinkage(linkage, file='Linkage')
```

If you don't specify a file extension, drawLinkage will create an ".html" file using the specified file name.

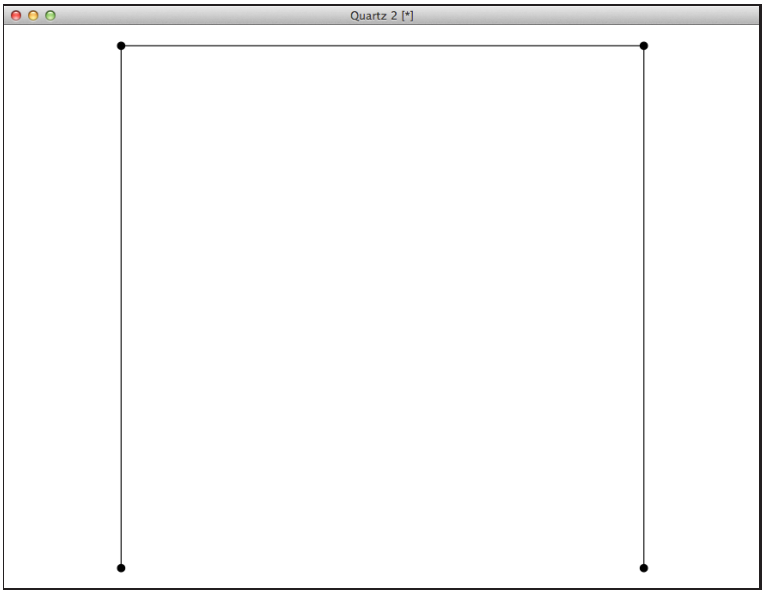


Interactive linkage visualization in a web browser created by drawLinkage (using the R package svgViewR).

Once you open this file in a web browser you can hold down the "r" key and click-and-drag the mouse to rotate the linkage in space and scroll to zoom in and out. The viewing file is also entirely self-contained so it can be moved and viewed on any platform. See svgViewR interactive commands for more details on using the svgViewR interactive viewer.

You can also visualize the linkage using the native R plot tools by calling drawLinkage without an input for 'file':

```
> drawLinkage(linkage)
```

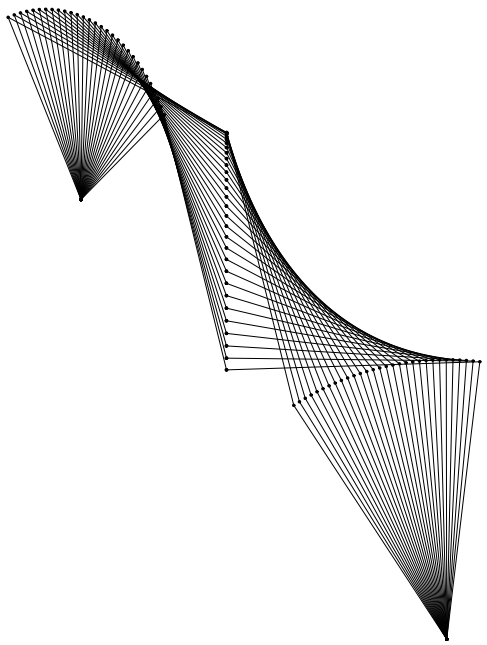


The four-bar linkage visualized in the R plot device.

However, this does not provide the animation or interactive features of the svgViewR visualization.

Simulating linkage motion

The linkR function "animateLinkage" allows users to simulate linkage motion, or kinematics, for a specified motion at an input link. The function uses analytical geometry to solve for the joint positions that satisfy the specified joint constraints and maintain constant link lengths.



Simulated motion of a five-bar linkage with an intermediate planar sliding joint.

Currently, linkR only predicts the motion of linkages having a single degree-of-freedom (single solution). Rotating links in linkages often yield two possible solutions, including the four-bar linkage introduced previously. However these solutions are discontinuous and linkR can use the previous positions of joints in the simulation to find the solution closest to the previous time step, resulting in an unambiguous single solution (with the exception of toggle points). Linkages with multiple degrees of freedom can be solved in linkR by including a sufficient number of input parameters to create a single degree-of-freedom linkage.

Once you've defined a linkage you can input this linkage to the function "animateLinkage" along with the input motion and the joint at which motion is to be input. The input parameter 'input.param' is a vector of rotations or translations to be applied at an R-joint and L- or P-joint, respectively, and the input parameter 'input.joint' is the joint at which the input motion is to be applied.

For example, the following will rotate one of the rotating input links of a simple four-bar linkage over three angles from 0 to 0.8 radians (0 to about 45 degrees):

