skijumpdesign Documentation

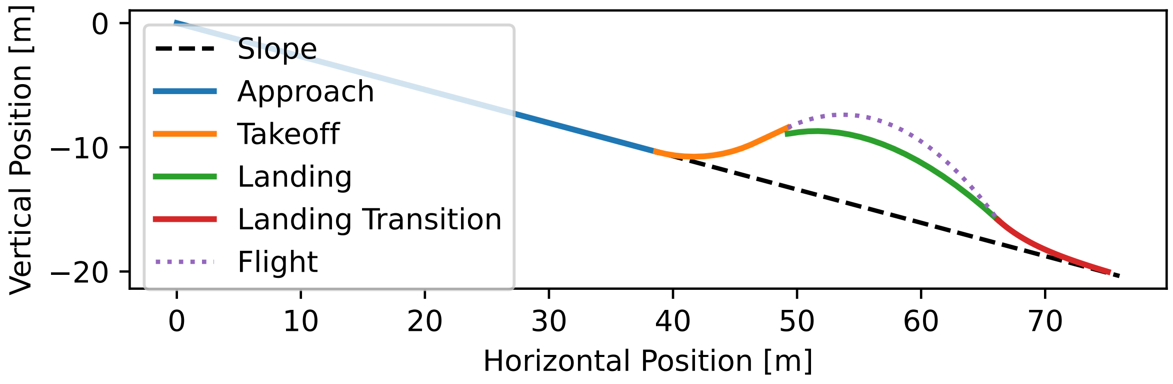
Release 1.5.0.dev0

Jason K. Moore, Bryn Cloud, Mont Hubbard

Feb 03, 2024

CONTENTS:

This is the documentation for "skijumpdesign: A Ski Jump Design and Analysis Tool for Equivalent Fall Height" based on the work presented in[2]. The software includes a library for two dimensional skiing simulations and a graphical web application for designing and analyzing basic ski jumps. The primary purpose of the software is to provide an open source, layperson-friendly web application for designing and assessing ski jumps for the purposes of minimizing equivalent fall height (EFH). Ski jumps that are designed with low equivalent fall heights will likely reduce injuries. See the references below for a more thorough discussion of the reasons. A current version of the web application can be accessed at http://www.skijumpdesign.info.



[2] Levy, Dean, Mont Hubbard, James A. McNeil, and Andrew Swedberg. "A Design Rationale for Safer Terrain Park Jumps That Limit Equivalent Fall Height." Sports Engineering 18, no. 4 (December 2015): 227–39. https://doi.org/10.1007/s12283-015-0182-6.

CHAPTER ONE

INSTALLATION

skijumpdesign can be installed using several tools. Below are recommended options, in order of the developers' preference.

1.1 conda

The library and web application can be installed into the root conda environment from the Conda Forge channel at anaconda.org. This requires installing either miniconda or Anaconda first. Once conda is available run:

```
$ conda install -c conda-forge skijumpdesign
```

The Anaconda Navigator graphical installer can also be used to accomplish the same result.

1.2 pip

The library and web application can be installed from PyPi using pip[1]:

```
$ pip install skijumpdesign
```

If you want to run the unit tests and/or build the documentation use:

```
$ pip install skijumpdesign[dev]
```

to also install the development dependencies.

1.3 setuptools

Download and unpack the source code to a local directory, e.g. `/path/to/skijumpdesign`.

Open a terminal. Navigate to the `skijumpdesign` directory:

```
$ cd /path/to/skijumpdesign
```

Install with[1]:

```
$ python setup.py install
```

[1] Note that you likely want to install into a user directory with pip/setuptools. See the pip and setuptools documentation on how to do this.

1.4 Optional dependencies

If pycvodes is installed it will be used to speed up the flight simulation and the landing surface calculation significantly. This library is not trivial to install on all operating systems, so you will need to refer its documentation for installation instructions. If you are using conda Linux or OSX, this package can be installed using conda with:

```
$ conda install -c conda-forge pycvodes
```

1.5 Development Installation

Clone the repository with git:

```
$ git clone https://gitlab.com/moorepants/skijumpdesign
```

Navigate to the cloned `skijumpdesign` repository:

```
$ cd skijumpdesign/
```

Setup the custom development conda environment named `skijumpdesign` to ensure it has all of the correct software dependencies. To create the environment type:

```
$ conda env create -f conda/skijumpdesign-lib-dev.yml
```

To activate the environment type[2]:

```
$ conda activate skijumpdesign-lib-dev
(skijumpdesign-lib-dev)$
```

Optionally, install in development mode using setuptools for use from any directory:

```
(skijumpdesign-lib-dev)$ python setup.py develop
```

There are several conda environment files provided in the source code that may be of use:

- `skijumpdesign-app.yml`: Installs the versions of the required dependencies to run the library and the web app pinned to specific versions for the app. These are the versions we use to run the official web app.
- `skijumpdesign-app-opt.yml`: Installs the versions of the required and optional dependencies to run the library and the web app pinned to specific versions for the app. These are the versions we use to run the official web app.
- `skijumpdesign-app-dev.yml`: Installs the versions of the required dependencies to run the library and the web app pinned to specific versions for the app plus tools for development. These are the versions we use to run the official web app.
- `skijumpdesign-app-opt-dev.yml`: Installs the versions of the required and optional dependencies to run the library and the web app pinned to specific versions for the app plus tools for development. These are the versions we use to run the official web app.
- `skijumpdesign-lib.yml`: Installs the latest version of the required dependencies to run the library and the web app.
- `skijumpdesign-lib-opt.yml`: Installs the latest version of the required and optional dependencies to run the library and the web app.

[2] This environment will also show up in the Anaconda Navigator program.

- `skijumpdesign-lib-dev.yml`: Installs the latest version of the required dependencies to run the library and the web app, test the code, and build the documentation.
- `skijumpdesign-lib-opt-dev.yml`: Installs the latest version of the required and optional dependencies to run the library and the web app, test the code, and build the documentation.

1.6 Render.com Installation

When installing into a Render web service, the application will make use of the `requirements.txt` file included in the source code which installs all of the dependencies needed to run the software on a live onrender.com instance. You need to set some environment variables for the Render app:

- `ONRENDER=true`: Lets the app know if it is running on Render.com.
- `GATRACKINGID`: Set the value as a string with your Google Analytics tracking id.

CHAPTER

TWO

RUNNING THE WEB APPLICATION

2.1 User

After *installing* skijumpdesign type:

```
$ skijumpdesign
```

to run the web application. This should launch the application on your computer's port 8050. You can view the application by visiting `http://localhost:8050` in your preferred web browser. `<CTRL + C>` will stop the server.

2.2 Developer

The `bin/skijumpdesign` entry point is not available unless you install the software. The following shows how to launch the app from the Python file.

2.2.1 In a terminal

Navigate to the `skijumpdesign` directory on your computer:

```
$ cd /path/to/skijumpdesign
```

Activate the custom Conda environment with:

```
$ conda activate skijumpdesign-lib-dev
```

Now run the application with:

```
(skijumpdesign-lib-dev)$ python -m skijumpdesign.app
```

You should see something like:

```
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Open your web browser and enter the displayed URL to interact with the web app. Type `<CTRL>+C` in the terminal to shutdown the web server.

2.2.2 In Spyder

Open Anaconda Navigator, switch to the `skijumpdesign-lib-dev` environment, and then launch Spyder. Set the working directory to the `/path/to/skijumpdesign` directory. In the Spyder IPython console execute:

```
In [1]: run skijumpdesign/app.py
```

If successful, you will see something like:

```
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Open your web browser and enter the displayed URL to interact with the web app.

To shutdown the web app, close the tab in your web browser. Go back to Spyder and execute `<CTRL>+C` to shutdown the web server.

CHAPTER

THREE

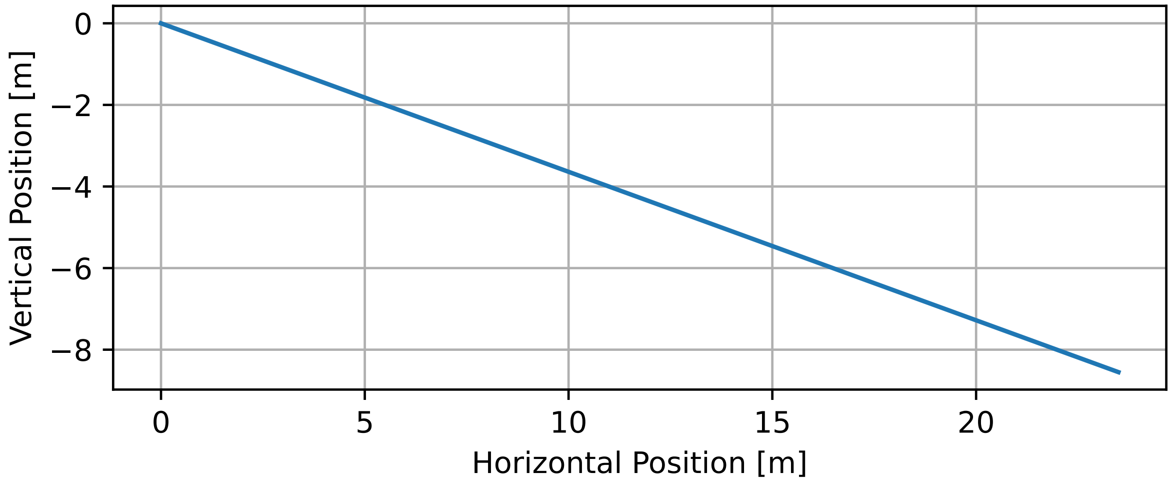
EXAMPLE: DESIGN EFH JUMP

The following page describes how to construct an example constant equivalent fall height ski jump landing surface using the `skijumpdesign` *API*. Make sure to *install* the library first.

3.1 Approach

Start by creating a 25 meter length of an approach path (also called the in-run) which is flat and has a downward slope angle of 20 degrees. The resulting surface can be visualized with the `plot()` method.

```
from skijumpdesign import FlatSurface

approach_ang = -np.deg2rad(20)  # radians
approach_len = 25.0  # meters

approach = FlatSurface(approach_ang, approach_len)
approach.plot()
```



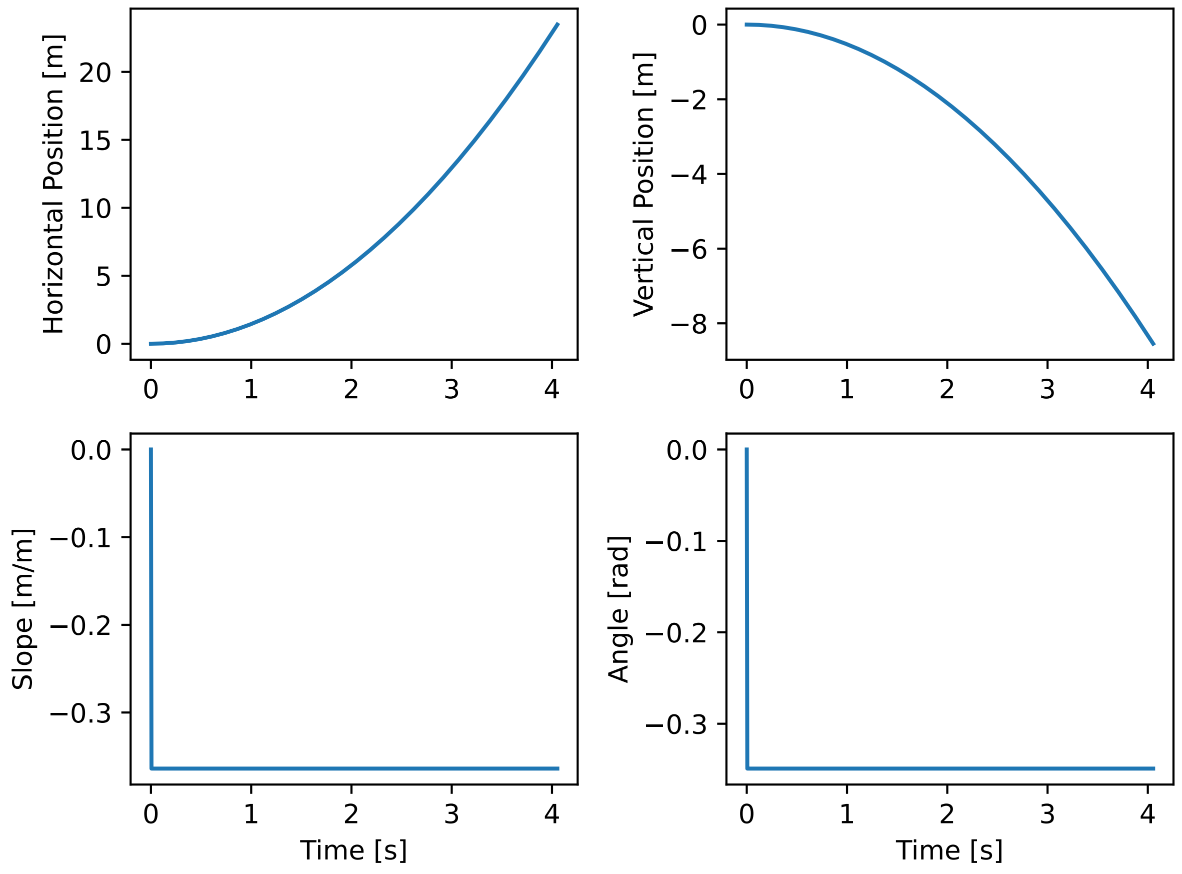
Now that a surface has been created, a skier can be created. The skier can "ski" along the approach surface using the `slide_on()` method which generates a skiing simulation trajectory.

```
from skijumpdesign import Skier

```

(continues on next page)

(continued from previous page)

```
skier = Skier()

approach_traj = skier.slide_on(approach)

approach_traj.plot_time_series()
```

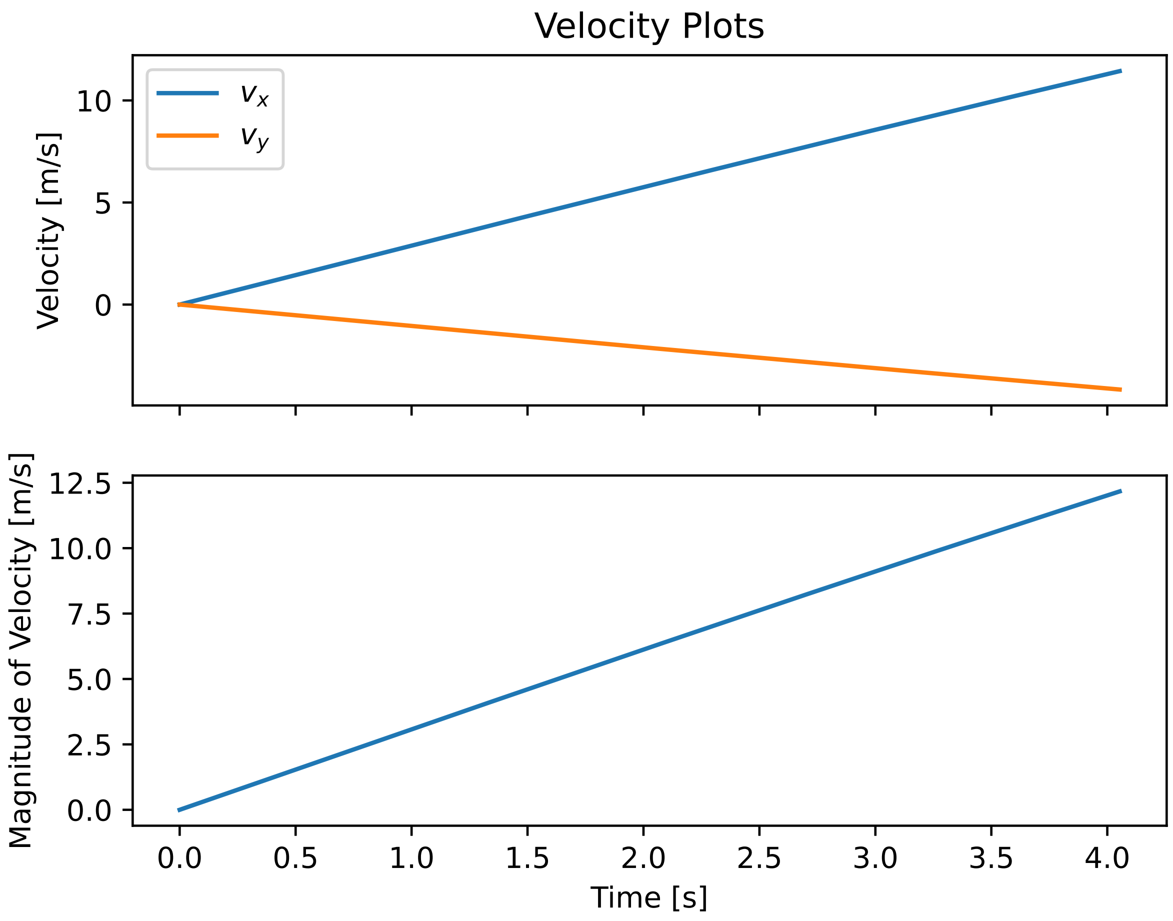


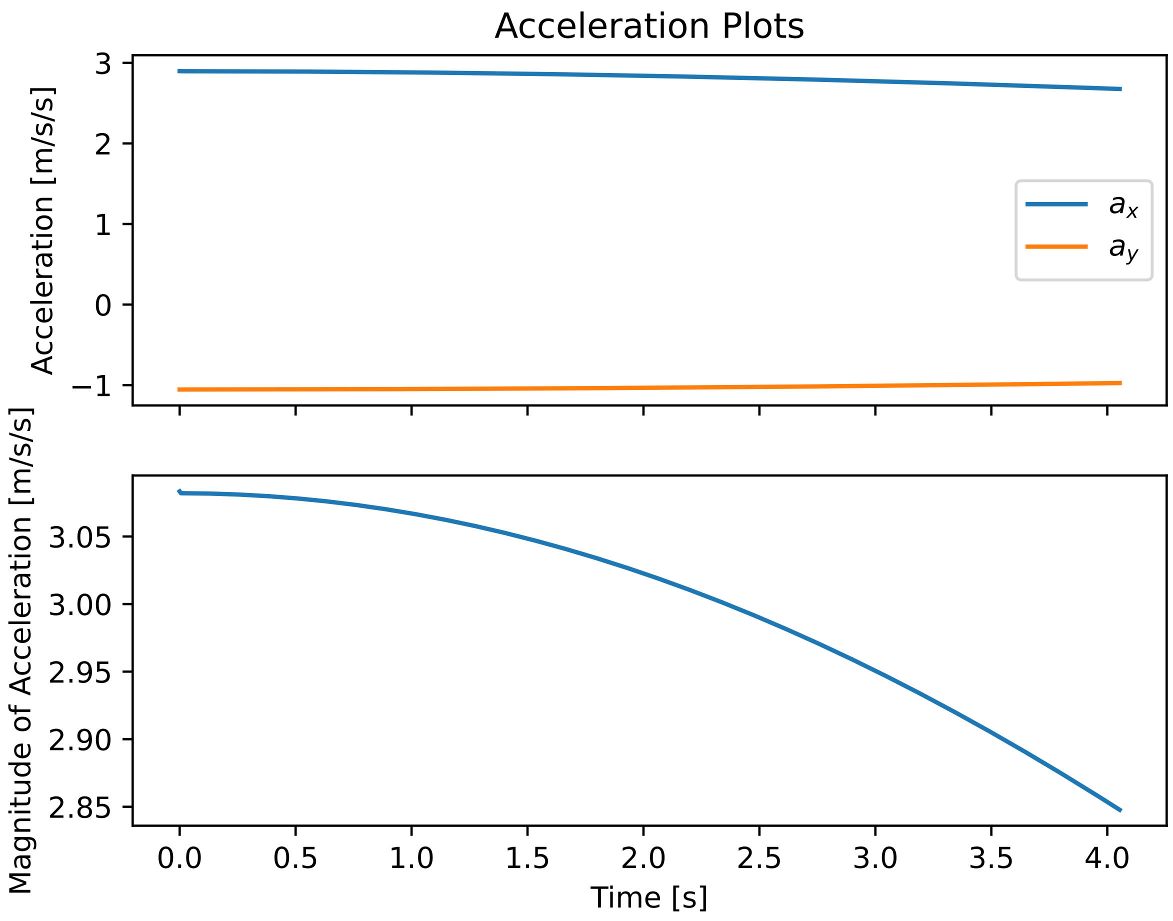
3.2 Approach-Takeoff Transition

The approach-takeoff transition is constructed with a clothoid-circle-clothoid-flat surface to transition from the parent slope angle to the desired takeoff angle, in this case 15 degrees.

```
from skijumpdesign import TakeoffSurface

takeoff_entry_speed = skier.end_speed_on(approach)

takeoff_ang = np.deg2rad(15)

takeoff = TakeoffSurface(skier, approach_ang, takeoff_ang,
                         takeoff_entry_speed, init_pos=approach.end)
```

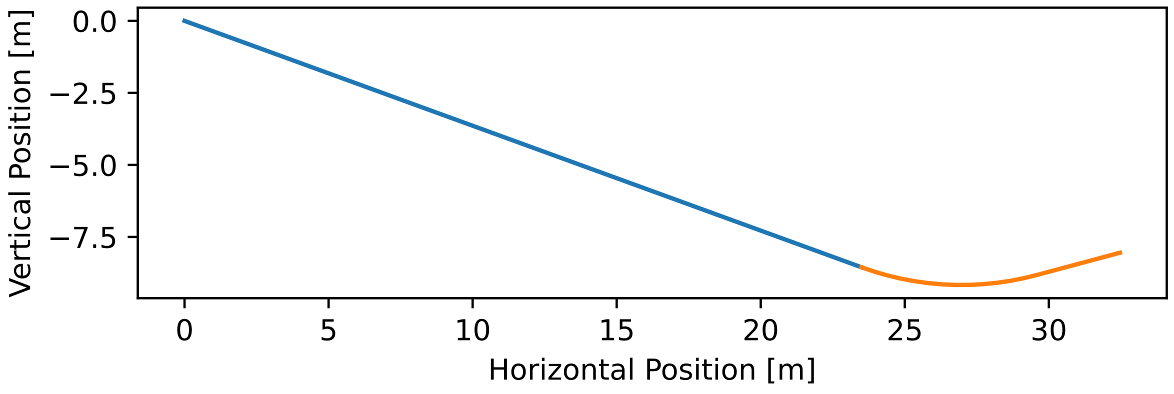
(continues on next page)


Velocity Plots
v_x
v_y
Velocity [m/s]
10
5
0
Magnitude of Velocity [m/s]
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time [s]


Acceleration Plots
Acceleration [m/s/s]
3
2
1
0
−1
a_x
a_y
Magnitude of Acceleration [m/s/s]
3.05
3.00
2.95
2.90
2.85
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time [s]

(continued from previous page)

```
ax = approach.plot()
takeoff.plot(ax=ax)
```



The trajectory of the skier on the takeoff can be examined also.

```
takeoff_traj = skier.slide_on(takeoff, takeoff_entry_speed)

takeoff_traj.plot_time_series()
```

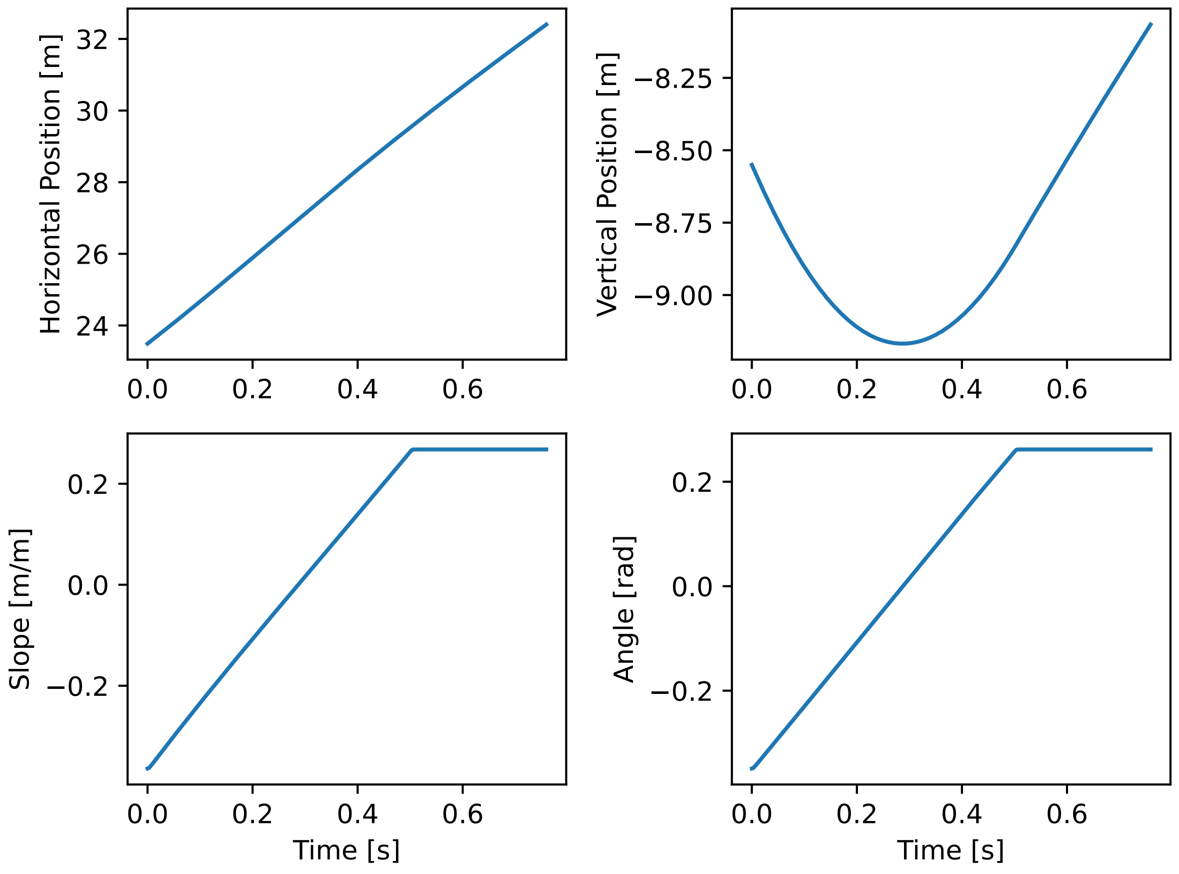
3.3 Flight

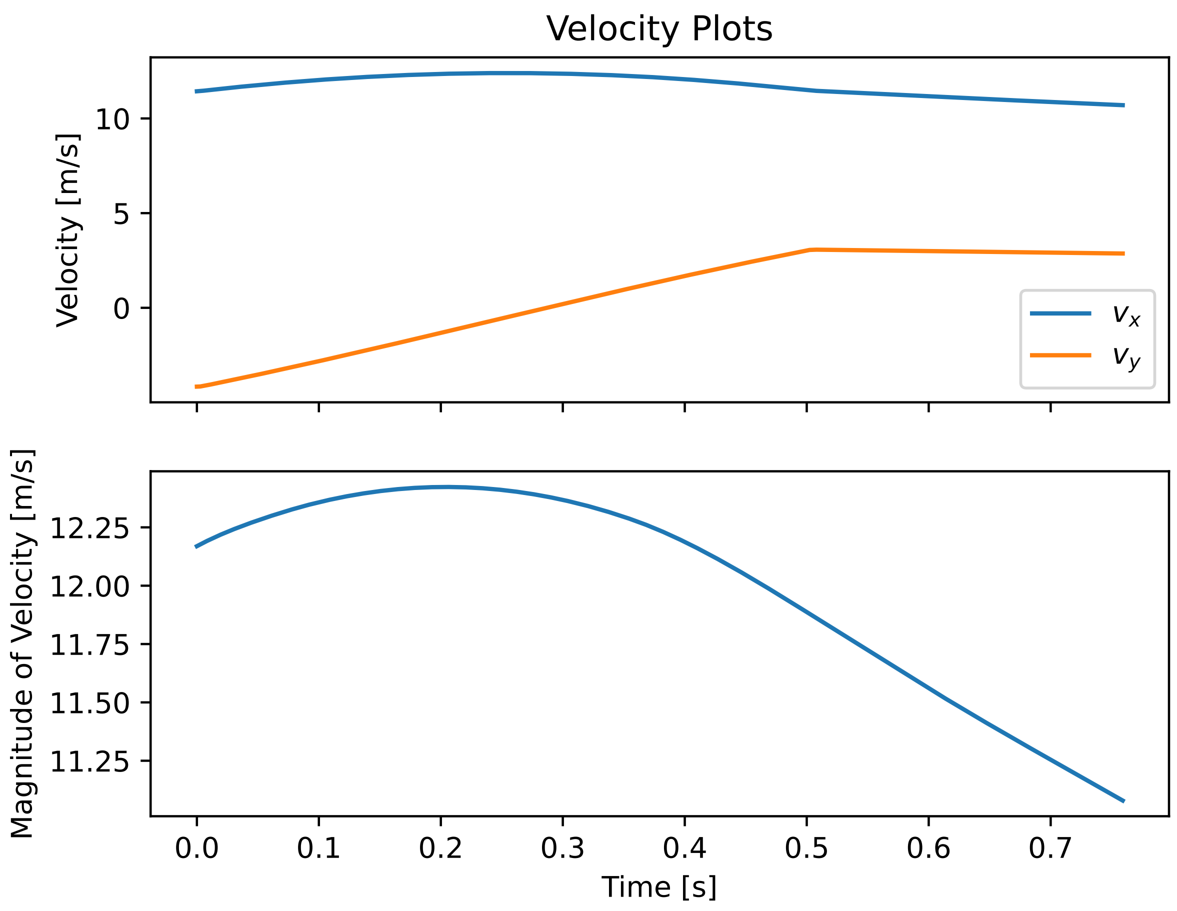
Once the skier leaves the takeoff ramp at the maximum (design) speed they will be in flight. The *fly_to()* method can be used to simulate this longest flight trajectory.

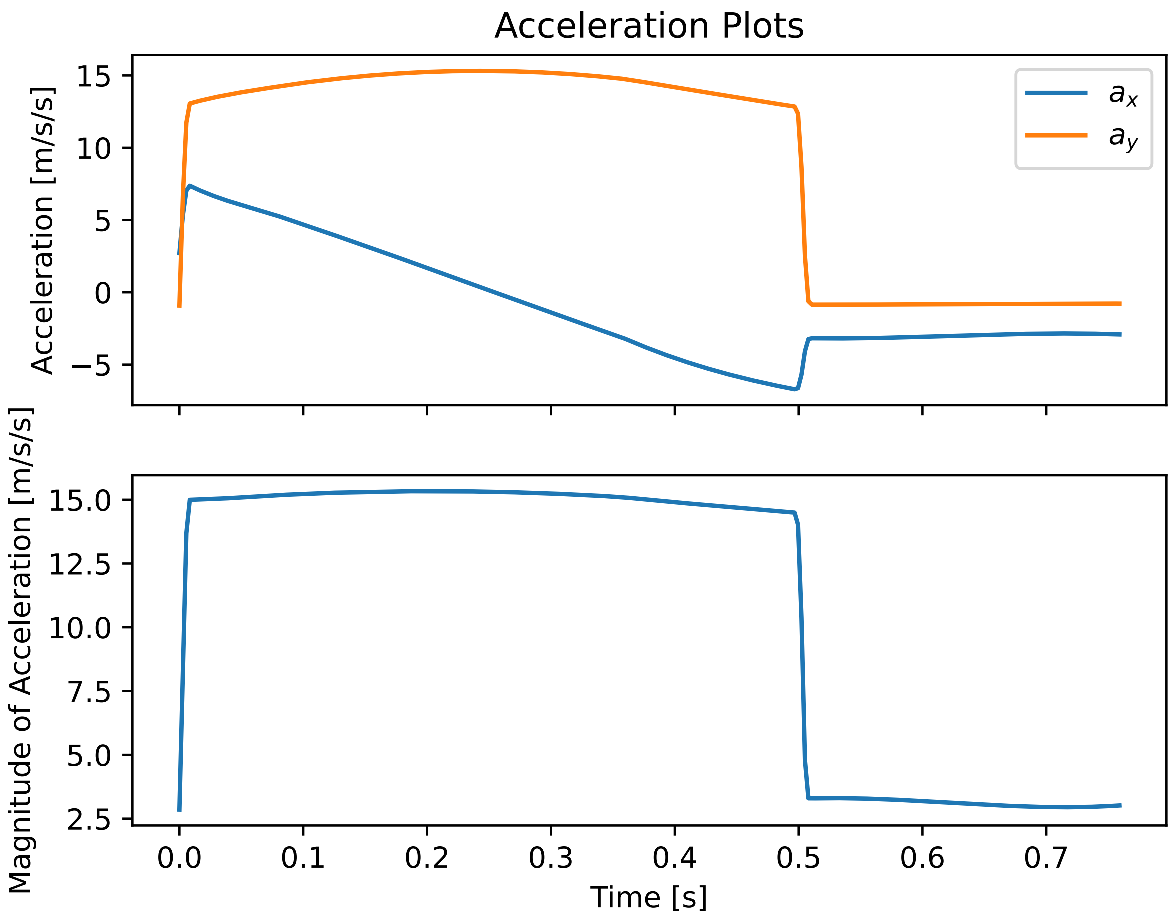
```
takeoff_vel = skier.end_vel_on(takeoff, init_speed=takeoff_entry_speed)

flight = skier.fly_to(approach, init_pos=takeoff.end,
                      init_vel=takeoff_vel)

flight.plot_time_series()
```

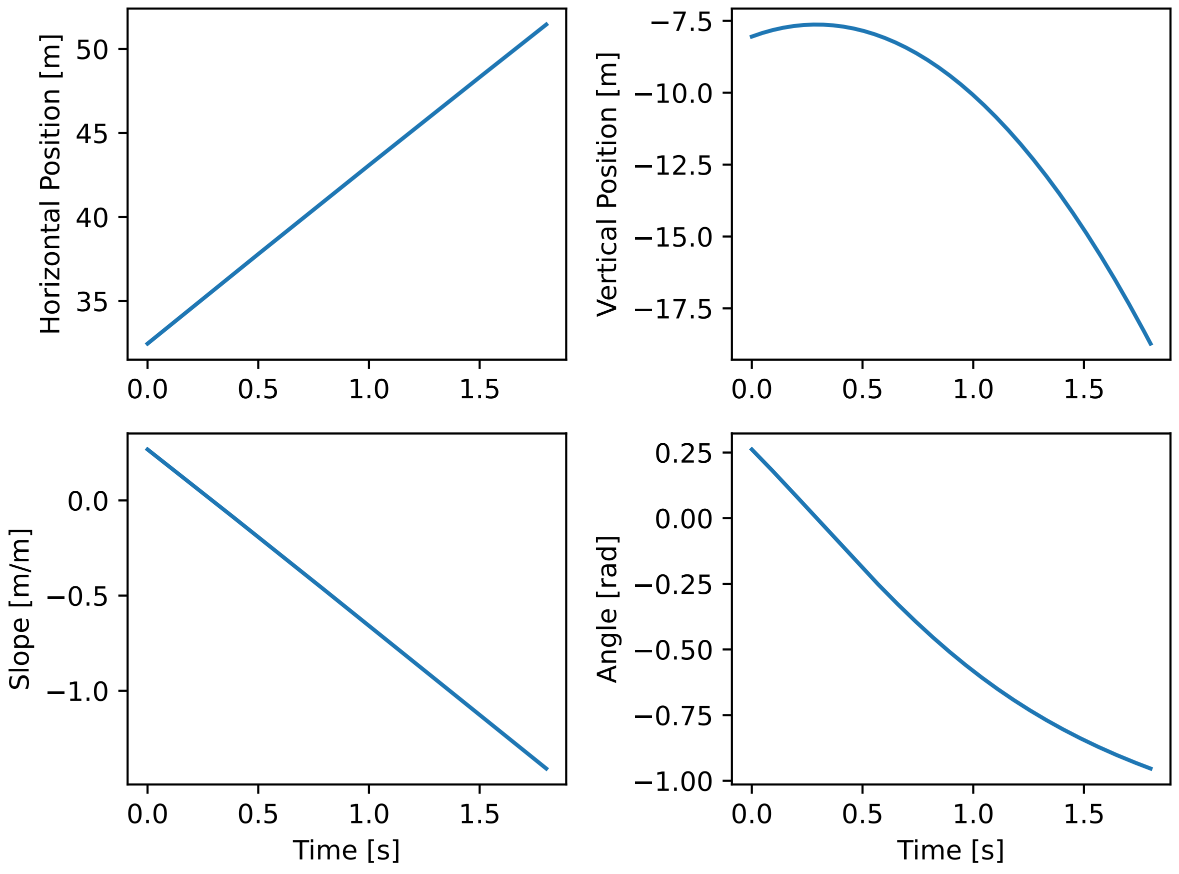
The design speed flight trajectory can be plotted as an extension of the approach and takeoff surfaces.

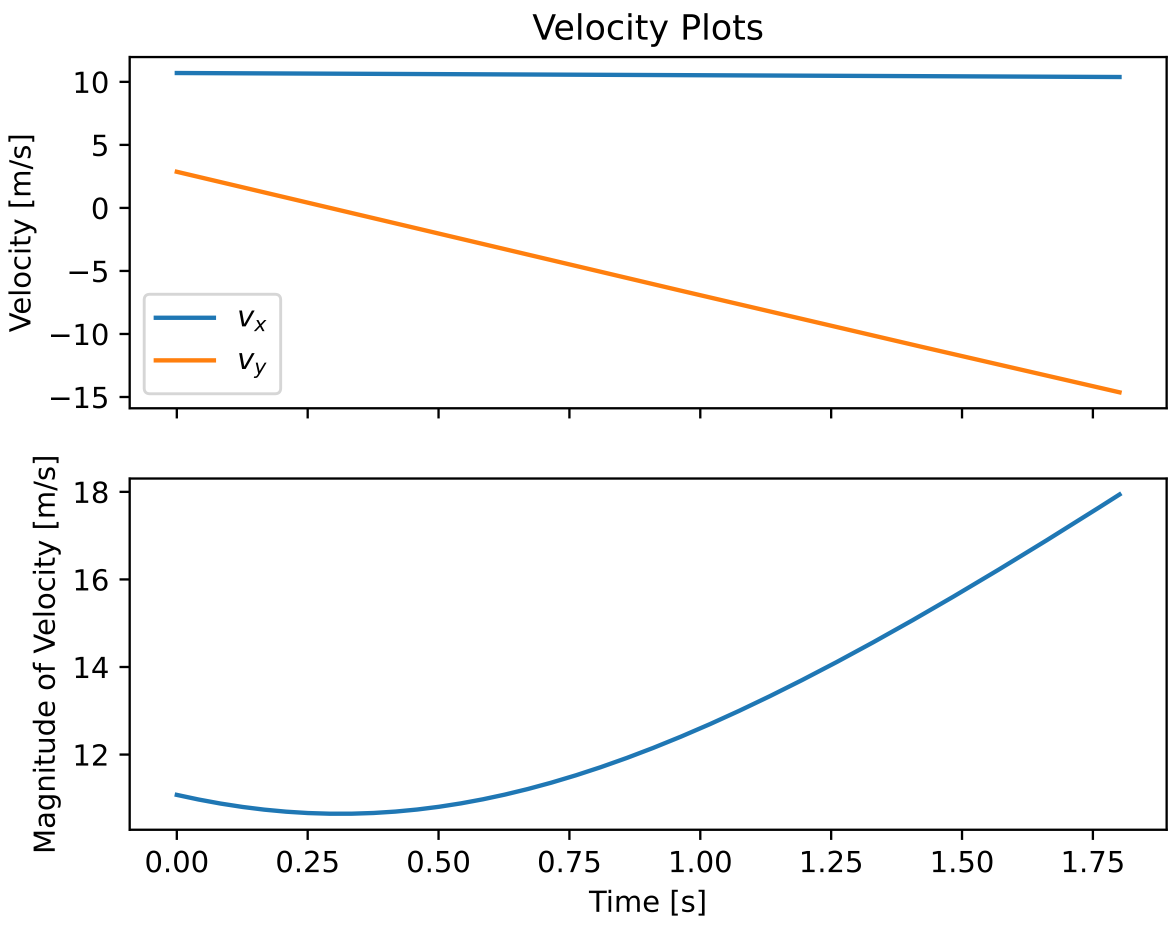
```
ax = approach.plot()
ax = takeoff.plot(ax=ax)
flight.plot(ax=ax, color='#9467bd')
```

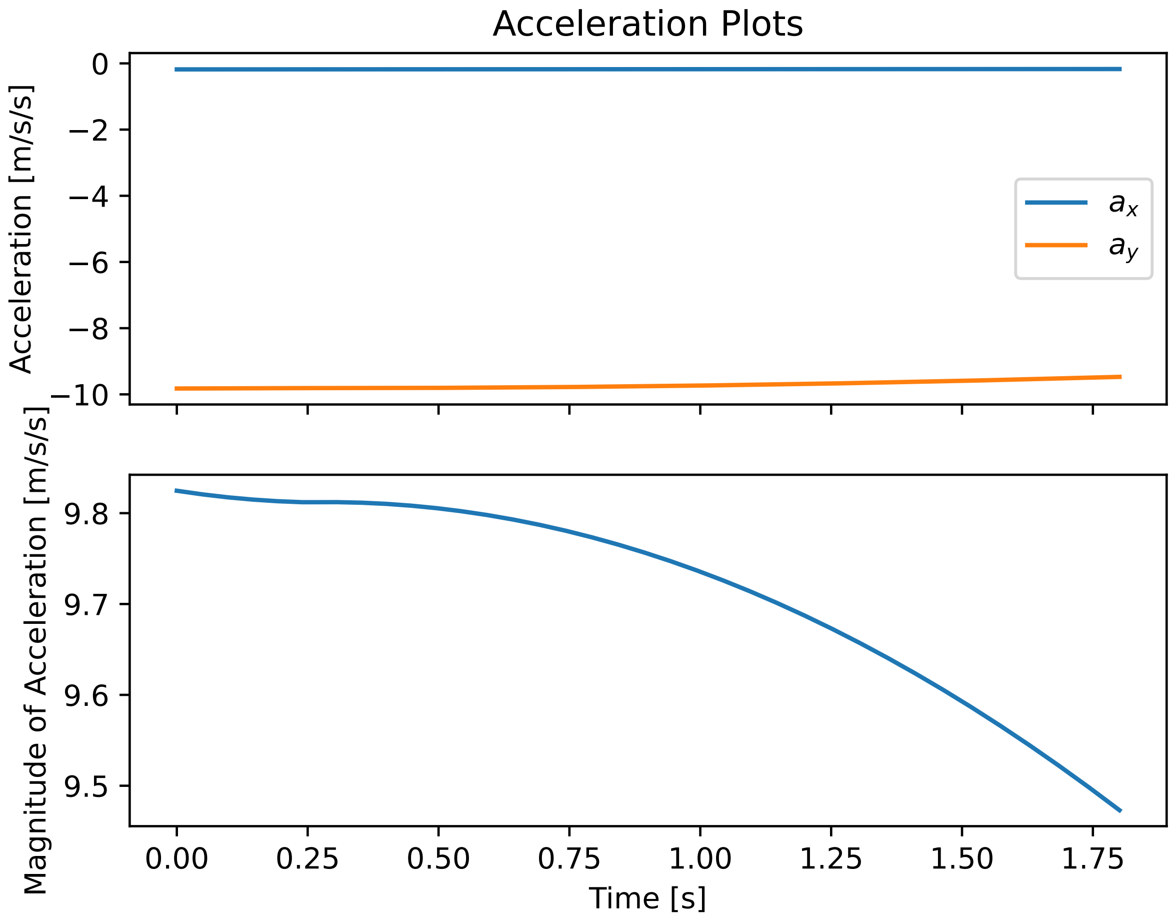

Horizontal Position [m]
Vertical Position [m]
Slope [m/m]
Angle [rad]
Time [s]
Time [s]