```
> animate <- animateLinkage(linkage, input.param=c(0,0.3,0.8),
   input.joint=1)
```

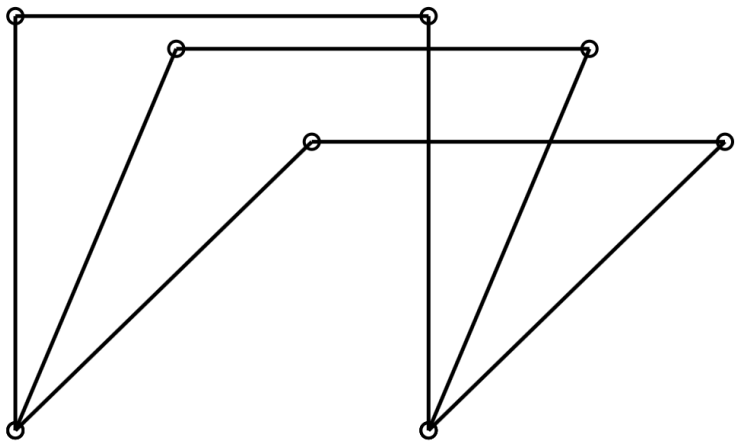
You can visualize this motion using the "drawLinkage" function:

```
> drawLinkage(animate, file='Animate')
```

By default, the drawLinkage function creates an interactive animation file using the R package svgViewR that can be opened in any major web browser. Once you open this file in a web browser you can hold down the "r" key and click-and-drag the mouse to rotate the linkage in space and scroll to zoom in and out. See svgViewR interactive commands for more details on using the svgViewR interactive viewer.

You can also visualize the motion by superimposing each frame in a single, static object by setting the 'animate' parameter to FALSE:

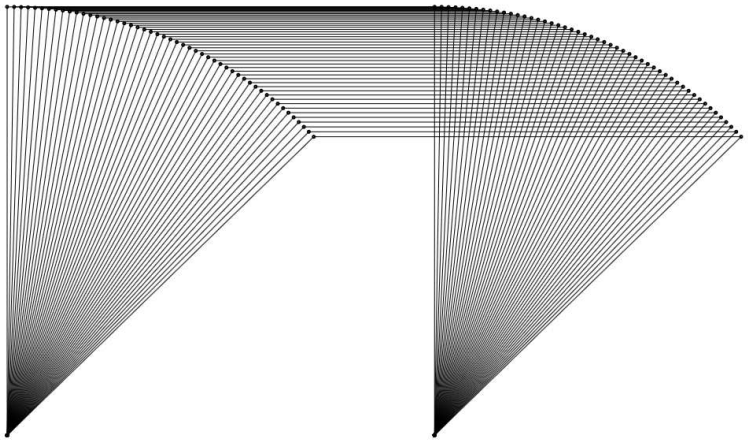
```
> drawLinkage(animate, file='Animate', animate=FALSE)
```



A simple four-bar linkage rotated over three angles, each frame superimposed.

To simulate motion over a finer scale and create a smoother animation, you can increase the number of angles in 'input.param'.

```
> animate <- animateLinkage(linkage, input.param=seq(0,0.8,length=50),
   input.joint=1)
> drawLinkage(animate, file='Animate')
```



A simple four-bar linkage rotated over 50 angles, each frame superimposed.

You can run the simulation in the reverse direction to create a looped animation by either adding angles in the reverse direction through 'input.param'