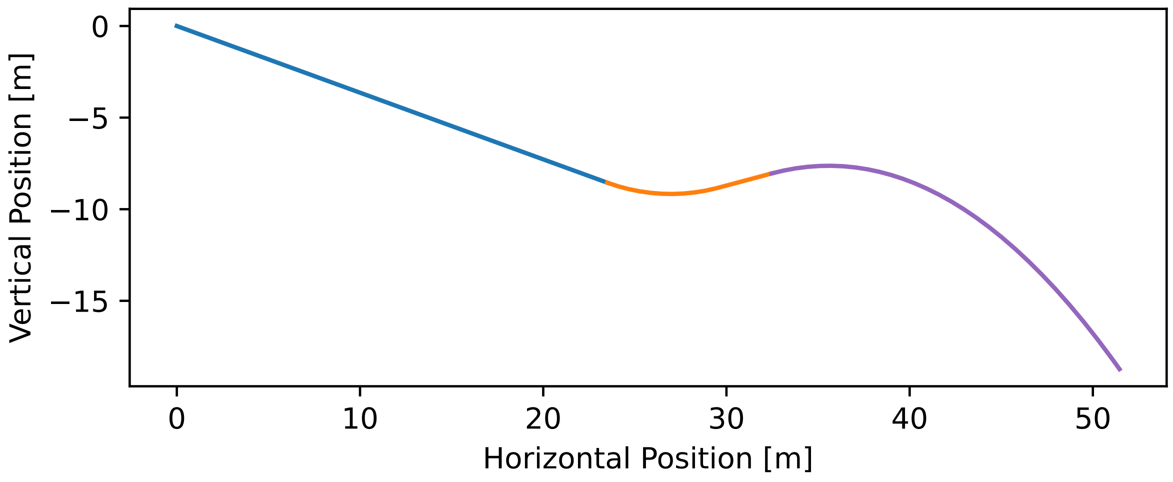

Velocity Plots
Velocity [m/s]
10
5
0
v_x
v_y
Magnitude of Velocity [m/s]
12.25
12.00
11.75
11.50
11.25
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time [s]


Acceleration Plots
Acceleration [m/s/s]
a_x
a_y
15
10
5
0
−5
Magnitude of Acceleration [m/s/s]
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time [s]


Horizontal Position [m]
Vertical Position [m]
Slope [m/m]
Angle [rad]
Time [s]
Time [s]


Velocity Plots
Velocity [m/s]
v_x
v_y
Magnitude of Velocity [m/s]
Time [s]
10
5
0
−5
−10
−15
18
16
14
12
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

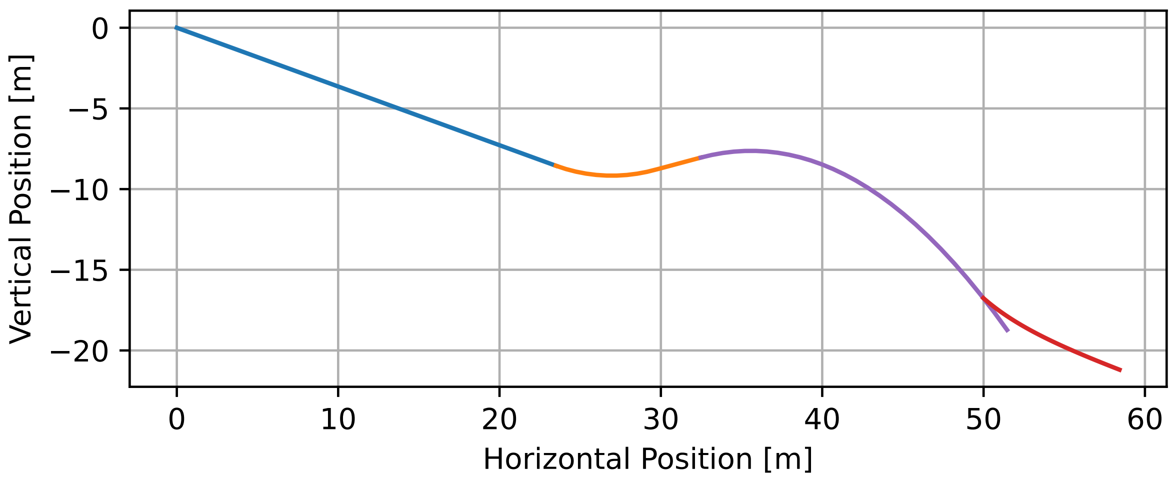

Acceleration Plots
Acceleration [m/s/s]
a_x
a_y
Magnitude of Acceleration [m/s/s]
Time [s]


Vertical Position [m]
Horizontal Position [m]

3.4 Landing Transition

The final part of this step is to determine the landing transition curve (shown in red below) which connects the optimum (cheapest) constant EFH landing surface to the parent slope. There are an infinite number of landing surfaces that satisfy the EFH differential equation and provide the desired equivalent fall height. The algorithm selects the one of these that is closest to the parent slope, and hence is least expensive to build (in terms of snow volume), but which still is able to transition back to the parent slope with slope continuity and simultaneously is constrained to experience limited normal acceleration.

```
from skijumpdesign import LandingTransitionSurface

fall_height = 0.5

landing_trans = LandingTransitionSurface(approach,
    flight, fall_height, skier.tolerable_landing_acc)

ax = approach.plot()
ax = takeoff.plot(ax=ax)
ax = flight.plot(ax=ax, color='#9467bd')
landing_trans.plot(ax=ax, color='#d62728')
```



3.5 Constant EFH Landing

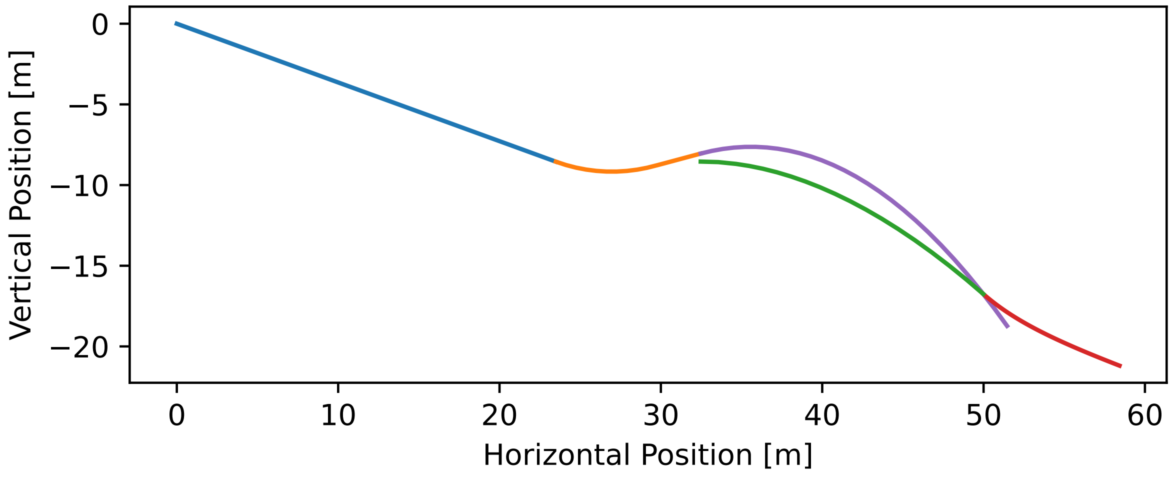
Finally, the cheapest equivalent fall height landing surface (shown in green below) can be calculated. This surface is continuous in slope with the landing transition surface at the impact point. It accommodates all takeoff speeds below the maximum takeoff (design) speed above.

```
from skijumpdesign import LandingSurface

slope = FlatSurface(approach_ang, np.sqrt(landing_trans.end[0]**2 +
                                          landing_trans.end[1]**2) + 1.0)


landing = LandingSurface(skier, takeoff.end, takeoff_ang,
```

(continues on next page)

(continued from previous page)

```
                         landing_trans.start, fall_height,
                         surf=slope)

ax = approach.plot()
ax = takeoff.plot(ax=ax)
ax = flight.plot(ax=ax, color='#9467bd')
ax = landing_trans.plot(ax=ax, color='#d62728')
landing.plot(ax=ax, color='#2ca02c')
```



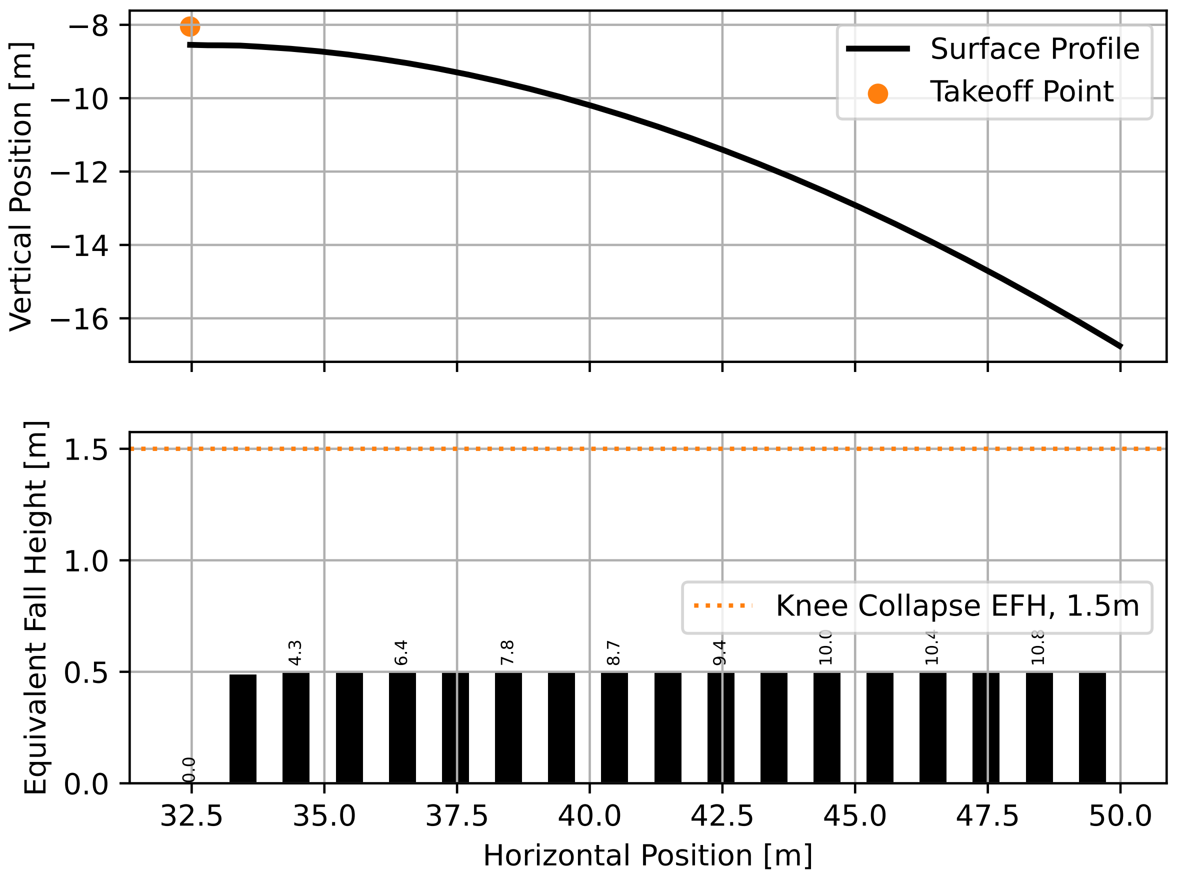
The design calculates a landing surface shape that produces a constant equivalent fall height. This can be verified using the `calculate_efh()` function that calculates the equivalent fall height for the surface that was produced. See the *analyze jump* page to learn more about this function.

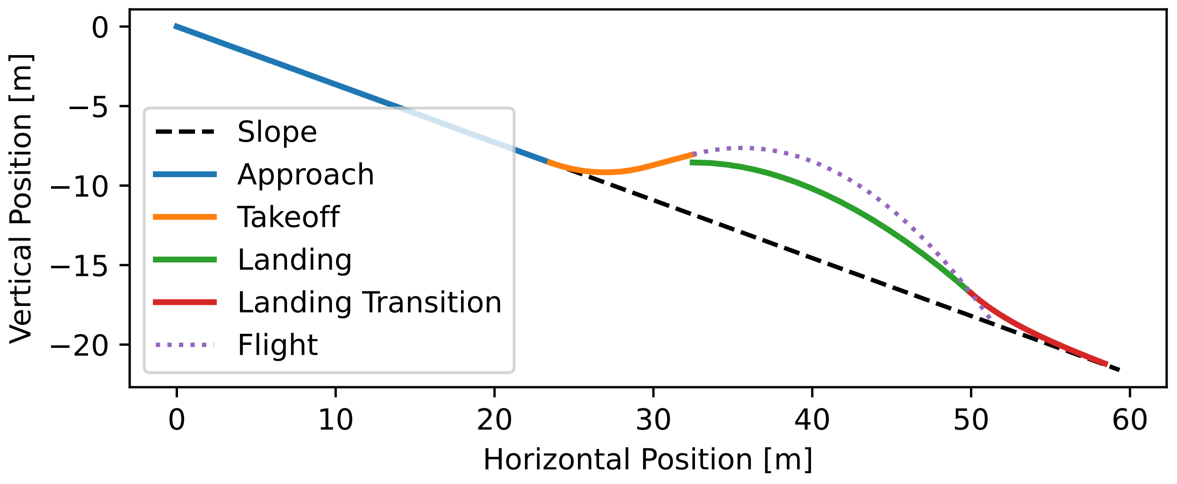
```
from skijumpdesign.functions import plot_efh

dist, efh, speeds = landing.calculate_efh(takeoff_ang, takeoff.end,
                                          skier, increment=1.0)
plot_efh(landing, np.rad2deg(takeoff_ang), takeoff.end, increment=1.0)
```

3.6 Entire Jump

There is also convenience function for plotting the jump:

```
from skijumpdesign import plot_jump

plot_jump(slope, approach, takeoff, landing, landing_trans, flight)
```


Vertical Position [m]
Surface Profile
Takeoff Point
Equivalent Fall Height [m]
Knee Collapse EFH, 1.5m
0.0
4.3
6.4
7.8
8.7
9.4
10.0
10.4
10.8
Horizontal Position [m]


Vertical Position [m]
Slope
Approach
Takeoff
Landing
Landing Transition
Flight
Horizontal Position [m]

CHAPTER

FOUR

EXAMPLE: ANALYZE JUMP EFH

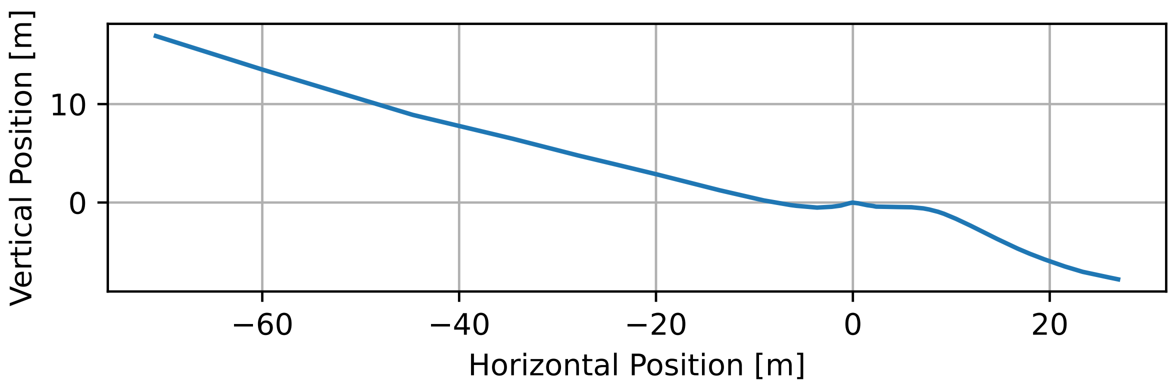
The following page describes how to analyze an existing or hypothetical ski jump landing surface by calculating it's equivalent fall height using the `skijumpdesign` *API*. Make sure to *install* the library first.

4.1 Load Data

Start by loading data of a ski jump surface's horizontal and vertical coordinates measured in meters. The resulting surface can be visualized with the `plot()` method. Create a tuple for the takeoff point coordinates, and a variable for the takeoff angle. Data for this example is taken from a jump measured with a level and tape measure and translated into horizontal (x) and vertical (y) components. The sample surface below was collected from a real slope and jump.

```
from skijumpdesign import Surface
import numpy as np

takeoff_ang = 10  # degrees
takeoff_point = (0, 0)  # meters

x_ft = [-232.3,-203.7,-175.0,-146.3,-117.0,-107.4,-97.7,-88.0,-78.2,
        -68.5,-58.8,-49.1,-39.4,-34.5,-29.7,-24.8,-19.8,-17.8,-15.8,
        -13.8,-11.8,-9.8,-7.8,-5.9,-3.9,-2.0,0.0,0.0,0.0,2.0,3.9,5.9,
        7.9,9.9,11.9,13.9,15.9,17.9,19.9,21.9,23.9,25.8,27.8,29.7,
        31.5,33.4,35.2, 37.0,38.8,43.3,47.8,52.3,56.8,61.5,66.2,70.9,
        75.7,80.6,85.5,88.4,88.4] # feet

y_ft = [55.5,46.4,37.7,29.1,22.2,19.7,17.2,14.8,12.5,10.2,7.7,5.2,
        2.9,1.8,0.7,-0.2,-1.0,-1.2,-1.4,-1.6,-1.7,-1.6,-1.5,-1.3,
        -1.0,-0.4,0.0,0.0,0.0,-0.3,-0.8,-1.0,-1.4,-1.4,-1.5,-1.5,-1.5,
        -1.5,-1.6,-1.8,-2.0,-2.4,-2.9,-3.5,-4.2,-5.0,-5.8,-6.7,-7.5,
        -9.8,-12.0,-14.2,-16.2,-18.1,-19.8,-21.4,-22.9,-24.0,-25.0,
        -25.6,-25.6] # feet

x = np.asarray(x_ft)*0.3048 # convert to meters
y = np.asarray(y_ft)*0.3048 # convert to meters

measured_surf = Surface(x, y)

measured_surf.plot()
```

Now that a surface has been created, a skier can be created. The skier can "ski" along the surface by extracting the data in the array before takeoff and using the `slide_on()` method which generates a skiing simulation trajectory.



```
from skijumpdesign import Skier

x_beforetakeoff = x[x<=takeoff_point[0]]
y_beforetakeoff = y[x<=takeoff_point[0]]

before_takeoff = Surface(x_beforetakeoff, y_beforetakeoff)

skier = Skier()

beforetakeoff_traj = skier.slide_on(before_takeoff)

beforetakeoff_traj.plot_time_series()
```

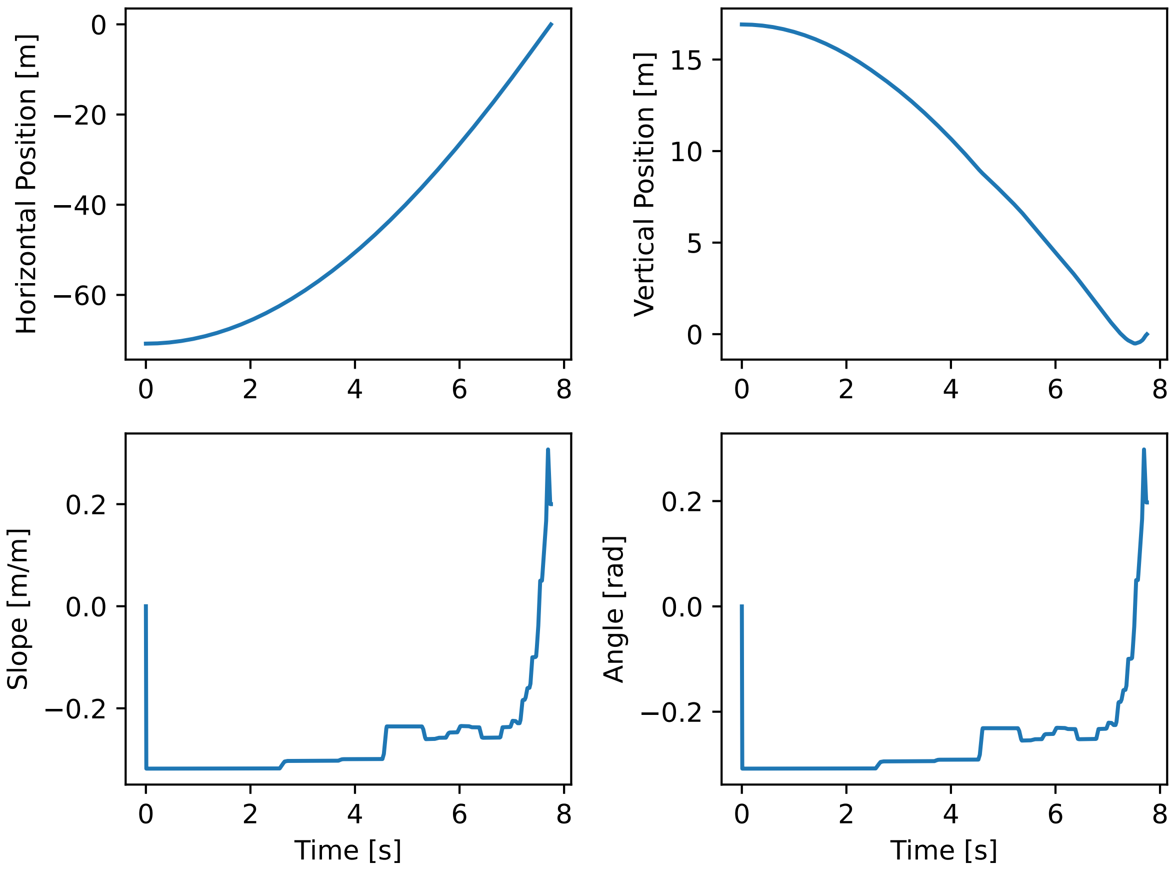
4.2 Flight

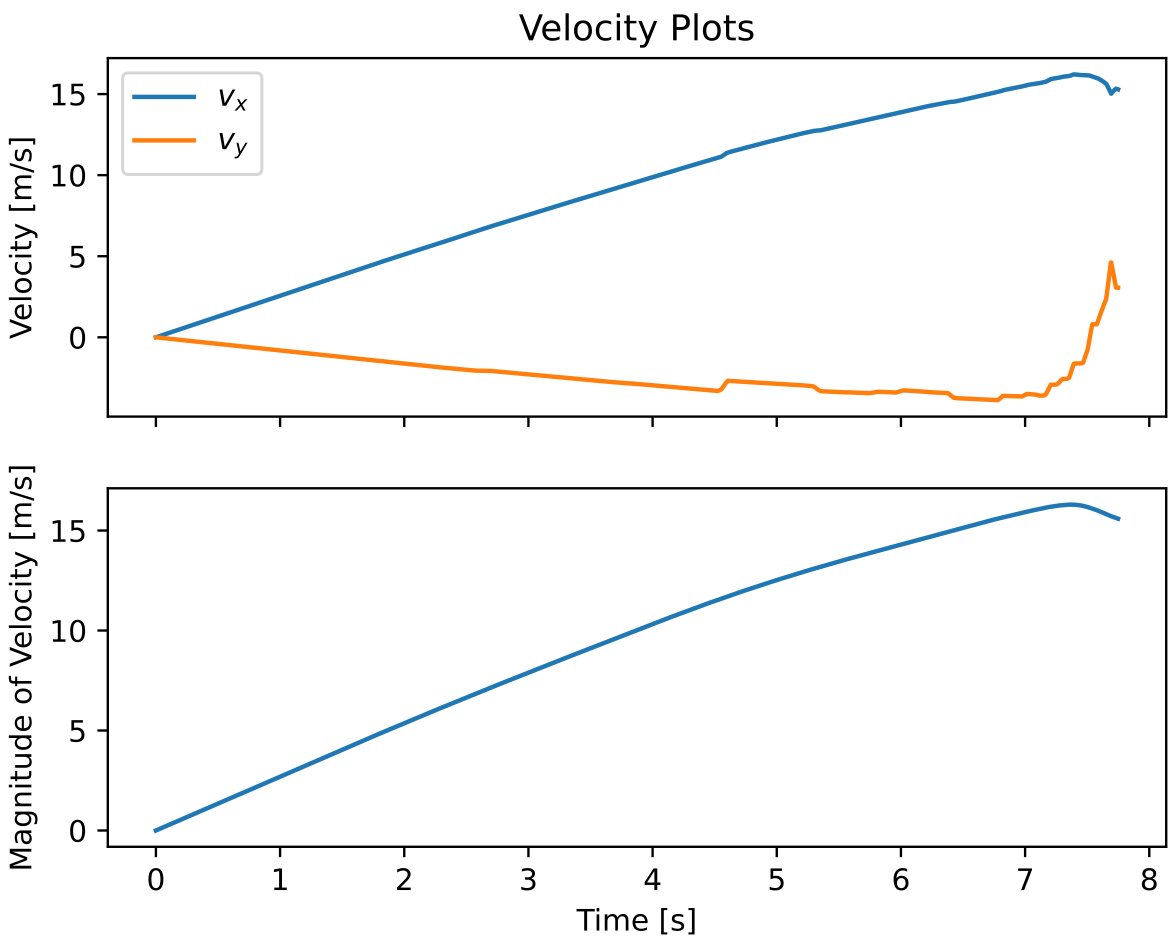
Once the skier leaves the takeoff ramp at the maximum (design) speed they will be in flight. The *fly_to()* method can be used to simulate this longest flight trajectory.