```
> input.param <- c(seq(0,0.8,length=50), seq(0.8,0,length=50))
> animate <- animateLinkage(linkage, input.param=input.param,
   input.joint=1)
```

or by setting the 'animate.reverse' argument for drawLinkage to TRUE.

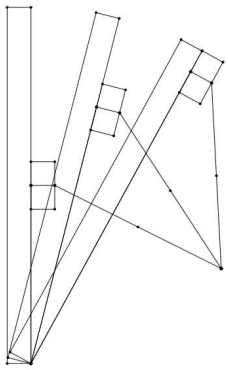
```
> drawLinkage(animate, file='Animate', animate.reverse=TRUE)
```

Examples

The following section contains examples to demonstrate the different types of linkage networks that can be modeled using linkR. To run each example, first launch R (see Getting Started). Set your working directory to whichever folder you like (e.g. Desktop). Then copy and paste the code in each example directly from the PDF into the R console. Each example can be run independently. The example will create an interactive animation file in the current working directory that can be opened in most major web browsers. See svgViewR interactive commands for how to interact with the animation.

RLSS

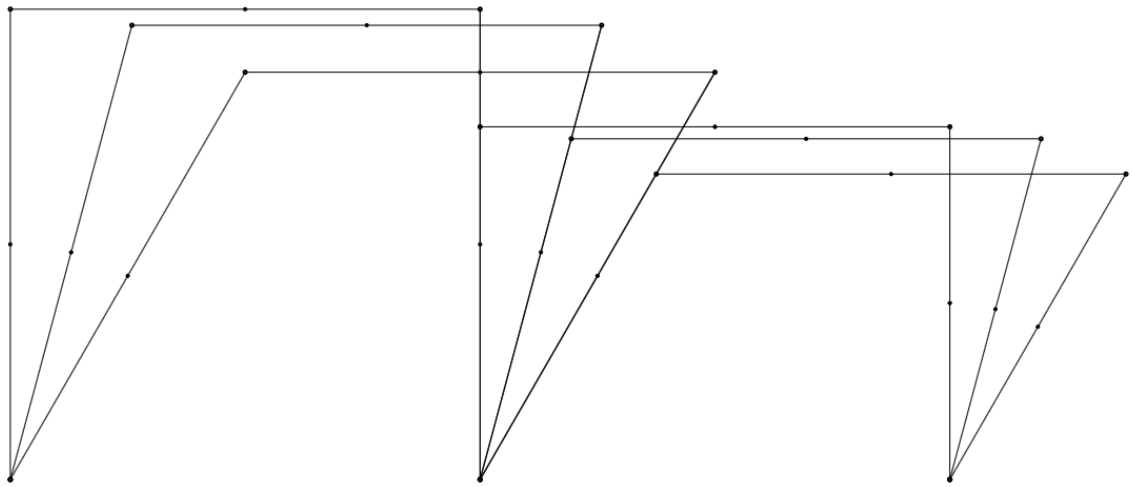
This example creates a sliding link that slides along a rotating link. The sliding link itself is constrained to rotate about a point by a fixed length.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.75,0), c(0.1,0.75,0), c(0.8,0.4,0))

# Joint types
joint.types <- c("R", "L", "S", "S")

# Joint constraint vectors
joint.cons <- list(c(0,0,1), c(0,1,0), NA, NA)

# Names of each link
link.names <- c('Ground', 'Link1', 'Link2', 'Link3')

# The links connected by each joint
joint.conn <- rbind(c('Ground', 'Link1'), c('Link1', 'Link2'),
       c('Link2', 'Link3'), c('Link3', 'Ground'))

# Input parameters, with 100 iterations
input.param <- seq(from=0, to=0.5, length=100)

# Joint at which to apply input parameters
input.joint <- 1

# Add points to clarify the visualization
points <- rbind(c(0,0.65,0), c(0.1,0.65,0), c(0.1,0.85,0), c(0,0.85,0), c(0,0,0),
       c(-0.1,0,0), c(-0.1,1.5,0), c(0,1.5,0), c(0.45, 0.575, 0))

# The link each point is associated with (the index of the corresponding link in link.names)
link.assoc <- c(2,2,2,2,1,1,1,1,3)

# Lines to connect points for easier visualization
path.connect <- list(c(1:4,1), c(5:8,5))

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn, link.names=link.names, points=points, link.assoc=link.assoc)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RLSS', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, path.connect=path.connect, window.title='RLSS')
```

RSSRSSR

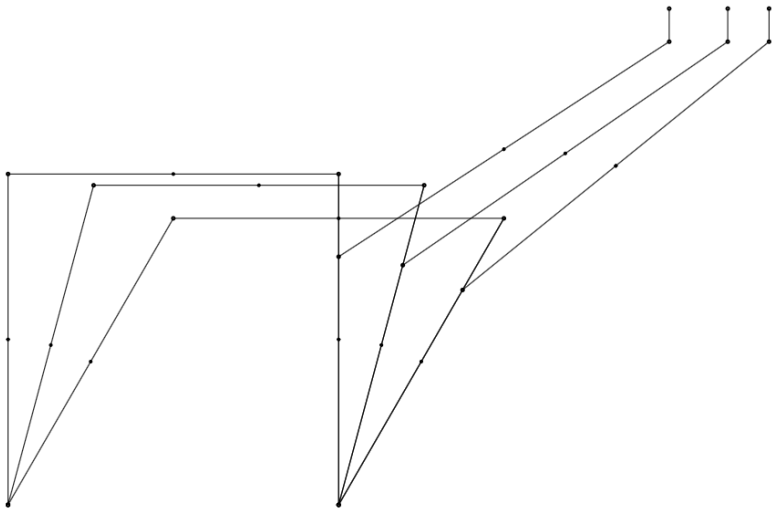
This example creates two four-bar linkages that share a rotating link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,1,0), c(1,0,0), c(1,0.75,0), c(2,0.75,0), c(2,0,0))

# Joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "R")

# Joint constraint vectors
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1), NA, NA, c(0,0,1))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Input parameters, with 100 iterations
input.param <- seq(0*(pi/180), 30*(pi/180), length=100)

# Joint at which to apply input parameters
input.joint <- 1

# Add points to clarify the visualization
points <- rbind(c(0,0.5,0), c(0.5,1,0), c(1,0.5,0), c(1.5,0.75,0), c(2,0.375,0))

# The link each point is associated with (the index of the corresponding link in link.names)
link.assoc <- c(1,2,3,4,5)

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn, points=points, link.assoc=link.assoc)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RSSRSSR', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, window.title='RSSRSSR')
```

RSSR(SSL)

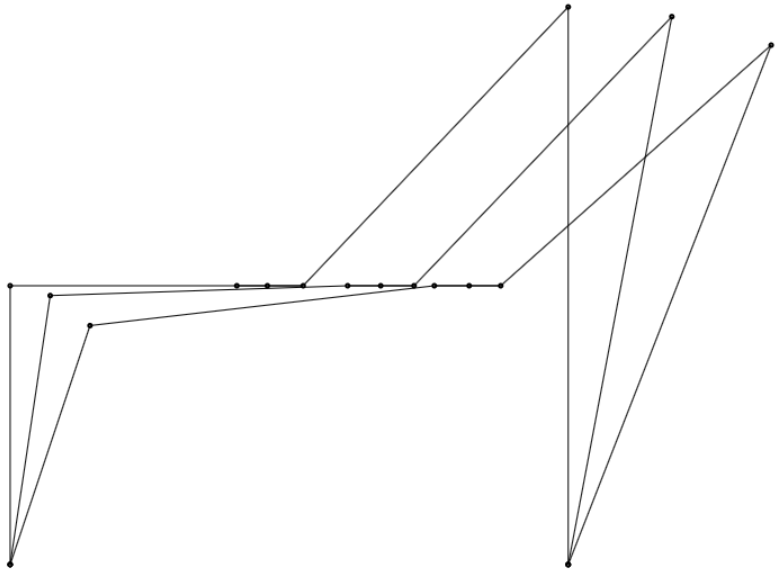
This example creates a four-bar linkage with a fixed link joining one of the rotating links and a sliding link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,1,0), c(1,0,0), c(1,0.75,0), c(2,1.4,0), c(2,1.5,0))

# Joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "L")

# Joint constraint vectors
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1), NA, NA, c(1,0,0))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Input parameters, with 100 iterations
input.param <- seq(0*(pi/180), 30*(pi/180), length=100)

# Joint at which to apply input parameters
input.joint <- 1

# Add points to clarify the visualization
points <- rbind(c(0,0.5,0), c(0.5,1,0), c(1,0.5,0), c(1.5,1.075,0))

# The link each point is associated with (the index of the corresponding link in link.names)
link.assoc <- c(1,2,3,4)

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn, points=points, link.assoc=link.assoc)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RSSR(SSL)', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, window.title='RSSR(SSL)')
```

RSSLSSR

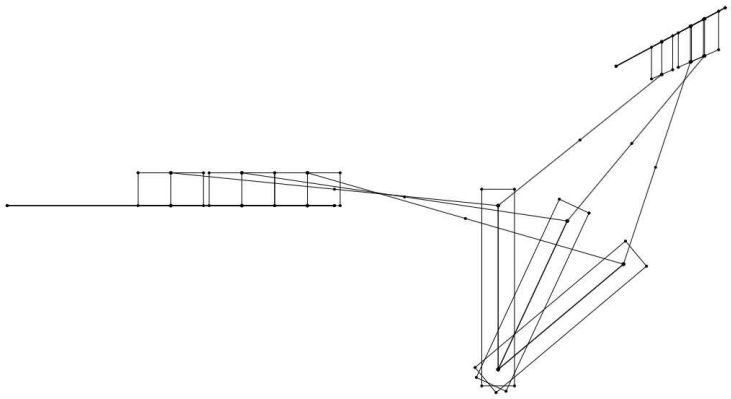
This example creates a 3D (spatial) linkage with two rotating links coupled by a slider link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1.3,0), c(0.9,1,0.9), c(1,1,1), c(1.1,1,0.9), c(2,2,0), c(2,0,0))

# Joint types
joint.types <- c("R", "S", "S", "L", "S", "S", "R")

# Joint constraint vectors
joint.cons <- list(c(-1,0,1), NA, NA, c(1,0,0), NA, NA, c(0,0,1))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Input parameters, with 100 iterations
input.param <- seq(0*(pi/180), 30*(pi/180), length=100)