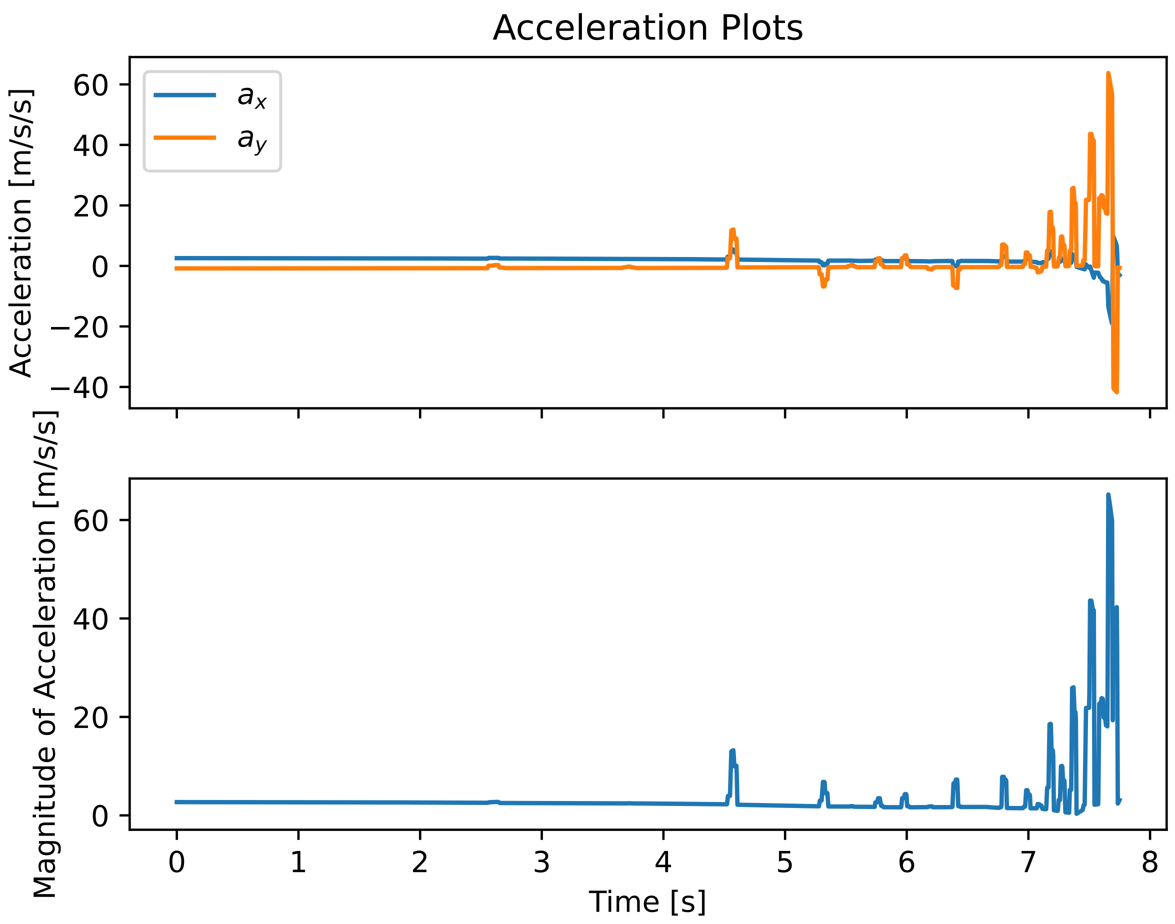
```
takeoff_vel = skier.end_vel_on(before_takeoff)

flight = skier.fly_to(measured_surf, init_pos=before_takeoff.end,
                      init_vel=takeoff_vel)

flight.plot_time_series()
```

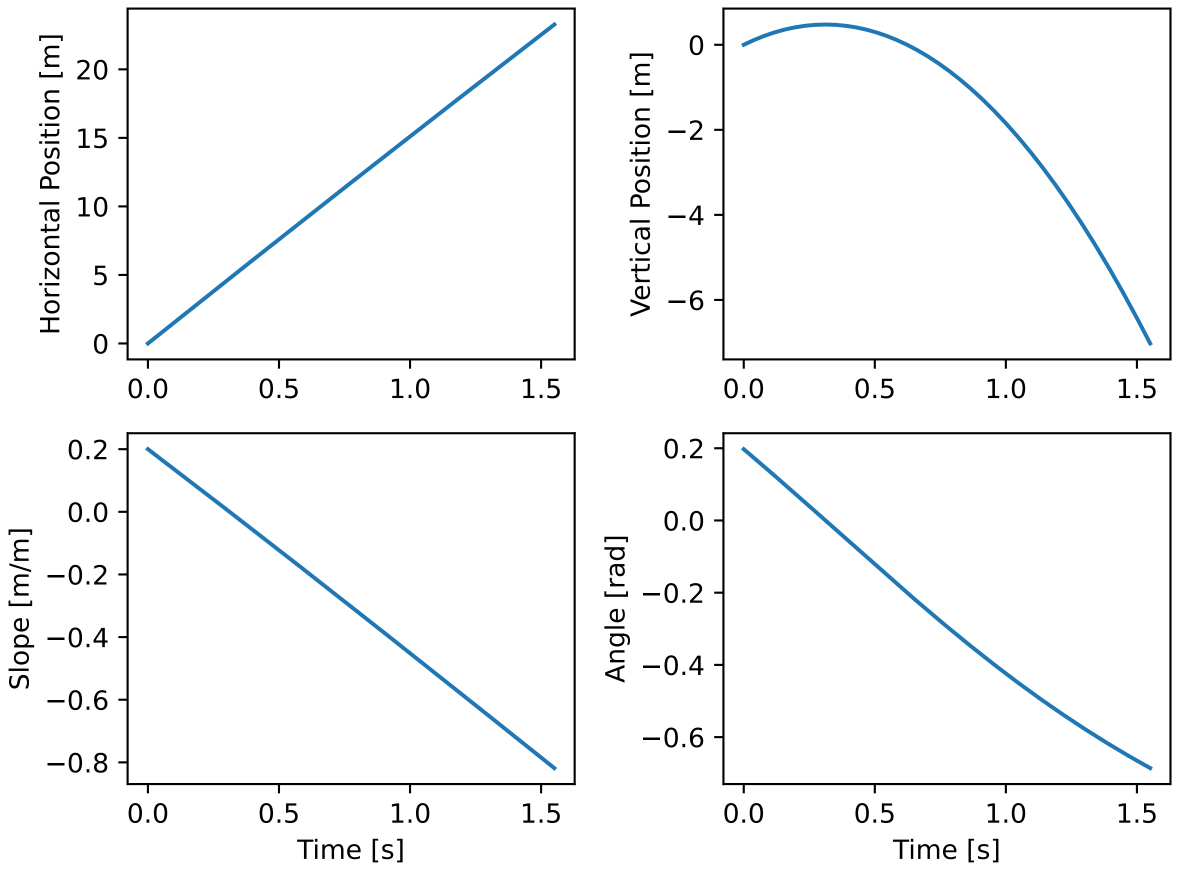
The design speed flight trajectory can be plotted in addition to the surface.

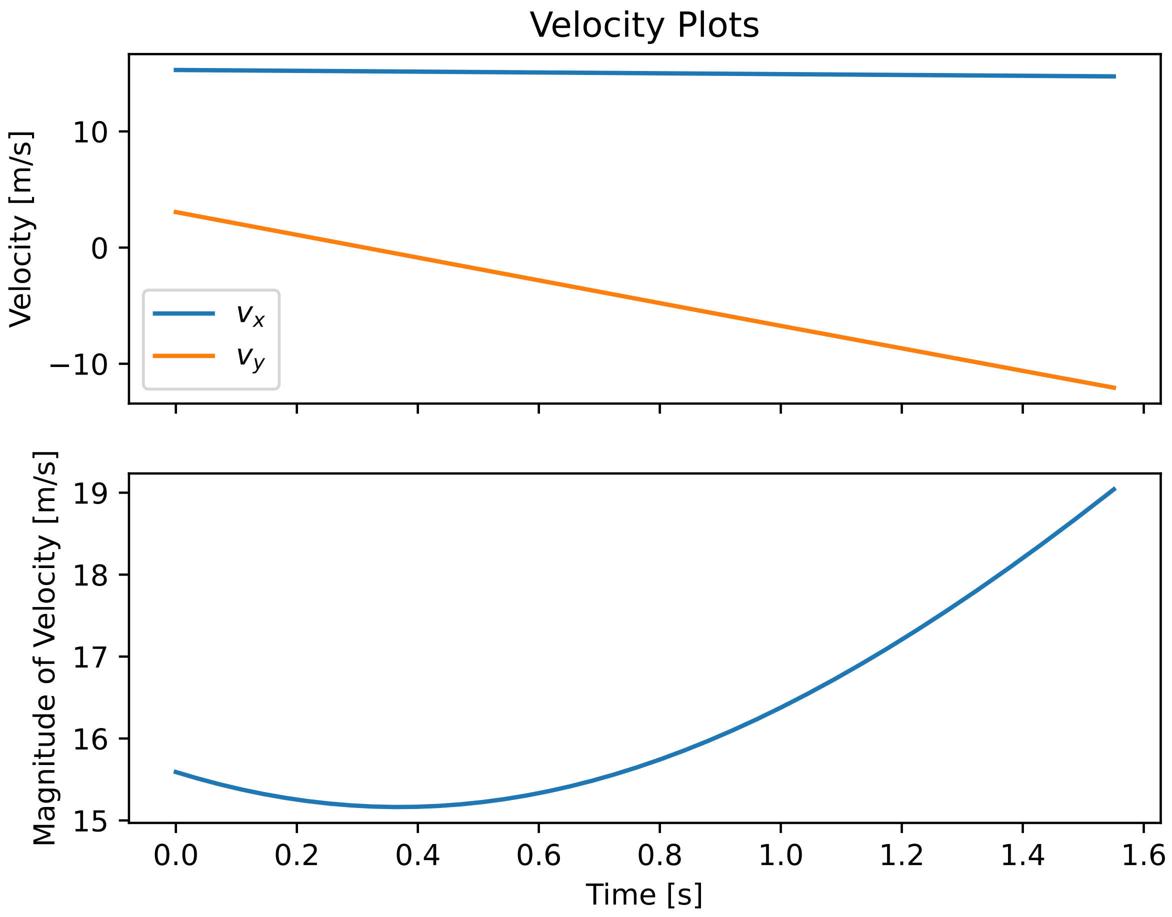
```
ax = measured_surf.plot()
flight.plot(ax=ax, color='#9467bd')
```

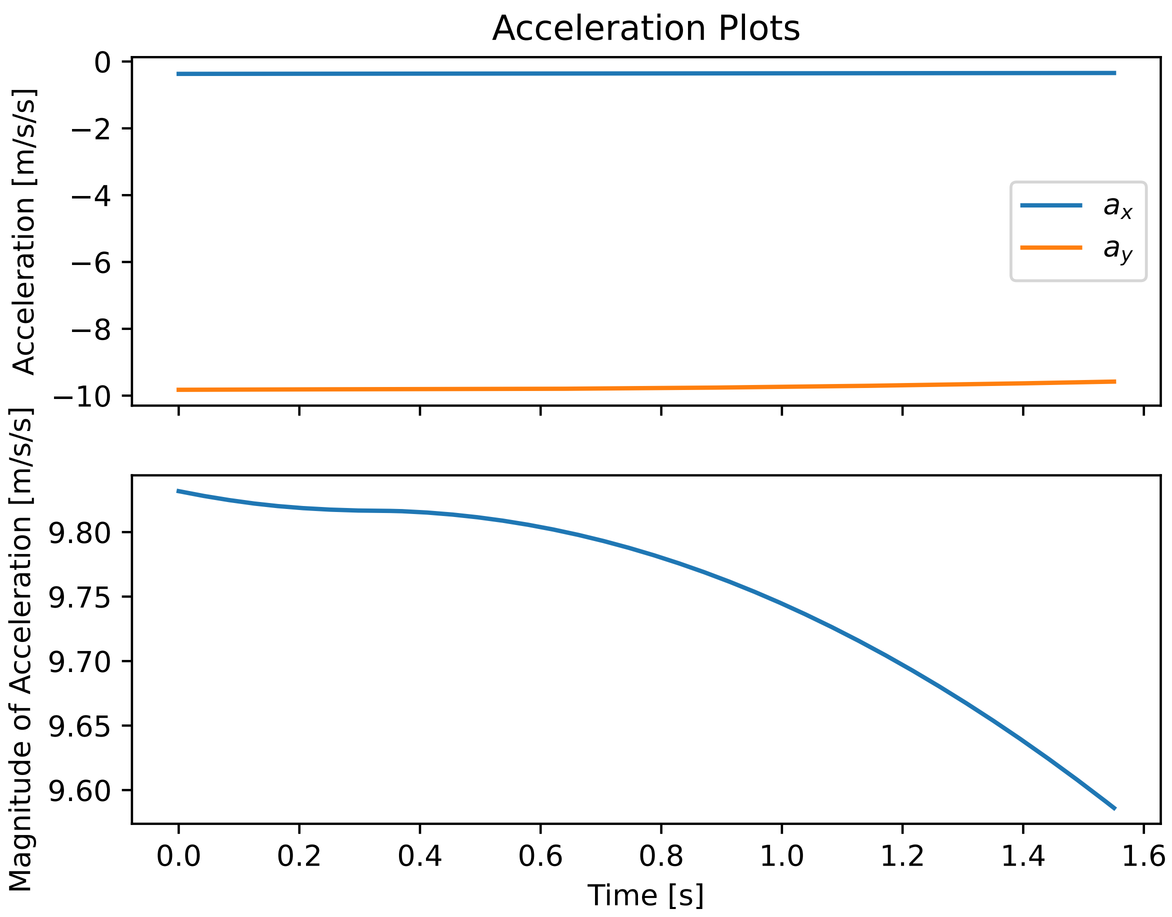

Horizontal Position [m]
Vertical Position [m]
Slope [m/m]
Angle [rad]
Time [s]
Time [s]

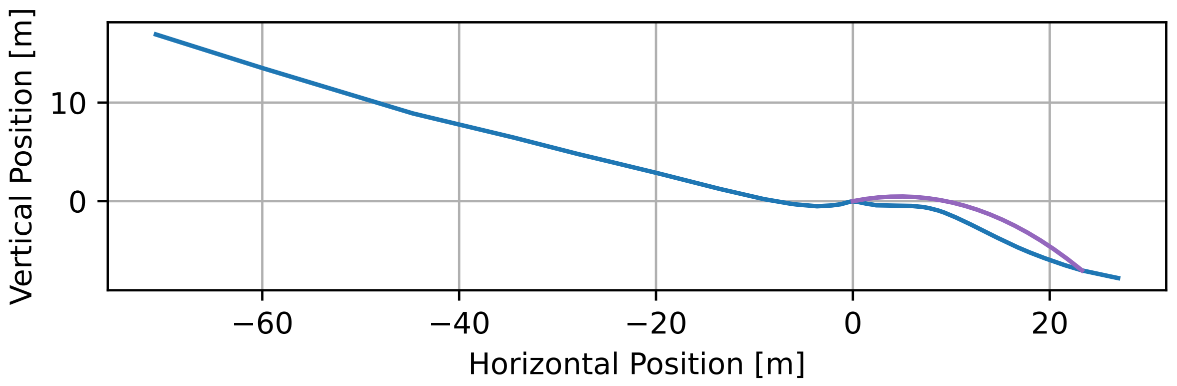

Velocity Plots
v_x
v_y
Velocity [m/s]
Magnitude of Velocity [m/s]
Time [s]
0
5
10
15
0
1
2
3
4
5
6
7
8


Acceleration Plots
a_x
a_y
Acceleration [m/s/s]
Magnitude of Acceleration [m/s/s]
Time [s]
60
40
20
0
−20
−40
0
1
2
3
4
5
6
7
8


Horizontal Position [m]
Vertical Position [m]
Slope [m/m]
Angle [rad]
Time [s]
Time [s]


Velocity Plots
Velocity [m/s]
10
0
−10
v_x
v_y
Magnitude of Velocity [m/s]
19
18
17
16
15
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time [s]


Acceleration Plots
Acceleration [m/s/s]
0
−2
−4
−6
−8
−10
a_x
a_y
Magnitude of Acceleration [m/s/s]
9.80
9.75
9.70
9.65
9.60
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time [s]


Vertical Position [m]
10
0
−60
−40
−20
0
20
Horizontal Position [m]

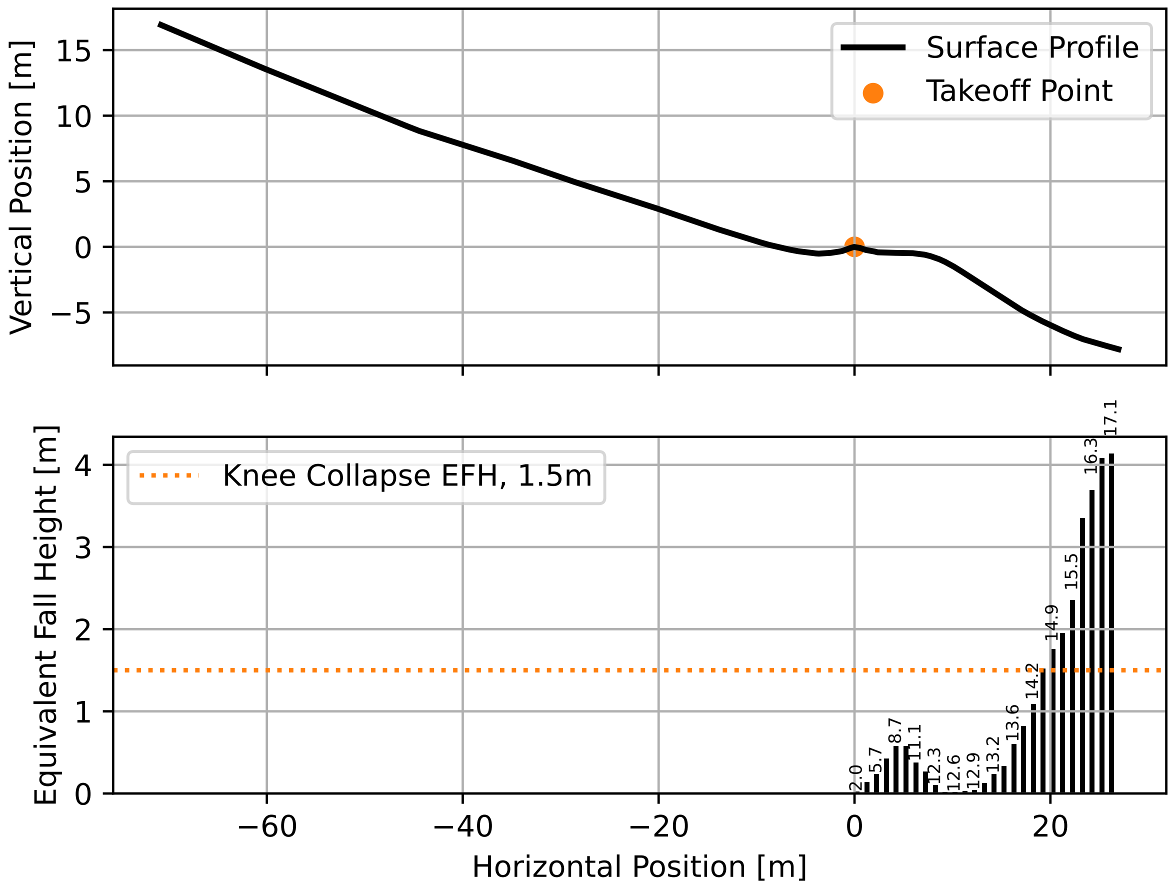
4.3 Calculate Equivalent Fall Height

The equivalent fall height of the landing surface is calculated at constant intervals relative to the provided takeoff point or start of the surface.

```
dist, efh, speeds = measured_surf.calculate_efh(
    np.deg2rad(takeoff_ang), takeoff_point,skier, increment=1.0)
```

There is a convenience function for plotting the calculated efh.

```
from skijumpdesign.functions import plot_efh

plot_efh(measured_surf, takeoff_ang, takeoff_point, skier=skier,
         increment=1.0)
```



CHAPTER

FIVE

EXAMPLES: ANALYSIS OF REAL JUMPS

This page analyses several jumps in which people have been injured on. The jumps have been measured by the package authors over the years. The intention is to show the utility of the software for analyzing arbitrary jump shapes and to highlight the large equivalent fall heights of these jump constructions. The jumps are idenified by location and the year it was measured.