# Joint at which to apply input parameters
input.joint <- 1

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RSSLSSR', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, window.title='RSSLSSR')
```

LSSRSSL

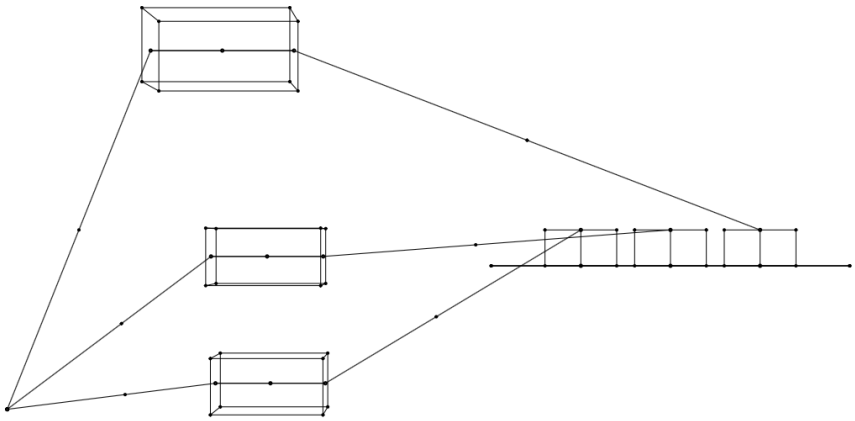
This example creates a 3D (spatial) linkage with two sliding links coupled through a rotating link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(-0.5,0.5,0), c(-0.5,0.6,0), c(0.5,0.5,0),
       c(0.5,0,0), c(0.5,0.5,0), c(1,0.9,0), c(1,1,0))

# Joint types
joint.types <- c("L", "S", "S", "R", "S", "S", "L")

# Joint constraint vectors
joint.cons <- list(c(1,0,0), NA, NA, c(0,0,1), NA, NA, c(0,0,1))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Input parameters, with 100 iterations
input.param <- seq(0*(pi/180), 50*(pi/180), length=100)

# Joint at which to apply input parameters
input.joint <- 4

# Add points to clarify the visualization
points <- rbind(c(-0.6,0.5,0), c(-0.6,0.6,0), c(-0.4,0.6,0), c(-0.4,0.5,0), c(1,0.9,-0.1),
       c(1,1,-0.1), c(1,1,0.1), c(1,0.9,0.1), c(-1,0.5,0), c(0,0.5,0), c(1,1,-0.5), c(1,1,0.5),
       c(0.45,-0.05,0), c(0.45,0.55,0), c(0.55,0.55,0), c(0.55,-0.05,0), c(0,0.55,0), c(0.75,0.7,0))

# The link each point is associated with (the index of the corresponding link in link.names)
link.assoc <- c(rep(1,4), rep(5,4), rep(0,4), rep(3,4), 2, 4)

# Lines to connect points for easier visualization
path.connect <- list(c(1:4,1), c(5:8,5), c(9,10), c(11,12), c(13:16,13))

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn, points=points, link.assoc=link.assoc)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='LSSRSSL', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, path.connect=path.connect, window.title='LSSRSSL')
```

SSPSSL

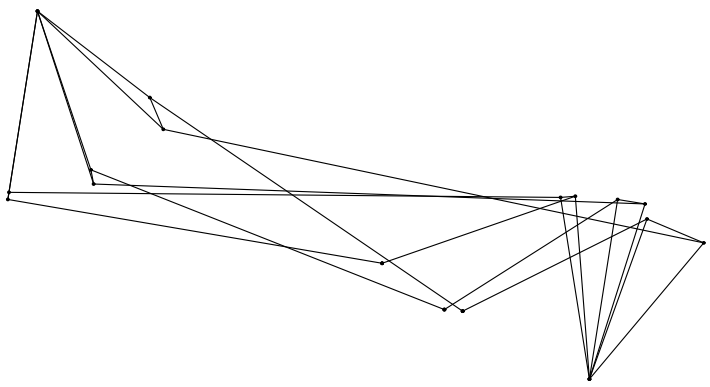
This example creates a 3D (spatial) linkage with planar sliding link coupled to a linear sliding link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(-0.6,0,0), c(-0.2,1,0), c(0,1,0), c(0.2,1,0), c(1.5,0.5,0), c(1.5,0.4,0))

# Joint types
joint.types <- c("S", "S", "P", "S", "S", "L")

# Joint constraint vectors
joint.cons <- list(NA, NA, c(1,0,0), NA, NA, c(1,0,0))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,0), c(2,3), c(3,4), c(4,0))

# Input parameters, with 100 iterations
input.param <- seq(0, -0.5, length=100)

# Joint at which to apply input parameters
input.joint <- 6

# Add points to clarify the visualization
points <- rbind(c(-0.4,0.5,0), c(1.4,0.4,0), c(1.4,0.5,0), c(1.6,0.5,0), c(1.6,0.4,0),
       c(0.75,0.4,0), c(1.75,0.4,0), c(0.85,0.75,0),
       c(-0.2,0.9,0.1), c(0.2,0.9,0.1), c(0.2,0.9,-0.1), c(-0.2,0.9,-0.1),
       c(-0.2,1.1,0.1), c(0.2,1.1,0.1), c(0.2,1.1,-0.1), c(-0.2,1.1,-0.1))