Import packages needed on this page:

```
import numpy as np
import matplotlib.pyplot as plt
from skijumpdesign import Skier, Surface
from skijumpdesign.functions import (
    make_jump, plot_efh, cartesian_from_measurements)
```

5.1 Selection of an Equivalent Fall Height

Beginner ski slopes range from 6% to 25% grade (3 to 14 degrees) and intermediate ski slopes range from 25% to 40% (14 to 22 degrees)[1]. Terrain parks are typically built on steeper beginner slopes or shallow intermediate slopes, thus a parent slope grade of 25% (14 degrees) is a reasonable choice to compare the snow budgets of different jumps designed with equivalent fall heights. The following plot shows how the snow budget increases as EFH decreases.

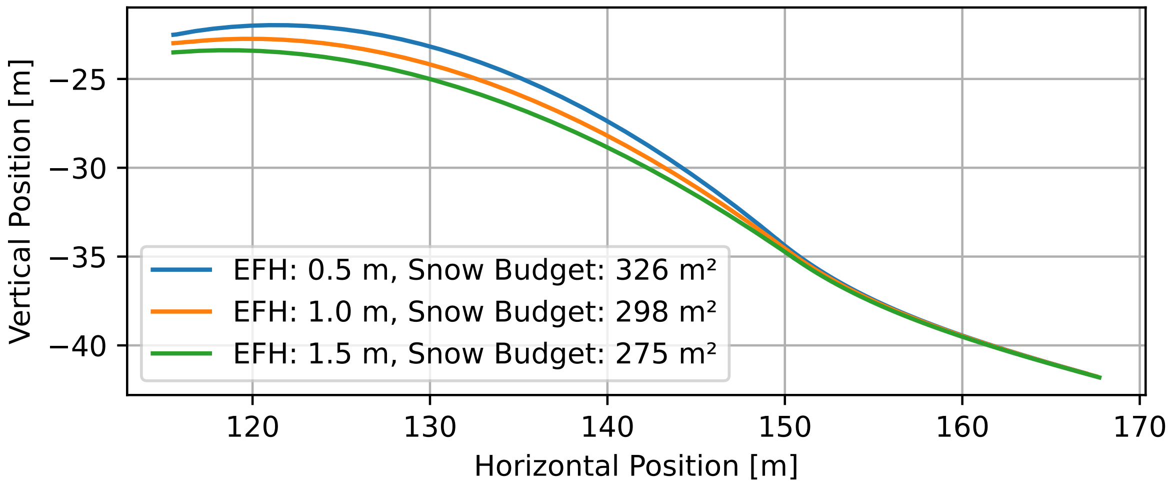
```
parent_slope_grade = 0.25  # percent grade
parent_slope_angle = -np.rad2deg(np.arctan(parent_slope_grade))  # degrees
approach_length = 100.0  # meters
takeoff_angle = 20.0  # degrees

fig, ax = plt.subplots(1, 1)

for efh, color in zip((0.5, 1.0, 1.5), ('C0', 'C1', 'C2')):
   jump = make_jump(parent_slope_angle, 0.0, approach_length,
                    takeoff_angle, efh)
   snow_budget = jump[-1]['Snow Budget']
   for i, surf in enumerate(jump[3:-2]):
      if i % 2 == 1:
         lab = 'EFH: {:1.1f} m, Snow Budget: {:1.0f} m²'.format(efh, snow_budget)
      else:
         lab = None
      surf.plot(ax=ax, color=color, label=lab)
```

(continues on next page)

[1] https://en.wikipedia.org/wiki/Piste

(continued from previous page)

```
ax.set_xlabel('Horizontal Position [m]')
ax.set_ylabel('Vertical Position [m]')
ax.set_aspect('equal')
ax.legend()
ax.grid()
```



An equivalent fall height of 1.5 m will, on average, cause knee collapse in an adult[2]. This is a sensible absolute maximal boundary for equivalent fall height in constructed jumps. An equivalent fall height of 0.5 m is a fairly benign height, similar to falling from two or three stair steps. A height of 1 m is a good compromise between these two numbers that has a reasonable snow budget with moderate height. The following jumps will compared to jumps designed with a 1 m equivalent fall height for this reason.

5.2 Design Speed

The "design speed" for a constant equivalent fall height jump is defined in[3] as:

> The maximum takeoff velocity (resulting from the highest start point and minimum snow friction μ and air drag η) is called the design speed.

It is the maximum expected takeoff speed of a skier or snowboarder, which is a function of the inrun length, slope, friction coefficient, and air drag. The designed jumps ensure a constant equivalent fall height up to this design speed.

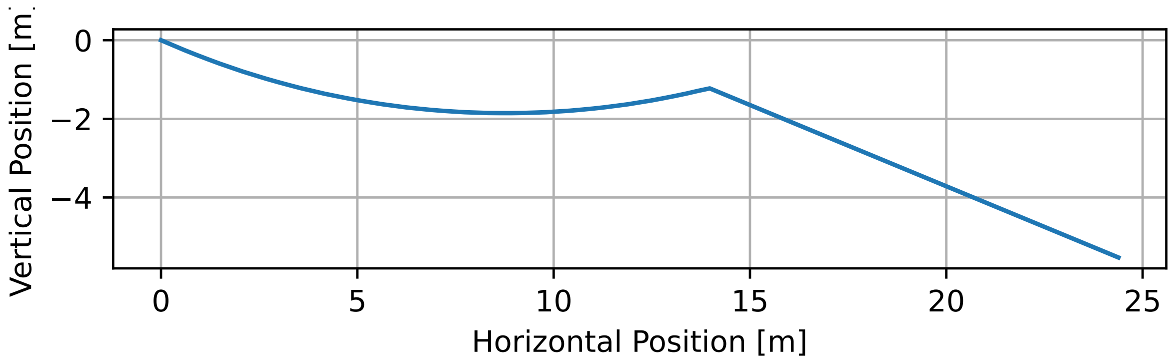
[2] A. E. Minetti, "Using leg muscles as shock absorbers: theoretical predictions and experimental results of drop landing performance," Ergonomics, vol. 41, no. 12, pp. 1771–1791, Dec. 1998, doi: 10.1080/001401398185965.

[3] Levy, Dean, Mont Hubbard, James A. McNeil, and Andrew Swedberg. "A Design Rationale for Safer Terrain Park Jumps That Limit Equivalent Fall Height." Sports Engineering 18, no. 4 (December 2015): 227–39. https://doi.org/10.1007/s12283-015-0182-6.

5.3 California 2002

The `california-2002-surface.csv` file contains the horizontal (x) and vertical (y) coordinates of a jump measured at a ski resort in California, USA in 2002. The comma separated value file can be loaded with `numpy.loadtxt()` and used to create a *Surface*. The *plot()* method is used to quickly visualize the measured landing surface. The takeoff location is situated at (x=0 m, y=0 m).

```
landing_surface_data = np.loadtxt('california-2002-surface.csv',
                                  delimiter=',',  # comma separated
                                  skiprows=1)  # skip the header row

landing_surface = Surface(landing_surface_data[:, 0],  # x values in meters
                          landing_surface_data[:, 1])  # y values in meters

ax = landing_surface.plot()
```



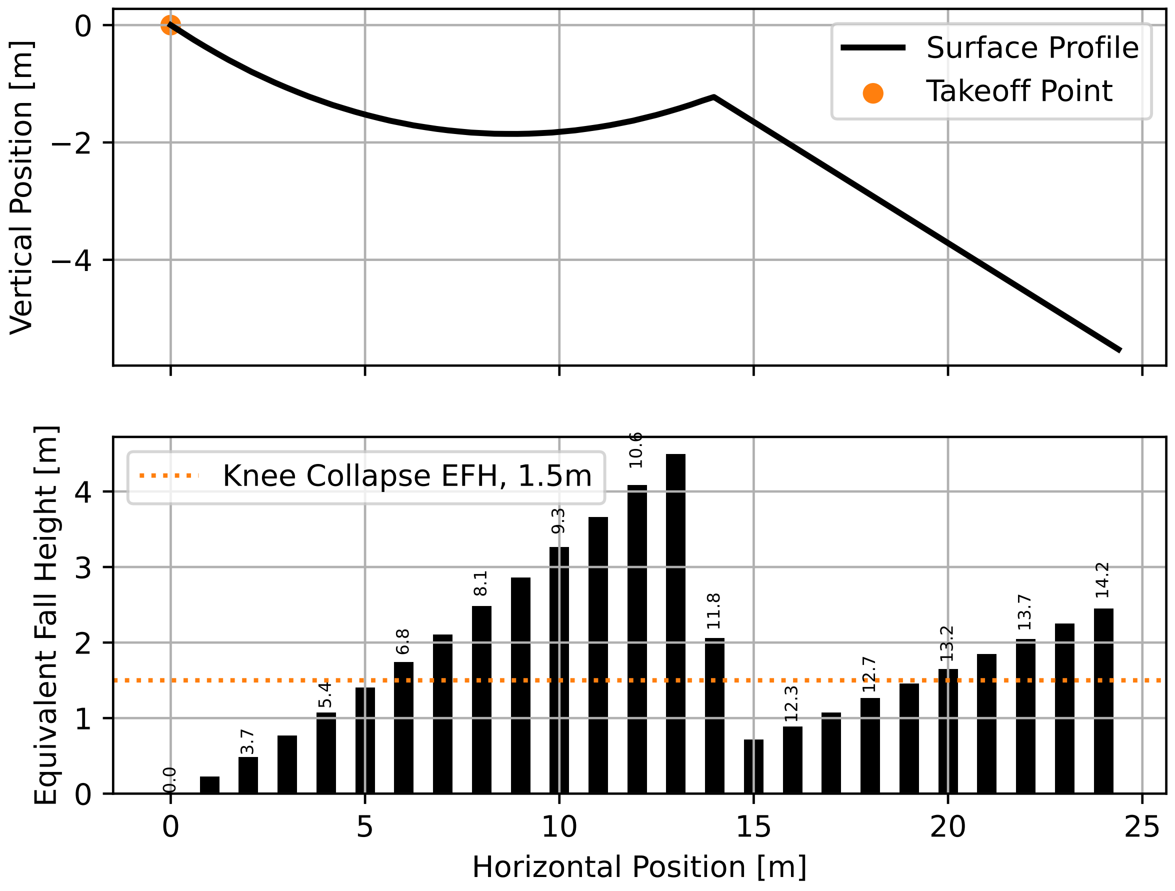
The takeoff angle of this jump was measured as 13 degrees. Using this angle the equivalent fall height can be visualized across the landing surface.

```
takeoff_angle = 30.0  # degrees
takeoff_point = (0.0, 0.0)  # meters

skier = Skier()

plot_efh(landing_surface, takeoff_angle, takeoff_point,
         skier=skier, increment=1.0)
```

The actual jump can be compared to a jump designed with a constant equivalent fall height. The figure below shows such a comparison.

```
def compare_measured_to_designed(measured_surface, equiv_fall_height,
                                 parent_slope_angle, approach_length,
                                 takeoff_angle, skier):

    # NOTE : A different Skier() object is used internally in make_jump()
    slope, approach, takeoff, landing, landing_trans, flight, outputs = \
        make_jump(parent_slope_angle, 0.0, approach_length, takeoff_angle,
                  equiv_fall_height)

    measured_surface.shift_coordinates(takeoff.end[0], takeoff.end[1])
```

(continues on next page)


Surface Profile
Takeoff Point
Vertical Position [m]
Knee Collapse EFH, 1.5m
Equivalent Fall Height [m]
Horizontal Position [m]
0.0
3.7
5.4
6.8
8.1
9.3
10.6
11.8
12.3
12.7
13.2
13.7
14.2

(continued from previous page)

```
design_speed = flight.speed[0]
low_speed = 1/2*design_speed
med_speed = 3/4*design_speed

vel_vec = np.array([np.cos(np.deg2rad(takeoff_angle)),
                    np.sin(np.deg2rad(takeoff_angle))])

flight_low = skier.fly_to(measured_surface, init_pos=takeoff.end,
                          init_vel=tuple(low_speed*vel_vec))
flight_med = skier.fly_to(measured_surface, init_pos=takeoff.end,
                          init_vel=tuple(med_speed*vel_vec))

fig, (prof_ax, efh_ax) = plt.subplots(2, 1, sharex=True,
                                      constrained_layout=True)

increment = 1.0

dist, efh, _ = measured_surface.calculate_efh(np.deg2rad(takeoff_angle),
                                              takeoff.end, skier, increment)

efh_ax.bar(dist, efh, color='black', align='center', width=increment/2,
           label="Measured Landing Surface")

dist, efh, _ = landing.calculate_efh(np.deg2rad(takeoff_angle),
                                     takeoff.end, skier, increment)

efh_ax.bar(dist, efh, color='C2', align='edge', width=increment/2,
           label="Designed Landing Surface")

dist, efh, _ = landing_trans.calculate_efh(np.deg2rad(takeoff_angle),
                                           takeoff.end, skier, increment)

efh_ax.bar(dist, efh, color='C2', align='edge', width=increment/2,
           label=None)

efh_ax.axhline(5.1, color='C1', label='Avg. 2 Story Fall Height')
efh_ax.axhline(2.6, color='C1', linestyle='dashed',
               label='Avg. 1 Story Fall Height')
efh_ax.axhline(1.5, color='C1', linestyle='dashdot',
               label='Knee Collapse Height')

prof_ax = takeoff.plot(ax=prof_ax, linewidth=2, color='C2', label=None)

prof_ax = flight_low.plot(ax=prof_ax, color='black', linestyle='dashdot',
                          label='Flight @ {:1.0f} m/s'.format(low_speed))
prof_ax = flight_med.plot(ax=prof_ax, color='black', linestyle='dashed',
                          label='Flight @ {:1.0f} m/s'.format(med_speed))
prof_ax = flight.plot(ax=prof_ax, color='black', linestyle='dotted',
                      label='Flight @ {:1.0f} m/s'.format(design_speed))

prof_ax = landing.plot(ax=prof_ax, color='C2', linewidth=2, label=None)
```

(continues on next page)

(continued from previous page)

```
    prof_ax = landing_trans.plot(ax=prof_ax, color='C2', linewidth=2,
                                 label='Designed Landing Surface')

    prof_ax = measured_surface.plot(ax=prof_ax, color='black',
                                    label="Measured Landing Surface")

    prof_ax.set_title('Design Speed: {:1.0f} m/s'.format(design_speed))

    prof_ax.set_ylabel('Vertical Position [m]')
    efh_ax.set_ylabel('Equivalent Fall Height [m]')
    efh_ax.set_xlabel('Horizontal Position [m]')

    efh_ax.grid()
    prof_ax.grid()
    efh_ax.legend(loc='upper left')
    prof_ax.legend(loc='lower left')

    return prof_ax, efh_ax
```

```
fall_height = 1.0  # meters
slope_angle = -8.0  # degrees
approach_length = 180.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
                             approach_length, takeoff_angle, skier)
```

The average story heights are estimated from[4].

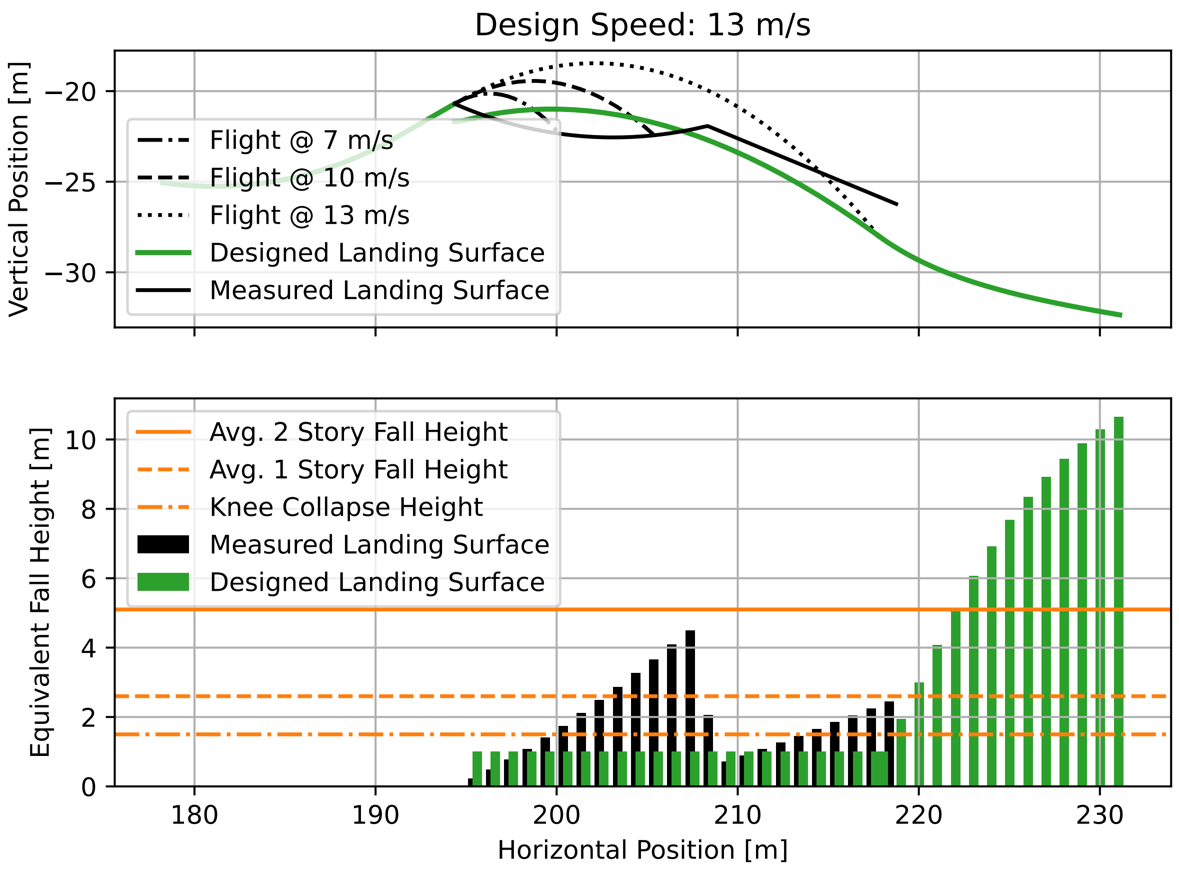
5.4 Washington 2004

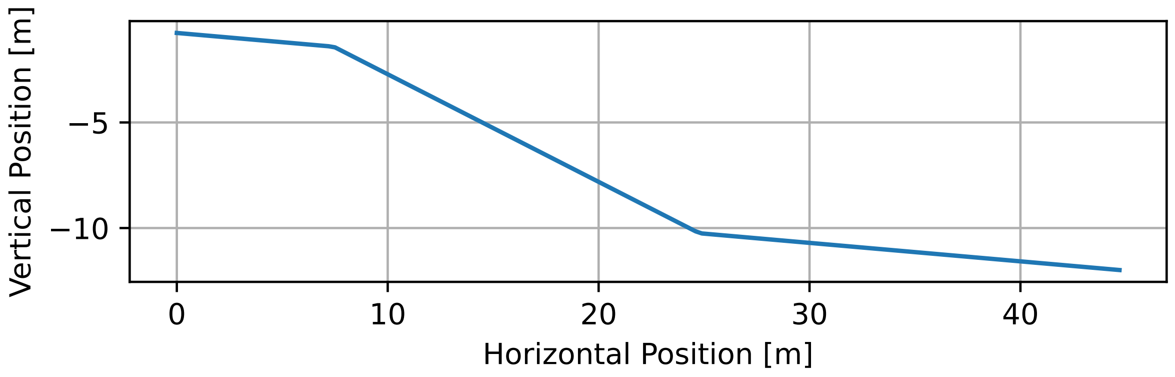
The `washington-2004-surface.csv` file contains the horizontal (x) and vertical (y) coordinates of a jump measured at a Washington, USA ski resort in 2004. The comma separated value file can be loaded with `numpy.loadtxt()` and used to create a *Surface*. The *plot()* method is used to quickly visualize the measured landing surface. The takeoff location is situated at (x=0 m, y=0 m).

```
landing_surface_data = np.loadtxt('washington-2004-surface.csv',
                                  delimiter=',',  # comma separated
                                  skiprows=1)  # skip the header row

landing_surface = Surface(landing_surface_data[:, 0],  # x values in meters
                          landing_surface_data[:, 1])  # y values in meters

ax = landing_surface.plot()
```

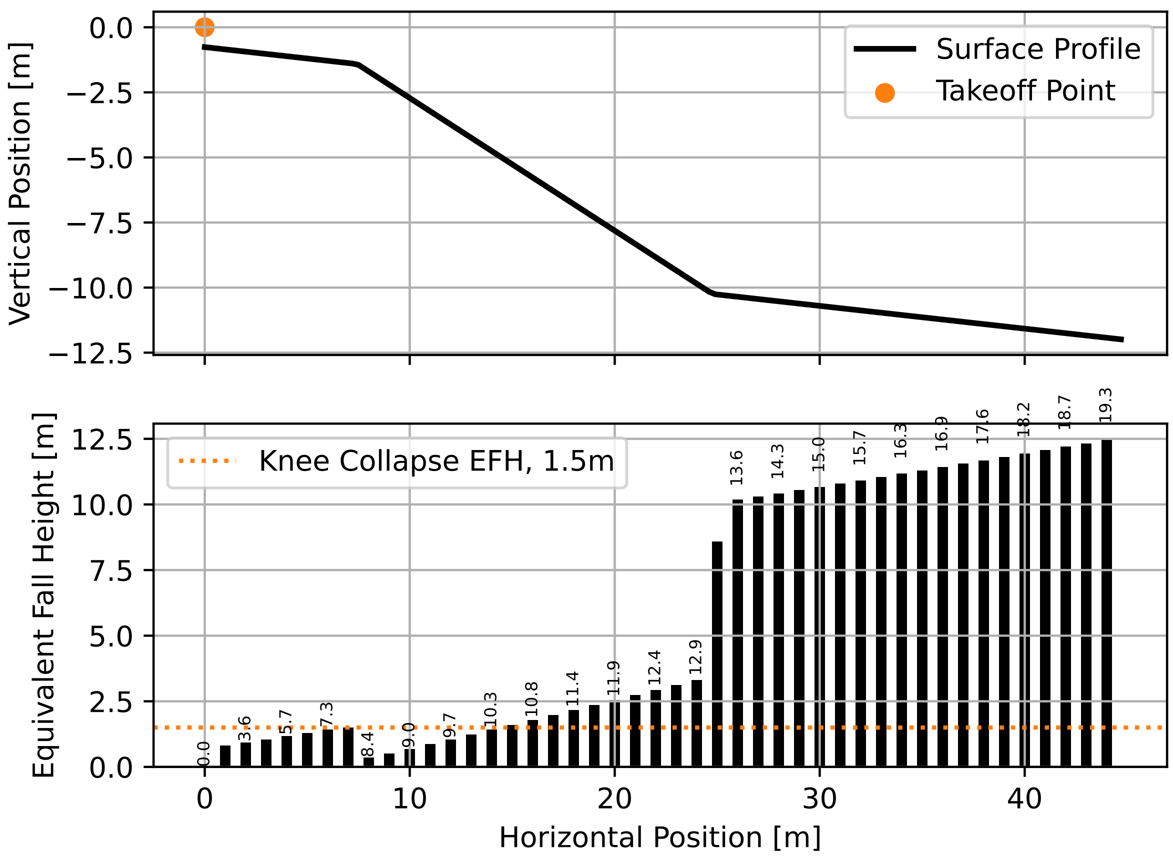
The takeoff angle of this jump was measured as 25 degrees. Using this angle the equivalent fall height can be visualized across the landing surface.