# The link each point is associated with (the index of the corresponding link in link.names)
link.assoc <- c(1,4,4,4,4,0,0,3,rep(2,8))

# Lines to connect points for easier visualization
path.connect <- list(c(2:5,2), c(6:7), c(9:12,9), c(13:16,13), c(9,13), c(10,14), c(11,15), c(12,16))

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn, points=points, link.assoc=link.assoc)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='SSPSSL', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, path.connect=path.connect, window.title='SSPSSL')
```

RSSRSSPSS

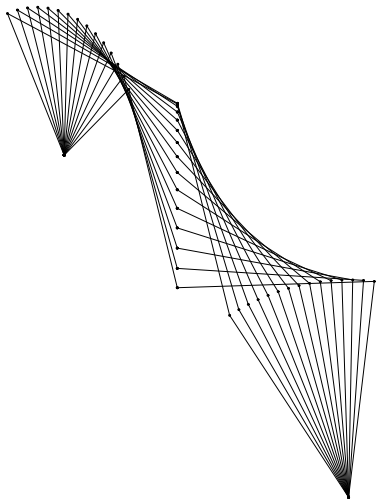
This example creates a 3D (spatial) 4-bar linkage coupled through a single coupler link and a coupler link broken by a sliding link.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(-1,2,0), c(-1,1,0), c(2,1,0), c(2,0,0),
       c(-1,1,0.5), c(1,0.5,0.5), c(1,0.5,0.5), c(1,0.5,0.5), c(2,1,0.5))

# Joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "P", "S", "S")

# Joint constraint vectors
joint.cons <- list(c(-0.5,0,1), NA, NA, c(0,1,1), NA, NA, c(0,1,1), NA, NA)

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(1,4), c(4,5), c(5,0), c(5,6), c(6,3))

# Input parameters, with 100 iterations
input.param <- seq(10*(pi/180), -50*(pi/180), length=100)

# Joint at which to apply input parameters
input.joint <- 1

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RSSRSSPSS', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, window.title='RSSRSSPSS')
```

RSSPSSR

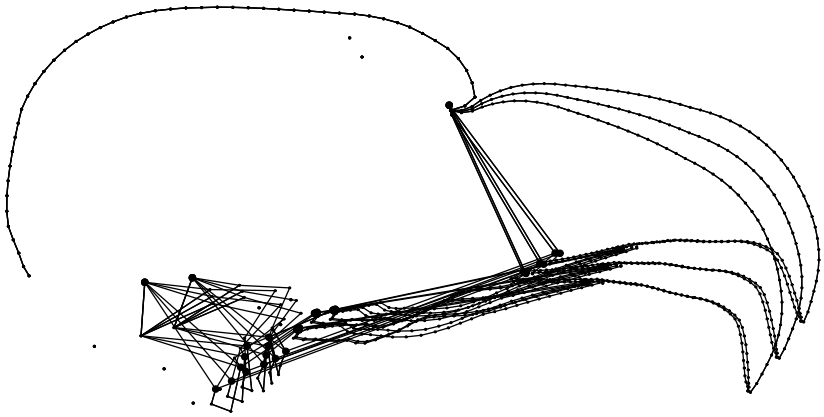
This example creates a 3D (spatial) 5-bar linkage with two rotating links coupled by a planar sliding link. This is a two degree-of-freedom mechanism which means that input motion parameters must be specified at each of the rotating joints.



```
# Load the linkR library
library(linkR)

# Joint coordinates
joint.coor <- rbind(c(-2,9,-2), c(-4,14,-2), c(2,11,0.5), c(2,11,0.5), c(2,11,0.5), c(5,4,0), c(8,-3,0))

# Joint types
joint.types <- c("R", "S", "S", "P", "S", "S", "R")

# Joint constraint vectors
joint.cons <- list(c(-1,0,1), NA, NA, c(1,0,0), NA, NA, c(0,0,1))

# The links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Input parameters, with 100 iterations
length <- 100
input.param <- list(seq(0*(pi/180), 80*(pi/180), length=length),
       seq(-10*(pi/180), 30*(pi/180), length=length))

# Joint at which to apply input parameters
input.joint <- c(1,7)

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=joint.cons,
       joint.conn=joint.conn)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw the linkage
drawLinkage(animate, file='RSSPSSR', animate=TRUE, animate.duration=3,
       animate.reverse=TRUE, window.title='RSSPSSR')
```

Owl cranial linkage network

The skulls of most birds have at least nine mobile bones that allow them to rotate their upper beak. This example simulates the motion of these bones in the Great Horned Owl, using data collected from a skull (specimen number 488595) in the bird collection at the Field Museum of Natural History. The data are included with the linkR package and are loaded using the linkR_examples function.