[4] N. L. Vish, "Pediatric window falls: not just a problem for children in high rises," Injury Prevention, vol. 11, no. 5, pp. 300–303, Oct. 2005, doi: 10.1136/ip.2005.008664.


Design Speed: 13 m/s
Vertical Position [m]
−20
−25
−30
Flight @ 7 m/s
Flight @ 10 m/s
Flight @ 13 m/s
Designed Landing Surface
Measured Landing Surface
Equivalent Fall Height [m]
0
2
4
6
8
10
Avg. 2 Story Fall Height
Avg. 1 Story Fall Height
Knee Collapse Height
Measured Landing Surface
Designed Landing Surface
180
190
200
210
220
230
Horizontal Position [m]


Vertical Position [m]
−5
−10
0
10
20
30
40
Horizontal Position [m]

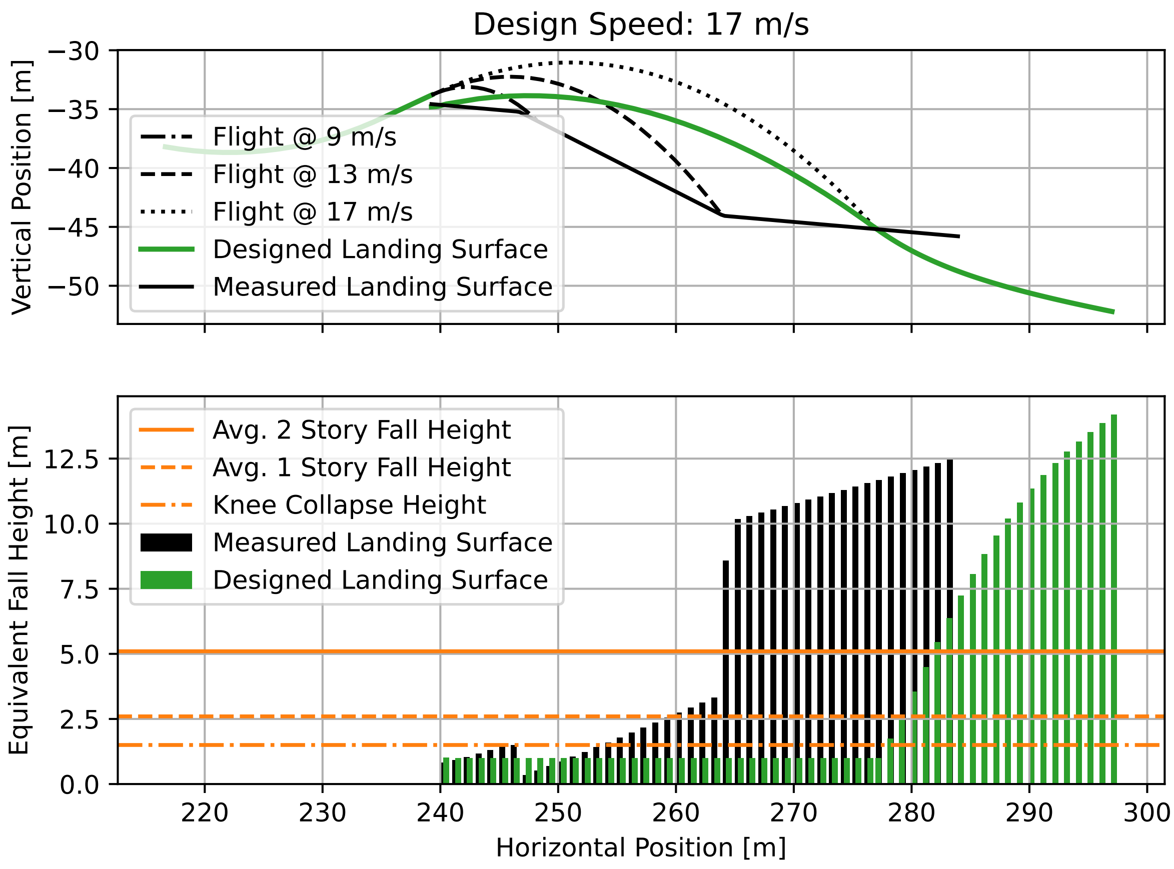
```
takeoff_angle = 25.0  # degrees
takeoff_point = (0.0, 0.0)  # meters

skier = Skier()

plot_efh(landing_surface, takeoff_angle, takeoff_point,
         skier=skier, increment=1.0)
```



For high takeoff speeds, this jump has very large equivalent fall heights (3 m to 13 m).

The actual jump can be compared to a jump designed with a constant equivalent fall height. The figure below shows such a comparison. Note that the first 15 meters or so of the surface is reasonable, but if a jumper lands beyond 15 m they will be subjected to dangerous impact velocities.

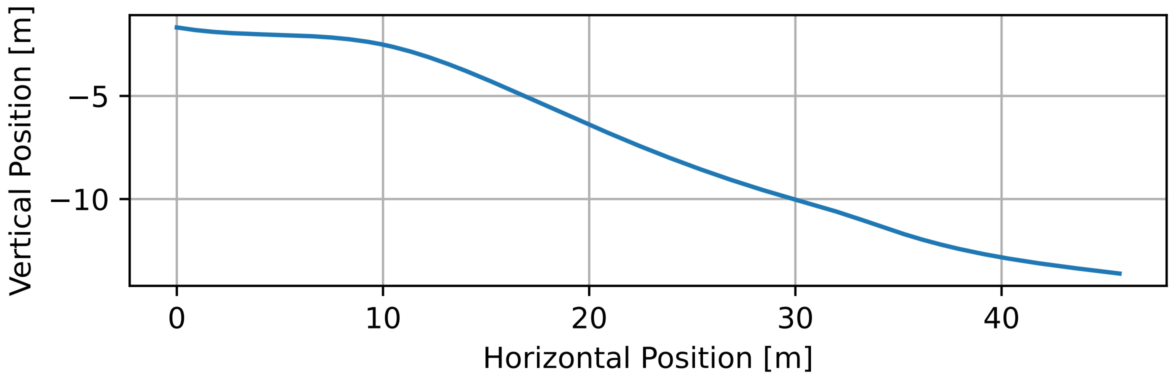
```
fall_height = 1.0  # meters
slope_angle = -10.0  # degrees
approach_length = 220.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
                             approach_length, takeoff_angle, skier)
```


Design Speed: 17 m/s
Vertical Position [m]
−30
−35
−40
−45
−50
Flight @ 9 m/s
Flight @ 13 m/s
Flight @ 17 m/s
Designed Landing Surface
Measured Landing Surface
Equivalent Fall Height [m]
12.5
10.0
7.5
5.0
2.5
0.0
Avg. 2 Story Fall Height
Avg. 1 Story Fall Height
Knee Collapse Height
Measured Landing Surface
Designed Landing Surface
220
230
240
250
260
270
280
290
300
Horizontal Position [m]

5.5 Utah 2010

The `utah-2010-surface.csv` file contains the horizontal (x) and vertical (y) coordinates of a jump measured at a Utah, USA ski resort in February 2010. The comma separated value file can be loaded with `numpy.loadtxt()` and used to create a *Surface*. The *plot()* method is used to quickly visualize the measured landing surface. The takeoff location is situated at (x=0 m, y=0 m).

```
landing_surface_data = np.loadtxt('utah-2010-surface.csv',
                                  delimiter=',',  # comma separated
                                  skiprows=1)  # skip the header row

landing_surface = Surface(landing_surface_data[:, 0],  # x values in meters
                          landing_surface_data[:, 1])  # y values in meters

ax = landing_surface.plot()
```



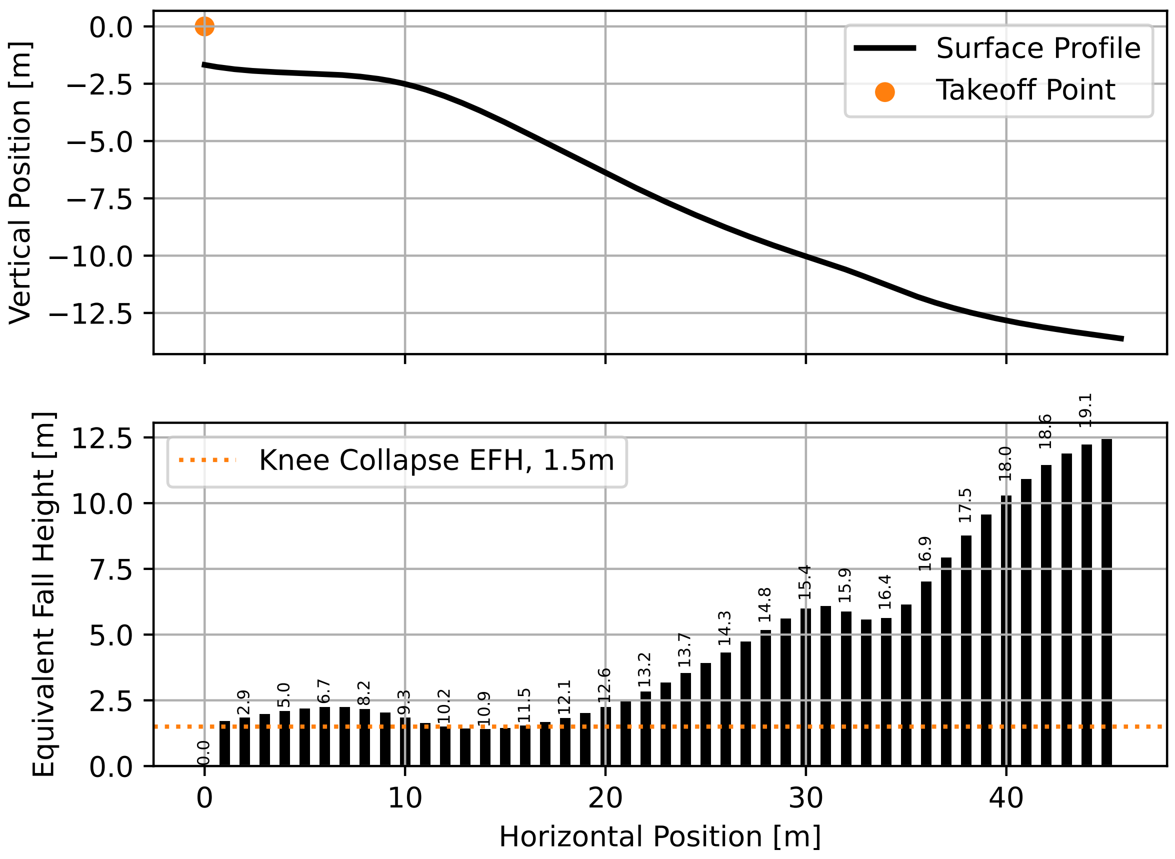
The takeoff angle of this jump was measured as 23 degrees. Using this angle the equivalent fall height can be visualized across the landing surface.

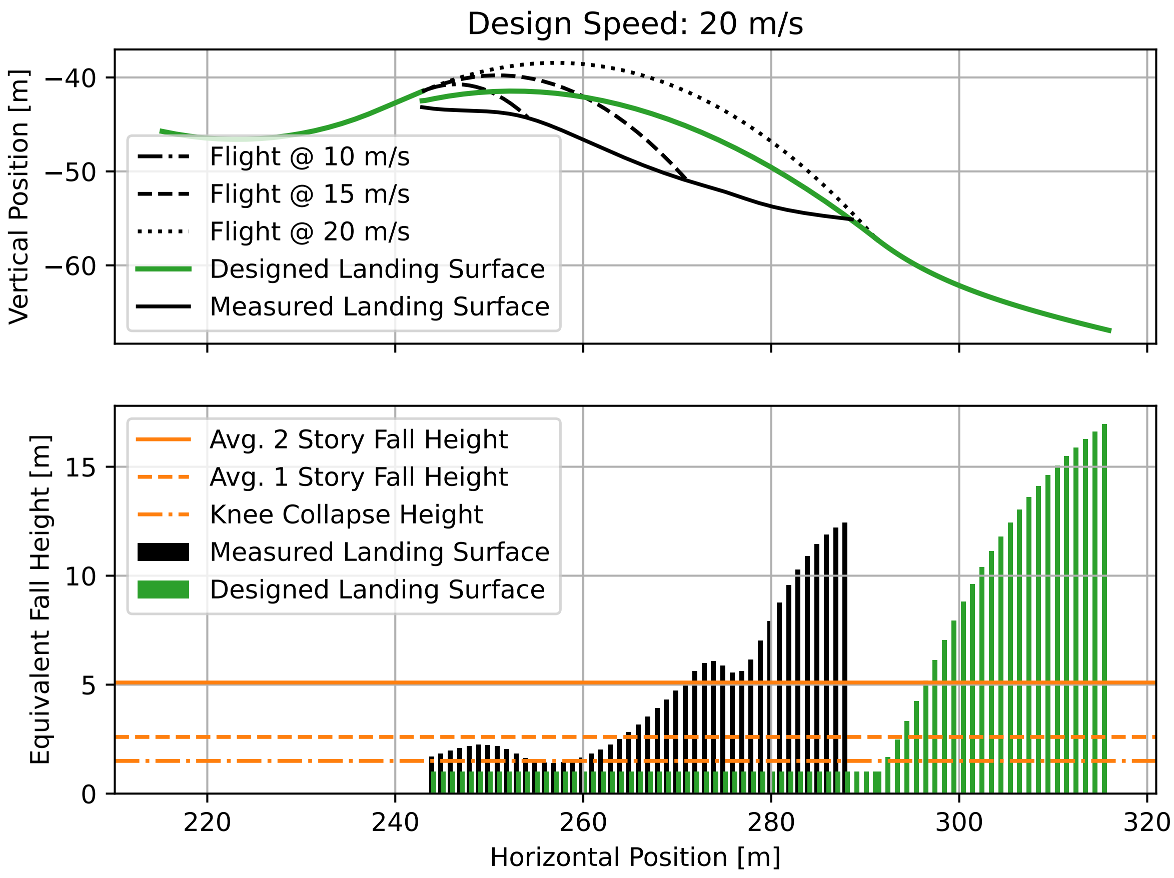
```
takeoff_angle = 23.0  # degrees
takeoff_point = (0.0, 0.0)  # meters

skier = Skier()

plot_efh(landing_surface, takeoff_angle, takeoff_point,
         skier=skier, increment=1.0)
```

For high takeoff speeds, this jump has very large equivalent fall heights (5 m to 10 m). And no mater the takeoff speed, the equivalent fall height is greater than or equal to the 1.5 m threshold for knee collapse.

The measured jump can be compared to a jump designed to ensure a constant equivalent fall height of 1.5 m at any takeoff speed. The figure below shows such a comparison. Note that the first 15 meters or so of the surface is reasonable, but if a jumper lands beyond 15 m they will be subjected to dangerous impact speeds.

```
fall_height = 1.0  # meters
slope_angle = -12.0  # degrees
approach_length = 220.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
                             approach_length, takeoff_angle, skier)
```

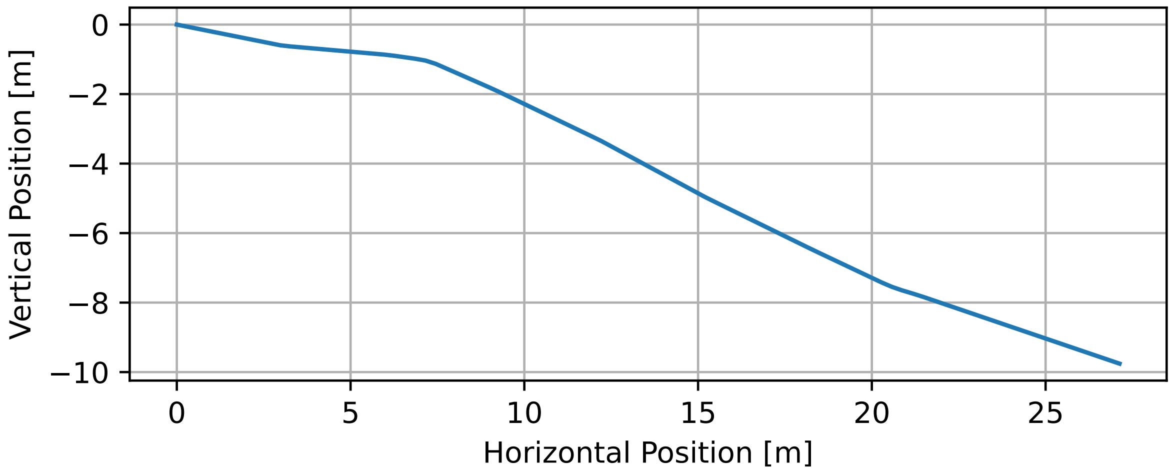

Vertical Position [m]
Surface Profile
Takeoff Point
Equivalent Fall Height [m]
Knee Collapse EFH, 1.5m
Horizontal Position [m]
0.0
2.9
5.0
6.7
8.2
9.3
10.2
10.9
11.5
12.1
12.6
13.2
13.7
14.3
14.8
15.4
15.9
16.4
16.9
17.5
18.0
18.6
19.1


Design Speed: 20 m/s
Vertical Position [m]
−40
−50
−60
Flight @ 10 m/s
Flight @ 15 m/s
Flight @ 20 m/s
Designed Landing Surface
Measured Landing Surface
Equivalent Fall Height [m]
15
10
5
0
Avg. 2 Story Fall Height
Avg. 1 Story Fall Height
Knee Collapse Height
Measured Landing Surface
Designed Landing Surface
220
240
260
280
300
320
Horizontal Position [m]

5.6 Colorado 2009

The `colorado-2009-surface.csv` file contains the horizontal (x) and vertical (y) coordinates of a jump measured by professional surveyors at a Colorado, USA ski resort in March 2009. The comma separated value file can be loaded with `numpy.loadtxt()` and used to create a *Surface*. The *plot()* method is used to quickly visualize the measured landing surface. The takeoff location is situated at (x=0 m, y=0 m).

```
landing_surface_data = np.loadtxt('colorado-2009-surface.csv',
                                  delimiter=',',  # comma separated
                                  skiprows=1)  # skip the header row

landing_surface = Surface(landing_surface_data[:, 0],  # x values in meters
                          landing_surface_data[:, 1])  # y values in meters

ax = landing_surface.plot()
```

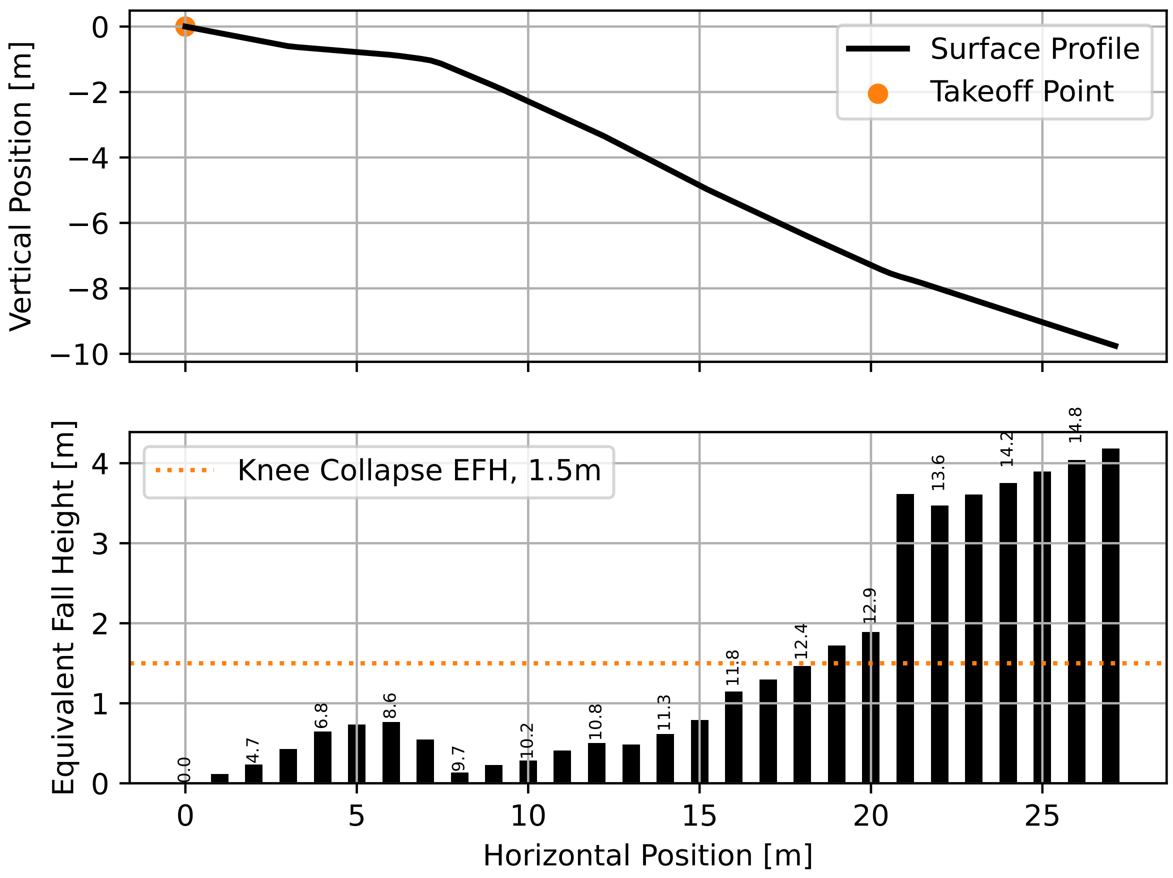


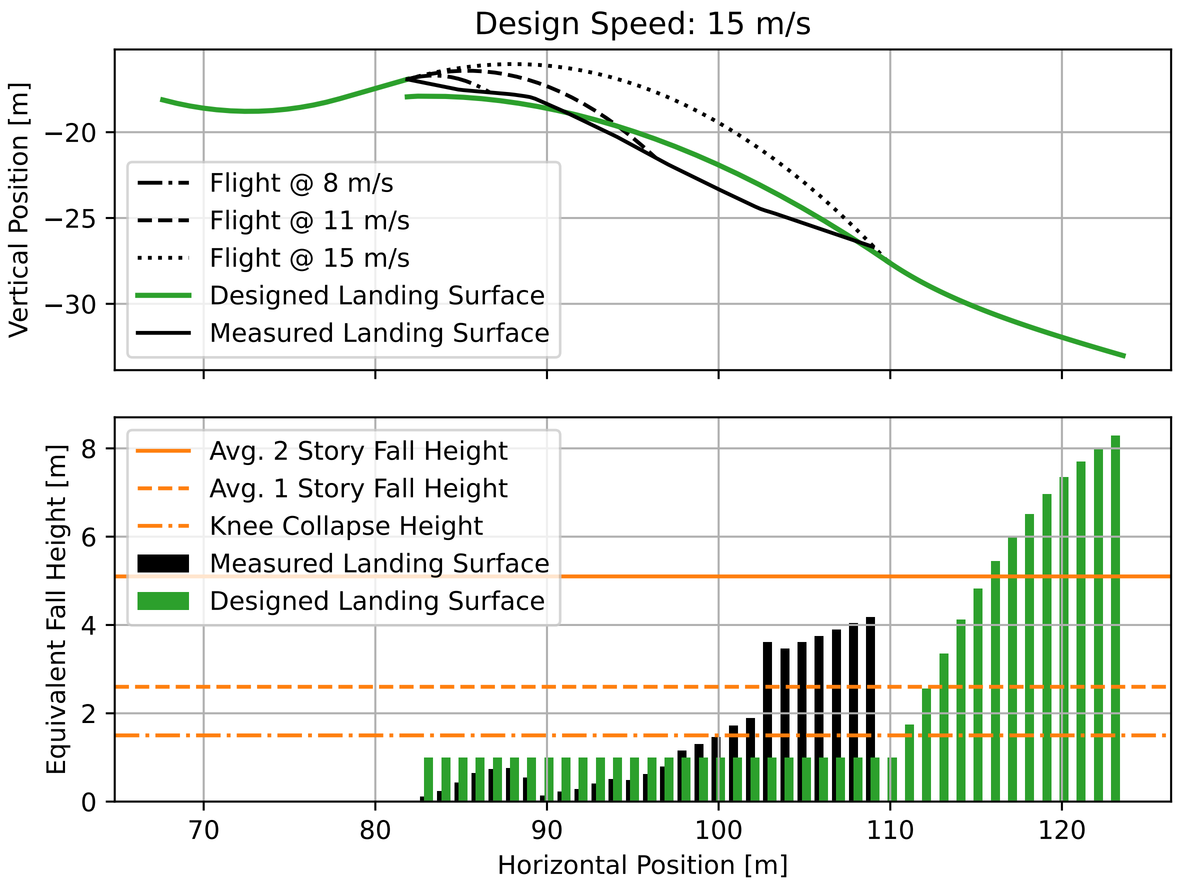
The takeoff angle of this jump was measured as 16 degrees. Using this angle the equivalent fall height can be visualized across the landing surface.

```
takeoff_angle = 16.0  # degrees
takeoff_point = (0.0, 0.0)  # meters

skier = Skier()

plot_efh(landing_surface, takeoff_angle, takeoff_point,
         skier=skier, increment=1.0)
```

The actual jump can be compared to a jump designed with a constant equivalent fall height. The figure below shows such a comparison.

```
fall_height = 1.0  # meters
slope_angle = -15.0  # degrees
approach_length = 70.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
                             approach_length, takeoff_angle, skier)
```

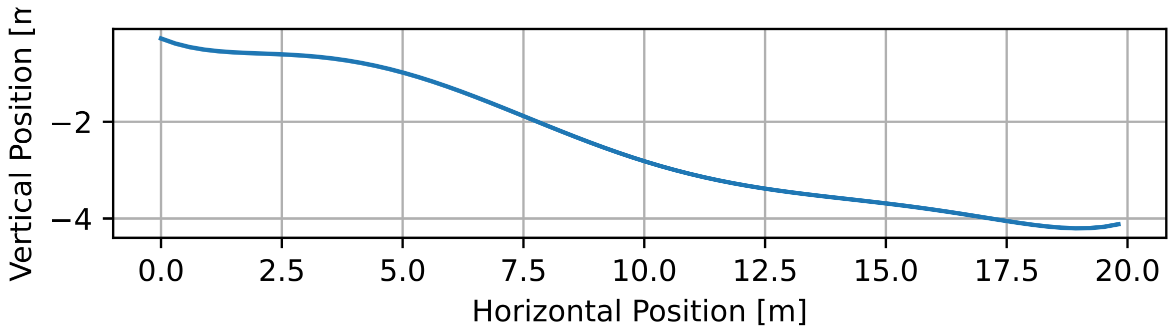

Surface Profile
Takeoff Point
Vertical Position [m]
0
−2
−4
−6
−8
−10
Knee Collapse EFH, 1.5m
Equivalent Fall Height [m]
0.0
4.7
6.8
8.6
9.7
10.2
10.8
11.3
11.8
12.4
12.9
13.6
14.2
14.8
0
1
2
3
4
0
5
10
15
20
25
Horizontal Position [m]


Design Speed: 15 m/s
Vertical Position [m]
−20
−25
−30
Flight @ 8 m/s
Flight @ 11 m/s
Flight @ 15 m/s
Designed Landing Surface
Measured Landing Surface
Equivalent Fall Height [m]
0
2
4
6
8
Avg. 2 Story Fall Height
Avg. 1 Story Fall Height
Knee Collapse Height
Measured Landing Surface
Designed Landing Surface
70
80
90
100
110
120
Horizontal Position [m]

5.7 Wisconsin 2015

The `wisconsin-2015-surface.csv` file contains the horizontal (x) and vertical (y) coordinates of a jump measured at a Wisconsin, USA ski resort in 2015. The comma separated value file can be loaded with `numpy.loadtxt()` and used to create a *Surface*. The *plot()* method is used to quickly visualize the measured landing surface. The takeoff location is situated at (x=0 m, y=0 m).

```
landing_surface_data = np.loadtxt('wisconsin-2015-surface.csv',
                                  delimiter=',',  # comma separated
                                  skiprows=1)  # skip the header row

landing_surface = Surface(landing_surface_data[:, 0],  # x values in meters
                          landing_surface_data[:, 1])  # y values in meters

ax = landing_surface.plot()
```

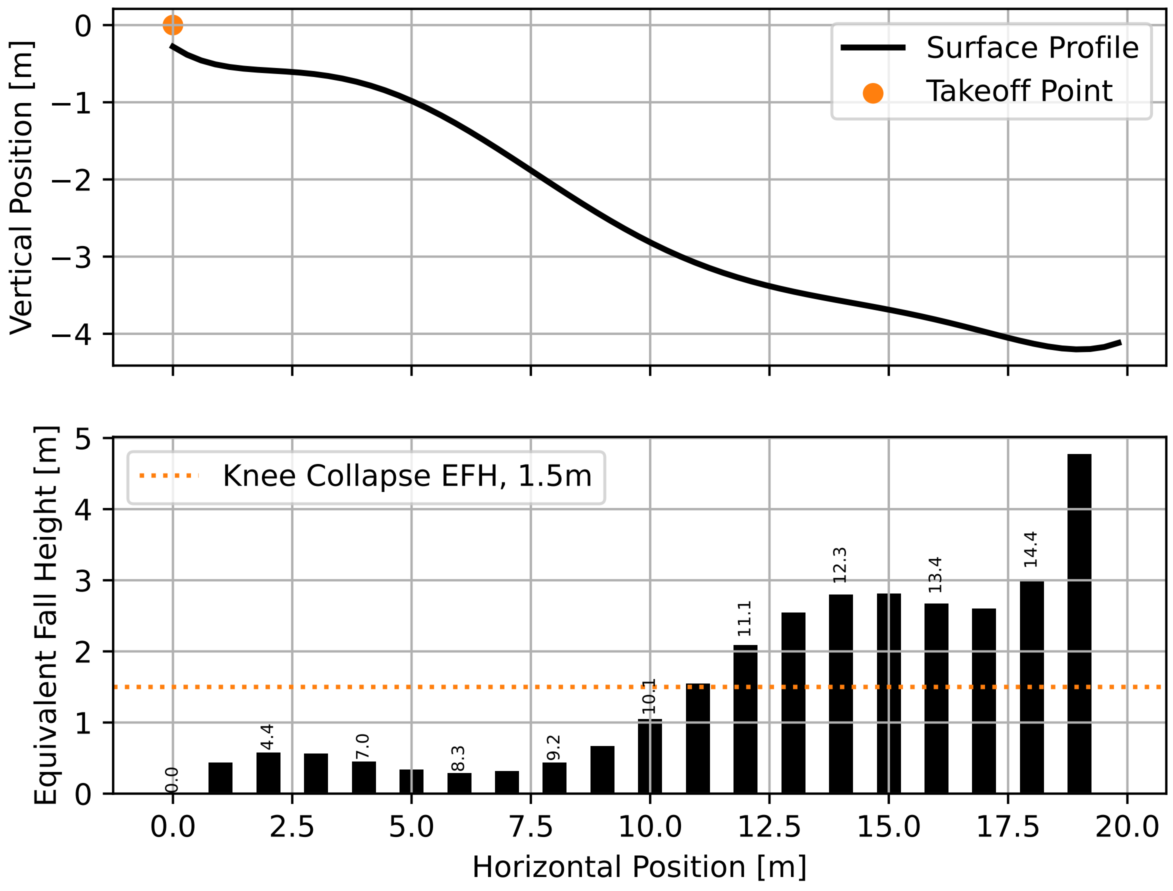


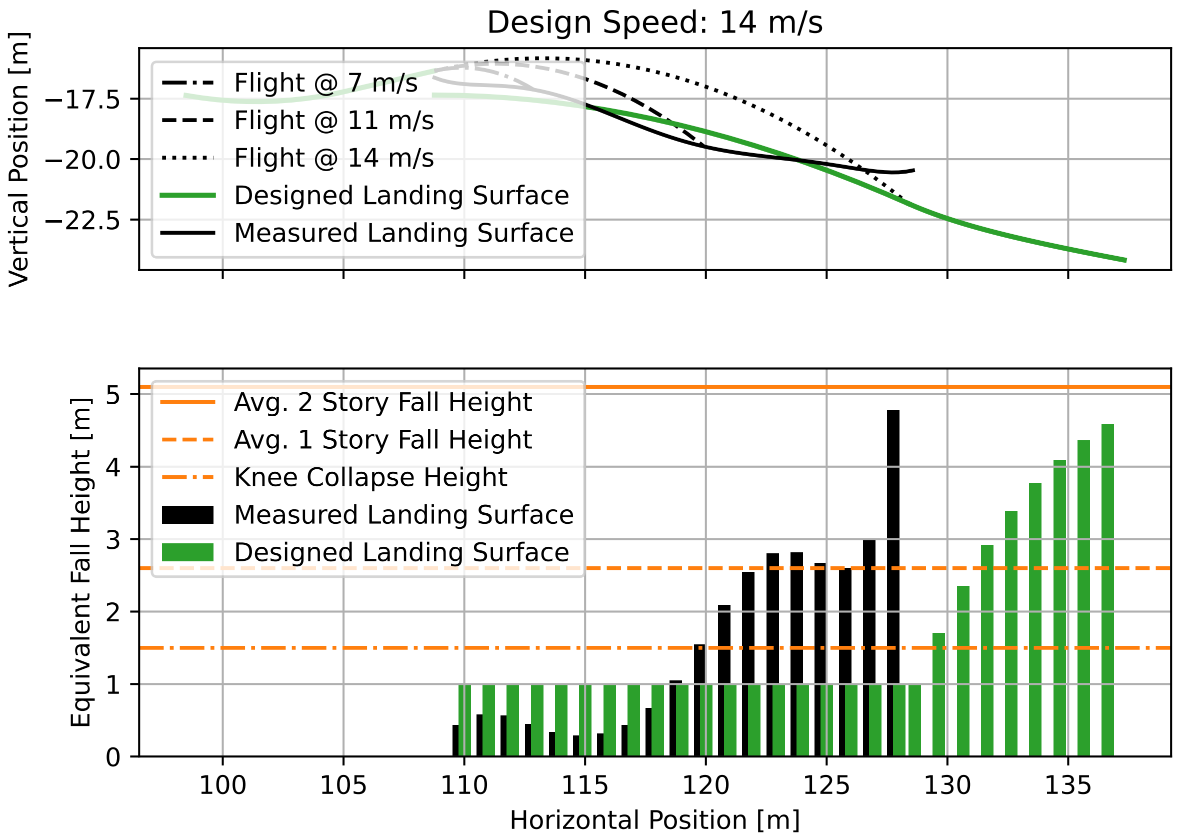
The takeoff angle of this jump was measured as 13 degrees. Using this angle the equivalent fall height can be visualized across the landing surface.

```
takeoff_angle = 13.0  # degrees
takeoff_point = (0.0, 0.0)  # meters

skier = Skier()

plot_efh(landing_surface, takeoff_angle, takeoff_point,
         skier=skier, increment=1.0)
```

The actual jump can be compared to a jump designed with a constant equivalent fall height. The figure below shows such a comparison.

```
fall_height = 1.0  # meters
slope_angle = -10.0  # degrees
approach_length = 100.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
                             approach_length, takeoff_angle, skier)
```


Surface Profile
Takeoff Point
Vertical Position [m]
Knee Collapse EFH, 1.5m
Equivalent Fall Height [m]
Horizontal Position [m]
0.0
4.4
7.0
8.3
9.2
10.1
11.1
12.3
13.4
14.4

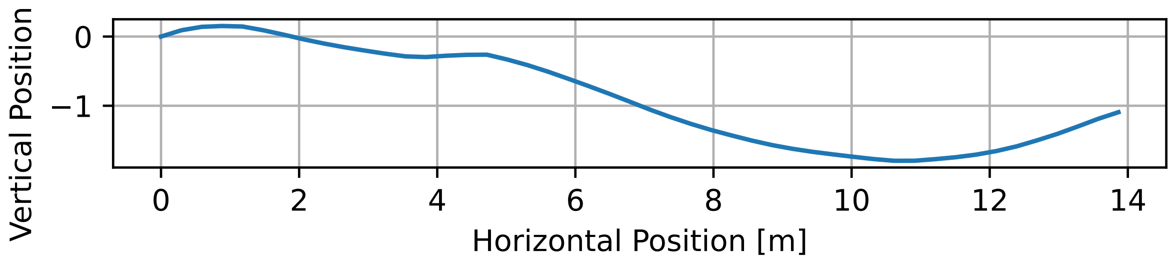


5.8 Sydney 2020

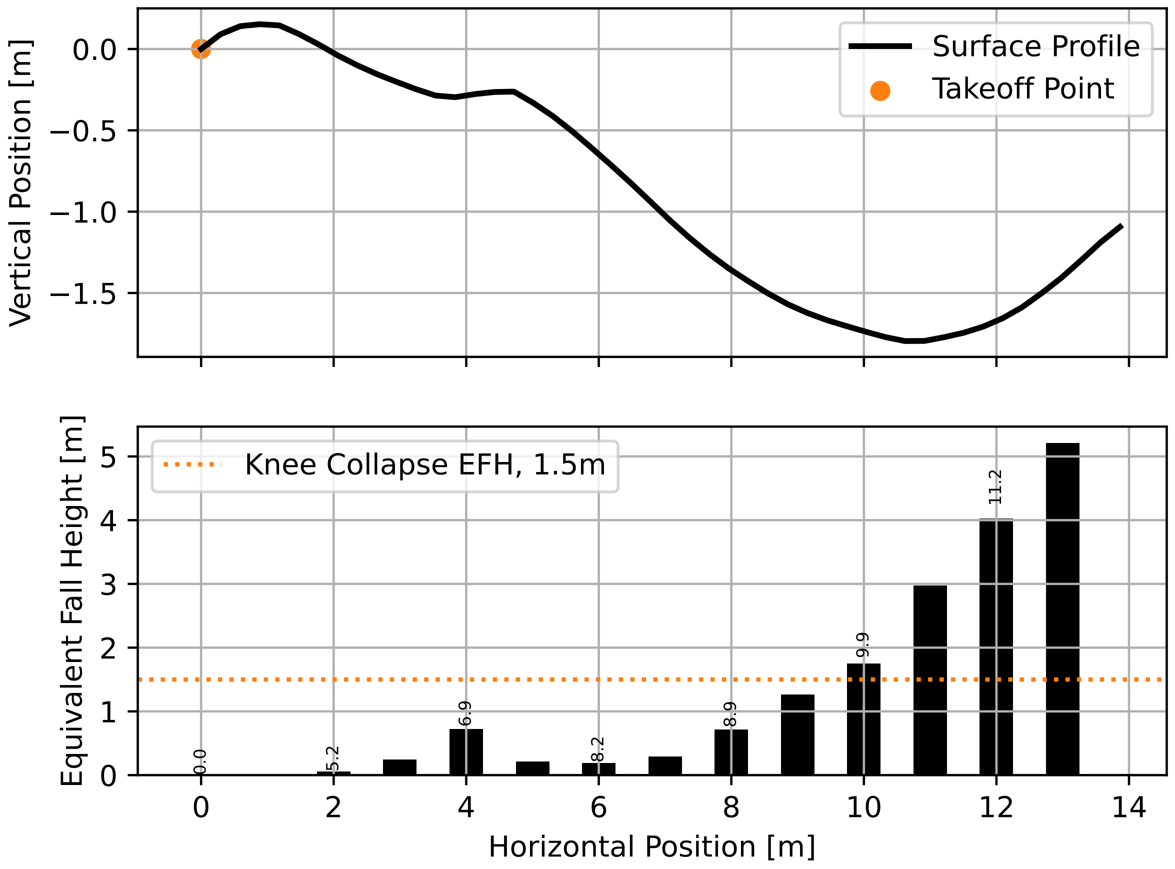
The `sydney-measurements-2020.csv` file contains the distance along the jump surface and absolute angle measurements (different measures than all above files) of a single-track dirt mountain bike jump measured near Sydney, Australia in 2020. The comma separated value file can be loaded with `numpy.loadtxt()`. These measurements require conversion to the Cartesian coordinates for constructing the surface using *`cartesian_from_measurements()`*. After conversion the data can be used to create a *`Surface`*. The *`plot()`* method is used to quickly visualize the measured landing surface. The takeoff location is situated at the first measurement point.

```
surface_measurement_data = np.loadtxt('sydney-measurements-2020.csv',
                                      delimiter=',',  # comma separated
                                      skiprows=1)  # skip the header row

x, y, takeoff_point, takeoff_angle = cartesian_from_measurements(
    surface_measurement_data[:, 0],  # distance along surface in meters
    np.deg2rad(surface_measurement_data[:, 1]))  # absolute angle deg -> rad

landing_surface = Surface(x,  # x values in meters
                          y)  # y values in meters

ax = landing_surface.plot()
```

The takeoff angle is taken from the angle measurements. Using this angle the equivalent fall height can be visualized across the landing surface.



```
skier = Skier()

plot_efh(landing_surface, np.rad2deg(takeoff_angle), takeoff_point,
         skier=skier, increment=1.0)
```



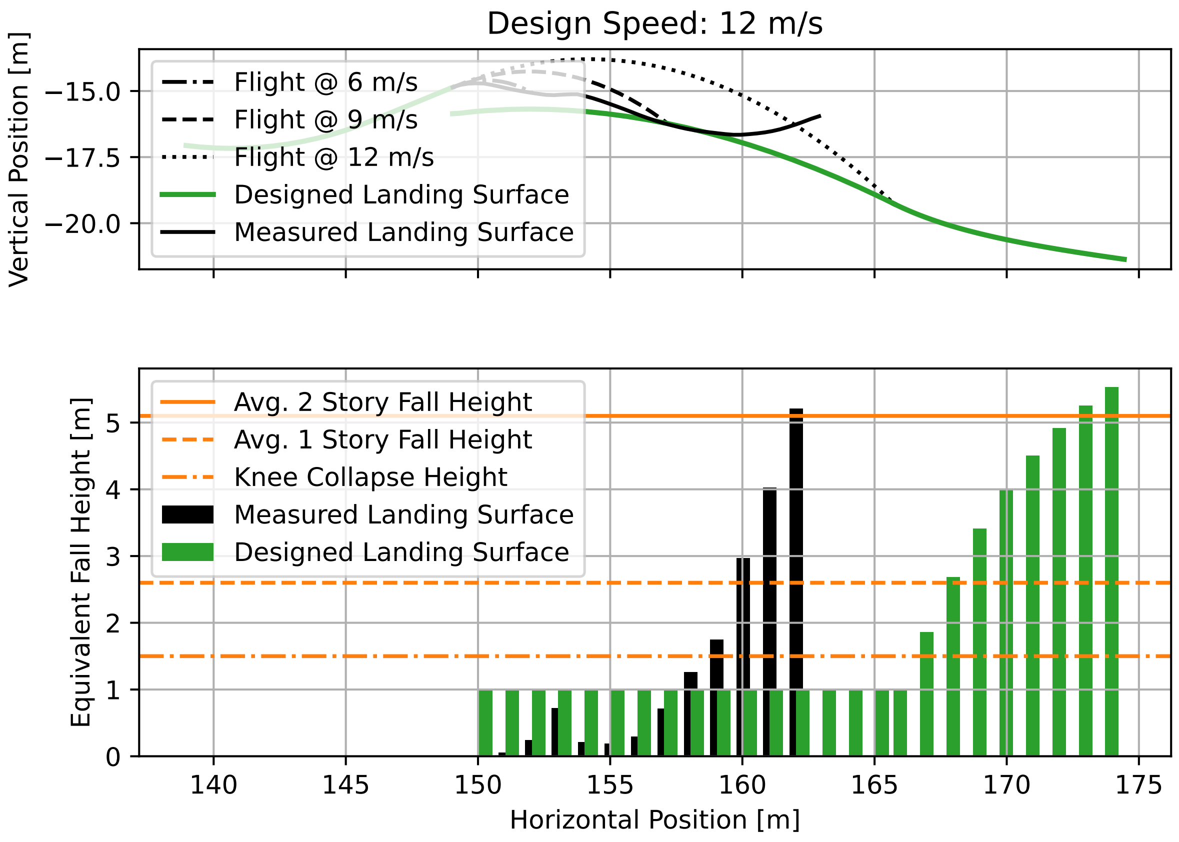
The actual jump can be compared to a jump designed with a constant equivalent fall height. The figure below shows such a comparison.

```
fall_height = 1.0  # meters
slope_angle = -7.0  # degrees
approach_length = 140.0  # meters

compare_measured_to_designed(landing_surface, fall_height, slope_angle,
```

(continues on next page)

(continued from previous page)

```
                                  approach_length, np.rad2deg(takeoff_angle), skier)
```



CHAPTER

SIX

APPLICATION PROGRAMMING INTERFACE (API)

6.1 skijumpdesign/functions.py

skijumpdesign.functions.**cartesian_from_measurements**(*distances*, *angles*, *takeoff_distance=None*)

Returns the Cartesian coordinates of a surface given measurements of distance along the surface and angle measurements at each distance measure along with the takeoff point and takeoff angle.