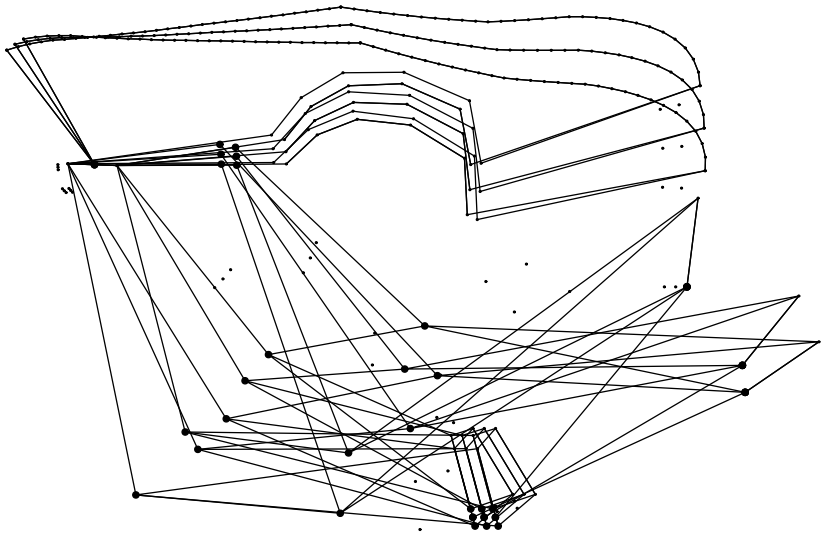


```
# Load the linkR library
library(linkR)

# Load example data for owl cranial linkages
owl <- linkR_examples('owl')

# Landmark data, joints used to construct the model and associated points for visualization
lm <- owl$landmarks

# Link name with which each landmark is associated
lm.assoc <- owl$lm.assoc

# Which points to connect with paths for easier visualization
path.connect <- owl$path.connect

# Set joint coordinates
joint.coor <- lm[c('quadrate_cranium_lc_R', 'quadrate_jugal_R', 'jugal_upperbeak_R',
       'nasalfrontalhinge_cranium_R', 'quadrate_pterygoid_R', 'pslider_pterygoid_R',
       'pterygoid_palatine_R', 'pslider_palatine_R', 'upperbeak_palatine_R',
       'quadrate_cranium_lc_L', 'quadrate_jugal_L', 'jugal_upperbeak_L',
       'quadrate_pterygoid_L', 'pslider_pterygoid_L', 'pterygoid_palatine_L',
       'pslider_palatine_L', 'upperbeak_palatine_L'), ]

# Joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "P", "S", "S",
       "R", "S", "S", "S", "S", "P", "S", "S")

# Joint constraint vectors
joint.cons <- list(
       lm['quadrate_cranium_lc_R', ] - lm['quadrate_cranium_mc_R', ], NA, NA,
       lm['nasalfrontalhinge_cranium_L', ] - lm['nasalfrontalhinge_cranium_R', ],
       NA, NA, lm['nasalfrontalhinge_cranium_L', ] - lm['nasalfrontalhinge_cranium_R', ], NA, NA,
       lm['quadrate_cranium_lc_L', ] - lm['quadrate_cranium_mc_L', ], NA, NA,
       NA, NA, lm['nasalfrontalhinge_cranium_L', ] - lm['nasalfrontalhinge_cranium_R', ], NA, NA)
```

```
# The links connected by each joint
joint.conn <- rbind(
       c('neurocranium', 'quadrate_R'), c('quadrate_R', 'jugal_R'),
       c('jugal_R', 'upperbeak'), c('upperbeak', 'neurocranium'),
       c('quadrate_R', 'pterygoid_R'), c('pterygoid_R', 'pp-slide_R'),
       c('pp-slide_R', 'neurocranium'), c('pp-slide_R', 'palatine_R'),
       c('palatine_R', 'upperbeak'), c('neurocranium', 'quadrate_L'),
       c('quadrate_L', 'jugal_L'), c('jugal_L', 'upperbeak'),
       c('quadrate_L', 'pterygoid_L'), c('pterygoid_L', 'pp-slide_L'),
       c('pp-slide_L', 'neurocranium'), c('pp-slide_L', 'palatine_L'),
       c('palatine_L', 'upperbeak'))

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
       joint.cons=joint.cons, joint.conn, points=lm, link.assoc=lm.assoc,
       ground.link='neurocranium')

# Set the animation input parameters
input.param <- seq(5*(pi/180), -10*(pi/180), length=100)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=1)

# Create an interactive linkage visualization
drawLinkage(animate, file="Owl cranial linkages", animate=TRUE, window.title="Owl cranial linkages",
       animate.duration=3, animate.reverse=TRUE, connect.joints=FALSE, path.connect=path.connect,
       point.cex=0.3, joint.cex=1, path.lwd=1)
```

Salmon cranial linkage network

The skulls of fish contain several parallel sets of mobile bones interconnected by joints and connective tissue. This example features the motion of a subset of these bones in the skull of the Atlantic Salmon, demonstrating how motion of the hyoid and neurocranium cause expansion of the mouth cavity during suction feeding. The data are included with the linkR package and are loaded using the linkR_examples function.



```
# Load the linkR library
library(linkR)

# Load example data for owl cranial linkages
salmon <- linkR_examples('salmon')

# Landmark data, joints used to construct the model and associated points for visualization
lm <- salmon$landmarks

# Link name with which each landmark is associated
lm.assoc <- salmon$lm.assoc

# Which points to connect with paths for easier visualization
path.connect <- salmon$path.connect

# Set joint coordinates
joint.coor <- lm[c('NC_VT', 'SUSP_NC_ANT_L', 'HYD_SUSP_L', 'HYD_BHYL_L',
       'BHYL', 'HYD_BHYL_R', 'HYD_SUSP_R', 'SUSP_NC_ANT_R',
       'BHYL_LJ', 'LJ_BHYL', 'LJ_SYMPH_VEN', 'LJ_QUAD_L',
       'LJ_SYMPH_L', 'LJ_SYMPH_R', 'LJ_QUAD_R'), ]

# Joint types
joint.types <- c("R", "R", "S", "S",
       "P", "S", "S", "R", "S", "S",
       "P", "S", "S", "S", "S")

# Joint constraint vectors
joint.cons <- list(
       lm['SUSP_NC_POS_L',]-lm['SUSP_NC_POS_R',],
       lm['SUSP_NC_ANT_L',]-lm['SUSP_NC_POS_L',],NA,NA,
       lm['SUSP_NC_ANT_L',]-lm['SUSP_NC_ANT_R',],NA,NA,
       lm['SUSP_NC_ANT_R',]-lm['SUSP_NC_POS_R',],NA,NA,
       lm['SUSP_NC_ANT_L',]-lm['SUSP_NC_ANT_R',],NA,NA,NA,NA)
```

```
# The links connected by each joint
joint.conn <- rbind(
       c('vert_column', 'neurocranium'), c('neurocranium', 'suspensorium_L'),
       c('suspensorium_L', 'hyoid_L'), c('hyoid_L', 'basihyal'),
       c('basihyal', 'vert_column'), c('basihyal', 'hyoid_R'),
       c('hyoid_R', 'suspensorium_R'), c('suspensorium_R', 'neurocranium'),
       c('basihyal', 'hyoid_lowerjaw'), c('hyoid_lowerjaw', 'lowerjaw_symph'),
       c('lowerjaw_symph', 'vert_column'), c('suspensorium_L', 'lowerjaw_L'),
       c('lowerjaw_L', 'lowerjaw_symph'), c('lowerjaw_symph', 'lowerjaw_R'),
       c('lowerjaw_R', 'suspensorium_R')
)

# Set long axis rotation constraints
lar_vec <- lm['SUSP_NC_POS_L',]-lm['SUSP_NC_POS_R',];
lar.cons <- list(
       list('link'='lowerjaw_L','type'='P','point'=lm['LJ_SYMPH_DOR',], 'vec'=lar_vec),
       list('link'='lowerjaw_R','type'='P','point'=lm['LJ_SYMPH_DOR',], 'vec'=lar_vec),
       list('link'='hyoid_R','type'='P','point'=lm['URO_HYP_DOR_R',], 'vec'=lar_vec),
       list('link'='hyoid_L','type'='P','point'=lm['URO_HYP_DOR_L',], 'vec'=lar_vec)
)

# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
       joint.cons=joint.cons, joint.conn, points=lm, link.assoc=lm.assoc,
       ground.link='vert_column', lar.cons=lar.cons)

# Set the animation input parameters
anim_len <- 100;
hyoid_tvec <- c(-1,0,0);
input.param <- list(
       seq(from=0*(pi/180), to=8*(pi/180), length=anim_len),
       rbind(matrix(uvector(hyoid_tvec), nrow=anim_len, ncol=3, byrow=TRUE)*
              matrix(seq(0, 3, length=anim_len), nrow=anim_len, ncol=3)
       )
)

# Animate the linkage
animate <- animateLinkage(linkage, input.param=input.param, input.joint=c(1,5))

# Create an interactive linkage visualization
drawLinkage(animate, file="Salmon cranial linkages", animate=TRUE,
       window.title="Salmon cranial linkages", animate.duration=3,
       animate.reverse=TRUE, connect.joints=FALSE,
       path.connect=path.connect, point.cex=0.3, joint.cex=1, path.lwd=1)
```