Parameters

- **distances** (*array_like, shape(n,)*) – Distances measured from an origin location on a jump surface along the surface of the jump.
- **angles** (*array_like, shape(n,)*) – Angle of the slope surface at the distance measures in radians. Positive about a right handed z axis.
- **takeoff_distance** (*float*) – Distance value where the takeoff is located (only if the takeoff is on the surface on the measured portion of the surface).

Returns

- **x** (*ndarray, shape(n-1,)*) – Longitudinal coordinates of the surface.
- **y** (*ndarray, shape(n-1,)*) – Vertical coordinates of the surface.
- **takeoff_point** (*tuple of floats*) – (x, y) coordinates of the takeoff point.
- **takeoff_angle** (*float*) – Angle in radians at the takeoff point.

skijumpdesign.functions.**make_jump**(*slope_angle*, *start_pos*, *approach_len*, *takeoff_angle*, *fall_height*, *plot=False*)

Returns a set of surfaces and output values that define the equivalent fall height jump design and the skier's flight trajectory.

Parameters

- **slope_angle** (*float*) – The parent slope angle in degrees. Counter clockwise is positive and clockwise is negative.
- **start_pos** (*float*) – The distance in meters along the parent slope from the top (x=0, y=0) to where the skier starts skiing.
- **approach_len** (*float*) – The distance in meters along the parent slope the skier travels before entering the takeoff.
- **takeoff_angle** (*float*) – The angle in degrees at end of the takeoff ramp. Counter clockwise is positive and clockwise is negative.
- **fall_height** (*float*) – The desired equivalent fall height of the landing surface in meters.

- **plot** (*boolean*) – If True a matplotlib figure showing the jump will appear.

Returns

- **slope** (*FlatSurface*) – The parent slope starting at (x=0, y=0) until a meter after the jump.
- **approach** (*FlatSurface*) – The slope the skier travels on before entering the takeoff.
- **takeoff** (*TakeoffSurface*) – The circle-clothoid-circle-flat takeoff ramp.
- **landing** (*LandingSurface*) – The equivalent fall height landing surface.
- **landing_trans** (*LandingTransitionSurface*) – The minimum exponential landing transition.
- **flight** (*Trajectory*) – The maximum velocity flight trajectory.
- **outputs** (*dictionary*) – A dictionary of output values with keys: `Takeoff Speed`, `Flight Time`, and `Snow Budget`.

`skijumpdesign.functions.`**`plot_efh`**(*surface*, *takeoff_angle*, *takeoff_point*, *show_knee_collapse_line=True*, *skier=None*, *increment=0.2*, *ax=None*)

Returns a matplotlib axes containing a plot of the surface and its corresponding equivalent fall height.

Parameters

- **surface** (`Surface`) – A Surface for a 2D curve expressed in a standard Cartesian coordinate system.
- **takeoff_angle** (*float*) – Takeoff angle in degrees.
- **takeoff_point** (*2-tuple of floats*) – x and y coordinates relative to the surface's coordinate system of the point at which the skier leaves the takeoff ramp.
- **show_knee_collapse_line** (*bool, optional*) – Displays a line on the EFH plot indicating the EHF above which even elite ski jumpers are unable to prevent knee collapse. This value is taken from [Minetti].
- **skier** (`Skier`, *optional*) – A skier instance. This is passed to `calculate_efh`.
- **increment** (*float, optional*) – x increment in meters between each calculated landing location. This is passed to `calculate_efh`.
- **ax** (*array of Axes, shape(2,), optional*) – An existing matplotlib axes to plot to - ax[0] equivalent fall height, ax[1] surface profile.

References

`skijumpdesign.functions.`**`plot_jump`**(*slope*, *approach*, *takeoff*, *landing*, *landing_trans*, *flight*)

Returns a matplotlib axes with the jump and flight plotted given the surfaces created by `make_jump()`.

`skijumpdesign.functions.`**`snow_budget`**(*parent_slope*, *takeoff*, *landing*, *landing_trans*)

Returns the jump's cross sectional snow budget area of the EFH jump.

Parameters

- **parent_slope** (`FlatSurface`) – A FlatSurface that spans before and after the jump.
- **takeoff** (`TakeoffSurface`) – The clothiod-circle-clothiod-flat takeoff surface.
- **landing** (`LandingSurface`) – The EFH landing surface.
- **landing_trans** (`LandingTransitionSurface`) – The EFH landing transition surface.

Returns
The cross sectional snow budget (area between the parent slope and jump curve) in meters squared.

Return type
float

6.2 skijumpdesign/skiers.py

`class skijumpdesign.skiers.`**`Skier`**`(`*mass=75.0*, *area=0.34*, *drag_coeff=0.821*, *friction_coeff=0.03*, *tolerable_sliding_acc=1.5*, *tolerable_landing_acc=3.0*`)`

Bases: `object`

Class that represents a two dimensional skier who can slide on surfaces and fly in the air.

Instantiates a skier with default properties.

Parameters

- **mass** (*float, optional*) – The mass of the skier in kilograms.
- **area** (*float, optional*) – The frontal area of the skier in squared meters.
- **drag_coeff** (*float, optional*) – The air drag coefficient of the skier.
- **friction_coeff** (*float, optional*) – The sliding friction coefficient between the skis and the slope.
- **tolerable_sliding_acc** (*float, optional*) – The maximum normal acceleration in G's that a skier can withstand while sliding.
- **tolerable_landing_acc** (*float, optional*) – The maximum normal acceleration in G's that a skier can withstand when landing.

drag_force(*speed*)
Returns the drag force in Newtons opposing the speed in meters per second of the skier.

end_speed_on(*surface*, ***kwargs*)
Returns the ending speed after sliding on the provided surface. Keyword args are passed to Skier.slide_on().

end_vel_on(*surface*, ***kwargs*)
Returns the ending velocity (vx, vy) after sliding on the provided surface. Keyword args are passed to Skier.slide_on().

fly_to(*surface*, *init_pos*, *init_vel*, *fine=True*, *compute_acc=True*, *logging_type='info'*)
Returns the flight trajectory of the skier given the initial conditions and a surface which the skier contacts at the end of the flight trajectory.

Parameters

- **surface** (`Surface`) – A landing surface. This surface must intersect the flight path.
- **init_pos** (*2-tuple of floats*) – The x and y coordinates of the starting point of the flight in meters.
- **init_vel** (*2-tuple of floats*) – The x and y components of the skier's velocity at the start of the flight in meters per second.
- **fine** (*boolean*) – If True two integrations occur. The first finds the landing time with coarse time steps and the second integrates over a finer equally spaced time steps. False will skip the second integration.

- **compute_acc** (*boolean, optional*) – If true acceleration will be calculated. If false acceleration is set to zero.
- **logging_type** (*string*) – The logging level desired for the non-debug logging calls in this function. Useful for suppressing too much information since this runs a lot.

Returns
trajectory – A trajectory instance that contains the time, position, velocity, acceleration, speed, and slope of the flight.

Return type
Trajectory

Raises
InvalidJumpError – Error if the skier does not contact a surface within Skier.max_flight_time.

friction_force(*speed*, *slope=0.0*, *curvature=0.0*)
Returns the friction force in Newtons opposing the speed of the skier.

Parameters

- **speed** (*float*) – The tangential speed of the skier in meters per second.
- **slope** (*float, optional*) – The slope of the surface at the skier's point of contact.
- **curvature** (*float, optional*) – The curvature of the surface at the skier's point of contact.

max_flight_time = 30.0

samples_per_sec = 360

slide_on(*surface*, *init_speed=0.0*, *fine=True*)
Returns the trajectory of the skier sliding over a surface.

Parameters

- **surface** (Surface) – A surface that the skier will slide on.
- **init_speed** (*float, optional*) – The magnitude of the velocity of the skier at the start of the surface which is directed tangent to the surface.
- **fine** (*boolean*) – If True two integrations occur. The first finds the exit time with coarse time steps and the second integrates over a finer equally spaced time steps. False will skip the second integration.

Returns
trajectory – A trajectory instance that contains the time, position, velocity, acceleration, speed, and slope of the slide,

Return type
Trajectory

Raises
InvalidJumpError – Error if skier can't reach the end of the surface within 1000 seconds.

speed_to_land_at(*landing_point*, *takeoff_point*, *takeoff_angle*, *surf*)
Returns the magnitude of the velocity required to land at a specific point given launch position and angle.

Parameters

- **landing_point** (*2-tuple of floats*) – The (x, y) coordinates of the desired landing point in meters.

- **takeoff_point** (*2-tuple of floats*) – The (x, y) coordinates of the takeoff point in meters.
- **takeoff_angle** (*float*) – The takeoff angle in radians.
- **surf** (Surface) – This should most likely be the parent slope but needs to be something that ensures the skier flies past the landing point.

Returns
takeoff_speed – The magnitude of the takeoff velocity.

Return type
float

6.3 skijumpdesign/surfaces.py

class skijumpdesign.surfaces.**ClothoidCircleSurface**(*entry_angle*, *exit_angle*, *entry_speed*, *tolerable_acc*, *init_pos=(0.0, 0.0)*, *gamma=0.99*, *num_points=200*)

Bases: *Surface*

Class that represents a surface made up of a circle bounded by two clothoids.

Instantiates a clothoid-circle-clothoid curve.

Parameters

- **entry_angle** (*float*) – The entry angle tangent to the start of the left clothoid in radians.
- **exit_angle** (*float*) – The exit angle tangent to the end of the right clothoid in radians.
- **entry_speed** (*float*) – The magnitude of the skier's velocity in meters per second as they enter the left clothiod.
- **tolerable_acc** (*float*) – The tolerable normal acceleration of the skier in G's.
- **init_pos** (*2-tuple of floats*) – The x and y coordinates of the start of the left clothoid.
- **gamma** (*float*) – Fraction of circular section.
- **num_points** (*integer, optional*) – The number of points in each of the three sections of the curve.

class skijumpdesign.surfaces.**FlatSurface**(*angle*, *length*, *init_pos=(0.0, 0.0)*, *num_points=100*)

Bases: *Surface*

Class that represents a flat surface angled relative to the horizontal.

Instantiates a flat surface that is oriented at a counterclockwise angle from the horizontal.

Parameters

- **angle** (*float*) – The angle of the surface in radians. Counterclockwise (about z) is positive, clockwise is negative.
- **length** (*float*) – The distance in meters along the surface from the initial position.
- **init_pos** (*2-tuple of floats, optional*) – The x and y coordinates in meters that locate the start of the surface.
- **num_points** (*integer, optional*) – The number of points used to define the surface coordinates.

property angle

Returns the angle wrt to horizontal in radians of the surface.

distance_from(*xp*, *yp*)

Returns the shortest distance from point (xp, yp) to the surface.

Parameters

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

Returns

distance – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

Return type

float

class skijumpdesign.surfaces.**HorizontalSurface**(*height*, *length*, *start=0.0*, *num_points=100*)

Bases: *Surface*

Instantiates a class that represents a horizontal surface at a height above the x axis.abs

Parameters

- **height** (*float*) – The height of the surface above the horizontal x axis in meters.
- **length** (*float*) – The length of the surface in meters.
- **start** (*float, optional*) – The x location of the start of the left most point of the surface.
- **num_points** (*integer, optional*) – The number of (x,y) coordinates.

distance_from(*xp*, *yp*)

Returns the shortest distance from point (xp, yp) to the surface.

Parameters

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

Returns

distance – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

Return type

float

class skijumpdesign.surfaces.**LandingSurface**(*skier*, *takeoff_point*, *takeoff_angle*, *max_landing_point*, *fall_height*, *surf*)

Bases: *Surface*

Class that defines an equivalent fall height landing surface.

Instantiates a surface that ensures impact velocity is equivalent to that from a vertical fall.

Parameters

- **skier** (Skier) – A skier instance.
- **takeoff_point** (*2-tuple of floats*) – The point at which the skier leaves the takeoff ramp.
- **takeoff_angle** (*float*) – The takeoff angle in radians.

- **max_landing_point** (*2-tuple of floats*) – The maximum x position that the landing surface will attain in meters. In the standard design, this is the start of the landing transition point.
- **fall_height** (*float*) – The desired equivalent fall height in meters. This should always be greater than zero.
- **surf** (Surface) – A surface below the full flight trajectory, the parent slope is a good choice. It is useful if the distance_from() method runs very fast, as it is called a lot internally.

property allowable_impact_speed

Returns the perpendicular speed one would reach if dropped from the provided fall height.

class skijumpdesign.surfaces.**LandingTransitionSurface**(*parent_surface*, *flight_traj*, *fall_height*, *tolerable_acc*, *num_points=100*)

Bases: *Surface*

Class representing a acceleration limited exponential curve that transitions the skier from the landing surface to the parent slope.

Instantiates an exponentially decaying surface that connects the landing surface to the parent slope.

Parameters

- **parent_surface** (FlatSurface) – The parent slope in which the landing transition should be tangent to on exit.
- **flight_traj** (Trajectory) – The flight trajectory from the takeoff point to the parent slope.
- **fall_height** (*float*) – The desired equivalent fall height for the jump design in meters.
- **tolerable_acc** (*float*) – The maximum normal acceleration the skier should experience in the landing.
- **num_points** (*integer*) – The number of points in the surface.

acc_error_tolerance = 0.001

property allowable_impact_speed

Returns the perpendicular speed one would reach if dropped from the provided fall height.

calc_trans_acc(*x*)

Returns the acceleration in G's the skier feels at the exit transition occurring if the transition starts at the provided horizontal location, x.

delta = 0.01

find_parallel_traj_point()

Returns the position of a point on the flight trajectory where its tangent is parallel to the parent slope. This is used as a starting guess for the start of the landing transition point.

find_transition_point()

Returns the horizontal position indicating the intersection of the flight path with the beginning of the landing transition. This is the last possible transition point, that by definition minimizes the transition snow budget, that satisfies the allowable transition acceleration.

Notes

This uses Newton's method to find an adequate point but may fail to do so with some combinations of flight trajectories, parent slope geometry, and allowable acceleration. A warning will be emitted if the maximum number of iterations is reached in this search and the curve is likely invalid.

`max_iterations = 1000`

`class skijumpdesign.surfaces.Surface(x, y)`

Bases: `object`

Base class for a 2D curve that represents the cross section of a surface expressed in a standard Cartesian coordinate system.

Instantiates an arbitrary 2D surface.

Parameters

- **x** (*array_like, shape(n,)*) – The horizontal, x, coordinates of the slope. x[0] should be the left most horizontal position and corresponds to the start of the surface. This should be monotonically increasing and ideally have no adjacent spacings less than 0.3 meter.
- **y** (*array_like, shape(n,)*) – The vertical, y, coordinates of the slope. y[0] corresponds to the start of the surface.

Warns

- **x and y values that have any x spacings larger than 0.3 meters will be**
- **resampled at x spacings of approximately 0.3 meters.**

`area_under(x_start=None, x_end=None, interval=0.05)`

Returns the area under the curve integrating wrt to the x axis at 0.05 m intervals using the trapezoidal rule.

`calculate_efh(takeoff_angle, takeoff_point, skier, increment=0.2)`

Returns the equivalent fall height for the surface at the specified constant intervals relative to the provided takeoff point or the start of the surface.

Parameters

- **takeoff_angle** (*float*) – Takeoff angle in radians.
- **takeoff_point** (*2-tuple of floats*) – x and y coordinates of the point at which the skier leaves the takeoff ramp.
- **skier** (Skier) – A skier instance.
- **increment** (*float, optional*) – x increment in meters between each calculated landing location.

Returns

- **distance_x** (*ndarray, shape(n,)*) – Horizontal x locations of the equivalent fall height measures spaced at the specified meter intervals relative to leftmost point on the surface or the takeoff point, whichever is greater.
- **efh** (*ndarray, shape(n,)*) – The equivalent fall height corresponding to each value in `distance_x`.
- **takeoff_speeds** (*ndarray, shape(n,)*) – The takeoff speed required to land the corresponding x coordinate.

`distance_from`(*xp*, *yp*)

Returns the shortest distance from point (xp, yp) to the surface.

Parameters

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

Returns

distance – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

Return type

float

Note: This general implementation can be slow, so implement overloaded `distance_from()` methods in subclasses when you can.

`property end`

Returns the x and y coordinates at the end point of the surface.

`height_above`(*surface*)

Returns an array of values giving the height each point in this surface is above the provided surface.

`length`()

Returns the length of the surface in meters via a numerical line integral.

`max_x_spacing = 0.3`

`plot`(*ax=None*, ***plot_kwargs*)

Returns a matplotlib axes containing a plot of the surface.

Parameters

- **ax** (*Axes*) – An existing matplotlib axes to plot to.
- **plot_kwargs** (*dict*) – Arguments to be passed to Axes.plot().

`shift_coordinates`(*delx*, *dely*)

Shifts the x and y coordinates by delx and dely respectively. This modifies the surface in place.

`property start`

Returns the x and y coordinates at the start point of the surface.

`class skijumpdesign.surfaces.TakeoffSurface`(*skier*, *entry_angle*, *exit_angle*, *entry_speed*, *time_on_ramp=0.25*, *gamma=0.99*, *init_pos=(0.0, 0.0)*, *num_points=200*)

Bases: *Surface*

Class that represents a surface made up of a circle bounded by two clothoids with a flat exit surface.

Instantiates the takeoff curve with the flat takeoff ramp added to the terminus of the clothoid-circle-clothoid curve.

Parameters

- **skier** (*Skier*) – A skier instance.
- **entry_angle** (*float*) – The entry angle tangent to the start of the left clothoid in radians.

- **exit_angle** (*float*) – The exit angle tangent to the end of the right clothoid in radians.
- **entry_speed** (*float*) – The magnitude of the skier's velocity in meters per second as they enter the left clothiod.
- **time_on_ramp** (*float, optional*) – The time in seconds that the skier should be on the takeoff ramp before launch.
- **gamma** (*float, optional*) – Fraction of circular section.
- **init_pos** (*2-tuple of floats, optional*) – The x and y coordinates of the start of the left clothoid.
- **num_points** (*integer, optional*) – The number of points in each of the three sections of the curve.

6.4 skijumpdesign/trajectories.py

`class skijumpdesign.trajectories.`**`Trajectory`**`(`*t*, *pos*, *vel=None*, *acc=None*, *speed=None*`)`

Bases: `object`

Class that describes a 2D trajectory.

Instantiates a trajectory.

Parameters

- **t** (*array_like, shape(n,)*) – The time values of the trajectory.
- **pos** (*array_like, shape(n, 2)*) – The x and y coordinates of the position.
- **vel** (*array_like, shape(n, 2), optional*) – The x and y components of velocity. If not provided numerical differentiation of position will be used.
- **acc** (*array_like, shape(n, 2), optional*) – The x and y components of acceleration. If not provided numerical differentiation of velocity will be used.
- **speed** (*array_like, shape(n, 2), optional*) – The magnitude of the velocity. If not provided it will be calculated from the velocity components.

`property duration`

Returns the duration of the trajectory in seconds.

`plot``(`*ax=None*, ***plot_kwargs*`)`

Returns a matplotlib axes containing a plot of the trajectory position.

Parameters

- **ax** (*Axes*) – An existing matplotlib axes to plot to.
- **plot_kwargs** (*dict*) – Arguments to be passed to Axes.plot().

`plot_time_series``()`

Plots all of the time series stored in the trajectory.

`shift_coordinates``(`*delx*, *dely*`)`

Shifts the x and y coordinates by delx and dely respectively. This modifies the surface in place.

6.5 skijumpdesign/utils.py

exception `skijumpdesign.utils.`**`InvalidJumpError`**

Bases: `Exception`

Custom class to signal that a poor combination of parameters have been supplied to the surface building functions.

`skijumpdesign.utils.`**`speed2vel`**(*speed*, *angle*)

Returns the x and y components of velocity given the magnitude and angle of the velocity vector.

Parameters

- **`speed`** (*`float`*) – Magnitude of the velocity vector in meters per second.
- **`angle`** (*`float`*) – Angle of velocity vector in radians. Clockwise is negative and counter clockwise is positive.

Returns

- **vel_x** (*float*) – X component of velocity in meters per second.
- **vel_y** (*float*) – Y component of velocity in meters per second.

`skijumpdesign.utils.`**`vel2speed`**(*hor_vel*, *ver_vel*)

Returns the magnitude and angle of the velocity vector given the horizontal and vertical components.

Parameters

- **`hor_vel`** (*`float`*) – X component of velocity in meters per second.
- **`ver_vel`** (*`float`*) – Y component of velocity in meters per second.

Returns

- **speed** (*float*) – Magnitude of the velocity vector in meters per second.
- **angle** (*float*) – Angle of velocity vector in radians. Clockwise is negative and counter clockwise is positive.

CHAPTER

SEVEN

REFERENCES

The following references provide background information on the theory and rationale of the software implementation.

A paper on this software implementation:

Which is based on this primary reference:

The following are also useful for more in-depth study (in chronological order):

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Minetti] Minetti AE, Ardigo LP, Susta D, Cotelli F (2010) Using leg muscles as shock absorbers: theoretical predictions and experimental results of drop landing performance. Ergonomics 41(12):1771–1791

PYTHON MODULE INDEX

INDEX